CompSci Project 1:
Regression Analysis and Resampling Methods

Kosio Beshkov*
Department of Bioscience, University of Oslo, Norway

Mohamed Safy†
Department of Chemistry, University of Oslo, Norway

Tim Zimmermann‡
Institute of Theoretical Astrophysics, University of Oslo, Norway
(Dated: September 1, 2024)

Purpose of this work is to illustrate the capabilities and theoretical properties of foundational, partially self-implemented, supervised learning methods and associated evaluation tools for both regression and classification problems. Our in-depth investigations may be separated into three, loosely connected, aspects:

Linear Regression: We investigate the behavior of *Ordinary Least Squares*, *Ridge Regression* and *LASSO* in the context of fitting noisy realisations of Franke's function. Special attention will be drawn to the problem of model selection & assessment. To this end, from scratch-implementations of established resampling techniques are employed to (i) select the optimal model parameters and (ii) understand the properties of the model's expected prediction error and associated error contributions as a function of their (hyper)parameter space.

Minimization Techniques: *Stochastic Gradient Descent* (SGD), a randomized, low cost variant of the infamous steepest decent algorithm is introduced in the context of cost function minimization for regression parameter estimation when no analytical solution is available. We benchmark our implementation's performance against (i) analytical results and (ii) an industry-grade implementation of SGD in scikit-learn.

Binary Classification: We derive/describe the cost functions and algorithmic steps for (i) *Logistic Regression* and (ii) *Support Vector Machines* (SVM) and evaluate their respective performance on a popular classification data set (Wisconscin Breast Cancer data set). This includes a discussion on how the features in this data set relate to each other and how many of them are really needed to achieve good classification performance.

I. INTRODUCTION

Point of departure for our discussion is a reasonably self-contained introduction of *linear regression* in the context of Franke's function, [1], i.e. the problem of reconstructing — or fitting — a continuous function from noisy data by means of a linear model. After recapitulating the main assumptions of linear regression, we proceed by formulating its solution as a *cost function minimization problem* and explore both theoretical properties and practical details of well-known unconstrained (*Ordinary Least Squares*) and constrained solutions (*Ridge Regression, LASSO*) to it.

In the same vein, section I B presents *classification*, i.e. the prediction of categorical responses, in the context of the Wisconsin Breast Cancer data set, [2]. Following a brief description of *Logistic Regression* along with its underlying model and assumptions, special attention will be drawn to *stochastic gradient descent* — a numerical minimization algorithm applicable if closed form, analytical solutions are not available. We close our discussion with a brief exposition of *support vector machines* in section I B 2.

Section I C discusses the important subject of model selection and assessment and collects important theoretical results, most notably the *bias-variance-decomposition* of the *expected prediction error*, section I C 1, as well as procedures to *estimate* the latter, see section I C 2.

We conclude our introductory remarks in section I D, with a quick summary of implementational details of the code used for the detailed numerical study of section VI.

Important terminology and nomenclature will be introduced along the way.

A. Linear Regression

Suppose we are given a realization of a noisy measurement vector $\boldsymbol{z} \in \mathbb{R}^N$ of *Franke function* values $f\colon \mathbb{R}^2 \to \mathbb{R}$,

* constb@ibv.uio.no
† m.e.a.safy@kjemi.uio.no
‡ tim.zimmermann@astro.uio.no

with,

$$
\begin{aligned}
f(\boldsymbol{x}) = &\frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) \\
&+ \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)^2}{10}\right) \\
&+ \frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) \\
&- \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right) ,
\end{aligned}
\tag{1}
$$

at randomly chosen but fixed observation sights $\{\boldsymbol{x}_i^\intercal = \begin{pmatrix} x_i & y_i \end{pmatrix}^\intercal\}_{i=1..N_\mathcal{T}} \in [0,1)^2$. Moreover, assume each observation z_i may be written as:

$$
z_i = f(\boldsymbol{x}_i) + \epsilon_i \ , \quad \epsilon_i \mid \boldsymbol{x}_i \sim \mathcal{N}(0,\sigma) \ , \tag{2}
$$

with ϵ_i denoting iid random variables — the noise — drawn from a normal distribution of common variance σ^2. We refer to Figure 1 for a visualization. Note that the only source of stochasticity is due to $\boldsymbol{\epsilon}$ and observation sights are understood as given in the context of regression.

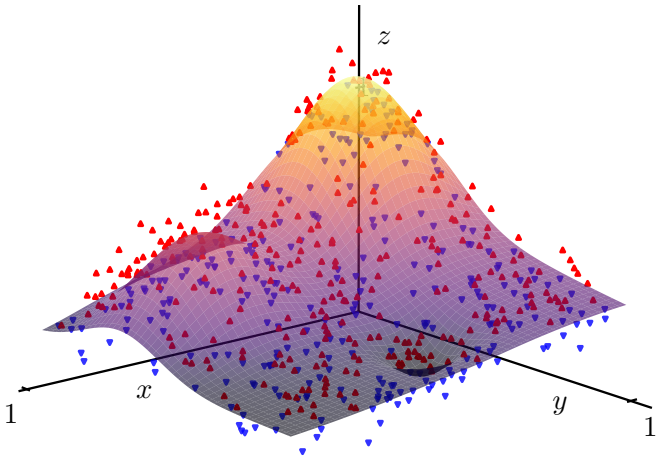

FIG. 1. Franke's function on $\mathcal{D} = [0,1)^2$ alongside a measurement of it contaminated by gaussian noise $\mathcal{N}(0, 0.1)$. Red triangles lie above, blue triangles below the function surface.

Given such an observation vector, *linear* regression attempts to model the deterministic function f — in our case Franke's function — by means of a predictor function $\hat{f}\colon \mathbb{R}^2 \to \mathbb{R}$ that can be written as a linear combination of so called *features* X_k. In what follows, we assume $\hat{f}$ to be a two-dimensional polynomial of maximal degree p. Thus, our regression model takes the form:

$$
\begin{aligned}
\hat{f}(\boldsymbol{x}) &= \beta_0 + \sum_{i=1}^{p}\sum_{j=0}^{i} \beta_{k(i,j)} x^j y^{i-j} \\
&= \beta_0 + \sum_{i=1}^{p}\sum_{j=0}^{i} \beta_{k(i,j)} X_{k(i,j)}(\boldsymbol{x}) \ ,
\end{aligned}
\tag{3}
$$

with $\boldsymbol{\beta} \equiv \begin{pmatrix} \beta_1 & \dots & \beta_P \end{pmatrix}^\intercal$ denoting the yet unspecified *regression parameters* and β_0 the model's intercept, which we treat independently — a convention which will become clear in section I A 2. Note that (i) the total number of features P is set by the chosen maximal degree p according to $P(p) = \frac{1}{2}(p+1)(p+2)$, (ii) the index function $k(i,j) = P(i-1) + j$ and (iii) each monomial provides one feature $X_{k(i,j)}(\boldsymbol{x}) \equiv x^j y^{p-j}$.

Evaluating the predictor function at all sights $\{\boldsymbol{x}_i\}_{i=1..N}$ may be conveniently written as:

$$
\tilde{\boldsymbol{z}} = \beta_0 \mathbf{1} + \boldsymbol{X}\boldsymbol{\beta} \ , \tag{4}
$$

such that one observation of all features represents one row in the *design matrix* $\boldsymbol{X} \in \mathbb{R}^{N_\mathcal{T} \times (P-1)}$. Notice we choose to *exclude* the constant intercept column.

What remains is to infer — or learn — the yet unknown regression parameters. This is achieved by minimizing an algorithm-specific cost function $C(\boldsymbol{r}, \boldsymbol{\lambda})$, dependent on the residual error $\boldsymbol{r} \equiv \boldsymbol{z} - \tilde{\boldsymbol{z}}(\beta_0, \boldsymbol{\beta})$ and, potentially, other *hyperparameters* $\boldsymbol{\lambda}$. Thus, all regression algorithms are concerned with solving:

$$
\{\hat{\beta}_0, \hat{\boldsymbol{\beta}}\} = \underset{\{\beta_0, \boldsymbol{\beta}\}}{\arg\min}\, C(\boldsymbol{r}(\beta_0, \boldsymbol{\beta}), \boldsymbol{\lambda}) \ . \tag{5}
$$

In what follows, we distinguish three types of cost functions, each of which yielding a different *estimator* $\hat{\boldsymbol{\beta}}$ to the regression parameter vector $\boldsymbol{\beta}$.

1. *Ordinary Least Squares*

As the name suggests, in ordinary least squares (OLS) one chooses the cost function:

$$
C_{\text{OLS}}(\beta_0, \boldsymbol{\beta})) = ||\tilde{\boldsymbol{z}} - \boldsymbol{z}(\beta_0, \boldsymbol{\beta})||_2^2 \ , \tag{6}
$$

and the solution to eq. (5), found by satisfying $\nabla_{\boldsymbol{\beta}} C_{\text{OLS}} = 0$, enjoys a closed form, analytical expression:

$$
\hat{\boldsymbol{\beta}}_{\text{OLS}} = (\boldsymbol{X}^\intercal \boldsymbol{X})^+ \boldsymbol{X}^\intercal \boldsymbol{z} \ , \quad \hat{\beta}_0 = \bar{z} - \bar{\boldsymbol{X}}^\intercal \hat{\boldsymbol{\beta}}_{\text{OLS}} \ , \tag{7}
$$

with $(\boldsymbol{X}^\intercal \boldsymbol{X})^+$ as the unique Moore-Penrose pseudoinverse of the symmetric matrix $\boldsymbol{X}^\intercal \boldsymbol{X}$ and $\bar{X}_k = N_\mathcal{T}^{-1} \sum_i X_{ik}$ denoting the elements of the observation averaged feature vector $\bar{\boldsymbol{X}} \in \mathbb{R}^{P-1}$.

A couple of remarks are in order:

Firstly, the strict convexity of eq. (6) in the regression parameters implies that the solution in eq. (7) is *unique*.

Secondly, if we happen to find ourselves in the "data rich" regime $N_\mathcal{T} > P$ and don't suffer from super-colinear feature columns, $\boldsymbol{X}^\intercal \boldsymbol{X}$ is non-singular. In this case, eq. (7) reduces to the well-known maximum likelihood estimator $\hat{\boldsymbol{\beta}}_{\text{ML}} = (\boldsymbol{X}^\intercal \boldsymbol{X})^{-1} \boldsymbol{X}^\intercal \boldsymbol{z}$, see [3].

Thirdly, if the singular value decomposition (SVD) of $\boldsymbol{X}^\intercal \boldsymbol{X} = \boldsymbol{U}\Sigma\boldsymbol{V}^\intercal$, then its pseudoinverse $(\boldsymbol{X}^\intercal \boldsymbol{X})^+$ is:

$$
(\boldsymbol{X}^\intercal \boldsymbol{X})^+ = \boldsymbol{U}\boldsymbol{\Sigma}^+\boldsymbol{V}^\intercal \ , \quad \Sigma_{ij}^+ = \begin{cases} \Sigma_{ij}^{-1} & \text{if } \Sigma_{ij} \neq 0 \\ 0 & \text{else} \end{cases} \ . \tag{8}
$$

A practical implementation of this approach is provided by `numpy`'s `pinv`-function.

Finally, if we assume (temporally) that the sought after function is indeed linear and $\boldsymbol{\beta}^*$ its coefficient vector, then it's straightforward to see that the OLS *estimator* of eq. (7) is *unbiased*,

$$\text{Bias}\left[\hat{\boldsymbol{\beta}}_{\text{OLS}}\right] \equiv \boldsymbol{\beta}^* - \mathbb{E}\left[\hat{\boldsymbol{\beta}}_{\text{OLS}}\right] = 0\,, \tag{9}$$

and has variance:

$$\text{Cov}\left[\hat{\boldsymbol{\beta}}_{\text{OLS}}, \hat{\boldsymbol{\beta}}_{\text{OLS}}\right] = \sigma^2 \left(\boldsymbol{X}^\intercal \boldsymbol{X}\right)^{-1}\,. \tag{10}$$

We return to eq. (10) in Section VI A 1 when we analyze OLS's bias-variance trade off and its dependence on the observation vector size $N_{\mathcal{T}}$.

2. Ridge Regression

Ridge regression amends the OLS cost function in eq. (6) by a L_2-regularization term for the non-intercept regression parameters:

$$C_{\text{R}}\left(\beta_0, \boldsymbol{\beta}\right) = ||\tilde{\boldsymbol{z}} - \boldsymbol{z}(\beta_0, \boldsymbol{\beta})||_2^2 + \lambda||\boldsymbol{\beta}||_2^2\,, \tag{11}$$

with $\lambda > 0$ as global penalization parameter acting on all regression parameters *except* the intercept. We exclude the intercept to enforce "translation invariance" of our prediction, i.e. adding a constant to our observation $\boldsymbol{z}$ results in our prediction $\tilde{\boldsymbol{z}}$ being shifted by the same amount, cf. [4], — clearly a desirable property.

Again, $\nabla_{\boldsymbol{\beta}} C_{\text{R}} = 0$ implies an unique, closed-form, analytical solution for eq. (5) with eq. (11) as cost function. One finds:

$$\hat{\boldsymbol{\beta}}_{\text{R}}(\lambda) = \left(\boldsymbol{X}^\intercal \boldsymbol{X} + \lambda \mathbb{1}\right)^{-1} \boldsymbol{X}^\intercal \boldsymbol{z}\,, \quad \hat{\beta}_0 = \bar{z} - \bar{\boldsymbol{X}}^\intercal \hat{\boldsymbol{\beta}}_{\text{R}}(\lambda)\,. \tag{12}$$

It is wort emphasizing that expression (12) is well-defined for all design matrices $\boldsymbol{X}$: Since the identity matrix commutes with all matrices, a common eigen-decomposition, $\boldsymbol{X}^\intercal \boldsymbol{X} + \lambda \mathbb{1} = \boldsymbol{U}\ \mathbf{Diag}[\sigma_i^2 + \lambda]\ \boldsymbol{U}^T$, exists. By definition $\lambda > 0$ implying all eigenvalues are strictly positive which assures the existence of the matrix inverse that enters the Ridge estimator in eq. (12). Using `numpy`'s `inv`-function is hence sufficient to implement eq. (12). In fact, this result was the initial motivation to add a L_2-regularizer to eq. (6) as it allows to extend the maximum-likelihood estimator to singular $\boldsymbol{X}^\intercal \boldsymbol{X}$, [5].

Aside from this *ad-hoc* fix for $\hat{\boldsymbol{\beta}}_{\boldsymbol{ML}}$, non-zero values of the penalty parameter λ lead to a *shrinkage of the regression* parameters, which can easily be inferred by considering a SVD of eq. (12), see [3] for the calculation.

Our focus, however, will lie on what implications the addition of a regularizer has for the expected squared error, cf. section I C. At this stage, we can already report that the inclusion of an L_2-penalty term induces bias (i.e. if f is linear):

$$\text{Bias}\left[\hat{\boldsymbol{\beta}}_{\text{R}}(\lambda)\right] = \left(\boldsymbol{U}\mathbf{Diag}\left[\frac{\sigma_i^2}{\sigma_i^2 + \lambda}\right]\boldsymbol{U}^\intercal - \mathbb{1}\right)\boldsymbol{\beta}^*\,, \tag{13}$$

but decreases the variance of the *estimator*:

$$\text{Cov}\left[\hat{\boldsymbol{\beta}}_{\text{R}}(\lambda), \hat{\boldsymbol{\beta}}_{\text{R}}(\lambda)\right] = \sigma^2 \boldsymbol{U}\mathbf{Diag}\left[\frac{\sigma_i^2}{(\sigma_i^2 + \lambda)^2}\right]\boldsymbol{U}^\intercal\,. \tag{14}$$

We return to these equations in section VI A 2 when we analyze the asymptotic behavior of the *predictor's* bias-variance decomposition, see section I C 1, as a function of the penalty parameter λ.

3. LASSO

LASSO regression trades the L_2-regularizer for a L_1-penalty:

$$C_{\text{L}}\left(\boldsymbol{r}(\beta_0, \boldsymbol{\beta})\right) = ||\tilde{\boldsymbol{z}} - \boldsymbol{z}(\beta_0, \boldsymbol{\beta})||_2^2 + \lambda||\boldsymbol{\beta}||_1\,. \tag{15}$$

Note that although eq. (15) is convex in $\boldsymbol{\beta}$, it is not differentiable. Thus, no analytical results exist and one is confined to suitable numerical minimization techniques, see [6]. For the scope of this work, we restrict ourselves to `sklearn`'s `Lasso` regression class and fit functionality therein.

Qualitatively, and in accordance with Ridge, LASSO shrinks regression parameters compared to the OLS estimator. Additionally, it performs *feature selection* by setting small regression parameters *exactly* to zero — a property not found for Ridge regression.

Unfortunately, apart from oversimplified scenarios such as orthogonal design matrices, see e.g. [3], no generally applicable results for the moments of $\hat{\boldsymbol{\beta}}_{\text{LASSO}}$, in particular its expectation and variance, are known.

B. Classification

In contrast to regression, classification problems are concerned with finding a function $\hat{\psi}$ that maps a set of features $\mathcal{D}$ to a set of classes usually identified with integers $\mathcal{C} = \{0, 1, 2, ...\}$.

Here, we will explore *binary* classification on the Wisconsin breast cancer data set, [2]. It contains $P = 30$ numerically quantifiable features extracted from images of breast mass. Among these features are the radius, texture, symmetry, fractal dimension etc.

The goal of the classification task is to map the features to determine whether a tumor is benign or malignant, or more formally, we attempt to approximate the functional relation:

$$\psi : \mathcal{D} \subseteq \mathbb{R}^P \to \{0, 1\} \simeq \{\text{benign}, \text{malignant}\}\,. \tag{16}$$

1. Logistic Regression and Gradient Descent

Logistic regression tries to solve the binary classification problem by applying a monotonically increasing function, bounded between 0 and 1 (or any two integers that the

two classes are assigned to), to a linear combination of the features. One popular choice of such a function is the *sigmoid function* $s(x)$.

Let Θ_H denote the Heaviside step function. Our model then takes the form:

$$\begin{aligned}\hat{\psi}(\boldsymbol{x}) &= \Theta_H\left(s(\beta_0 + \boldsymbol{\beta}^\intercal \boldsymbol{x}) - \frac{1}{2}\right) \\ &= \Theta_H\left(\frac{1}{1+\exp\left(-\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x}\right)} - \frac{1}{2}\right) ,\end{aligned} \tag{17}$$

and we may interpret the value of $s(\boldsymbol{x}_i)$ as the probability of observing a benign tumor, y_i, under feature combination $\boldsymbol{x}_i$ *given* data set $\mathcal{D}$ and regression parameters $\boldsymbol{\beta}$:

$$\begin{aligned}p(y_i = 1|\mathcal{D}, \boldsymbol{\beta}) &= s(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x_i}), \\ p(y_i = 0|\mathcal{D}, \boldsymbol{\beta}) &= 1 - p(y_i = 1|\mathcal{D}, \boldsymbol{\beta})\end{aligned} \tag{18}$$

As for linear regression, section I A, the parameters $\boldsymbol{\beta}$ are found by extremizing (here maximizing) the probability for seeing the correct classifications $\boldsymbol{y} \in \{0,1\}^{N_\mathcal{T}}$ given the parameters $\boldsymbol{\beta}$:

$$\mathcal{L}\left(\boldsymbol{y}|\boldsymbol{\beta}\right) = \prod_{i=1}^{N_\mathcal{T}} p(y_i = 1|\mathcal{D}, \boldsymbol{\beta})^{y_i} \left[1 - p(y_i = 1|\mathcal{D}, \boldsymbol{\beta})\right]^{1-y_i} . \tag{19}$$

It is algebraically more convenient to minimize $-\log \mathcal{L}\left(\boldsymbol{y}|\boldsymbol{\beta}\right)$. This leads to the cost function of Logistic Regression known as the *cross-entropy*:

$$\begin{aligned}C(\beta_0, \boldsymbol{\beta}) = -\sum_{i=1}^{N_\mathcal{T}} \Big\{ & y_i \log(s(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x_i})) \\ & + (1 - y_i) \log(1 - s(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x_i})) \Big\} .\end{aligned} \tag{20}$$

We note in passing that the monotony of the log-transformation preserves the position of the likelihoods' extremal points.

Taking the derivative with respect to the intercept β_0 and the other parameters results in the equations:

$$\begin{aligned}\frac{\partial C}{\partial \beta_0} &= -\sum_{i=1}^{N_\mathcal{T}} (y_i - s(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x_i})) , \\ \nabla_{\boldsymbol{\beta}} C &= -\sum_{i=1}^{N_\mathcal{T}} (\boldsymbol{x_i} y_i - \boldsymbol{x_i} s(\beta_0 + \boldsymbol{\beta}^T \boldsymbol{x_i})) .\end{aligned} \tag{21}$$

By treating the classifications and the features as a vector and a design matrix respectively, these expressions can be condensed in vector notation. Let $p(y_i|\mathcal{D}, \boldsymbol{\beta}) = \boldsymbol{p}$, then the derivative becomes:

$$\frac{\partial C}{\partial \boldsymbol{\beta}} \equiv \begin{pmatrix} \partial_{\beta_0} C \\ \nabla_{\boldsymbol{\beta}} C \end{pmatrix} = -\begin{pmatrix} \mathbf{1}^\intercal \left(\boldsymbol{y} - \boldsymbol{p}\right) \\ \boldsymbol{X}^T \left(\boldsymbol{y} - \boldsymbol{p}\right) \end{pmatrix} . \tag{22}$$

Traditionally, a Ridge (or LASSO) penalty can be added to the cost function which leads to the final form of the non-intercept parameter gradient:

$$\nabla_{\boldsymbol{\beta}} C = -\boldsymbol{X}^T (\boldsymbol{y} - \boldsymbol{p}) + 2\lambda \boldsymbol{\beta} . \tag{23}$$

Due to the nonlinearity of eq. (23) in $\boldsymbol{\beta}$, no analytical solution to $\frac{\partial C}{\partial \boldsymbol{\beta}} = 0$ exists and one is therefore confined to numerical, iterative methods. We restrict our discussion to the application of the *gradient descent method* and it's stochastic extensions.

At its heart, lies the iterative update rule:

$$\hat{\boldsymbol{\beta}}^{j+1} = \hat{\boldsymbol{\beta}}^j - \gamma \frac{\partial C}{\partial \boldsymbol{\beta}} \left(\hat{\boldsymbol{\beta}}^j\right) = \hat{\boldsymbol{\beta}}^j - \gamma \sum_{i=1}^{N_\mathcal{T}} \frac{\partial C_i}{\partial \boldsymbol{\beta}}\bigg|_{\hat{\boldsymbol{\beta}}^j} , \tag{24}$$

with C_i denoting the cost contribution *per observation*, cf. eq. (21). Formally, one may regard eq. (24) as a simplified Newton-Raphson update in which the inverse Hessian matrix is traded for a constant, at this stage global, learning rate γ. More intuitively, the regression parameters at step $j+1$ are determined from the current parameters of step j by following the local gradient of the cost function landscape for a distance γ. Recall that $\frac{\partial C}{\partial \boldsymbol{\beta}}$ is the direction of steepest *ascent*.

Gradient descent (GD), i.e. eq. (24), is computationally tractable and guaranteed to converge to the global minimum for sufficiently small learning rates if the minimized cost function is convex — a property inherent to all cost functions used in this work.

That said, if one were to deal with a non-convex optimization problem, the deterministic nature of the algorithm allows it to get stuck in shallow, local minima. Moreover, adding up all per-observation-gradients C_i to form a *single* GD-update may be a costly operation if the data set is of enormous size.

A simple modification to the update rule of eq. (24) that introduces both stochasticity and mitigates the cost of the gradient computation is to base the parameter update on a *single, randomly chosen* observation $\{\boldsymbol{x}_i, y_i\}$, i.e.:

$$\hat{\boldsymbol{\beta}}^{j+1} = \hat{\boldsymbol{\beta}}^j - \gamma \frac{\partial C}{\partial \boldsymbol{\beta}}\bigg|^{\text{SGD}}_{\hat{\boldsymbol{\beta}}^j} = \hat{\boldsymbol{\beta}}^j - \gamma \frac{\partial C_i}{\partial \boldsymbol{\beta}}\bigg|_{\hat{\boldsymbol{\beta}}^j} . \tag{25}$$

This is known as *stochastic gradient descent* (SGD) and one typically sweeps through the entire data set multiple times until (i) a predefined number of full sweeps, or epochs, has passed or (ii) a per-epoch evaluation of the *total* cost function statisfies some tolerance argument. We refer to section I D for more details.

For practical reasons, see section I D, implementing a performant SGD routine can be tricky, while a naive GD implementation performs quite well for problem sizes we encounter in this work. On the other hand, stochasticity may be able to improve the convergence rate of the numerical optimization.

A compromise between the two extremes is *mini-batch gradient descent* (MGD), in which the data set is split

into B (ideally) equally sized, random mini-batches B_b with $\sum_{b=1}^{B}|B_b| = N_{\mathcal{T}}$. A single update step is then based on one mini-batch via:

$$\hat{\boldsymbol{\beta}}^{j+1} = \hat{\boldsymbol{\beta}}^{j} - \gamma \frac{\partial C}{\partial \boldsymbol{\beta}}\bigg|_{\hat{\boldsymbol{\beta}}^j}^{\text{MGD}} = \hat{\boldsymbol{\beta}}^{j} - \gamma \sum_{i\in B_b} \frac{\partial C_i}{\partial \boldsymbol{\beta}}\bigg|_{\hat{\boldsymbol{\beta}}^j} , \quad (26)$$

and the epoch ends after all mini-batches contributed one update step.

In what follows, we adopt the established, somewhat sloppy, terminology and allow ourselves to use the terms stochastic and mini-batch gradient descent interchangeably. That said, it will always be clear what the value of B is and thus whether reference to true SGD ($B = N_{\mathcal{T}}$), MGD ($1 < B < N_{\mathcal{T}}$) or randomized GD ($B = 1$) is made.

We close by mentioning that, aside from a plethora of other hyperparameters, the value and potential evolution of the learning rate γ can have profound implications on the effectiveness of any gradient decent method.

Besides a global and constant value of γ, power-law decays — or "inverse scaling" in `sklearn` parlance — of the form:

$$\gamma(j) = \frac{\gamma_0}{j^{\alpha}}, \quad \alpha > 0 , \quad (27)$$

are heavily employed in practice, with the rational being that early on, one is far away from the global minimum and thus requires large steps to enter the vicinity of the extremal point around which smaller step sizes are more appropriate.

More elaborate schemes, like AdaGrad, [7], improve the descent path by exploiting not global, but *feature-specific* learning rates $\boldsymbol{\gamma}(j) \in \mathbb{R}^P$. For AdaGrad, the learning schedule takes the following form:

$$\boldsymbol{\gamma}(j) = \gamma_0 \left(\epsilon \mathbf{1} + \sum_{j'=0}^{j} \left(\frac{\partial C}{\partial \boldsymbol{\beta}} \right)^2 \bigg|_{\hat{\boldsymbol{\beta}}^{j'}} \right)^{-\frac{1}{2}} , \quad \epsilon = 10^{-8} , \quad (28)$$

i.e. we keep track of the sum of the component-squared gradients encountered during the descent. This is quite intuitive: a large gradient is associate with a steep ravine in the direction of the measured gradient and setting the learning rate small in that direction avoids overshooting and potential oscillations.

A modification of AdaGrad also considered in this work is RMSProp [8]:

$$\boldsymbol{\gamma}(j) = \gamma_0 \left(\epsilon \mathbf{1} + \frac{9}{10} \left(\frac{\partial C}{\partial \boldsymbol{\beta}} \right)^2 \bigg|_{\hat{\boldsymbol{\beta}}^{j-1}} + \frac{1}{10} \left(\frac{\partial C}{\partial \boldsymbol{\beta}} \right)^2 \bigg|_{\hat{\boldsymbol{\beta}}^{j}} \right)^{-\frac{1}{2}} , \quad (29)$$

which replaces the sum of squared gradients by a moving, weighted average of the latter.

We refer the reader to section VI B for an in-depth study on the behavior of (S/M)GD as a function of learning rate, mini-batch number, allowed number of epochs and learning rate schedule.

2. Support Vector Machines

Another popular algorithm for classification is the Support Vector Machine (SVM) which tries to identify a separating hyperplane that maximizes the distance between itself and the samples being classified. The expression for a hyperplane is:

$$\sum \beta_n x_n + \beta_0 = 0 . \quad (30)$$

Then, in the case of binary classification $y \in \{-1, 1\}$, the samples can be classified by looking whether they are over or under the hyperplane. In other words:

$$\boldsymbol{\beta}^T \boldsymbol{x}_i + \beta_0 \begin{cases} > 0 \text{ if } & y_i = 1 \\ < 0 \text{ if } & y_i = -1 \end{cases} . \quad (31)$$

Because the sign of the left hand side coincides with the classification label y_i, a natural expression for the cost function can be defined:

$$C(\boldsymbol{\beta}) = -\sum_i y_i(\boldsymbol{\beta}^T \boldsymbol{x}_i + \beta_0) . \quad (32)$$

The cost will always be high when there are missclassified samples, or in other words when the product of the left hand side and the classification labels have different signs.

It is also possible to scale the parameters to be inversely proportional to a margin variable which defines the separation between classes $||\boldsymbol{\beta}||_2 = \frac{1}{M}$. Then, to find the largest margin M that separates the two classes, one has to minimize the norm of the parameters while keeping the classifications correct. This is equivalent to doing optimization with *Lagrangian multipliers*.

Following this procedure, the Lagrangian can be written as:

$$\mathcal{L}(\boldsymbol{\beta}, \beta_0, \lambda) = \frac{1}{2}||\boldsymbol{\beta}||_2 - \sum_i \lambda_i [y_i(\boldsymbol{\beta}^T \boldsymbol{x}_i + \beta_0) - 1] , \quad (33)$$

and model parameters can be derived by taking the derivatives:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} &= \boldsymbol{\beta} - \sum_i \lambda_i y_i \boldsymbol{x}_i = 0, \\ \frac{\partial \mathcal{L}}{\partial \beta_0} &= -\sum_i \lambda_i y_i = 0 , \end{aligned} \quad (34)$$

and the parameters are just the solution to $\boldsymbol{\beta} = \sum_i \lambda_i y_i \boldsymbol{x}_i$.

However, there is still the problem of finding the Lagrange multipliers λ. This can be done by rewriting and then optimizing the Lagrangian with the equalities derived from the partial derivatives, yielding:

$$\mathcal{L} = -\frac{1}{2} \sum_{ij} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i \boldsymbol{x}_j + \sum_i \lambda_i . \quad (35)$$

In practice data is often not linearly separable, which has motivated the use of Kernel methods. In these models,

a suitable transformation of the data is chosen that lifts it to a higher dimensional space (usually) where things are easily separable. The choices for such transformations can be many, however one of the most popular choices is the radial basis function kernel also used as the default option in the `sklearn` SVM classifier:

$$K(x_i, x_j) = \exp(-\gamma ||\boldsymbol{x}_i - \boldsymbol{x}_j||) \ . \tag{36}$$

C. Model Selection & Assessment

A natural, yet unanswered question is: Given a data set $\mathcal{D}$, a class of models, say polynomials up to degree p, and a hyperparameter-dependent learning method, say Ridge Regression, how do we pick the (hyper)parameters that yield the "best" model? One may phrase this question as the *model selection* problem.

Moreover, having picked an, in some sense, ideal model, how do we benchmark its performance? We coin this problem *model assessment*.

Note that both of these questions have to be answered within the scope of the available data $\mathcal{D}$ which, depending on its scarcity, can be no mean feat. In what follows, we assume ideal, data-rich conditions so that the selection and assessment step can be well separated. It is worth noting that such conditions are likely unrealistic for a given data set $\mathcal{D}$ of size N, but we settle for conceptual clarity at this point, as should become clear shortly.

In a first attempt to make both selection and assessment well-defined, we randomly split the total data set $\mathcal{D}$ into a training set $\mathcal{T}$, validation set $\mathcal{V}$ and test set $\mathcal{S}$ such that $\mathcal{D} = \mathcal{T} \cup \mathcal{V} \cup \mathcal{S}$. Model selection will only operate on $\mathcal{T}$ and $\mathcal{V}$, while the final assessment step gets access to the entire data set $\mathcal{D}$ in form of $\mathcal{T} \cup \mathcal{V}$ and $\mathcal{S}$. If data is scarse and model training data-greedy, one usually sets $\mathcal{V} = \mathcal{S}$.

Let us start with model selection. A trained model $\hat{f}$, specified by its set of (hyper)parameters $\mathcal{P}$, is considered optimal if it realizes a minimal prediction error on unseen data points $(\boldsymbol{X}_\mathcal{V}, Y) \in \mathcal{V}$. Formally, ideal parameters $\mathcal{P}^*$ are therefore found by minimizing the *expected prediction error* $\mathcal{E}$ under squared loss:

$$\begin{aligned}
\mathcal{P}^* &\equiv \arg\min_{\mathcal{P}} \mathcal{E} \\
&= \arg\min_{\mathcal{P}} \mathbb{E}_{(\boldsymbol{X}_\mathcal{V}, Y), \mathcal{T}} \left[(Y - \hat{f}(\boldsymbol{X}_\mathcal{V}; \mathcal{P}))^2 \right] \\
&= \arg\min_{\mathcal{P}} \mathbb{E}_{\boldsymbol{X}_\mathcal{V}} \left[\mathbb{E}_{Y, \mathcal{T}} \left[(Y - \hat{f}(\boldsymbol{x}_\mathcal{V}; \mathcal{P}))^2 \mid \boldsymbol{X}_\mathcal{V} = \boldsymbol{x}_\mathcal{V} \right] \right] \\
&= \arg\min_{\mathcal{P}} \mathbb{E}_{\boldsymbol{X}_\mathcal{V}} \left[\mathcal{E}(\boldsymbol{x}_\mathcal{V}) \right] \ ,
\end{aligned} \tag{37}$$

where expected value is taken over the factorizable, probability distribution of unseen validation points $(\boldsymbol{X}_\mathcal{V}, Y)$ and the training set $\mathcal{T}$. Recall that the stochasticity in the latter (and hence in the predictor $\hat{f}$) is only due to noise in the dependent variable, not the regressors (the feature measurements), which are considered fixed for the training procedure, see section I A. The validation point $(\boldsymbol{X}_\mathcal{V}, Y)$, on the other hand, is considered a random variable in both components, i.e. in Y due to its intrinsic measurement noise ϵ and the feature vector $\boldsymbol{X}_\mathcal{V}$ since $\mathcal{V}$ is a random sample of the underlying data population.

Phrasing the solution of model selection as the minimization of an objective function also clarifies, why we opted for a three-fold data split rather than a train-test split: Solving eq. (37) means sweeping through the (hyper)parameter plane, retraining our model for each parameter combination and *estimating* $\mathcal{E}$, see section I C 2. If we were to use the test set $\mathcal{S}$ for estimating $\mathcal{E}$, information about its characteristics would leak back into the training process, which would bias the model assessment step. *Test data should never, in any way, participate in the training process.* Hence the evaluation of $\mathcal{E}$ on a dedicated (if $\mathcal{D}$ allows for it) validation set.

Compared to model selection, the final assessment step is rather simple:

Firstly, we retrain the ideal model $\hat{f}(\boldsymbol{x}; \mathcal{P}^*)$ with $\mathcal{T}^* = \mathcal{T} \cup \mathcal{V}$ with the rational being that both data sets were already exposed to the training either directly or indirectly.

Secondly, we assess the final trained model on the still untouched test set $\mathcal{S}$ by means of some suitable metric. A common choice is the test mean squared error:

$$\mathrm{MSE}_\mathcal{S} = \frac{1}{N_\mathcal{S}} \sum_{i=1}^{N_\mathcal{S}} \left(y_i - \hat{f}(\boldsymbol{x}_i; \mathcal{P}^*) \right)^2 \ , (\boldsymbol{x}_i, y_i) \in \mathcal{S} \ . \tag{38}$$

1. The Bias Variance Decomposition

Choosing a squared loss function in eq. (37) allows us to further decompose the expected prediction error into different error contributions. The derivation of this, so called, *bias-variance decomposition*, can be found in any textbook, e.g. [4], and is omitted for brevity. One finds:

$$\begin{aligned}
\mathcal{E} &= \mathbb{E}_{\boldsymbol{X}_\mathcal{V}} \left[\mathbb{E}_{Y, \mathcal{T}} \left[(Y - \hat{f}(\boldsymbol{x}_\mathcal{V}; \mathcal{P}))^2 \mid \boldsymbol{X}_\mathcal{V} = \boldsymbol{x}_\mathcal{V} \right] \right] \\
&= \sigma^2 + \mathbb{E}_{\boldsymbol{X}_\mathcal{V}} \left[(f - \mathbb{E}_\mathcal{T}[\hat{f}])^2 \right] + \mathbb{E}_{\boldsymbol{X}_\mathcal{V}} \left[\mathbb{E}_\mathcal{T} \left[(\hat{f} - \mathbb{E}_\mathcal{T}[\hat{f}])^2 \right] \right] \\
&= \sigma^2 + \mathbb{E}_{\boldsymbol{X}_\mathcal{V}} \left[\mathrm{Bias}_\mathcal{T}[\hat{f}](\boldsymbol{x}_\mathcal{V}) \right] + \mathbb{E}_{\boldsymbol{X}_\mathcal{V}} \left[\mathrm{Var}_\mathcal{T}[\hat{f}](\boldsymbol{x}_\mathcal{V}) \right] \ ,
\end{aligned} \tag{39}$$

where σ^2 denotes the *irreducible error* intrinsic to the validation response Y. Note that it is independent of our model/predictor $\hat{f}$ and thus constitutes a lower bound of the expected prediction error that even a perfect $\hat{f} = f$ is not able to circumvent.

Above this threshold, the value of $\mathcal{E}$ is set by the bias, $\mathbb{E}_{\boldsymbol{X}_\mathcal{V}} \left[\mathrm{Bias}_\mathcal{T}[\hat{f}](\boldsymbol{x}_\mathcal{V}) \right]$, and the predictor's variance, $\mathbb{E}_{\boldsymbol{X}_\mathcal{V}} \left[\mathrm{Var}_\mathcal{T}[\hat{f}](\boldsymbol{x}_\mathcal{V}) \right]$. Their respective meaning may be summarized as follows:

Bias — the deviation between true value and the expected predicted value — is large when, even after averaging over multiple training sets, the assumed model is too inflexible to reproduce the correct dependent variable $y = f(\boldsymbol{x}_\mathcal{V})$. Situations in which $\mathcal{E}$ is bias dominated is typically coined *underfitting* and it is a clear sign of a too simplistic model that is incapable of reproducing the fine-grained structures produced by the true function f.

Variance — the mean square deviation between a specific predictor realization $\hat{f}$ and its statistical average — tends to be large if $\hat{f}$ is highly adaptable and hence susceptible to the specific noise structure of the training set $\mathcal{T}$ it got exposed to. The result is vastly different predictions across different predictor realizations. Situations in which $\mathcal{E}$ is variance dominated is commonly referred to as *overfitting* and indicates a too complex model.

Understanding how OLSes, Ridge's and LASSOs (hyper)parameters influence these error contribution, and to what extend bias and variance can be traded for each other forms a major part of our analysis in section VI A.

At this stage, we only provide the theoretical link between bias and variance of linear models and the respective moments of their regression parameter estimators in eq. (9), (10) or eq. (13), (14) respectively. A simple calculation yields:

$$\begin{aligned} \mathrm{Bias}_\mathcal{T}[\hat{f}](\boldsymbol{x}_\mathcal{V}) &= \mathrm{Bias}[\hat{\boldsymbol{\beta}}]^\intercal \boldsymbol{x}_\mathcal{V} \ , \\ \mathrm{Var}_\mathcal{T}[\hat{f}](\boldsymbol{x}_\mathcal{V}) &= \boldsymbol{x}_\mathcal{V}^\intercal \mathrm{Cov}[\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\beta}}] \boldsymbol{x}_\mathcal{V} \ . \end{aligned} \tag{40}$$

These relations will proof useful in section VI A.

2. *Estimators for the Expected Prediction Error: Resampling Techniques*

As for any population statistic, reliable sample estimators are required to approximate the expected prediction error $\mathcal{E}$. This section is meant to provide procedures, so called resampling methods, to construct such.

One might be tempted to simply evaluate the validation mean squared error, $\mathrm{MSE}_\mathcal{V}$, i.e. eq. (38) with $\mathcal{S} \to \mathcal{V}$, as a function of the model's parameters to estimate the expected prediction error $\mathcal{E}$. In fact, one could even consider to abolish $\mathcal{V}$ all together, save some data and evaluate eq. (38) solely on the training set $\mathcal{T}$. Fig. 2 illustrates why this is insufficient. Here, we show both $\mathrm{MSE}_\mathcal{V}$ and $\mathrm{MSE}_\mathcal{T}$ of an OLS-trained predictor as a function of the polynomial degree p.

While $\mathrm{MSE}_\mathcal{T}$ suggests a monotonically increasing prediction accuracy, the validation data based $\mathrm{MSE}_\mathcal{V}$ reveals a strong degradation of the model performance for increasing complexity. This is the effect of overfitting: the adaptation of highly flexible models to the particular (noise) structure of the training data. Clearly, $\mathrm{MSE}_\mathcal{T}$ is insensitive to the overfitting scenario and is therefore a low-quality, biased estimator for the expected prediction error in eq. (39). Additionally, note how the training error takes on values with $\mathrm{MSE}_\mathcal{T} < \sigma^2$ (gray shaded area), thus contradicting the lower bound implied by eq. (39).

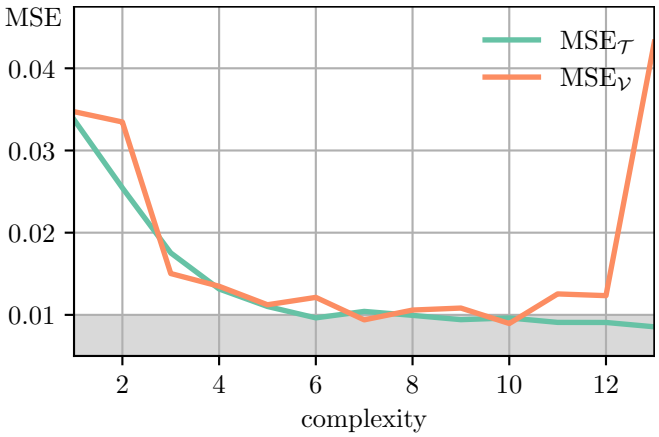

FIG. 2. Mean squared error (MSE), see eq. (38), of training data (green) and unseen validation data (orange) as a function of the model complexity, i.e. the maximal polynomial degree p. Fit was based of an data set with $N = 600$ points, randomly split into a training set ($N_\mathcal{T} = 450$) and validation set ($N_\mathcal{V} = 150$). Gaussian noise was drawn from $\mathcal{N}(0, \sigma = 0.1)$ and the gray shaded area depicts the theoretically inaccessible region $\mathrm{MSE} < \sigma^2$.

Although, $\mathrm{MSE}_\mathcal{V}$ captures the qualitative behavior of the error decomposition of eq. (39) correctly, it is, as an estimator, highly variable. Put differently, rerunning the experiment of Fig. 2, and thereby constructing a new *single* predictor, yields a different $\mathrm{MSE}_\mathcal{V}$ that is qualitatively similar but quantitatively different. Most notably, its minimum, the point that defines the optimal parameter set $\mathcal{P}^*$, is non-stationary. Consequently, $\mathrm{MSE}_\mathcal{V}$ is not a robust estimator for $\mathcal{E}$ either.

The reason for $\mathrm{MSE}_\mathcal{V}$'s failure to be reliable estimator is that it is constructed from a *single* realization of $\hat{f}$. In other words, it does not estimate $\mathbb{E}_\mathcal{T}$ appropriately. *A sound estimator $\hat{\mathcal{E}}$ to $\mathcal{E}$ must average over a set of predictor realizations.*

Resampling techniques such as *bootstrapping* and *k-fold cross-validation* allow to construct such a set of predictor realizations if only one training set $\mathcal{T}$ is available.

a. ***Bootstrap-based Estimator:*** Bootstrapping exploits a *draw-and-replace* strategy to construct a set of training sets $\{\mathcal{T}_b\}_{b=1\ldots B}$ from the original $\mathcal{T}$:

$$\mathcal{T}_b = \{(\boldsymbol{x}_\mathcal{T}, y)_i \in \mathcal{T} \mid i \in I_b \sim \mathcal{U}(1, N_\mathcal{T})\} \tag{41}$$

with I_b denoting the index set of bootstrap sample b randomly drawn from the uniform discrete distribution $U(1, N_\mathcal{T})$. We emphasize $|I_b| = N_\mathcal{T}$ and I_b is allowed to hold duplicate indices by construction (thus slightly abusing the standard set notation).

Each $\mathcal{T}_b$ then yields a predictor $\hat{f}_b$ and an estimate for the prediction error $\mathcal{E}$ follows from:

$$\hat{\mathcal{E}} = \frac{1}{N_\mathcal{V} B} \sum_{b=1}^{B} \sum_{i=1}^{N_\mathcal{V}} \left(y_i - \hat{f}_b(\boldsymbol{x}_i) \right)^2 , \quad (\boldsymbol{x}_i, y_i) \in \mathcal{V} \tag{42}$$

The pythionic-pseudo code below illustrates the computation of the bootstrap-based prediction error estimator $\hat{\mathcal{E}}$ in more detail.

```
def pred_error_bs(loss, model, X, y, B):

    X_t, X_v, y_t, y_v = \
                train_test_split(X, y)

    N_V = X_val.shape[0]
    N_T = X_tr.shape[0]

    y_pred = np.empty((N_V, B))

    rng = np.random.default_rng()
    bs_indices = \
        rng.integers(N_T, size=(B, N_T))

    for b in range(B):
        X_t_b = X_t[bs_indices[b, :]]
        y_t_b = y_t[bs_indices[b, :]]

        model.fit(X_t_b,y_t_b)
        y_pred[:, b] = \
            model.predict(X_val)

    prediction_error = np.mean(
        np.mean(
            loss(y_v, y_pred),
            axis=-1
        )
    )

    return prediction_error
```

Estimators for the remaining two error contributions follow by a trivial extension of the provided code snippet.

b. ***Cross-Validation-based Estimator:*** Cross-validation is a resampling technique particular useful if the scarcity of the total data set does not allow for a three-fold train-validate-test split as it is able to produce a sequence of train-validate sets $(\mathcal{T}_k, \mathcal{V}_k)_{k=1\dots K}$ from $\mathcal{T}$. A twofold train-test split of $\mathcal{D}$ is therefore sufficient.

To construct this set sequence, $\mathcal{T}$ is split into K (ideally) equally sized folds

$$\mathcal{F}_k = \left\{ (\boldsymbol{x}_\mathcal{T}, y)_{I(i)} \mid \forall i \leq N_\mathcal{T} : I(i) = k \right\} , \quad k = 1 \dots N_\mathcal{T} \tag{43}$$

We emphasize the use of the fold-distributing index *function* $I : \{1 \dots N_\mathcal{T}\} \to \{1 \dots K\}$ is deliberate as it prevents multiple occurrences of the same training point $(\boldsymbol{x}_\mathcal{T}, y)$ in (i) the same folds and (ii) across different folds. Put differently, *cross-validation operates without replacement.*

The training set $\mathcal{T}_k$ and validation set $\mathcal{V}_k$ then follow from:

$$\mathcal{T}_k = \bigcup_{i \in \{1\dots K\} \setminus \{k\}} \mathcal{F}_i , \quad \mathcal{V}_k = \mathcal{F}_k , \tag{44}$$

and if $\hat{f}_k$ denotes the estimator originating from $\mathcal{T}_k$, we arrive at the estimator:

$$\hat{\mathcal{E}} = \frac{1}{N_\mathcal{V} K} \sum_{k=1}^{K} \sum_{i=1}^{N_\mathcal{V}} \left(y_i - \hat{f}_k(\boldsymbol{x}_i) \right)^2 , \quad (\boldsymbol{x}_i, y_i) \in \mathcal{V}_k \tag{45}$$

Consider the pseudo-code implementation of `def_error_CV` for further details.

```
def pred_error_CV(loss, model, X, y, k):

    def split(X):
        idx = np.arange(X.shape[0])
        folds = np.array_split(idx, k)
        for i in range(k):
            yield ( np.hstack(folds[1:]),
                    folds[0] )
            folds.append(folds.pop(0))

    loss_per_fold = np.empty(k)

    i=0
    for (t_idx, v_idx) in  split(X):
        X_t, y_t = X[t_idx], y[t_idx]
        X_v y_v = X[v_idx], y[v_idx]

        model.fit(X_t, y_t)
        y_pred = model.predict(X_v)
        loss_per_fold[i] = np.mean(
            loss(y_v, y_pred)
        )
        i += 1

    return np.mean(loss_per_fold)
```

D. Implementational Details

The code developed for this work can be found at the following -repo. Conceptually, it is close to `sklearn`'s design philosophy. It's usage should therefore be mostly self-explanatory. Here, we provide some additional details and rationals that went into its development.

Consider Fig. 3 for the dependence (`import`) graph of all participating `py`-files and Fig. 4 for an UML diagram of the implemented class hierarchies and their public API. The color coding of each class designates its respective implementation file.

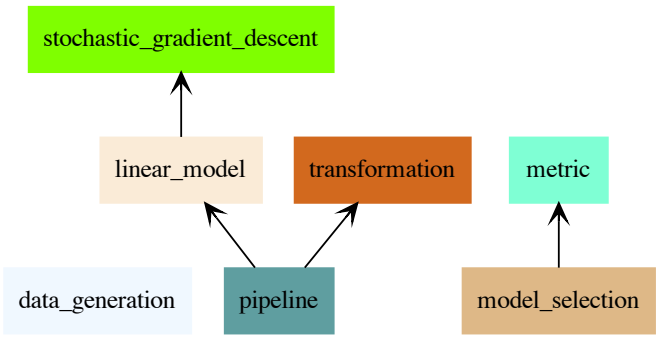

FIG. 3. Visualization of the dependence graph of all implementation files.

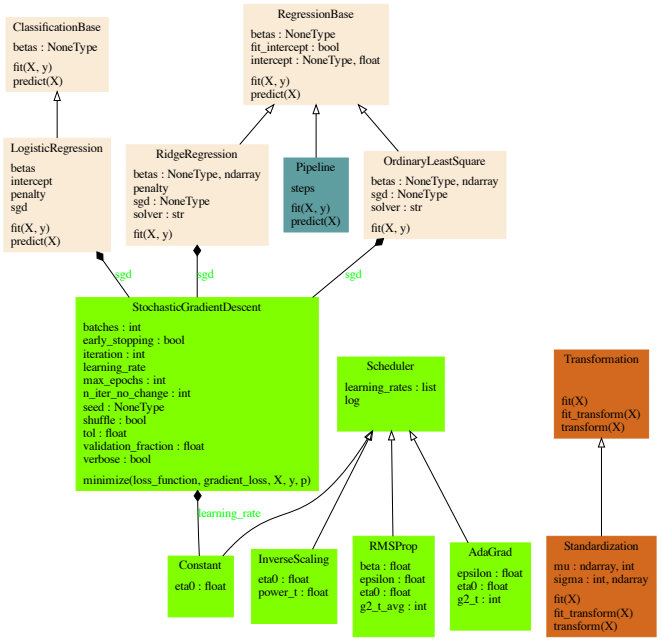

FIG. 4. UML diagram of the class hierarchies implemented for this work. Colors designate their place of definition, cf. Fig. 3.

II. THE LINEAR REGRESSION HIERARCHY

OLS (`OrdinaryLeastSquare`) and Ridge regression (`RidgeRegression`) are implemented as specializations of the `RegressionBase`-class. Their sole difference lies in the implementation of `fit(X,y)`, which either uses eq. (7) or eq. (12) as estimation of the regression parameters or forwards the computation to `StochasticGradientDescent` in case the user sets `solver="sgd"`.

Besides implementing a common interface, `RegressionBase`'s main purpose lies in (i) providing a common implementation of the (trivial) `predict(X,y)` function — a matrix-vector multiplication — and (ii) taking care of the necessary data pre-processing before eq. (7)/(12) or `StochasticGradientDescent` can be applied.

Notice how the intercept term is treated independently from all other, potentially penalized, regression parameters. This independent treatment, as described in eq. (7) and (12), is only sound if the design matrix $\boldsymbol{X}$ *and* the response vector $\boldsymbol{z}$ were zero-centered, i.e. each column of both objects was shifted by the respective column mean $\bar{\boldsymbol{X}}/\bar{z}$ — a detail we omitted in the discussion of section I A. Thus, $\mathcal{T} \simeq \{\boldsymbol{X}, \boldsymbol{z}\}$ needs to be transformed to $\bar{\mathcal{T}} \simeq \{\boldsymbol{X} - \mathbb{1}\bar{\boldsymbol{X}}, \boldsymbol{z} - \bar{z}\mathbf{1}\}$. `RegressionBase` takes care of this transformation, as well as the final intercept computation after the cost function for all other parameters has been minimized.

III. HIERARCHY

`LogisticRegression` derives from `ClassificationBase` which, compared to `RegressionBase` is trivial and only of sentinel-type. Hence, all classification procedures, e.g. (17), are implemented in `LogisticRegression`. Note that the nonlinearity of eq. (23) in $\boldsymbol{\beta}$ makes it impossible to separate the intercept computation from the rest of the regression parameters. Therefore, the intercept needs to be determined via the descent procedure and it is for this reason that the gradient function, `_gradient_loss(X,y,p)` handed to `StochasticGradientDescent.minimize()` differs in its zero-component (compared to all other components).

IV. `StochasticGradientDescent`

In essence, `StochasticGradientDescent` is a glorified version of eq. (26) parametrized on the number of mini-batches B, the cost function gradient $\frac{\partial C}{\partial \boldsymbol{\beta}}$, the cost function C and the learning rate schedule with $\boldsymbol{\beta}^0 = \mathbf{0}$ as initial conditions.

It is possible to phrase Ridge Regression as a constrained minimization problem: Instead of minimizing eq. (11), one can equivalently minimize the OLS problem, i.e. eq. (5) with cost (6), under the constraint $||\boldsymbol{\beta}||_2 \leq L$ for some $L(\lambda)$. Clearly $\boldsymbol{\beta} = \mathbf{0}$ satisfies this inequality condition for all λ and is therefore a reasonable educated guess that complies with all possible regularizer strengths.

Access to the cost function is required if early convergence checks should be performed. In this case `StochasticGradientDescent` computes the *total* data set cost after each epoch, C_E, and checks if:

$$C_E > \min\{C_e \mid e \leq E\} - \Delta \,, \tag{46}$$

for a user-provided tolerance Δ. The descent is assumed to be completed once eq. (46) was satisfied $M \simeq 5$ consecutive times.

Learning rate schedules are implemented as specializations of `Scheduler` providing simplistic logging functionality. We provide specialized implementations for power-law scaling (`InverseScaling`), `AdaGrad` and `RMSProp`.

Let us close by mentioning that `StochasticGradientDescent` with $B = 1$ is about $30 - 40$ times slower than `SGDRegressor`, `sklearn`'s version of stochastic gradient descent. After spending significant time with profiling our implementation, we believe that the function call overhead of the pythionic-version for $\frac{\partial C}{\partial \boldsymbol{\beta}}$ is responsible for this behavior. It is no coincidence that `SGDRegressor` chose to implement this part in `cython`. The authors opted for parallelizing the parameter sweep loops in `05_stochastic_gradient_descent.ipynb` to combat the issue (rather than improving the SGD implementation).

V. FEATURE SCALING & PIPELINES

Feature scaling, i.e. centering all columns of the design matrix $\boldsymbol{X}$ and normalizing to the column standard deviation, is a common pre-processing step. The performance of stochastic gradient descent, for instance, is highly dependent on the magnitude and variation of the feature measurements.

`Standardization` implements this functionality. Note that feature scaling can already be considered a part of the model training, and as such should only infer data, e.g. feature mean or feature standard deviation, from the training data.

To make this conceptional constraint manifest, we re-implement a layman's version of `sklearn`'s pipeline mechanism: Handed a list of `Transformations` that ends with a `RegressionBase` or `ClassificationBase` instance, our `Pipeline` class guarantees that (i) all sample statistics are inferred exclusively from the training data, (ii) data transformation are applied before the call to `fit` and (iii) predicting new values from the validation/test set uses the sample statistics from the training set only.

VI. NUMERICAL STUDY

A. Regression on the Franke Function

Following the model selection approach outlined in section I C, the main purpose of this section is to *estimate* the expected prediction error of the squared error loss function $\hat{\mathcal{E}}$ — loosely speaking the *mean squared error* (MSE) — by means of independent bootstrap and cross-validation experiments.

Doing so will allow us to (i) explore potential discrepancies between both resampling strategies, (ii) understand how data-, model-, and hyperparameters influence the MSE as well as its error decomposition, and most importantly (iii) select the best polynomial model for Franke's function given one of the regression methods of section I A 1 - I A 3.

If not specified otherwise, conditions of section I A apply, meaning all analysis steps are based of a random realization with $N = N_{\mathcal{T}} + N_{\mathcal{V}}$ observations $z_i = f(\boldsymbol{x}_i) + \epsilon_i$ of Franke's function at random but fixed domain sights $\boldsymbol{x}_i$ and $\epsilon_i \sim \mathcal{N}(0, \sigma = 0.1)$. $N_{\mathcal{T}}$ denotes the cardinality of the training set $\mathcal{T}$, whereas $N_{\mathcal{V}}$ represents the size of the validation set $\mathcal{V}$.

Apart from section VI A 1, in which we observe the dependence of the MSE $\hat{\mathcal{E}}$ as a function of N, our standard set sizes are chosen as $N = 600$, $N_{\mathcal{T}} = 0.75N = 450$ and $N_{\mathcal{V}} = 0.25N = 150$.

The design matrix $\boldsymbol{X} \in \mathbb{R}^{N_{\mathcal{T}} \times (P-1)}$ that enters the optimization procedure in eq. (5) will be standardized by default, such that on top of the centering step carried out by `RegressionBase`, see section I D, a unit variance for all features is enforced.

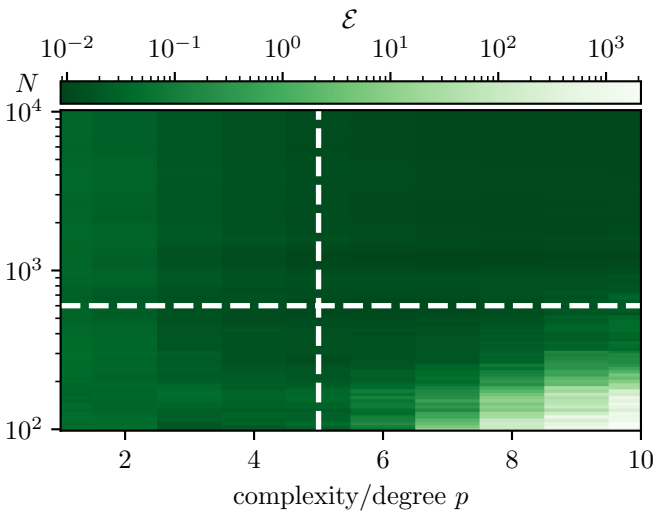

FIG. 5. MSE $\hat{\mathcal{E}}$ as a function of polynomial degree p and data set size N acquired through a bootstrap analysis with $B = 50$ re-drawn training sets of size $N_{\mathcal{T}} = 0.75N$ and evaluated on the fixed validation set $\mathcal{V}$ of size $N_{\mathcal{V}} = 0.25N$. The white dashed lines designate the cross sections illustrated in Fig. 6 and Fig. 7. Note how increasing the data set size mitigates overfitting at high polynomial order but leaves the behavior of $\hat{\mathcal{E}}$ unaffected at low p.

At last, we remark that, if not explicitly stated otherwise, all reported quantities are averaged over $\mathcal{V}$ in order to approximate the outer expectation value in eq. (39). To simplify notation, this average will be absorbed into the standard "estimator head", so that e.g. $\widehat{\mathrm{Var}}[\hat{f}]$ estimates $\mathbb{E}_{\boldsymbol{X}_{\mathcal{V}}}\left[\mathrm{Var}_{\mathcal{T}}[\hat{f}](\boldsymbol{x}_{\mathcal{V}})\right]$.

1. *Ordinary Least Squares*

We begin our discussion with an in-depth analysis for the OLS predictor in eq. (7) and a bootstrap-computation of the MSE $\hat{\mathcal{E}}$ with $B = 50$ bootstrap samples and a train-validate split of $N_{\mathcal{T}} = 0.75N$ and $N_{\mathcal{V}} = 0.25N$. Recall that the bootstrap variant employed here only operates on $\mathcal{T}$ for drawing resampled training data and always evaluates the squared error on $\mathcal{V}$.

OLS is free of any hyperparameters. Thus, given a data set of size N, one is left with the polynomial degree p as the only degree of freedom. Fortunately, since the data generative process, i.e. one draw of a noisy observation of the *known* eq. (1), is completely under our control, we are in the position to explore the implications of a changing data set size N in addition to a varying model complexity. A parameter sweep through the $N - p$-plane results in Figure 5.

Two important observations can be made:

Firstly, for sufficiently low N, say $N < 1000$, $\hat{\mathcal{E}}$ reproduces the behavior of the mean square test error $\mathrm{MSE}_{\mathcal{T}}$ of Fig. 2: For simple, low-degree polynomials $\hat{\mathcal{E}}$ is large due

to underfitting — the employed model is too inflexible to adapt to the fine-structure that Franke's function, not the intrinsic noise, requires. For complex, high-degree models $\hat{\mathcal{E}}$ is again large, due to overfitting — too many degrees of freedom allow the model to adapt to the noise of the training data which will be different when applied to the validation set, hence resulting in a large error even when averaged over multiple bootstrap training sets.

Secondly, increasing N reduces the magnitude of $\hat{\mathcal{E}}$ for high polynomial orders. Put differently, *adding data points, counteracts overfitting.* Note that no significant error reduction is observed at low p, implying *underfitting cannot be reduced by exposing the learning algorithm to more data.*

A theoretical understanding for both observations can be acquired if one is able to associate different elements of the mean squared error decomposition with either the overfitting or underfitting behavior of $\hat{\mathcal{E}}$ and subsequently explore their dependence on N.

In section I C 1, we explained overfitting as the result of a high variance, $\mathbb{E}_{\boldsymbol{X}_\mathcal{V}}\left[\mathrm{Var}_\mathcal{T}[\hat{f}](\boldsymbol{x}_\mathcal{V})\right]$, and underfitting originating from an overall bias, $\mathbb{E}_{\boldsymbol{X}_\mathcal{V}}\left[\mathrm{Bias}_\mathcal{T}[\hat{f}](\boldsymbol{x}_\mathcal{V})\right]$, of the predictor $\hat{f}$. Fig. 6 yields a numerical justification for this claim. Here, we illustrate a horizontal cross section through Fig. 5 at our default value $N = 600$ (white dashed line) for all error component estimators that contribute to $\hat{\mathcal{E}}$. In accordance with our expectation, one finds the increase of $\hat{\mathcal{E}}$ to be mainly driven by the estimated predictor variance $\widehat{\mathrm{Var}}[\hat{f}]$. The reported bias estimate $\widehat{\mathrm{Bias}}[\hat{f}]$, on the other hand, is significant in the underfitting regime at low p, but decays towards zero as the model complexity increases. We emphasize that (i) adding up all error component estimators yields $\hat{\mathcal{E}}$, thus recovering the theoretical decomposition in eq. (39), in an "estimator sense" and (ii) the MSE is bounded from below by the irreducible error σ^2 present in the validation set $\mathcal{V}$.

Figure 6 suggests that the observed overfitting mitigation for larger values of N originates from the predictor variance $\mathbb{E}_{\boldsymbol{X}_\mathcal{V}}\left[\mathrm{Var}_\mathcal{T}[\hat{f}](\boldsymbol{x}_\mathcal{V})\right]$. Indeed, a lengthy but simple calculation shows that for a fixed feature vector $\boldsymbol{x}_\mathcal{V} \in \mathbb{R}^P$:

$$\begin{aligned}\mathrm{Var}_\mathcal{T}[\hat{f}]\,(\boldsymbol{x}_\mathcal{V}) &= \boldsymbol{x}_\mathcal{V}^\intercal \mathrm{Cov}\left[\begin{pmatrix}\hat{\beta}_0\\ \hat{\boldsymbol{\beta}}\end{pmatrix}\right]\boldsymbol{x}_\mathcal{V}\\ &= \boldsymbol{x}_\mathcal{V}^\intercal \begin{pmatrix}\frac{\sigma^2}{N_\mathcal{T}} + \bar{\boldsymbol{X}}^\intercal \mathrm{Cov}[\hat{\boldsymbol{\beta}}]\bar{\boldsymbol{X}} & -\bar{\boldsymbol{X}}^\intercal \mathrm{Cov}[\hat{\boldsymbol{\beta}}]\\ -\mathrm{Cov}[\hat{\boldsymbol{\beta}}]^\intercal \bar{\boldsymbol{X}} & \mathrm{Cov}[\hat{\boldsymbol{\beta}}]\end{pmatrix}\boldsymbol{x}_\mathcal{V}\end{aligned} \tag{47}$$

for *linear* regression models. Setting $\hat{\boldsymbol{\beta}} = \hat{\boldsymbol{\beta}}_{\mathrm{OLS}}$ and

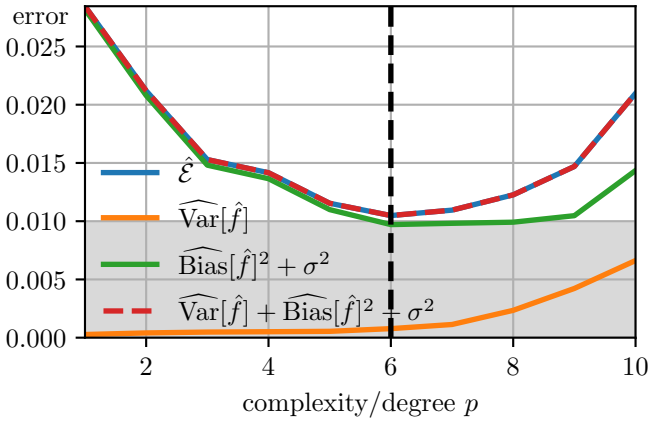

FIG. 6. Cross section through Fig. 5 at $N = 600$. Shown is the estimated bias-variance decomposition as a function of p. We find the MSE $\hat{\mathcal{E}}$ to recover the qualitative behavior of the test error in Fig. 2, i.e. large values of $\hat{\mathcal{E}}$ for low and high model complexities due to under- and overfitting respectively.

substituting eq. (10) yields:

$$\begin{aligned}&\mathrm{Var}_\mathcal{T}[\hat{f}]\,(\boldsymbol{x}_\mathcal{V}) = \sigma^2 \times\\ &\boldsymbol{x}_\mathcal{V}^\intercal \begin{pmatrix}\frac{1}{N_\mathcal{T}} + \bar{\boldsymbol{X}}^\intercal (\boldsymbol{X}^\intercal \boldsymbol{X})^{-1}\bar{\boldsymbol{X}} & -\bar{\boldsymbol{X}}^\intercal (\boldsymbol{X}^\intercal \boldsymbol{X})^{-1}\\ -(\boldsymbol{X}\boldsymbol{X}^\intercal)^{-1}\bar{\boldsymbol{X}} & (\boldsymbol{X}^\intercal \boldsymbol{X})^{-1}\end{pmatrix}\boldsymbol{x}_\mathcal{V}\,,\end{aligned} \tag{48}$$

where $\bar{\boldsymbol{X}}$ denotes the mean feature vector, $\mathrm{Cov}[\boldsymbol{v}] \equiv \mathrm{Cov}[\boldsymbol{v}, \boldsymbol{v}]$ and $\boldsymbol{X}^\intercal \boldsymbol{X}$ is expected to be non-singular — an assumption which holds true for our application.

Recall that $\boldsymbol{X}^\intercal \boldsymbol{X}$ estimates the sample covariance matrix of the measured feature vectors: $\boldsymbol{X}^\intercal \boldsymbol{X} \approx N_\mathcal{T} \widehat{\mathrm{Cov}}(\boldsymbol{X})$. Inserting into eq. (48) then yields:

$$\widehat{\mathrm{Var}}[\hat{f}] \approx \mathbb{E}_\mathcal{V}\left[\mathrm{Var}_\mathcal{T}[\hat{f}](\boldsymbol{x}_\mathcal{V})\right] \sim \frac{1}{N_\mathcal{T}} \propto \frac{1}{N} \quad (N \to \infty) \tag{49}$$

which is exactly the asymptotic behavior we find for the estimated predictor variance in Fig. 7 obtained by taking the cross section in Fig. 5 at fixed $p = 5$.

Back to our original objective in selecting the most robust polynomial given a data set of total size N. If we fix $N = 600$, the error cross section in Fig. 6 suggests $p = 6$ (black dashed line) to generalize best to unseen data. To validate this result, we conduct a k-fold cross-validation analysis for the same data set in Fig. 8 and compare it with results of Fig. 6 (reproduced in Fig 8 as solid black line). On closer inspection, it becomes apparent that both resampling methods yield results consistent with each other: both minimize the mean squared error $\hat{\mathcal{E}}$ at $p = 6$ and the bootstrap error is mostly consistent with the 1σ-confidence region of the $k = 5$ and $k = 9$ cross-validation error. We note in passing that the error region for $k = 9$ is slightly larger than for $k = 5$. This is to be expected, as higher values of k imply resampled training sets with a larger element overlap. The per-fold values $\hat{\mathcal{E}}_k$ should thus experience a stronger correlation which translates into a larger variance after averaging.

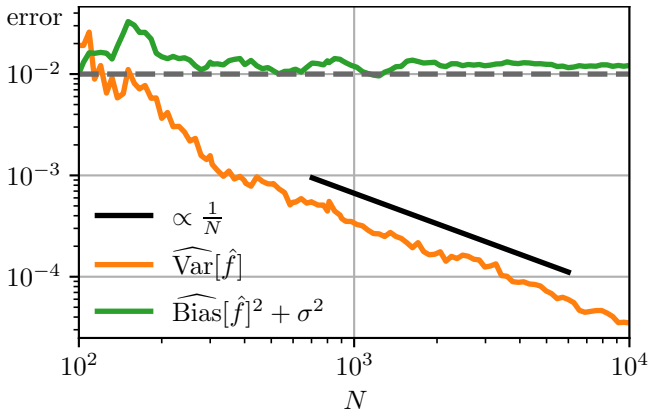

FIG. 7. Cross section through Fig. 5 at $p = 5$. Shown is the dependency of the estimated bias, $\widehat{\mathrm{Bias}}[\hat{f}]$, and variance, $\widehat{\mathrm{Var}}[\hat{f}]$, on the data set size N. While the bias attains a non-zero but constant value, the estimated variance approaches an $\frac{1}{N}$-asymptote, consistent with theory, cf. eq. (49). The gray dashed line shows the irreducible error contribution σ^2.

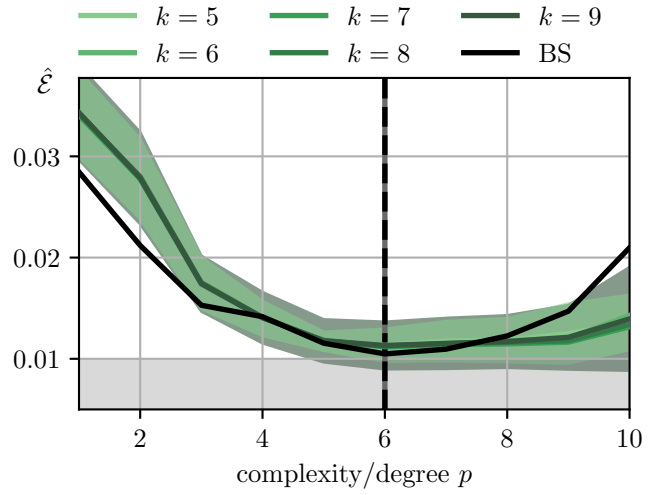

FIG. 8. Comparison of the k-fold cross-validation MSE and the bootstrap MSE of Fig. 6. Green shaded areas show the 1σ-confidence region of $k = 5$ and $k = 9$ respectively. Our data suggests accordance between both techniques: The error minimum is attained for the same polynomial order ($p = 6$) and the bootstrap MSE resides mostly inside the 1σ-confidence interval of the cross-validation result. Also note that the lower bound $\hat{\mathcal{E}} > \sigma^2$ (gray shaded region) is respected by our estimators.

Table I summarizes the result of the OLS-based model selection.

2. *Ridge Regression*

Turning our attention to Ridge Regression, cf. section I A 2, we repeat the analysis of section VI A 2 for the L_2-penalized solution of eq. (12).

Fig. 9 illustrates the bootstrap-inferred MSE as a function of the polynomial degree p and the penalty hyperparameter λ. Multiple observations can be made:

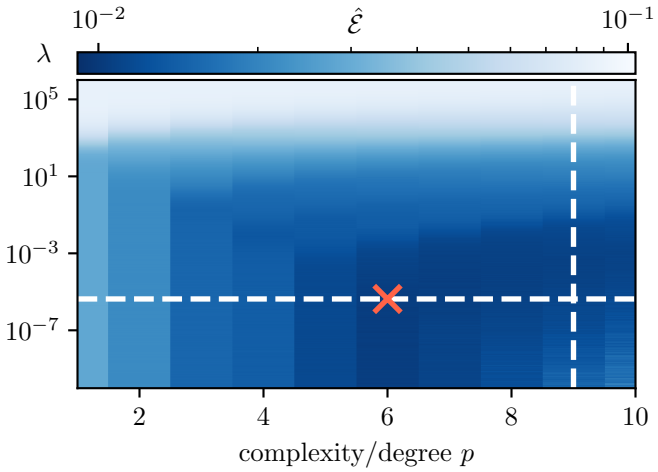

FIG. 9. Mean squared error $\hat{\mathcal{E}}$ under Ridge regression as a function of penalty parameter and model complexity obtained from bootstrap resampling. Note how (i) the MSE saturates for large values of λ (ii) we approach the prototypical OLS behavior for $\lambda \to 0$ with large MSE at low p (underfitting, high bias) and high p (overfitting, high variance) and (iii) an intermediate range of penalties exists which admits a trade-off between the two regimes. The red cross marks the minimum of $\hat{\mathcal{E}}$ and therefore defines the optimal (hyper)parameters $(p_{\mathrm{opt,BS}}, \lambda_{\mathrm{opt,BS}}) = (6, 4 \times 10^{-6})$. A cross section along the white, dashed lines is given in Fig. 10 and 12.

We first assess $\hat{\mathcal{E}}$ at fixed penalization but varying model complexity and observe an error saturation for unrealistically large values of $\lambda > 10^4$ attained independent of the value of p, see also Fig. 10B. This is caused by the parameter shrinkage property of Ridge regression: At some point the penalization term in the cost function of eq. (11) becomes so dominant that all regression parameters $\beta_{k,\mathrm{R}}(\lambda) \to 0$, irrespective of how many there are. One is then effectively left with a trivial intercept model that only captures the mean value of the response variable $\hat{\boldsymbol{z}}$, cf. eq. (12), which is clearly a too simplistic model for Franke's function. Inspection of the MSE cross section at fixed $p = 9$, cf. Fig. 10B, suggests the MSE is dominated by the bias contribution in the high-λ, saturation regime and thus in accordance with our understanding that simplistic models — here essentially just an intercept — increase the predictor bias $\mathbb{E}_{\boldsymbol{X}_\mathcal{V}}\left[\mathrm{Bias}_\mathcal{T}[\hat{f}](\boldsymbol{x}_\mathcal{V})\right]$

In the OLS-limit, i.e. for $\lambda \to 0$, Fig. 9 recovers the qualitative behavior already shown in Fig. 6: Low polynomial degrees suffer from underfitting whereas high values of p tend to overfit the data. The result is the prototypical, trough-shaped MSE curve at fixed, small values of λ.

At intermediate values of λ, say $\lambda = \lambda_{\mathrm{opt,\,BS}} = 4 \times 10^{-6}$ a cross section along the p-direction, see Fig. 12, reveals that *the predictor variance* $\mathbb{E}_{\boldsymbol{X}_\mathcal{V}}\left[Var_\mathcal{T}[\hat{f}](\boldsymbol{x}_\mathcal{V})\right] \approx \widehat{Var}[\hat{f}]$ *is reduced in the presence of* L_2*-regularization compared*

to the OLS scenario with $\lambda = 0$, cf. Fig. 6.

Thus, in total we observe (i) high-bias saturation for $\lambda \to \infty$, (ii) the OLS behavior with high-variance MSEs for $\lambda \to 0$ and (iii) a MSE minimum in between at which point a trade-off between both error contributions defines an optimal value of λ given a fixed model complexity.

Let us attempt to quantify the bias-variance trade off in λ-direction more accurately by analyzing the error component cross section depicted in Fig. 10B taken at $p = 9$. The polynomial order was chosen such that the predictor variance is appreciable for $\lambda \to 0$

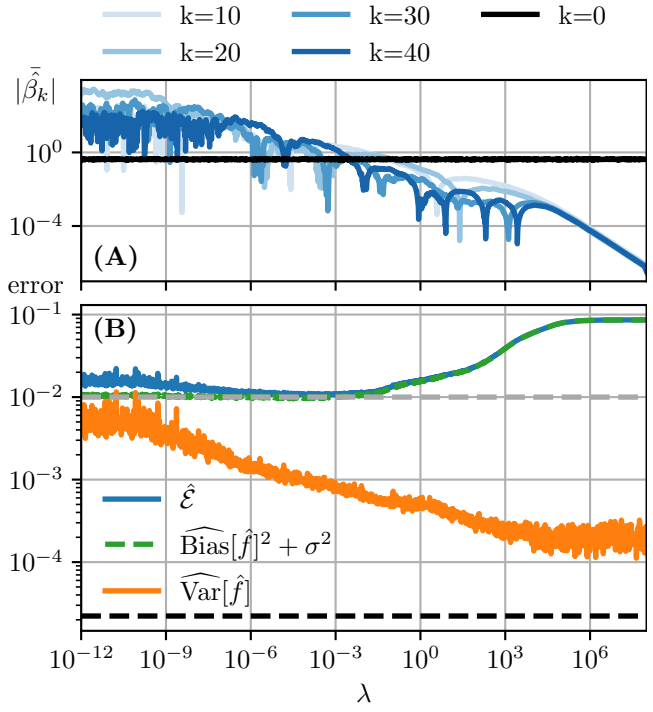

FIG. 10. **(A)**: Magnitude of four L_2-penalized regression parameters $|\bar{\hat{\beta}}_k(\lambda)|$ as a function of λ alongside the intercept $\hat{\beta}_0$. Minimal smoothing was applied to non-intercept parameters to increase visibility. **(B)**: Cross section through Fig. 9 at fixed $p = 9$. Notice how $\widehat{\mathrm{Bias}}[\hat{f}]$ is an increasing function of λ, whereas $\widehat{\mathrm{Var}}[\hat{f}]$ decreases and is bound from below by eq. (51). The MSE, being the sum of both terms, is minimized at $\lambda^* \approx 10^{-3}$.

Here, the MSE $\hat{\mathcal{E}}$ is minimal for $\lambda^* \approx 10^{-3}$ and, as expected, it is the variance estimator $\widehat{\mathrm{Var}}[\hat{f}]$ that dominates for $\lambda < \lambda^*$, while $\widehat{\mathrm{Bias}}[\hat{f}]$ determines $\hat{\mathcal{E}}$ for $\lambda > \lambda^*$. Note how the saturation of $\widehat{\mathrm{Bias}}[\hat{f}]$ sets in when the regression parameters in Fig. 10A attain their asymptotic decay at $\lambda_{\mathrm{sat}} \gtrapprox 10^4$. What remains past this point is the constant, mildly fluctuating "intercept model" mentioned before. At the same time, the decaying variance estimate $\widehat{\mathrm{Var}}[\hat{f}]$ reaches a lower bound which can be associated with the intercept fluctuation.

To justify this claim, we recall the general result for the variance of linear predictors in eq. (47) and set $\hat{\boldsymbol{\beta}} = \hat{\boldsymbol{\beta}}_{\mathrm{R}}(\lambda)$. Substituting eq. (14) yields:

$$\mathrm{Var}_{\mathcal{T}}[\hat{f}](\boldsymbol{x}_{\mathcal{V}}) = \sigma^2 \times \boldsymbol{x}_{\mathcal{V}}^{\intercal} \begin{pmatrix} \frac{1}{N_{\mathcal{T}}} + \bar{\boldsymbol{X}}^{\intercal}\boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^{\intercal}\bar{\boldsymbol{X}} & -\bar{\boldsymbol{X}}^{\intercal}\boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^{\intercal} \\ -\boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^{\intercal}\bar{\boldsymbol{X}} & \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^{\intercal} \end{pmatrix} \boldsymbol{x}_{\mathcal{V}} \,, \quad (50)$$

with $\boldsymbol{D} = \mathbf{Diag}\left[\frac{\sigma_i^2}{(\sigma_i^2+\lambda)^2}\right]$ and $\boldsymbol{U}$ denoting the orthogonal matrix of $\boldsymbol{X}^{\intercal}\boldsymbol{X}$'s eigenvectors. Thus, for $\lambda \to \infty$, one arrives at the lower bound:

$$\widehat{\mathrm{Var}}[\hat{f}] \approx \mathbb{E}_{\boldsymbol{X}_{\mathcal{V}}}\left[\mathrm{Var}_{\mathcal{T}}[\hat{f}](\boldsymbol{x}_{\mathcal{V}})\right] \geq \frac{\sigma^2}{N_{\mathcal{T}}} \quad (51)$$

which is shown as black dashed line in Fig. 10. Interestingly, our experiments show that the derived bound is rather conservative and only becomes sharp for noise with $\sigma > 0.3$.

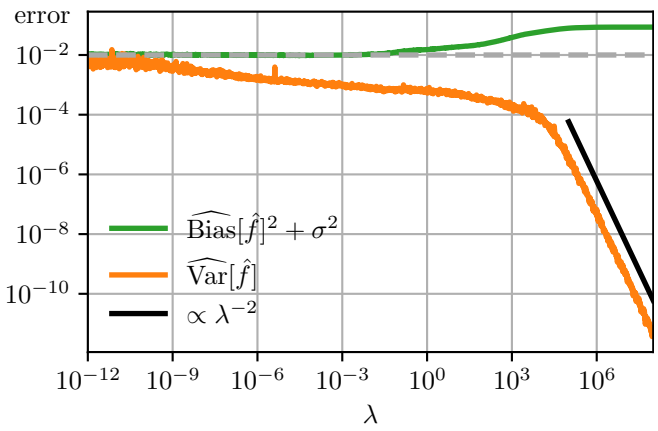

FIG. 11. As in Figure 10 but *excluding* the intercept in the linear model. This allows to observe the $\frac{1}{\lambda^2}$ asymptote of the estimated predictor variance $\widehat{\mathrm{Var}}[\hat{f}]$ as suggested by eq. (50).

Evidently eq. (50), suggests $\mathbb{E}_{\boldsymbol{X}_{\mathcal{V}}}\left[\mathrm{Var}_{\mathcal{T}}[\hat{f}](\boldsymbol{x}_{\mathcal{V}})\right]$ to be asymptotic to $\frac{1}{\lambda^2}$, but the presence of a non-zero, and thus fluctuating, intercept disguises this behavior in Fig. 10. We therefore rerun our analysis by setting the intercept *exactly* to zero. The result in Fig. 11 is in accordance with our theoretical argument.

Moving on to model selection under Ridge regression and Fig. 12 illustrating a fixed $\lambda = \lambda_{\mathrm{opt,BS}} = 4 \times 10^{-6}$ cross section through Fig. 9 that passes through the global minimum of $\hat{\mathcal{E}}$. The characteristics of this plot have been thoroughly described in section VI A 1 and we only mention in passing that the main difference to the OLS version in Fig. 6 is the suppressed variance at high p. We infer $p = 6$ to be the ideal model complexity (black dashed line).

What remains is checking the bootstrap results against the optimal (hyper)parameters inferred from cross-validation. Settling for $K = 5$ folds, one arrives at Fig. 13 and its cross section at the optimal penalty parameter $\lambda = \lambda_{\mathrm{opt,CV}} = 8 \times 10^{-6}$. Given the enormous range of

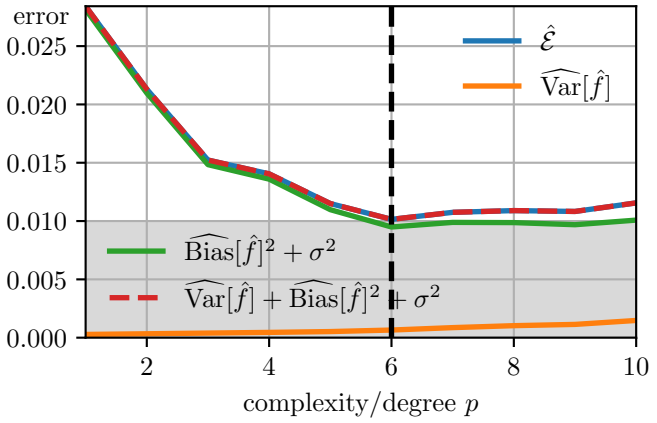

FIG. 12. Bias-variance decomposition at fixed optimal value $\lambda = \lambda_{\mathrm{opt,BS}}$. Notice how the variance is suppressed at high p compared to the OLS scenario in Fig. 6. The black dashed line depicts the polynomial order $p = 6$ minimizing $\hat{\mathcal{E}}$.

penalty parameters spanned, we deem this result to be consistent with $\lambda_{\mathrm{opt,BS}}$, although a more rigorous comparison would certainly be possible.

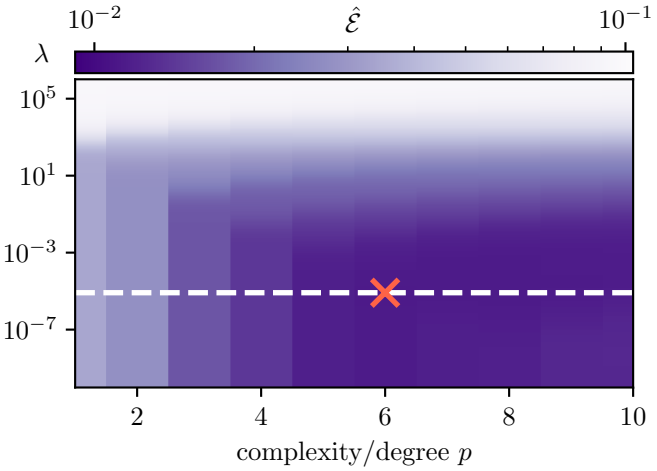

FIG. 13. Mean squared error $\hat{\mathcal{E}}$ under Ridge regression as a function of penalty parameter and model complexity obtained from 5-fold cross-validation. The result is in qualitative accordance with its bootstrap counterpart in Fig. 9. The global minimum of the MSE is attained at $(p, \lambda_{opt,CV}) = (6, 8 \times 10^{-6})$ corresponding to the red cross. The error cross section along the white, dashed line is illustrated in Fig. 14.

By minimizing the $\hat{\mathcal{E}}$ cross section, Fig. 14 yields $p = 6$ as the ideal polynomial degree, exactly as suggested by the bootstrap analysis. The compatibility of both MSE curves in Fig. 14 is reassuring. We refer to Table I for an overview of the model selection results under Ridge regression.

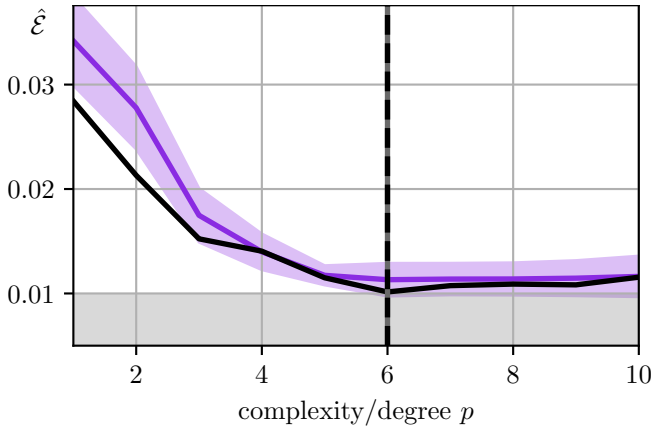

FIG. 14. Comparison between bootstrap and cross-validation along their respective optimal values of λ. The purple, shaded area corresponds to the 1σ-confidence interval of $\hat{\mathcal{E}}$ deduced from cross-validation, while the gray shaded region is the lower bound for the MSE suggested by the irreducible error σ^2. One finds satisfactory agreements between both methods and a shared optimal value of $p_{\mathrm{opt,BS}} = p_{\mathrm{opt,CV}} = 6$ (black, dashed and solid, gray line).

3. *LASSO*

We conclude this section with an analysis of LASSOs predictor characteristics and performance when applied to noisy Franke function realizations. That said, our discussion will be brief compared to section VI A 1 or VI A 2 for multiple reasons: Firstly, many qualitative properties found for Ridge regression carry over to the L_1-regularization case and we will therefore only focus on the differences between the two methods. Secondly, the lack of an analytical result for the regression parameters under LASSOs minimization procedure makes it impossible to predict its asymptotic behavior. Moreover, the computational expense required to arrive at converged results, especially when the penalization parameter is small and bootstrapping is employed, prohibits elaborate sweeps through the $\lambda - p$-(hyper)parameter space. Cross-validation is less affected from this as $\frac{K}{B} \ll 1$. However, to keep the comparison between the two resampling strategies consistent, one is confined to a patch of the parameter space that is accessible to both strategies. In practice that means instead of spanning more than 20 orders of magnitude in λ we confine ourselves to $\lambda \in [2 \times 10^{-4}, 1]$, knowing that this will have implications for the selection and performance of the best regression model.

Let us turn our attention to the bootstrap-inferred MSE $\hat{\mathcal{E}}$ in Fig. 15 and note in passing that the computation of the illustrated results required $t \approx 30\,\mathrm{min}$ on 8 CPU cores. Pushing the penalization down to $\lambda < 10^4$ resulted in runtimes well in excess of 1 h and were therefore omitted.

Evidently, the high-bias, saturation regime already known from Ridge regression, is recovered for penalizations $\lambda > 0.1$ and $\hat{\mathcal{E}}$ decreases monotonically as

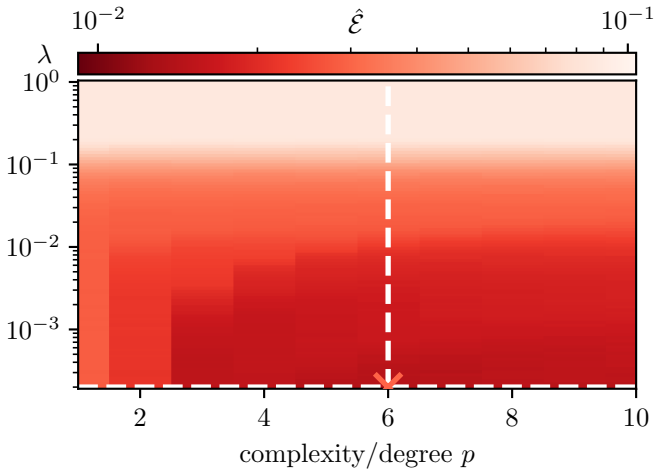

FIG. 15. Bootstrap-based estimation of the MSE $\hat{\mathcal{E}}$ as a function of L_1-penalization and model complexity under LASSO regression. The red cross depicts the localization of the minimum of $\hat{\mathcal{E}}$ and the cross section along the white, dashed lines are depicted in Fig. 16 and Fig. 17. Note how an increasing value of λ degrades the estimated, expected prediction error.

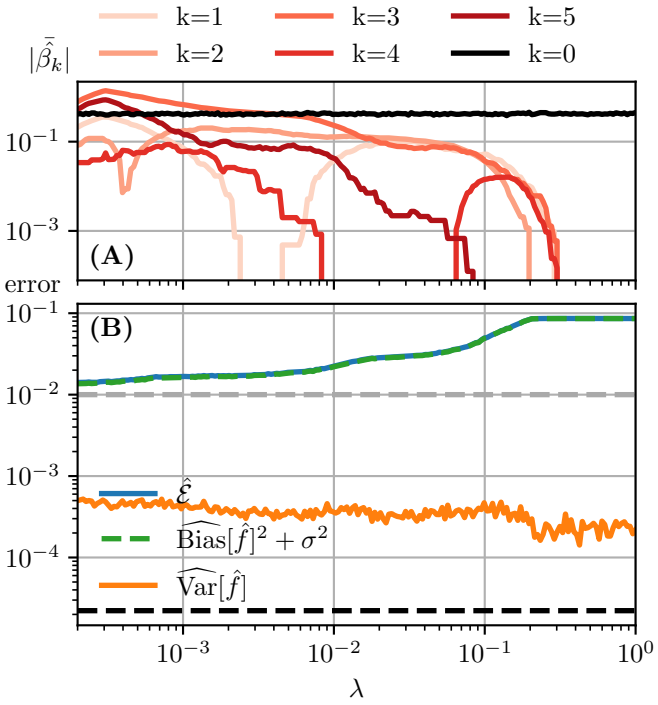

FIG. 16. **(A)**: Magnitude of 5 L_1-penalized regression parameters $|\bar{\hat{\beta}}_k(\lambda)|$ as a function of λ alongside the intercept $\hat{\beta}_0$. Minimal smoothing was applied to non-intercept parameters to increase visibility. A crucial difference to its Ridge counterpart in Fig. 10 is LASSOs feature selection property — regression parameters are set to zero depending on the value of λ. **(B)**: Cross section through Fig. 15 at fixed $p_{\mathrm{opt,BS}} = 6$. Notice how $\widehat{\mathrm{Bias}}[\hat{f}]$ is an increasing function of λ. The MSE, being the sum of both contributions, does not attain a true minimum due to trade-off lack between $\widehat{\mathrm{Var}}[\hat{f}]$ and $\widehat{\mathrm{Bias}}[\hat{f}]$ in the probed range of penalization parameters. $\hat{\mathcal{E}}$ saturates for $\lambda > \lambda_{\mathrm{sat}} \approx 0.2$ when all of the regression parameters are set to zero and a trivial intercept model remains.

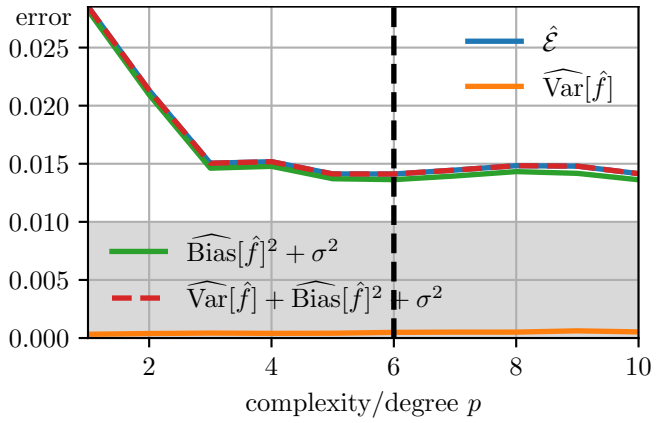

FIG. 17. Cross section through Fig. 15 at fixed $p_{\mathrm{opt,BS}} = 2 \times 10^{-4}$. As in the case of Fig. 12, one finds the addition of an L_1-regularizer to suppress the predictor variance and increase the predictor bias. Notice the gap between $\hat{\mathcal{E}}$ and its lower bound set by the irreducible error σ^2 (gray shaded area) — a consequence of a non-ideal penalization parameter.

$\lambda \to 2 \times 10^{-4}$. Unfortunately, and in contrast to the situation of Fig. 9 or Fig. 10, our limited penalty range prohibits us to observe a bias-variance trough in λ direction.

A consequence of our failure to do so directly implies that we should not expect to find LASSOs true global minimum and thus not the best possible model. Deducing the local minimum of Fig. 15 at $(p_{\mathrm{opt,BS}}, \lambda_{\mathrm{opt,BS}}) = (6, 2 \times 10^{-4})$ (the red cross) confirms this deficiency as it sits right on the boundary of the probed parameter space patch. Put differently, for the situation at hand, LASSO favours penalization values even smaller than were computationally accessible (at least if we insist on the presented parameter space resolution and polynomial order range).

Taking the error cross section at $p_{\mathrm{opt,BS}} = 6$ yields Fig. 16B which we amend with the regression parameter evolution of the first five, non-trivial features and the unpenalized intercept β_0 in Fig. 16A. A notable difference between Fig. 16 and its Ridge counterpart in Fig. 10 is that LASSO sets non-essential regression parameters *exactly* to zero if the penalization is sufficiently strong. This occurs significantly earlier (around $\lambda_{\mathrm{sat}} \approx 0.2$) compared to Ridge, and the MSE remains in a saturated, high-bias state past this point. The estimated predictor variance $\widehat{\mathrm{Var}}[\hat{f}]$ shows no major decline in the probed penalization range and may already be dominated by the intercept variance deduced in eq. (51) — a result which should still hold true in the present case given that the intercept does not enter into the analytically inaccessible cost function minimization.

By keeping $\lambda = \lambda_{\mathrm{opt,\,BS}}$ fixed, one arrives at the bias-variance decomposition of Fig. 17 from which two observations can be deduced.

Firstly, as was the case under L_2-penalization, the estimated predictor variance $\widehat{\mathrm{Var}}[\hat{f}]$ is highly suppressed.

Secondly, the estimated MSE $\hat{\mathcal{E}}$ is mainly driven by the residual bias due to our rather large value of λ and the prediction performance is significantly reduced compared to both OLS, cf. Fig. 6, and Ridge, see Fig. 12 which reduce $\hat{\mathcal{E}}$ almost down to the irreducible noise σ^2.

Moving on to the 5-fold cross-validation-based prediction error estimation of Fig. 18 and Fig. 19, one finds, yet again, good qualitative and quantitative accordance of both resampling techniques: Both methods report $\lambda_{\mathrm{opt,BS}} = \lambda_{\mathrm{opt,CV}} = 2 \times 10^{-4}$ as "optimal" regularization parameter and overlaying both MSE curves for this penalization yields mostly compatible results, cf. Fig 19. The only difference lies in the selected optimal polynomial degree for which cross validation reports $p_{\mathrm{opt,\ CV}} = 10 \neq 6 = p_{\mathrm{opt,\ BS}}$ — a result somewhat irritating given LASSOs parameter shrinkage *and* selection property. That said, the MSE declines only mildly as a function of p and all $\hat{\mathcal{E}}$-values past $p \geq 5$ are essentially compatible with each other. Nevertheless, we will choose $p_{\mathrm{opt,\ CV}} = 10$ as our final, ideal LASSO model to adhere to the outlined selection procedure of section I C. The reader is referred to Table I for an overview of the selected "ideal" LASSO-optimized model (hyper)parameters.

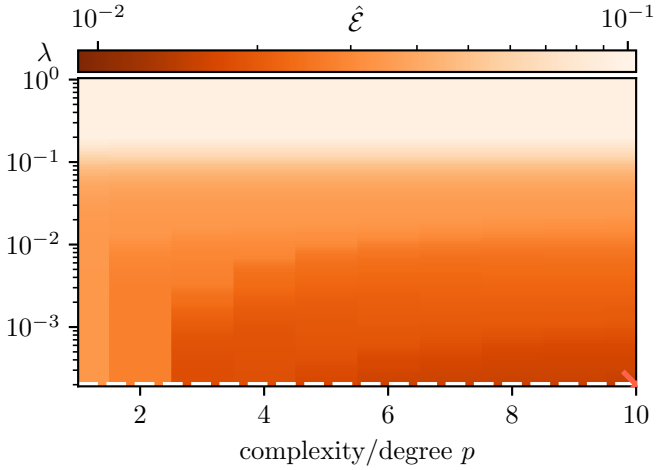

FIG. 18. MSE as a function of L_1-penalization and polynomial order inferred from 5-fold cross-validation. The minimal value of $\hat{\mathcal{E}}$ is attained at $(p_{\mathrm{opt,CV}}, \lambda_{\mathrm{opt,CV}}) = (10, 2 \times 10^{-4})$. We illustrate the cross section at $\lambda = \lambda_{\mathrm{opt,CV}}$ (white, dashed line) in Fig. 15.

4. *Summary*

With optimal OLS, Ridge or LASSO-trained models at our disposal, we are now in a position to assess the performance of the three side-by-side.

Fig. 20 illustrates the function surface predicted by the optimal models when exposed to unseen data — in this case an uniform $x - y$ cartesian grid. Also shown are the

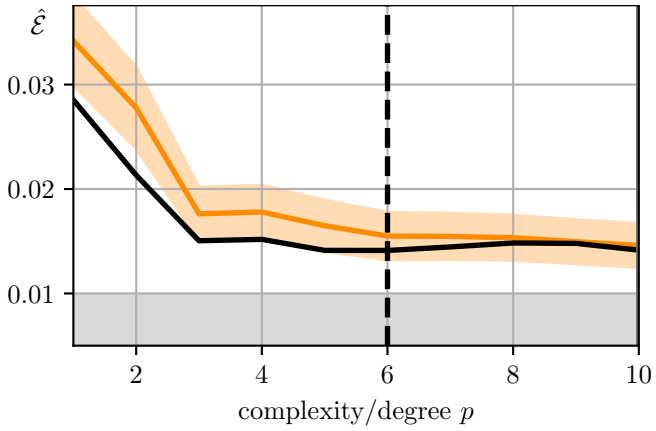

FIG. 19. Comparison of the bootstrap and cross-validation MSE at $\lambda = \lambda_{\mathrm{opt,CV}} = \lambda_{\mathrm{opt,BS}}$. As before, one finds reasonable agreement between both resampling strategies. While bootstrapping suggests an ideal polynomial order of $p_{\mathrm{opt,BS}} = 6$, 5-fold cross-validation favours $p_{\mathrm{opt,BS}} = 10$. We deem this discrepancies insignificant given that the MSE difference at $p = 6$ and $p = 10$ is only marginal and well within the respective confidence intervals (orange shaded area). The gray shaded area illustrates the excluded region for $\hat{\mathcal{E}}$ set by the irreducible error σ^2.

associated residuals, i.e. the absolute difference between Fig. 20A/C/E and the ground truth of Fig. 1.

A superficial inspection suggests that the OLS and Ridge-trained model perform equally well as there are no apparent discrepancies in the characteristics or magnitude of their respective residuals. LASSO, by comparison, performs significantly worse: The main peak close to $\boldsymbol{x}^\intercal = \begin{pmatrix} 0 & 0 \end{pmatrix}$ is to broad and shallow and the side peak of Franke's function along the x-axis is barely visible. In general LASSOs predicted function surface appears to rigid and not sufficiently adapted to the fine-grain details of Fig. 1. As explained in section VI A 3, this is a clear indication of a too large penalty parameter implying a high-bias in the prediction performance.

We express these qualitative findings more quantitatively in Table I, which reports — aside from the results of the model selection process — the mean square test error inferred from a newly generated, and thus unseen, test set $\mathcal{S}$ of $N_{\mathcal{S}} = 200$ noisy Franke function values at randomly located sights $\boldsymbol{x} \in [0, 1)^2$. Unsurprisingly, OLS and Ridge regression perform equally well: both produce mean squared errors which are identical in magnitude and also comply with the MSE concordance interval set by the cross-validation analysis. LASSOs test MSE, $\mathrm{MSE}_{\mathcal{S}} = 0.018$, on the other hand, is 38% worse than the OLS or Ridge regression result.

We conclude that for the regression problem at hand, adding a numerically involved L_1-regularizer does not yield any benefits in terms of minimizing the overall mean squared error by adding bias but reducing variance. In fact, it actually degrades the model performance.

The uncompetetive model performance under LASSO

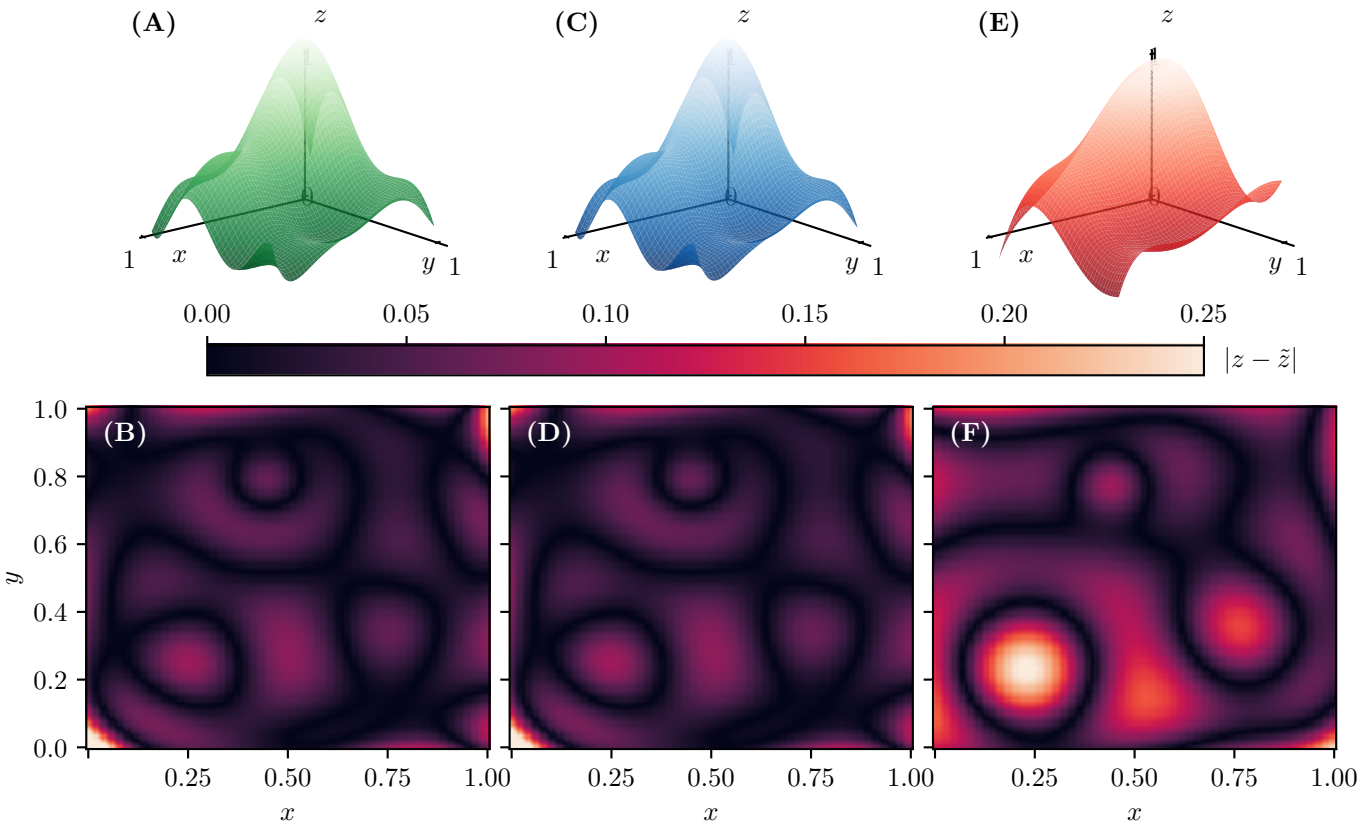

FIG. 20. Visual comparison of the three optimal models, trained by **(A/B)** OLS, **(C/D)** Ridge and **(E/F)** LASSO. The bottom row depicts the residual deviation $|\boldsymbol{z} - \tilde{\boldsymbol{z}}|$ between prediction and the true values of Franke's function.

	OLS	Ridge	LASSO
p_{opt} (CV)	6	6	10
λ_{opt} (CV)	-	8×10^{-6}	2×10^{-4}
$\hat{\mathcal{E}}$ (CV)	$0.011^{+0.002}_{-0.001}$	$0.011^{+0.002}_{-0.001}$	0.015 ± 0.002
$\text{MSE}_{\mathcal{S}}$, eq. (38)	0.013	0.013	0.018

TABLE I. Summary of the model selection part carried out in section VI A 1 - VI A 3. Based on the overall consistency between bootstrap and cross-validation results, we only report parameters deduced from the latter. It is worth noting that LASSOs results are not optimal in the sense that $(p_{\text{opt}}, \lambda_{\text{opt}})$ are not the location of the global minimum of $\hat{\mathcal{E}}$, which was not observable due to constraints on the probed parameter space volume. The last line reports the mean square test error $\text{MSE}_{\mathcal{S}}$, cf. eq. (38), of each optimal model evaluated on newly generate, and hence unseen, data points.

can certainly be mitigated if one accepts a longer time-to-solution and explores a larger patch of the (hyper)parameter-space. That said, it is questionable whether this is worth doing as even with an analytical, and hence fast, L_2-penalization approach, Ridge regression was unable to outperform vanilla OLS. Therefore, we deem *OLS as the most suited approach to approximate Franke's function by means of a two-dimensional polynomial model.*

B. Stochastic Gradient Descent for Linear Regression

Section I B introduced the gradient descent machinery as an effective tool for minimizing differentiable cost functions when no analytical formula is available. Clearly, it's application for training a Logistic Regression model is natural and we refer to section VI C for that matter.

For the purpose of assessing SGDs behavior as a function of its hyperparameters and exposing potential pitfalls when choosing them, this section is concerned with its application to OLS and Ridge Regression, i.e. when a "ground-truth" in form of a numerically tractable, analytical solutions, cf. eq. (7) and eq. (12), exists.

We base our study on a Franke data set with $N = N_{\mathcal{T}} = 2^{11}$ data points, a polynomial model of degree $p = 4$ or $P = 15$ features (including the intercept), and the relative deviation:

$$\mathcal{R}_{\boldsymbol{v}}(\boldsymbol{w}) \equiv \frac{||\boldsymbol{v} - \boldsymbol{w}||_2}{||\boldsymbol{v}||_2}. \tag{52}$$

Common choices will be $\boldsymbol{v} = \hat{\boldsymbol{\beta}}_{\text{OLS}}$ or $\boldsymbol{v} = \hat{\boldsymbol{\beta}}_{\text{R}}(\lambda)$ taken from eq. (7) or eq. (12) respectively and $\boldsymbol{w}$ will denote the solution obtained via SGD configured by a varying set of hyperparameters. Our study will focus on (i) the magnitude of a constant learning rate γ, (ii) the number

of mini-batches B, (iii) the number of epochs and (iv) the learning rate schedule. Note that early convergence tests, as in section VI C, are not used. Intuitive explanations of the observed effects will be preferred over rigorous mathematical justifications.

Whenever informative, an additional comparison with `sklearn`'s $B = N_{\mathcal{T}}$ SGD implementation, `SGDRegressor` (abbreviated `sk`), is provided. Note that several, partially regression method dependent, hyperparameter rescalings are necessaries to make these comparisons consistent — a consequence of slightly varying cost function definitions. To make the issue manifest, we remind the reader that `SGDRegressor` minimizes, at least conceptually, the cost function, [9]:

$$C_R^{\text{sk}}(\beta_0, \boldsymbol{\beta}) = \frac{1}{2N_{\mathcal{T}}} \sum_{i=1}^{N_{\mathcal{T}}} (y_i - f(\boldsymbol{x}_i, \beta_0, \boldsymbol{\beta}))^2 + \frac{\lambda}{2} \sum_{k=1}^{P-1} \beta_k^2 \,. \tag{53}$$

A trivial rescaling directly inferable is that the learning rate entering `SGDRegressor` $\gamma_{\text{sk}} = 2\gamma$ in order to be consistent with the discussion in section VI B. Additional rescalings, due to an imbalance between the regularizer term and the OLS part in eq. (53) are introduced when necessary.

1. *Ordinary Least Squares*

We begin by applying OLS to our synthetic Franke function training data set and infer the regression parameters with both eq. (7) and our SGD implementation `StochasticGradientDescent` (abbreviated `SGD`). Figure 21 reports the latter's accuracy compared to the analytical solution as a function of fixed, but different learning rates γ. All other hyperparameters, i.e. mini-batch number ($B = N$) and number of epochs ($E = 1000$), are fixed.

Evidently, convergence is not achieved for extremely small learning rates, but improves steadily until $\gamma \approx 0.02$, when the minimal deviation of $\mathcal{R}_{\hat{\boldsymbol{\beta}}_{\text{OLS}}} \approx 5\%$ is reached. For $\gamma > 0.02$ a strong divergence is observable.

The behavior is completely plausible: Given a fixed number of epochs and identical initial conditions ($\hat{\boldsymbol{\beta}}^0 = \mathbf{0}$), miniscule learning rates prohibit the algorithm to descent into the global minimum as "not enough ground can be covered" in the time available, whereas larger learning rates imply larger descent path lengths which ultimately lead into the vicinity of the global minimum. Extremely large values of γ, on the other hand, are numerically unstable as they overshoot into regions far away from the minimum with continuously increasing gradients thus propelling the divergence.

We also draw attention to the superb agreement between `sklearn`'s `SGDRegressor` and our modest implementation.

To assess how the number of mini-batches B influences the convergence behavior, we fix $\gamma = 0.02$, stick with $E = 1000$ epochs and perform a sweep in B. For simplicity, we confine ourselves to powers-of-two, but note

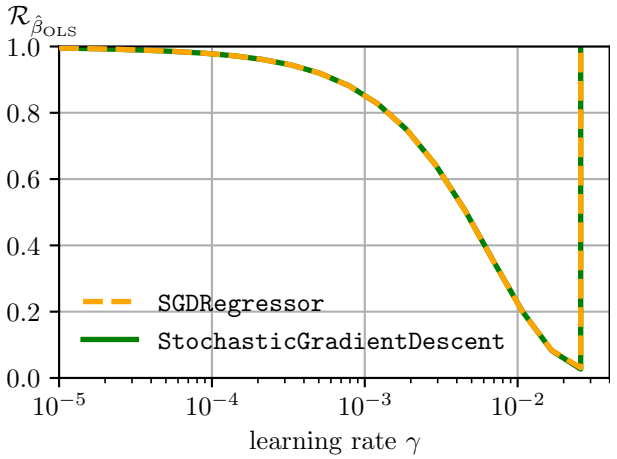

FIG. 21. Comparison of the relative deviation of `StochasticGradientDescent` and `SGDRegressor` to the analytical solution of eq. (7). Experiment conducted with $B = M$ mini-batches, i.e. true SGD, and $E = 1000$ epochs. Besides the excellent accordance between both implementations, we observe continuously increasing accuracy until $\gamma \approx 0.02$ and a strong numerical instability for $\gamma > 0.02$.

that `StochasticGradientDescent` is perfectly capable to deal with any mini-batch number $1 \leq B \leq N_{\mathcal{T}}$. Figure 22 illustrates the results.

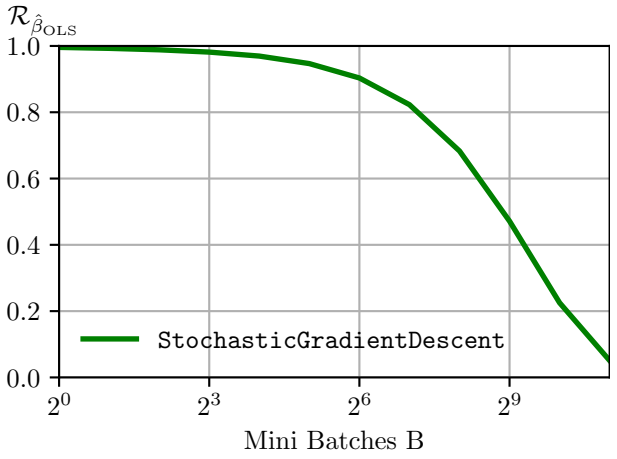

FIG. 22. Accuracy of `StochasticGradientDescent` as a function of the number of mini-batches B. Evidently, increasing B improves the accuracy. We explain this behavior as a consequence of the higher number of equally effective descent updates at high B if the averaged gradient per mini-batch remains a sound approximation to the full-set gradient.

One finds randomized GD with $B = 1$ to perform worst and $B = N_{\mathcal{T}} = 2^{11}$ SGD to be most accurate. In general, increasing the number of mini-batches appears to increase the observed accuracy. We deem this observation to be plausible as well: If the mean gradient derived from the randomized mini-batch is a solid approximation to the full-set gradient used in GD, then having more mini-batches increases the number of descent steps per

epoch. Put differently, while GD with $B = 1$ performs only one update per epoch, B-mini-batch SGD undergoes B, potentially equally good, updates per epoch and thus reduces the total number of iterations required to converge to the global minimum. We conjecture the strict convexity of the cost function in eq. (6) might help the averaged mini-batch gradient in approximating the averaged full-set gradient well enough, although this is nothing more than a claim at this point.

Next, we fix the number of mini-batches to $B \in \{\frac{N_\mathcal{T}}{4}, N_\mathcal{T}\}$, stick with $\gamma = 0.02$ and observe `SGD`'s accuracy as a function of the number of epochs. Fig. 23 illustrates the result including a comparison to the `sklearn` implementation. Note that we fix the seed of the random number generator for this experiment, such that stopping M `SGD` descents at epochs $E = E_1 \dots E_M$ respectively is equivalent to following one descent until $E = E_M$ and taking data snapshots at $E = E_1 \dots E_M$.

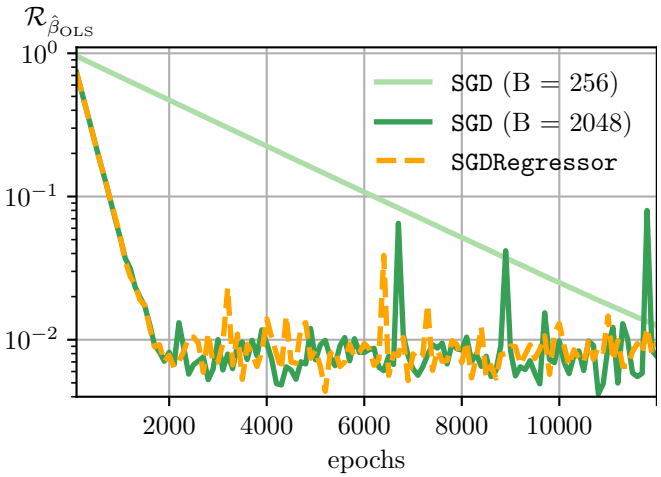

FIG. 23. Comparison of the relative deviation of `StochasticGradientDescent (SGD)` and `SGDRegressor` to the analytical solution of eq. (7) as a function of decent epoch. Other hyperparameters were fixed to $\gamma = 0.02$ and $B \in \{\frac{N_\mathcal{T}}{4}, N_\mathcal{T}\}$ mini-batches. Before a lower threshold for the accuracy is reached, all illustrated descents approach the global minimum $\hat{\boldsymbol{\beta}}_{\mathrm{OLS}}$ in an exponential manner.

Interestingly, and somewhat surprising, `SGD` reduces the distance to the global minimum with *exponential* speed before it reaches a lower accuracy limit. The authors are neither aware of an intuitive explanation nor a rigorous mathematical argument for this observation. We again hypothesize that the strict convexity of eq. (6) might be responsible for this favorable result. A more elaborate study might investigate the actual convergence rate $\rho = \lim_{j\to\infty} \frac{|\hat{\boldsymbol{\beta}}^{j+1} - \hat{\boldsymbol{\beta}}_{\mathrm{OLS}}|}{|\hat{\boldsymbol{\beta}}^{j} - \hat{\boldsymbol{\beta}}_{\mathrm{OLS}}|}$ to gain further insights.

Before turning our focus to Ridge regression, let us conclude with a brief comparison of the different learning rate schedules introduced in section I B. Consider Fig. 24 for the result in which both the achieved accuracy, Fig 24B, and the associate learning rate evolution, Fig. 24A, are depicted. Recall `AdaGrad` and `RMSProp` use feature-specific learning rates and Fig. 24A therefore shows the feature-averaged value of γ at descent time j. The remaining hyperparameters wer fixed to $E = 1000$ epochs, an initial learning rate of $\gamma_0 = 0.02$ and $B = 2^8 = \frac{N_\mathcal{T}}{4}$ mini-batches.

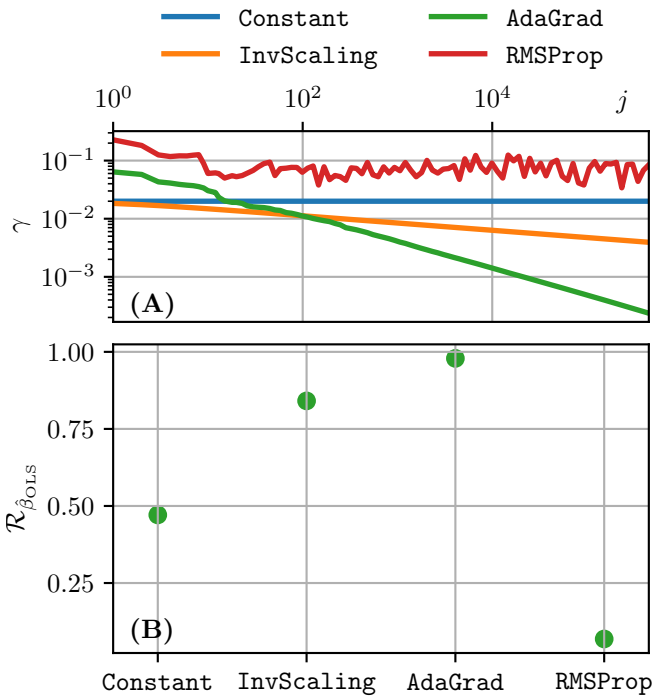

FIG. 24. **(A)**: Learning rate evolution under `InvScaling`, `AdaGrad` and `RMSProp`, see section I B, with $\gamma = 0.02$ **(B)**: Comparison of the relative deviation of `StochasticGradientDescent` to the analytical solution of eq. (7) for the aforementioned learning rate schedules. Other hyperparameters were fixed to $E = 1000$ epochs and $B = 2^8$ mini-batches. One finds schedules which adopt a (quasi)-fixed, large learning rate (`Constant`, `RMSProp`) to outperform schedules with decaying learning rates (`InvScaling`,`AdaGrad`).

Our results indicates that for the problem at hand `RMSProp` outperforms all other learning schedules, followed by a constant rate of $\gamma = \gamma_0$. The power-law decay schedule, `InvScaling`$(j) = \frac{\gamma_0}{j^{1/8}}$, and `AdaGrad` yield poor accuracy in comparison.

Consulting the learning rate evolution in Fig 24A explains this behavior quite naturally: Both `InvScaling` and `AdaGrad` reduce the learning rate, while `RMSProp` equilibrates at a large, albeit fluctuating, value of $\gamma > \gamma_0$. Recall that `AdaGrad` sums up *all* past squared gradients, hence the continuous decay in λ. `RMSProp`, by contrast, uses a moving average for $(\nabla_{\boldsymbol{\beta}} C_{\mathrm{OLS}})^2$. In accordance with the results of Fig. 21, large values of γ yield better results for the problem at hand. It is worth emphasizing that the equilibirum value for γ under `RMSProp` is not at odds with the divergence for learning rates $\gamma > 0.02$ since we only report the feature-averaged values.

2. Ridge Regression

Next, we reproduce the above analysis for the Ridge regression solution of eq. (12). To do so, an addtional hyperparameter rescaling must be introduced to establish comparability between (i) the analytical solution of eq. (12), (ii) `StochasticGradientDescent` and (iii) `SGDRegressor`. For this, we turn to eq. (11) and eq. (53).

Realize that in the analytical setting, there is *one* regularization term for $N_{\mathcal{T}}$ OLS terms, whereas the cost function of `SGDRegressor` implies one regularization term per sample (due to the leading $\frac{1}{N_{\mathcal{T}}}$ normalization). Hence, to reestablish the relative importance of both contributions, we must set $\lambda_{\texttt{sk}} = \frac{\lambda}{N_{\mathcal{T}}}$ and by extension of this argument to B-mini-batch SGD, $\lambda_{\texttt{SGD}} = \frac{\lambda}{B}$. Put differently, to ensure we minimize the same cost function as eq. (11), we need to distribute the L_2- regularizer equally across all $1 \leq B \leq N_{\mathcal{T}}$ mini-batches.

Now, reproducing the conditions of Fig. 21, i.e. $B = N_{\mathcal{T}}$ and $E = 1000$, yields Fig. 25A, illustrating the relative accuracy of `StochasticGradientDescent` and Fig. 25B depicting the absolut deviation of our implementation to `SGDRegressor`. Gray areas represent divergent results defined by $\mathcal{R}_{\hat{\boldsymbol{\beta}}_R} > 1$.

The low-λ, OLS limit is identical to the results of section VI B 1 and we thus concentrate on the qualitatively new convergence regime emerging at $\lambda > 10^{-1}$. More precisely, we find a rapidly converging parameter patch for $\lambda > \frac{1}{N\gamma}$ (red line).

Our interpretation for this is quite simple. Recall Ridge regression's shrinkage property: For increasing penalizations, all regression parameters decay continuously to zero (but are not exactly set to zero). In section I D it was argued $\hat{\boldsymbol{\beta}}^0 = \mathbf{0}$ are reasonable initial conditions for the descent as it complies with the inequality constraint imposed by fixing a particular, but arbitrary, value of λ. Now, by increasing the penalty strength, parameter shrinkage moves the global minimum of the cost function closer to the origin. Consequently, descent paths of ever smaller learning rates are able to reach the minimum within the maximum descent time set by the number of epochs. The observed inverse proportionality $\lambda\gamma = \text{const.}$ supports this argument.

Fixing $\gamma = 0.02$, $E = 1000$ but varying the number of mini-batches B yields qualitatively similar results depicted in Figure 26

For $\gamma < 10^{-1}$, the OLS behavior of Fig. 22 is recovered, i.e. an increasing number of mini-batches improves accuracy due to a higher, similarly effective number of descent steps compared to low values of B. In the penalization dominated regime, $\gamma > 10^{-1}$ the same fast convergence region emerges as observed in Fig. 25. Since the mini-batch number is proportional to the number of effective updates per epoch, the argument for the convergence patch of Fig. 25 carries over with minor modification: For sufficiently large values of λ, even a limited number of descent steps (mini-batch number) is able to reach the origin-approaching global minimum. By contrast, if the penalization is less dominant, more steps (more mini-batches) are required to cover the distance to the (far away) global minimum.

Fig. 27 depicts the situation for fixed learning rate $\gamma = 0.02$, constant number of mini-batches $B = \frac{N_{\mathcal{T}}}{4} = 2^8$ but varying epoch number until the descent is terminated.

Our results are in complete accordance with our expectation: Consulting Fig. 25, taken at $E = 1000$ epochs, suggests that for $\gamma = 0.02$, penalties around and above $\lambda = 10^{-1}$ should be fully converged. This is confirmed by Fig. 27 reporting the transition between not-yet-converged descents and fully converged runs at $\lambda = 10^{-1}$.

At last, the adaptive learning schedules of section I B are compared for $E = 1000$, $\gamma_0 = 0.02$ and $N = 2^8$. We refer to Fig. 28 for the results.

In accordance with Fig. 24, quasi-constant schedules like `Constant` and `RMSProp` outperform decaying learning rates (`InvScaling` and `AdaGrad`) substantially. This changes in the regularizer-dominated regime for $\lambda > 10^{-1}$. Here, decaying strategies reach almost perfect accuracy compared to the steadily degrading accuracy of `Constant` and `RMSProp`.

We interpret this result in the light of previous findings: Since the global minimum moves closer to the origin, smaller, ultimately infinitesimal step sizes are better suited to hit $\hat{\boldsymbol{\beta}}_R$ exactly. Larger learning rates, by contrast, constantly overshoot the global minimum and thus have a larger distance to $\hat{\boldsymbol{\beta}}_R$ on average. Further investigations would be required to pin down the exact cause of the accuracy loss. That said, our result clearly shows that sophisticated strategies for tweaking γ are not a "be-all and end-all" solution for setting the hyperparameters of stochastic gradient descent.

C. Classification for Breast Cancer Data

Finally, we shift our attention to binary classification and evaluate the classification performance of logistic regression and kernel support vector machines, described in sections I B 1 and I B 2 respectively, on the classic Wisconscin breast cancer data set, [2]. In all of the following experiments we found it beneficial to scale the features by subtracting the mean and dividing by the standard deviation.

As mentioned in section I B, the data set in question contains 30 numerically quantifiable features. That said, it is *a priori* not clear which of these features, or linear combinations thereof, are most informative when mapping a particular observation to either represent a benign or malignant tumor.

In fact, we noticed that most of the features are similar to each other which means that a significant amount of their information will be redundant. Tumor traits like its mean radius and mean area are an example for such a redundancy since both should be almost perfectly correlated. In order to test this intuition, we first explored

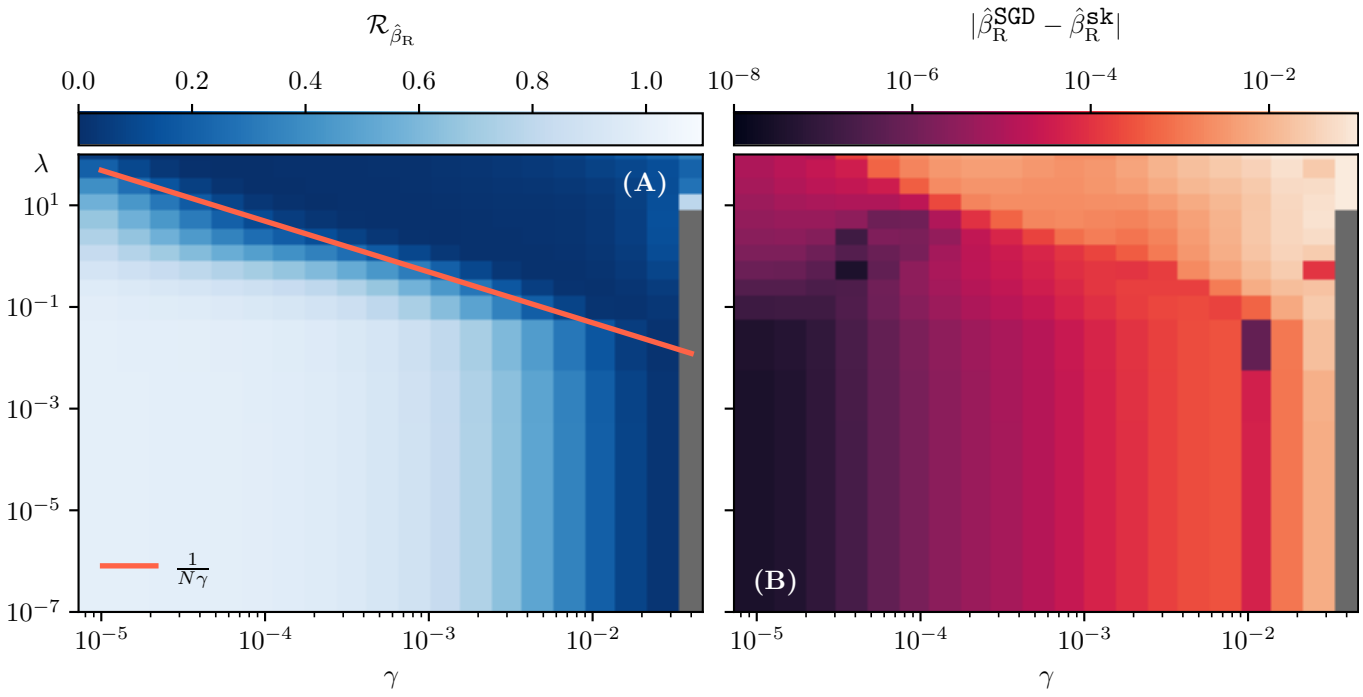

FIG. 25. **(A)**: Relative accuracy of `StochasticGradientDescent` compared to the analytical solution in eq. (12) as a function of regularizer strength λ and learning rate γ. Other hyperparameters fixed to $E = 1000$ and $B = N_\mathcal{T} = N$. Gray cells correspond to diverging descents. Below $\lambda \lessapprox 10^{-1}$ the OLS behavior of Fig. 21 is recovered. Above $\lambda \gtrapprox 10^{-1}$, and $\lambda > \frac{1}{N\gamma}$ in particular (red line), one observes fast convergence due to the diminishing distance between global minimum and origin, serving as initial guess. **(B)**: Absolute deviation of `StochasticGradientDescent` and `SGDRegressor`. The correspondence is reassuring. Note that in the fast convergence region, our implementation actually *outperforms* `SGDRegressor`.

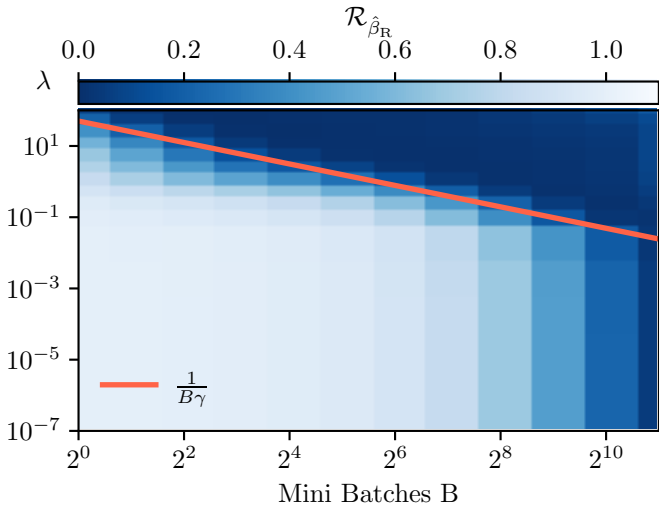

FIG. 26. Accuracy of `StochasticGradientDescent` for Ridge regression as a function of mini-batch number B and L_2-penalty strength λ. In essence, this Figure is a trivial modification of Fig. 25 with $N = N_\mathcal{T} \to B$.

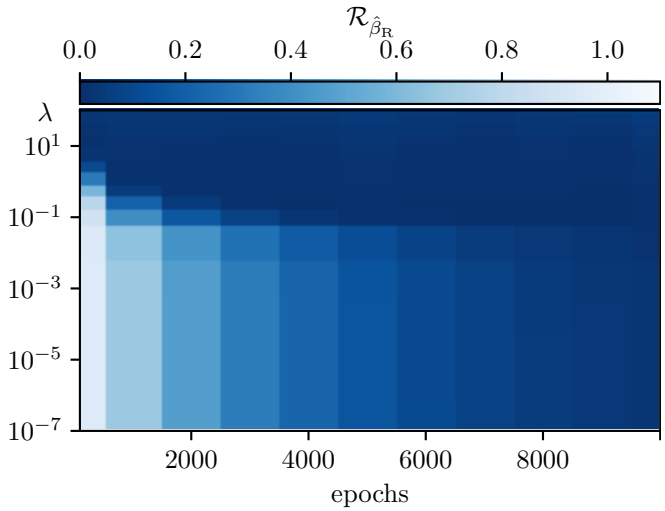

FIG. 27. Accuracy of `StochasticGradientDescent` for Ridge regression as a function of epoch number E and penalty λ. Other hyperparameters set as $\gamma = 0.02$ and $B = 2^8$. A clear separation, in accordance with Fig. 25, between slow OLS convergence and fast Ridge descent is observable.

the correlation matrix, plotted in Fig. 29, of the features present in the Wisconsin breast cancer data set and noticed that in fact quite a few of the features are highly correlated.

The next pressing question was whether only a few linear combinations of these features are needed for good performance on the binary classification task. To check

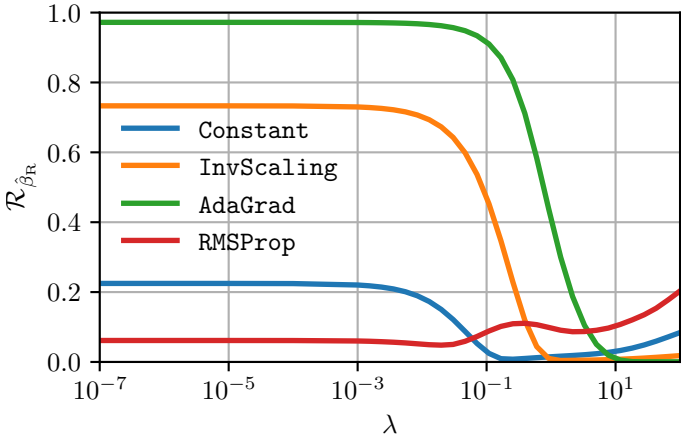

FIG. 28. Comparison of the `Constant`, `InvScaling`, `AdaGrad` and `RMSProp` schedules for Ridge regression as a function of the penalty strength. While quasi-constant strategies (`Constant`, `RMSProp`) outperform decaying schedules (`InvScaling`, `AdaGrad`) for $\gamma < 10^{-1}$, the latter are better suited in the regularizer-dominated regime for $\lambda > 10^{-1}$.

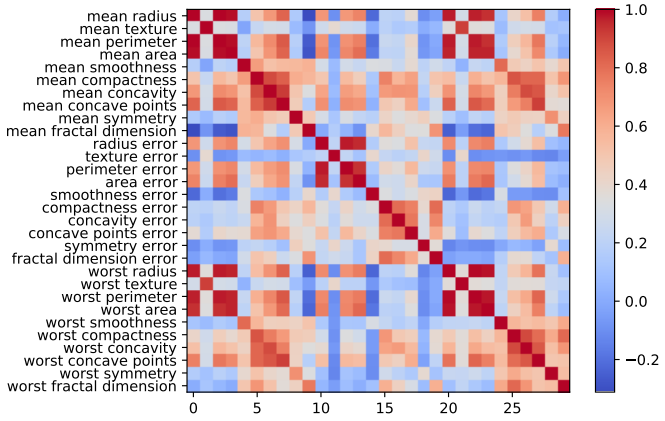

FIG. 29. Correlation matrix of the features in the Wisconsin breast cancer data set.

this, we used `sklearn`'s principal components analysis (PCA) functionality, which projects the data to the eigenvectors of the covariance matrix. In other words, PCA helps one find linear combinations of the features that explain the largest amount of variance in the data.

We found out that, not only are most of the features redundant but almost all of the variance (98%) is explained by the first principal component, shown in Figure 30. This implies that the current classification problem can be treated as finding a separating straight line between the two classes, whose projection is shown in Figure 31. In the light of this observation, we present the performance of all algorithms on both the full feature set as well as on the reduced feature set, defined as the one dimensional linear combination of features suggested by PCA.

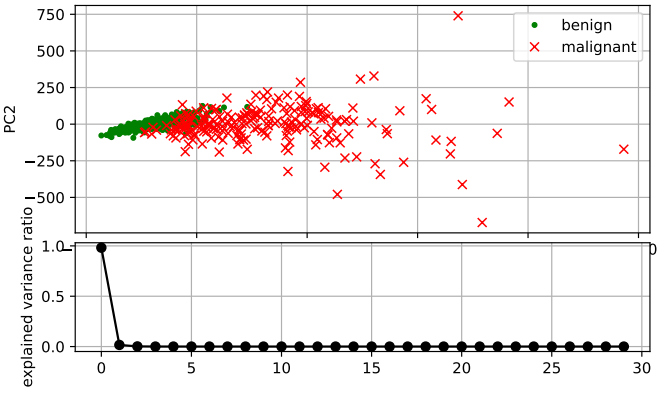

FIG. 30. PCA projection of each sample to the top two principal components (**top**) and the explained variance by each principal component (**bottom**).

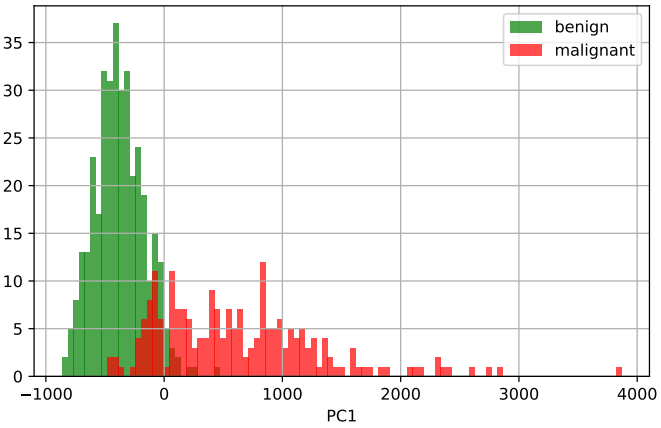

FIG. 31. Histogram of the first principal component values. A clear separation between the benign and malignant samples is visible.

1. *Logistic Regression*

Let us first focus on logistic regression and recall that the nonlinearity of the cross-entropy gradient directly implies the usage of `StochasticGradientDescent` as method for deducing the regression parameters. For this, we settled for a batch size of 1, a maximum epoch number of $E = 1000$ and an absolute loss threshold of $\Delta = 0.001$ for the early convergence check of section IV.

To evaluate the performance of the classifiers, it is sufficient to calculate the fraction of correct classifications. Additionally, we always compare our implementation with `sklearn`'s multipurpose classification capabilities — SGDClassifier — with its loss set to `'log'` (implying a cross-entropy loss).

We explore what happens for different values of the learning rate (ranging from $\gamma = 10^{-6} - 1$) and the regularization value (ranging from $\lambda = 10^{-9} - 1$). The results are shown in Figure 32. In both implementations it is clear that the best performance is achieved for average learning rates (0.001 and 0.01) and there is weak dependence on

penalties.

FIG. 32. The performance of our implementation of the Logistic Regression algorithm (**left**) and `SGDClassifier(loss="log")` (**right**). All of the results were achieved by averaging the outcome of a k-folds validation with 5 folds.

We now repeat the same analysis for the reduced feature set, in which we only using the first principal component. The results shown in Figure 33 paint a clear picture that the solution of the problem is independent of both the choice of learning rate and the size of the penalty. This is probably due to the simplicity of the problem, which just requires the fitting of a straight line that separates the two classes. Nevertheless the performance is quite good (around 91%), which is most likely the result of the low dimensional structure of the extracted features.

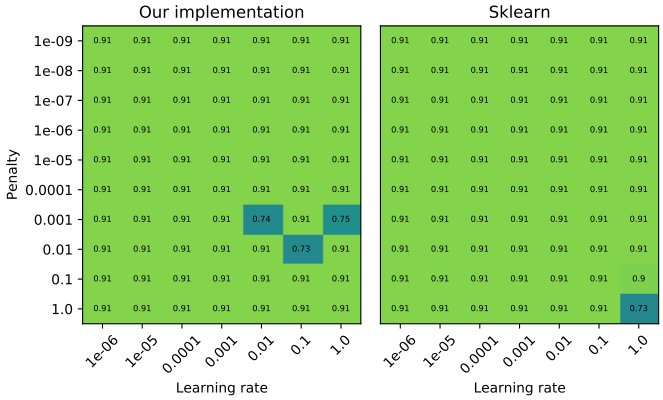

FIG. 33. The performance of our implementation of the Logistic Regression algorithm (**left**) and `SGDClassifier(loss="log")` (**right**) using only the largest principal component. All of the results were achieved by averaging the outcome of a k-folds validation with 5 folds.

2. *Support Vector Machines*

As mentioned in section I B 2, we will study the classification performance of a Support Vector Machine with a radial basis function kernel. In this case, we use the default `sklearn` parameter for γ and run the algorithm over different penalties over the same range. One important difference to keep note of is that in `sklearn`, the penalty for an SVM classifier is defined as $\frac{1}{\lambda}$, so higher values of λ correspond to a weaker penalty. To keep the penalties consistent with the ones used in logistic regression, we give $\frac{1}{\lambda}$ as the argument to the `sklearn` function.

As before, the analysis is carried out for both the full and the reduced feature sets. The results are shown in Figure 34.

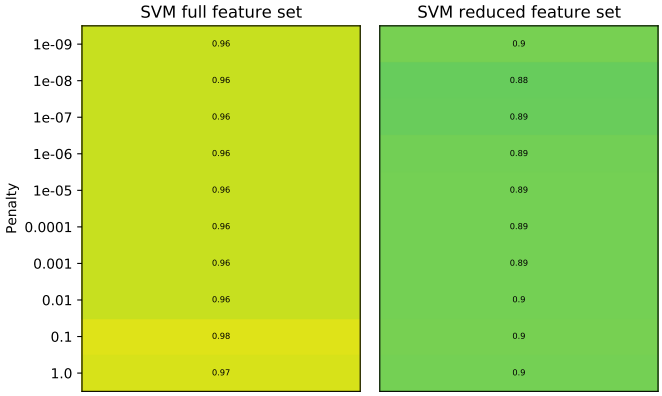

FIG. 34. The performance of the `sklearn` SVM algorithm with a radial basis function kernel for the full feature set (**left**) and the reduced feature set (**right**). All of the results were achieved by averaging the outcome of a k-folds validation with 5 folds.

In general, the SVM performs about as well as both implementations of logistic regression. The performance improves with higher penalties and reaches its optimal value at 0.1. Surprisingly, it failed to outperform logistic regression in the case for the reduced feature set, where it achieved a performance of about 90%. Intuitively, one might expect the opposite to happen as the radial basis function kernel and the kernel trick used in SVMs allow one to find non-linear separating curves between the two classes in higher dimensions. However, this does not happen in our case, although it is possible that the performance might improve for other values of the penalty.

VII. CONCLUSION

The aim of this work was to showcase the capabilities, properties and assessment procedures of foundational supervised learning methods for regression and classification problems.

For linear regression, we compared ordinary least squares, Ridge regression and Lasso regression in the context of polynomial fitting for Franke's function. Elaborate, resampling-based parameter studies allowed us to (i) numerically justify theoretical results such as the bias-variance trade-off for penalized regression methods or the asymptotic behavior of the models prediction error as a

function of regularization strength or data set size and (ii) deduce optimal (hyper)parameters for each regression method. A direct comparison of all three, optimally tuned, regression methods revealed the superiority of the ordinary least squares approach for the "synthetic" problem of approximating Franke's function by means of a linear, polynomial model.

The simplicity of linear regression methods, and in particular the existence of analytical results for their regression parameters, make them an excellent test bed for benchmarking iterative cost function minimization algorithms such as stochastic gradient descent. To this end, we explored the accuracy of `StochasticGradientDescent` — a self-implemented, versatile version of mini-batch gradient descent — in the context of OLS and Ridge regression as a function of its hyperparameter space. If tuned properly, our results suggest excellent qualitative accordance with analytical solutions and `SGDRegressor` — an industry-grade implementation of stochastic gradient descent. More precisely, and not surprisingly, we find `StochasticGradientDescent` to perform best for (i) large learning rates and (ii) large number of mini-batches. Adaptive learning rate policies operate best if they attain a quasi-equilibrium characterized by a large magnitude learning rate. If the penalization strength in the to-be-minimized cost function becomes dominant, fast convergence is observed due to the parameter shrinkage property of Ridge regression.

Finally, we explored the performance of *Logistic Regression* and *Support Vector Machines* for the classification of benign and malignant breast cancers from the Wisconscin breast cancer data set. We found that, for an optimal choice of parameters, our implementation and the widely used `SGDClassifier` and `SVC` classes in `sklearn`, give a very similar classification performance. Additionally we noticed that many of the features in the data set were quite correlated, which lead us to explore their dimensionality with the help of PCA. This analysis led us to the intriguing conclusion that a single linear combination of the features, which explained 98% of the variance, can be used to perform classification with up to 91% accuracy.

Looking back, the authors regard the project as a well-defined and mostly interesting introduction into foundational machine learning techniques. Although somewhat repetitive at times, at least for a research project on a PhD level, the problem phrasing always allowed to take a more theoretical (e.g. deriving the asymptotic behavior of the predictor variances) or practical detour (e.g. profiling `SGDRegressor` and `StochasticGradientDescent`, or embedding the breast cancer data set into a lower dimensional space) and we believe to have found a reasonable balance between what-was-asked vs. what-was-interesting to us.

Workload-wise, we believe the scope of the carried out project is acceptable for three, *equally proficient* students. That said, if an appreciable knowledge gradient is present in the group, the workload tends to be distributed quite unfavorably at the expense of $1-2$ people. We acknowledge this is an intrinsic problem of team work, especially in newly formed teams, and not a property of the project. We also understand that this also provides several valuable learning opportunities for everyone in the group. While we do not have an immediate solution to this problem, we hope that this feedback is still helpful

ACKNOWLEDGMENTS

The authors would like to express their gratitude to Morten Hjorth-Jensen for his organizational flexibility in general and pushing the deadline of this project *twice* in particular. ☺

[1] R. Franke, *A critical comparison of some methods for interpolation of scattered data*, Tech. Rep. (Naval Postgraduate School Monterey CA, 1979).
[2] D. Dua and C. Graff, UCI machine learning repository (2017).
[3] W. N. van Wieringen, Lecture notes on ridge regression (2021), arXiv:1509.09169 [stat.ME].
[4] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning* (Springer New York, 2009).
[5] A. E. Hoerl and R. W. Kennard, Ridge regression: Biased estimation for nonorthogonal problems, Technometrics **42**, 80 (2000).
[6] R. Tibshirani, Regression shrinkage and selection via the lasso, Journal of the Royal Statistical Society: Series B (Methodological) **58**, 267 (1996).
[7] J. Duchi, E. Hazan, and Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, Journal of Machine Learning Research **12**, 2121 (2011).
[8] G. Hinton, Neural Networks for Machine Learning: Lecture 6a Overview of mini-batch gradient descent.
[9] scikit-learn developers, Stochastic gradient descent: Mathematical formulation.