Feature-space selection with banded ridge regression

Tom Dupré la Tour[1], Michael Eickenberg[1,2], Anwar O. Nunez-Elizalde[1], and Jack L. Gallant[1,3,*]

[1]Helen Wills Neuroscience Institute, University of California, Berkeley, CA 94720, USA
[2]Center for Computational Mathematics, Flatiron Institute, 162 5th Ave, New York, NY 10100, USA
[3]Department of Psychology, University of California, Berkeley, CA 94720, USA
[*]corresponding author: gallant@berkeley.edu

May 2022

Abstract

Encoding models provide a powerful framework to identify the information represented in brain recordings. In this framework, a stimulus representation is expressed within a feature space and is used in a regularized linear regression to predict brain activity. To account for a potential complementarity of different feature spaces, a joint model is fit on multiple feature spaces simultaneously. To adapt regularization strength to each feature space, ridge regression is extended to banded ridge regression, which optimizes a different regularization hyperparameter per feature space. The present paper proposes a method to decompose over feature spaces the variance explained by a banded ridge regression model. It also describes how banded ridge regression performs a feature-space selection, effectively ignoring non-predictive and redundant feature spaces. This feature-space selection leads to better prediction accuracy and to better interpretability. Banded ridge regression is then mathematically linked to a number of other regression methods with similar feature-space selection mechanisms. Finally, several methods are proposed to address the computational challenge of fitting banded ridge regressions on large numbers of voxels and feature spaces. All implementations are released in an open-source Python package called Himalaya.

Keywords Neuroimaging; Encoding models; Regularized regression; Variance decomposition; Group sparsity; Hyperparameter optimization

Introduction

A fundamental problem in neuroscience is to identify the information represented in different brain areas. In the encoding model framework, this identification amounts to finding the features of the stimulus (or task) that predict the activity in each brain area [Wu et al., 2006; Naselaris et al., 2011]. Encoding models have been used extensively to predict blood oxygen level dependent (BOLD) signals in functional magnetic resonance imaging (fMRI). When encoding models are used independently to model each spatial sample in fMRI recordings (*i.e.* on each *voxel*), they are called *voxelwise* encoding models. Voxelwise encoding models have been used to predict BOLD activity generated by visual images and videos [Hansen et al., 2004; Kay et al., 2008; Nishimoto et al., 2011; Huth et al., 2012; Eickenberg et al., 2016; Wen et al., 2018; Lescroart and Gallant, 2019], music [Kell et al., 2018], semantic concepts [Mitchell et al., 2008; Wehbe et al., 2014; Huth et al., 2016] and narrative language [de Heer et al., 2017; Huth et al., 2016; Jain and Huth, 2018; Deniz et al., 2019]. Encoding models have also been applied to data acquired using other neuroimaging techniques, such as neurophysiological recordings (see [Wu et al., 2006] and references therein), calcium imaging [Miri et al., 2011; Pinto and Dan, 2015; Oldfield et al., 2020], magneto-encephalography [Schwartz et al., 2019; Yang et al., 2019], and electro-corticography [Holdgraf et al., 2016, 2017].

In the encoding model framework, brain activity is recorded while subjects perceive a stimulus or perform a task. Then, a set of features (also known as a *feature space*) is extracted from the stimulus or task at each point in time. For example, a video might be represented in terms of amount of motion in each part of the screen [Nishimoto et al., 2011], or in terms of semantic categories of the objects and actions present in the scene [Huth et al., 2012]. Each feature space corresponds to a different representation of the stimulus- or task-related information. The encoding model framework aims to identify if each representation is similarly encoded in brain activity. Each feature space thus corresponds to a hypothesis about the stimulus- or task-related information encoded in brain activity. To test the hypothesis associated with a feature space, a regression model is trained to predict brain activity from the feature space. If the regression model predicts brain activity significantly in a voxel, then one may conclude that the information represented in the feature space is also represented in brain activity [Wu et al., 2006; Naselaris et al., 2011] (see more details about encoding models in Section 1).

Comparing the prediction accuracy of different feature spaces amounts to comparing competing hypotheses. In each brain voxel, the best-predicting feature space corresponds to the best hypothesis about the information encoded in this voxel. However, it is important not to view feature-space comparison as a winner-take-all situation. It is possible (and in fact common) to find that a brain area represents multiple feature spaces simultaneously. To take this possibility into account, a joint regression is fit on multiple feature spaces. The joint regression automatically combines the information from all feature spaces to maximize the joint prediction accuracy. Then, a variance decomposition method (*e.g.* variance partitioning [Lescroart et al., 2015; Çukur et al., 2016; de Heer et al., 2017; Lescroart and Gallant, 2019; LeBel et al., 2021]) is used to decompose the variance explained by the joint model into separate contributions from each feature space. However, variance partitioning is limited in practice to small numbers of feature spaces m (*e.g.* 2 or 3), because it decomposes the variance into $2^m - 1$ values that are hard to interpret. There is a need for a method to decompose the variance over larger numbers of feature spaces.

The regression model trained to predict brain activity from a feature space is usually regularized in order to improve generalization. The most common regularized regression model is ridge regression (see more details in Section 1.2). However, ridge regression is not suited to fit joint models on multiple feature spaces, because it uses the same regularization hyperparameter for all feature spaces. In practice, it is rare for all feature spaces to require the same level of regularization, because the optimal level of regularization depends on factors such as the number of features, the feature covariances, and the predictive power of the feature space. To address this challenge, ridge regression is naturally extended to use a separate regularization level per feature space. This extension is called *banded ridge regression* [Nunez-Elizalde et al., 2019]. By learning a separate regularization hyperparameter for every feature space, banded ridge regression can optimize regularization strength separately for each feature space. As a side effect, learning separate hyperparameters leads to a feature-space selection mechanism that was not described in the original paper [Nunez-Elizalde et al., 2019]. There is a need for a proper description of this mechanism.

Another challenge with banded ridge regression is its computational cost. Because this method requires the optimization of a separate hyperparameter for every feature space, the optimization algorithm is necessarily more complicated than that used for simple ridge regression. For example, using a grid search (as is common in ridge regression) is impractical in banded ridge regression with more than three or four feature spaces because its computational cost scales exponentially with the number of feature spaces. This computational challenge is particularly difficult when modeling hundreds of thousands of voxels independently. There is a need for an algorithm to efficiently fit models with banded ridge regression.

Overview of the paper. This work expands on earlier banded ridge regression work from our laboratory [Nunez-Elizalde et al., 2019] in three ways, described respectively in the three sections.

Section 1 first provides a brief overview of the encoding model framework. Then, it proposes an alternative method to variance partitioning, to decompose variance over large numbers of feature spaces. The proposed method is based on the product measure [Hoffman, 1960; Pratt, 1987], whose computational cost scales linearly with the number of feature spaces. This variance decomposition can be applied to any linear regression fitting method, including ridge regression and banded ridge regression.

Section 2 reviews banded ridge regression, and describes how it performs a feature-space selection by discarding non-predictive or redundant feature spaces. This feature-space selection mechanism explains how banded ridge regression is able to effectively disentangle correlated feature spaces [Nunez-Elizalde et al., 2019]. Then, it discusses the relationship between banded ridge regression and other regression methods with a similar feature-space selection mechanism, such as automatic relevance determination, multiple-kernel learning, and the group lasso.

Section 3 addresses the practical challenge of solving banded ridge regressions with many feature spaces on large numbers of voxels. Two methods are proposed to efficiently solve banded ridge regression. The first uses hyperparameter random search over a Dirichlet distribution, while the second uses hyperparameter gradient descent with implicit differentiation. Both methods provide speed improvements of several orders of magnitude compared to the Bayesian search proposed earlier [Nunez-Elizalde et al., 2019]. Finally, all implementations are made available in an open-source Python package called *Himalaya*[1], with both CPU and GPU support, rigorous unit-testing, and extensive documentation.

This paper focuses on fMRI datasets and the associated terminology. However, the methods presented here can also be applied to data acquired using other neuroimaging modalities, and more generally to any similarly structured regression setting. The methods could even be used to integrate across multiple neuroimaging modalities Rasero et al. [2021].

[1] https://github.com/gallantlab/himalaya

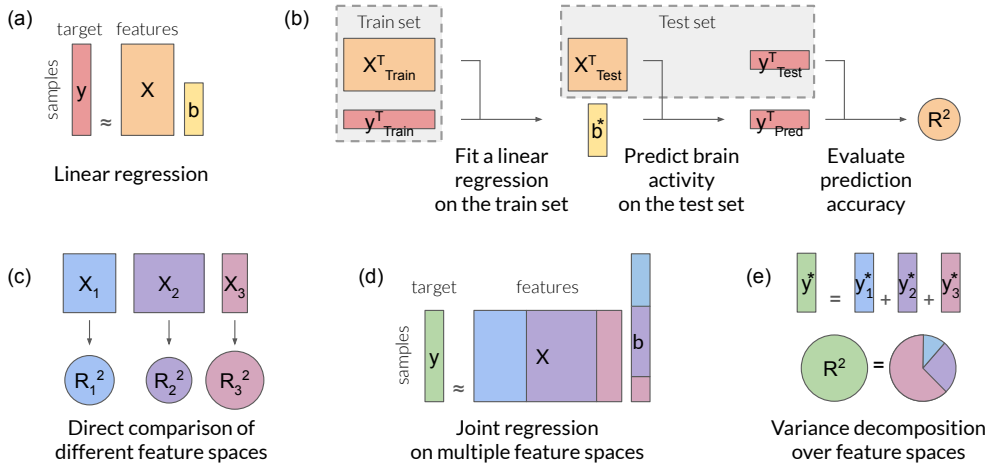

Figure 1: Overview of the encoding model framework. (a) The encoding model framework uses a regression model to predict a brain activity $y \in \mathbb{R}^n$ with features $X \in \mathbb{R}^{n \times p}$ extracted from the stimulus (or task). The regression model is usually linear (see Section 1.1), $y \approx Xb$, with a weight vector $b \in \mathbb{R}^p$. Regularization can be also used to improve the regression model (see Section 1.2). **(b)** A cornerstone of the encoding model framework is the separation of the data into a train set and a test set. The regression model is fit on the train set only. Then, the fit model is used to predict brain activity on the test set. Finally, the prediction accuracy is evaluated on the test set by comparing the prediction vector with the recorded brain activity. The prediction accuracy is quantified for instance with the R^2 score, which can be interpreted as the explained variance (see Section 1.3). **(c)** When multiple feature spaces are available, a separate model can be fit on each feature space, to compare the prediction accuracy of each model (see Section 1.4). However, considering only the best-predicting feature space ignores the possibility that feature spaces can be complementary. **(d)** To take into account complementarity between feature spaces, a joint regression can be fit on multiple feature spaces simultaneously. The feature spaces are concatenated into a large feature matrix X, and a single weight vector b is learned. **(e)** Then, the prediction vector y can be decomposed into a sum of partial predictions from each feature space $y_i = X_i b_i$. Finally, the explained variance R^2 can be decomposed into the contribution of each feature space (see Section 1.5), for instance using the product measure (see Section 1.6) or variance partitioning (see Section 1.7).

1 Variance decomposition over feature spaces

This section starts with a brief overview of the encoding model framework, introducing key concepts such as linear regression, ridge regression, explained variance, and feature-space comparison. Then, it explains the benefit of fitting joint models over multiple feature spaces, and discusses how to decompose the explained variance over feature spaces by adapting the product measure [Hoffman, 1960; Pratt, 1987].

1.1 Linear regression

The regression method used in encoding models is often linear in the parameters. Using linear regression has several benefits compared to non-linear regression. The main benefit is to only focus on *explicit* representations [Kamitani and Tong, 2005; Wu et al., 2006; Naselaris et al., 2011; Kriegeskorte and Kievit, 2013; King et al., 2018; Ivanova et al., 2021]. With non-linear regression, some features might be able to predict brain activity even though they do not explicitly encode the same information as the brain region (*e.g.* pixel features do not explicitly encode the semantic content of an image, even though semantic content can be extracted from the pixels). Restricting the regression method to be linear is thus a way to only consider features that explicitly represent brain activity. Additionally, linearity enables the use of various methods to interpret the regression model, such as variance partitioning or feature importance methods (see Section 1.5). Finally, linear regression methods are computationally efficient and robust to small sample sizes that are common in neuroimaging studies. Note that encoding models that use linear regression are sometimes called *linearized* encoding models [Wu et al., 2006; Naselaris et al., 2011], to emphasize that feature extraction can be non-linear.

Mathematically, let us note $X \in \mathbb{R}^{n \times p}$ the matrix of features, with n samples and p features, and $y \in \mathbb{R}^n$ the brain activity vector (or target vector). Linear regression seeks a weight vector $b \in \mathbb{R}^p$ such that the target vector is approximated as a linear combination of the features $y \approx Xb$. It is standard to model the error as additive Gaussian white noise, so that $y = Xb + \varepsilon$, with a noise vector $\varepsilon \in \mathbb{R}^n$ drawn from a centered Gaussian distribution. (The Gaussian assumption is reasonable for de-trended fMRI recordings, but might not be valid for single-trial neurophysiology). This regression method is called *ordinary least squares*, and can be written as an optimization problem

$$b^* = \underset{b}{\operatorname{argmin}} \, ||y - Xb||_2^2, \tag{1}$$

where $||a||_2^2 = \sum_i a_i^2$ is the squared L_2 norm.

The main limitation of ordinary least squares is that it is unstable in some situations. To understand this issue, note that solving ordinary least squares amounts to inverting the eigenvalues σ_i of the matrix $X^\top X$. When features are correlated, some eigenvalues can be small, which reduces the stability of the inversion (a small change in the features can have a large effect on the result). In some cases, some eigenvalues can even be equal to zero, which makes the inversion ill-defined. Eigenvalues equal to zero can arise when some features are fully redundant (*e.g.* one feature being the sum of two other features), or when the number of independent features is greater than the number of samples (p > n). The system is then called underdetermined, and the optimization problem has infinitely many solutions, of which most do not generalize well to new data. To solve these issues, ordinary least squares can be extended into ridge regression [Hoerl and Kennard, 1970].

1.2 Ridge regression

Ridge regression [Hoerl and Kennard, 1970] is a generalization of ordinary least squares which adds a regularization term to the optimization problem. The optimization problem becomes

$$b^* = \underset{b}{\operatorname{argmin}} \, ||y - Xb||_2^2 + \lambda ||b||_2^2, \tag{2}$$

where $\lambda > 0$ is a hyperparameter controlling the regularization strength. This regularization improves the robustness of the model by adding a positive value λ to all eigenvalues $\sigma_i \geq 0$ of $X^\top X$ before the matrix

inversion. Inverting $(\sigma_i + \lambda)$ instead of σ_i reduces the instability caused by small eigenvalues, and allows inverting underdetermined systems (where $\sigma_i = 0$). Small eigenvalues and underdetermined systems are common when using large numbers of features. Because naturalistic stimuli often involve large numbers of highly correlated features, this regularization scheme facilitates the analysis of naturalistic experiments and so increases ecological validity [Wu et al., 2006; Huth et al., 2016; Sonkusare et al., 2019].

In ridge regression, the regularization hyperparameter λ must be estimated from the data. If the estimated value is too low then the model will tend to overfit to noise in the data, and if it is too high then the model will not predict as well as it might. The regularization parameter is typically selected through a grid search with cross-validation [Hastie et al., 2009, Ch. 7]. First, a set of hyperparameter candidates λ is defined, typically over a grid of logarithmically spaced values to span multiple orders of magnitude. Then, the data set is split into a training set $(X_{\text{train}}, y_{\text{train}})$ and a validation set $(X_{\text{val}}, y_{\text{val}})$. The regression model is fit on the training set, and its prediction accuracy is evaluated on the validation set. This process is performed for all hyperparameter candidates, and the candidate leading to the highest prediction accuracy is selected. To improve the robustness of the hyperparameter selection, the prediction accuracy is usually averaged over multiple splits of the data into a training and a validation set. Finally, a ridge regression is fit on the full data set using the selected hyperparameter.

Ridge regression is the most popular regression method in voxelwise encoding models. Other popular linear regression methods include the lasso [Tibshirani, 1996] or linear support-vector machines [Cortes and Vapnik, 1995]. However, both methods have a much larger computational cost than ridge regression. In particular, their computational cost increases linearly with the number of voxels. Because a typical fMRI dataset contains about 10^5 voxels, fitting an independent regression model on each voxel is computationally challenging. In the case of ridge regression, the computational bottleneck (inverting the linear system) can be shared for all voxels (see Section 3.1 for more details). This is not the case in the lasso or in support-vector machines. For these reasons, solving ridge regression on large numbers of voxels is fast compared to solving the lasso or linear support-vector machines.

1.3 Explained variance

To estimate the prediction accuracy of a model, the model prediction is compared with the recorded brain response. However, encoding models that are relatively more complex are more likely to fit noise in the training data and so they may not generalize well to new data. To avoid artificially inflated performances due to overfitting, this comparison should be performed on a separate data set not used during model training nor during hyperparameter optimization. The ability to evaluate a model on a separate data set is critical to the encoding model framework. It provides a principled way to build complex models while controlling the amount of overfitting. This separate data set is either called the *test set* or the *validation set* depending on the community. This paper uses the term *test set*.

The two standard metrics to quantify prediction accuracy of linear regression models are the R^2 score and the Pearson correlation r. This paper only uses the R^2 score, due to the possibility to decompose the R^2 score over feature spaces (see Section 1.5). The R^2 score is generally defined as

$$R^2 = 1 - \frac{||y - \hat{y}||_2^2}{||y - y_{\text{mean}}||_2^2}. \tag{3}$$

The R^2 score is usually interpreted as the explained variance, *i.e.* the part of the variance that is explained by the model. This interpretation originates from ordinary least squares, where the prediction vector $\hat{y}$ and the error vector $y - \hat{y}$ are always orthogonal on the train set. Indeed, $\hat{y}$ is the vector that minimizes the error $||y - \hat{y}||_2^2$, so it is the orthogonal projection of y over the subspace of all linear combinations of X. Because of this orthogonality, the variance of the brain activity vector y on the train set can be decomposed between the variance of the prediction vector and the variance of the error vector

$$\text{var}(y) = \text{var}(\hat{y}) + \text{var}(y - \hat{y}). \tag{4}$$

Because $(y - \hat{y})$ is zero-mean in ordinary least square on the train set, we get $\text{var}(y - \hat{y}) = ||y - \hat{y}||_2^2/n$. Then,

dividing (4) by $\text{var}(y)$ leads to

$$R^2 = \frac{\text{var}(\hat{y})}{\text{var}(y)}, \tag{5}$$

which is naturally interpreted as the explained variance ratio. However, in the test set and in the case of regularized regression methods such as ridge regression, the prediction vector $\hat{y}$ and the error vector $y - \hat{y}$ are not always orthogonal. Therefore, equations (3) and (5) are not equivalent, and only (5) corresponds to the explained variance ratio. The definition in (3) is however preferred because it is more general [Kvålseth, 1985], and it is still (improperly) interpreted as the explained variance.

The R^2 score takes values in $(-\infty, 1]$. Larger values are better, and a value of 1 corresponds to a perfect prediction. A value of 0 corresponds to the baseline prediction of a constant signal equal to the mean value of y. Negative values can be obtained when the variance of the error $||y - \hat{y}||_2^2$ is larger than the variance of the signal $||y - y_{\text{mean}}||_2^2$. This is often (though not always) a sign of overfitting.

Noise floor. Accurate interpretation of the R^2 score requires comparison to a lower bound given by the statistical significance threshold, and an upper bound given by the noise ceiling. The statistical significance threshold is often estimated by first randomly permuting the prediction and then computing the R^2 score for each permuted prediction. This produces a distribution of R^2 scores under the null hypothesis that the predictions are independent of the recorded brain activity. The statistical significance threshold can be identified from this distribution for any critical p-value that the experimenter might choose. Because this threshold reflects what would be expected to be obtained by chance alone, it is also called the *noise floor*.

Importantly, a statistically significant value does not necessarily correspond to a strong effect size. Any small effect can be significant if the dataset is large enough. Therefore, the effect size should always be reported along with the significance threshold. The R^2 score gives an intuitive quantification of the effect size as the ratio of variance explained by the model.

Noise ceiling. Second, it is necessary to estimate an upper bound for the R^2 score. In theory the R^2 score can reach a value as high as 1, but this is never achieved in practice when there is noise in the data. Therefore, interpretation of the R^2 score also requires estimating the upper bound on possible values of R^2, the *noise ceiling*. If brain responses are stationary with respect to repetitions of the experiment then this upper bound can be estimated by repeating the same experiment several times. Given stationarity, some fraction of the brain response will be identically reproduced over repetitions (the *reproducible signal*), while the rest will vary across repetitions (the *noise*). Because an encoding model can only explain reproducible signals and not noise, the variance of the reproducible signal can be used to estimate the variance that could be potentially explained by an ideal encoding model. The explainable variance can then be used to compute the maximum R^2 score that can be obtained with an encoding model [Sahani and Linden, 2003; Hsu et al., 2004; Schoppe et al., 2016].

1.4 Feature-space comparison

A key element of the encoding model framework is the ability to compare different feature spaces based on their prediction accuracies. A feature space is a set of features that reflects information contained in the experimental stimulus or task. For example, stimulus-related features might be obtained by computing the spatio-temporal wavelet transform of images and videos [Kay et al., 2008; Nishimoto et al., 2011], by labeling the actions and objects present in visual scenes [Huth et al., 2012], or by computing word embeddings that capture the semantic properties of speech [Wehbe et al., 2014; Huth et al., 2016; Deniz et al., 2019]. Task-related features might be defined as indicator variables of attention tasks [Çukur et al., 2013], behavioral states of a video game task [Zhang et al., 2021], or progress during a navigation task [Zhang and Gallant, 2020].

In principle a feature space could reflect any type of stimulus- or task-related information that might potentially be represented in the brain. Thus, a feature space can be viewed as a specific scientific hypothesis about functional brain representations. To test the hypothesis associated with some specific feature space, a regression model is trained to predict brain activity from that feature space. When the regression model has significant prediction accuracy in a particular brain voxel, it suggests that the information represented in the features

correlates with the functional activity measured in that brain voxel [Wu et al., 2006; Naselaris et al., 2011].

Multiple hypotheses can be tested on the same dataset by instantiating multiple feature spaces and then comparing their prediction accuracies. If the prediction accuracy of one feature space is significantly higher than the performance of other feature spaces, then it is common to conclude that the corresponding hypothesis provides the best description about how information is represented in the voxel [Nishimoto et al., 2011; Çukur et al., 2016; Lescroart et al., 2015; de Heer et al., 2017]. However, this approach ignores the possibility that a feature space with lower performance might reflect some information that is represented in the brain, but which is not captured by the best predicting feature space. In this case, both feature spaces would be complementary, in the sense that using both of them jointly would provide a better description of brain function than would be obtained by using either one individually.

By fitting a joint model using multiple feature spaces simultaneously, it becomes possible to optimize the use of each feature space within the model. A joint model maximizes the model prediction accuracy, taking into account potential complementarity of feature spaces. One can then make inferences about the information encoded in a given voxel by decomposing the prediction accuracy of the joint model over the feature spaces.

1.5 Variance decomposition

In a joint model, it is likely that all feature spaces will not contribute equally to the prediction. Some feature spaces might predict a large fraction of brain activity while other feature spaces might have almost no effect. To interpret a joint model, it is thus essential to quantify the relative importance of each feature space in the model. However, most work on this issue has focused on the importance of individual features, not feature spaces. Many *feature importance* measures have been proposed and discussed in the linear regression literature (see reviews in [Nathans et al., 2012; Grömping, 2015]). The most common approach is to standardize the features to zero mean and unit variance, and to then use the magnitude of the regression weights b_j^* as a feature importance measure. Another commonly used feature importance measure is the correlation coefficient $r(X_j, y)$ between feature X_j and the target vector y (*e.g.* the activity of one voxel). Both measures quantify different aspects of feature importance, but they also have limitations (see [Grömping, 2015] for further discussion).

One limitation of both feature importance measures described above is that they are not variance decompositions. To be a variance decomposition, a measure must have two properties. First, the sum over the feature importance measure for all features must be equal to the R^2 score of the model. Second, when all features are orthogonal then the importance of each feature must be equal to the R^2 score obtained using each feature alone. With these two properties, a feature importance measure can be naturally summed over multiple features, to get the relative importance of entire feature spaces. Multiple feature importance measures have these two properties, and each has different limitations (see a list of properties and measures in [Grömping, 2015]). One variance decomposition measure is called variance partitioning, and has been used in the past in voxelwise encoding models. However, variance partitioning is impractical for more than 2 or 3 feature spaces (see Section 1.7). Therefore, this paper focuses on the product measure (described in the next section), because its computational cost scales linearly with the number of feature spaces.

1.6 The product measure

The product measure [Hoffman, 1960; Pratt, 1987] is a feature importance measure. It is defined as the product $\hat{b}_j r(x_j, y)$, where $\hat{b}_j$ is the regression weight associated with feature x_j, and $r(x_j, y)$ is the correlation between feature x_j and the target vector y. (This definition assumes that y and all features x_j are standardized to have zero mean and unit variance.) Importantly, when the linear regression is fit with ordinary least squares, the product measure is a variance decomposition on the training set. Indeed, when summing over features j, we have

$$\sum_j \hat{b}_j r(x_j, y) = \sum_j \hat{b}_j \sum_t x_j[t] y[t] = \sum_t \hat{y}[t] y[t], \tag{6}$$

where $y[t]$ is the value of y at time t. On the other hand, the R^2 score can be written

$$\tilde{R}^2 = 1 - \frac{\sum_t (y[t] - \hat{y}[t])^2}{\sum_t (y[t] - y_{\text{mean}})^2} = \frac{\sum_t \hat{y}[t](2y[t] - \hat{y}[t])}{\sum_t y[t]^2} = \sum_t \hat{y}[t]y[t]. \tag{7}$$

Here, the first equality uses the fact that y is zero-mean. The second equality uses the fact that with ordinary least squares and on the train set, the error vector $\hat{y} - y$ is orthogonal to the target vector y, that is $\sum_t (\hat{y}[t] - y[t])y[t] = 0$. Using both (6) and (7), the sum of the product measure over features is equal to the R^2 score. Moreover, when all features x_j are orthogonal, the product measure decomposition is identical to using the R^2 scores computed on each features alone. With these two properties the product measure is a variance decomposition, and it can be naturally summed over multiple features to obtain the relative importance of entire feature spaces.

Unfortunately, the variance decomposition property holds only for ordinary least squares and on the training set. Therefore, the product measure is not a variance decomposition when using a regularized regression method or when computing the R^2 score on a held-out test set. To fix this issue, we propose here another definition of the product measure. If y is zero-mean, we can write

$$R^2 = \frac{\sum_t \hat{y}[t](2y[t] - \hat{y}[t])}{\sum_t y[t]^2}. \tag{8}$$

Then, we decompose the prediction $\hat{y}$ into $\hat{y} = \sum_j \hat{y}_j$, where $\hat{y}_j = X_j b_j$ is the sub-prediction computed on feature space X_j alone using the weights b_j of the joint model. From this decomposition, we define the variance explained by feature space j as

$$\tilde{R}_j^2 = \frac{\sum_t \hat{y}_j[t](2y[t] - \hat{y}[t])}{\sum_t y[t]^2}. \tag{9}$$

Using this new definition, the variance explained R^2 by the joint model is the sum of the product measure over feature spaces: $\sum_j \tilde{R}_j^2 = R^2$. Moreover, when the sub-predictions $\hat{y}_j$ are all orthogonal, the decomposition is identical to using the R^2 scores computed on each sub-prediction, $\tilde{R}_j^2 = R^2(\hat{y}_j)$. With these two properties the adapted product measure is a variance decomposition. Note that our new definition of the product measure is equal to the original definition in the case of ordinary least squares and on the training set. For this reason, this paper simply refers to it as the product measure.

One problem with the product measure is that it can yield negative values, even when the R^2 score is positive [Thomas et al., 1998]. Negative values that are small relative to other values of the same decomposition might be ignored, but larger negative values are harder to interpret. Large negative values can occur when the regression model uses a positive weight on a feature negatively correlated with the target, or a negative weight on a feature positively correlated with the target. For example, given an orthonormal basis (u, v), two features $x_1 = u - v$ and $x_2 = -u + 2v$ can be used to predict $y = v$ with a regression model $\hat{y} = x_1 + x_2$. The feature x_1 is negatively correlated with the target, but it is used with a positive weight in the model. Thus, the sub-prediction $\hat{y}_1 = x_1$ is projected negatively on $\hat{y}$, which leads to a negative share $\tilde{R}_1^2$. When considering the relative importance of entire feature spaces, this situation can sometimes be resolved by removing or merging feature spaces in the joint model. Note that this limitation of the product measure is also shared with many variance decomposition measures, including the variance partitioning measure described in the next subsection.

1.7 Other variance decomposition measures

Several other variance decomposition measures have been proposed in the linear regression literature, and in theory they could substitute for the product measure. One commonly used measure for variance decomposition of fMRI data is *variance partitioning* [Lescroart et al., 2015; Greene et al., 2016; Çukur et al., 2016; de Heer et al., 2017; Groen et al., 2018; Lescroart and Gallant, 2019; Snoek et al., 2019; LeBel et al., 2021]. This method is also known as commonality analysis in the statistics literature [Mood, 1969; Mayeske, 1969; Bring, 1995; Nathans et al., 2012]. Variance partitioning decomposes the explained variance into shared and unique

components for each feature space, providing a more nuanced interpretation of the contribution of each feature space to predictions. However, variance partitioning decomposes the variance into $2^m - 1$ values which are hard to interpret when the number of feature spaces m is large. To create these $2^m - 1$ values, variance partitioning requires fitting a separate regression model on every subselection of feature spaces, which leads to $2^m - 1$ regression models. The examples shown in this paper use up to 22 separate feature spaces. It would be impossible to use variance partitioning to understand the contributions of so many feature spaces. However, the product measure can still be used even in extreme cases such as this.

2 Banded ridge regression and feature-space selection

The previous section motivates the use of joint models fit over multiple feature spaces simultaneously, and proposes to use the product measure to decompose the variance explained by the model over feature spaces. This section tackles an additional challenge when fitting joint models, which is that different feature spaces might require different regularization strengths. This challenge can be solved using banded ridge regression [Nunez-Elizalde et al., 2019], which optimizes a separate regularization hyperparameter per feature space. Interestingly, this hyperparameter optimization leads to a feature-space selection mechanism. This section describes this feature-space selection mechanism, proposes a metric to quantify feature-space selection, and demonstrates its effect on two examples. Finally, a discussion presents the link between banded ridge regression and other regression methods with a similar feature-space selection mechanism.

2.1 A need for multiple regularization hyperparameters

When fitting a regularized regression such as ridge regression, the regularization strength has a large impact on the model prediction accuracy. When the regularization is too strong, then regression coefficients are too biased toward zero and so relevant features may be underused. On the other hand, when the regularization is too weak, then the model may overfit the training set and so generalization performance will be reduced. Therefore, the regularization strength must be optimized for the dataset under consideration. In particular, the optimal regularization strength depends on many characteristics of the feature space, such as its covariance, its number of features, or its predictive power. When fitting a joint model over multiple feature spaces simultaneously, ridge regression uses a single regularization hyperparameter and so it does not account for the different levels of regularization required for each feature space.

To solve this issue, ridge regression can be extended to *banded ridge regression* [Nunez-Elizalde et al., 2019], a special case of the more general Tikhonov regression [Tikhonov et al., 1977]. Banded ridge regression uses a separate regularization hyperparameter per feature space (Figure 2), and it learns these hyperparameters using cross-validation. Through these multiple hyperparameters, banded ridge regression is able to optimize the level of regularization for each feature space separately.

Banded ridge regression has been proposed multiple times in the scientific literature, either with a Bayesian approach [Perrakis et al., 2020; Ignatiadis and Lolas, 2020; van Nee et al., 2021] or with a frequentist approach [Nunez-Elizalde et al., 2019; van de Wiel et al., 2021]. In the Bayesian approach, a prior assumption is made on the regularization hyperparameter distribution to simplify the optimization problem. In the frequentist approach, no prior assumption is made on the hyperparameter distribution, and the hyperparameters are optimized based on the average cross-validated prediction accuracy. The present paper uses the frequentist approach. Note that banded ridge regression is sometimes called *group-regularized ridge regression* [van de Wiel et al., 2021]. However, the present paper uses the term *banded ridge regression* [Nunez-Elizalde et al., 2019] because in Neuroscience studies, "group" is most often used for groups of subjects and not groups of features.

2.2 Banded ridge regression - model definition

In banded ridge regression, the features are grouped into m feature spaces. A feature space i is formed by a matrix of features $X_i \in \mathbb{R}^{n \times p_i}$, with n samples and p_i features. Each feature space is associated with a

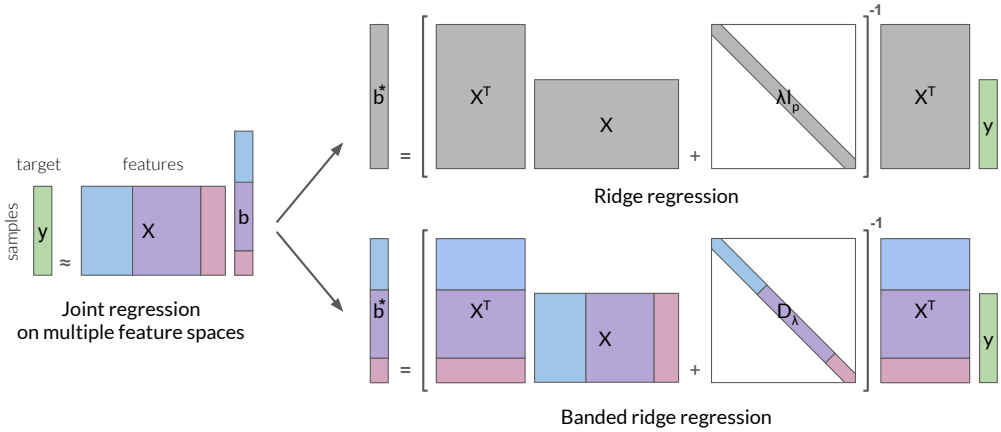

Figure 2: Ridge regression and banded ridge regression. In both regression methods, the brain activity vector $y \in \mathbb{R}^n$ is approximated as a linear sum of the input features $X \in \mathbb{R}^{n \times p}$, using a weight vector $b \in \mathbb{R}^p$, specifically $y \approx Xb$. In ridge regression, the weight vector is computed with $b^* = (X^\top X + \lambda I_p)^{-1} X^\top y$, where $\lambda > 0$ is a regularization hyperparameter common to all features, and $I_p \in \mathbb{R}^{p \times p}$ is the identity matrix. In banded ridge regression, features are grouped into different feature spaces of size p_i, and a different regularization hyperparameter $\lambda_i > 0$ is used for each feature space i. The weight vector is then computed with $b^* = (X^\top X + D_\lambda)^{-1} X^\top y$, where $D_\lambda \in \mathbb{R}^{p \times p}$ is a diagonal matrix with λ_i repeated p_i times. In banded ridge regression, the optimal λ_i for each feature space are learned by cross-validation. Banded ridge regression is thus able to select the feature spaces relevant for prediction, and optimally scales them to maximize cross-validated prediction accuracy.

different regularization hyperparameter $\lambda_i > 0$. To model brain activity $y \in \mathbb{R}^n$ on a particular voxel, banded ridge regression computes the weights $b_i^* \in \mathbb{R}_i^p$ (concatenated into $b^* \in \mathbb{R}^p$ with $p = \sum_i p_i$) defined as

$$b^* = \underset{b}{\operatorname{argmin}} \, || \sum_i X_i b_i - y||_2^2 + \sum_i \lambda_i ||b_i||_2^2. \tag{10}$$

Similarly to ridge regression, the parameters b_i^* are learned on the training data set, and the hyperparameters λ_i are learned by cross-validation [Nunez-Elizalde et al., 2019]. Because banded ridge regression has multiple hyperparameters λ_i, the optimization is more challenging than in ridge regression. (Different strategies for optimizing banded ridge regression are presented in Section 3.)

The benefit of banded ridge regression is its ability to optimize separately the regularization strength of each feature space. This optimization is equivalent to optimizing the relative scale of each feature space. To see this equivalence, the model definition can be rewritten with a change of variable $\tilde{b}_i = b_i \sqrt{\lambda_i / \mu}$, and $\tilde{X}_i = X_i \sqrt{\mu / \lambda_i}$, where $\mu > 0$ is an arbitrary scalar. The equation becomes $\tilde{b}^* = \operatorname{argmin}_b || \sum_i \tilde{X}_i \tilde{b}_i - y||_2^2 + \mu \sum_i || \, b_i||_2^2$. One can then rewrite $\tilde{b}^* = \operatorname{argmin}_b ||\tilde{X}\tilde{b} - y||_2^2 + \mu ||\tilde{b}||_2^2$ by concatenating $\tilde{X} = \tilde{X}_1, \dots, \tilde{X}_m]$ and $\tilde{b} = [\tilde{b}_1, \dots, \tilde{b}_m]$. This equation reveals that banded ridge regression is equivalent to performing ridge regression on the scaled features $\tilde{X}$ [Hansen, 1998; Nunez-Elizalde et al., 2019], with a regularization strength μ.

2.3 A feature-space selection mechanism

Banded ridge regression extends ridge regression to adapt the regularization hyperparameter for each feature space. Importantly, the optimization of multiple regularization hyperparameters also leads to a feature-space selection mechanism. Feature-space selection consists in selecting a subset of feature spaces to be used in the regression, and ignoring the rest of the feature spaces. When a feature space has little predictive power, removing it from the joint model can improve generalization performance by reducing the possibility of overfitting to these features. During cross-validation, banded ridge regression is able to learn to ignore some feature spaces, in order to improve generalization performance. To ignore a feature space i, banded ridge

regression sets a large value for the regularization hyperparameter λ_i, and the coefficients b_i^* are shrunk toward zero. This process effectively removes the feature space i from the model.

With its feature-space selection mechanism, banded ridge regression can select feature spaces that have good predictive power, and suppress those that have little predictive power or that are redundant with other feature spaces. For example, given three correlated feature spaces, banded ridge regression is able to automatically choose between (a) splitting the variance between all three feature spaces, (b) ignoring a redundant feature space and only using the two other ones, or (c) using only one feature space. This example is demonstrated in simulated examples in Appendix 2.4. The feature-space selection thus follows the principle of parsimony (Occam's razor), while allowing multiple feature spaces to be used jointly.

In voxelwise encoding models, the feature-space selection can be performed independently on each voxel. The joint model is thus able to select different feature spaces depending on the voxel. To illustrate this powerful property, [Nunez-Elizalde et al., 2019] presented the following example (although not explaining it as a feature-space selection mechanism). In their analysis, two feature spaces were used to predict brain activity during a visual task recorded with fMRI. The first feature space was a low-level representation of motion in the stimulus (motion energy, [Nishimoto et al., 2011]), and the second feature space was a semantic representation of the objects and actions present in the scene [Huth et al., 2012]. Because the dataset used naturalistic stimuli, the two feature spaces are correlated. Because of these correlations, a regression model fit with the semantic feature space had significant prediction accuracy in low-level visual areas. This model could lead to the incorrect interpretation that low-level visual areas contain visual semantic representations. To address this issue, both feature spaces were used in a joint model fit with banded ridge regression. The joint model ignored the semantic feature space in low-level visual areas, ignored the low-level feature space in visual semantic areas, and used both feature spaces in other visual areas. This example illustrates how having a different feature-space selection in each voxel can avoid incorrect interpretations due to correlations in naturalistic stimuli.

2.4 Example A: Feature-space selection in simulated examples

To illustrate the feature-space selection mechanism of banded ridge regression, four simulated signals were generated from three simulated feature spaces. In two simulated signals, only a subset of the three feature spaces had predictive power. In two other simulated signals, some feature spaces were redundant with other feature spaces. The goal of these simulations is to demonstrate that banded ridge regression is able to ignore the non-predictive or redundant feature spaces.

Compared models. Three different regression models are compared. For the first model, a separate ridge regression model was fit on each feature space independently, and the best feature space per voxel was selected based on the cross-validation prediction accuracy. This first model implements the simple winner-take-all model comparison procedure described in Section 1.4, and it does not use a joint model or variance decomposition. The second model is a ridge regression model fit jointly on all feature spaces. This second model takes into account a possible complementarity of feature spaces, but the ridge regularization does not lead to any feature-space selection. The third model is a banded ridge regression model fit on all feature spaces jointly. This third model takes into account a possible complementarity of feature spaces, and the banded ridge regularization leads to a feature-space selection as described in Section 2.3.

For each model, prediction accuracy was measured by the explained variance (R^2-score) on a separate test set. The explained variance was then decomposed over the three feature spaces using the product measure. The results are shown in Figure 3.

Simulations with non-predictive feature spaces. In the first two simulations, some feature spaces are non-predictive. The ridge regression model fit on the best feature space is not affected by these non-predictive feature spaces, but it cannot use multiple feature spaces jointly. The ridge regression model fit on all feature spaces is able to use multiple feature spaces jointly, but it tends to overfit the non-predictive feature spaces, which reduces its prediction accuracy. The banded ridge regression model fit on all feature spaces leads to the best prediction accuracy, because it is able to use multiple feature spaces jointly while ignoring the non-predictive ones.

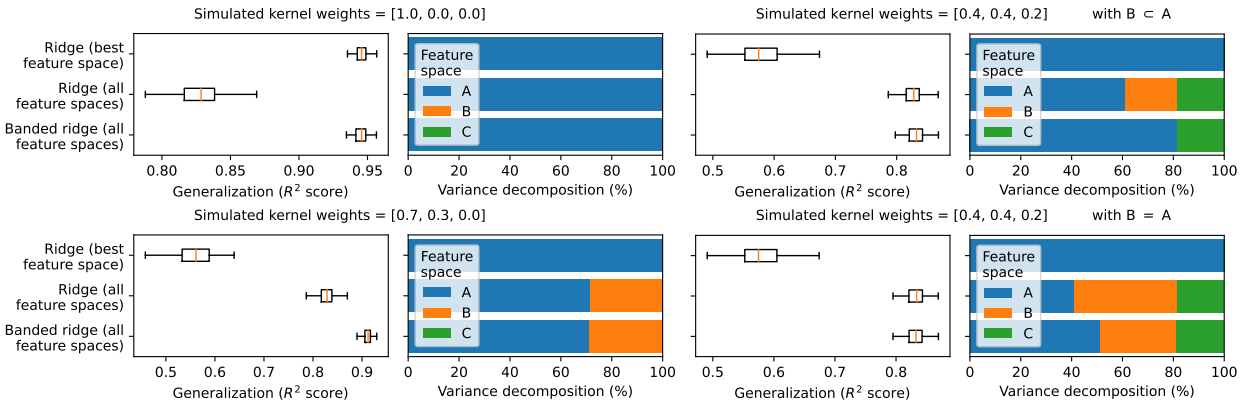

Figure 3: Feature-space selection in four simulated examples. To demonstrate the feature-space selection of banded ridge regression, three different models were fit on four simulated examples, each containing three feature spaces (A, B, C). For each model, prediction accuracy was measured by the explained variance (R^2-score) on a separate test set, using a box plot to show the distribution of score over 100 repetitions of the simulation. The box plot indicates the distribution median (orange bar), the first and third quartiles (box borders), and the 5-th and 95-th percentiles (whiskers). The explained variance was decomposed over the three feature spaces using the product measure, using a geometric median to find a typical decomposition over 100 repetitions. **(Top left)** In the first simulation, the only predictive feature space is A. All three models recover the correct variance decomposition. The prediction accuracy is lower for the ridge regression model fit on all feature spaces than for the two other models, because the model is affected by noise in the non-predictive feature spaces (B, C). **(Bottom left)** In the second simulation, two feature spaces have predictive power (A, B). The ridge regression model fit on the best feature space only uses feature space A, which leads to low prediction accuracy. The ridge regression model fit on all feature spaces recover the correct variance decomposition, but its prediction accuracy is affected by noise in the non-predictive feature space C. The banded ridge regression model fit on all feature spaces recovers the correct variance decomposition, and correctly ignores the non-predictive feature space C to maximize prediction accuracy. **(Top right)** In the third simulation, all three feature spaces are predictive, but feature space B is a subset of A. The ridge regression model fit on all feature spaces uses all three feature spaces, whereas the banded ridge regression model fit on all feature spaces only uses feature spaces A and C, detecting that B is redundant with A. **(Bottom right)** In the fourth simulation, all three feature spaces are predictive, but feature spaces A and B are identical. The ridge regression model fit on all feature spaces uses A and B equally. The banded ridge regression model fit on all feature spaces uses either (A, B, C), or only (A, C), or only (B, C), depending on the repetition, because all three possibilities lead to similar cross-validation scores.

Simulations with non-predictive feature spaces. In the other two simulations, some feature spaces are redundant with other feature spaces. The ridge regression model fit on all feature spaces is not affected by this redundancy, and the shared variance is equally decomposed between the feature spaces. In the case of the banded ridge regression model fit on all feature spaces, if one feature space is strictly better than another, it is selected and the other is ignored. However, if two feature spaces are identical, banded ridge regression is not able to choose between using (i) the first feature space, (ii) the second feature space, or (iii) both at the same time. All three possibilities lead to similar cross-validation scores. The final variance decomposition might thus depend on minor elements such as the initialization in hyperparameter gradient descent, the Dirichlet concentration in the hyperparameter random search, or small variations between the two feature spaces.

2.5 Quantifying feature-space selection

In Section 2.6 and Section 2.7, two additional examples are proposed to demonstrate the feature-space selection mechanism in banded ridge regression. For the sake of demonstration, these examples need a metric to quantify the number of feature spaces effectively used in the model. One candidate metric is to count the

number of feature spaces with a non-zero effect on the prediction. However, because the hyperparameters optimized by banded ridge regression never reach infinity, the coefficients b^* are never exactly zero, even for feature spaces that are effectively ignored. For this reason, all feature spaces have non-zero values in the variance decomposition, and one cannot simply count the number of non-zero values.

To address this issue, we leverage a metric called the *effective rank* [Roy and Vetterli, 2007; Langeberg et al., 2019; Bartlett et al., 2020]. Starting from a variance decomposition $\rho \in \mathbb{R}^m$ (for example, using the product measure as described in Section 1.6), negative values are first clipped to zero, and the variance decomposition vector is normalized to sum to one. Then, the effective rank is computed, as defined in Roy and Vetterli [2007] (see Appendix A.1 for alternative definitions)

$$\tilde{m} = \exp(-\sum_i \rho_i \log(\rho_i)). \tag{11}$$

The effective rank is a continuous function of the m variance decomposition values ρ_i, with values in the interval $[1, m]$. The effective rank is equal to k when the variance decomposition is equally split between k feature spaces. Thus, the effective rank is used in the following two examples to quantify the feature-space selection in banded ridge regression. (Note that the effective rank is independent of the number of features in each feature space. The following two examples only use the effective rank to measure *feature-space* selection.)

2.6 Example B: Feature-space selection over 22 feature spaces

This subsection presents a second example to demonstrate the feature-space selection mechanism in banded ridge regression. The example uses an fMRI dataset recorded while a subject watched several short films [Nunez-Elizalde et al., 2018]. The dataset contains brain responses of 85483 cortical voxels recorded every two seconds. The brain responses were modeled using 22 feature spaces extracted from the stimuli [Nunez-Elizalde et al., 2018]. Examples of feature spaces are the spatio-temporal wavelet transform magnitudes of visual stimuli [Nishimoto et al., 2011], hierarchical classification of objects present in the scenes [Huth et al., 2012], and word embeddings of speech content [Huth et al., 2016] (see Appendix A.2 for a list of all 22 feature spaces). Of the 4379 total time samples, 3572 samples were used in the train set and 807 samples in the test set.

Compared models. This example compares the same three models used in Example A (Section 2.4): (1) ridge regression fit on the best predicting feature space per voxel, (2) ridge regression fit jointly on all feature spaces, and (3) banded ridge regression fit jointly on all feature spaces.

Hyperparameter optimization. For the ridge regression models, the regularization hyperparameter λ was optimized over a grid-search with 30 values spaced logarithmically from 10^{-5} to 10^{20}. This range was large enough to ensure that the algorithm explored many different possible regularization strengths for each feature space. For banded ridge regression, the hyperparameters were optimized with the hyperparameter random-search solver described in Section 3.5, using the settings described in Section 3.8.

Because neighboring time samples are correlated in fMRI recordings, any cross-validation scheme should avoid separating correlated time samples into the train and validation sets. Therefore, time samples recorded consecutively were grouped together into "runs", and a leave-one-run-out cross-validation scheme was used to optimize the hyperparameters. After hyperparameter optimization, all models were refit on the entire training dataset with the optimized hyperparameters.

Prediction accuracy comparison. The three models were first compared in terms of prediction accuracy. To estimate prediction accuracy, the explained variance (R^2) was computed on a separate test set that was not used during model fitting. The results are shown in Figure 4. The left panel compares prediction accuracy of the banded ridge regression model to the winner-take-all ridge regression model. The banded ridge regression outperforms the winner-take-all model, especially for voxels that have relatively more accurate predictions. The right panel compares prediction accuracy of the banded ridge regression model to the ridge regression model that learned a single regularization value for all 22 feature spaces. Here again, the banded ridge regression model outperforms the ridge regression model. These results show that prediction accuracy is

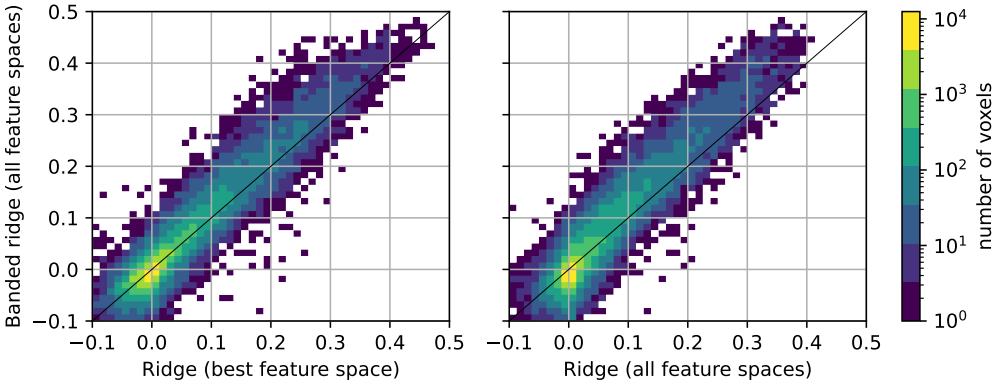

Figure 4: Comparison of prediction accuracy of three different regression models. Three different models were fit using 22 feature spaces to predict brain activity of a subject watching several short films [Nunez-Elizalde et al., 2018]. The prediction accuracy of each model was measured by the explained variance (R^2 score) computed on a separate test set. Each panel compares two models using a 2D histogram over voxels. The diagonal indicates equal performance for both models. **(Left)** Comparison of ridge regression fit on the best feature space per voxel (horizontal) to banded ridge regression fit on all feature spaces (vertical). The mass of the histogram is above the diagonal, indicating better performance for banded ridge regression. It shows that prediction accuracy was improved when multiple feature spaces were fit simultaneously with a separate hyperparameter per feature space, instead of using only the best predicting feature space per voxel. **(Right)** Comparison of ridge regression fit on all feature spaces (horizontal) to banded ridge regression fit on all feature spaces (vertical). The mass of the histogram is again above the diagonal, indicating better performance for banded ridge regression. In both cases, learning a separate regularization per feature space improves the performance of the regression model, especially for voxels that have relatively more accurate predictions.

improved when multiple feature spaces are fit simultaneously and when each space is fit using an optimal regularization parameter for that specific space.

Effective rank comparison. We propose that banded ridge regression produces more accurate predictions than ridge regression because banded ridge regression performs feature-space selection. To quantify feature-space selection, the effective rank (see Section 2.5) was computed on each voxel with positive explained variance. The effective rank gives a continuous measure of the number of feature spaces used in the model.

The top row of Figure 5 compares the effective rank $\tilde{m}$ (horizontal) and the explained variance R^2 (vertical) on each of the three models. In the winner-take-all ridge regression model, the effective rank is always equal to 1, because the model only uses the best feature space per voxel. In the ridge regression model fit on all 22 feature spaces, most values of the effective rank are between 2.4 and 10.3 (5th and 95th percentiles) for the best-predicted voxels ($R^2 > 0.05$). These effective rank values are high because ridge regression does not remove redundant feature spaces, so many feature spaces contribute to the explained variance. In the banded ridge regression model, most values of the effective rank are between 1.0 and 3.7 (5th and 95th percentiles) for the best-predicted voxels ($R^2 > 0.05$). These results indicate that banded ridge regression effectively uses fewer feature spaces than ridge regression, but that banded ridge regression can still use multiple feature spaces jointly when necessary.

Variance decomposition comparison. In the previous paragraph, the feature-space selection of banded ridge regression is measured with the effective rank. To understand more intuitively what the effective rank measures, the variance decomposition results can be visualized for voxels of increasing effective rank. The variance decomposition results were computed by first selecting a subset of best-predicted voxels ($R^2 > 0.05$). Then, the product measure (as defined in Section 1.6) was used to compute the variance decomposition, $\rho \in \mathbb{R}^m$, of these voxels. In this setting, negative values, if present, were very small, and could be clipped to 0. The results were normalized to sum to one to obtain the relative contribution of each feature space to the explained variance. Next, for each voxel separately, the 22 feature spaces were sorted by their relative

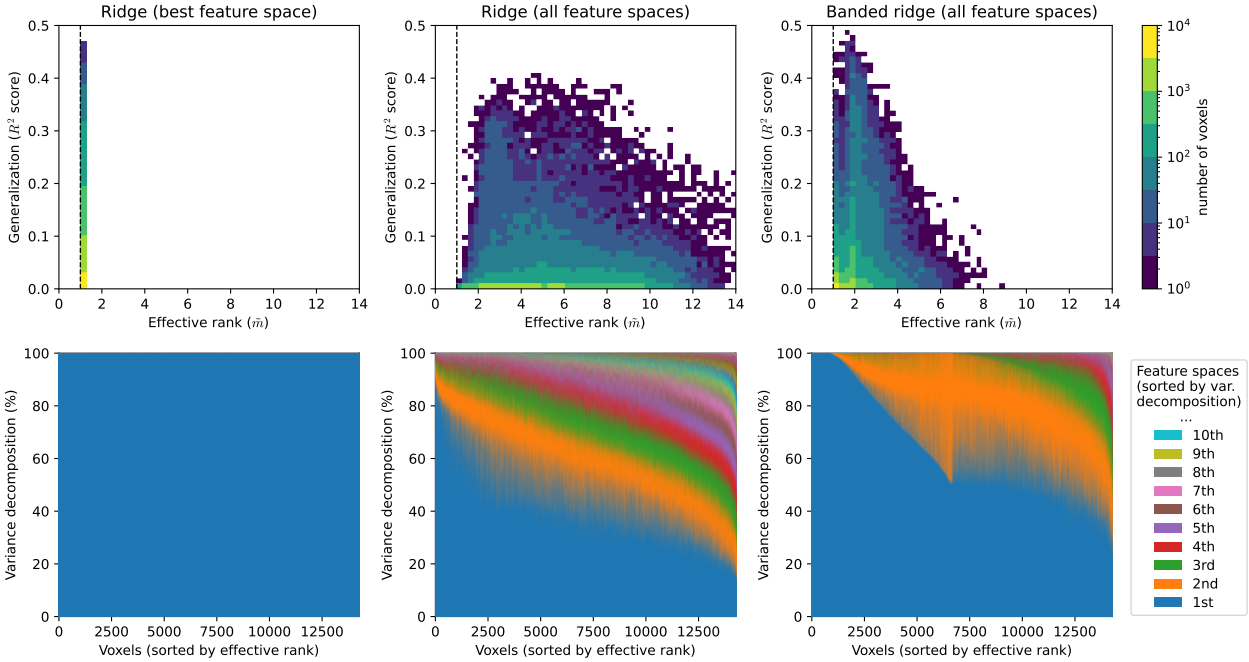

Figure 5: Feature-space selection over 22 feature spaces in three different regression models. To describe the feature-space selection mechanism in banded ridge regression, feature-space selection is quantified using the effective rank $\tilde{m}$. The effective rank is a measure that quantifies the number of feature spaces effectively used by a model in a particular voxel. The three columns correspond to three different models. The (top row) shows a 2D histogram over voxels with $R^2 \geq 0$, comparing the effective rank $\tilde{m}$ (horizontal) and the explained variance R^2 (vertical). This plot describes the range of values obtained with the effective rank. The (bottom row) shows the variance decomposition over feature spaces, for voxels with $R^2 \geq 0.05$. For each voxel, the 22 feature spaces are sorted by contribution to the variance decomposition. Then, voxels are sorted by effective rank $\tilde{m}$. The variance decomposition of each voxel is then displayed using stacked areas. This plot gives an intuitive view of what is captured by the effective rank metric. **(Left)** Ridge regression on the best feature space. On each voxel, the best performing feature space is selected. The effective rank is thus always equal to 1. This winner-take-all selection ignores the possibility that different feature spaces might be complementary on some voxels. **(Middle)** Ridge regression. Many feature spaces contribute to part of the variance, as captured by larger effective rank values. This model does not remove redundant feature spaces. **(Right)** Banded ridge regression. This model contains an implicit feature-space selection mechanism, as indicated by the lower effective rank values. This feature-space selection helps remove redundant feature spaces, while still allowing multiple feature spaces to be used jointly.

contribution ($\rho_1 \geq \rho_2 \geq \ldots > \rho_m$).

The bottom row of Figure 5 shows the normalized variance decomposition as stacked areas, where voxels are sorted along the horizontal axis by their effective rank $\tilde{m}$. For example, the first stacked area (dark blue) corresponds to the relative contribution of the first feature space (*i.e.* the feature space with the highest relative contribution). In the winner-take-all ridge regression model, the first feature space contributes 100% of the explained variance in every voxel. In the ridge regression model fit on all 22 feature spaces, the first feature space contributes between 20% and 90% of the explained variance. In banded ridge regression, the first feature space contributes between 30% and 100% of the explained variance. Overall, these results indicate that banded ridge regression leads to fewer feature spaces being effectively used than with ridge regression, but that banded ridge regression can still use multiple feature spaces jointly when necessary.

2.7 Example C: Feature-space selection over 7 neural network layers

This subsection presents a third example to further demonstrate the feature-space selection mechanism in banded ridge regression. It reuses part of the results described in a previous conference publication of our lab [Dupré la Tour et al., 2021]. This example uses an fMRI dataset recorded while a subject watched short movie clips [Nishimoto et al., 2011]. The dataset contains brain responses of 73211 cortical voxels recorded every two seconds. The brain responses were modeled using 7 feature spaces extracted from the stimuli using intermediate layers of a pre-trained convolutional neural network (CNN) (see the feature extraction paragraph below). Of the 3870 total time samples, 3600 samples were used in the train set and 270 samples in the test set.

Several previous studies have used these sorts of features to compare visual representations in CNNs versus the visual cortex. For example, several studies have used this approach to argue that early CNN layers best predict brain activity in low-level visual areas; late layers best predict brain activity in intermediate and higher-level visual areas; and that the CNN layer that best describe visual tuning changes gradually across the cortical visual hierarchy [Yamins et al., 2014; Agrawal et al., 2014; Cichy et al., 2016; Güçlü and van Gerven, 2015; Eickenberg et al., 2016; Yamins and DiCarlo, 2016; Wen et al., 2018; St-Yves and Naselaris, 2018; Zhang et al., 2019; Zhuang et al., 2021; Nonaka et al., 2021]. This approach has also been used to map the hierarchical representation of features in speech [Kell et al., 2018; Millet and King, 2021] and language [Jain and Huth, 2018; Toneva and Wehbe, 2019].

One limitation of using CNNs as sources of features for modeling brain data is that the features represented in different layers of a CNN are usually strongly correlated. Thus, different encoding models trained with features from different CNN layers have similar prediction accuracies [Eickenberg et al., 2016; Jain and Huth, 2018; Toneva and Wehbe, 2019; Wang et al., 2019]. It is then not clear how to interpret these different models, and disentangle their relative contributions. Most studies ignore this issue and select the best-predicting layer for each voxel [Agrawal et al., 2014; Güçlü and van Gerven, 2015; Eickenberg et al., 2016; Wen et al., 2018; Kell et al., 2018; Schrimpf et al., 2020; Zhang et al., 2019], but this winner-take-all approach is oversimplistic, not robust to noise, and ignores potential complementarities between layers. Some studies use variance partitioning [Groen et al., 2018] or canonical component analysis [Yang et al., 2019] to disentangle the different layers, but these approaches cannot disentangle more than two or three layers. The present example demonstrates how variance decomposition and banded ridge regression can be used to disentangle the contributions of many different CNN layers. The obtained contributions of all CNN layers can then be aggregated into a continuous layer mapping with increased smoothness over the cortical surface.

Feature extraction. To extract features, each image of the stimuli was first presented to a pretrained image-based CNN "Alexnet" [Krizhevsky et al., 2012]. Then, the activations of a CNN layer were extracted (after ReLU and max-pooling layers). Features were extracted frame by frame, and thus needed to be down-sampled to the brain imaging sampling frequency (typically 0.5 Hz). To avoid removing high-frequency information, the layer activations were filtered with eight complex-valued band-pass filters (of frequency bands [0, 0.5], [0.5, 1.5], ..., [6.5, 7.5] Hz) (see more details in [Dupré la Tour et al., 2021]). These band-pass filters were designed to preserve information from different temporal frequency bands, similarly to spatio-temporal features from [Nishimoto et al., 2011]. Then, the amplitude of each filtered complex-valued signal was computed to extract the envelope of the signal (the envelope is the slowly varying amplitude modulation of the high-frequency signal). Next, the envelope was down-sampled to the fMRI sampling frequency with an anti-aliasing low-pass filter. Next, a compressive nonlinearity $x \mapsto \log(1 + x)$ was applied, and features were centered individually along the train set. Finally, to account for the delay between the stimulus and the hemodynamic response, features were duplicated with four temporal delays of $d \in \{2, 4, 6, 8\}$ seconds. This process was repeated on seven CNN layers, to create seven separate feature spaces.

Compared models. This example compares the same three models used in Example A (Section 2.4): (1) ridge regression fit on the best predicting layer per voxel, (2) ridge regression fit jointly on all layers, and (3) banded ridge regression fit jointly on all layers.

Hyperparameter optimization. The hyperparameter optimization settings were identical to the ones in Example B (Section 2.6).

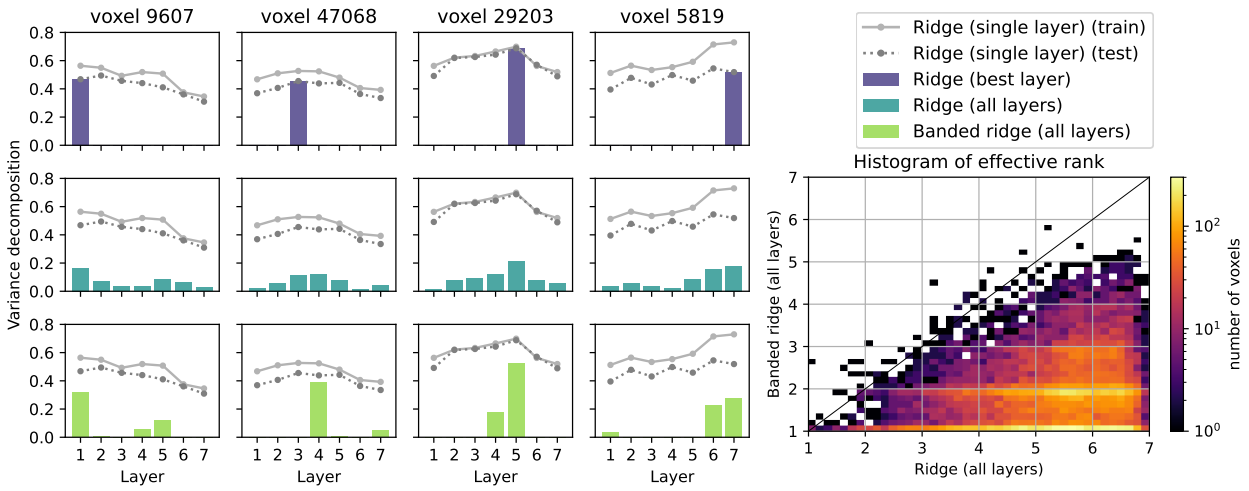

Figure 6: Feature-space selection on seven convolutional neural networks layers. Seven feature spaces were extracted from a movie stimulus based on activations from seven layers of a pretrained CNN (Alexnet [Krizhevsky et al., 2012]). These feature spaces were then used in three different encoding models to predict brain responses to the movie stimulus. The product measure was then used to decompose the explained variance (R^2 score) over the seven layers. **(a)** Comparison of three models in four voxels of brain activity. The first model (*top row*) uses ridge regression fit on the best layer only, ignoring potential complementarity between layers. The second model (*middle row*) uses ridge regression fit jointly on all layers. The variance decomposition shows that this second model uses almost all layers to make the predictions. The third model (*bottom row*) uses banded ridge regression fit jointly on all layers. The variance decomposition shows that this third model uses only two or three layers to make the predictions. Indeed, banded ridge regression performs a feature-space selection, yet allows multiple layers to be used simultaneously. Two gray lines are also given as references, showing the prediction accuracy of a ridge regression model fit on a single layer. These two prediction accuracies are computed either on the train set or on the test set. **(b)** To quantify the number of layers effectively used by each model, the effective rank was computed on the variance decomposition. The 2D histogram of effective rank over voxels shows that banded ridge regression (*vertical*) used fewer feature-spaces (smaller effective rank) than ridge regression (*horizontal*) in almost all voxels.

Prediction accuracy comparison. The three models were first compared in terms of prediction accuracy. To estimate prediction accuracy, the explained variance (R^2) was computed on a separate test set that was not used during model fitting. A detailed comparison of the prediction accuracy of these three models can be found in [Dupré la Tour et al., 2021] and is omitted here for the sake of brevity. In particular, the study shows that the banded ridge regression model outperforms both the ridge regression model fit on the best feature space, and the ridge regression model fit on all the feature spaces.

Feature-space selection comparison. Again, we propose that banded ridge regression produces more accurate predictions than ridge regression because banded ridge regression performs a feature-space selection. To visualize feature-space selection, the explained variance was decomposed over the seven feature spaces (*i.e.* over the seven CNN layers) using the product measure (described in Section 1.6).

Figure 6a shows examples of this variance decomposition on four well-predicted voxels. In ridge regression fit on the best layer, because the model uses only one layer, the variance decomposition is concentrated in a single contribution, and ignores potential layer complementarity. In ridge regression fit jointly on all layers, the variance decomposition is spread across all seven layers, showing that the model uses all layers to make the predictions. In banded ridge regression, the variance decomposition is concentrated on a small number of layers. The model contains a feature-space selection mechanism, leading to sparsity at the layer level, yet allowing multiple layers to be used for complementarity and robustness.

To quantify feature-space selection, the effective rank (see Section 2.5) was computed over all significantly

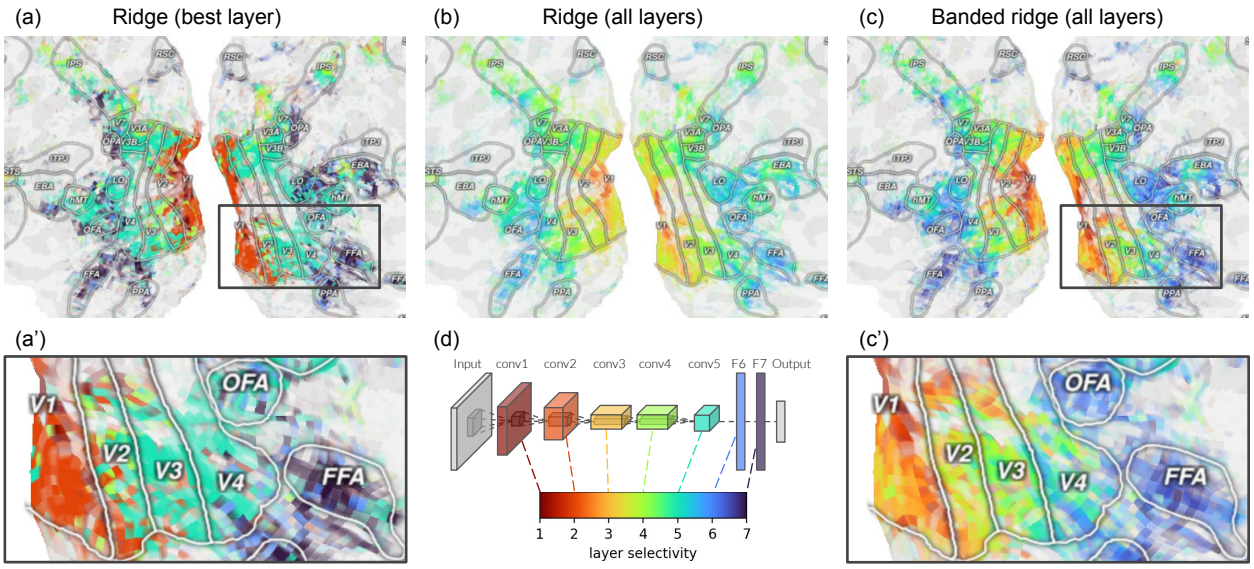

Figure 7: Mapping convolutional neural network layers to the visual cortex. Seven feature spaces were extracted from a movie stimulus based on activations from seven layers of a pretrained CNN (Alexnet [Krizhevsky et al., 2012]). These feature spaces were then used in three different encoding models to predict brain responses to the movie stimulus. Using the fit models, these layers were then mapped to each voxel in the visual cortex. The layer mapping of each voxel is computed as a weighted average of the layer indices, weighted by the decomposition over layers of the explained variance. For visualization, the layer mapping is projected on a flattened cortical surface using Pycortex [Gao et al., 2015]. **(a)** The first model uses ridge regression fit on the best layer only. This winner-take-all approach gives a non-robust estimate of layer mapping, because the best-layer selection can flip from one layer to another due to small variations in prediction accuracy. **(b)** The second model uses ridge regression fit jointly on all layers. It leads to a continuous measure of layer mapping, with a smooth gradient over the cortical surface. However, fitting the joint model with ridge regression gives a biased estimate of layer mapping toward middle values, because its variance decomposition tends to use all layers. **(c)** The third model uses banded ridge regression fit jointly on all layers. Fitting with banded ridge regression mitigates the bias toward middle values thanks to its feature-space selection. **(a', c')** Zoomed views of (a, c). **(d)** Correspondence with the CNN architecture. The layer mapping is derived from the layer indices of a pretrained Alexnet model.

predicted voxels ($p < 0.01$, permutation test). Comparing the distribution of effective rank over voxels, Figure 6b shows that on each voxel, banded ridge regression used fewer layers than ridge regression.

Layer mapping comparison. The three models can also be compared in terms of layer mapping. To describe similarities of representations between CNNs and the visual cortex, it is common to use the index of the best predicting layer as a layer mapping metric on each voxel [Yamins et al., 2014; Agrawal et al., 2014; Cichy et al., 2016; Güçlü and van Gerven, 2015; Eickenberg et al., 2016; Yamins and DiCarlo, 2016; Wen et al., 2018; Dupré la Tour et al., 2021]. As many studies have shown, plotting this layer mapping on the cortex reveals a gradient over the cortical surface (Figure 7a). However, because of correlation between layers, two layers can lead to very similar prediction accuracies. In this case, the best-layer selection picks one layer over the other, even if the prediction accuracy difference is small. The best-layer selection is thus non-robust.

To solve this issue, we extend the definition of layer mapping using a weighted average of layer indices, where the weights are the variance decomposition results. Starting from a variance decomposition $\rho \in \mathbb{R}^m$, where m is the number of layers, negative values are clipped to zero, and ρ is normalized to sum to one. Then, the layer mapping is computed as a weighted average $s = \sum_{i=1}^{m} i\rho_i$. This weighted average leads to a *continuous* layer mapping, which increases both its robustness and its ability to describe gradual changes over the cortical surface.

The weighted average can be computed on any joint model, either fit with ridge regression (Figure 7b) or with banded ridge regression (Figure 7c). In both cases, the continuous layer mapping results in a smoother gradient of layer mapping across the cortical surface. However, the layer mapping estimated by the ridge regression model is biased toward middle values, because the ridge regression model tends to use all layers. Banded ridge regression reduces this bias toward middle values thanks to its feature-space selection mechanism.

2.8 Other models with feature-space selection mechanisms

Feature space selection is an important aspect of many regression algorithms. In the statistics literature, feature-space selection is called *group sparsity*. This subsection reviews a number of models that induce group sparsity, and it describes how each model is related to banded ridge regression.

Automatic relevance determination. Automatic relevance determination [MacKay, 1994; Neal, 1995] is a framework initially developed for Bayesian neural networks [MacKay, 1992b], a class of neural networks where all parameters have explicit prior distributions. In automatic relevance determination, a separate hyperparameter is optimized on each input feature (or each group of features). This hyperparameter optimization induces a feature (or feature-space) selection, automatically determining which features (or feature spaces) are most relevant.

Automatic relevance determination was later applied to Bayesian ridge regression [Box and Tiao, 1973; MacKay, 1992a], a formulation of ridge regression where all parameters and hyperparameters have explicit prior distributions. Using automatic relevance determination on Bayesian ridge regression produces a model called sparse Bayesian learning [Tipping, 2001; Wipf and Nagarajan, 2007]. Sparse Bayesian learning is the direct Bayesian equivalent of banded ridge regression. The main benefit of the Bayesian framework is that all hyperparameters have explicit priors, which leads to efficient optimization solvers without cross-validation [van Nee et al., 2021; Perrakis et al., 2020; Ignatiadis and Lolas, 2020]. However, the lack of cross-validation increases the risk of overfitting. In contrast, banded ridge regression does not make assumptions about hyperparameter prior distributions. To optimize the hyperparameters, banded ridge regression uses cross-validation to select hyperparameters with the best generalization performance.

Multiple-kernel learning. Multiple-kernel learning [Lanckriet et al., 2004; Bach et al., 2004] is a framework that was developed for use with support vector machines [Boser et al., 1992], a class of versatile models for classification and regression. In support vector machines, the features X are only used through a pairwise similarity matrix $K(X)$ called the kernel matrix. In the kernel matrix, each entry is a pairwise similarity between two samples $K(X)_{ij} = k(X_i, X_j)$, where k is a kernel function. Different kernel functions can be used depending on the problem at hand, for example the linear kernel $k(x, y) = x^\top y$, or the radial-basis-function kernel $k(x, y) = \exp(-||x - y||^2/2\sigma^2)$ of size σ. Because different kernels (or different kernel hyperparameters) can lead to different predictive performances, a cross-validation is generally performed to select the best kernel. To leverage the strength of multiple kernels jointly, the multiple-kernel learning framework uses a weighted kernel $K = \sum_i \gamma_i K_i$, where γ_i are positive kernel weights learned by the model. If a kernel weight γ_i has a value close to zero, the associated kernel K_i has little effect in the model. Therefore, learning the kernel weights leads to a kernel-selection mechanism, where only a subset of the kernels are effectively used by the model. If the different kernels are built on different feature spaces, the kernel-selection mechanism corresponds to a feature-space selection mechanism.

This framework has also been applied to ridge regression [Hoerl and Kennard, 1970]. To do so, ridge regression is first re-formulated as a kernel ridge regression [Saunders et al., 1998] with a linear kernel (see more details in Section 3.2). Then, the multiple-kernel framework is used to produce a model called multiple-kernel ridge regression [Bach, 2008]. Interestingly, multiple-kernel ridge regression is *almost* equivalent to banded ridge regression. This quasi-equivalence is described in further details in Appendix A.3.

Group lasso. The group lasso [Yuan and Lin, 2006] is a generalization of the lasso [Tibshirani, 1996]. The lasso is a linear regression model similar to ridge regression but with a different regularization term. The lasso is defined as $b^* = \text{argmin}_b \, ||Xb - y||_2^2 + \lambda ||b||_1$, where $||b||_1 = \sum_j |b_j|$ is the L_1 norm. Due to the

geometry of the L_1 norm, the lasso is known to induce a feature selection [Tibshirani, 1996]. The group lasso is a generalization of the lasso which extends this selection mechanism to feature spaces. The group lasso is defined as $b^* = \text{argmin}_b ||Xb - y||_2^2 + \lambda ||b||_{2,1}$, where $||b||_{2,1} = \sum_i ||b_i||_2$ is the $L_{2,1}$ norm. Due to the geometry of the $L_{2,1}$ norm, the group lasso induces a feature-space selection. (To induce both a feature selection and a feature-space selection, the group lasso can be extended to the sparse group lasso [Simon et al., 2013].) A notable variant of the group lasso is the squared group lasso [Bach, 2008], which uses the squared-$L_{2,1}$ norm $||b||_{2,1}^2 = (\sum_i ||b_i||_2)^2$. The squared group lasso also induces a feature-space selection.

Interestingly, the squared group lasso has been shown to be equivalent to multiple-kernel ridge regression [Bach, 2008; Rakotomamonjy et al., 2008]. Because banded ridge regression is quasi-equivalent to multiple-kernel ridge regression, it is also quasi-equivalent to the squared group lasso. The details of this quasi-equivalence are described in Appendix A.3.

3 Efficient banded ridge regression solvers

As shown in the previous section, banded ridge regression and its feature-space selection mechanism provide a powerful framework for fitting complex encoding models to brain data. However, one obstacle to widespread adoption of this framework is that solving banded ridge regression remains a computational challenge. This section presents several methods for solving banded ridge regression efficiently, and proposes an empirical comparison of these methods. All methods are implemented in an open-source Python package called *Himalaya*.

3.1 Ridge regression solver

Before considering banded ridge regression, it is useful to recall how to solve ridge regression. Let $X \in \mathbb{R}^{n \times p}$ be the matrix of features, with n samples and p features, $y \in \mathbb{R}^n$ the brain activity vector in one particular voxel, and $\lambda > 0$ a fixed regularization hyperparameter. Ridge regression [Hoerl and Kennard, 1970] considers the weight vector $b^* \in \mathbb{R}^p$ defined by the optimization problem

$$b^* = \underset{b}{\text{argmin}} ||y - Xb||_2^2 + \lambda ||b||_2^2. \tag{12}$$

This optimization problem has a closed-form solution $b^* = My$, where $M = (X^\top X + \lambda I_p)^{-1} X^\top \in \mathbb{R}^{p \times n}$.

Efficient solver for multiple voxels. In the closed-form solution above, the resolution matrix M does not depend on the brain activity vector y. Therefore, when fitting a ridge regression on each voxel independently, the matrix M can be precomputed and reused on all voxels. Reusing M on all voxels dramatically reduces the computations compared to solving a linear system per voxel. With t voxels, the precomputation reduces the complexity of solving the full system from $\mathcal{O}(p^3 t + p^2 nt)$ to $\mathcal{O}(p^3 + p^2 n + pnt)$. To further decrease the computation time, one can concatenate all voxels into a matrix Y and write in matrix form $B^* = MY$. The matrix form uses a matrix-matrix multiplication that is very fast on modern hardware, rather than multiple matrix-vector multiplications that would be inevitably slower. Using these methods, ridge regression can be solved efficiently on large numbers of voxels.

Efficient solver for multiple hyperparameters. In ridge regression, the optimal value for the hyperparameter λ is unknown, so it is typically selected through a grid-search with cross-validation (see Section 1.2). Thus, there is a need to solve the model efficiently for multiple hyperparameters λ. To do so, the training input matrix can be decomposed using the singular value decomposition [Golub and Reinsch, 1971; Hastie and Tibshirani, 2004; Rifkin and Lippert, 2007], $X_{\text{train}} = USV^\top$, where $U \in \mathbb{R}^{n \times p}$ is orthonormal, $S \in \mathbb{R}^{p \times p}$ is diagonal, and $V \in \mathbb{R}^{p \times p}$ is orthonormal. (The matrix sizes are given here assuming that $p \leq n$.) Then, for each hyperparameter λ, the resolution matrix is given by $M(\lambda) = V(S^2 + \lambda I_p)^{-1} SU^\top$. This expression is inexpensive to compute because $(S^2 + \lambda I_p)^{-1} S$ is diagonal. By using the singular value decomposition and looping over r values of λ, the complexity of computing the resolution matrix decreases from $\mathcal{O}(p^3 r + pnr)$ to $\mathcal{O}(p^2 nr)$. Using this method, ridge regression can be solved efficiently even when the hyperparameter grid search is large.

To evaluate each hyperparameter, the predictions on the validation set can be computed with $\hat{y}_{\text{val}}(\lambda) = X_{\text{val}} M(\lambda) Y_{\text{train}} = X_{\text{val}} V(S^2 + \lambda I_p)^{-1} S U^\top Y_{\text{train}}$. The optimal order to compute this matrix product depends on p, n_{val}, n_{train}, and t. To further decrease the computation time, the loop over hyperparameters λ can also be implemented with a tensor product.

3.2 Kernel ridge regression solver

Solving ridge regression as described in Section 3.1 requires $\mathcal{O}(p^3)$ operations, and this can become expensive when the number of features p is large. Fortunately, if the number of features is larger than the number of time samples ($p > n$), ridge regression can be reformulated to be solved more efficiently. By the Woodbury matrix identity, b^* can be rewritten $b^* = X^\top (XX^\top + \lambda I_n)^{-1} y$, or $b^* = X^\top w^*$ for some $w^* \in \mathbb{R}^n$. Given the linear kernel $K = XX^\top \in \mathbb{R}^{n \times n}$, this leads to the equivalent formulation

$$w^* = \underset{w}{\text{argmin}} \, ||y - Kw||_2^2 + \lambda w^\top K w. \tag{13}$$

This model is called *kernel ridge regression* [Saunders et al., 1998], and can be used with arbitrary positive semidefinite kernels K. When the kernel is linear ($K = XX^\top$), kernel ridge regression is equivalent to ridge regression.

The kernel formulation has a closed-form solution, $w^* = (K + \lambda I_n)^{-1} y$. Interestingly, this closed-form solution requires inverting a $(n \times n)$ matrix, whereas solving ridge regression requires inverting a $(p \times p)$ matrix. Specifically, the computational complexity of the multiple-voxel case is $\mathcal{O}(n^3 + pn^2 + n^2 t)$ for kernel ridge regression, and $\mathcal{O}(p^3 + p^2 n + pnt)$ for ridge regression. It is thus more efficient to use the kernel formulation when the number of features is larger than the number of time samples $p > n$, and it is more efficient to use the regular formulation otherwise.

Efficient solver for multiple voxels. Similarly to ridge regression, the resolution matrix $M = (K + \lambda I_n)^{-1}$ can be precomputed and efficiently applied to all voxels. To further decrease the computation time, one can again concatenate all voxels into a matrix Y and write in matrix form $w^* = MY$. Using these methods, kernel ridge regression can be solved efficiently on large numbers of voxels.

Efficient solver for multiple hyperparameters. Similarly to ridge regression, the hyperparameter λ is unknown and needs to be selected with cross-validation. To efficiently solve kernel ridge regression during hyperparameter grid search, λ, the training kernel $K_{\text{train}} = X_{\text{train}} X_{\text{train}}^\top$ can be diagonalized into $K_{\text{train}} = UDU^\top$, where $D \in \mathbb{R}^{n \times n}$ is diagonal, and $U \in \mathbb{R}^{n \times n}$ is orthogonal. Then, for each hyperparameter candidate λ, the resolution matrix can be inverted with $(K + \lambda I_n)^{-1} = U(D + \lambda I_n)^{-1} U^\top$. Here, the kernel diagonalization replaces the matrix inversion with a matrix multiplication, which is less computationally expensive. However, it does not change the computational complexity, which is $\mathcal{O}(n^3 r)$ for r values of λ. Using this method, kernel ridge regression can be solved efficiently even when searching through many different values of the hyperparameters.

Finally, search can also be sped up when computing predictions on the validation set $\hat{y}_{\text{val}}(\lambda) = K_{\text{val}} U (D + \lambda I_n)^{-1} U^\top Y_{\text{train}}$, where $K_{\text{val}} = X_{\text{val}} X_{\text{train}}^\top$. The speed-up depends on the optimal order to compute this matrix product, which depends on p, n_{val}, n_{train}, and t.

3.3 Banded ridge regression solver

As discussed in Section 2.2, banded ridge regression is a natural extension of ridge regression, where the features are grouped into m feature spaces. A feature space i is formed by a matrix of features $X_i \in \mathbb{R}^{n \times p_i}$, with n samples and p_i features, and is associated with a regularization hyperparameter $\lambda_i > 0$. To model brain activity $y \in \mathbb{R}^n$ on a particular voxel, banded ridge regression considers the weights $b_i^* \in \mathbb{R}_i^p$ (concatenated into $b^* \in \mathbb{R}^p$ with $p = \sum_i p_i$) defined as

$$b^* = \underset{b}{\text{argmin}} \, || \sum_i X_i b_i - y||_2^2 + \sum_i \lambda_i ||b_i||_2^2. \tag{14}$$

For a fixed hyperparameter vector $\lambda \in \mathbb{R}_+^m$, banded ridge regression has a closed-form solution $b^* = (X^\top X + D_\lambda)^{-1} X^\top y$, where $D_\lambda \in \mathbb{R}^{p \times p}$ is a diagonal matrix with λ_i repeated p_i times, and $X = (X_1, ..., X_m) \in \mathbb{R}^{n \times p}$ is the concatenation of the X_i.

Efficient solver for multiple hyperparameters. Because the hyperparameters in banded ridge regression $\lambda \in \mathbb{R}_+^m$ are optimized by cross-validation, hyperparameter optimization must be implemented efficiently. However, the trick used in ridge regression to factorize computations for all hyperparameter candidates cannot be used in banded ridge regression. Indeed, with more than one hyperparameter ($m > 1$), the singular value decomposition of X does not codiagonalize D_λ. We thus propose two other methods to solve banded ridge regression efficiently: *hyperparameter random search* (described in Section 3.5), and *hyperparameter gradient descent* (described in Section 3.6).

3.4 Multiple-kernel ridge regression

For maximal efficiency, the proposed banded ridge regression solvers are described with the kernel formulation of banded ridge regression, called multiple-kernel ridge regression (see Appendix A.3). This formulation uses kernels, and leads to more efficient solvers than banded ridge regression when $p > n$. In standard ridge regression, when the number of features is larger than the number of samples $p > n$, ridge regression can be solved more efficiently using kernel ridge regression (see Section 3.2). Similarly, when $p > n$, banded ridge regression can be solved more efficiently through its equivalent kernel formulation.

As described in Appendix A.3, multiple-kernel ridge regression uses a set of kernels $(K_1, \dots, K_m)$ with $K_i \in \mathbb{R}^{n \times n}$, a regularization strength $\mu > 0$, and a kernel weight vector γ defined on the simplex $\Delta_{m-1} = \{\gamma \in \mathbb{R}_+^m, \sum_i = 1^m \gamma_i = 1\}$. Then, to model brain activity $y \in \mathbb{R}^n$ on a particular voxel, multiple-kernel ridge regression considers the dual coefficients $w^* \in \mathbb{R}^n$ defined by

$$w^* = \underset{w}{\operatorname{argmin}} \, || \sum_i \gamma_i K_i w - y||_2^2 + \mu w^\top \sum_i \gamma_i K_i w. \tag{15}$$

The brain activity is then modeled using $\hat{y} = \sum_i \gamma_i K_i w^*$. To ensure that the kernel formulation is equivalent to banded ridge regression, the kernels need to be defined using one linear kernel $K_i = X_i X_i^\top$ per feature space i. The correspondence with banded ridge regression hyperparameters is given by $\mu = (\sum_i 1/\lambda_i)^{-1}$ and $\gamma_i = \mu/\lambda_i$.

In the following two subsections, two solvers for multiple-kernel ridge regression are proposed. Both methods are summarized in Figure 8, using a simulated toy example. Because the datasets used in this paper are in the setting $p > n$, these two methods are described for multiple-kernel ridge regression. To be more efficient when $p < n$, similar methods could be derived for banded ridge regression.

3.5 Method 1: Hyperparameter random search

The first proposed method to solve banded ridge regression is hyperparameter random search [Bergstra and Bengio, 2012]. Hyperparameter random search consists in randomly sampling hyperparameter candidates, and then selecting the candidates that produce the lowest cross-validation error. It is often more efficient than grid search, because the random sampling of hyperparameter space is less redundant. For example, when adding useless hyperparameters to a search, a grid search becomes less efficient, while random search efficiency remains unchanged [Bergstra and Bengio, 2012].

In the specific case of multiple-kernel ridge regression, m hyperparameters are optimized, parameterized with $\gamma \in \Delta_{m-1}$ and $\mu > 0$. Formally, the training and validation loss functions are defined as

$$\mathcal{L}_{\text{train}}(w, \gamma, \mu) = \left\| \sum_{i=1}^m \gamma_i K_{\text{train},i} w - y_{\text{train}} \right\|_2^2 + \mu w^\top \sum_{i=1}^m \gamma_i K_{\text{train},i} w, \tag{16}$$

$$\mathcal{L}_{\text{val}}(w^*(\gamma, \mu), \gamma, \mu) = \left\| \sum_{i=1}^m \gamma_i K_{\text{val},i} w^*(\gamma, \mu) - y_{\text{val}} \right\|_2^2, \tag{17}$$

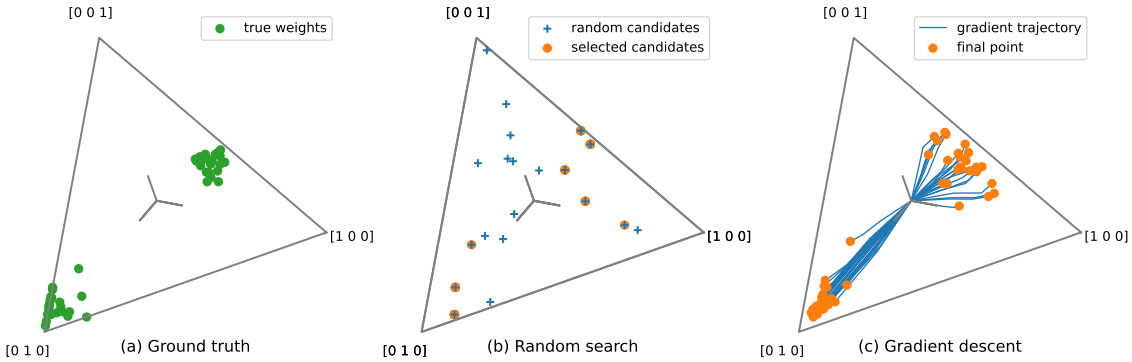

Figure 8: Overview of the two proposed methods to learn hyperparameters in banded ridge regression. In banded ridge regression, a different regularization hyperparameter $\lambda_i > 0$ is used on each feature space i. All hyperparameters are learned using cross-validation. These hyperparameters can be reparameterized into a regularization strength $\mu = (\sum_{i=1}^{m} \lambda_i^{-1})^{-1}$, and a set of kernel weights $\gamma_i = \mu/\lambda_i$. The kernel weight vector is thus defined on the "simplex" $\Delta_{m-1} = \{\gamma \in \mathbb{R}_+^m, \sum_{i=1}^{m} \gamma_i = 1\}$. The figure presents the simplex with $m = 3$, which is a 2-dimensional surface in the shape of a triangle. **(a) Ground truth.** Ground-truth kernel weights on the simplex Δ_2, for 40 different voxels. Brain activity was simulated on 40 voxels, as a function of three feature spaces. On each voxel, the three feature spaces were balanced using a ground-truth kernel-weights vector $\gamma \in \Delta_2$. Note that the ground-truth kernel weights are not necessarily the solution of the optimization problem, because the simulated dataset has a finite size. **(b) Random search.** The random-search method randomly samples candidates from the simplex, and selects for each voxel the candidate leading to the best cross-validation loss. The method is fast because it factorizes computations over all voxels. However, the results directly depend on the random sampling of candidates. **(c) Gradient descent.** The gradient-descent method iteratively optimizes the kernel weights for each voxel independently, using the gradient of the cross-validation loss. The method is more reliable than random search, but it can be slower because it considers each voxel independently.

where $w^*(\gamma, \mu) = \operatorname{argmin}_w \mathcal{L}_{\text{train}}(w, \gamma, \mu)$, and for each feature space i, $K_{\text{train}}, i = X_{\text{train}}, iX_{\text{train}}^\top, i$ is the training kernel, and $K_{\text{val}}, i = X_{\text{val}}, iX_{\text{train}}^\top, i$ is the validation cross-kernel.

With this parameterization, hyperparameter random search is particularly efficient for multiple voxels. For each potential value of the kernel weights $\gamma \in \Delta_{m-1}$, the subproblem is a kernel ridge regression. Using the computational tricks described in Section 3.2, this kernel ridge regression is solved efficiently for multiple voxels and for multiple regularization parameters μ.

Dirichlet distribution. The kernel weights vector γ is defined on the simplex Δ_{m-1}. To generate a candidate γ for random search, the Dirichlet distribution is used. The Dirichlet distribution is specifically defined on the simplex, and it is parametrized to prioritize different levels of sparsity. Its probability density function is parameterized by a set of concentration parameters $\alpha \in \mathbb{R}_+^m$, and reads $p(\gamma) = B(\alpha)^{-1} \Pi_{i=1}^{m} \gamma_i^{\alpha_i - 1}$, where $B(\alpha)$ is a normalization factor written with the gamma function γ. Because in the encoding model framework there is no prior reason to favor one feature space over another, the same value α_i is used for all dimensions. The value $\alpha_i = 1$ for all i leads to a uniform distribution over the simplex. Smaller values increase the density near corners and edges of the simplex, leading to kernel weights that prioritize smaller subsets of kernels.

Full algorithm. The steps to implement hyperparameter random search are as follows. First, a set of candidates for μ is defined. These are most often specified on a unidimensional grid of logarithmically spaced values (*e.g.* $\{1, 10, 100\}$). Second, a set of candidates for γ is defined, sampled from the Dirichlet distribution. Third, for each candidate γ, the weighted-average kernel $K = \sum_i \gamma_i K_i$ is computed. With this single kernel, a kernel ridge regression is solved for all cross-validation splits, all regularizations μ, and all voxels. Fourth, the validation losses are averaged over splits, and their minimum is taken over regularizations μ. Finally, the hyperparameters μ and γ that minimize the average validation loss are selected on each voxel independently.

(The entire algorithm is listed in pseudo-code in Appendix A.7.) Note that this algorithm is equivalent to the one described in [van de Wiel et al., 2021], except for the use of the Dirichlet distribution.

Hyperparameter random search is particularly efficient for large numbers of voxels, because the most costly computations can be factorized and reused on each voxel. This property is critical in voxelwise encoding models, where a separate model is fit independently on about 10^5 voxels or more. However, hyperparameter random search has limited efficiency when the number of dimensions m (the number of feature spaces) increases. Indeed, the number of samples required to cover a m-dimensional space is of the order of $\mathcal{O}(e^m)$. Therefore, hyperparameter random search is not efficient when using large numbers of feature spaces. To address this issue, we propose a second method to optimize hyperparameters in multiple-kernel ridge regression.

3.6 Method 2: Hyperparameter gradient descent

Because hyperparameter random search has limited efficiency when the number of feature spaces is large, we propose a second method to solve banded ridge regression. The second method is hyperparameter gradient descent [Bengio, 2000]. Hyperparameter gradient descent is a method that iteratively improves hyperparameters. At each iteration, the update is based on the gradient of the cross-validation loss $\mathcal{L}_{\text{val}}$ with respect to hyperparameters. Thus, the updated hyperparameters progressively converge toward the hyperparameters that minimize the cross-validation loss. The gradient of the cross-validation loss $\mathcal{L}_{\text{val}}$ with respect to hyperparameters can be computed with implicit differentiation [Larsen et al., 1996; Chapelle et al., 2002; Foo et al., 2007], as described in details in Appendix A.4.

In the specific case of multiple-kernel ridge regression, the method is used to optimize m hyperparameters. In Section 3.5, the hyperparameters are parameterized with $\gamma \in \Delta_{m-1}$ and $\mu > 0$, but this parameterization is inefficient for gradient descent. Indeed, gradient descent does not guarantee that the updated hyperparameters satisfy the constraints $\gamma \in \Delta_{m-1}$ and $\mu > 0$. Thus, after each gradient descent update, an additional step is required to project the hyperparameters on their constrained subspace. Projecting μ on the positive subspace is computationally cheap, but projecting γ on the simplex Δ_{m-1} is more expensive. To avoid this cost, another parameterization is used here, defining $\delta_i = \log(\gamma_i/\mu)$. Because $\delta \in \mathbb{R}^m$ is unconstrained, no additional projection step is required. Furthermore, the logarithm improves the gradient conditioning, because γ_i/μ typically spans multiple orders of magnitude. The training and validation loss functions are then defined as

$$\mathcal{L}_{\text{train}}(w, \delta) = \left\| \sum_{i=1}^{m} e^{\delta_i} K_{\text{train},i} w - y_{\text{train}} \right\|_2^2 + w^\top \sum_{i=1}^{m} e^{\delta_i} K_{\text{train},i} w, \tag{18}$$

$$\mathcal{L}_{\text{val}}(w^*(\delta), \delta) = \left\| \sum_{i=1}^{m} e^{\delta_i} K_{\text{val},i} w^*(\delta) - y_{\text{val}} \right\|_2^2, \tag{19}$$

where $w^*(\delta) = \operatorname{argmin}_w \mathcal{L}_{\text{train}}(w, \delta)$, and for each feature space i, $K_{\text{train}}, i = X_{\text{train}}, i X_{\text{train}}^\top, i$ is the training kernel, and $K_{\text{val}}, i = X_{\text{val}}, i X_{\text{train}}^\top, i$ is the validation cross-kernel. Note that the optimization problem (19) is identical to (17), up to a reparametrization. See Appendix A.4 for the computation of the gradient of the cross-validation loss $\mathcal{L}_{\text{val}}$ with respect to hyperparameters δ.

3.7 Computational complexity comparison

One straightforward way to evaluate different optimization methods is to compare them in terms of computational complexity. For example, one can estimate the computational cost of increasing the number of voxels or the number of feature spaces. Table 1 summarizes the computational complexity of the methods presented above for optimizing hyperparameters for multiple-kernel ridge regression: random search and gradient descent.

The most important difference between both proposed methods is the complexity with respect to the number of voxels v. In random search, the diagonalization is the most expensive operation, but it is done only once for all voxels. Therefore, the computational cost of random search remains almost the same for small or large numbers of voxels. On the contrary in gradient descent, the gradient computations are proportional to the

number of voxels v. Therefore, the computational cost of gradient descent is low for small numbers of voxels, and high for large numbers of voxels. Comparing both methods, gradient descent is likely to be faster than random search for small numbers of voxels, and random search is likely to be faster than gradient descent for large numbers of voxels.

Another key parameter that affects computational complexity is the number of kernels m (*i.e.* the number of feature spaces in banded ridge regression). In the case of random search, the computational cost per iteration is barely affected by m, but the required number of iterations in a random search grows exponentially with m. In the case of gradient descent, the number of iterations required to reach convergence is invariant with respect to m, but the computation of each iteration is proportional to m. Comparing both methods, random search is likely to be faster than gradient descent for small numbers of feature spaces, and gradient descent is likely to be faster than random search for large numbers of feature spaces.

Note that in gradient descent, although the Lipschitz constant computation is inexpensive compared to the other steps, its complexity follows m^2. Its cost might therefore become prohibitive for large numbers of feature spaces m, and an adaptive step-size strategy might become necessary (see for instance [Pedregosa, 2016]). In the example presented in Section 3.8, the Lipschitz constant was computed for up to $m = 22$ feature spaces, which is probably sufficient for most applications.

Random search	
Kernel sum	$\mathcal{O}(mn_{\text{train}}^2)$
Diagonalization	$\mathcal{O}(n_{\text{train}}^3 s)$
Predictions	$\mathcal{O}(n_{\text{val}} n_{\text{train}}(n_{\text{train}} + v)rs)$
Gradient descent	
Lipschitz for w (only once)	$\mathcal{O}(mn_{\text{train}}^3 s)$
Gradient	$\mathcal{O}(mn_{\text{train}}(n_{\text{train}} + n_{\text{val}})vs)$
Lipschitz for δ	$\mathcal{O}(m^2 n_{\text{val}} vs)$

Table 1: Dominant computational complexities per iteration, for both hyperparameter optimization methods. The computational cost of the dominant computations changes when the size of the problem varies. The computational complexities give the order of change in computational cost when each dimension of the problem varies. Here, the different dimensions are the number of kernels m (*i.e.* the number of feature spaces), the number of training samples n_{train}, the number of validation samples n_{val}, the number of voxels v , the number of regularization candidates (for μ) r, and the number of cross-validation splits s. Importantly, the random-search diagonalization, which is the most expensive computation in the random-search method, is independent of the number of voxels v. On the contrary, the gradient-descent gradient computation is proportional to v. Therefore, gradient descent is fast for small numbers of voxels, but as the number of voxels increases, random search becomes more and more competitive compared to gradient descent.

3.8 Banded ridge regression solver comparison

To demonstrate the computational efficiency of the proposed solvers, different banded ridge regression solvers were compared on three fMRI datasets. The number of time samples, feature spaces, cross-validation splits, and voxels of each dataset are detailed in Table 2. The stimuli used in each dataset were silent natural short clips watched with fixation (*"short-clips"* dataset) [Huth et al., 2012], natural stories listened with eyes closed (*"stories"* dataset) [Huth et al., 2016], and non-silent natural short films watched with eye tracking (*"short-films"* dataset) [Nunez-Elizalde et al., 2018]. Each dataset contained a number of manually engineered feature spaces, such as spatio-temporal wavelet transform magnitudes of visual stimuli [Nishimoto et al., 2011], hierarchical classification of objects present in the scenes [Huth et al., 2012], or word embeddings of speech content [Huth et al., 2016].

In all datasets, the number of features was larger than the number of time samples. Thus, all solvers used the multiple-kernel ridge regression formulation. A separate linear kernel was defined per feature space, and

the different solvers described in Section 3.5, Section 3.6, and Appendix A.5 were compared.

dataset	$n_{\text{train}} + n_{\text{val}}$	n_{test}	m	s	v
short-clips	3600	270	2	12	73221
stories	3737	291	4	10	73165
short-films	3572	807	22	12	85483

Table 2: Main dimensions of the three datasets used in our example. The dimensions include the number of training time samples $n_{\text{train}} + n_{\text{val}}$, the number of testing time samples n_{test}, the number of feature spaces m, the number of cross-validation splits s, and the number of voxels v. All datasets had a large number of voxels ($v \approx 10^5$). The largest number of feature spaces is $m = 22$ (see Appendix A.2 for a list of the 22 feature spaces).

Compared methods. In this example, nine different methods were empirically compared. The first three methods were used as baselines, and did not use banded ridge regression. The first baseline method used a ridge regression model fit on all feature spaces jointly, and with a hyperparameter μ shared across all voxels (as in [Huth et al., 2016]). The second baseline method used a ridge regression model fit on all feature spaces jointly, and with a different hyperparameter μ per voxel. In the third baseline, a separate ridge regression model was first fit on each feature space. Then, the best feature space was selected for each voxel independently, as measured by the average cross-validation loss.

The other six compared methods were fitting a banded ridge regression model, formulated as a multiple-kernel ridge regression. They only differed in the method used to solve the optimization problem. The hyperparameter gradient-descent was used with three variants, using either the direct gradient approximation (*"direct"*), the conjugate gradient approximation [Pedregosa, 2016] (*"conjugate"*), or the finite Neumann series approximation [Lorraine et al., 2019] (*"Neumann"*). The hyperparameter random search was used as described in Section 3.5. The proposed methods were also compared to the *Tikreg*[2] implementation published previously by our group [Nunez-Elizalde et al., 2019], which explores hyperparameter space using Bayesian optimization of the average voxel performance (see more details in Appendix A.5). All methods were implemented on GPU, except the Tikreg implementation, which was only available on CPU. To give a fair comparison with the Tikreg implementation, the hyperparameter random search was also run on CPU.

Hyperparameter optimization. For the ridge regression models, the regularization hyperparameter λ was optimized over a grid-search with 20 values spaced logarithmically from 10^{-5} to 10^{15}. This range was large enough to explore many different possible regularization strengths for each feature space. For hyperparameter random search, the Dirichlet concentration α alternated between three values $[0.1/m; \sqrt{(}0.1/m); 1]$, to explore both small and large subsets of kernels. A scaling by $1/m$ was used to get consistent kernel subset sizes over the different datasets. A grid of 20 values was used for μ, spaced logarithmically from 10^{-5} to 10^{15}.

All gradient descent methods were initialized with the optimal regularization hyperparameter of the ridge regression baseline, computed on each voxel independently. For the conjugate gradient approximation, a decreasing precision ε was used with an exponential schedule [Pedregosa, 2016] going from 10^1 to 10^{-1}, for both subproblems (other precision schedules were tested and led to slower convergence). The other gradient-descent approximations used a single iteration of gradient descent to update the dual weights. The Neumann approximation was used with $k = 5$ (other values were tested and led to slower convergence).

As in previous examples, a leave-one-run-out cross-validation scheme was used, and the model was refit on the entire training dataset with the best hyperparameters. The solver comparison was performed on a Nvidia Titan X GPU (12 GB RAM). Because Tikreg was only available on CPU, the random-search solver was also run on CPU along with Tikreg, using a 8-core Intel CPU (3.4 GHz, 30 GB RAM).

Method evaluation. To evaluate the different methods, two criteria were used. The first criterion was the convergence of the validation loss $\mathcal{L}_{\text{val}}$ averaged across cross-validation splits. The convergence of the

[2] https://github.com/gallantlab/tikreg

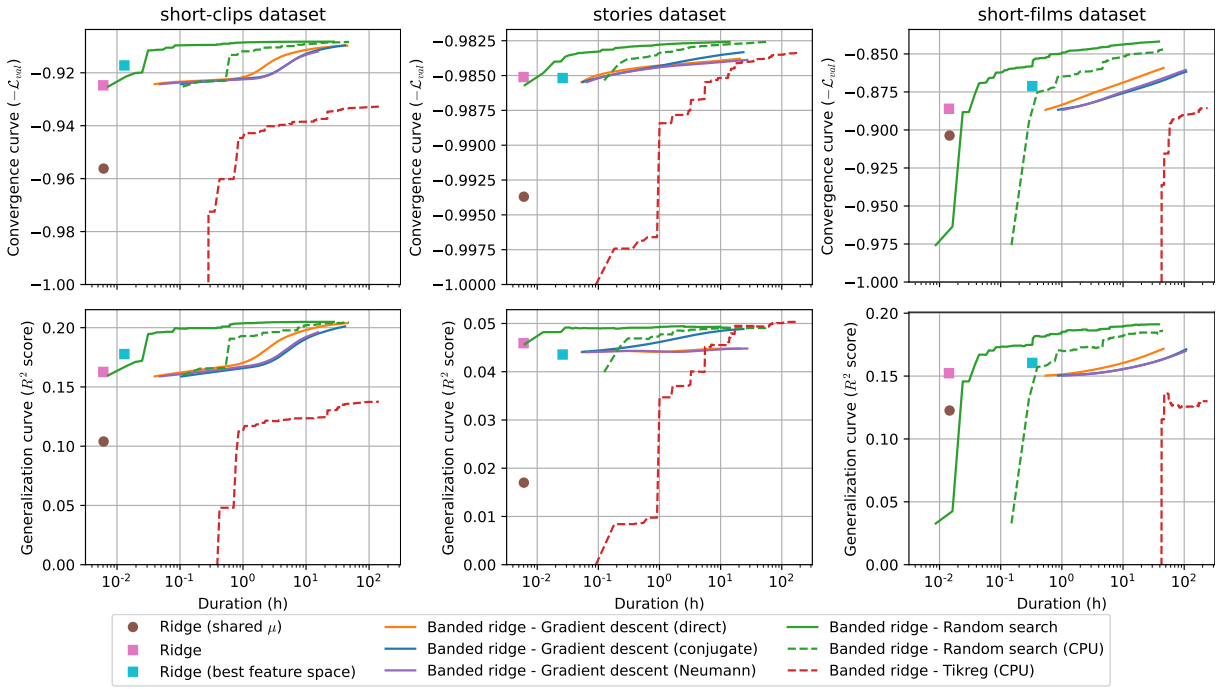

Figure 9: Convergence and generalization of different banded ridge regression solvers, on three different fMRI datasets. All x-axes indicate duration in logarithmic scale. All methods were run on GPU, except when plotted with dashed lines. **(Top row)** Convergence over time, as measured with the negative validation loss ($-\mathcal{L}_{\text{val}}$) (higher is better), averaged over splits and voxels. Convergence over time measures the speed of convergence of the different methods used to fit banded ridge regression. Random search is the fastest method on all three datasets, thanks to its efficient factorization over voxels. In particular, it is more efficient than Tikreg's Bayesian search by several orders of magnitude, even when performed on CPU. **(Bottom row)** Generalization performances over time, as measured by the R^2 score computed on a test set (higher is better), averaged over voxels. The generalization score is useful to verify that the changes in the validation loss are meaningful and not merely overfitting. The generalization score is also useful to compare banded ridge regression with different baseline models. Overall, banded ridge regression reaches higher generalization scores than the ridge regression baselines. The proposed solvers also reach higher generalization scores than Tikreg's Bayesian search, given a reasonable time budget.

validation loss over time is useful to compare the speed of the different methods to solve the optimization problem. The second criterion was the generalization R^2 score on a test data set not used in model fitting. The generalization score is useful to verify that the changes in the validation loss are meaningful and not merely overfitting. The generalization score is also useful to compare banded ridge regression with the different baseline methods based on ridge regression. Note that in all three datasets, most voxels did not get any predictive power from the available feature spaces. Thus, the scores are averaged on a selection of best predicted voxels, selected using the top 10% generalization scores of ridge regression with shared regularization μ. Durations correspond to computations on all voxels.

Results. The results of the method comparison are displayed in Figure 9. First, the Bayesian search implemented in Tikreg appeared several orders of magnitude slower than the proposed methods. This difference is not only due to the GPU implementation, because Tikreg's Bayesian search was also slower than random search run on CPU. Bayesian search also sometimes generalized poorly (short-clips and short-films datasets). This poor generalization can be explained by the optimization being stopped early due to computational constraints. Another possible explanation is that Bayesian search is solving an approximation of the problem, while the proposed methods solve the correct problem (see Appendix A.5).

Among the proposed methods, random search was faster than gradient descent, even for a medium number of feature spaces $m = 22$ (short-films dataset). This might be attributed to the factorization over voxels of random search, which is critical for large numbers of voxels $v \approx 10^5$. This is especially important as the number of voxels can grow even larger for higher resolution fMRI (up to 10^7 in recent datasets [Feinberg et al., 2018]. However, it is expected that gradient descent outperforms random search for larger numbers of feature spaces.

The comparison of the three gradient-descent approximations was not conclusive. On the stories dataset, both direct and Neumann approximations did not generalize as well as other solvers. On the short-clips and short-films datasets, the direct gradient was faster, though one would need further convergence to compare generalization. Importantly, the direct gradient approximation performed well in two out of three datasets. This result is surprising, as the direct gradient is a strong approximation with no guarantee to converge toward a correct solution.

Finally, banded ridge regression generalized better than all ridge regression baselines. This improvement can be explained by the feature-space selection mechanism of banded ridge regression, which effectively removes non-predictive and redundant feature spaces, while still allowing multiple feature spaces to be complementary.

3.9 Recommended strategy

Based on the empirical solver comparison, we recommend using random search to efficiently solve banded ridge regression on large numbers of voxels. However, the convergence of random search is fast in the first iterations but slower afterward. Moreover, the Dirichlet sampling used in random search introduces a prior which biases the hyperparameter optimization. To find more precise feature-space selection and fix the prior bias, one can use gradient descent to refine the solution on a selection of best performing voxels. To do so, the hyperparameters selected by random search can be used to initialize gradient descent. This refinement can lead to a better feature-space selection, and thus improve the variance decomposition interpretation. To limit the computational cost of the gradient-descent refinement, the refinement can be performed only on a selection of best-predicted voxels, as measured by the cross-validation loss.

3.10 Python package

All methods described in this paper are implemented in an open-source Python package called *Himalaya*. All methods are both available through a functional API and through a class API compatible with scikit-learn [Pedregosa et al., 2011]. Moreover, to use either CPU or GPU resources, the package can be used seamlessly with different computational backends based on *Numpy* [Harris et al., 2020], *Pytorch* [Paszke et al., 2019], or *Cupy* [Nishino and Loomis, 2017]. Finally, the package includes extensive documentation, rigorous unit testing (including scikit-learn's estimator checks), and a gallery of examples to facilitate its use.

Conclusion

Banded ridge regression is a natural extension to ridge regression that takes into account any predefined group structure in the features. In neuroimaging studies using encoding models, a group structure is naturally defined when different feature spaces correspond to different hypotheses or different representations of the stimuli and tasks. Banded ridge regression is able to adapt regularization to this structure and to learn optimal scalings for each feature space. These optimal scalings are learned by cross-validation, a principled method that focuses on maximizing generalization to unseen datasets. Banded ridge regression is thus a robust model that adapts regularization to predefined group structures in the features to maximize generalization in linear encoding models.

In this paper, we argue that the feature-space selection mechanism of banded ridge regression partly explains its good performance. By learning optimal feature-space scalings during cross-validation, banded ridge regression is able to ignore some non-predictive or redundant feature spaces. Ignoring these feature spaces reduces the

tendency to overfit, and focuses prediction on feature spaces that are most likely to generalize. Moreover, this feature-space selection improves interpretability, because it reduces the number of feature spaces that contribute to model predictions. This feature-space selection is especially interesting when using naturalistic stimuli that are often best modeled using highly correlated feature spaces. Because banded ridge regression disentangles correlated feature spaces it clarifies which specific feature spaces best predict brain activity.

In this paper we also relate feature-space selection to a large literature of sparsity-inducing linear regression methods. In algorithms ranging from automatic relevance determination to multiple-kernel learning and the group lasso, the selection of feature spaces (or features) has been shown multiple times to lead to better generalization and better interpretation. In the encoding model framework, generalization to unseen data is a powerful way to validate the model and to improve reproducibility. Generalization performance also serves as a guide for model selection and interpretation. By maximizing both interpretation and generalization, banded ridge regression is a powerful tool for data-driven analysis of rich naturalistic experiments in neuroimaging encoding models.

Finally, we demonstrate how the computational cost of banded ridge regression can be largely reduced compared to the original implementation. Our proposed methods use principled algorithms such as hyperparameter random search and hyperparameter gradient descent. In particular, we show how to use these algorithms to fit a separate banded ridge regression on each voxel independently, scaling efficiently to large numbers of voxels. With the advent of layer-specific 7T BOLD fMRI at sub-millimeter resolution, the number of voxels is expected to increase by one or two orders of magnitude compared to the 3T fMRI datasets used in this work. Using efficient algorithms implemented on GPU is thus critical to be able to scale the analysis to the new orders of magnitude of data to come in the future. To facilitate dissemination, all algorithm implementations are released in an open-source, GPU-compatible Python package called *Himalaya*[3].

Acknowledgments

This work was supported by grants from the National Eye Institute (R01-EY031455), the National Science Foundation (Nat-1912373), the Office of Naval Research (N00014-20-1-2002), the National Institutes of Health (B-U01EB02), the Weill Neurohub at University of California, Berkeley, and internal funds from University of California, Berkeley.

This work was also made possible thanks to the scientific Python ecosystem, including *Numpy* [Harris et al., 2020], *Scipy* [Virtanen et al., 2020], *Matplotlib* [Hunter, 2007], *Scikit-learn* [Pedregosa et al., 2011], *Pycortex* [Gao et al., 2015], *Pytorch* [Paszke et al., 2019], and *Cupy* [Nishino and Loomis, 2017].

We also thank Bin Yu, Fabian Pedregosa, Matteo Visconti di Oleggio Castello, Storm Slivkoff, and members of the Gallant lab for fruitful discussions and comments.

Contributions

Conceptualization, TDT, ME, and AONE; Software, TDT and ME; Methodology, TDT; Formal analysis, TDT and ME; Investigation, TDT; Visualization, TDT; Writing - Original Draft, TDT; Writing - Review & Editing, TDT, ME, AONE, and JLG; Funding acquisition, JLG.

Code availability statement

All methods described in this paper are implemented in an open-source Python package called *Himalaya* (https://github.com/gallantlab/himalaya).

[3] https://github.com/gallantlab/himalaya

Ethics statement

Data was acquired on human subjects during previous studies ([Huth et al., 2012, 2016; Nunez-Elizalde et al., 2018]. The experimental protocols were approved by the Committee for the Protection of Human Subjects at University of California, Berkeley. All subjects were healthy, had normal hearing, and had normal or corrected-to-normal vision. Written informed consent was obtained from all subjects. The authors declare no competing financial interests.

References

Agrawal, P., Stansbury, D., Malik, J., and Gallant, J. L. (2014). Pixels to voxels: Modeling visual representation in the human brain. *arXiv preprint arXiv:1407.5104*.

Andor, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K., Petrov, S., and Collins, M. (2016). Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042*.

Araki, H. and Lieb, E. H. (2002). Entropy inequalities. In *Inequalities*, pages 47–57. Springer.

Bach, F. (2022). Information theory with kernel methods. *arXiv preprint arXiv:2202.08545*.

Bach, F. R. (2008). Consistency of the group lasso and multiple kernel learning. *Journal of Machine Learning Research*, 9(Jun):1179–1225.

Bach, F. R., Lanckriet, G. R., and Jordan, M. I. (2004). Multiple kernel learning, conic duality, and the SMO algorithm. In *International conference on machine learning*, page 6.

Bach, F. R., Thibaux, R., and Jordan, M. I. (2005). Computing regularization paths for learning multiple kernels. In *Advances in neural information processing systems*, pages 73–80.

Bartlett, P. L., Long, P. M., Lugosi, G., and Tsigler, A. (2020). Benign overfitting in linear regression. *Proceedings of the National Academy of Sciences*.

Bengio, Y. (2000). Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(Feb):281–305.

Bergstra, J., Yamins, D., and Cox, D. D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *Journal of machine learning research*.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152.

Box, G. E. and Tiao, G. C. (1973). *Bayesian inference in statistical analysis*, volume 40. John Wiley & Sons.

Bring, J. (1995). Variable importance by partitioning r2. *Qual Quant*, 29:173–189.

Chapelle, O., Vapnik, V., Bousquet, O., and Mukherjee, S. (2002). Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1):131–159.

Cichy, R. M., Khosla, A., Pantazis, D., Torralba, A., and Oliva, A. (2016). Comparison of deep neural networks to spatio-temporal cortical dynamics of human visual object recognition reveals hierarchical correspondence. *Scientific reports*, 6:27755.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Çukur, T., Huth, A. G., Nishimoto, S., and Gallant, J. L. (2016). Functional subdomains within scene-selective cortex: parahippocampal place area, retrosplenial complex, and occipital place area. *Journal of Neuroscience*, 36(40):10257–10273.

Çukur, T., Nishimoto, S., Huth, A. G., and Gallant, J. L. (2013). Attention during natural vision warps semantic representation across the human brain. *Nature neuroscience*, 16(6):763–770.

de Heer, W. A., Huth, A. G., Griffiths, T. L., Gallant, J. L., and Theunissen, F. E. (2017). The hierarchical cortical organization of human speech processing. *Journal of Neuroscience*, 37(27):6539–6557.

Deniz, F., Nunez-Elizalde, A. O., Huth, A. G., and Gallant, J. L. (2019). The representation of semantic information across human cerebral cortex during listening versus reading is invariant to stimulus modality. *Journal of Neuroscience*, 39(39):7722–7736.

Dupré la Tour, T., Lu, M., Eickenberg, M., and Gallant, J. L. (2021). A finer mapping of convolutional neural network layers to the visual cortex. In *SVRHM 2021 Workshop@ NeurIPS*.

Eickenberg, M., Gramfort, A., Varoquaux, G., and Thirion, B. (2016). Seeing it all: Convolutional network layers map the function of the human visual system. *NeuroImage*, 152:184–194.

Feinberg, D. A., Vu, A. T., and Beckett, A. (2018). Pushing the limits of ultra-high resolution human brain imaging with sms-epi demonstrated for columnar level fmri. *Neuroimage*, 164:155–163.

Foo, C.-s., Ng, A., et al. (2007). Efficient multiple hyperparameter learning for log-linear models. *Advances in neural information processing systems*, 20:377–384.

Gao, J. S., Huth, A. G., Lescroart, M. D., and Gallant, J. L. (2015). Pycortex: an interactive surface visualizer for fmri. *Frontiers in neuroinformatics*, 9:23.

Golub, G. H. and Reinsch, C. (1971). Singular value decomposition and least squares solutions. In *Linear Algebra*, pages 134–151. Springer.

Gönen, M. and Alpaydın, E. (2011). Multiple kernel learning algorithms. *The Journal of Machine Learning Research*, 12:2211–2268.

Greene, M. R., Baldassano, C., Esteva, A., Beck, D. M., and Fei-Fei, L. (2016). Visual scenes are categorized by function. *Journal of Experimental Psychology: General*, 145(1):82.

Groen, I. I., Greene, M. R., Baldassano, C., Fei-Fei, L., Beck, D. M., and Baker, C. I. (2018). Distinct contributions of functional and deep neural network features to representational similarity of scenes in human brain and behavior. *Elife*, 7:e32962.

Grömping, U. (2015). Variable importance in regression models. *Wiley Interdisciplinary Reviews: Computational Statistics*, 7(2):137–152.

Güçlü, U. and van Gerven, M. A. (2015). Deep neural networks reveal a gradient in the complexity of neural representations across the ventral stream. *Journal of Neuroscience*, 35(27):10005–10014.

Hansen, K. A., David, S. V., and Gallant, J. L. (2004). Parametric reverse correlation reveals spatial linearity of retinotopic human v1 bold response. *Neuroimage*, 23(1):233–241.

Hansen, P. C. (1998). *Rank-deficient and discrete ill-posed problems: numerical aspects of linear inversion*. SIAM.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., et al. (2020). Array programming with numpy. *Nature*, 585(7825):357–362.

Hastie, T. and Tibshirani, R. (2004). Efficient quadratic regularization for expression arrays. *Biostatistics*, 5(3):329–340.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.

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

Hoffman, P. J. (1960). The paramorphic representation of clinical judgment. *Psychological bulletin*, 57(2):116.

Holdgraf, C. R., De Heer, W., Pasley, B., Rieger, J., Crone, N., Lin, J. J., Knight, R. T., and Theunissen, F. E. (2016). Rapid tuning shifts in human auditory cortex enhance speech intelligibility. *Nature communications*, 7(1):1–15.

Holdgraf, C. R., Rieger, J. W., Micheli, C., Martin, S., Knight, R. T., and Theunissen, F. E. (2017). Encoding and decoding models in cognitive electrophysiology. *Frontiers in systems neuroscience*, 11:61.

Hsu, A., Borst, A., and Theunissen, F. E. (2004). Quantifying variability in neural responses and its application for the validation of model predictions. *Network: Computation in Neural Systems*, 15(2):91–109.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95.

Huth, A. G., De Heer, W. A., Griffiths, T. L., Theunissen, F. E., and Gallant, J. L. (2016). Natural speech reveals the semantic maps that tile human cerebral cortex. *Nature*, 532(7600):453–458.

Huth, A. G., Nishimoto, S., Vu, A. T., and Gallant, J. L. (2012). A continuous semantic space describes the representation of thousands of object and action categories across the human brain. *Neuron*, 76(6):1210–1224.

Ignatiadis, N. and Lolas, P. (2020). Group-regularized ridge regression via empirical bayes noise level cross-validation. *arXiv e-prints*, pages arXiv–2010.

Ivanova, A. A., Schrimpf, M., Anzellotti, S., Zaslavsky, N., Fedorenko, E., and Isik, L. (2021). Is it that simple? linear mapping models in cognitive neuroscience. *bioRxiv*.

Jain, S. and Huth, A. (2018). Incorporating context into language encoding models for fmri. In *Advances in neural information processing systems*, pages 6628–6637.

Kamitani, Y. and Tong, F. (2005). Decoding the visual and subjective contents of the human brain. *Nature neuroscience*, 8(5):679–685.

Kay, K. N., Naselaris, T., Prenger, R. J., and Gallant, J. L. (2008). Identifying natural images from human brain activity. *Nature*, 452(7185):352–355.

Keerthi, S. S., Sindhwani, V., and Chapelle, O. (2007). An efficient method for gradient-based adaptation of hyperparameters in SVM models. In *Advances in neural information processing systems*, pages 673–680.

Kell, A. J., Yamins, D. L., Shook, E. N., Norman-Haignere, S. V., and McDermott, J. H. (2018). A task-optimized neural network replicates human auditory behavior, predicts brain responses, and reveals a cortical processing hierarchy. *Neuron*, 98(3):630–644.

King, J.-R., Gwilliams, L., Holdgraf, C., Sassenhagen, J., Barachant, A., Engemann, D., Larson, E., and Gramfort, A. (2018). Encoding and decoding neuronal dynamics: Methodological framework to uncover the algorithms of cognition. *hal.archives-ouvertes.fr*.

Kipper, K., Korhonen, A., Ryant, N., and Palmer, M. (2006). Extending verbnet with novel verb classes. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*.

Klatzer, T. and Pock, T. (2015). Continuous hyper-parameter learning for support vector machines. In *Computer Vision Winter Workshop (CVWW)*, pages 39–47.

Kriegeskorte, N. and Kievit, R. A. (2013). Representational geometry: integrating cognition, computation, and the brain. *Trends in cognitive sciences*, 17(8):401–412.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Kvålseth, T. O. (1985). Cautionary note about r 2. *The American Statistician*, 39(4):279–285.

Lanckriet, G. R., Cristianini, N., Bartlett, P., Ghaoui, L. E., and Jordan, M. I. (2004). Learning the kernel matrix with semidefinite programming. *Journal of Machine learning research*, 5(Jan):27–72.

Langeberg, P., Balda, E. R., Behboodi, A., and Mathar, R. (2019). On the effect of low-rank weights on adversarial robustness of neural networks. *arXiv preprint arXiv:1901.10371*.

Larsen, J., Hansen, L. K., Svarer, C., and Ohlsson, M. (1996). Design and regularization of neural networks: the optimal use of a validation set. In *Neural Networks for Signal Processing VI. Proceedings of the 1996 IEEE Signal Processing Society Workshop*, pages 62–71. IEEE.

LeBel, A., Jain, S., and Huth, A. G. (2021). Voxelwise encoding models show that cerebellar language representations are highly conceptual. *Journal of Neuroscience*, 41(50):10341–10355.

Lescroart, M. D. and Gallant, J. L. (2019). Human scene-selective areas represent 3d configurations of surfaces. *Neuron*, 101(1):178–192.

Lescroart, M. D., Stansbury, D. E., and Gallant, J. L. (2015). Fourier power, subjective distance, and object categories all provide plausible models of bold responses in scene-selective visual areas. *Frontiers in computational neuroscience*, 9:135.

Lorraine, J., Vicol, P., and Duvenaud, D. (2019). Optimizing millions of hyperparameters by implicit differentiation. *arXiv preprint arXiv:1911.02590*.

MacKay, D. J. (1992a). Bayesian interpolation. *Neural computation*, 4(3):415–447.

MacKay, D. J. (1992b). A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472.

MacKay, D. J. (1994). Bayesian methods for backpropagation networks. In *Models of neural networks III*, pages 211–254. Springer.

Mayeske, G. W. (1969). *A study of our nation's schools*. US Office of Education.

Millet, J. and King, J.-R. (2021). Inductive biases, pretraining and fine-tuning jointly account for brain responses to speech. *arXiv preprint arXiv:2103.01032*.

Miri, A., Daie, K., Burdine, R. D., Aksay, E., and Tank, D. W. (2011). Regression-based identification of behavior-encoding neurons during large-scale optical imaging of neural activity at cellular resolution. *Journal of neurophysiology*, 105(2):964–980.

Mitchell, T. M., Shinkareva, S. V., Carlson, A., Chang, K.-M., Malave, V. L., Mason, R. A., and Just, M. A. (2008). Predicting human brain activity associated with the meanings of nouns. *Science*, 320(5880):1191–1195.

Mockus, J., Tiesis, V., and Zilinskas, A. (1978). The application of bayesian methods for seeking the extremum. *Towards global optimization*, 2(117-129):2.

Mood, A. M. (1969). Macro-analysis of the american educational system. *Operations Research*, 17(5):770–784.

Naselaris, T., Kay, K. N., Nishimoto, S., and Gallant, J. L. (2011). Encoding and decoding in fMRI. *Neuroimage*, 56(2):400–410.

Nathans, L. L., Oswald, F. L., and Nimon, K. (2012). Interpreting multiple linear regression: A guidebook of variable importance. *Practical Assessment, Research, and Evaluation*, 17(1):9.

Neal, R. M. (1995). *Bayesian learning for neural networks*, volume 118. Springer.

Nishimoto, S., Vu, A. T., Naselaris, T., Benjamini, Y., Yu, B., and Gallant, J. L. (2011). Reconstructing visual experiences from brain activity evoked by natural movies. *Current Biology*, 21(19):1641–1646.

Nishino, R. and Loomis, S. H. C. (2017). Cupy: A numpy-compatible library for nvidia gpu calculations. *31st confernce on neural information processing systems*, page 151.

Nonaka, S., Majima, K., Aoki, S. C., and Kamitani, Y. (2021). Brain hierarchy score: Which deep neural networks are hierarchically brain-like? *IScience*, 24(9):103013.

Nunez-Elizalde, A., Deniz, F., Gao, J. S., and Gallant, J. L. (2018). Discovering brain representations across multiple feature spaces using brain activity. In *48th Annual Meeting of Society for Neuroscience (SfN), San Diego*.

Nunez-Elizalde, A. O., Huth, A. G., and Gallant, J. L. (2019). Voxelwise encoding models with non-spherical multivariate normal priors. *NeuroImage*, 197:482–492.

Obozinski, G., Jacob, L., and Vert, J.-P. (2011). Group lasso with overlaps: the latent group lasso approach. *arXiv preprint arXiv:1110.0413*.

Oldfield, C. S., Grossrubatscher, I., Chávez, M., Hoagland, A., Huth, A. R., Carroll, E. C., Prendergast, A., Qu, T., Gallant, J. L., Wyart, C., et al. (2020). Experience, circuit dynamics, and forebrain recruitment in larval zebrafish prey capture. *Elife*, 9:e56619.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Pan, Q. and Zhao, Y. (2016). Integrative weighted group lasso and generalized local quadratic approximation. *Computational Statistics & Data Analysis*, 104:66–78.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035.

Pedregosa, F. (2016). Hyperparameter optimization with approximate gradient. In *International Conference on Machine Learning*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

Perrakis, K., Mukherjee, S., Initiative, A. D. N., et al. (2020). Scalable bayesian regression in high dimensions with multiple data sources. *Journal of Computational and Graphical Statistics*, 29(1):28–39.

Pinto, L. and Dan, Y. (2015). Cell-type-specific activity in prefrontal cortex during goal-directed behavior. *Neuron*, 87(2):437–450.

Pratt, J. W. (1987). Dividing the indivisible: Using simple symmetry to partition variance explained. In *Proceedings of the second international Tampere conference in statistics, 1987*, pages 245–260. Department of Mathematical Sciences, University of Tampere.

Rakotomamonjy, A., Bach, F. R., Canu, S., and Grandvalet, Y. (2008). Simplemkl. *Journal of Machine Learning Research*, 9(Nov):2491–2521.

Rasero, J., Sentis, A. I., Yeh, F.-C., and Verstynen, T. (2021). Integrating across neuroimaging modalities boosts prediction accuracy of cognitive ability. *PLoS computational biology*, 17(3):e1008347.

Rifkin, R. M. and Lippert, R. A. (2007). Notes on regularized least squares.

Roy, O. and Vetterli, M. (2007). The effective rank: A measure of effective dimensionality. In *2007 15th European Signal Processing Conference*, pages 606–610. IEEE.

Sahani, M. and Linden, J. (2003). How linear are auditory cortical responses. *Advances in neural information processing systems*, 15:301–308.

Saunders, C., Gammerman, A., and Vovk, V. (1998). Ridge regression learning algorithm in dual variables. In *International conference on machine learning*.

Schoppe, O., Harper, N. S., Willmore, B. D., King, A. J., and Schnupp, J. W. (2016). Measuring the performance of neural models. *Frontiers in computational neuroscience*, 10:10.

Schrimpf, M., Kubilius, J., Hong, H., Majaj, N. J., Rajalingham, R., Issa, E. B., Kar, K., Bashivan, P., Prescott-Roy, J., Geiger, F., et al. (2020). Brain-score: Which artificial neural network for object recognition is most brain-like? *BioRxiv*, page 407007.

Schwartz, D., Toneva, M., and Wehbe, L. (2019). Inducing brain-relevant bias in natural language processing models. In *Advances in Neural Information Processing Systems*, pages 14123–14133.

Simon, N., Friedman, J., Hastie, T., and Tibshirani, R. (2013). A sparse-group lasso. *Journal of computational and graphical statistics*, 22(2):231–245.

Snoek, L., Miletić, S., and Scholte, H. S. (2019). How to control for confounds in decoding analyses of neuroimaging data. *Neuroimage*, 184:741–760.

Sonkusare, S., Breakspear, M., and Guo, C. (2019). Naturalistic stimuli in neuroscience: Critically acclaimed. *Trends in cognitive sciences*, 23(8):699–714.

St-Yves, G. and Naselaris, T. (2018). The feature-weighted receptive field: an interpretable encoding model for complex feature spaces. *NeuroImage*, 180:188–202.

Thomas, D. R., Hughes, E., and Zumbo, B. D. (1998). On variable importance in linear regression. *Social Indicators Research*, 45(1-3):253–275.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.

Tikhonov, A. N., Arsenin, V. I., Arsenin, V., et al. (1977). *Solutions of ill-posed problems*. Vh Winston.

Tipping, M. E. (2001). Sparse bayesian learning and the relevance vector machine. *Journal of machine learning research*, 1(Jun):211–244.

Toneva, M. and Wehbe, L. (2019). Interpreting and improving natural-language processing (in machines) with natural language-processing (in the brain). In *Advances in Neural Information Processing Systems*, pages 14928–14938.

Tsuda, K., Uda, S., Kin, T., and Asai, K. (2004). Minimizing the cross validation error to mix kernel matrices of heterogeneous biological data. *Neural Processing Letters*, 19(1):63–72.

van de Wiel, M. A., van Nee, M. M., and Rauschenberger, A. (2021). Fast cross-validation for multi-penalty high-dimensional ridge regression. *Journal of Computational and Graphical Statistics*, 30(4):835–847.

van Nee, M. M., Wessels, L. F., and van de Wiel, M. A. (2021). Flexible co-data learning for high-dimensional prediction. *Statistics in medicine*, 40(26):5910–5925.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al. (2020). Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272.

Wang, A., Tarr, M., and Wehbe, L. (2019). Neural taskonomy: Inferring the similarity of task-derived representations from brain activity. In *Advances in Neural Information Processing Systems*, pages 15475–15485.

Wehbe, L., Murphy, B., Talukdar, P., Fyshe, A., Ramdas, A., and Mitchell, T. (2014). Simultaneously uncovering the patterns of brain regions involved in different story reading subprocesses. *PloS one*, 9(11).

Wen, H., Shi, J., Zhang, Y., Lu, K.-H., Cao, J., and Liu, Z. (2018). Neural encoding and decoding with deep learning for dynamic natural vision. *Cerebral Cortex*, 28(12):4136–4160.

Wipf, D. and Nagarajan, S. (2007). A new view of automatic relevance determination. *Advances in neural information processing systems*, 20:1625–1632.

Wu, M. C.-K., David, S. V., and Gallant, J. L. (2006). Complete functional characterization of sensory neurons by system identification. *Annu. Rev. Neurosci.*, 29:477–505.

Yamins, D. L. and DiCarlo, J. J. (2016). Using goal-driven deep learning models to understand sensory cortex. *Nature neuroscience*, 19(3):356–365.

Yamins, D. L., Hong, H., Cadieu, C. F., Solomon, E. A., Seibert, D., and DiCarlo, J. J. (2014). Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the National Academy of Sciences*, 111(23):8619–8624.

Yang, Y., Tarr, M. J., Kass, R. E., and Aminoff, E. M. (2019). Exploring spatiotemporal neural dynamics of the human visual cortex. *Human brain mapping*, 40(14):4213–4238.

Yuan, M. and Lin, Y. (2006). Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67.

Zhang, C., Qiao, K., Wang, L., Tong, L., Hu, G., Zhang, R.-Y., and Yan, B. (2019). A visual encoding model based on deep neural networks and transfer learning for brain activity measured by functional magnetic resonance imaging. *Journal of Neuroscience Methods*, 325:108318.

Zhang, T. and Gallant, J. L. (2020). A naturalistic navigation task reveals rich distributed representations of information across the human cerebral cortex. *Journal of Vision*, 20(11):462–462.

Zhang, T., Gao, J. S., Çukur, T., and Gallant, J. L. (2021). Voxel-based state space modeling recovers task-related cognitive states in naturalistic fmri experiments. *Frontiers in neuroscience*, page 1103.

Zhuang, C., Yan, S., Nayebi, A., Schrimpf, M., Frank, M. C., DiCarlo, J. J., and Yamins, D. L. (2021). Unsupervised neural network models of the ventral visual stream. *Proceedings of the National Academy of Sciences*, 118(3).

A Supplementary materials

A.1 Quantifying feature-space selection with the effective rank

In Section 2.4 and Section 2.7, two examples are proposed to demonstrate the feature-space selection mechanism in banded ridge regression. For the sake of demonstration, these examples use the *effective rank* to quantify the number of feature spaces effectively used in the model. This section explains why the effective rank metric was chosen, and compares alternative definitions of the effective rank.

To quantify the number of feature spaces effectively used in the model, one candidate metric is to count the number of feature spaces with a non-zero effect on the prediction. However, because the hyperparameters optimized by banded ridge regression never reach infinity, the coefficients b^* are never exactly zero, even for feature spaces that are effectively ignored. For this reason, all feature spaces have non-zero values in the variance decomposition, and one cannot simply count the number of non-zero values. To fix this issue, it is possible to define a positive threshold, and to count the number of values above the threshold. Yet, this threshold would have an arbitrary value, and would artificially separate values that are slightly above from values that are slightly below, leading to a discontinuity in the metric. Instead, a better metric of the number of feature spaces effectively used in the model would vary continuously as some feature space contribution goes to zero.

To address this issue, we leverage a metric to compute the rank of a linear system. The rank of a linear system is the number of independent equations in the system, and corresponds to the number of non-zero eigenvalues λ_i. The rank is thus a discontinuous function of the eigenvalues. To remove the discontinuity, the rank can be approximated by the *effective rank*. Several effective rank definitions exist, such as e [Roy and Vetterli, 2007], r_0 [Bartlett et al., 2020], R_0 [Langeberg et al., 2019; Bartlett et al., 2020], and h (proposed in this work). Assuming the eigenvalues are non-negative and sorted in decreasing order ($\lambda_0 \geq \lambda_1 \geq \ldots \geq \lambda_{m-1} \geq 0$), these four metrics are defined as

$$e = \exp(\sum_{i=0}^{m-1} \tilde{\lambda}_i \log(\tilde{\lambda}_i)), \qquad \text{with} \qquad \tilde{\lambda}_i = \frac{\lambda_i}{\sum_{i=0}^{m-1} \lambda_i}, \tag{20}$$

$$r_0 = \frac{\sum_{i=0}^{m-1} \lambda_i}{\lambda_0}, \qquad R_0 = \frac{(\sum_{i=0}^{m-1} \lambda_i)^2}{\sum_{i=0}^{m-1} \lambda_i^2}, \qquad h = 1 + 2\frac{\sum_{i=0}^{m-1} i\lambda_i}{\sum_{i=0}^{m-1} \lambda_i}. \tag{21}$$

All four metrics (e, r_0, R_0, and h) have similar useful properties. They are continuous functions of the m eigenvalues, with values in the interval $[1, m]$. Because they are continuous, they do not require defining any threshold. They are also not affected by additional unused dimensions (dimensions i with $\lambda_i = 0$). They are equal to k when k eigenvalues are equal (and non-zero) and all the other eigenvalues are equal to zero. The four metrics only differ in the way these integer values are interpolated. To describe how they interpolate these integer values, Figure 10 presents the metric values on different 2D and 3D spaces.

To use an effective rank metric to quantify the number of feature spaces effectively used in the model, the following procedure is used. First, the variance decomposition $\rho \in \mathbb{R}^m$ is computed, for example using the product measure as described in Section 1.6. Then, negative values are clipped to zero, and an effective rank metric is computed on the vector ρ. Although all four metrics give similar results, the definition e was used throughout this paper for its intuitive link to the entropy $H = -\sum_i \lambda_i \log(\lambda_i)$ [Roy and Vetterli, 2007; Araki and Lieb, 2002; Bach, 2022]. Because the effective rank estimates the number of feature spaces effectively used over m features spaces, it is noted $\tilde{m}$.

A.2 List of the 22 feature spaces from the short-films dataset

In Section 2.6 and Section 3.8, banded ridge regression is fit on 22 features spaces from the "short-films" dataset [Nunez-Elizalde et al., 2018]. Here, a short description of the 22 feature spaces is given.

- Stimulus motion energy: spatiotemporal Gabor filters computed on the stimulus luminosity (11,845

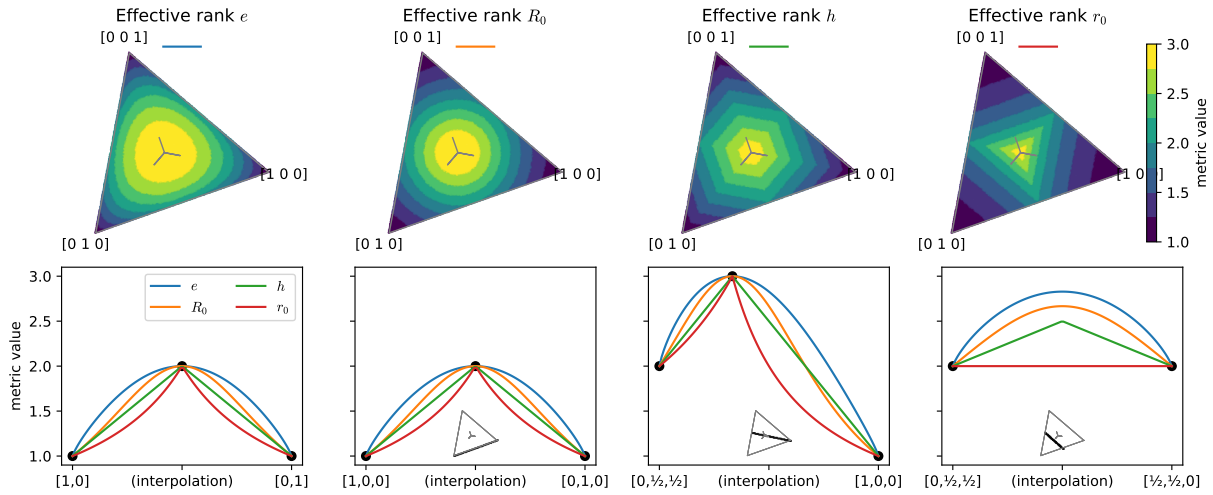

Figure 10: Comparison of four effective rank metrics. Effective rank metrics are useful to quantify the number of feature spaces used in banded ridge regression. **(Top)** Values of the four metrics on the simplex. The variance decompositions are defined on the simplex Δ_{m-1}, the space of vectors in dimension m with positive values that sum to one. With $m = 3$, the simplex is a 2-dimensional surface in the shape of a triangle. All four effective rank metrics are equal to 1 at the triangle corners, to 2 at the edge midpoints, and to 3 at the center. The four metrics differ in the way they interpolate these integer values. The color mapping is discrete to highlight the structures of the different metrics. **(Bottom)** Values of the four metrics on different 1-dimensional trajectories, linearly interpolating between two points in dimension 2 or 3. All four metrics are continuous, and equal to k when the values are equally split over k dimensions (depicted by the black dots). Moreover, adding unused dimensions (second versus first panel) does not alter the metric values. All four metrics are thus reasonable choices for quantifying the number of feature spaces used in banded ridge regression. The definition e was used throughout this paper for its intuitive link to entropy.

features; Nishimoto et al. [2011]).

- Retinotopic motion energy: spatiotemporal Gabor filters, computed on the stimulus luminosity, corrected for eye movements (11,845 features).
- Eye position: cubic polynomial expansion of the eye position during movie watching (36 features).
- Visual semantic: semantic labels of objects and actions present in the scene [Huth et al., 2012], projected in a semantic space (985 features; Huth et al. [2016]).
- Visual thematics (5 feature spaces): categorization of verb classes into categories based on the type of events described. VerbNet (85 features; Kipper et al. [2006]) was used to construct a thematic roles feature space (36 features), a selective restrictions feature space (38 features) and their interaction (71 features), and a verb categories feature space (101 features).
- Spectrogram: spectral power density of the sound wave (2388 features).
- Wavenet: Auditory features related to musical instruments extracted using a pretrained convolutional neural network (512 features; Oord et al. [2016]).
- Word rate: cubic polynomial expansion of word rate (13 features).
- Speech syntax position: part-of-speech tags of each word in the sentences, extracted using a pre-trained neural network (12 features; Andor et al. [2016]).
- Speech syntax labels: word dependencies in the sentence, extracted using a pre-trained neural network (43 features; Andor et al. [2016]).
- Speech semantic: spoken words projected in a semantic space (985 features; Huth et al. [2016]).
- Speech thematics (7 feature spaces): categorization of verb classes into categories based on the type of events described. VerbNet [Kipper et al., 2006] labels were used to construct a thematic role feature space (39 features), a verb class feature space (274 features), and a verb category feature space (102

features). Four thematic roles (verb, actor, recipient, place) were further decomposed by projecting words in a semantic space (985 feature each).

A.3 Banded ridge regression is equivalent to multiple-kernel ridge regression

As stated in Section 2.8, banded ridge regression is equivalent to multiple-kernel ridge regression, up to a subtle difference. This subsection first demonstrates the equivalence, then discusses the source and implications of the difference. Note that a deep understanding of the difference is not critical to the rest of the paper.

Equivalence. To reformulate banded ridge regression as a multiple-kernel ridge regression, one can note that banded ridge regression is equivalent to a ridge regression with weighted feature spaces $X_i = X_i * \sqrt{\mu/\lambda_i}$ (see Section 2.2). This ridge regression is in turn equivalent to a kernel ridge regression [Saunders et al., 1998] (see Section 3.2) with a linear kernel $K = \sum_i X_i X_i^\top = \sum_i X_i X_i^\top (\mu/\lambda_i)$. Writing $K_i = X_i X_i^\top$ and $\gamma_i = \mu/\lambda_i$, the model becomes a multiple-kernel ridge regression

$$w^* = \underset{w}{\operatorname{argmin}} \, || \sum_i \gamma_i K_i w y||_2^2 + \mu w^\top \sum_i \gamma_i K_i w. \tag{22}$$

Multiple-kernel ridge regression corresponds to a kernel ridge regression with a weighted kernel $K = \sum_i \gamma_i K_i$ and a regularization hyperparameter μ. Because μ is arbitrary in the change of variables, one can choose it such that $\sum_i \gamma_i = \sum_i \mu/\lambda_i = 1$, which leads to $\mu = (\sum_i 1/\lambda_i)^{-1}$. In this way, γ is restricted to a subspace called the simplex $\Delta_{m-1} = \{\gamma \in \mathbb{R}_+^m; \sum_i \gamma_i = 1\}$, which corresponds to the standard formulation in the multiple-kernel learning literature [Gönen and Alpaydın, 2011]. Banded ridge regression is thus equivalent to multiple-kernel ridge regression. In Section 3.5, this equivalence is used to solve banded ridge regression efficiently when the number of features is larger than the number of samples.

Key difference. The equivalence between banded ridge regression and multiple-kernel ridge regression hides a subtle difference in the way hyperparameters are optimized. In banded ridge regression, all hyperparameters $\lambda \in \mathbb{R}_+^m$ are optimized through cross-validation. Thus, in the equivalent multiple-kernel ridge regression, both the kernel weights $\gamma \in \Delta_{m-1}$ and the regularization $\mu > 0$ must be optimized through cross-validation. On the contrary, in the multiple-kernel learning literature, the kernel weights γ are usually learned within the training set [Gönen and Alpaydın, 2011], and only μ is learned through cross-validation. Some rare papers have proposed learning kernel weights through cross-validation [Tsuda et al., 2004; Keerthi et al., 2007; Klatzer and Pock, 2015], but this practice is not common in the multiple-kernel learning literature. This difference is important, and is further discussed in the rest of the subsection. To highlight the difference between the two strategies, the following notations is used:

- **MKRR-cv** is a multiple-kernel ridge regression with kernel weights learned over **c**ross-**v**alidation.
- **MKRR-in** is a multiple-kernel ridge regression with kernel weights learned with**in** the training set.

Using these notations, banded ridge regression is equivalent to MKRR-cv, whereas the standard formulation in multiple kernel learning is MKRR-in. Additionally, the squared group lasso is equivalent to MKRR-in [Bach, 2008; Rakotomamonjy et al., 2008].

Group scalings. To understand the difference between MKRR-cv and MKRR-in, it is useful to consider the concept of group scalings. When feature spaces are unbalanced (for example in terms of number of features), the regularization can have a different strength on each feature space. To fix this issue, multiple studies have proposed that the regularization strength on each feature space could be evened out by adding group scalings to the regularization. In the group lasso literature, group scalings are positive weights d_i defined for each feature space i, which creates a scaled $L_{2,1}$ norm $\sum_i d_i ||b_i||_2$. Similarly, in the multiple kernel learning literature, group scalings modify the optimization by constraining kernel weights to a scaled simplex $\tilde{\Delta}_{m-1} = \{\gamma \in \mathbb{R}_+^m; \sum_i \gamma_i d_i^2 = 1\}$ [Bach, 2008]. Note that group scalings are defined additionally to the kernel weights γ_i.

Because feature spaces can be unbalanced in different ways, many group scalings have been proposed in the group lasso and multiple-kernel learning literature: the square root of the number of features [Yuan and Lin,

2006], the kernel trace [Lanckriet et al., 2004], the square root of the kernel trace [Bach et al., 2004], the kernel empirical rank [Bach et al., 2005], or more sophisticated group scalings (*e.g.* [Bach, 2008; Obozinski et al., 2011; Pan and Zhao, 2016]). However, no group scaling emerged as being superior to the others. In MKRR-in, it is thus necessary to test different group scalings, and select the best one through cross-validation. On the contrary, in MKRR-cv the kernel weights $\gamma \in \Delta_{m-1}$ are learned by cross-validation. Thus, group scalings would have no effect on MKRR-cv. In other words, MKRR-cv is equivalent to using MKRR-in with group scalings optimized via cross-validation.

Summary. To summarize this subsection, MKRR-cv and MKRR-in are two variants of multiple-kernel ridge regression. In MKRR-cv, the kernel weights are learned using cross-validation, whereas in MKRR-in, the kernel weights are learned within the training set. Banded ridge regression is equivalent to MKRR-cv, whereas the squared group lasso is equivalent to MKRR-in. In MKRR-in, one can add group scalings to help balance the feature spaces, but the choice of group scaling needs to be cross-validated. In MKRR-cv, group scalings are less important because optimal kernel weights are already learned through cross-validation.

A.4 Derivation of the hyperparameter gradient

In Section 3.6, hyperparameter gradient descent is used to optimize hyperparameters in multiple-kernel ridge regression. Hyperparameter gradient descent is a method that iteratively improves hyperparameters. At each iteration, the update is based on the gradient of the cross-validation loss L_{val} with respect to hyperparameters δ. Here, the computation of this gradient is derived using the implicit differention framework.

As explained in Section 3.6, multiple-kernel ridge regression is reparameterized with hyperparameters $\delta \in \mathbb{R}^m$. The training and validation loss functions are defined as

$$\mathcal{L}_{\text{train}}(w, \delta) = \left\| \sum_{i=1}^{m} e^{\delta_i} K_{\text{train},i} w - y_{\text{train}} \right\|_2^2 + w^\top \sum_{i=1}^{m} e^{\delta_i} K_{\text{train},i} w, \tag{23}$$

$$\mathcal{L}_{\text{val}}(w^*(\delta), \delta) = \left\| \sum_{i=1}^{m} e^{\delta_i} K_{\text{val},i} w^*(\delta) - y_{\text{val}} \right\|_2^2, \tag{24}$$

where $w^*(\delta) = \operatorname{argmin}_w \mathcal{L}_{\text{train}}(w, \delta)$, and for each feature space i, $K_{\text{train}}, i = X_{\text{train}}, i X_{\text{train}}^\top, i$ is the training kernel, and $K_{\text{val}}, i = X_{\text{val}}, i X_{\text{train}}^\top, i$ is the validation cross-kernel.

Implicit differentiation framework. At each iteration, gradient descent updates the hyperparameters δ in the direction of the gradient of the cross-validation loss. The cross-validation loss $\mathcal{L}_{\text{val}}(w^*(\delta), \delta)$ has two dependencies in δ, one *direct*, and one *indirect* through the dual weights $w^*(\delta)$. The gradient of $\mathcal{L}_{\text{val}}$ with respect to δ is thus the sum of the direct gradient $\frac{\partial \mathcal{L}_{\text{val}}}{\partial \delta}$, and the indirect gradient $\frac{\partial \mathcal{L}_{\text{val}}}{\partial w^*} \frac{\partial w^*}{\partial \delta}$. In the indirect gradient, the Jacobian $\frac{\partial w^*}{\partial \delta}$ can be computed with *implicit differentiation* [Larsen et al., 1996; Chapelle et al., 2002; Foo et al., 2007], which leads to the full gradient expression

$$\frac{\partial \mathcal{L}_{\text{val}}^*}{\partial \delta} = \frac{\partial \mathcal{L}_{\text{val}}}{\partial \delta} - \frac{\partial \mathcal{L}_{\text{val}}}{\partial w^*} \left(\frac{\partial^2 \mathcal{L}_{\text{train}}}{\partial w \partial w^\top} \right)^{-1} \frac{\partial^2 \mathcal{L}_{\text{train}}}{\partial w \partial \delta^\top}. \tag{25}$$

In this expression, the most expensive term to compute is the inverse Hessian $(\frac{\partial^2 \mathcal{L}_{\text{train}}}{\partial w \partial w^\top})^{-1}$. To reduce this computational bottleneck, three separate approximations are considered. The first approximation, proposed in [Pedregosa, 2016], uses a conjugate gradient method to solve $\frac{\partial \mathcal{L}_{\text{val}}}{\partial w^*} (\frac{\partial^2 \mathcal{L}_{\text{train}}}{\partial w \partial w^\top})^{-1}$ up to a limited precision. The second approximation, proposed in [Lorraine et al., 2019], uses Neumann series to estimate the inverse Hessian with $H^{-1} \approx \sum_{j=0}^{k} (I_n - H)^j$, with typically $k = 5$. The third approximation is more aggressive, and approximates the full gradient with the direct gradient $\frac{\partial \mathcal{L}_{\text{val}}}{\partial \delta}$. This last approximation is equivalent to assuming the inverse Hessian to be 0. In many applications, this approximation is not available because the direct gradient is equal to zero [Lorraine et al., 2019]. However, the direct gradient is not equal to zero in multiple-kernel ridge regression, so it can be used to approximate the full gradient.

Application to multiple-kernel ridge regression. The implicit differentiation framework can be immediately applied to multiple-kernel ridge regression. Yet, some improvements can be done to maximize efficiency. For example, the gradient of the training loss at optimum $\frac{\partial \mathcal{L}_{\text{train}}}{\partial w}\Big|_{w^*} = K_{\text{train}}((K_{\text{train}} + I_n)w^* - y_{\text{train}}) = 0$ can be simplified to $(K_{\text{train}} + I_n)w^* - y_{\text{train}} = 0$ by using the gradient with respect to the scalar product $\langle x, y\rangle_K = x^\top K y$. This change improves gradient conditioning, and leads to the same solutions when the kernel has full rank. In addition, this change largely simplifies the following derivative expressions and decreases their computational cost

$$\frac{\partial \mathcal{L}_{\text{train}}}{\partial w} = (K_{\text{train}} + I_n)w - y_{\text{train}}, \qquad \frac{\partial^2 \mathcal{L}_{\text{train}}}{\partial w \partial w^\top} = K_{\text{train}} + I_n, \qquad \frac{\partial^2 \mathcal{L}_{\text{train}}}{\partial w \partial \delta_i} = e^{\delta_i} K_{\text{train},i} w. \tag{26}$$

The other derivative expressions are not affected by this change, and are equal to

$$\frac{\partial \mathcal{L}_{\text{val}}}{\partial w} = K_{\text{val}}(K_{\text{val}} w - y_{\text{val}}), \qquad \frac{\partial \mathcal{L}_{\text{val}}}{\partial \delta_i} = e^{\delta_i}(K_{\text{val}} w - y_{\text{val}})(K_{\text{val},i} w)^\top. \tag{27}$$

Once the gradient $\frac{\partial \mathcal{L}^*_{\text{val}}}{\partial \delta}$ is computed, gradient descent improves the hyperparameters δ with the update $\delta \leftarrow \delta - \eta \frac{\partial \mathcal{L}^*_{\text{val}}}{\partial \delta}$, where η is called the step size. To estimate the optimal step size, one could use the Lipschitz constant of $\frac{\partial \mathcal{L}^*_{\text{val}}}{\partial \delta}$, but that is hard to estimate [Pedregosa, 2016]. Fortunately, the Lipschitz constant of the direct gradient $\frac{\partial \mathcal{L}_{\text{val}}}{\partial \delta}$ can be estimated. Noting $\chi_i = e^{\delta_i} K_{\text{val},i} w$, the validation loss reads $\mathcal{L}_{\text{val}} = ||\sum_{i=1}^m \chi_i - y_{\text{val}}||_2^2$. Using $\frac{\partial \chi_i}{\partial \delta_i} = \chi_i$, the gradient and Hessian are derived as

$$\frac{\partial \mathcal{L}_{\text{val}}}{\partial \delta_i} = \chi_i^\top (\sum_{j=1}^m \chi_j - y_{\text{val}}), \qquad \frac{\partial^2 \mathcal{L}_{\text{val}}}{\partial \delta_i \partial \delta_j} = \chi_i^\top \chi_j + \chi_i^\top (\sum_{k=1}^m \chi_k - y_{\text{val}})[i = j], \tag{28}$$

where $[i = j]$ is equal to 1 if $i = j$ and 0 otherwise. The local Lipschitz constant L is given by the maximum eigenvalue of the Hessian, which can be estimated with power iteration. In our experiments, a step-size $\eta = 1/L$ was used, not only for the direct gradient, but also for the full gradient approximations. Several adaptive step-size strategies were also explored, but none were computationally as efficient as using the estimated Lipschitz constant of the direct gradient. The entire algorithm is listed in pseudo-code in Appendix A.7.

A.5 Hyperparameter Bayesian search

In Section 3.5 and Section 3.6, two methods are proposed to solved banded ridge regression. Another method proposed in the past to solve banded ridge regression is hyperparameter Bayesian search [Nunez-Elizalde et al., 2019]. Hyperparameter Bayesian search [Mockus et al., 1978] is a method similar to hyperparameter random search. Both search methods compare a number of hyperparameter candidates and select the candidates with the lowest cross-validation error. The specificity of Bayesian search is that each candidate is selected conditionally to the performances of previously tested candidates. The *Tikreg* Python package [Nunez-Elizalde et al., 2019] published previously by our group implements Bayesian search to solve banded ridge regression. Specifically, Tikreg implements a Bayesian search using trees of Parzen estimators [Bergstra et al., 2011] through the Hyperopt Python package [Bergstra et al., 2013].

A major limitation of hyperparameter Bayesian search is that it cannot optimize multiple losses simultaneously. For this reason, Tikreg cannot optimize a separate loss per voxel. Instead, Tikreg optimizes the *average* validation loss over all voxels. This approximation introduces a bias towards the preferred hyperparameters of the voxel majority. To mitigate this bias, Tikreg later selects the best hyperparameters per voxel out of all candidates explored by Bayesian search. However, this mitigation will only be successful if Bayesian search explores the hyperparameter space broadly and does not find its optimum quickly.

Another limitation of Tikreg's implementation of Bayesian search is that it does not use all computational tricks to solve the ridge regression subproblems. As described in Section 3.1, ridge regression can be solved efficiently by reusing computations for multiple regularization hyperparameters. To leverage this computational trick in random search (Section 3.5), hyperparameters are separated into $\gamma \in \Delta_{m-1}$ and $\mu > 0$.

For each randomly-sampled candidate γ, a ridge regression is solved efficiently for a large number of μ. This factorization is key to explore hyperparameter space efficiently, but is not present in Tikreg's implementation.

A.6 Algorithms for weighted-kernel ridge regression

To solve multiple-kernel ridge regression, the hyperparameter gradient descent solver updates the kernel weights $\delta \in \mathbb{R}^m$ at each iteration. Then, at each iteration, the regression weights w must be optimized for the current (fixed) kernel weights $\delta \in \mathbb{R}^m$. We call this sub-problem *weighted-kernel ridge regression*. Specifically, for a *fixed* $\delta \in \mathbb{R}^m$, weighted-kernel ridge regression is defined as:

$$w^* = \underset{w}{\operatorname{argmin}} \left\| \sum_{i=1}^{m} e^{\delta_i} K_i w - y \right\|_2^2 + \sum_{i=1}^{m} e^{\delta_i} w^\top K_i w. \tag{29}$$

This section describes how to solve this sub-problem.

This problem has a closed formed solution $w^* = (K + I_n)^{-1} y$, where K is the kernel weighted sum $K = \sum_{i=1}^{m} e^{\delta_i} K_i$. However, if different voxels have different kernel weights δ, this closed-form solution is not efficient, as it requries computing K and inverting $(K + I_n)^{-1}$ for each different kernel weights δ. During hyperparameter gradient descent, a precise solution of the weighted-kernel ridge regression sub-problem is not necessary, and an approximation is sufficient [Pedregosa, 2016]. Thus, an approximate solution can be obtained with gradient descent (Algorithm 1), conjugate gradient descent (Algorithm 3), or even finite Neumann series (Algorithm 2). (Neumann series approximates $(K + I_n)^{-1} \approx f \sum_{j=1}^{k} (I_n - f(K + I_n))^j$, where the factor $0 < f < 1$ needs to be tuned to make the series converge.) All three algorithms are presented here for a single voxel, yet they can be straightforwardly extended to multiple voxels using matrices and tensors.

Interestingly, another sub-problem of hyperparameter gradient descent can be solved as a weighted-kernel ridge regression. During hyperparameter gradient descent, the gradient computation requires computing $\frac{\partial \mathcal{L}_{\text{val}}}{\partial w^*} \left(\frac{\partial^2 \mathcal{L}_{\text{train}}}{\partial w \partial w^\top} \right)^{-1}$; in multiple-kernel ridge regression, this term is equal to $K_{\text{val}}(K_{\text{val}} w - y_{\text{val}})(K_{\text{train}} + I_n)^{-1}$. This expression can be considered as the solution of equation (29), with $y = K_{\text{val}}(K_{\text{val}} w - y_{\text{val}})$. Therefore, the three approximate solvers of weighted-kernel ridge regression presented in this section can also be used to compute this element of the hyperparameter gradient computation.

In the example presented in Section 3.8, the ridge dual weights were updated with Algorithm 1 because Algorithm 2 was too coarse and Algorithm 3 led to oscillations in the convergence curves. For the hyperparameter gradient term, because the term did not require a precise approximation, all three algorithms (Algorithm 1, Algorithm 3, and Algorithm 2) were tested and compared.

Algorithm 1 Weighted-kernel ridge regression solver - Gradient descent

```
Input: target y, kernels {K_i}_i, fixed kernel weights δ, tolerance ε, initial dual weights w
compute L(K_i)                                          // Lipschitz constants
η = 1/(Σ_{i=1}^m e^{δ_i} L(K_i) + 1)                    // step size
Δw = ∞
while max(|Δw/w|) > ε do
    ŷ = Σ_{i=1}^m e^{δ_i} (K_i w)                       // prediction
    Δw = −η(ŷ − y + w)                                  // update
    w = w + Δw                                          // gradient step
end while
Output: weights w
```

Algorithm 2 Weighted-kernel ridge regression solver - Neumann series

Input: target y, kernels $\{K_i\}_i$, fixed kernel weights δ, number of terms k, factor f
$w = 0$ // sum accumulator
$p = y$ // product accumulator
for $j = 1$ **to** k **do**
  $Kp = \sum_{i=1}^{m} e^{\delta_i}(K_i p)$
  $p = p(1 - f) - fKp$
  $w = w + p$
end for
$w = fw$ // rescaling factor
Output: weights w

Algorithm 3 Weighted-kernel ridge regression solver - Conjugate gradient

Input: target y, kernels $\{K_i\}_i$, fixed kernel weights δ, tolerance ε, initial dual weights w
$\hat{y} = \sum_{i=1}^{m} e^{\delta_i}(K_i w)$ // prediction
$r_1 = -(\hat{y} - y + w)$ // negative gradient
$p = r_1$
$\Delta w = \infty$
for $k = 1$ **to** #iterations **do**
  $Kp = \sum_{i=1}^{m} e^{\delta_i}(K_i p)$
  $\alpha = r_k^\top r_k / (p^\top Kp)$
  $\Delta w = \alpha p$ // update
  $w = w + \Delta w$ // conjugate step
  if $\max(|\Delta w / w|) \leq \varepsilon$ **then**
    break for loop
  end if
  $r_{k+1} = r_k - \alpha Kp$
  $\beta = (r_{k+1}^\top r_{k+1}) / (r_k^\top r_k)$
  $p = r_{k+1} + \beta p$
end for
Output: weights w

A.7 Algorithms for multiple-kernel ridge regression

This section describes algorithms to solve multiple-kernel ridge regression, optimizing $\delta \in \mathbb{R}^m$ over the validation loss:

$$\delta^* = \underset{\delta}{\operatorname{argmin}} \left\| \sum_{i=1}^{m} e^{\delta_i} K_{\text{val},i} w^*(\delta) - y_{\text{val}} \right\|_2^2, \tag{30}$$

where $w^*(\delta)$ is the solution of the weighted-kernel ridge regression (29). This problem can be either solved with hyperparameter random search (Algorithm 4) or with hyperparameter gradient descent (Algorithm 5).

Algorithm 4 Multiple-kernel ridge regression solver - Hyperparameter random search

Input: target Y, kernels $\{K_i\}_i$, Dirichlet's α, range of μ, cross-validation splits
for all voxels j **do**
  $\phi^*[j] = -\infty$ // initialization
  $\gamma^*[j] = \text{nan}$
  $\mu^*[j] = \text{nan}$
end for
for $\gamma \sim \text{Dirichlet}(\alpha)$ **do**
  $K = \sum_{i=1}^{m} \gamma_i K_i$ // kernel
  for all cross-validation splits s **do**
    $K_{\text{train}}, K_{\text{val}}, Y_{\text{train}}, Y_{\text{val}} = \text{split}_s(K, Y)$
    $UDU^\top = K_{\text{train}}$ // diagonalization
    for all regularizations μ **do**
      $M_\mu = K_{\text{val}} U (D + \mu_k I_n) U^\top$
      for all voxels j **do**
        $\hat{Y}_{\text{val}}[\mu, j] = M_\mu Y_{\text{train}}[j]$ // prediction
        $\phi[s, \mu, j] = ||Y_{\text{val}}[j] - \hat{Y}_{\text{val}}[\mu, j]||_2^2$
      end for
    end for
  end for
  $\bar{\phi}[\mu, j] = \text{mean}_s \phi[s, \mu, j]$ // mean over splits
  $\phi^*_\gamma[j] = \min_\mu \bar{\phi}[\mu, j]$ // min over mu
  $\mu^*_\gamma[j] = \text{argmin}_\mu \bar{\phi}[\mu, j]$
  for all voxels j **do**
    if $\phi^*[j] > \phi^*_\gamma[j]$ **then**
      $\phi^*[j] = \phi^*_\gamma[j]$ // update if improvement
      $\mu^*[j] = \mu^*_\gamma[j]$
      $\gamma^*[j] = \gamma$
    end if
  end for
end for
Output: $\gamma^*[j], \mu^*[j]$ for all voxels j

Algorithm 5 Multiple-kernel ridge regression solver - Hyperparameter gradient descent

Input: kernels $\{K_i\}_i$, initial δ, single target y
for k = 1 **to** #iterations **do**
  for all cross-validation splits s **do**
    $K_{\text{train}}, K_{\text{val}}, y_{\text{train}}, y_{\text{val}} = \text{split}_s(K, y)$

    // Update ridge dual weights w_s
    $w_s =$ Algorithm 1 $(y_{\text{train}}, \{K_{\text{train},i}\}_i, \delta, \varepsilon_k, w_s)$

    // Direct gradient
    $\hat{y}_{\text{val},i} = e^{\delta_i}(K_{\text{val},i} w_s)$ // partial prediction
    $r_{\text{val}} = (\sum_{i=1}^{m} \hat{y}_{\text{val},i} - y_{\text{val}})$ // residual
    $\frac{\partial \mathcal{L}_{\text{val}}}{\partial \delta_i} = r_{\text{val}} \hat{y}_{\text{val},i}$

    // Indirect gradient
    $\frac{\partial \mathcal{L}_{\text{val}}}{\partial w} = \sum_{i=1}^{m} e^{\delta_i}(K_{\text{val},i} r_{\text{val}})$
    $\omega_s =$ Algorithm 1, 2, or 3 $(\frac{\partial \mathcal{L}_{\text{val}}}{\partial w}, \{K_{\text{train},i}\}_i, \delta, \varepsilon_k, \omega_s)$
    $\hat{y}_{\text{train},i} = e^{\delta_i}(K_{\text{train},i} w_s)$ // partial prediction
    $\frac{\partial \mathcal{L}_{\text{val}}}{\partial w^*} \frac{\partial w^*}{\partial \delta} = \omega_s^\top \hat{y}_{\text{train},i}$

    // Full gradient
    $\frac{\partial \mathcal{L}^*_{\text{val}}}{\partial \delta_i}|_s = \frac{\partial \mathcal{L}_{\text{val}}}{\partial \delta_i} + \frac{\partial \mathcal{L}_{\text{val}}}{\partial w^*} \frac{\partial w^*}{\partial \delta}$

    // Step size
    $L_s =$ power iteration on the Hessian (28)
    $\eta_s = 1/L_s$
  end for
  $\delta = \delta - (\min_s \eta_s)(\text{mean}_s \frac{\partial \mathcal{L}^*_{\text{val}}}{\partial \delta_i}|_s)$
end for