Machine Learning

Supervised Machine Learning – Regression

Part 3: Non-Linear Regression, Hyperparameters, Regularization, Validation

Previously

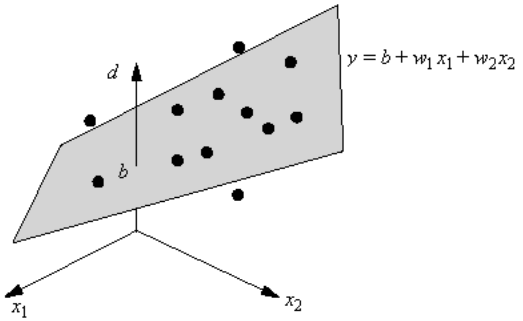
- Non-parametric regression
  - Makes no assumption on function structure
  - Needs a very large number of training data points, especially for complex functions
  - Need to choose correct number of training neighbors for averaging
- Parametric Regression
  - Choose hypothesis
  - Train parameters
    - min MSE (single shot or iterative)
  - Bias/Variance trade-off

Linear Regression

Linear Regression – Multiple Input Features

- Assume input vector with d features $\boldsymbol{x} \in R^d$. Start simple with linear regression.
- Linear regression assume structure:

$$\hat{m}(x, \hat{\boldsymbol{a}}) = \sum_{m=1}^{d} x_m \hat{a}_{m+1} + \hat{a}_1 = \tilde{\boldsymbol{x}}^T \hat{\boldsymbol{a}}$$



where $\tilde{\boldsymbol{x}} = [1, \boldsymbol{x}^T]^T = [1, x_1, x_2, \ldots, x_d]^T$.

Given training dataset $S = \{(x_n, y_n)\}_{n=1}^N$, train $\hat{\boldsymbol{a}}$ to minimize

$$\text{MSE}_{TR}(\hat{\boldsymbol{a}}; S) = \frac{1}{N} \sum_{n=1}^{N} |y_n - \hat{m}(\boldsymbol{x}_n; \hat{\boldsymbol{a}})|^2$$

- $\frac{1}{N}\sum_{n=1}^{N} |y_n - \hat{m}(x_n; \hat{\boldsymbol{a}})|^2 = \frac{1}{N} \|\boldsymbol{y} - \boldsymbol{X}^T \hat{\boldsymbol{a}}\|_2^2$, where $\tilde{\boldsymbol{x}} = [1, \boldsymbol{x}_n^T]^T$ and $\boldsymbol{X} = [\tilde{\boldsymbol{x}}_1, \tilde{\boldsymbol{x}}_2, \ldots, \tilde{\boldsymbol{x}}_N] \in R^{(d+1)\times N}$
- min-MSE = LS problem !
- If $d + 1 \geq N$, train-MSE will be 0 and we can have complete data fitting.
- What if data models is non-linear?

Non-Linear Regression

Polynomial Regression – Single Input Feature

- Considering single-dimensional input $x \in \mathbb{R}$ (i.e., $d = 1$)
- Hypothesis is a M-th degree polynomial:

$$\hat{m}(x, \hat{\boldsymbol{a}}) = \sum_{m=0}^{M} x^m \hat{a}_{m+1}$$

Weiestrass Approximation Theorem:

Suppose f is a continuous real-valued function defined on the real interval $[a, b]$. For every $\varepsilon > 0$, there exists a polynomial p, such that $|f(x) - p(x)| < \varepsilon \; \forall x \in [a, b]$.

Polynomial Regression – Single Input Feature (cont'd)

$$\hat{m}(x, \hat{\boldsymbol{a}}) = p_M(x, \hat{\boldsymbol{a}}) = \sum_{m=0}^{M} x^m \hat{a}_{m+1}$$

M is the polynomial degree. As M increases:

- More flexibility – ability to express more complex functions
- Number of parameters P increase linearly with M (specifically, $P = M + 1$)

Given training dataset $S = \{(x_n, y_n)\}_{n=1}^{N}$, train $\hat{\boldsymbol{a}}$ to minimize:

$$\text{MSE}_{TR}(\hat{\boldsymbol{a}}; S) = \frac{1}{N} \sum_{n=1}^{N} |y_n - \hat{m}(x_n; \hat{\boldsymbol{a}})|^2$$

Polynomial Regression – Single Input Feature (cont'd)

$$\hat{m}(x, \hat{\boldsymbol{a}}) = \sum_{m=0}^{M} x^m \hat{a}_{m+1} = \tilde{\boldsymbol{x}}^T \hat{\boldsymbol{a}}$$

where $\tilde{\boldsymbol{x}} = [1, x, x^2, \ldots, x^M]^T$

$$\text{MSE}_{TR}(\hat{\boldsymbol{a}}; S) = \frac{1}{N} \sum_{n=1}^{N} |y_n - \hat{m}(x_n; \hat{\boldsymbol{a}})|^2 = \frac{1}{N} \sum_{n=1}^{N} \left| y_n - \tilde{\boldsymbol{x}}_n{}^T \hat{\boldsymbol{a}} \right|^2 = \frac{1}{N} \|\boldsymbol{y} - \boldsymbol{X}^T \hat{\boldsymbol{a}}\|_2^2$$

where $\tilde{\boldsymbol{x}}_n = [1, x_n, x_n^2, \ldots, x_n^M]^T$ and $\boldsymbol{X} = [\tilde{\boldsymbol{x}}_1, \tilde{\boldsymbol{x}}_2, \ldots, \tilde{\boldsymbol{x}}_N] \in R^{(M+1)\times N}$

Polynomial Regression – Single Input Feature (cont'd)

Minimize train-MSE:

$$\text{MSE}_{TR}(\hat{\boldsymbol{a}}; S) = \frac{1}{N} \|\boldsymbol{y} - \boldsymbol{X}^T \hat{\boldsymbol{a}}\|_2^2$$

- Assume random training data are "in general position" so that $rank(\boldsymbol{X}^T) = \min(M+1, N)$

  If $M + 1 \geq N$, train-MSE will be 0 and we will have complete data fitting.

- Single-shot LS solution (inversion or SVD)

- Iterative LS solution (GD)

Polynomial Regression – Multiple Input Features

A multivariate polynomial of degree p in d variables is given by

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{k}:\, \mathbf{1}_d^T \boldsymbol{k} \leq p} a(\boldsymbol{k}) \prod_{c=1,\ldots,d} x_c^{k_c}$$

where $p \in \mathbb{N}_+$ is the degree of the polynomial, $d \in \mathbb{N}_+$is the number of input variables (features), and $\boldsymbol{x} = [x_1, x_2, \ldots, x_d]^T$ represents the input features.

A multivariate polynomial of degree p linearly combines all products of powers of the variables in $\boldsymbol{x}$, where the sum of the powers does not exceed the polynomial's degree $\boldsymbol{p}$.

Definitions:

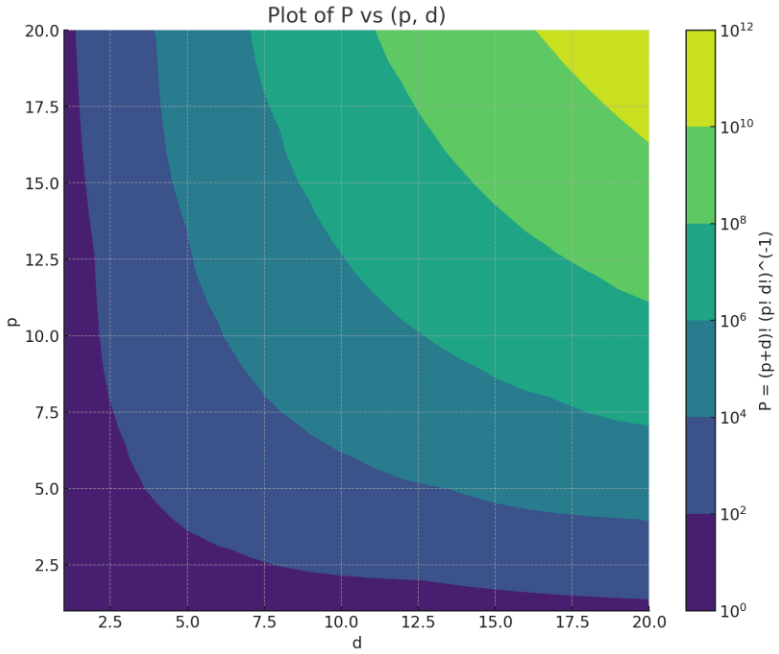
- $\boldsymbol{k} = [k_1, k_2, \ldots, k_d]^T$is a multi-index vector where each k_c represents the power to which the c-th feature x_c is raised
- The sum $\mathbf{1}_d^T \boldsymbol{k} = \sum_{c=1,\ldots,d} k_c$ gives the total degree of each term, which should not exceed p.
- $a(k)$ is the scalar coefficient corresponding to the multi-index k.
- $\prod_{c=1,\ldots,d}$ is the product over c from 1 to d.
- $x_c^{k_c}$ denotes the c-th feature raised to the power k_c.

Polynomial Regression – Multiple Input Features

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{k}:\, \mathbf{1}_d^T \boldsymbol{k} \leq p} a(\boldsymbol{k}) \prod_{c=1,\ldots,d} x_c^{k_c}$$

Denote by P be the number of parameters used.

- P equals the number of terms in the summation.
- Thus, P equals the number of distinct configurations of vector $\boldsymbol{k} \in \mathbb{N}^d$ that satisfy $\mathbf{1}_d^T \boldsymbol{k} \leq p$.



According to the "stars and bars" theorem:

$$P = \binom{d+p}{p} = \frac{(p+d)!}{p!\,d!}$$

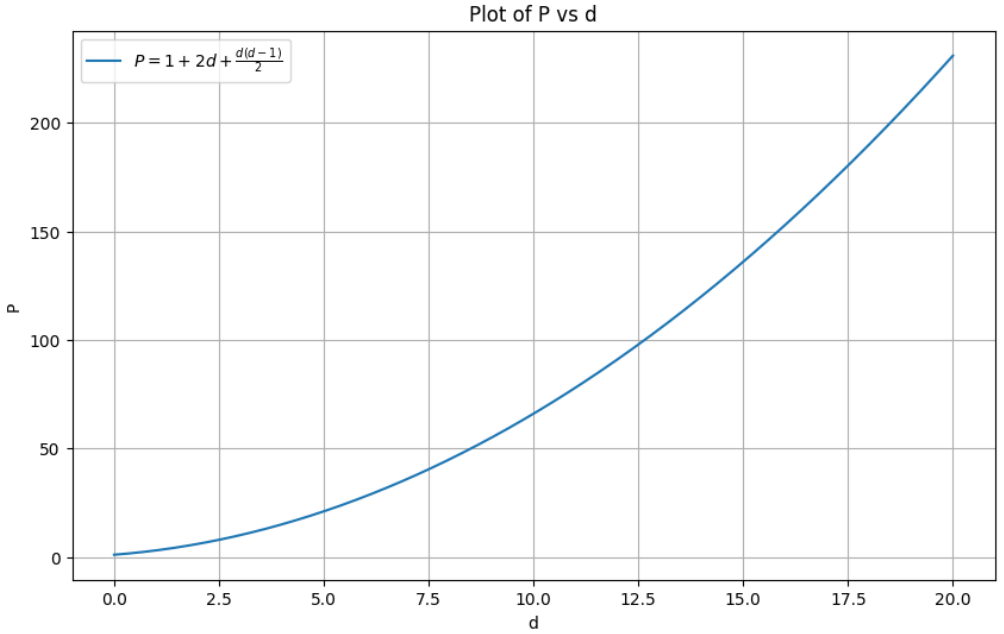
- For general p and d, this grows as $O\left(\frac{e^{p+d}}{\sqrt{pd}}\right)$, exponential in $p+d$ with a minor correcting factor. For fixed p w.r.t. d, this becomes $O(d^p)$, degree-p polynomial in d. For fixed d w.r.t. p, this becomes $O(p^d)$, degree-d polynomial in p.

Special case: Quadratic Regression – Multiple Input Features

- Fixed $p = 2$ and general d

$$P = \binom{d+2}{2} = \frac{(d+2)(d+1)}{2} = 1 + 2d + \frac{d(d-1)}{2} = \frac{1}{2}d^2 + \frac{3}{2}d + 1$$

- The number of parameters P is quadratic in d.



Polynomial Regression – Single Input Feature (cont'd)

$$\hat{m}(\boldsymbol{x}, \hat{\boldsymbol{a}}) = \sum_{m=1}^{P} \hat{a}_m \prod_{c=1,\dots,d} x_c^{k_c^m} = \tilde{\boldsymbol{x}}^T \hat{\boldsymbol{a}}$$

where

- $\tilde{\boldsymbol{x}} = \left[\prod_{c=1,\dots,d} x_c^{k_c^1}, \prod_{c=1,\dots,d} x_c^{k_c^2}, \dots, \prod_{c=1,\dots,d} x_c^{k_c^P}\right]$
- $\boldsymbol{k}^m$ is the the multi-index vector that contains the powers corresponding to each feature in $\boldsymbol{x}$ for the m-th summation term ($m = 1,2, \dots, P$).
- k_c^m is the c-th entry of $\boldsymbol{k}^m$, equal to the power corresponding to x_c for the m-th summation term.

$$\text{MSE}_{TR}(\hat{\boldsymbol{a}}; S) = \frac{1}{N} \sum_{n=1}^{N} |y_n - \hat{m}(\boldsymbol{x}_n; \hat{\boldsymbol{a}})|^2 = \frac{1}{N} \sum_{n=1}^{N} \left| y_n - \tilde{\boldsymbol{x}}_n{}^T \hat{\boldsymbol{a}} \right|^2 = \frac{1}{N} \|\boldsymbol{y} - \boldsymbol{X}^T \hat{\boldsymbol{a}}\|_2^2$$

where $\boldsymbol{X} = [\tilde{\boldsymbol{x}}_1, \tilde{\boldsymbol{x}}_2, \dots, \tilde{\boldsymbol{x}}_N] \in R^{(M+1)\times N}$

Polynomial Regression – Pros and Cons

Pros:

1. Polynomial regression serves as a global approximator. It provides an overall equation that applies to the entire feature space and can capture any function f, for sufficiently large p.
2. It offers a closed-form equation that provides insight into the relationships between variables

Cons:

1. As the number of features d and the polynomial degree p grow, the number of parameters P grows exponentially.

- For $N > P$, the exact solution of regression is low-bounded by $O(P^3)$. If P grows exponentially, so does the complexity of solving regression.
- Storage of parameters grows exponentially.
- Number of raining data points needed to avoid overfitting also grows exponentially.

General Non-Linear Regression – Basis Functions

- Polynomial regression is one kind of non-linear regression with the pros and cons discussed before.
- A general non-linear data model is of the form

$$f(x) = \sum_{m=1}^{P} \phi_m(\boldsymbol{x})\, a_m$$

- Where $\phi(\boldsymbol{x})$: $R^d \rightarrow R$ is function on $\boldsymbol{x}$ that can be linear or any non-linear.
- This model needs P basis functions and P parameters.
- Accordingly, the hypothesis model can be of the form:

$$\hat{m}(\boldsymbol{x}; \hat{\boldsymbol{a}}) = \sum_{m=1}^{P} \phi_m(\boldsymbol{x})\, \hat{a}_m$$

General Non-Linear Regression – Basis Functions (cont'd)

$$\hat{m}(\boldsymbol{x};\hat{\boldsymbol{a}}) = \sum_{m=1}^{M} \phi_m(\boldsymbol{x})\,\hat{a}_m = \boldsymbol{b}(\boldsymbol{x})^T\hat{\boldsymbol{a}}$$

- where $\boldsymbol{h}(\boldsymbol{x}) = [\phi_1(\boldsymbol{x}),\ \phi_2(\boldsymbol{x}), \dots, \phi_M(\boldsymbol{x})]^T$.

Train $\hat{\boldsymbol{a}}$ to minimize $\mathrm{MSE}_{TR}(\hat{\boldsymbol{a}}; S) = \frac{1}{N}\|\boldsymbol{y} - \boldsymbol{H}^T\hat{\boldsymbol{a}}\|_2^2$ where $\boldsymbol{H} = [\boldsymbol{h}(\boldsymbol{x}_1), \boldsymbol{h}(\boldsymbol{x}_2), \dots, \boldsymbol{h}(\boldsymbol{x}_N)] \in R^{M\times N}$.

That is:

$$\hat{\boldsymbol{a}} = \mathrm{argmin}_{\boldsymbol{w}\in H}\ \|\boldsymbol{y} - \boldsymbol{H}^T\boldsymbol{w}\|_2^2$$

- If P $\geq N$, (more parameters than data points) train-MSE will be 0 (complete data fitting).
- Solution can be given by SVD of the *design* matrix $\boldsymbol{H}$ or GD since LS is convex problem.

Special case: Quadratic Regression – 2 Input Features

- Generic non-linear form:

$$\hat{m}(\boldsymbol{x};\hat{\boldsymbol{a}}) = \sum_{m=1}^{P} \phi_m(\boldsymbol{x})\,\hat{a}_m$$

- Let $d = 2$ and $p = 2$ and:

$$\phi_1(\boldsymbol{x}) = 1, \phi_2(\boldsymbol{x}) = x_1^2, \phi_3(\boldsymbol{x}) = x_2^2, \phi_4(\boldsymbol{x}) = x_1x_2, \phi_5(\boldsymbol{x}) = x_1, \phi_6(\boldsymbol{x}) = x_2$$

$$\hat{m}(\boldsymbol{x};\hat{\boldsymbol{a}}) = \hat{a}_1 + x_1^2\hat{a}_2 + x_2^2\hat{a}_3 + x_1\,x_2\hat{a}_4 + x_1\hat{a}_5 + x_2\hat{a}_6$$

- $P = 6$ parameters

Other Basis Functions:

- Generic non-linear form: $\hat{m}(\boldsymbol{x}; \hat{\boldsymbol{a}}) = \sum_{m=1}^{P} \phi_m(\boldsymbol{x})\, \hat{a}_m$
- Linear regression: set $P = d + 1$, $\phi_1(\boldsymbol{x}) = 1$, and for all $m \in \{2, \dots, d+1\}$

$$\phi_m(\boldsymbol{x}) = x_{m-1} = [I_d]_{:,m-1}^T \boldsymbol{x}$$

- Single-feature polynomial regression: fix $c \in \{1, \dots, d\}$ and for all $m \in \{1, 2, \dots, P\}$ set

$$\phi_m(\boldsymbol{x}) = x_c^{m-1}$$

- p-degree polynomial regression on all d entries of $\boldsymbol{x}$: set $P = \binom{d+p}{p}$ and for all $m \in \{1,2, \dots, P\}$ define $\boldsymbol{k}^m \in \mathbb{N}^d$ as a distinct vector for which $\mathbf{1}_d^T \boldsymbol{k}^m \leq p$ and set

$$\phi_m(\boldsymbol{x}) = \prod_{c=1,\dots,d} x_c^{k_c^m}$$

- Fourier basis functions: Choose $\{f_m\}_{m=1}^P$ and for all $m \in \{1, 2, \dots, P\}$, set $\phi_m(\boldsymbol{x}) = \cos(2\pi\, f_m x_m)$.

Spline Basis Functions

- Spline Basis Functions: for all $m \in \{1,2,..,P\}$ set $\phi_m(\boldsymbol{x})$ to be a piecewise polynomial basis function (spline).
- E.g., for $P = 2$ and d, splines could be:
  - $\phi_1(\boldsymbol{x}) = \{1\ if\ x_1 < 1,\ \{1 - 3(x_1 - 1)\text{^}2 + 2(x_1 - 1)^3\ if\ 1 \leq x_1 < 2,\ \{0\ if\ x_1 \geq 2$
  - $\phi_2(\boldsymbol{x}) = \{0\ if\ x_2 < 1,\ \{3(x_2 - 1)\text{^}2 - 2(x_2 - 1)^3\ if\ 1 \leq x_2 < 2,\ \{1\ if\ x_2 \geq 2$

- Pros:
  - Controlled Shapes: They can model complex shapes with fewer parameters than higher-degree polynomials, reducing overfitting.
  - Changing a part of the spline typically does not affect the entire curve, which is a key advantage for localized adjustments.

Radial Basis Functions

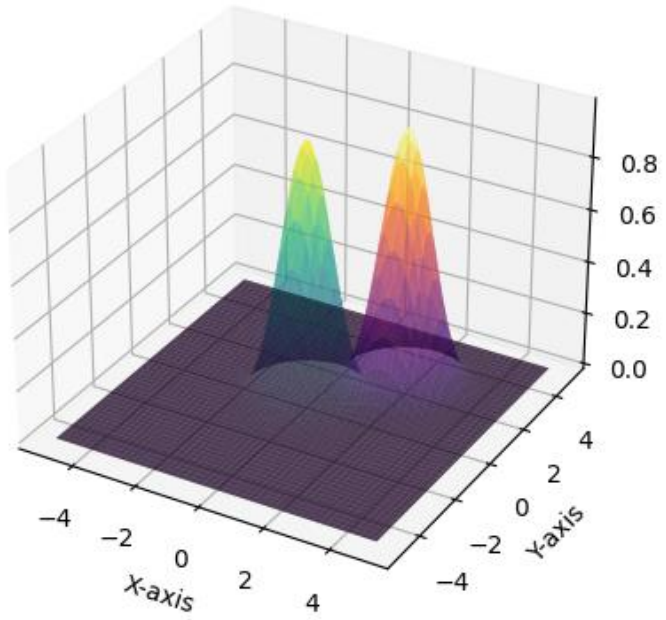
Radial Basis Functions (RBFs) used as basis functions in various machine learning algorithms.

- Hypothesis: $\hat{m}(\boldsymbol{x}; \hat{\boldsymbol{a}}) = \sum_{m=1}^{P} \phi_m(\boldsymbol{x})\, \hat{a}_m$
- **General RBFs**: $\phi_m(\boldsymbol{x}) = \psi(\|\boldsymbol{x} - \boldsymbol{c}_m\|)$, where ψ is some monotonically decreasing function (i.e., for all $a > b, \psi(a) \leq \psi(b)$)
- **Gaussian RBFs**: RBFs with $\psi_m(r) = \exp(-\lambda_m r^2)$. That is, $\phi_m(\boldsymbol{x}) = \exp(-\lambda_m \|\boldsymbol{x} - \boldsymbol{c}_m\|^2)$.
- By increasing M and tuning $\{(\boldsymbol{c}_m, \lambda_m)\}_{m=1}^{M}$ you tune the flexibility of the hypothesis.

Radial Basis Functions (RBFs)

Pros:

- High Flexibility: Can model complex relationships.
- Universality: Capable of approximating any continuous function, given enough basis functions.



Other RBFs

- Multi-quadratic:

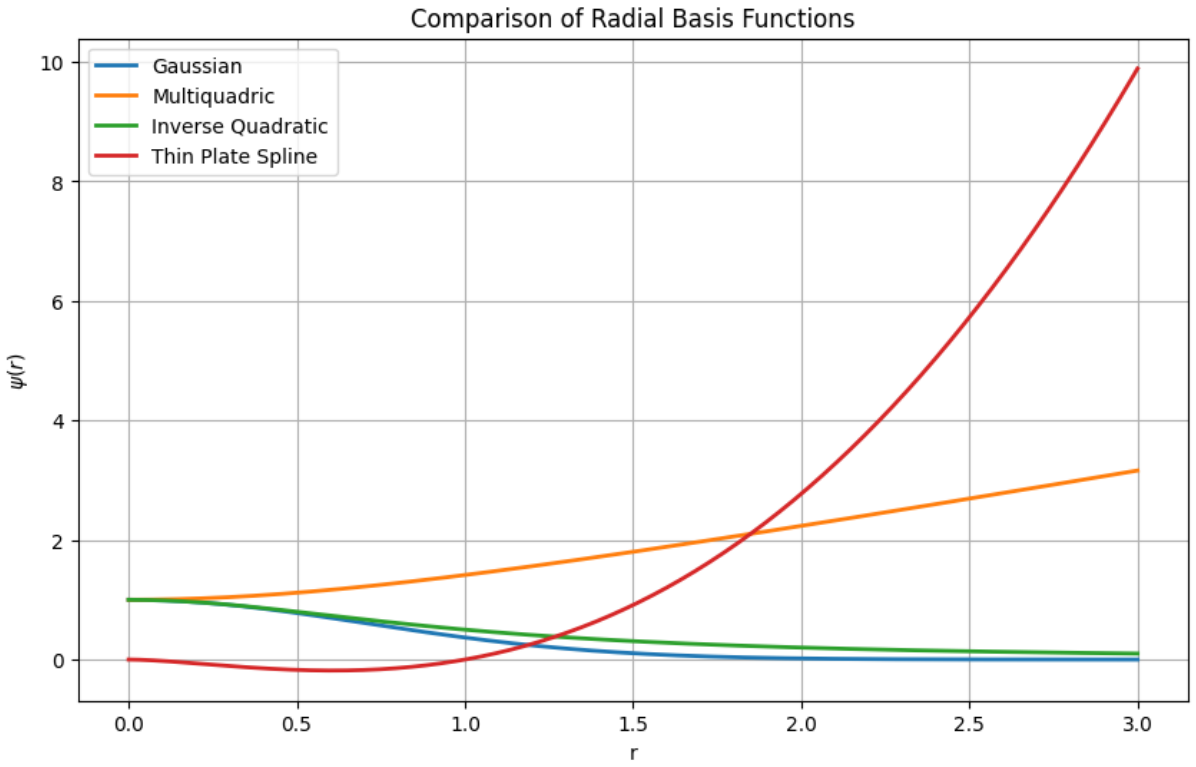
$$\psi_m(r) = \sqrt{1 + (\lambda_m r)^2}$$

- Inverse-quadratic:

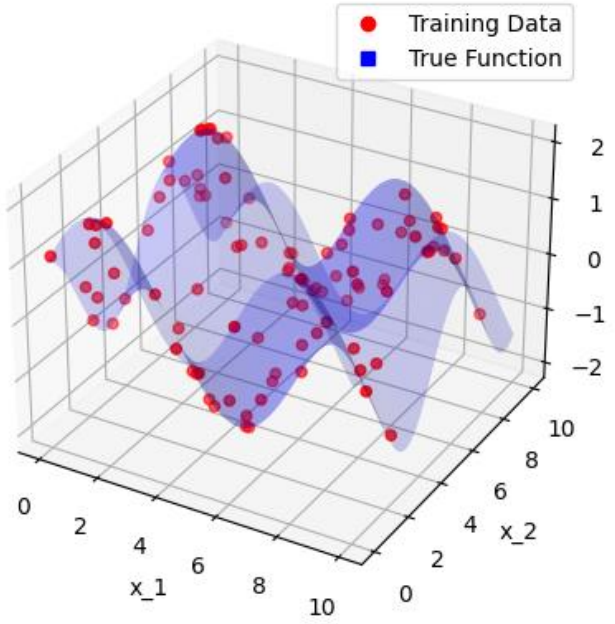
$$\psi_m(r) = \frac{1}{1 + (\lambda_m r)^2}$$

- Think-plate Spline:

$$\psi(r) = r^2 \log(r)$$



Gaussian RBFs – Example

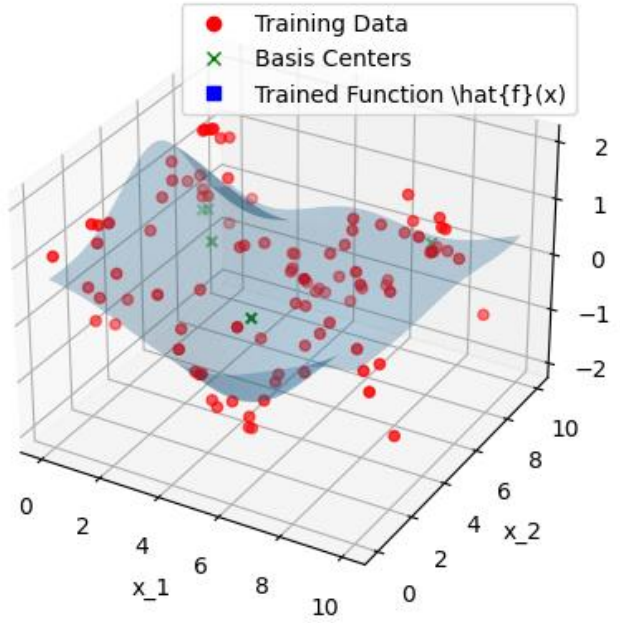


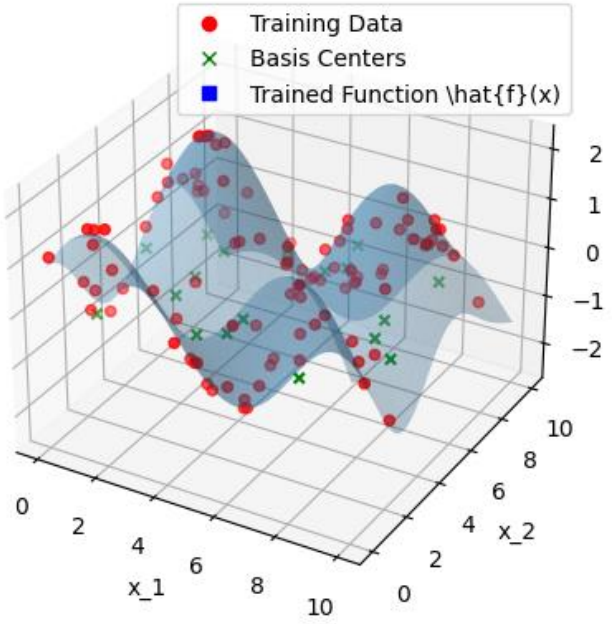
True function:

$$f(\boldsymbol{x}) = \sin(x_1) + \cos(x_2)$$

Training model w/ 5 GRBFs:

(arbitrary centers, $\lambda = 5$)





Training model w/ 20 GRBFs:

(arbitrary centers, $\lambda = 5$)

Play with code on Google Colab

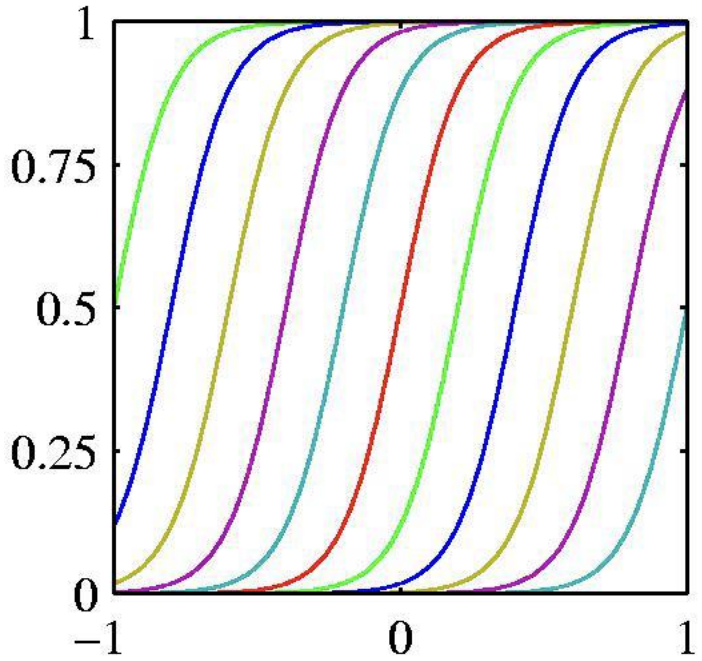
Sigmoidal Basis Functions

- Each basis has the same single sigmoidal mother function

$$\phi_m(x) = \sigma\left(\frac{x - c_m}{s_m}\right)$$

where

$$\sigma(h) = \frac{1}{1 + \exp(-h)}$$



- Instead of $\sigma(h)$ can also use $tanh(h) = 2\,\sigma(a) - 1$.
- Pros:
  - Suitable for problems requiring smooth transitions.
  - Easily adjusted via parameterization.

Hyper-Parameters

- Hyper-parameters are model parameters that are **not to be trained**.
- Selection of hyper-parameters is often problem-specific and can be guided, e.g., by **unsupervised-ML on the data**, **cross-validation**, **domain knowledge**, or **Bayesian optimization methods.**

E.g., in the case of GRBFs, hyper-parameters are M and $\{(\boldsymbol{c}_m, \lambda_m)\}_{m=1}^{M}$.

Optimizing Non-Linear Regression

Train $\hat{\boldsymbol{a}}$:

$$\hat{\boldsymbol{a}} = \text{argmin}_{\boldsymbol{w} \in H} \|\boldsymbol{y} - \boldsymbol{H}^T \boldsymbol{w}\|_2^2$$

where $\boldsymbol{H} = [\boldsymbol{h}(\boldsymbol{x}_1), \boldsymbol{h}(\boldsymbol{x}_2), \dots, \boldsymbol{h}(\boldsymbol{x}_N)] \in R^{P \times N}$ and $\boldsymbol{h}(\boldsymbol{x}) = [\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), \dots, \phi_P(\boldsymbol{x})]^T$.

- To increase flexibility, we increase P.
- If $P \geq N$, train-MSE can be 0 (complete fitting) attained by $\hat{\boldsymbol{a}} = \boldsymbol{H}(\boldsymbol{H}^T \boldsymbol{H})^{-1} \boldsymbol{y}$
  - **Overall cost** $O(PN^2)$: $\boldsymbol{H}^T \boldsymbol{H}$ costs $O(N^2 P)$; $(\boldsymbol{H}^T \boldsymbol{H})^{-1}$ costs $O(N^3)$; $\boldsymbol{H}(\boldsymbol{H}^T \boldsymbol{H})^{-1} \boldsymbol{y}$ costs $O(PN^2)$
- If $P < N$, train-MSE will be non-zero; minimum attained by $\hat{\boldsymbol{a}} = \left(\boldsymbol{H}\boldsymbol{H}^{\boldsymbol{T}}\right)^{-1} \boldsymbol{H}\boldsymbol{y}$

(assuming that the columns of $\boldsymbol{H}$ are in general position; all sets of up to N columns are linearly independent)

  - **Overall cost** $O(P^2 N)$: $\boldsymbol{H}\boldsymbol{H}^{\boldsymbol{T}}$ costs $O(P^2 N)$; $\left(\boldsymbol{H}\boldsymbol{H}^{\boldsymbol{T}}\right)^{-1}$ costs $O(P^3)$; $\left(\boldsymbol{H}\boldsymbol{H}^{\boldsymbol{T}}\right)^{-1} \boldsymbol{H}\boldsymbol{y}$ costs $O(NP^2)$
- SVD cost is $O(PN \min(P, N))$
- GD cost is $O(NPT)$ where T is the number of iterations.

Regularization

Optimistic: By minimizing train-MSE we will get low test-MSE. Solve:

$$\min_{\boldsymbol{w}\in H} \|\boldsymbol{y} - \boldsymbol{H}^T\boldsymbol{w}\|_2^2$$

Getting more cautious, we "regularize" and solve instead:

$$\min_{\boldsymbol{w}\in\mathbb{R}^{M+1}} \|\boldsymbol{y} - \boldsymbol{H}^T\boldsymbol{w}\|_2^2 + \xi \|\boldsymbol{w}\|_2^2$$

- This is known as "ridge regression".
- By limiting the norm of the parameters, it restricts fitting to training data. Introduces bias.
- $\xi \|\boldsymbol{w}\|^2$ is known as "shrinkage" in statistics or weigh—decay in neural networks
- Regression intensity depends on ξ. Too high, makes training data irrelevant. Too low, back to optimistic case.
- Still convex problem. Can solve in closed form (setting gradient to zero) or by gradient descent iterations.

Regularization (cont'd)

Norm-1 regularization:

$$\min_{\boldsymbol{w}\in\mathbb{R}^{M+1}} \|\boldsymbol{y}-\boldsymbol{H}^T\boldsymbol{w}\|_2^2 + \xi\|\boldsymbol{w}\|_1$$

- This is known as "LASSO regression"
- Minimizing the 1-norm promotes sparsity in argument
- Thus, many of the resulting parameters will have very low values
  - Very low values (e.g., with absolute value under $c \times \|\boldsymbol{w}\|_\infty$) and can be clipped (replaced by 0)
  - Accordingly, the corresponding basis functions can be removed
- LASSO regression allows for reducing number of RBFS or number of parameters, at the expense of fitting data, by just tuning ξ

Regularization Examples

Data model:

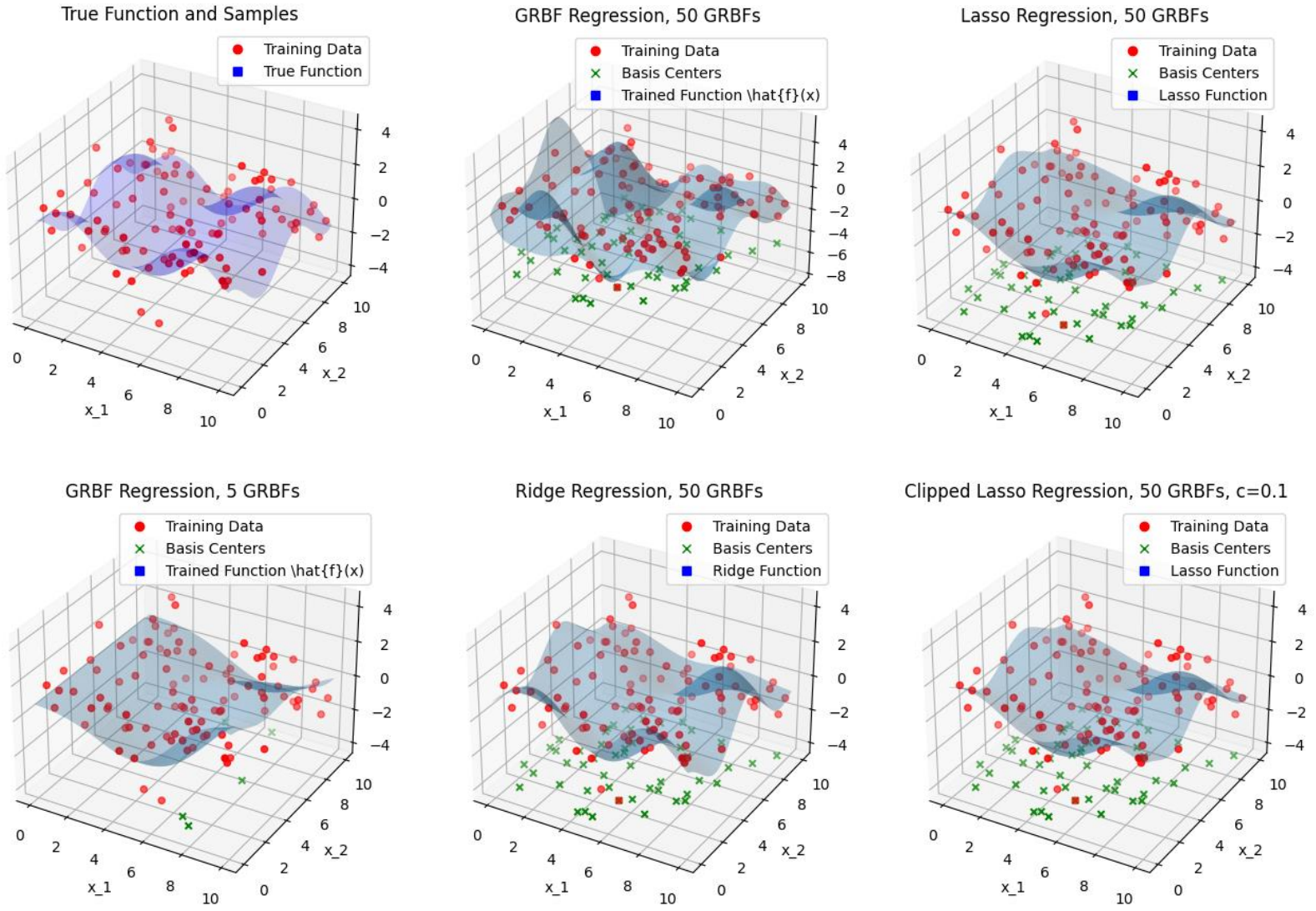
- $f(\boldsymbol{x}) = \sin(x_1) + \cos(x_2)$
- $y(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon$
- $\sigma_\epsilon = 1.2$

Training specs:

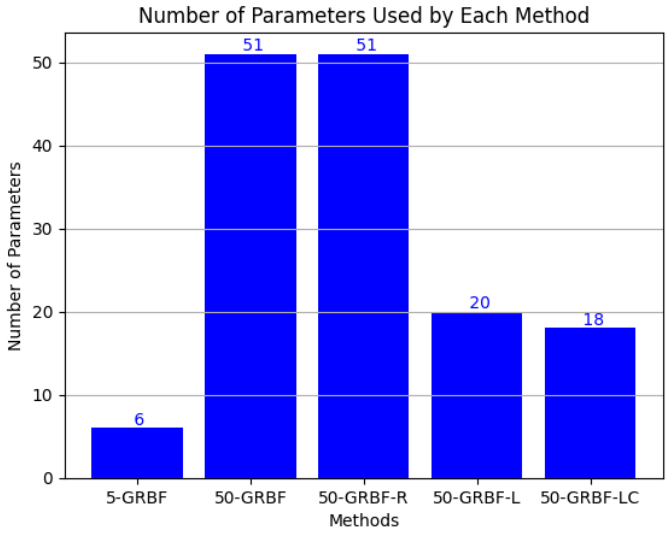
- Training data: $N = 100$
- $\xi = 10^{-3}$
- $M = 5$ and $M = 50$ GRBFs
  - $\lambda = 0.1$
  - Arbitrary centers
  - For clipped lasso, $c = 0.1$

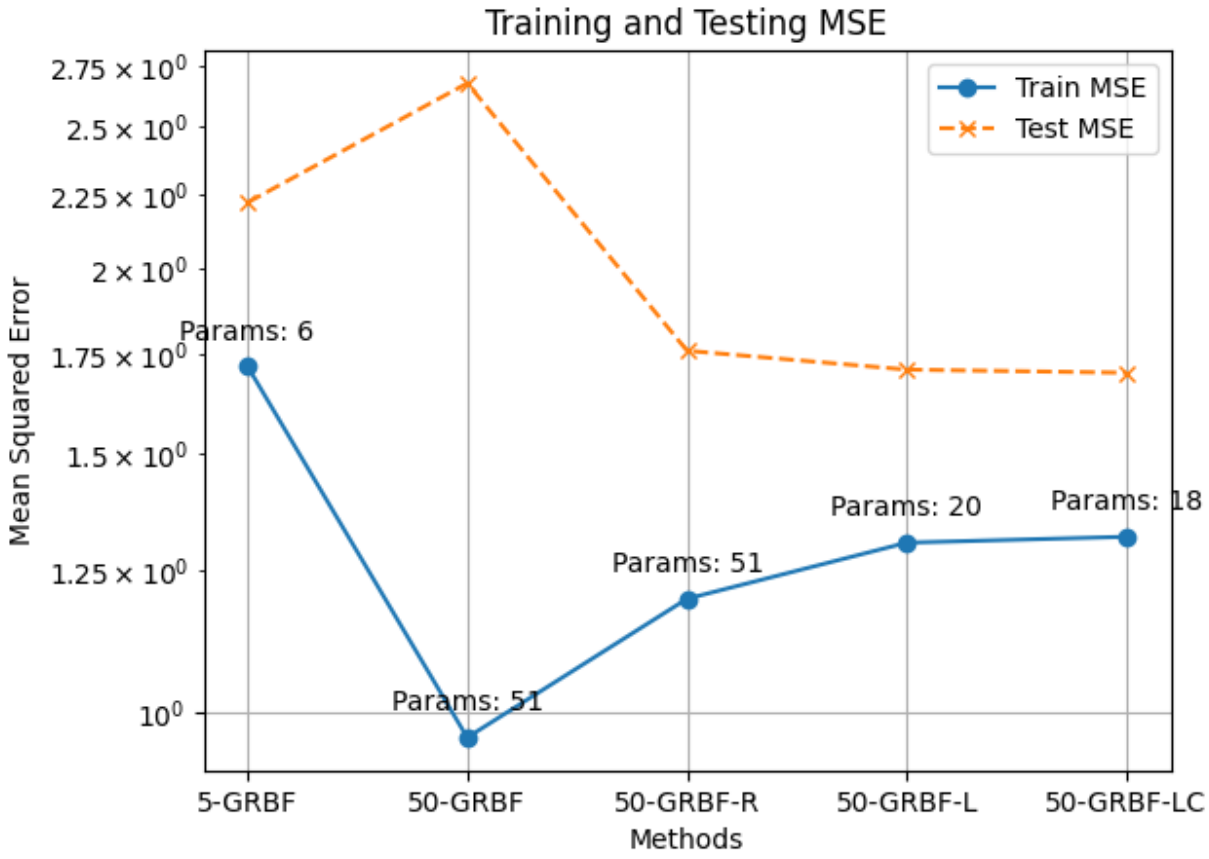
Testing specs:

Testing data: $N_T = 10000$



Regularization Examples

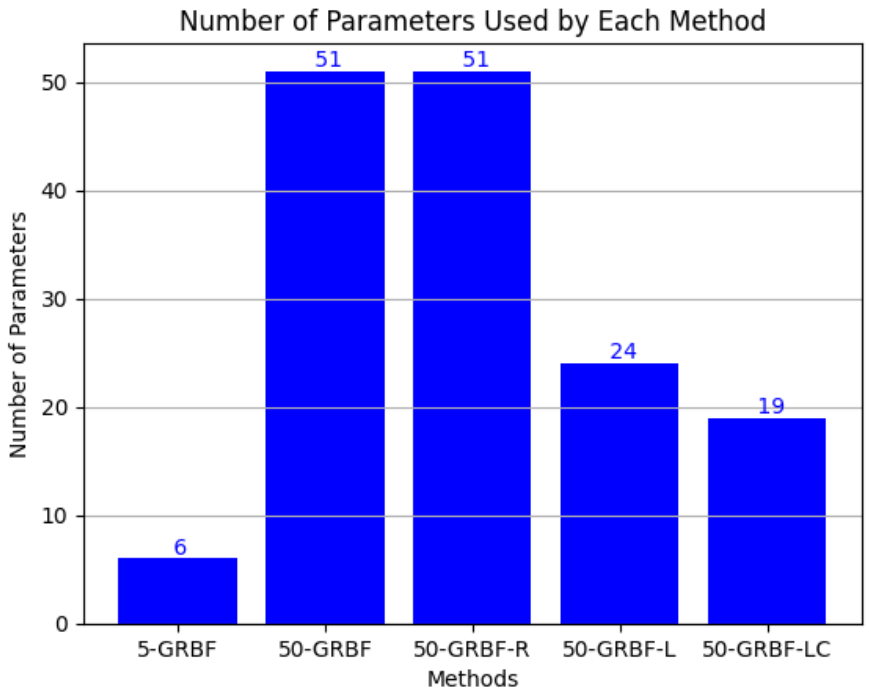


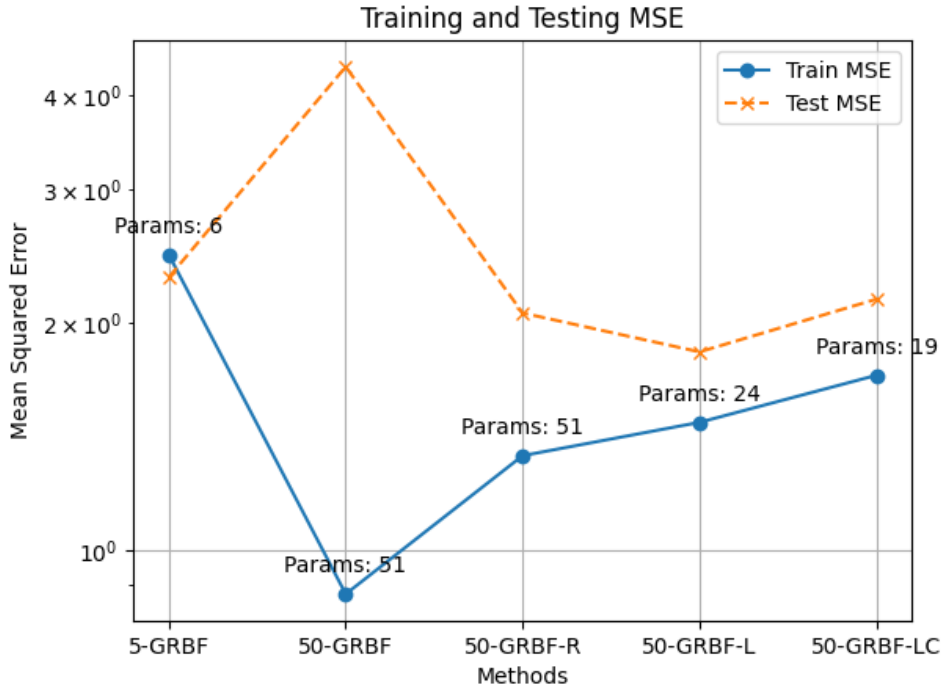


Regularization Examples

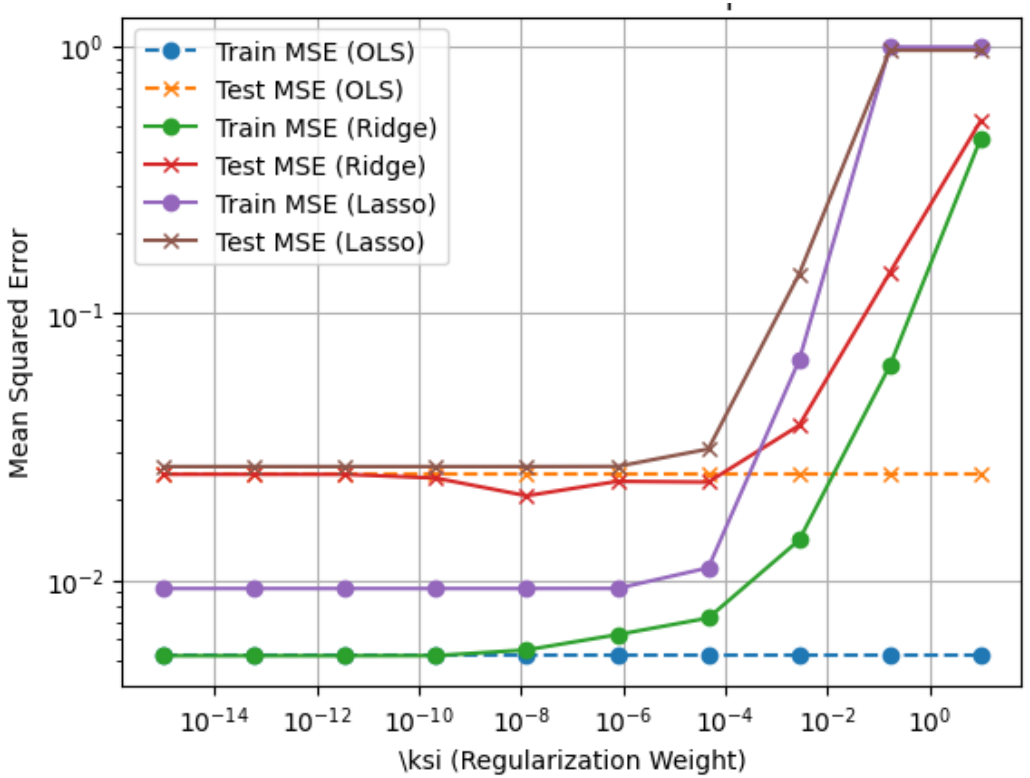
Data model:

- $f(\boldsymbol{x}) = \sin(x_1) + \cos(x_2)\,;\; y(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon;\; \sigma_\epsilon \;= 1.2$
- 2 outliers (data points w/ $\times$ 10 deviation)

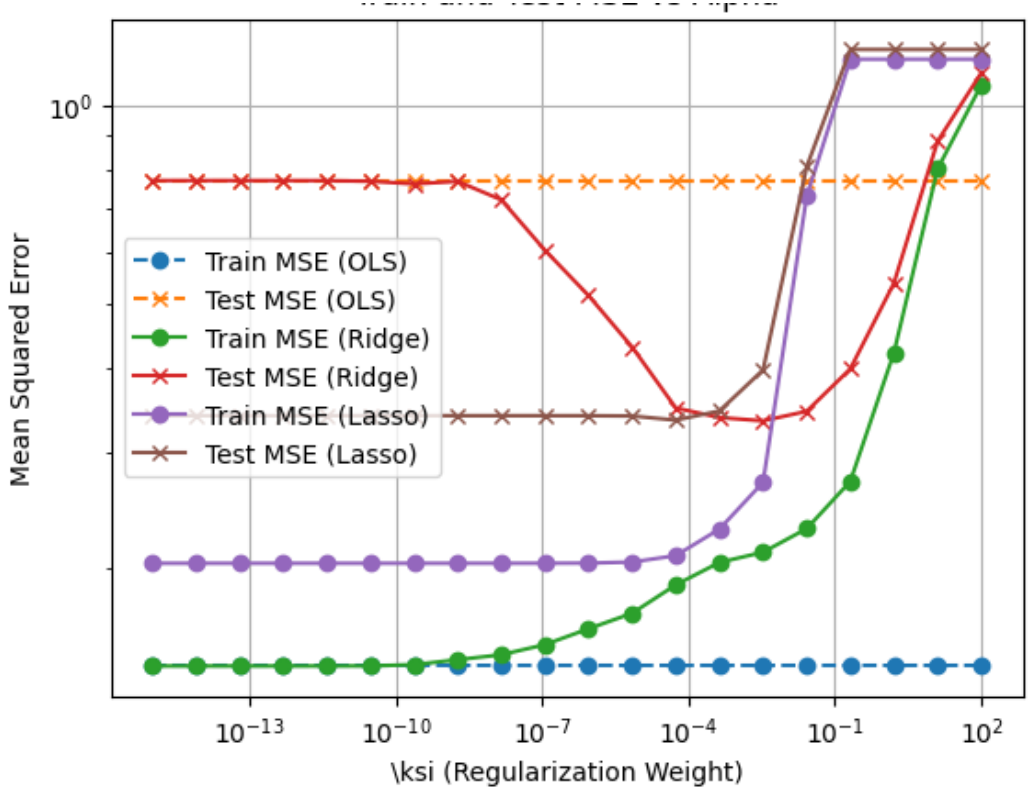




Regularization Examples



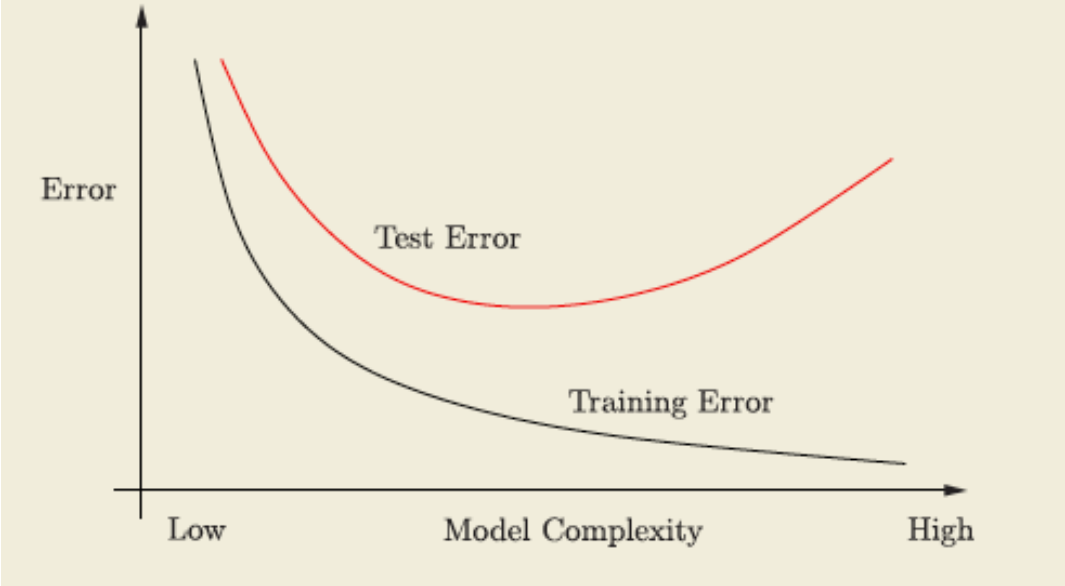
$\sigma_\epsilon = 0.1$



$\sigma_\epsilon = 0.5$

Validation

- Quantify / predict the performance of the designed model
- Assess performance of hyper-parameters such as M and λ
- Cannot do that against the training data which the model was designed to fit
  - E.g., an overfitting model (= poor model) can attain 0 MSE on the training data
- We need a different dataset that did not participate to training



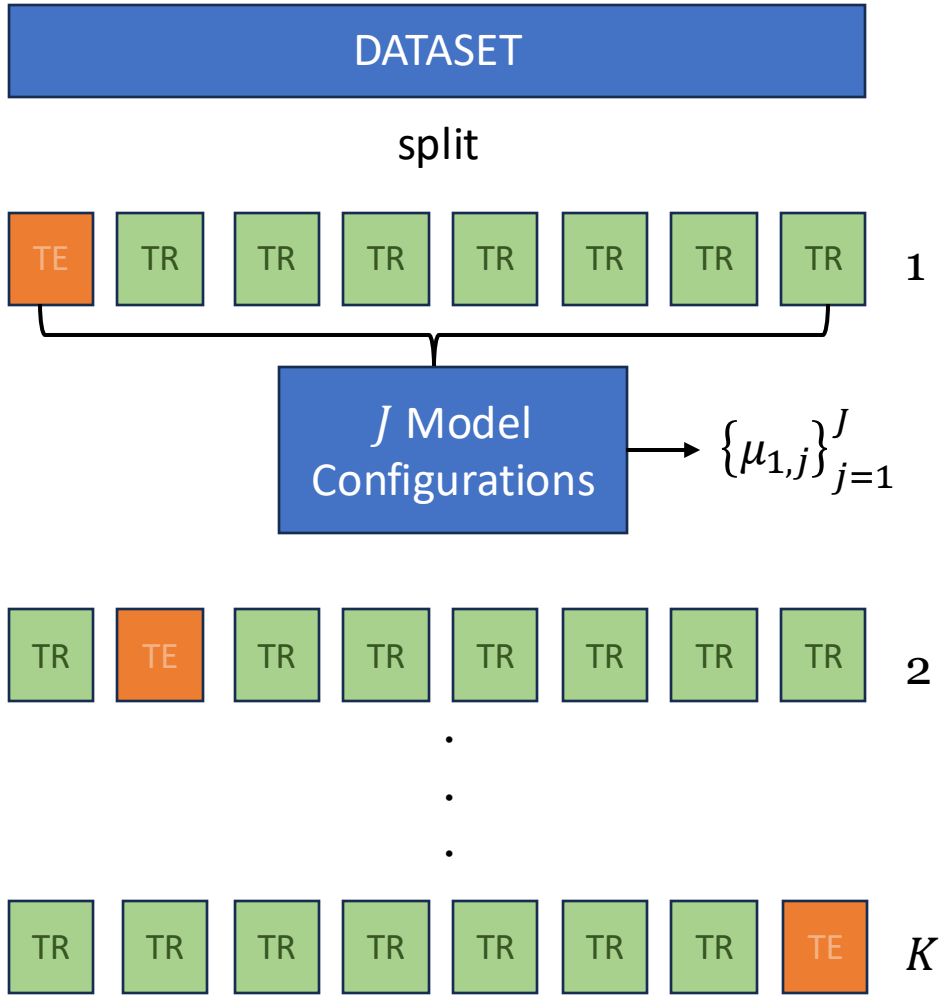
Python Companion

- Please experiment with the python companion found here:

https://colab.research.google.com/drive/1qHMuWTOPsXFiY4iBtzQhTEklyjEG4Tdu

K-Fold Cross-Validation

- Cross-validation is a very common technique that is usually employed in practice.
- We define J configurations of the hyper-parameters of interest (e.g., $(\lambda_1, M_1), \ldots, (\lambda_J, M_J)$)
- Split dataset into, say K, roughly equal-sized, partitions
- We repeat training K times, each time we train on $K - 1$ parts and test on the remaining, for all the hyper-parameter configurations
- Measure $\mu_{k,j}$ s the test-MSE attained for the j-th parameter configuration when tested on the k-th partition



K-Fold Cross-Validation

- At the end, choose j by evaluating $\left\{\left\{\mu_{k,j}\right\}_{k=1}^{K}\right\}_{j=1}^{J}$.
- For example, choose j that minimizes $\frac{1}{K}\sum_{k=1}^{K}\mu_{k,j}$
- We can repeat this on multiple independent data splits
- For $K = N$, we have the extreme case of Leave-One-Out (LOO) cross-validation

- The dataset split can also be used for limiting the overfitting (train K distinct models and average the parameters)