Università degli Studi di Padova

Machine Learning 2024/2025

AIMCO — Artificial Intelligence, Machine Learning and Control Research Group

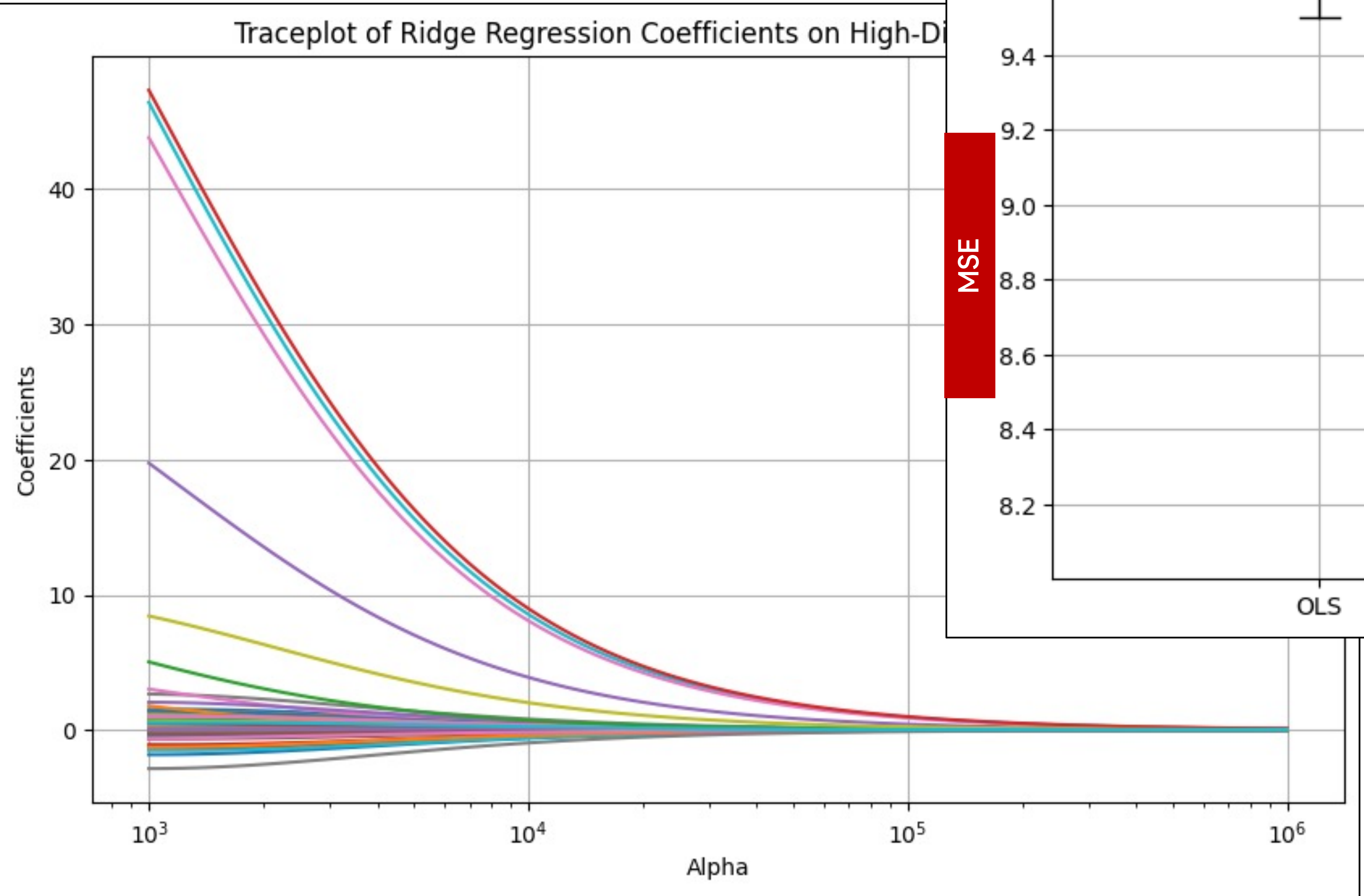
Lecture #10 LASSO & Gradient Descent

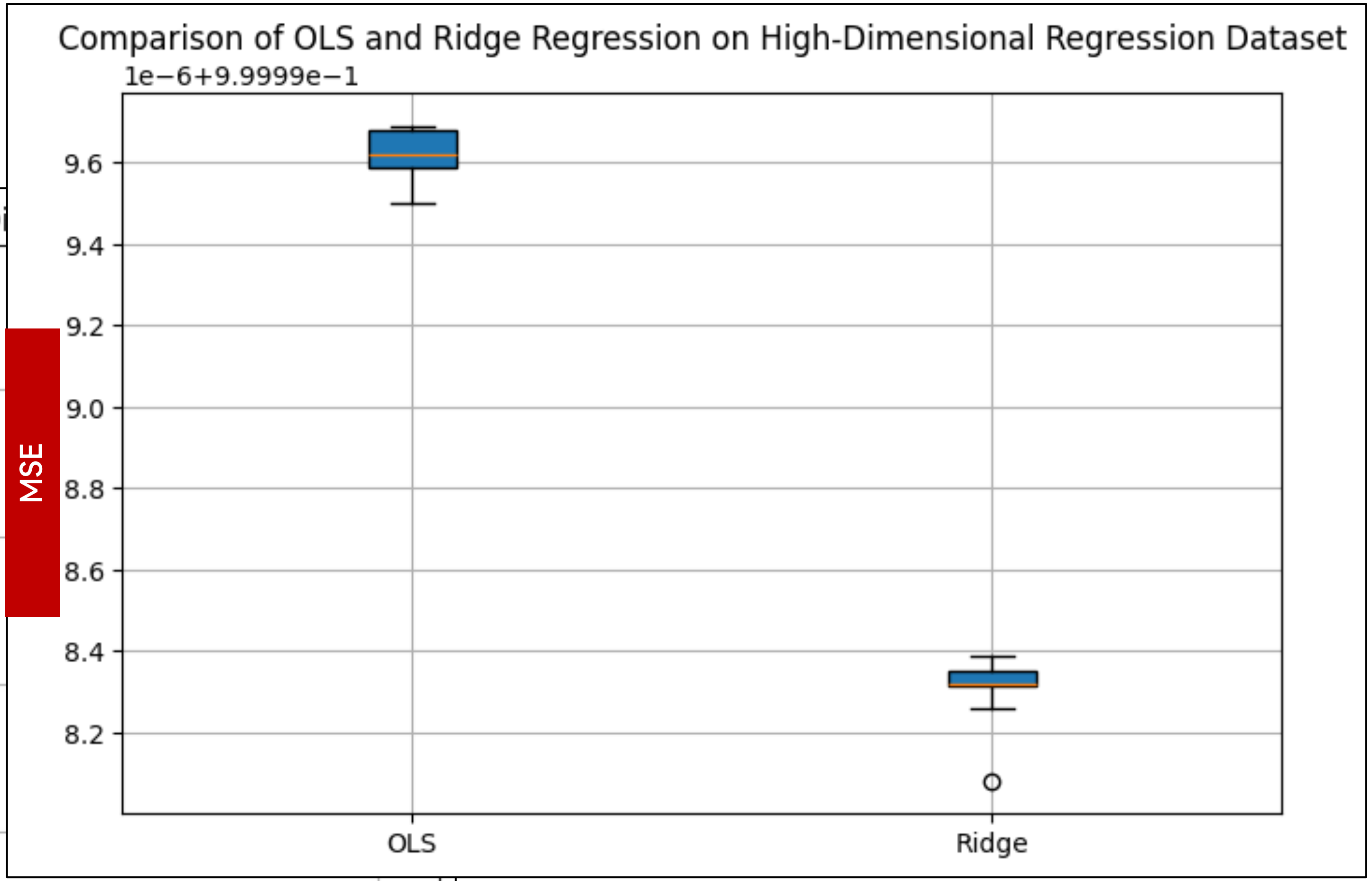
Gian Antonio Susto

Before starting

We will keep future labs @ Da Room

From last lecture: error



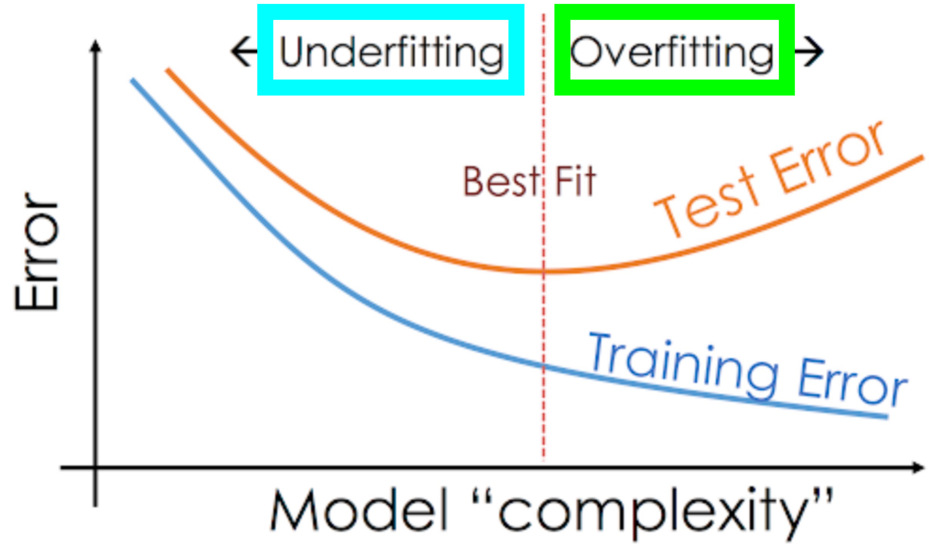


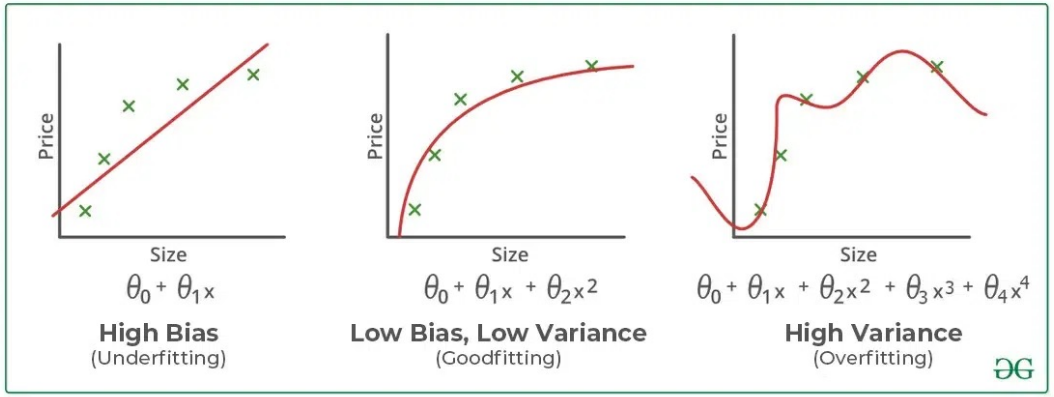
Recap: Bias & Variance / Under & Over-fitting

A complicated model is not always 'optimal':

- Overly complex models tend to 'overfit', meaning they fail to 'generalize.' This results in high variance, where the model captures noise rather than true patterns in the data.

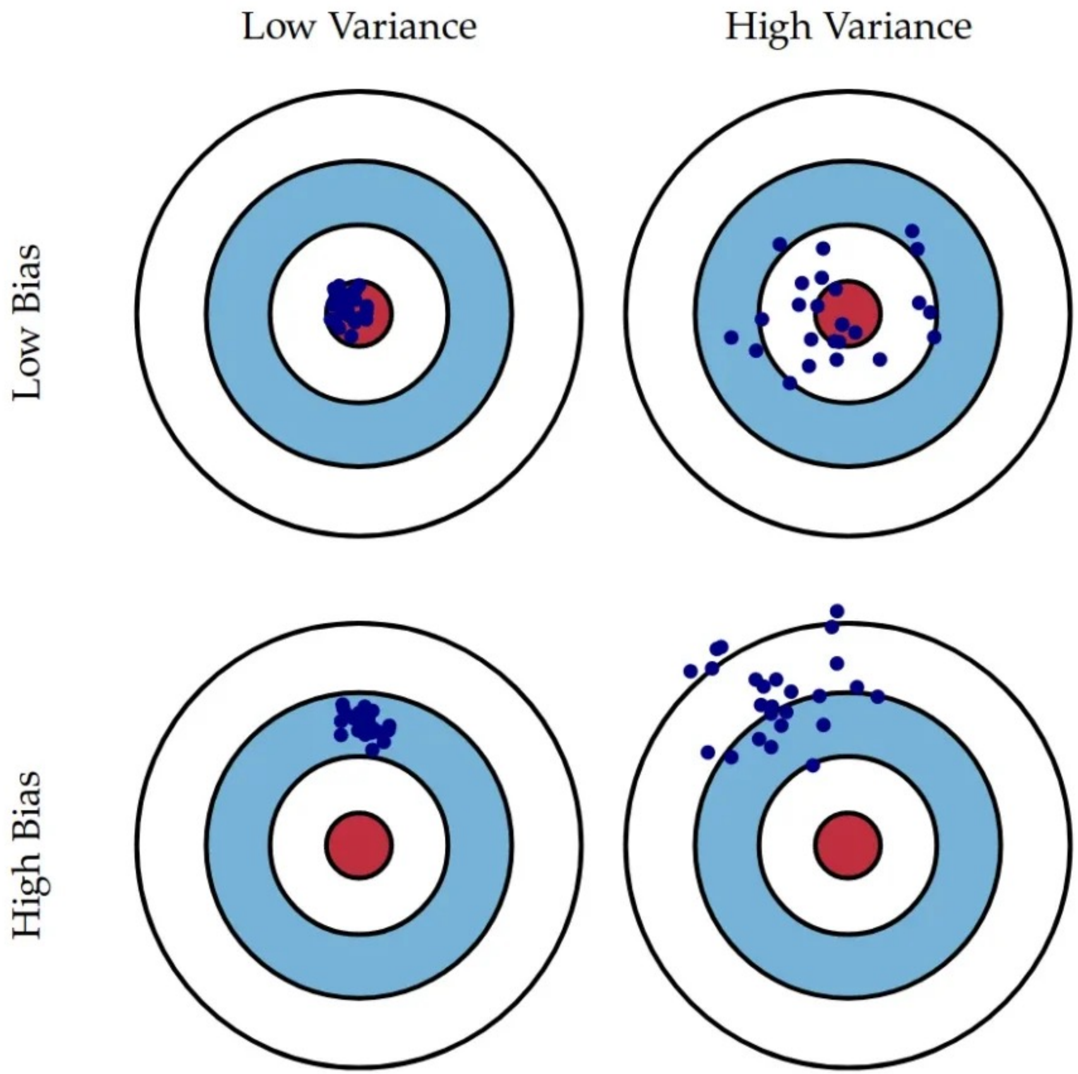
- On the other hand, very simple models lead to 'underfitting', where the available data is not fully leveraged ('the model has learned too little'). This is associated with high bias, as the model is too simplistic to capture the underlying structure of the data.

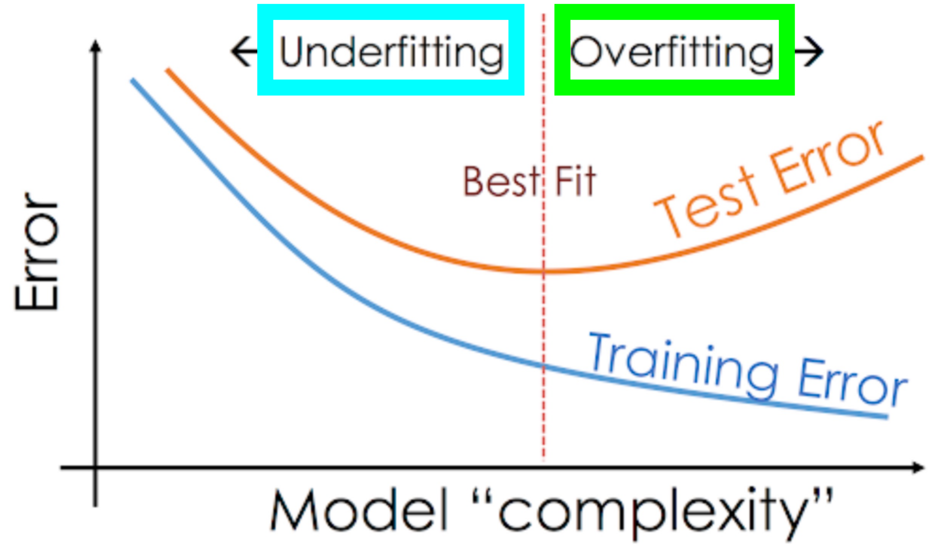


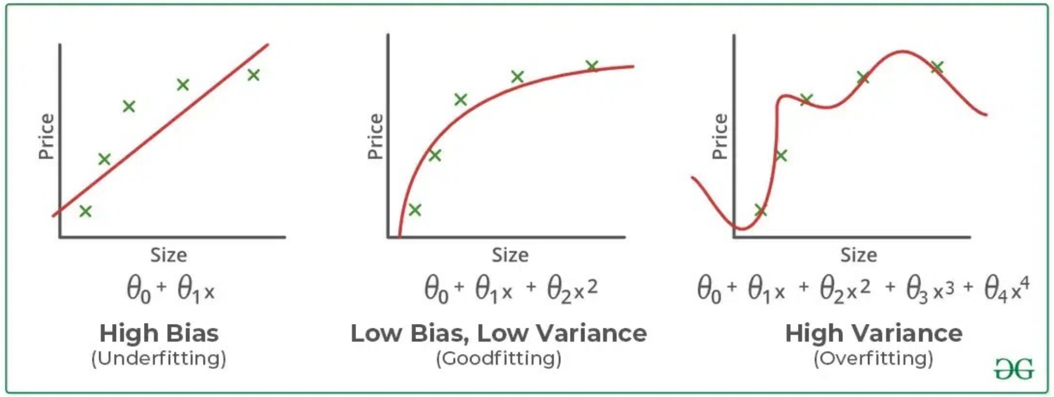


The key is to find a balance between bias and variance to achieve good generalization.

Recap: Bias & Variance / Under & Over-fitting







Recap: Regularization & Ridge Regression

Based on a test dataset, it is possible to choose the hyperparameter value to achieve the best trade-off between model complexity and accuracy

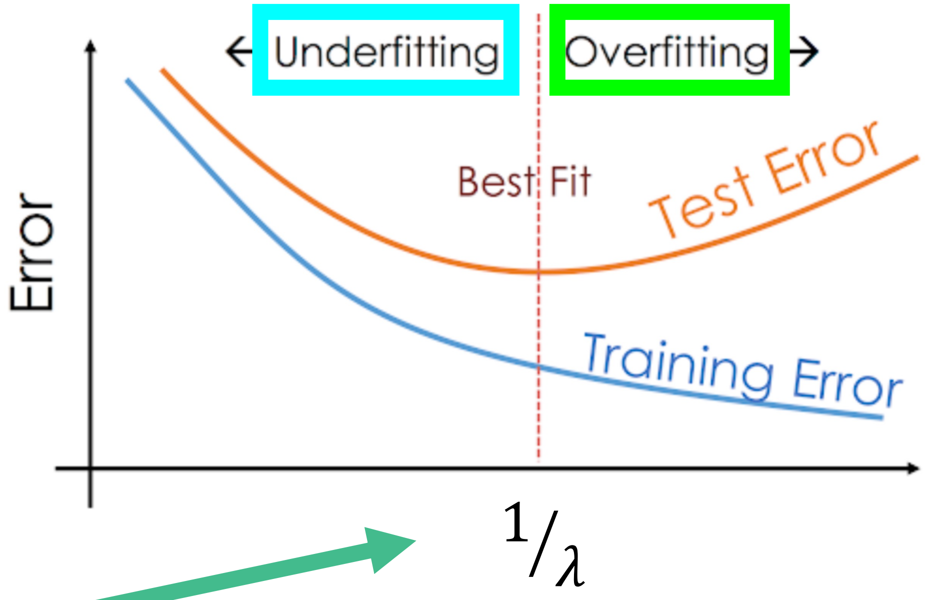
- Regularization is a technique used in ML to prevent overfitting by adding a penalty term to the loss function of a model.

- In linear regression, this is achieved by 'simply' changing the cost function: β s.t.

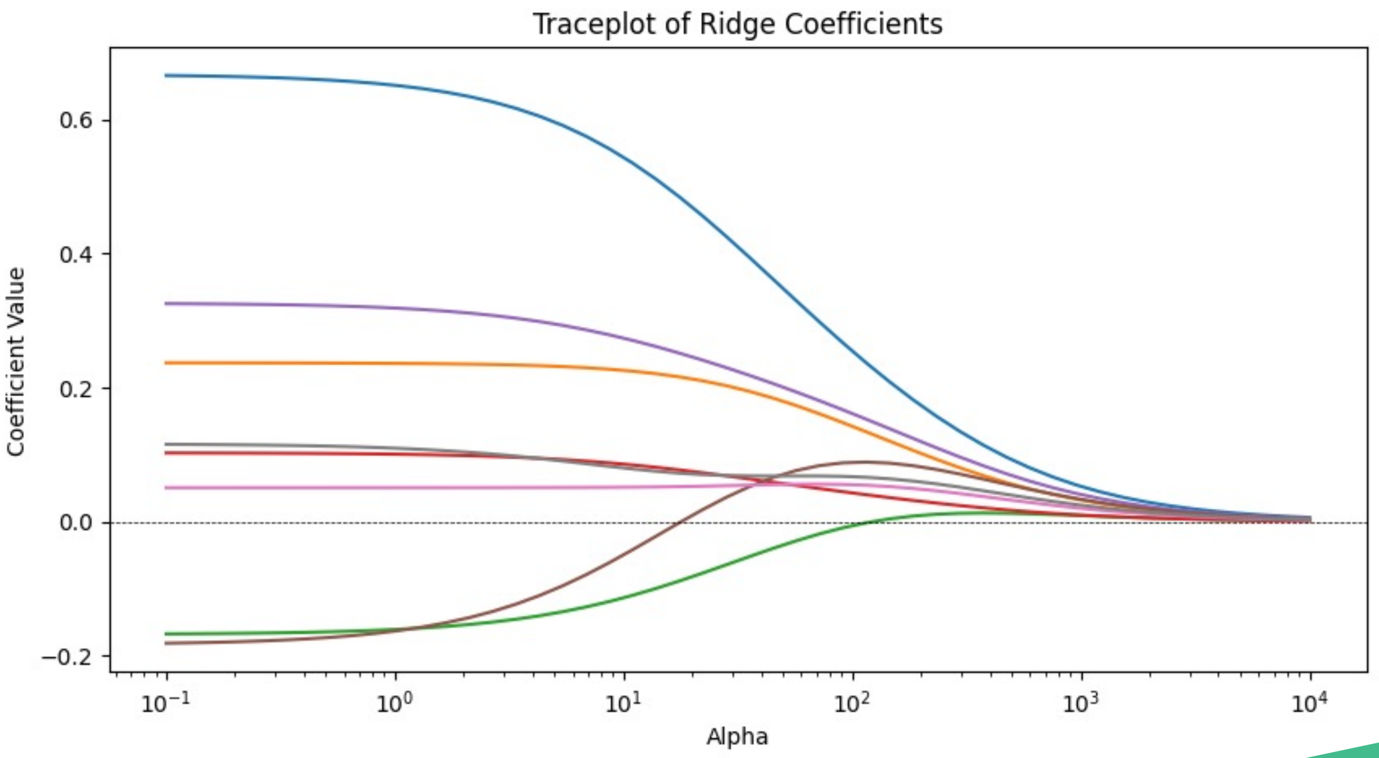
$$J = \sum_{i=1}^{n}[y^{(i)} - \widehat{y^{(i)}}]^2 + \boxed{\lambda} R,$$

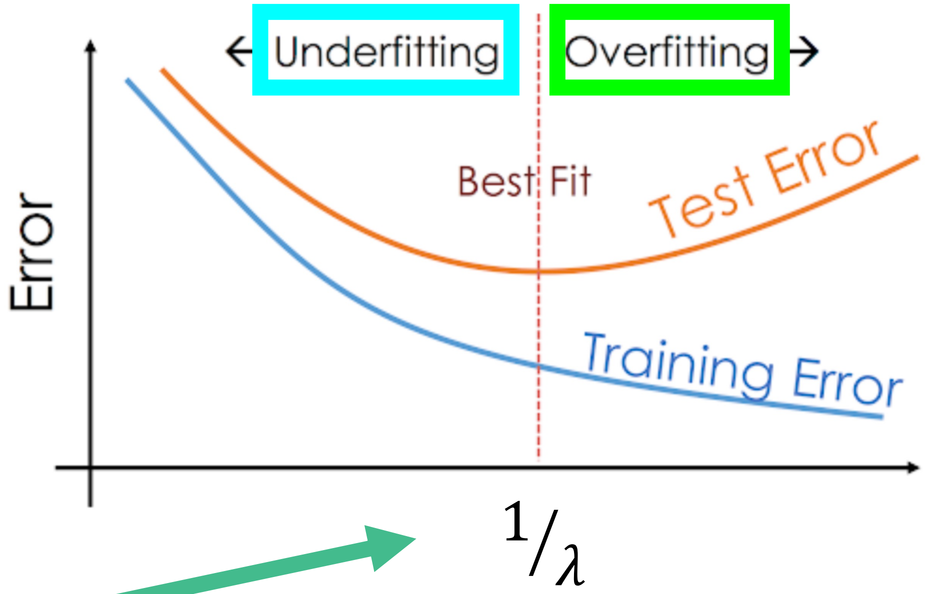
R is a penalty on model complexity.

Regularization parameter
(this is an hyperparameter)



Recap: Regularization & Ridge Regression



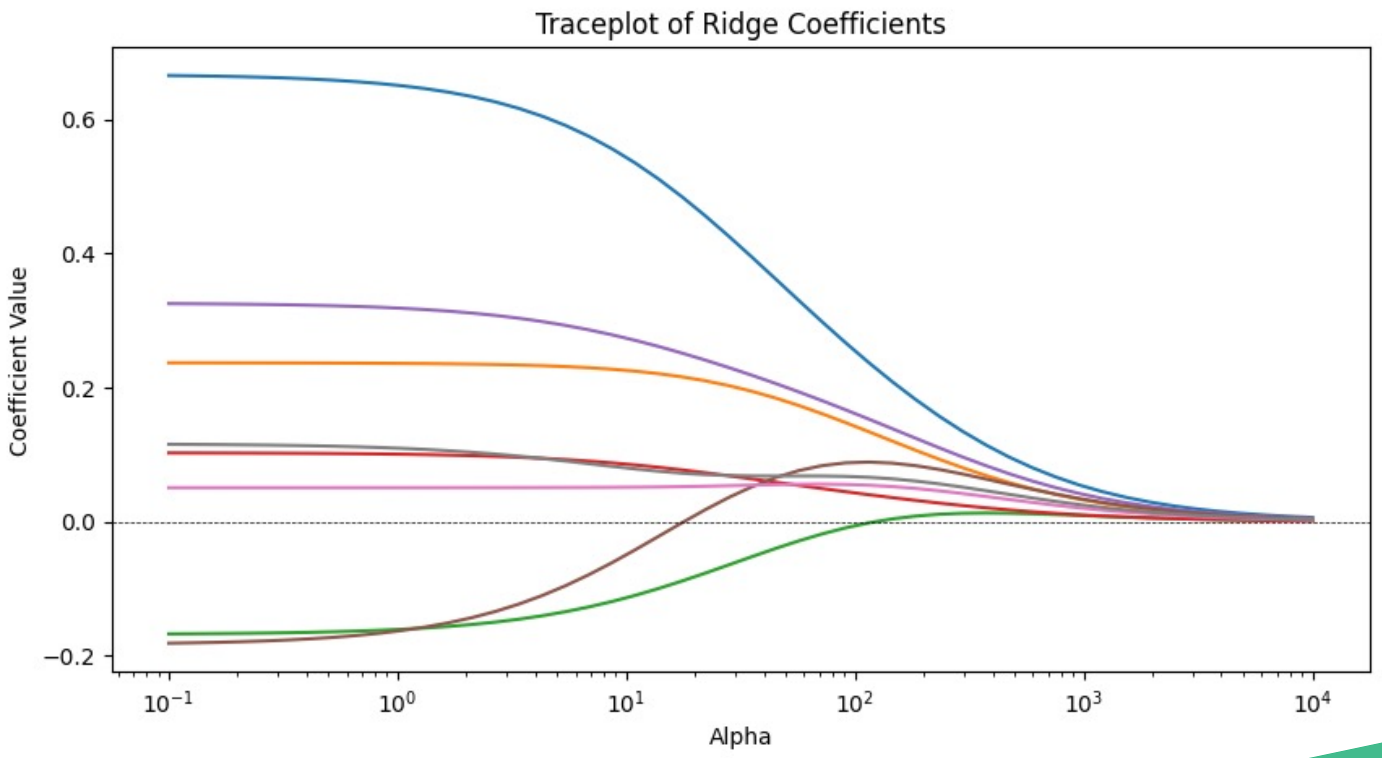


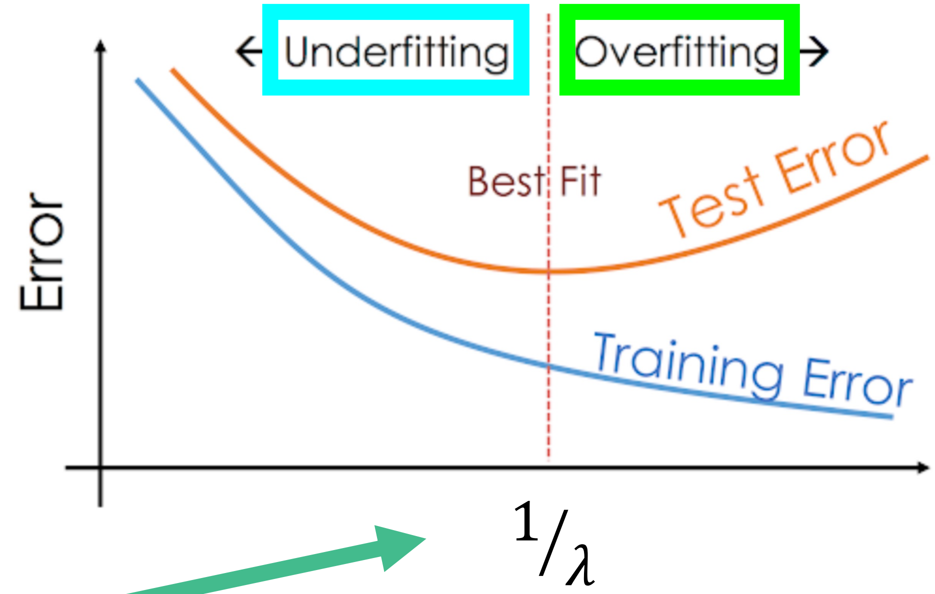
$$J = \sum_{i=1}^{n}[y^{(i)} - \widehat{y^{(i)}}]^2 + \boxed{\lambda}\, R,$$

If $R = \sum_{j=0}^{p} {\beta_j}^2$ (L2 penalization) we are dealing with Ridge Regression

What if we make other choices? Any ideas?

Recap: Regularization & Ridge Regression





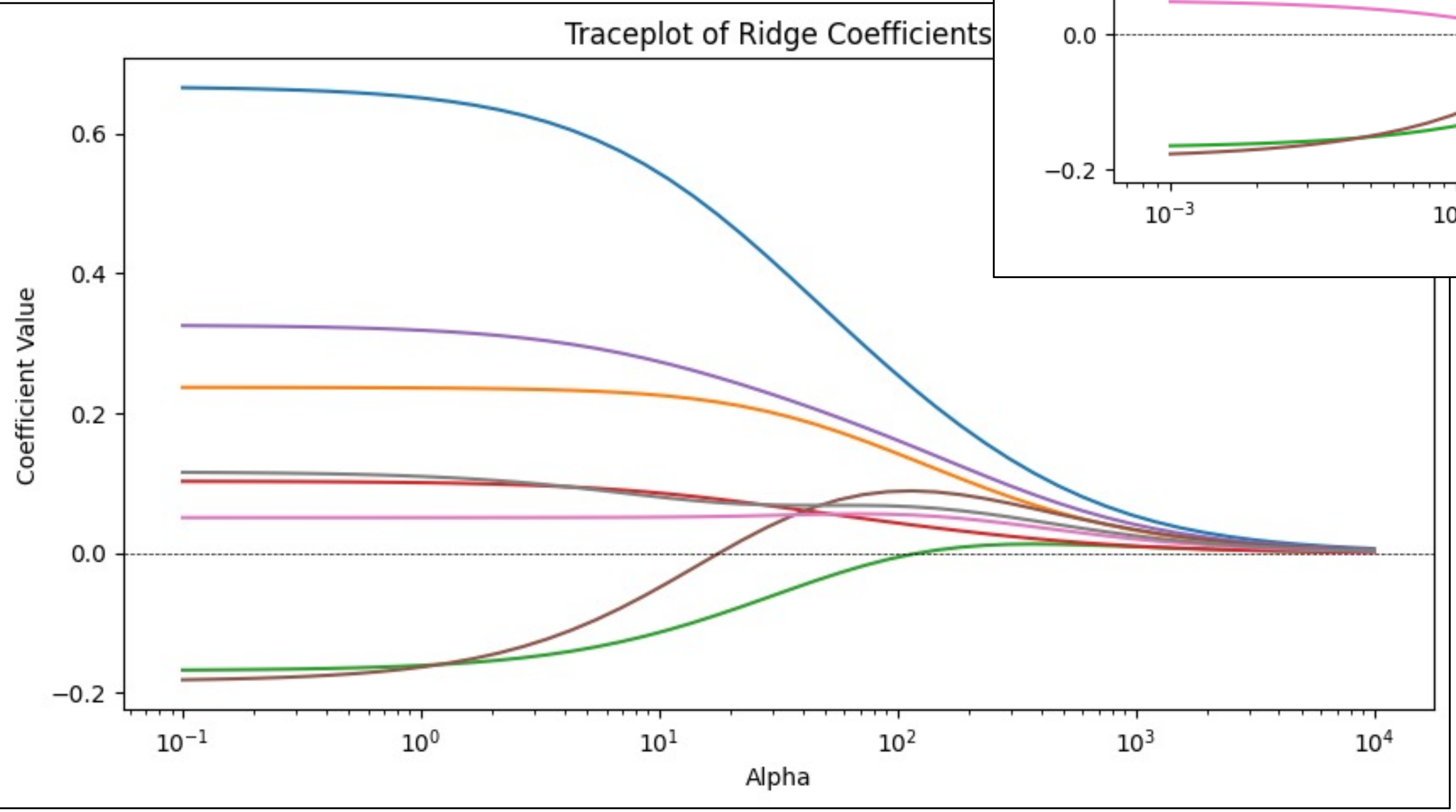
$$J = \sum_{i=1}^{n}[y^{(i)} - \widehat{y^{(i)}}]^2 + \boxed{\lambda} R,$$

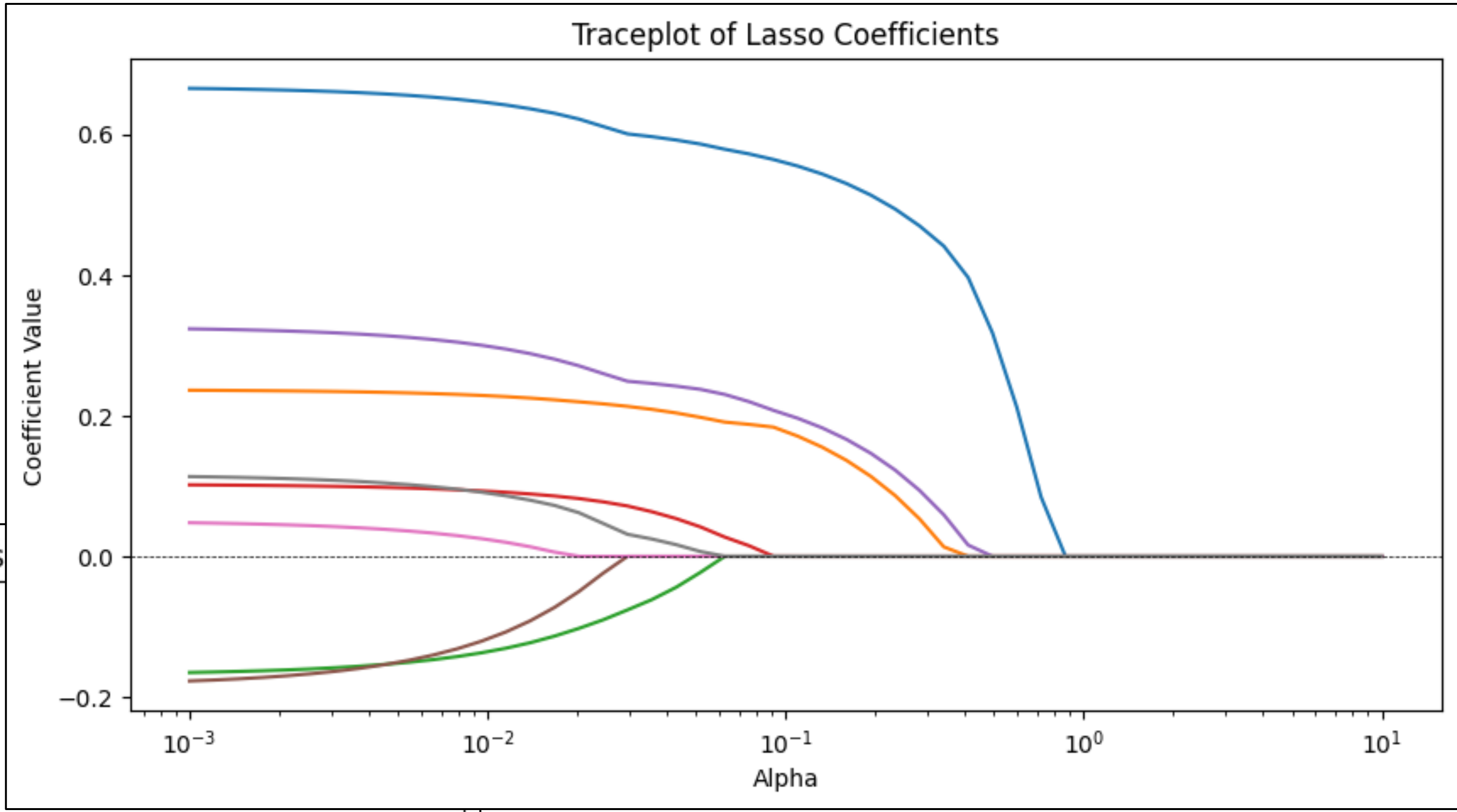
If $R = \sum_{j=0}^{p} \beta_j{}^2$ (L2 penalization) we are dealing with Ridge Regression

What if we make other choices? Any ideas?

If $R = \sum_{j=0}^{p} |\beta_j|$ (L1 penalization) we have Least Absolute Shrinkage and Selection Operator (LASSO)

Ridge Regression vs LASSO: traceplots

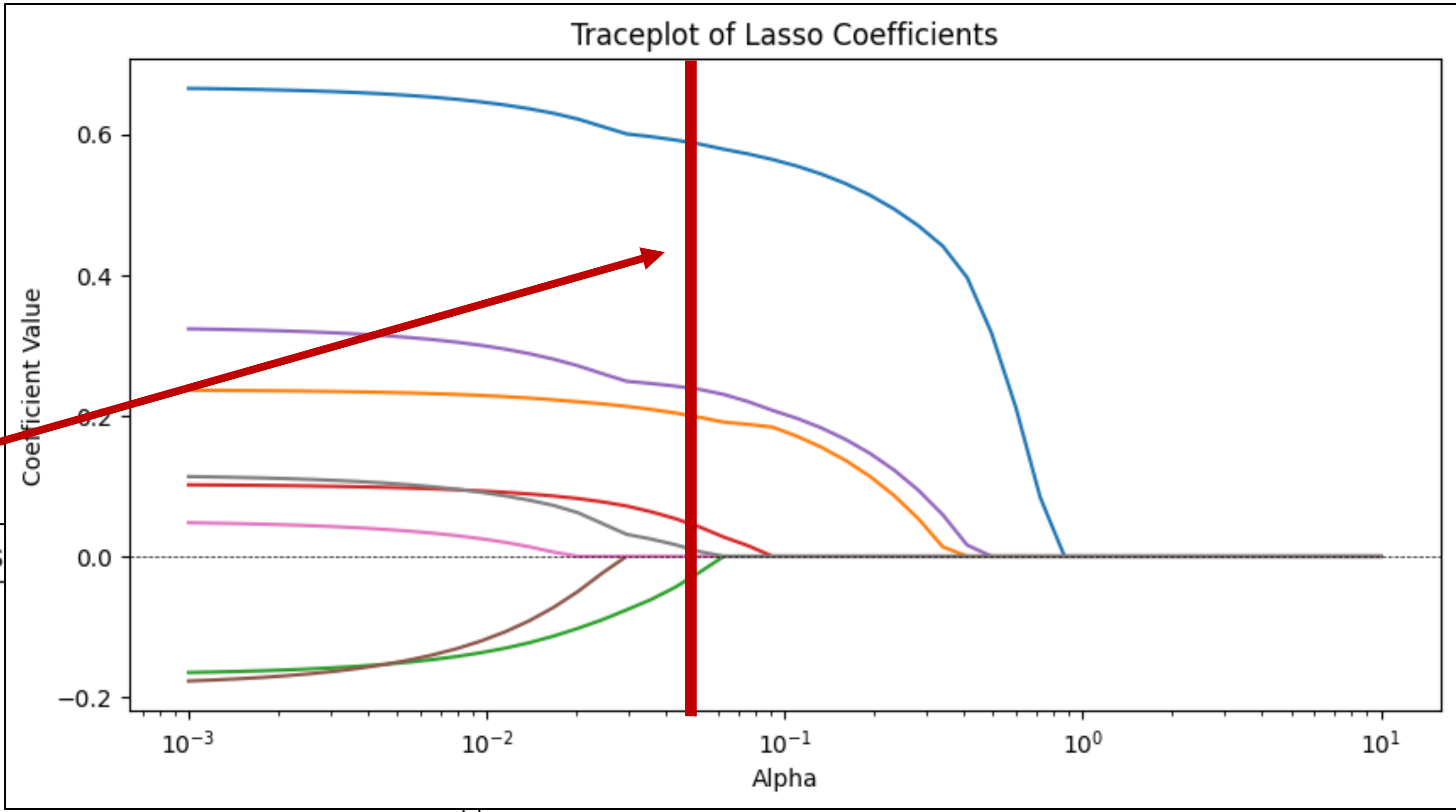


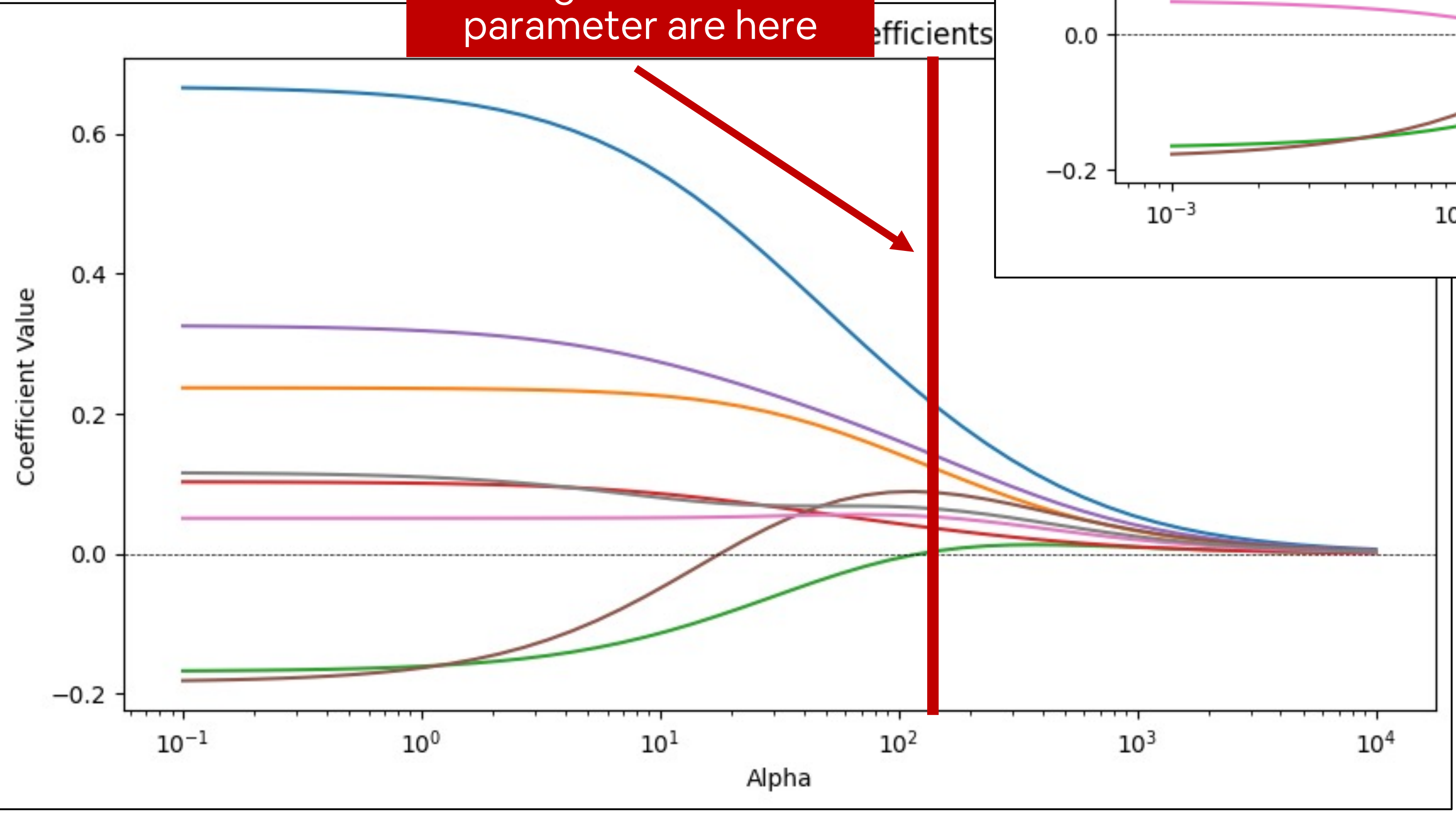


The Prostate dataset has 97 observations with 8 clinical features predicting log-PSA levels (proxy of the problem)
https://www.statlearning.com/resources-python

Ridge Regression vs LASSO: traceplots

Let's suppose the optimal values for the regularization parameter are here

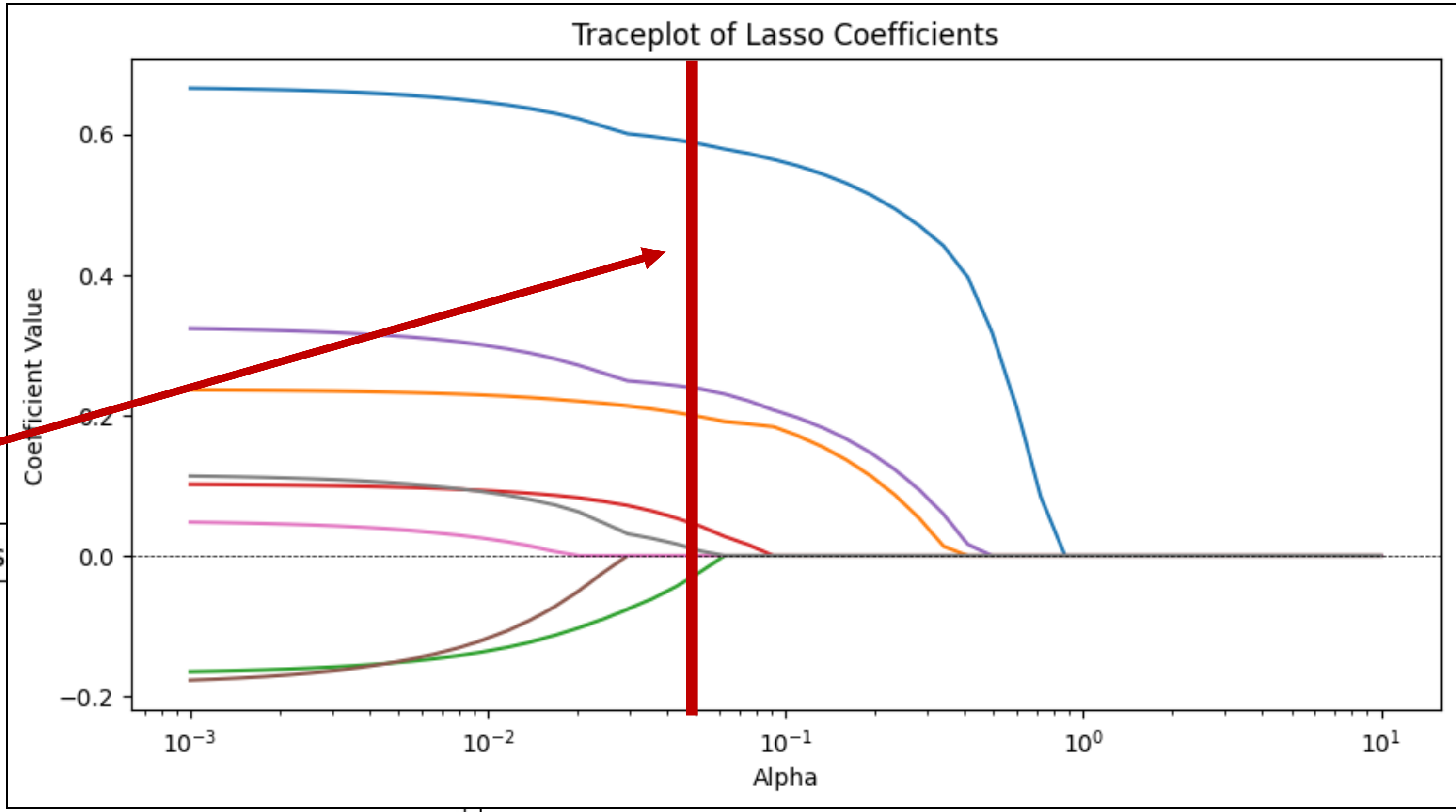


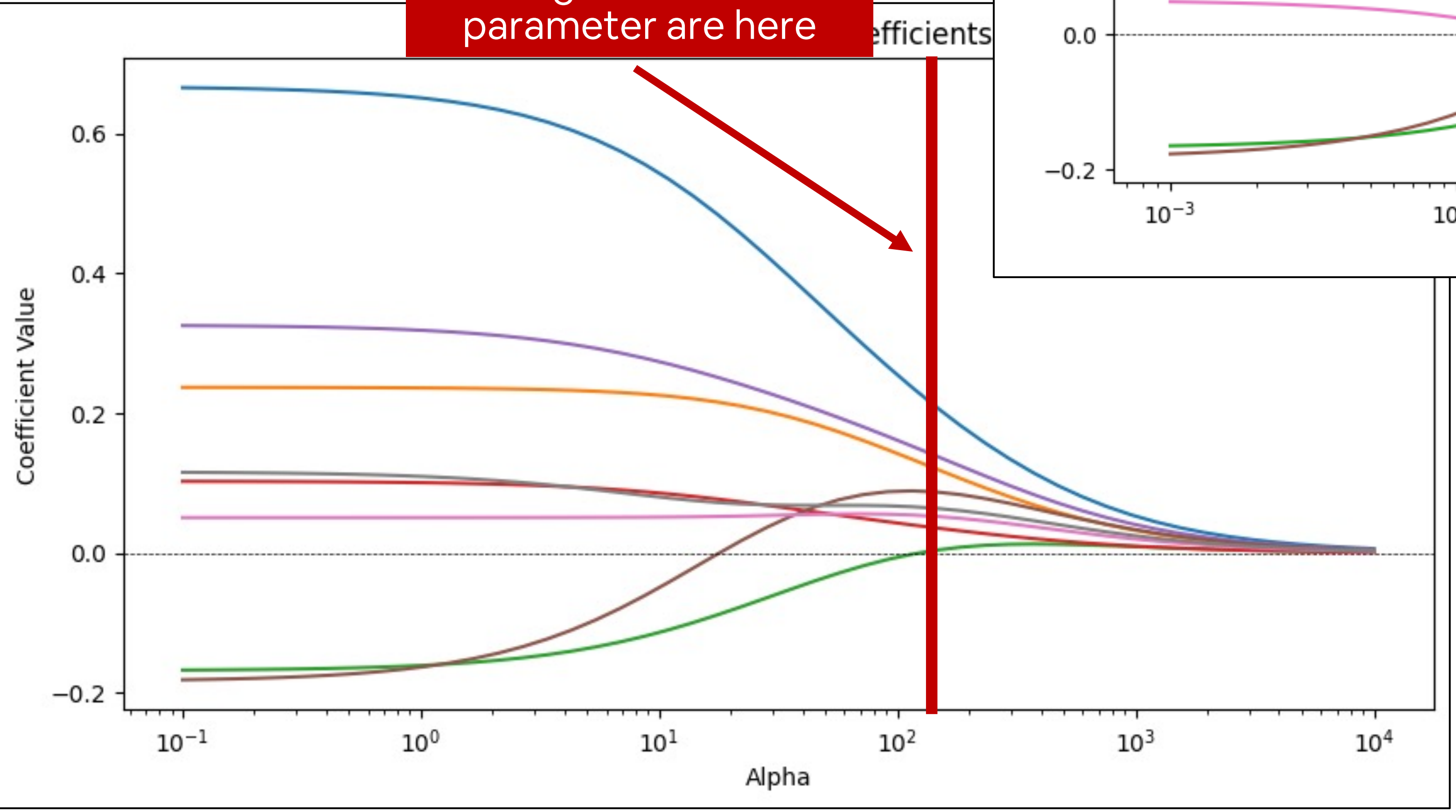


Which one will you choose?

Ridge Regression vs LASSO: traceplots

Let's suppose the optimal values for the regularization parameter are here





Which one will you choose?

- Performances

- Sparsity! Ie. having some coefficients equal to zero

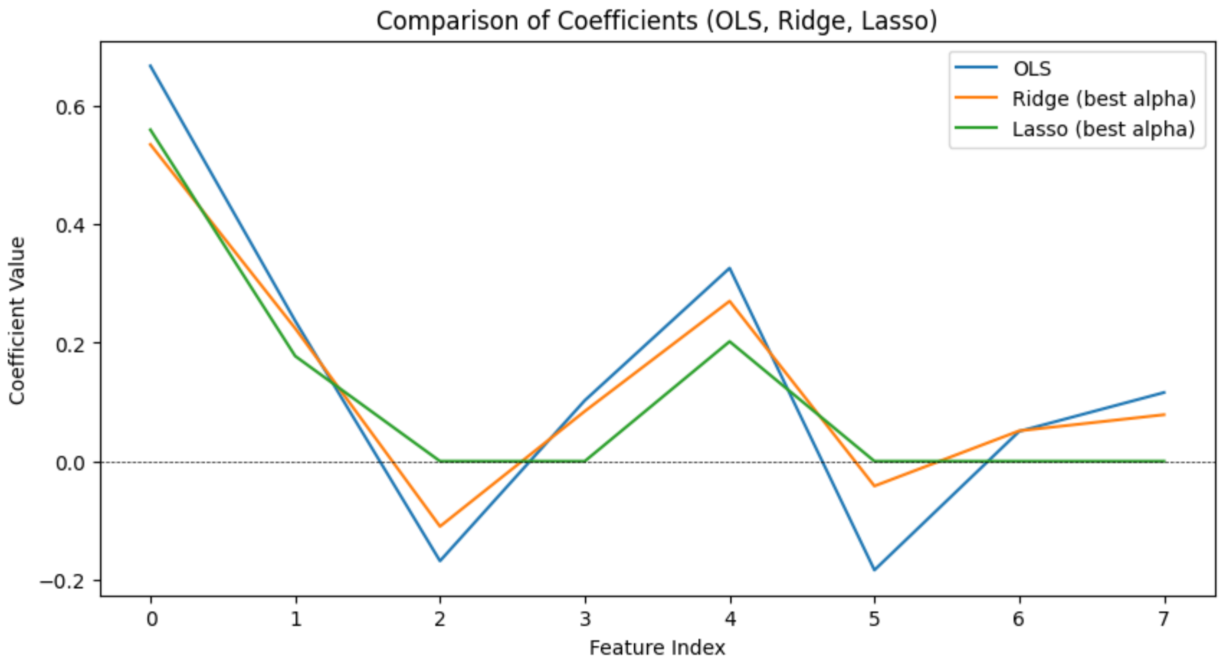
Benefits of a sparse solution

- <u>Computational efficiency</u> (faster prediction, lower storage and memory needs, efficient retraining, ...)

- <u>Robustness</u> (in case of noisy data for example)

- <u>Interpretability</u> (less variables => easier models to understand)

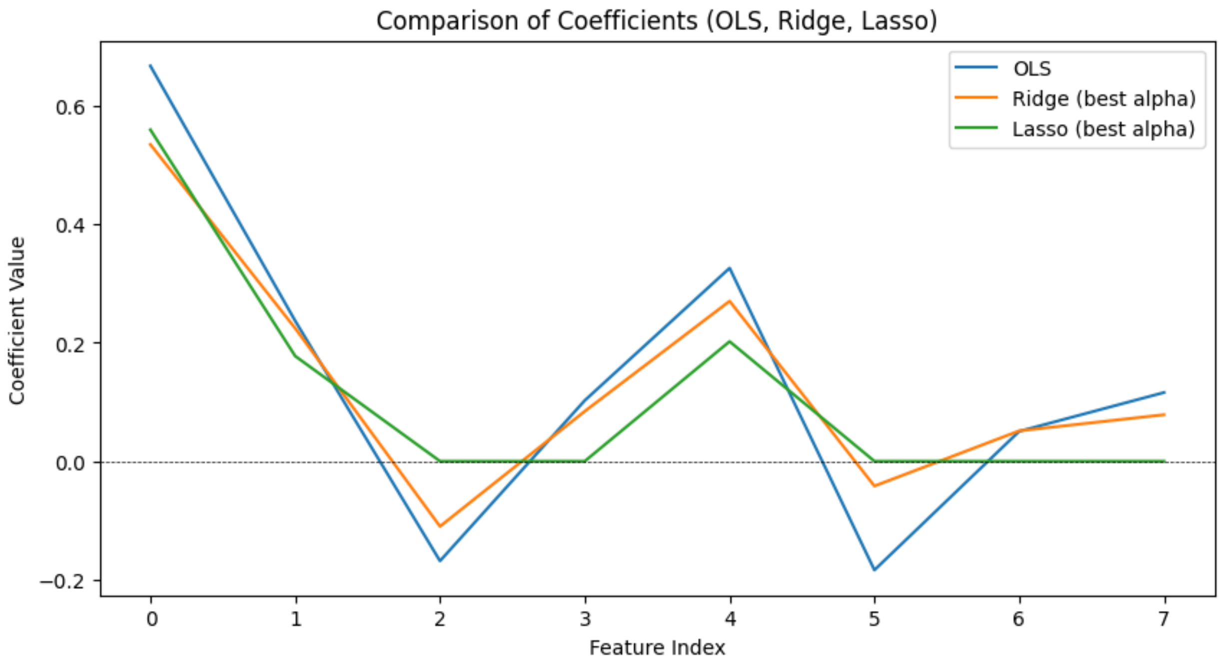
- <u>Easier management & deployment</u> (easier system management, edge implementation ...)



Benefits of a sparse solution

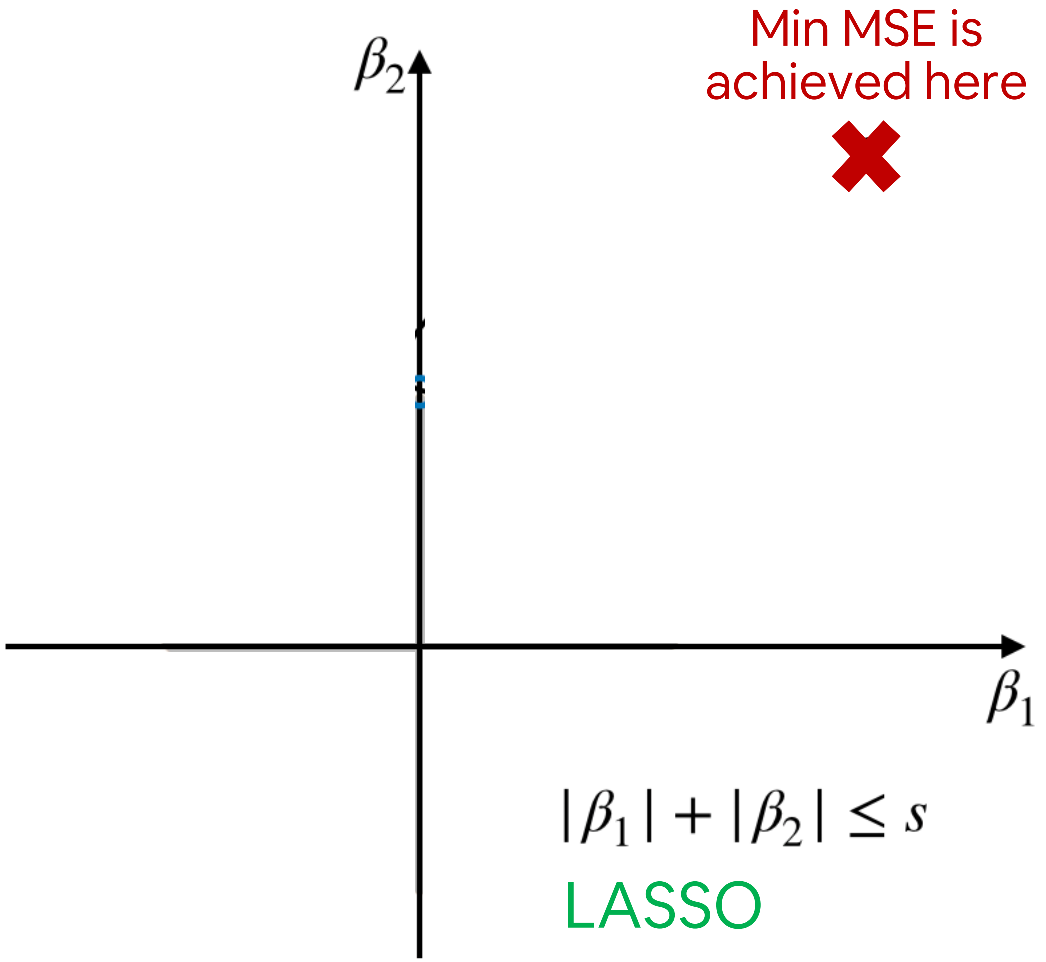
- Computational efficiency (faster prediction, lower storage and memory needs, efficient retraining, ...)
- Robustness (in case of noisy data for example)
- Interpretability (less variables => easier models to understand)
- Easier management & deployment (easier system management, edge implementation ...)

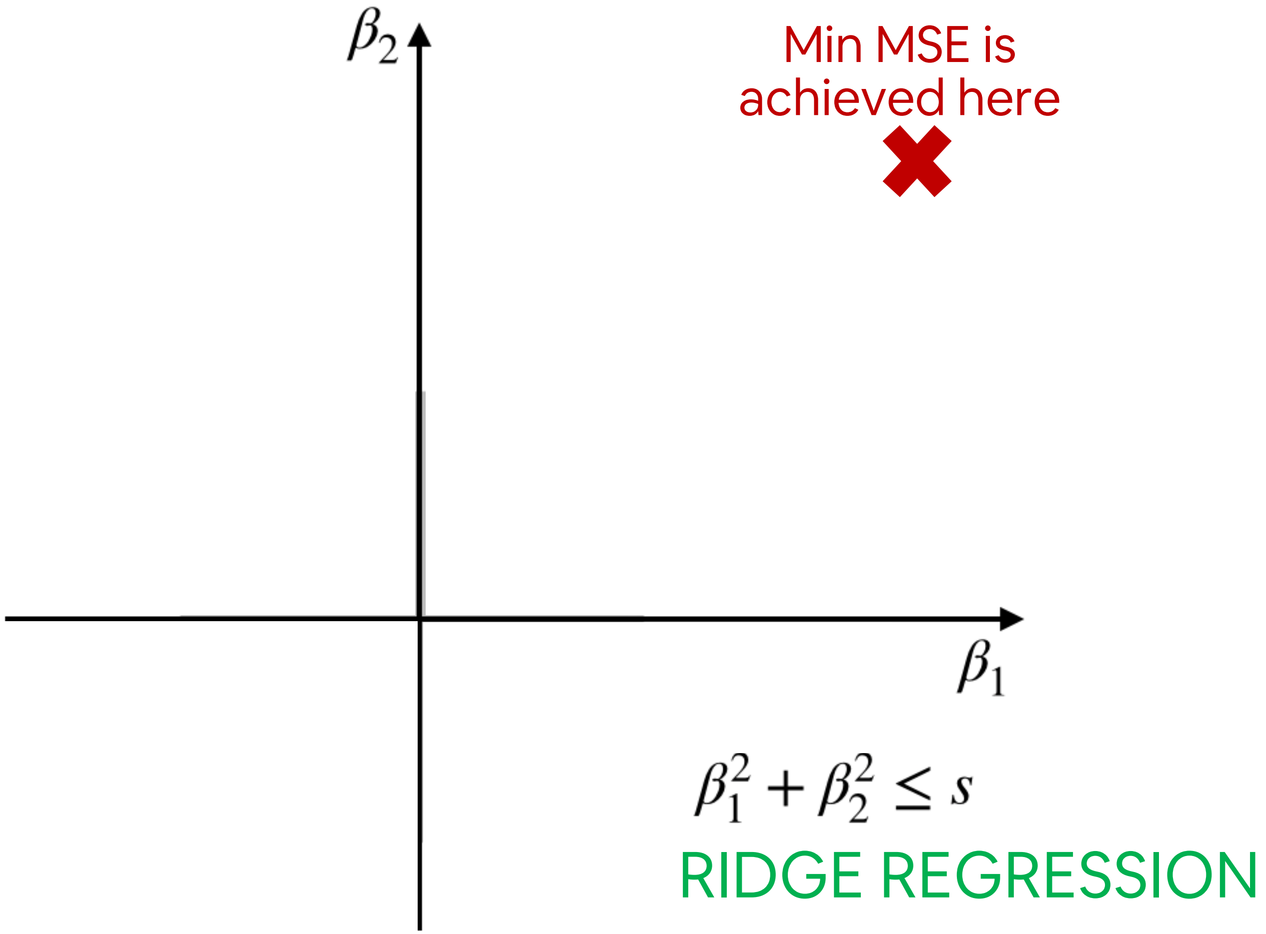
The LASSO performs feature selection, one of the preprocessing step, directly inside the modelling phase!



A geometric interpretation

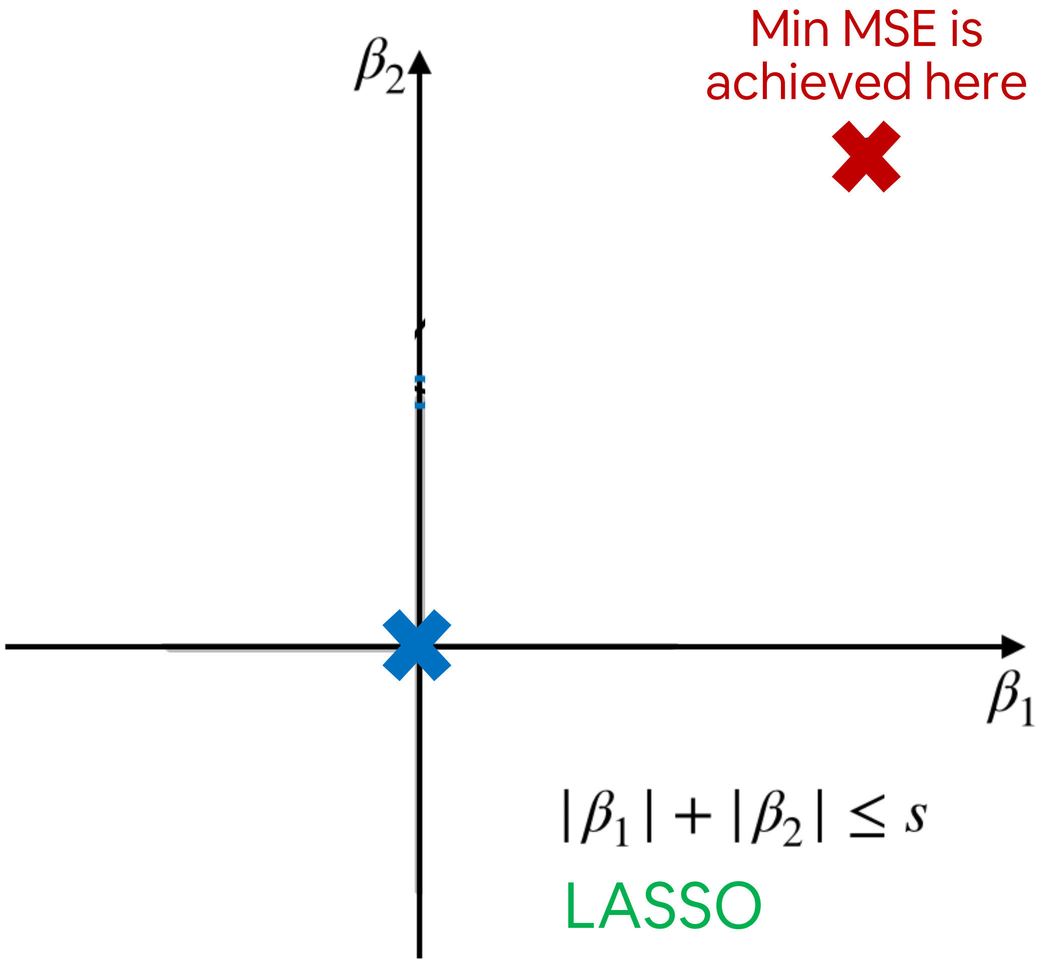
$$J = \boxed{\sum_{i=1}^{n}[y^{(i)} - \widehat{y^{(i)}}]^2} + \lambda R$$

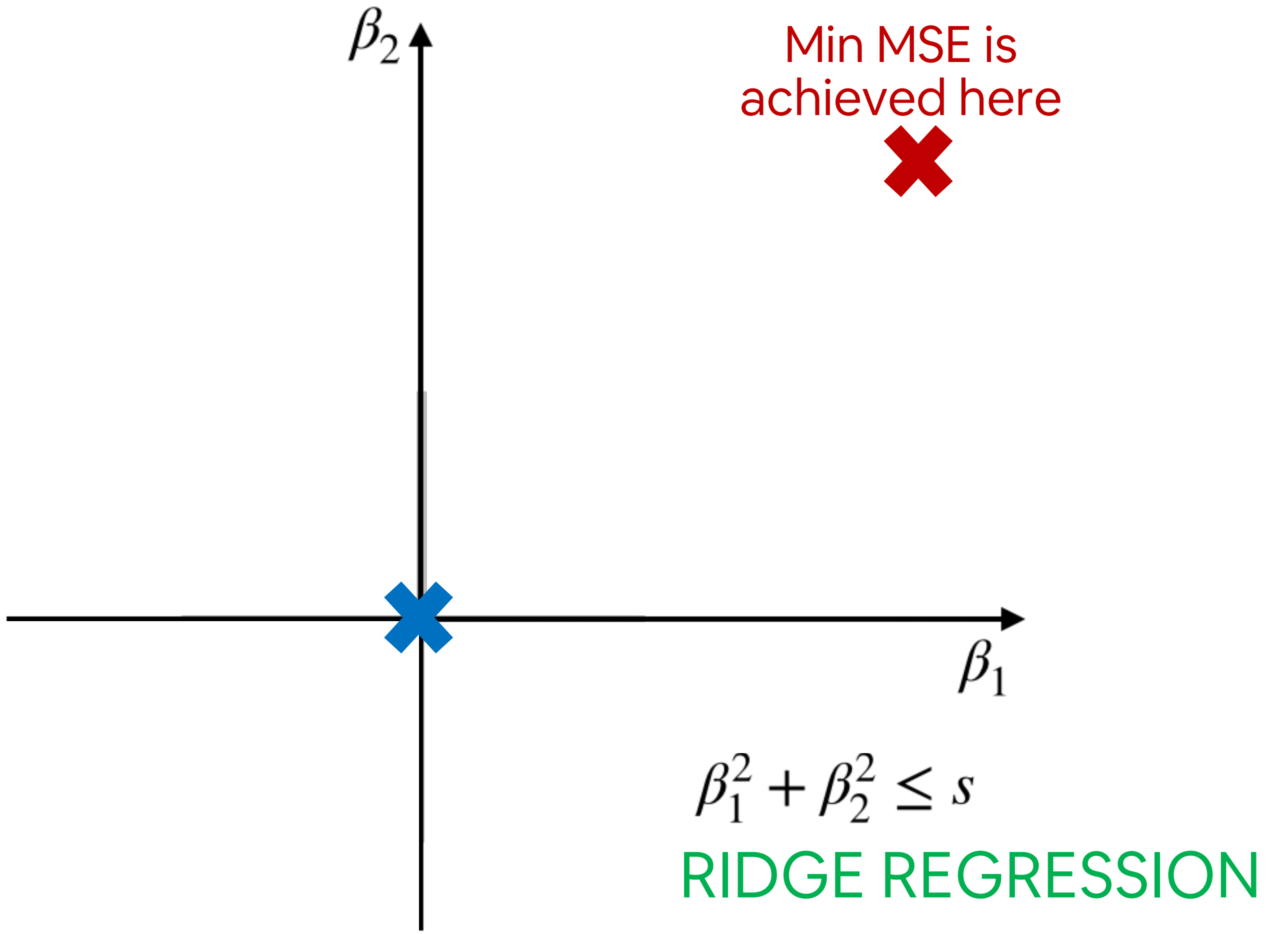




A geometric interpretation

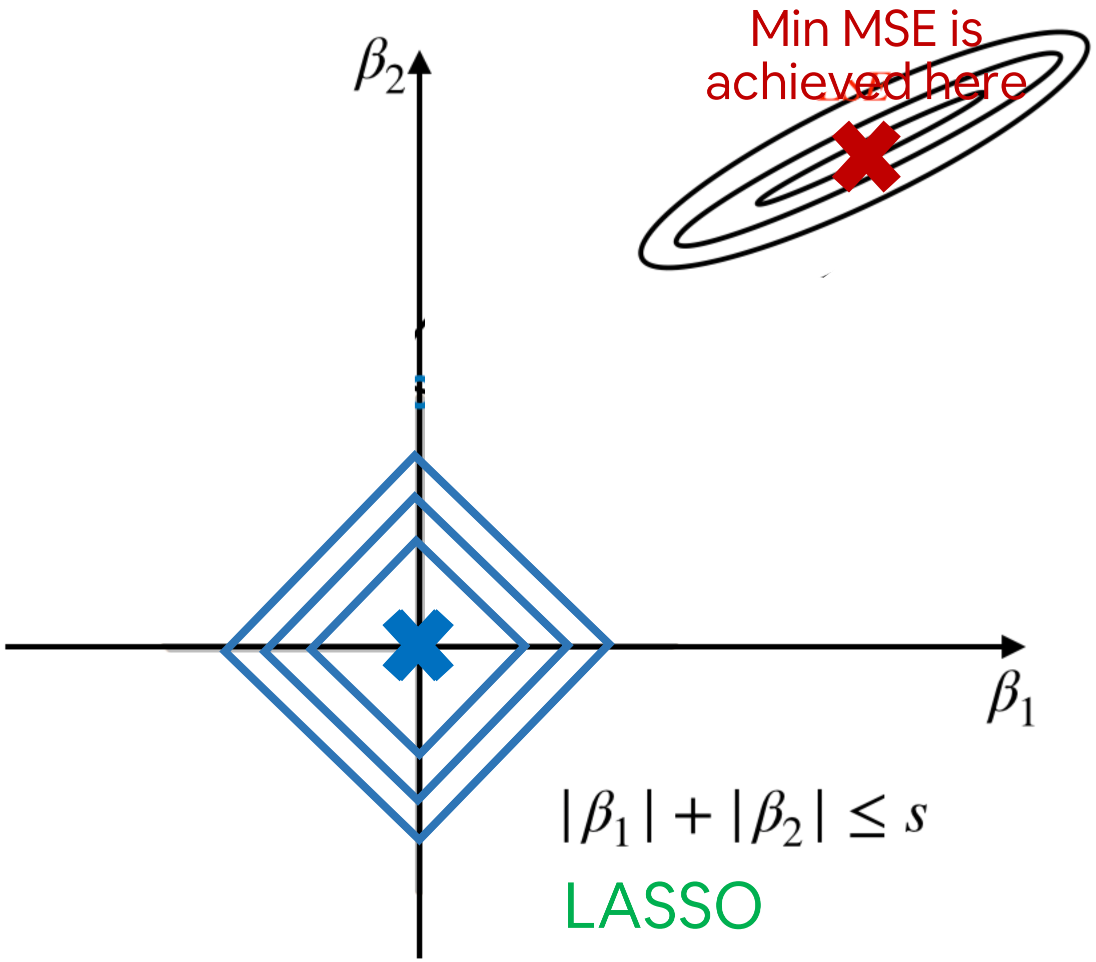
$$J = \sum_{i=1}^{n}[y^{(i)} - \widehat{y^{(i)}}]^2 + \lambda R$$

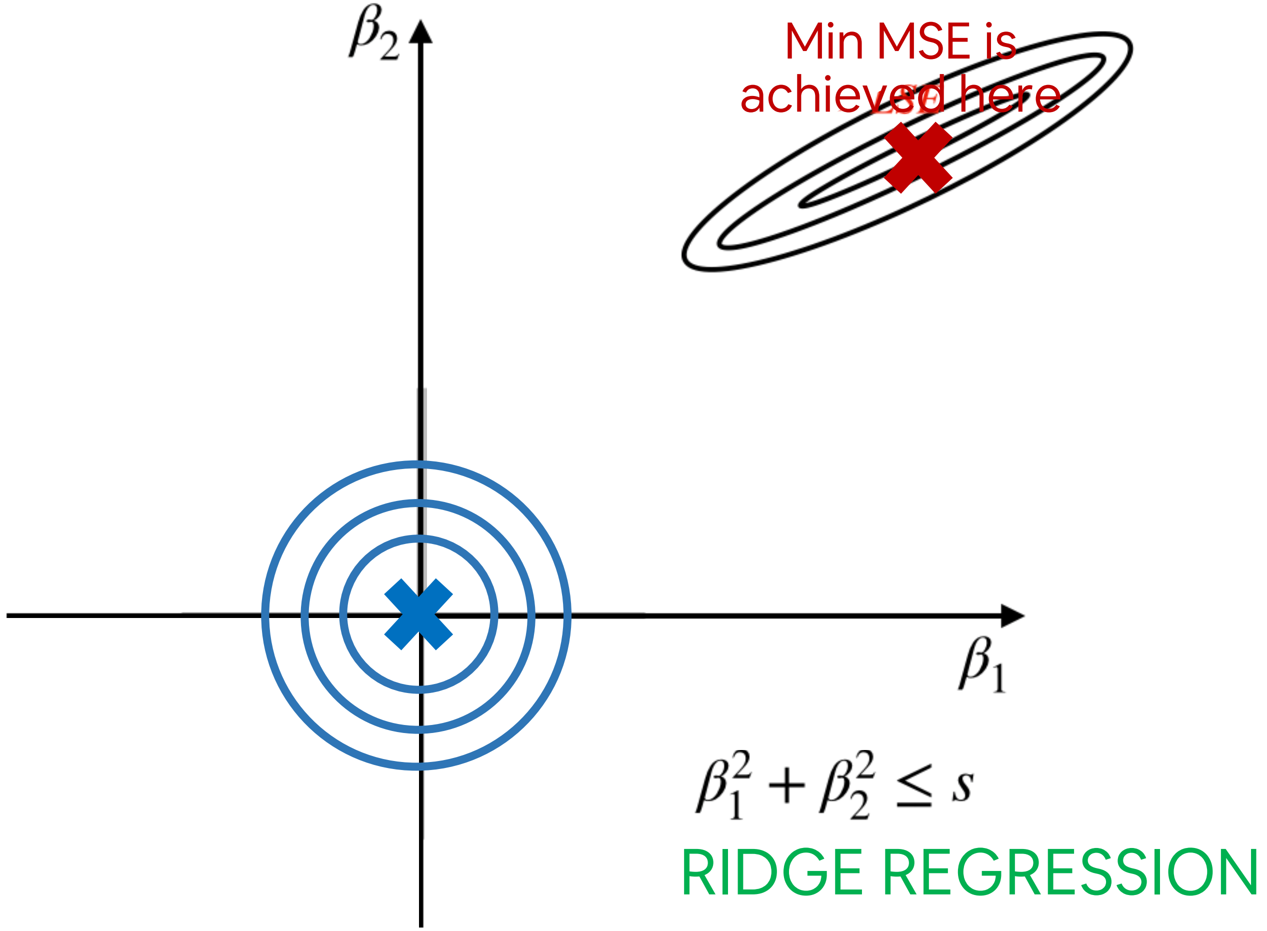




A geometric interpretation

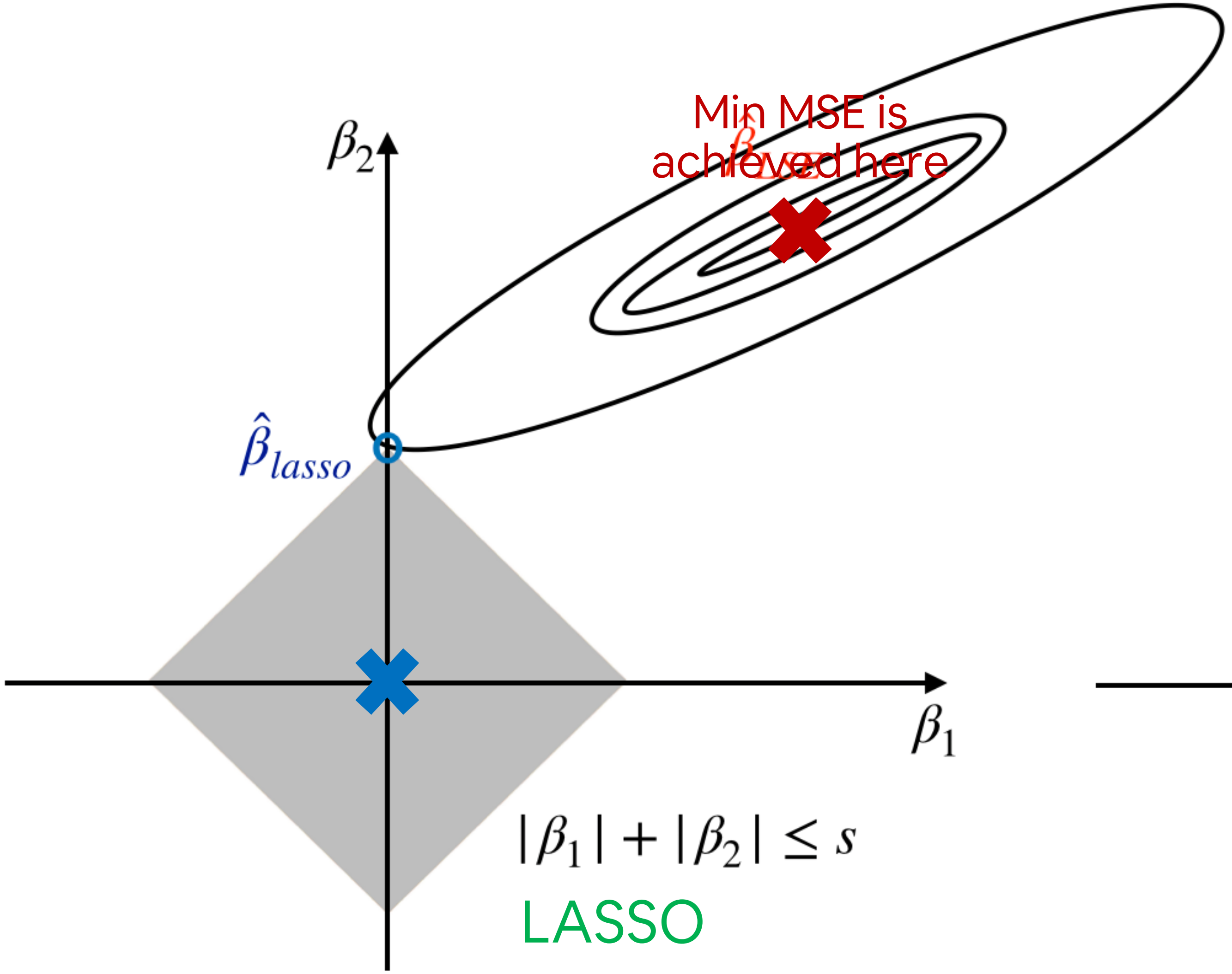
$$J = \sum_{i=1}^{n}[y^{(i)} - \widehat{y^{(i)}}]^2 + \lambda R$$

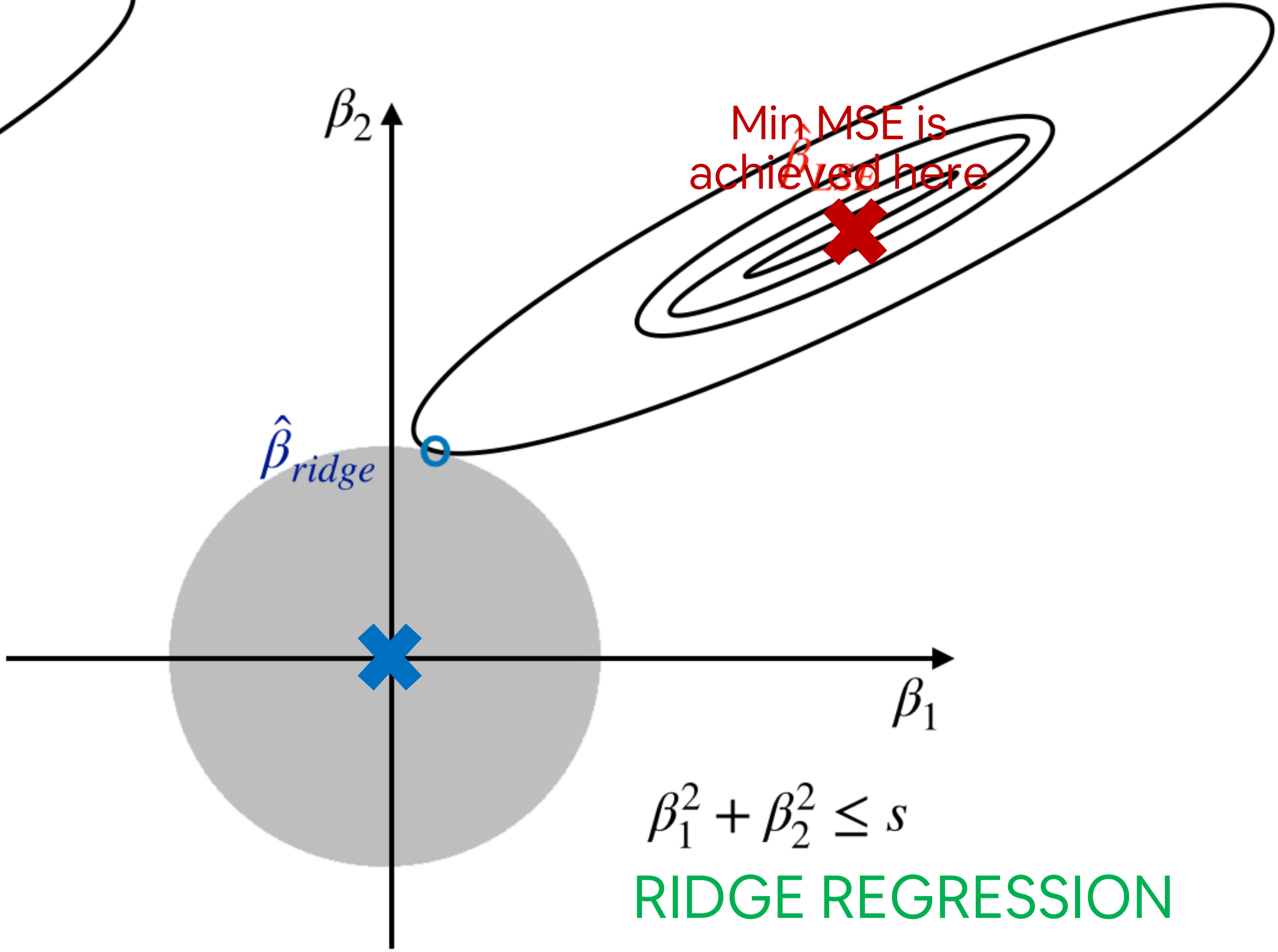




A geometric interpretation

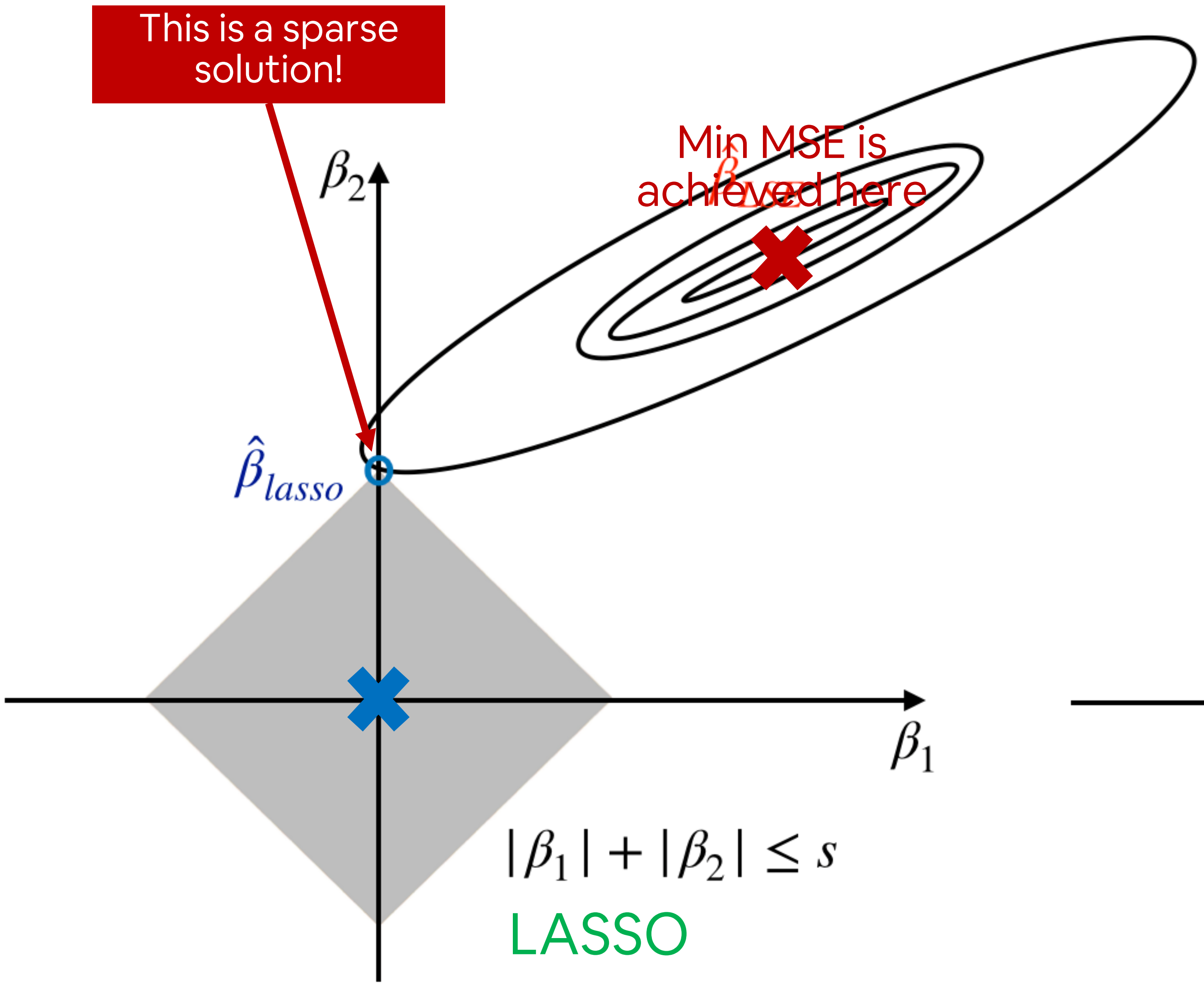
$$J = \sum_{i=1}^{n}[y^{(i)} - \widehat{y^{(i)}}]^2 + \lambda R$$

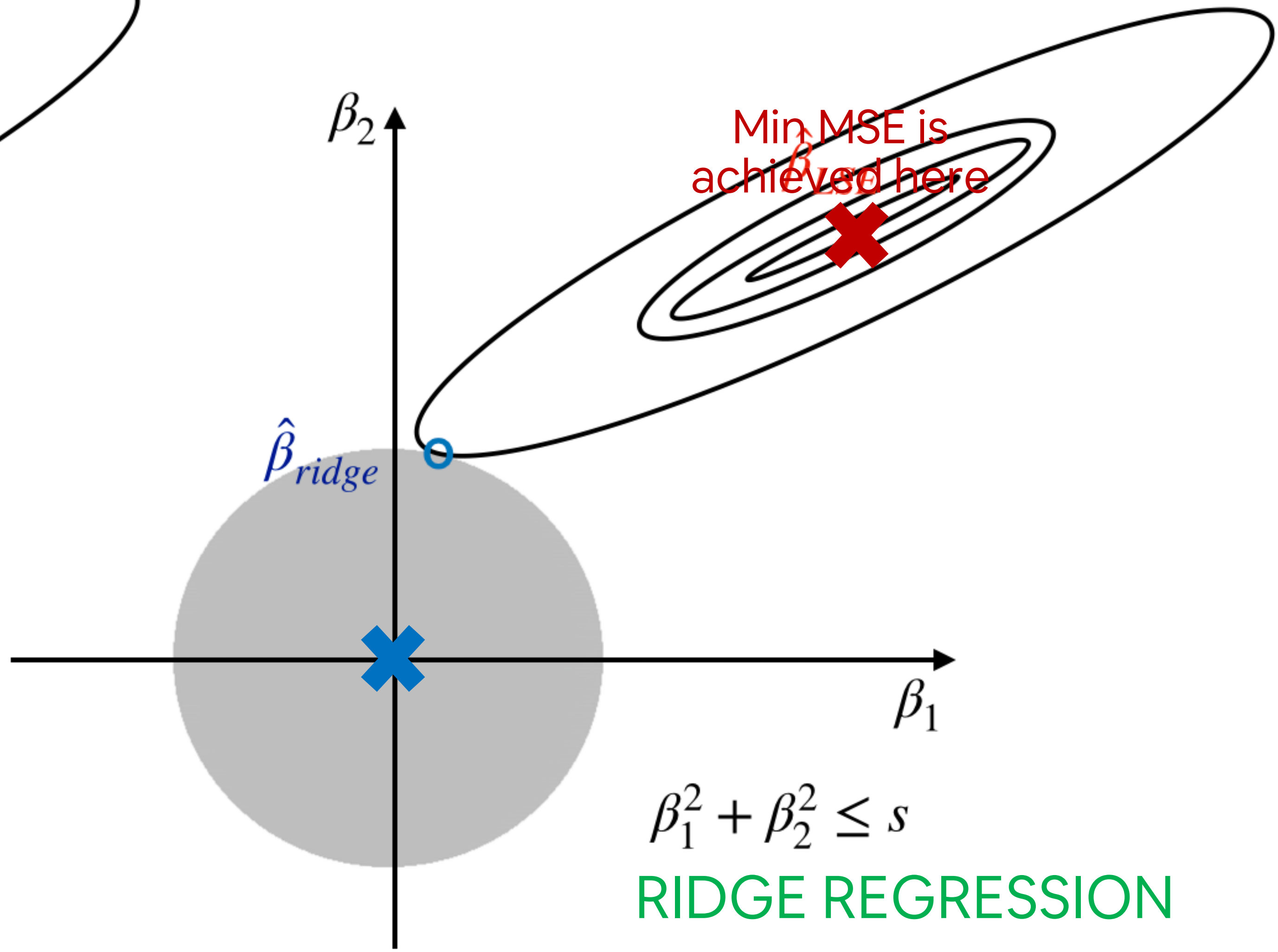




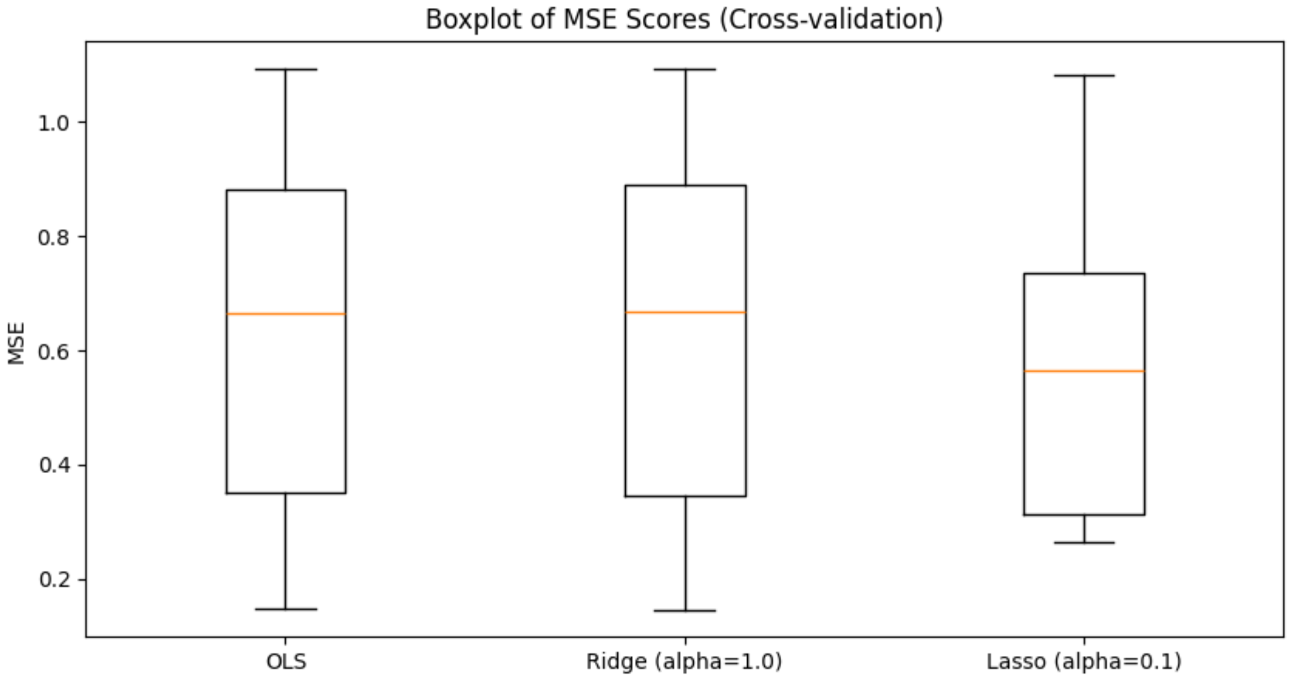
A geometric interpretation

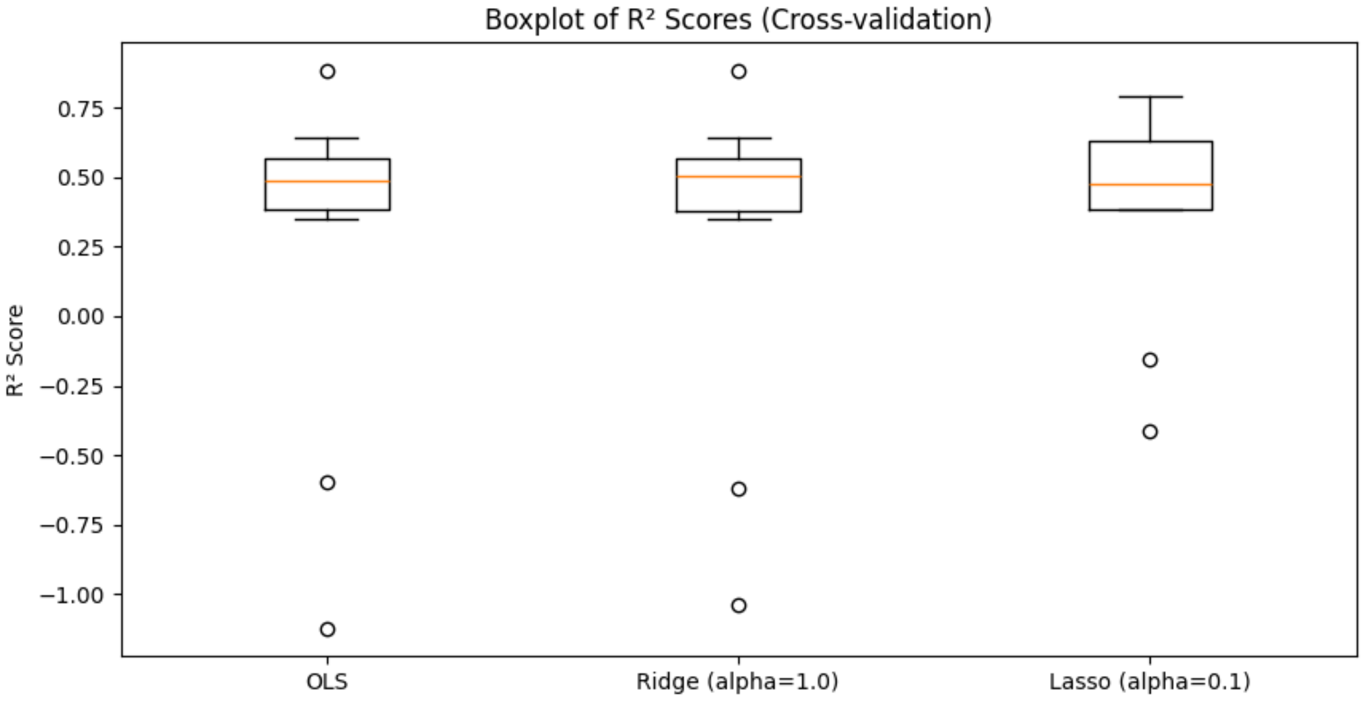
$$J = \sum_{i=1}^{n}[y^{(i)} - \widehat{y^{(i)}}]^2 + \lambda\, R$$



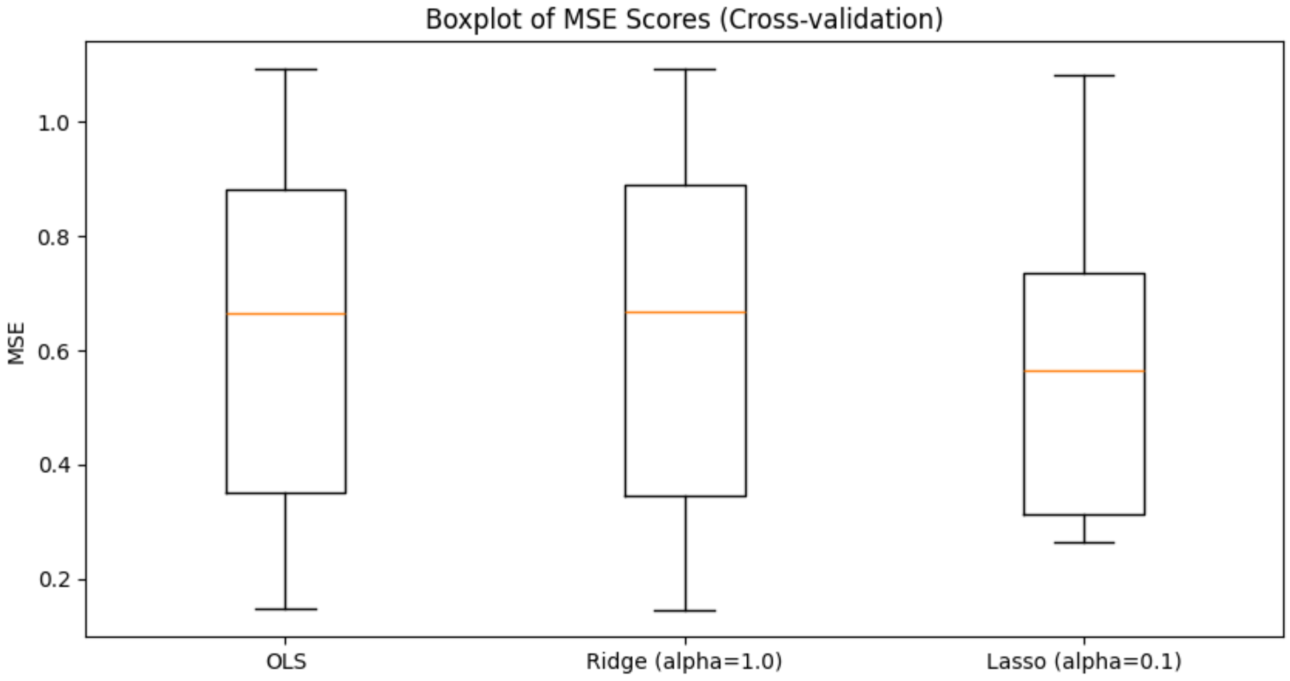


Performance on the Prostate Dataset

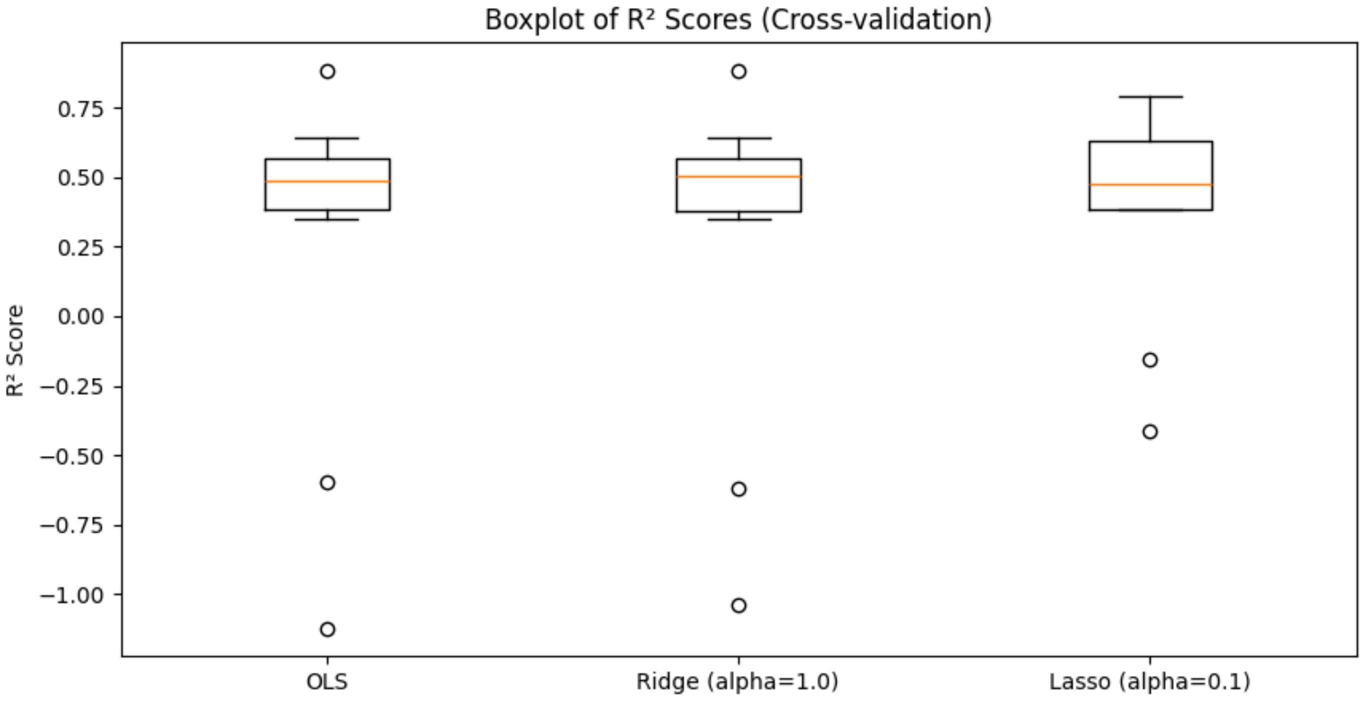




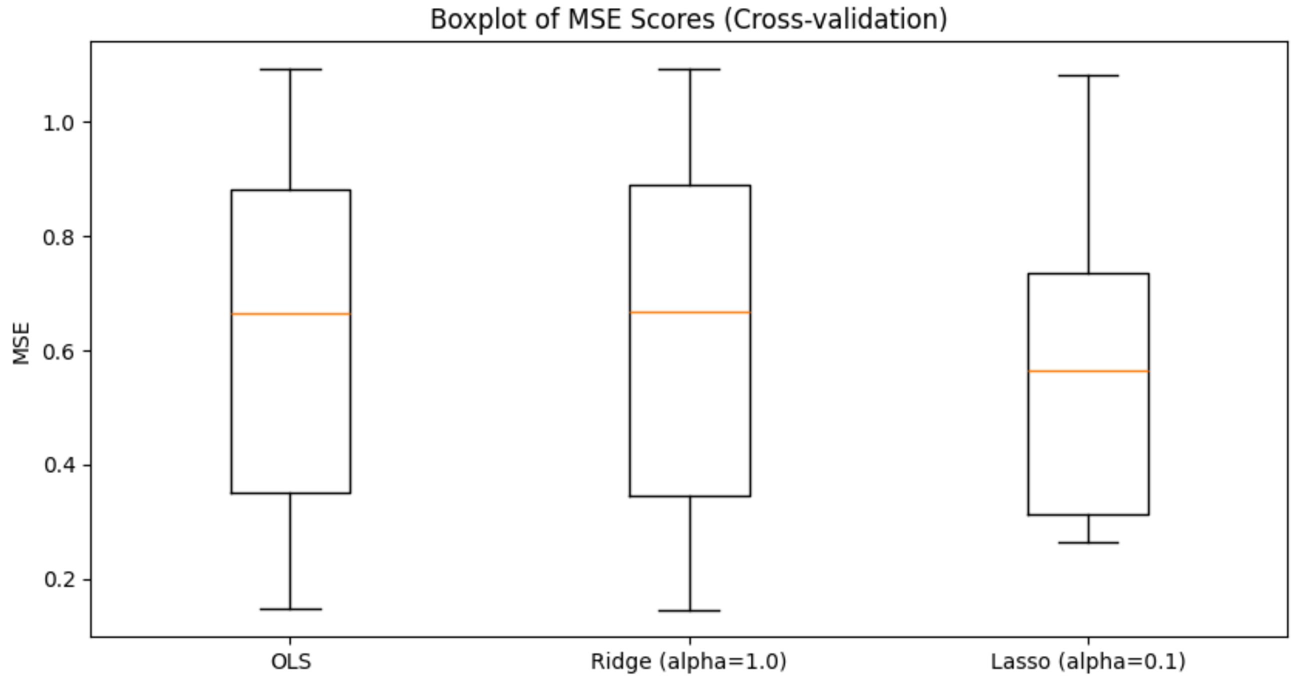
Performance on the Prostate Dataset



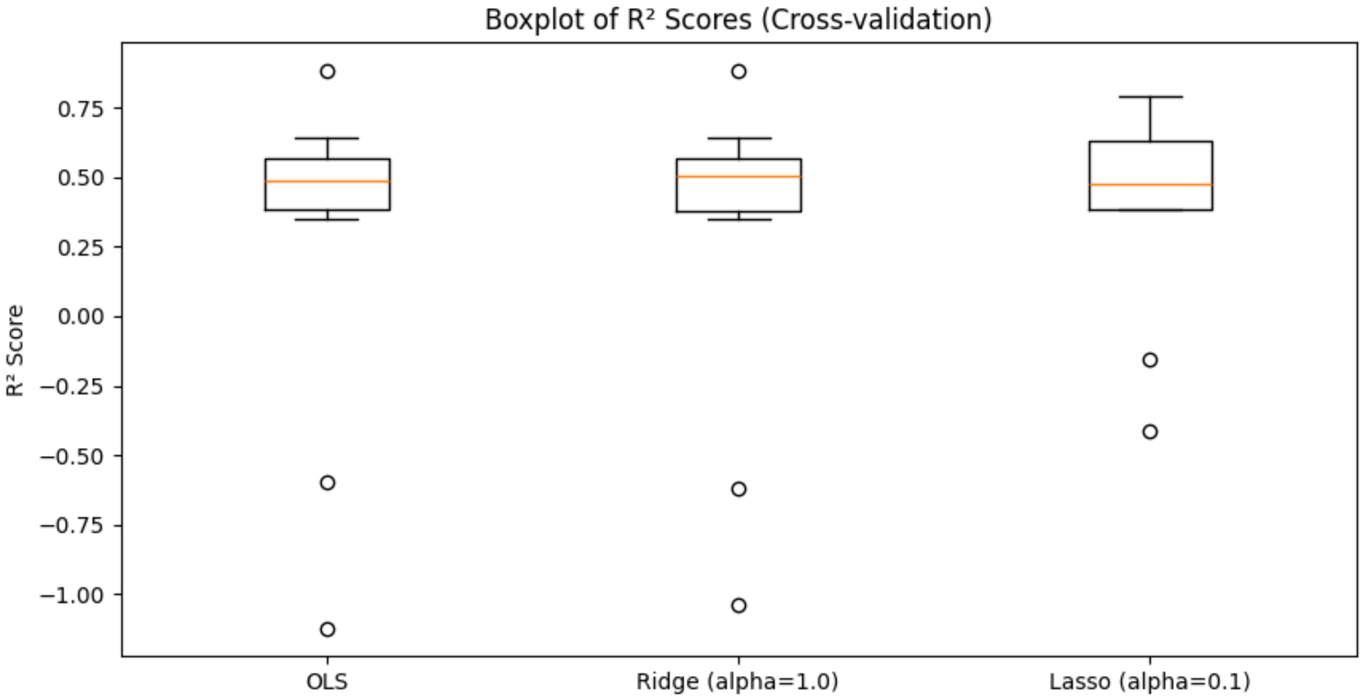
Any drawbacks in using LASSO?



Performance on the Prostate Dataset



Any drawbacks in using LASSO?



Unfortunately, we cannot rely on a closed-form solution to derive the parameters!

How do we solve this?

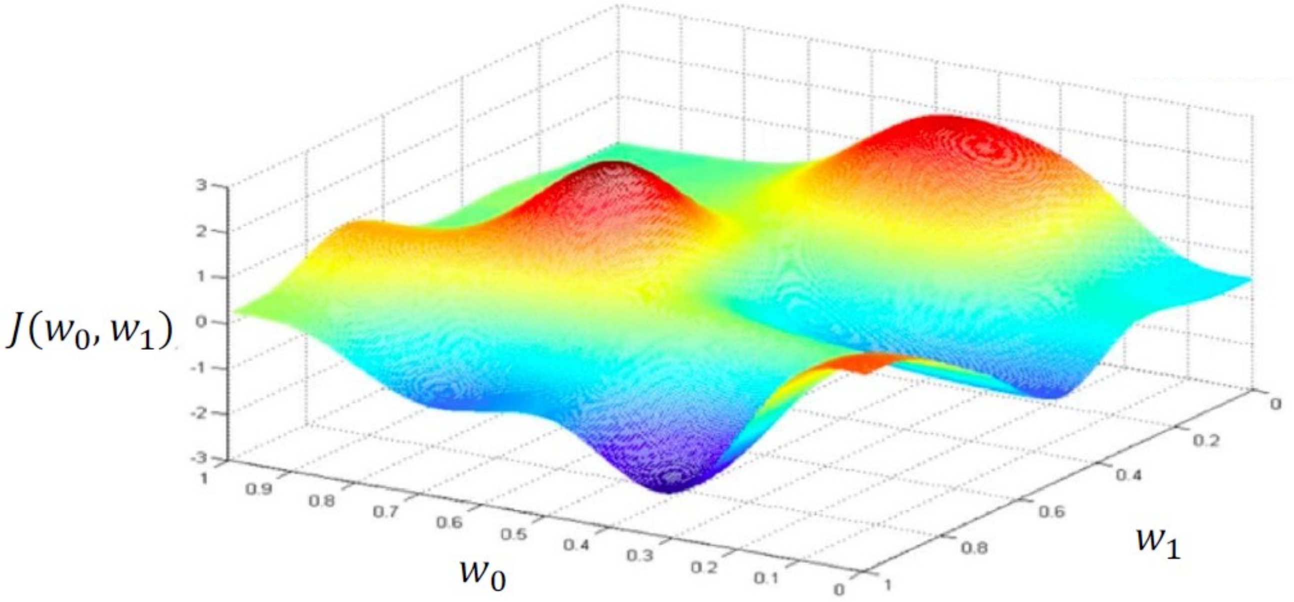
$$\min_{\beta}\left\{\sum_{i=1}^{n}(y_i - X_i\beta)^2 + \lambda\sum_{j=1}^{p}|\beta_j|\right\}$$

Training in supervised ML, typically involves minimizing a loss!

We seek for a set of weights that achieve minimal loss:

$$W^* = \underset{W}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$$

$$W^* = \underset{W}{\operatorname{argmin}}\, J(W)$$



Generic loss $\mathcal{L}$, it can be MSE, cross-entropy (we'll see it in classification), ...

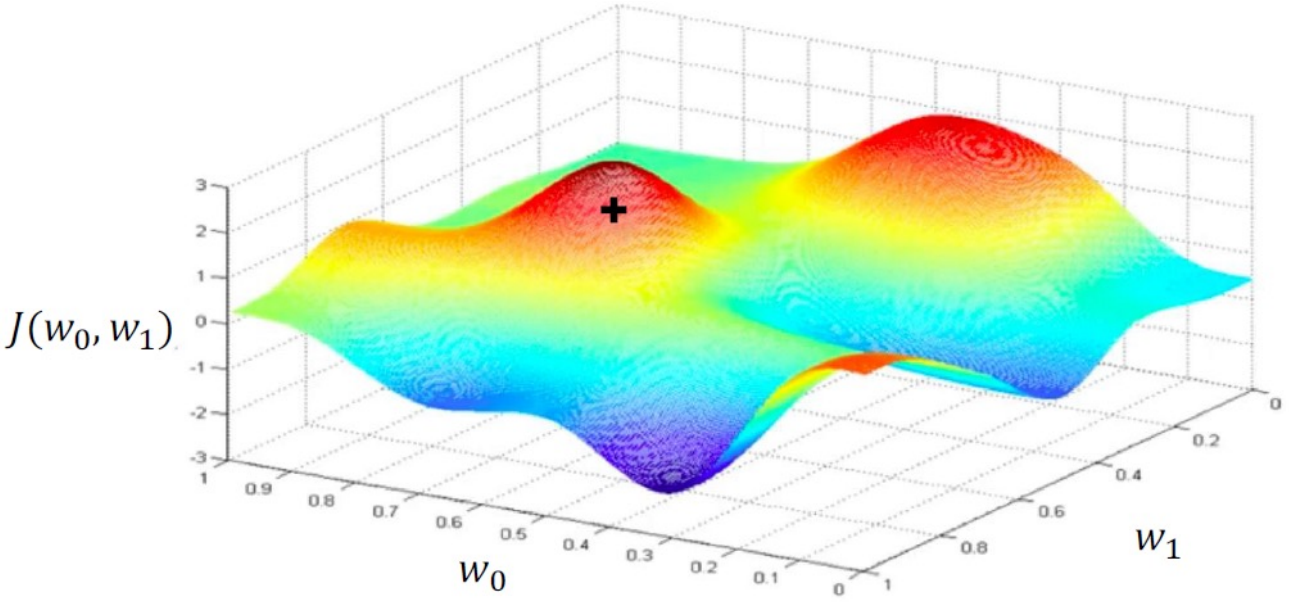
$$\min_{\beta} \left\{ \sum_{i=1}^{n} (y_i - X_i\beta)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

Training in supervised ML, typically involves minimizing a loss!

We seek for a set of weights that achieve minimal loss:

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$



Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$

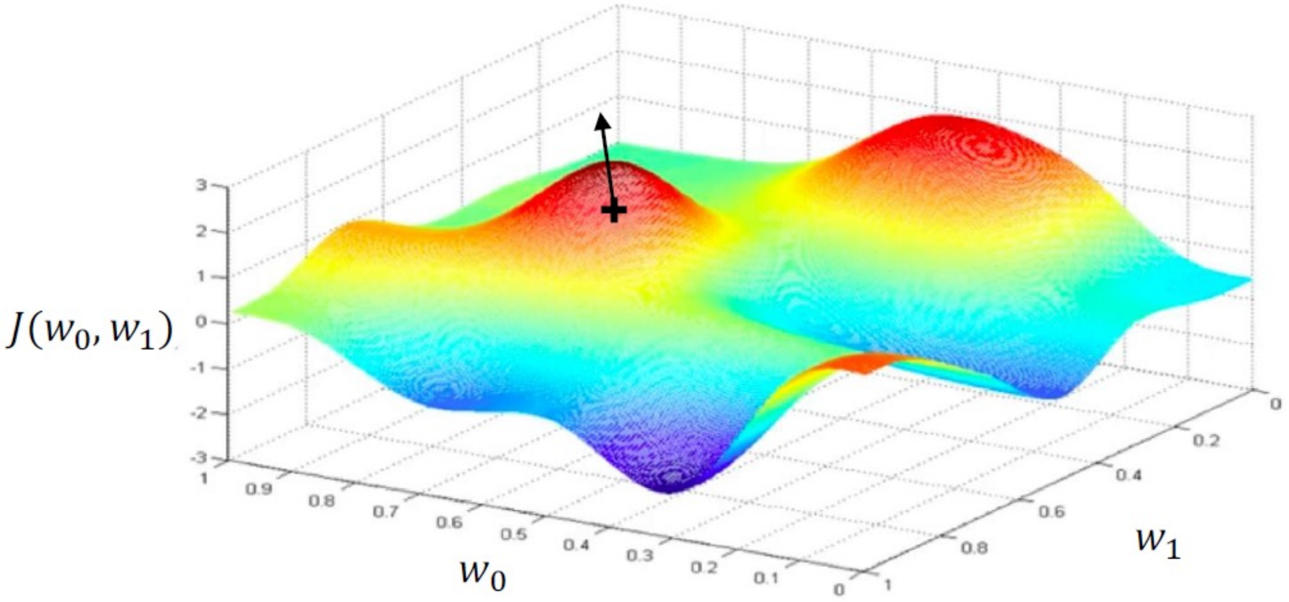
MIT *Introduction to Deep Learning* http://introtodeeplearning.com

Training in supervised ML, typically involves minimizing a loss!

We seek for a set of weights that achieve minimal loss:

$$W^* = \underset{W}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$$

$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$



Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$

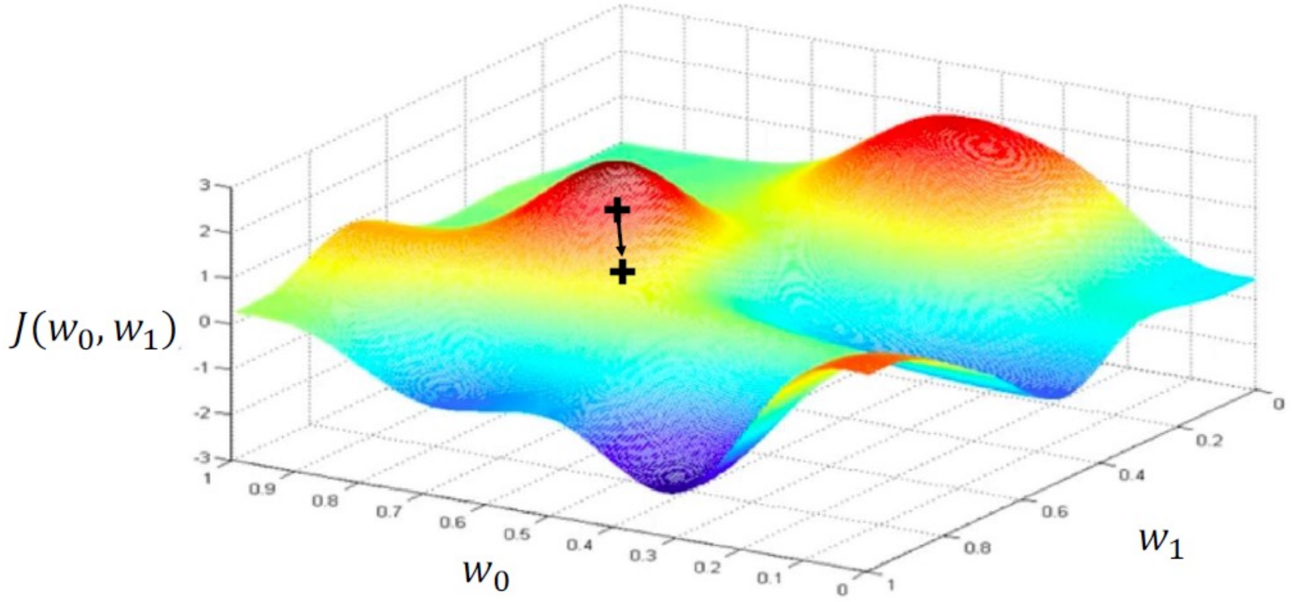
Compute gradient, $\frac{\partial J(W)}{\partial W}$

Training in supervised ML, typically involves minimizing a loss!

We seek for a set of weights that achieve minimal loss:

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$



Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$

Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$

Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$

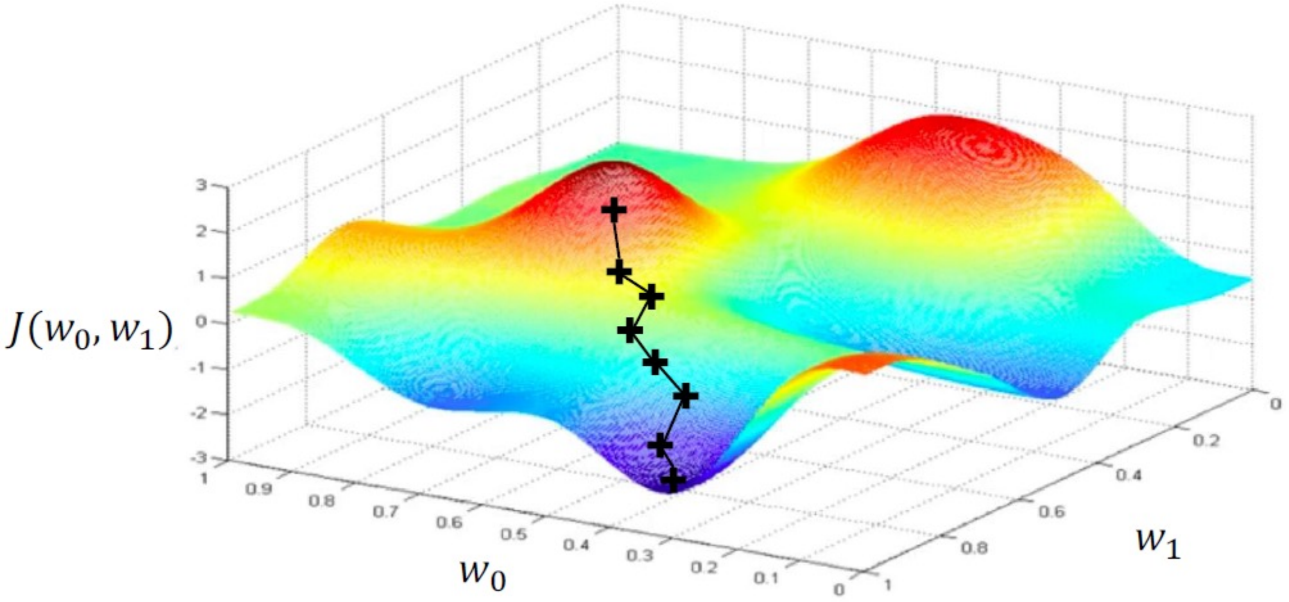
MIT *Introduction to Deep Learning* http://introtodeeplearning.com

Training in supervised ML, typically involves minimizing a loss!

We seek for a set of weights that achieve minimal loss:

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \, J(\boldsymbol{W})$$



Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

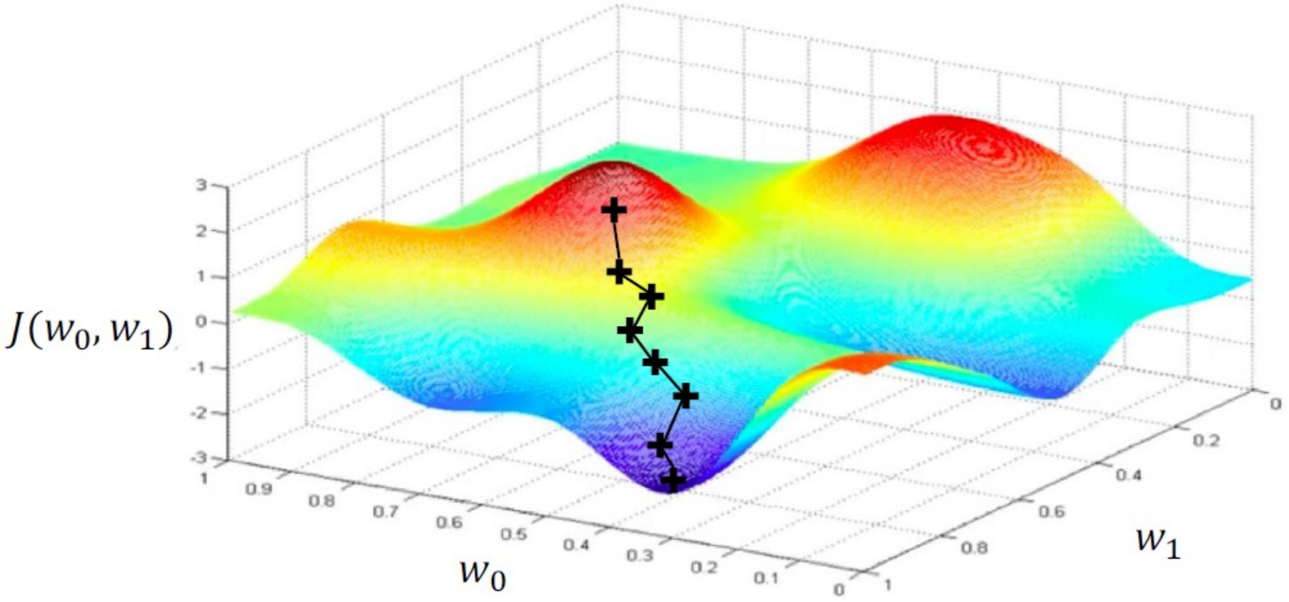
MIT *Introduction to Deep Learning* http://introtodeeplearning.com

Training in supervised ML, typically involves minimizing a loss!

We seek for a set of weights that achieve minimal loss:

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} J(\boldsymbol{W})$$



Algorithm (Gradient Descent)

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

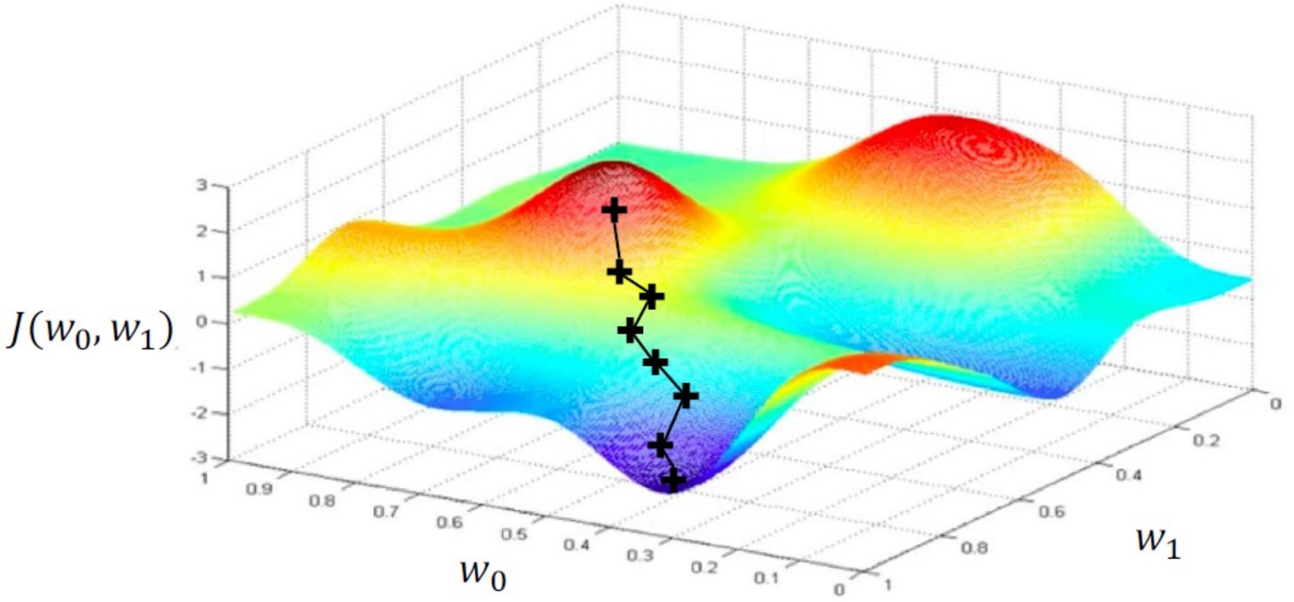
MIT *Introduction to Deep Learning* http://introtodeeplearning.com

Training in supervised ML, typically involves minimizing a loss!

We seek for a set of weights that achieve minimal loss:

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \, J(\boldsymbol{W})$$



Algorithm (Gradient Descent)

1. Initialize weights $\boxed{\text{randomly} \sim \mathcal{N}(0, \sigma^2)}$
2. Loop until convergence:
3. $\boxed{\text{Compute gradient, } \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \boxed{\eta} \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Learning Rate

Gradient Descent: initial choices of the parameters

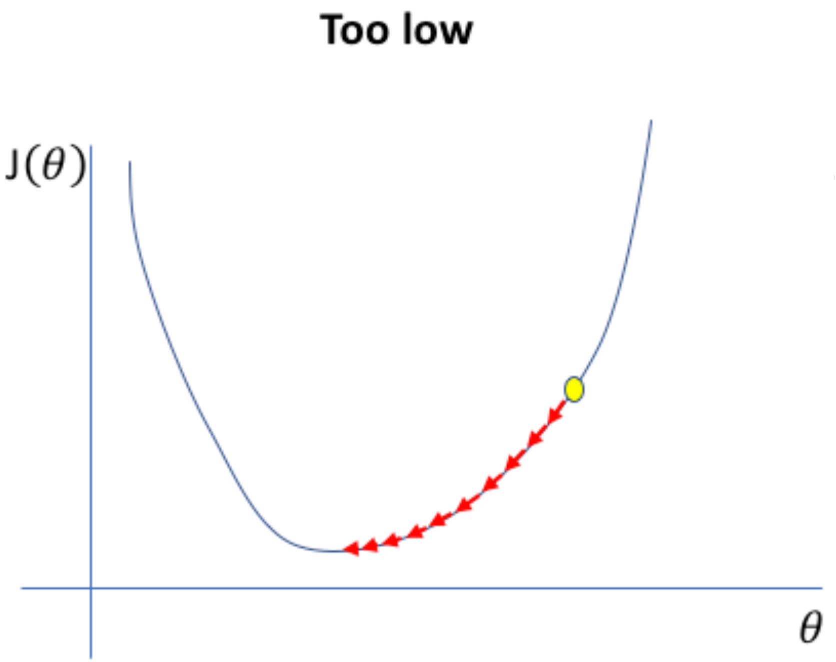
- Choices of the initial parameters can be completely random, however

- However, there are some guidelines to speed up the procedure:

1. If λ is large → Use zero initialization
2. If features are highly correlated → Use Ridge solution
3. If features are independent and λ is small → Use OLS solution
4. If unsure → Use Ridge or OLS, as they provide reasonable starting points.

Gradient Descent: learning rate

- The learning rate η is critical in gradient descent. If it's too large, the algorithm diverges; if it's too small, convergence is slow.

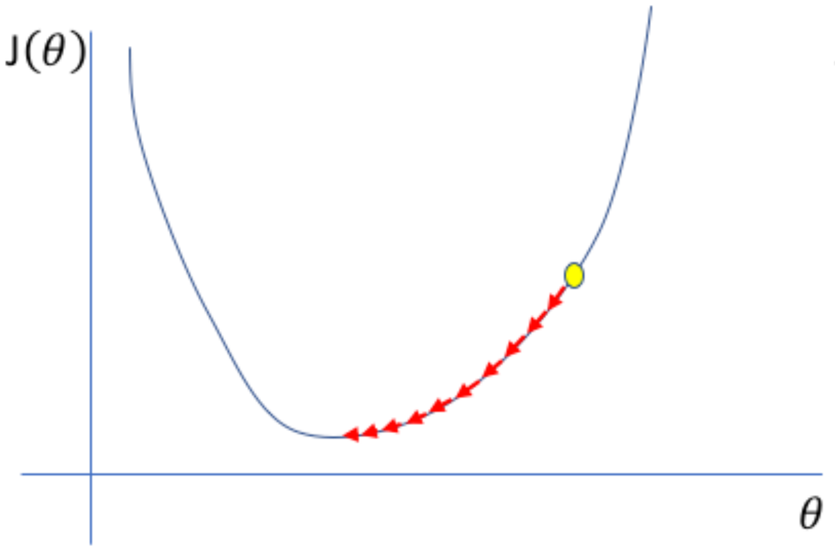


A small learning rate requires many updates before reaching the minimum point

Gradient Descent: learning rate

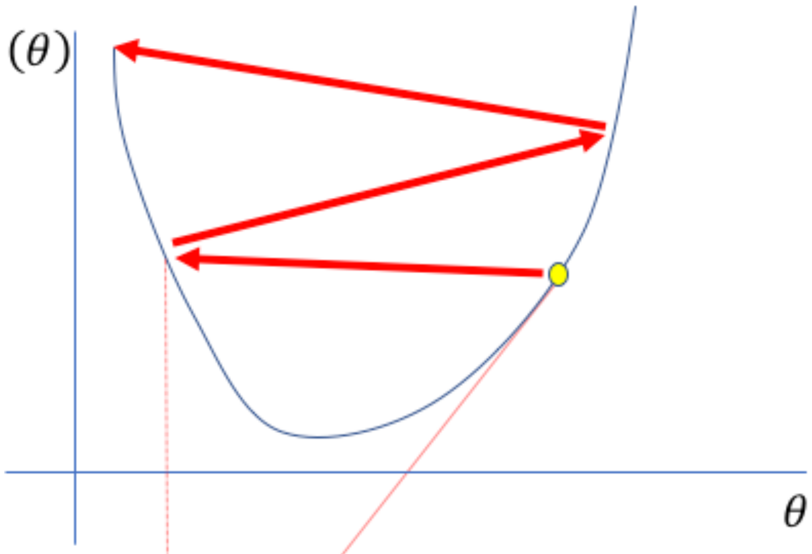
- The learning rate η is critical in gradient descent. If it's too large, the algorithm diverges; if it's too small, convergence is slow.

Too low



A small learning rate requires many updates before reaching the minimum point

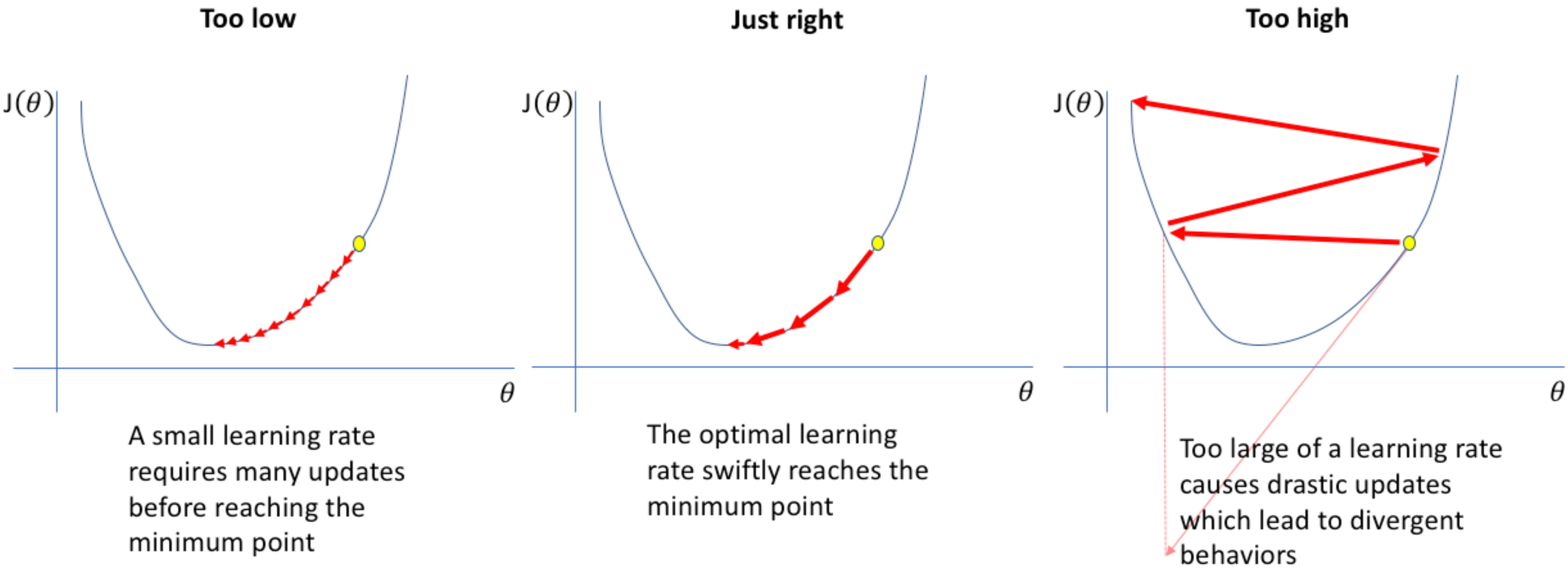
Too high



Too large of a learning rate causes drastic updates which lead to divergent behaviors

Gradient Descent: learning rate

- The learning rate η is critical in gradient descent. If it's too large, the algorithm diverges; if it's too small, convergence is slow.



Gradient Descent: learning rate

- The learning rate η is critical in gradient descent. If it's too large, the algorithm diverges; if it's too small, convergence is slow.

- Some guidelines:

1. Upper bound – The learning rate should satisfy $\eta < \frac{1}{L}$

where *L* is the largest eigenvalue of *X'X* (also called the Lipschitz constant)

2. Adaptive learning rate - We can adjust η dynamically: $\eta_t = \frac{\eta_0}{1 + \gamma t}$

- η0 is the initial learning rate

- t is the current iteration

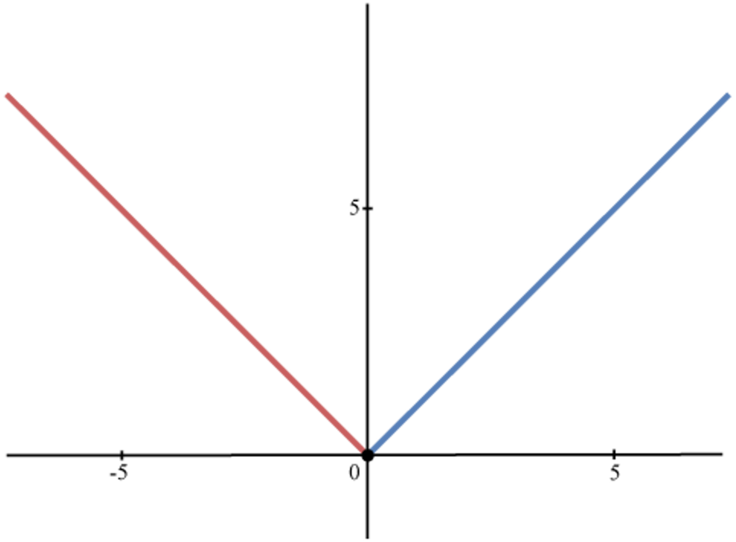
- Γ is a decay parameter (e.g., γ=0.01).

Gradient Descent: computing the gradient

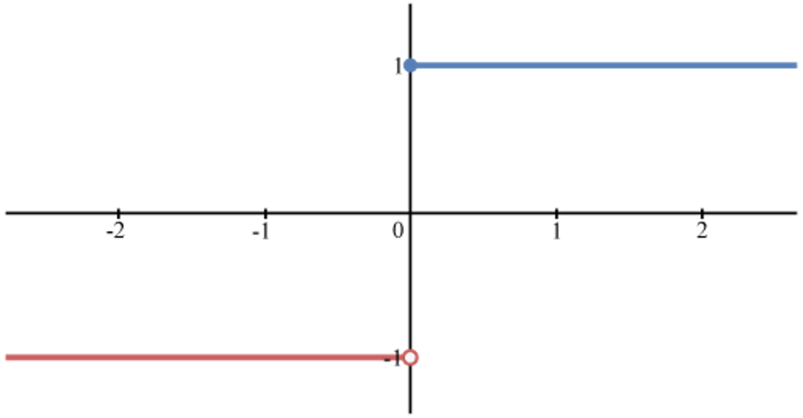
- Derivative of LASSO is not defined in 0

$$J(w) = \sum_{i=1}^{n}(y_i - x_i^T w)^2 + \lambda \sum_{j=1}^{p} |w_j|$$

$$\frac{d}{dw}|w| = \begin{cases} +1, & w > 0 \\ -1, & w < 0 \\ ???, & w = 0 \end{cases}$$



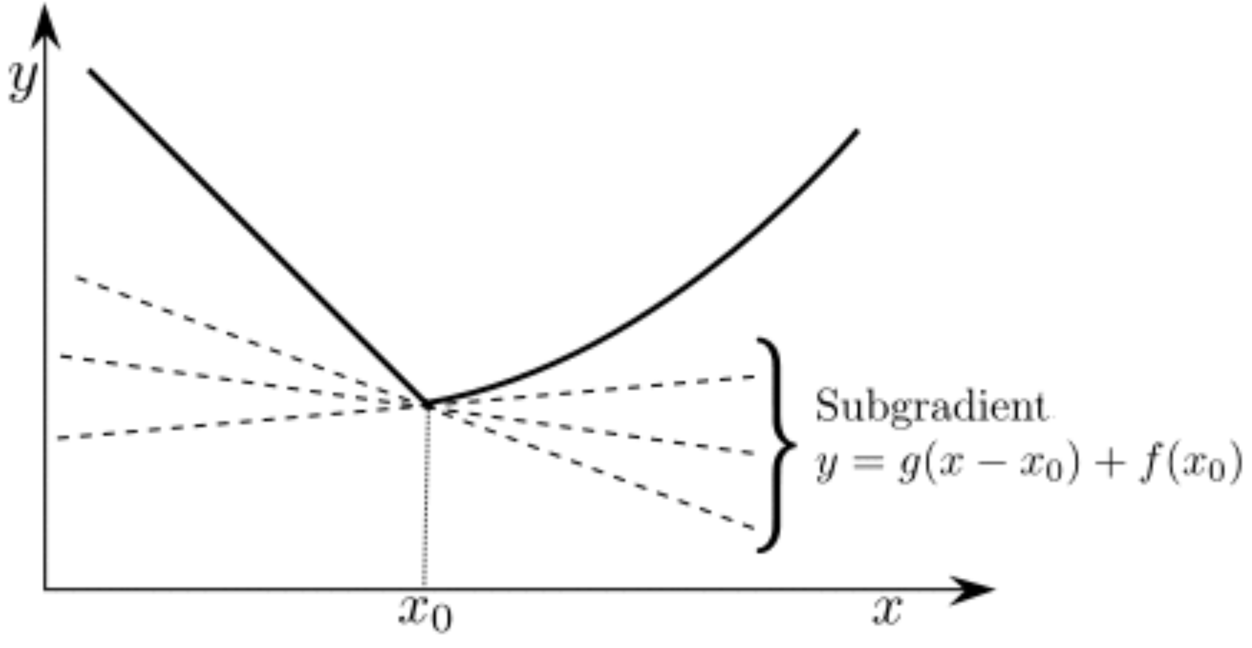
$$f(x) = |x| = \begin{cases} x & x \geq 0 \\ -x & x < 0 \end{cases}$$



$$f'(x) = \begin{cases} 1 & x > 0 \\ -1 & x < 0 \end{cases}$$

Gradient Descent: computing the gradient

- Derivative of LASSO is not defined in 0, we need to use a subgradient g



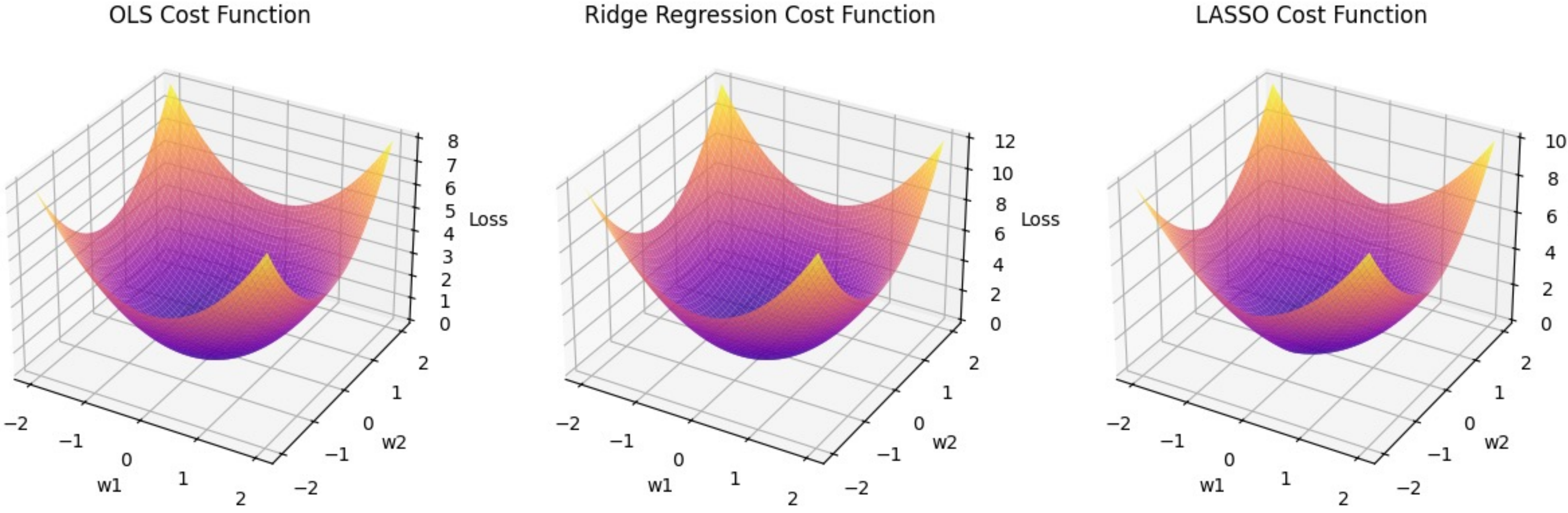
$$J(w) = \sum_{i=1}^{n}(y_i - x_i^T w)^2 + \lambda \sum_{j=1}^{p} |w_j|$$

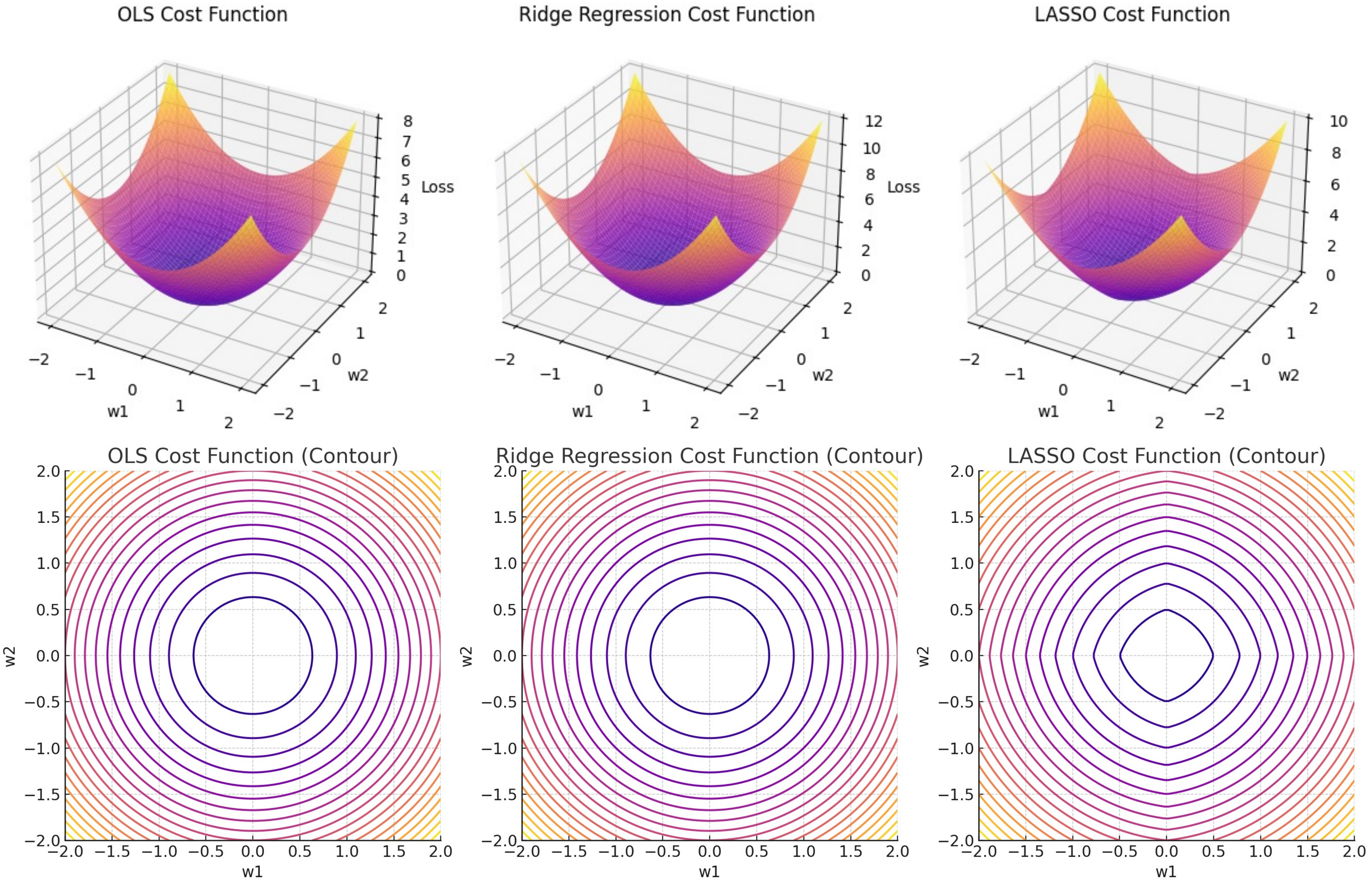
$$\boxed{f(w') \geq f(w) + g(w)(w' - w)}$$

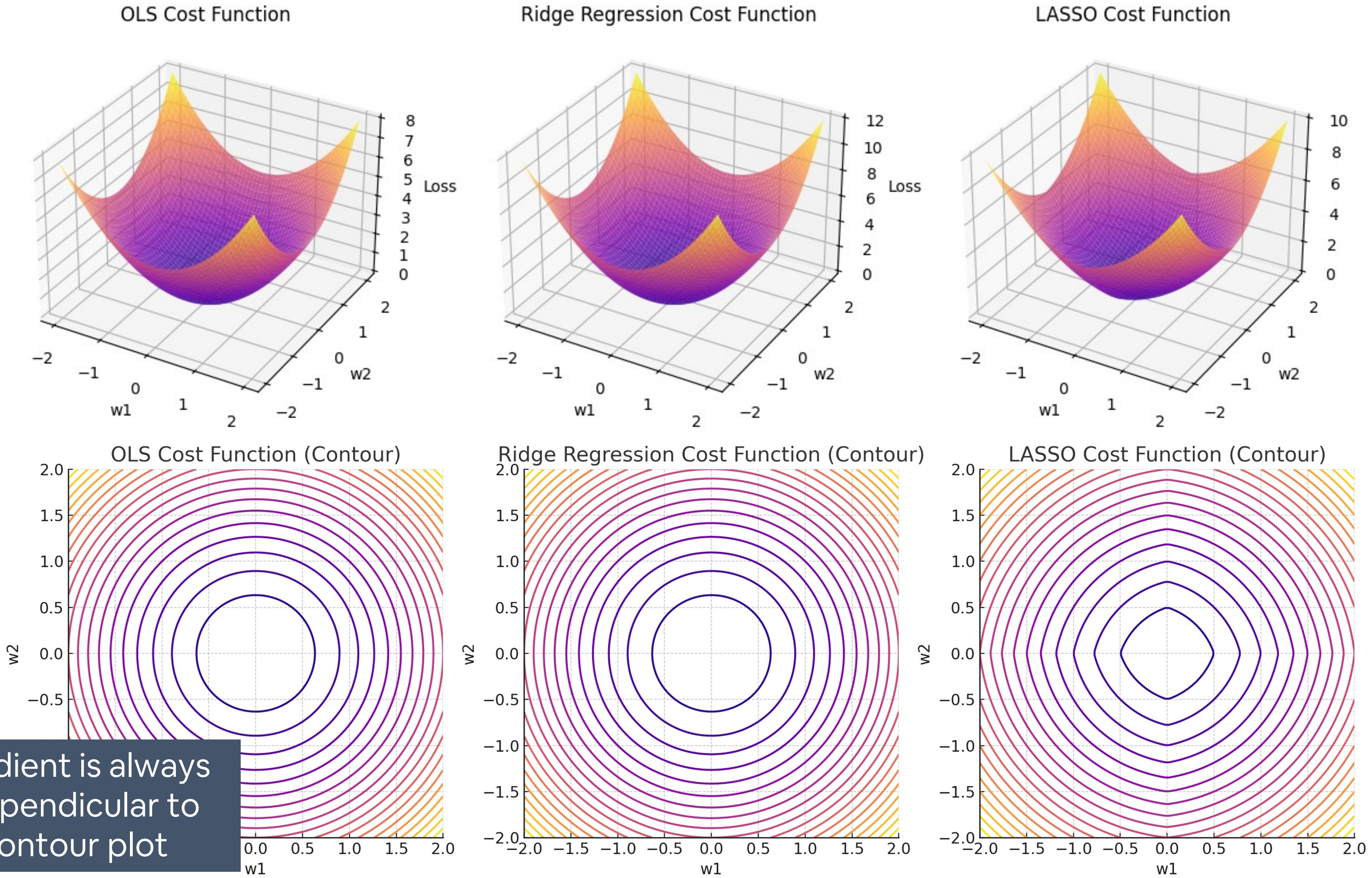
$$\frac{d}{dw}|w| = \begin{cases} +1, & w > 0 \\ -1, & w < 0 \\ ???, & w = 0 \end{cases}$$

$$\boxed{\partial f(w) = \begin{cases} +1, & w > 0 \\ -1, & w < 0 \\ \text{any value in } [-1, 1], & w = 0 \end{cases}}$$

Shape of cost function in OLS, RR & LASSO



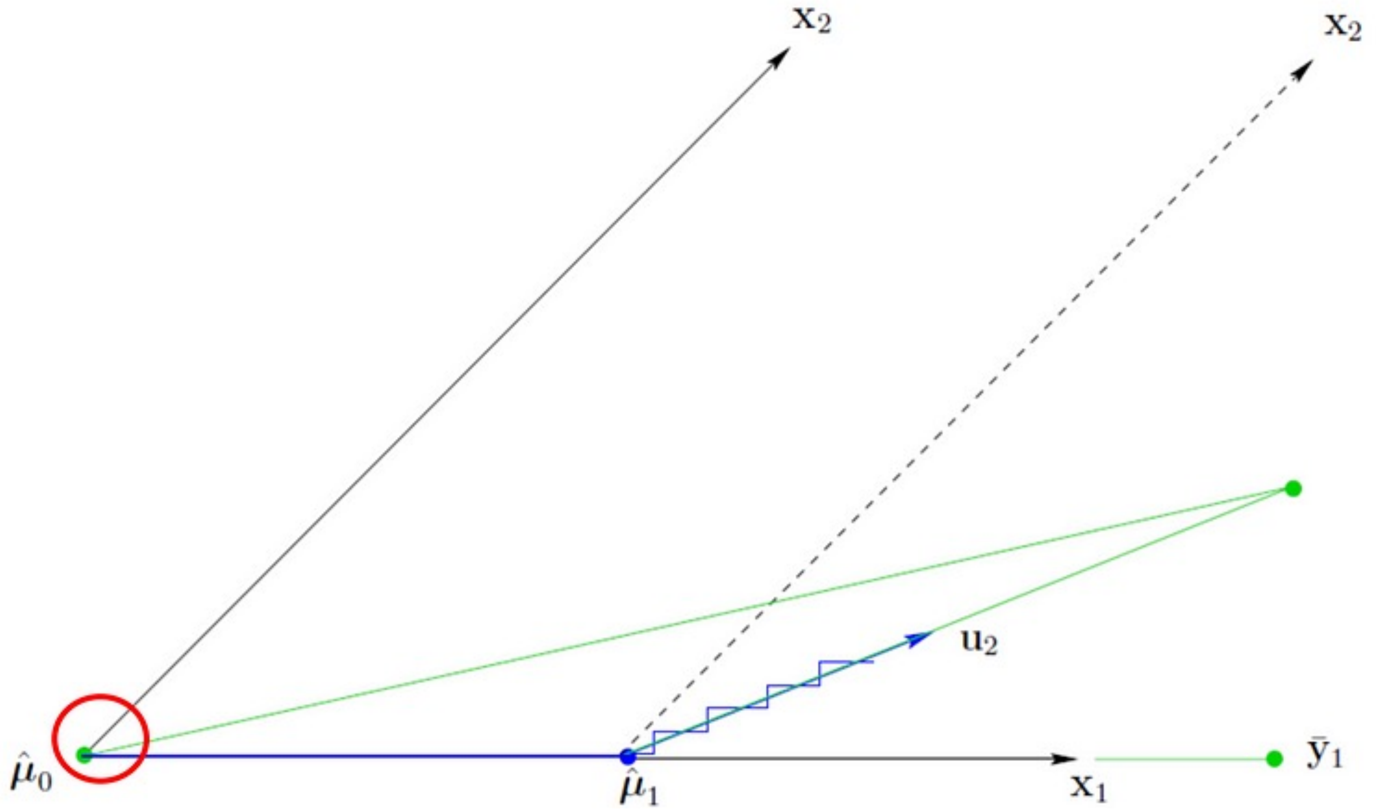

OLS Cost Function
Ridge Regression Cost Function
LASSO Cost Function
Loss
w1
w2
OLS Cost Function (Contour)
Ridge Regression Cost Function (Contour)
LASSO Cost Function (Contour)



Gradient is always perpendicular to contour plot

Other approaches for solving the LASSO

- Least Angle Regression (LARS)
- Subcoordinate Methods



Ridge vs LASSO

✅ Pros

- Prevents overfitting: It shrinks the coefficients, making the model more robust to noise.
- Works well with collinearity: If features are correlated, Ridge distributes weights more evenly.
- Closed–form solution

Ridge vs LASSO

Ridge

✅ Pros

- Prevents overfitting: It shrinks the coefficients, making the model more robust to noise.
- Works well with collinearity: If features are correlated, Ridge distributes weights more evenly.
- Closed–form solution

LASSO

✅ Pros

- Performs feature selection + sparse model: useful when you want a simpler, interpretable model with fewer nonzero coefficients.
- Works well in high-dimensional settings: Ideal when there are many irrelevant features (e.g., text, genomics).
- Helps with multicollinearity: LASSO automatically selects the most important feature among correlated ones.

Ridge vs LASSO

Ridge

✅ Pros

- Prevents overfitting: It shrinks the coefficients, making the model more robust to noise.
- Works well with collinearity: If features are correlated, Ridge distributes weights more evenly.
- Closed–form solution

❌ Cons

- No feature selection: Ridge shrinks coefficients but never eliminates them, so it doesn't provide a sparse model.
- Harder to interpret since all features remain, the model may be less interpretable than LASSO.
- Not ideal for sparse Data: if many features are irrelevant, Ridge does not remove them, potentially reducing efficiency.

LASSO

✅ Pros

- Performs feature selection + sparse model: useful when you want a simpler, interpretable model with fewer nonzero coefficients.
- Works well in high-dimensional settings: Ideal when there are many irrelevant features (e.g., text, genomics).
- Helps with multicollinearity: LASSO automatically selects the most important feature among correlated ones.

Ridge vs LASSO

Ridge

✅ Pros

- Prevents overfitting: It shrinks the coefficients, making the model more robust to noise.
- Works well with collinearity: If features are correlated, Ridge distributes weights more evenly.
- Closed–form solution

❌ Cons

- No feature selection: Ridge shrinks coefficients but never eliminates them, so it doesn't provide a sparse model.
- Harder to interpret since all features remain, the model may be less interpretable than LASSO.
- Not ideal for sparse Data: if many features are irrelevant, Ridge does not remove them, potentially reducing efficiency.

LASSO

✅ Pros

- Performs feature selection + sparse model: useful when you want a simpler, interpretable model with fewer nonzero coefficients.
- Works well in high-dimensional settings: Ideal when there are many irrelevant features (e.g., text, genomics).
- Helps with multicollinearity: LASSO automatically selects the most important feature among correlated ones.

❌ Cons

- Non-differentiable at zero and higher computational cost
- Unstable with correlated features: LASSO tends to arbitrarily select one feature from a group of correlated ones and ignore the rest

Ridge vs LASSO

Ridge

✅ Pros

- Prevents <u>overfitting</u>: It shrinks the coefficients, making the model more robust to noise.

- Works well with collinearity: If features are correlated, Ridge distributes weights more evenly.

- Closed–form solution

❌ Cons

- No <u>feature selection</u>: Ridge shrinks coefficients but never eliminates them, so it doesn’t provide a sparse model.

- Harder to interpret since all features remain, the model may be less interpretable than LASSO.

- Not ideal for sparse Data: if many features are irrelevant, Ridge does not remove them, potentially reducing efficiency.

LASSO

✅ Pros

- Performs feature selection + sparse model: useful when you want a simpler, interpretable model with fewer nonzero coefficients.

- Works well in high-dimensional settings: Ideal when there are many irrelevant … s (e.g., text, genomics).

… ith multicollinearity: LASSO … tically selects the most important feature among correlated ones.

❌ Cons

- Non-differentiable at zero and higher computational cost

- Unstable with correlated features: LASSO tends to arbitrarily select one feature from a group of correlated ones and ignore the rest

What if I cannot choose?

Elastic Net: A Hybrid of Ridge and LASSO

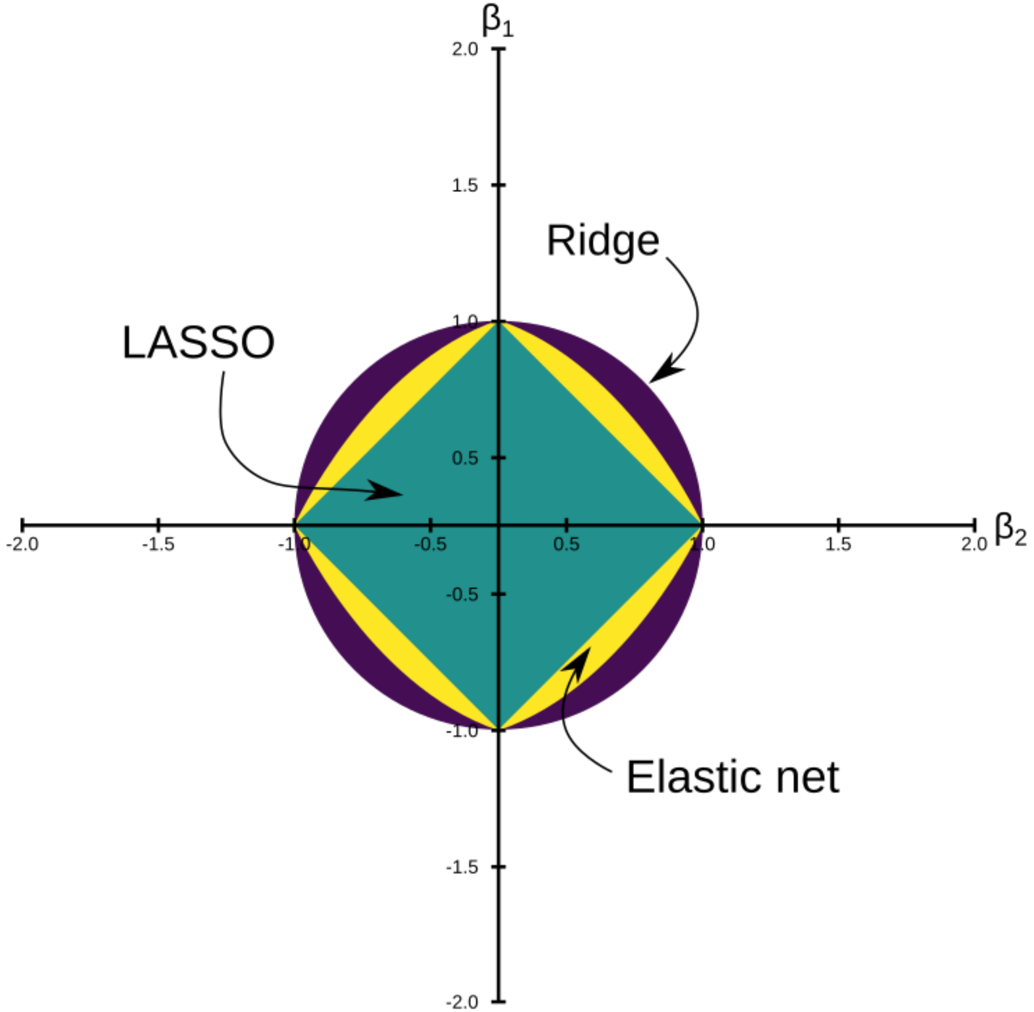
- Elastic Net is a regularized regression method that combines Ridge (L2) and LASSO (L1) penalties to overcome their individual weaknesses.

$$J(w) = \sum_{i=1}^{n} (y_i - x_i^T w)^2 + \lambda_1 \sum_{j=1}^{p} |w_j| + \lambda_2 \sum_{j=1}^{p} w_j^2$$

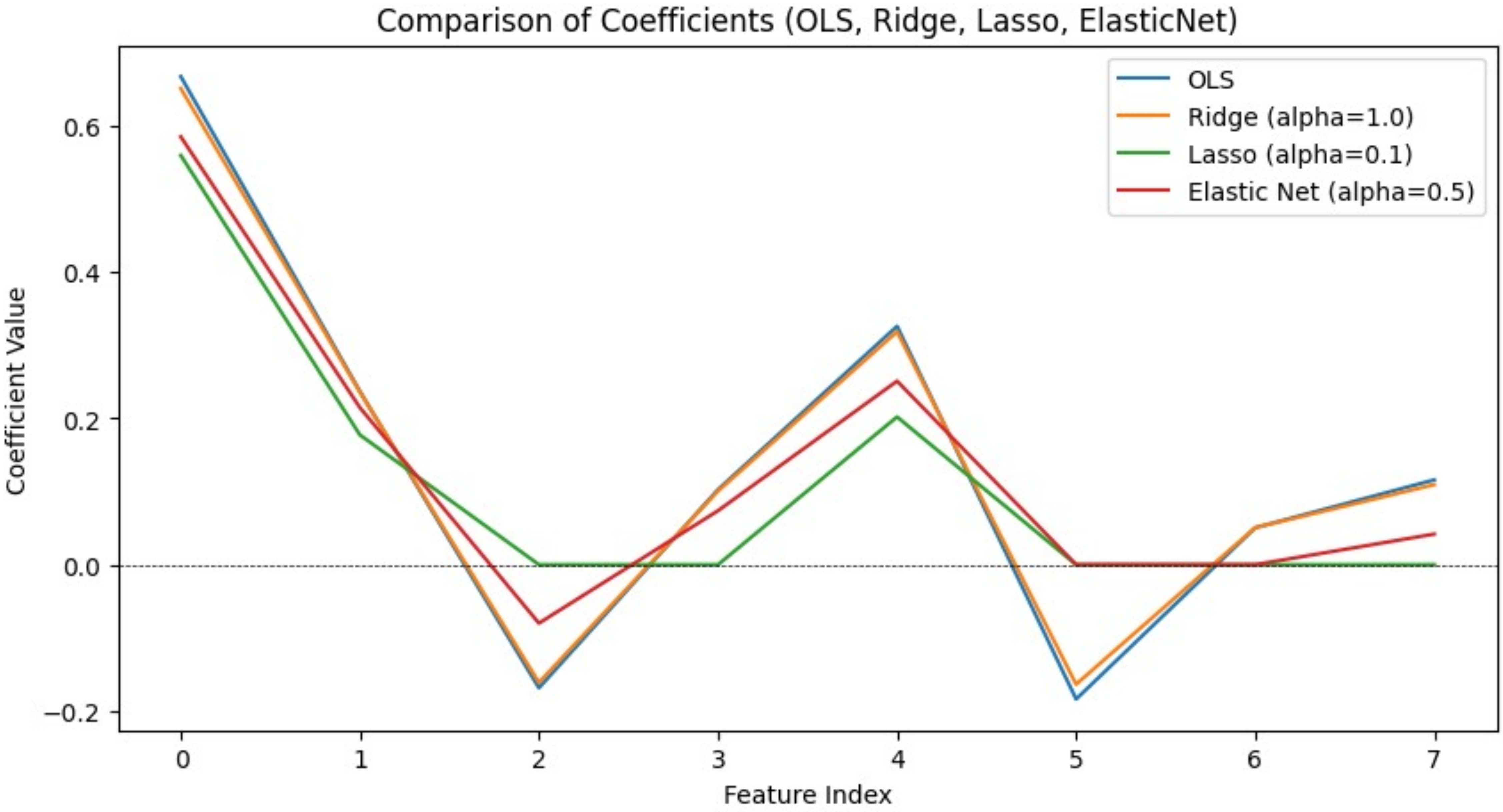
$$\lambda_1 = \alpha\lambda, \quad \lambda_2 = (1-\alpha)\lambda$$

α=1 ⟶ Pure LASSO

α=0 ⟶ Pure Ridge



Elastic Net: Prostate dataset



Thank you!

Gian Antonio Susto