Lecture 4 – Supervised Learning

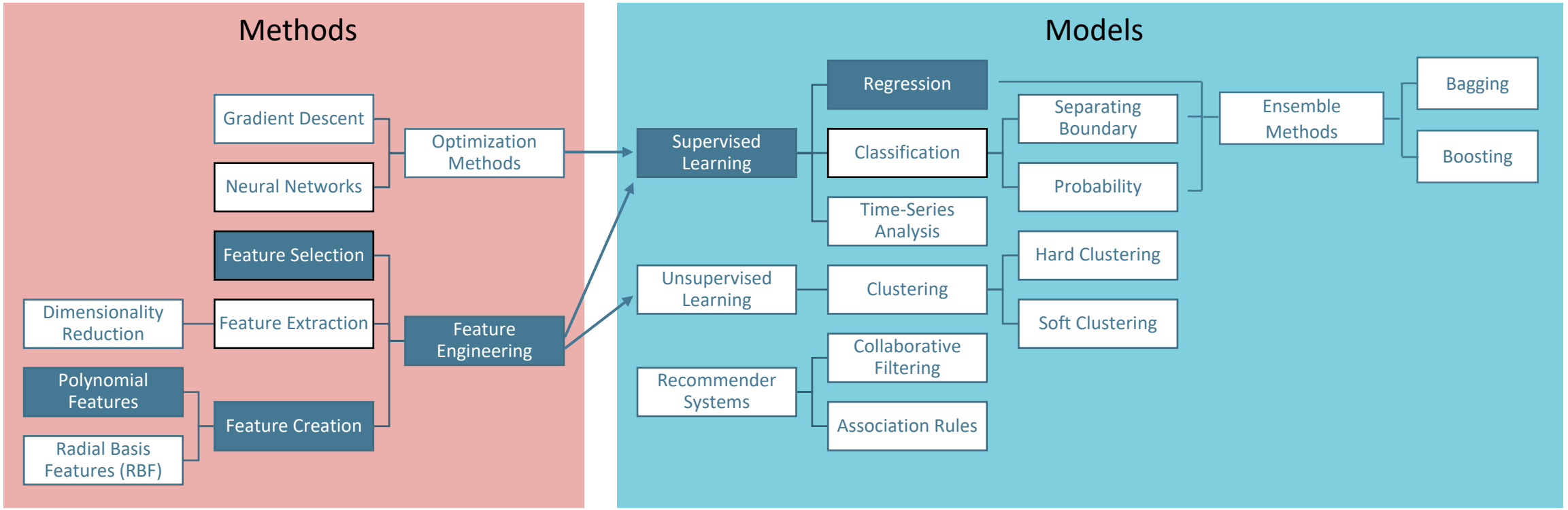
Metrics, Regularization

Information Systems for Sustainable Society (is3) | WiSo Faculty | Univ.-Prof. Dr. Wolfgang Ketter | 10/16/24

Agenda

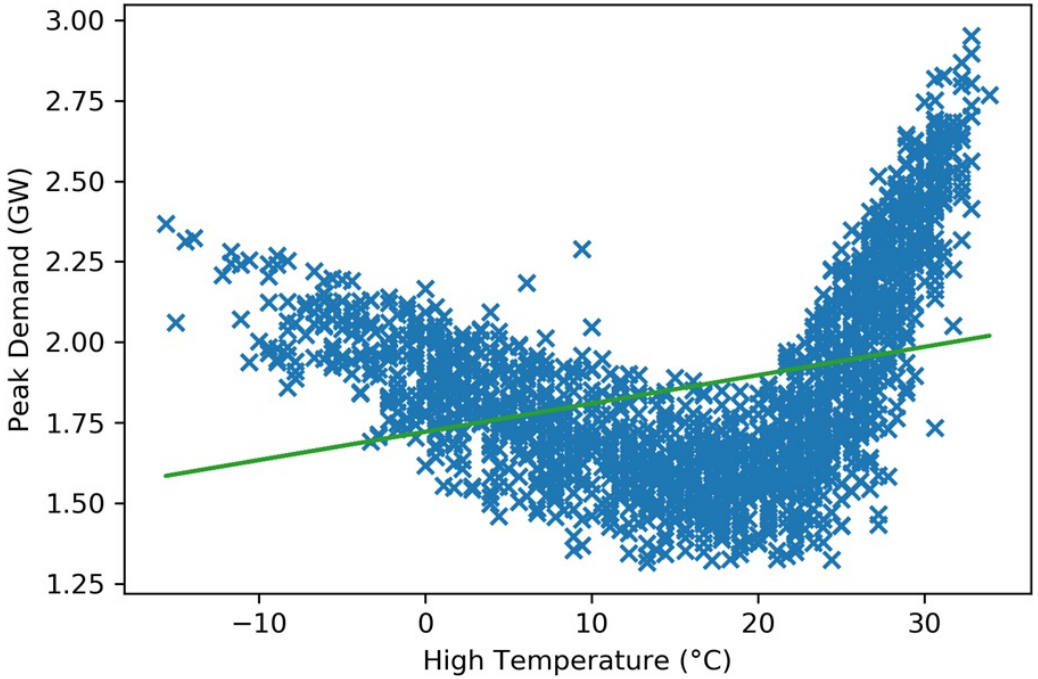

Evaluating Regression Model Performance
Regularization

Topic Structure – Today's Lecture

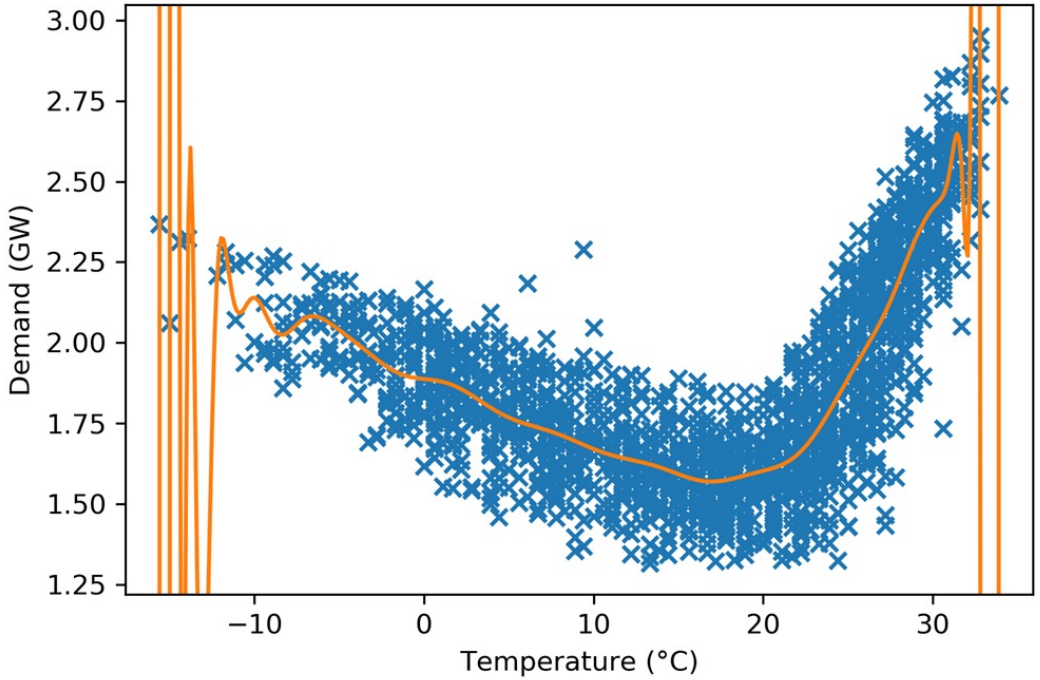


There are two types of situations that we want to avoid: Under- and Overfitting

Underfitting



Overfitting



There are two types of situations that we want to avoid: Under- and Overfitting

Underfitting

- The 1-degree polynomial passes straight through the data and underfits it
- Underfitting means high training and high testing error due to an excessively simplistic formulation
- An underfit model has **low variance and high bias**
  - Variance refers to how much the model is dependent on the training data (i.e. how flexibel it is)
  - Bias describes how strong the assumption about the functional realtionship of the target feature and the predictors is

Overfitting

- The 100-degree polynomial is excessively flexible and passes almost directly through all test data points – It essentially memorizes the entire data including the noise
- Overerfitting means extremely low training loss but high testing loss due to an excessively flexible formulation that does not generalize to new data
- An overerfit model **has high variance and low bias**

Our goal is to develop a model that generalizes well to new examples!

- The problem with the canonical machine learning problem is that we don't **really** care about minimizing this objective on the given data set

$$\underset{\theta}{Minimize} \sum_{i=1}^{m} \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

- What we really care about is how well our function will generalize to **new examples** that we **didn't** use to train the system (but which are drawn from the "same distribution" as the examples we used for training)
- The higher degree polynomials exhibited **overfitting**: they actually have very **low** loss on the training data, but create functions we don't expect to generalize well!

Returning to our electricity demand example – How may overfitting occur?

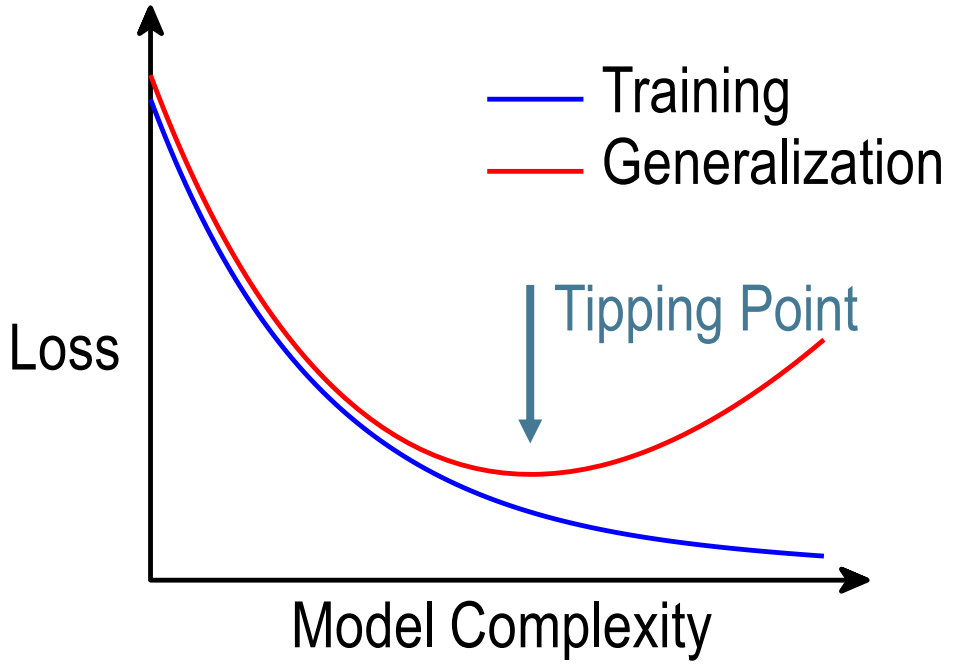
- Suppose we have m examples in our data set
- Further suppose we have $n = m$ features (plus assumption that features are linearly independent)
- Then $X \in \mathbb{R}^{m \times n}$ is a square matrix, and least squares solution is:

$$\theta = (X^T X)^{-1} X^T Y = X^{-1} X^{-T} X^T \, y = X^{-1} \, y$$

and we therefore have $\boldsymbol{X\,\theta = y}$ **(i.e., we fit data exactly)**

- Note that we can **only** perform the above operations when X is square, though if we have **more** features than examples, we can still get an exact fit by simply discarding features
- This is another illustration of why dimensionality can be problematic – It can cause overfitting if insufficient observations are available

Schematic version of under- and overfitting



- As a model becomes more complex (by adding more features, etc.), training loss always decreases; generalization loss decreases to a point, then starts to increase.

How to measure loss? – Absolute Regression Test Metrics

	Name	Term	Description
MSE/ MSD	Mean-Squared Error/Deviation	$\frac{1}{n}\sum_{i=1}^{n} e_i^2$	Average squared difference between the estimated values and what is estimated – Puts large emphasis on large deviations
RMSE	Root-Mean-Squared Error	$\sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}$	Square root of average squared difference between the estimated values and what is estimated (i.e. square root of MSE) – Sets MSE to same units as dependent var.
MAE/ MAD	Mean Absolute Error/ Deviation	$\frac{1}{n}\sum_{i=1}^{n} \lvert e_i \rvert$	Average absolute error – Provides an indication of the absolute deviation (positive or negative) of the responses in the same units as the dependent var.
Average error	Average Error	$\frac{1}{n}\sum_{i=1}^{n} e_i$	Indicates whether the predictions are on average over- or underpredicting the target response

How to measure loss? – Relative Regression Test Metrics

	Name	Term	Description
R^2	R-Squared	$1 - \frac{Sum\ of\ Squares_{res}}{Sum\ of\ Squares_{total}}$	Proportion of variance in the dependent variable that is accounted for by the model
MAPE	Mean absolute percentage error	$100\% \times \frac{1}{n}\sum_{i=1}^{n} \lvert e_i / y_i \rvert$	Gives a percentage score of how predictions deviate on average from the actual values
nRMSE	Normalized RMSE	$\frac{RMSE}{\bar{y}}$	RMSE normalized by the dependent variable to give a ratio of the RMSE compared to the mean of the target value (not very commonly used)
nMAE	Normalized MAE	$\frac{MAE}{\bar{y}}$	MAE normalized by the dependent variable to give a ratio of the MAE compared to the mean of the target value (not very commonly used)

Poll: Absolute vs. relative regression error metrics

Consider the following three situations: Which type of regression metric would you choose for evaluation and why?

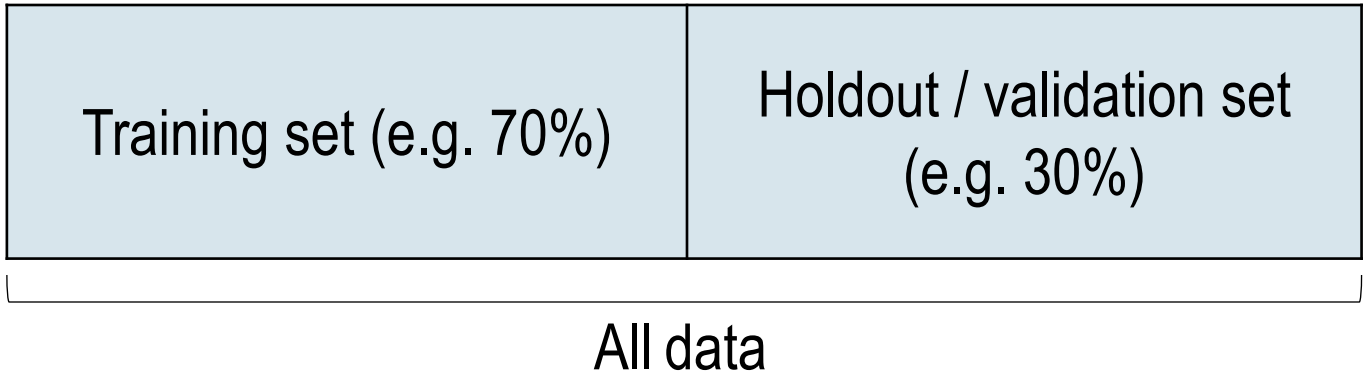
1. Compare two electricity forecasting models that predict **daily and hourly electricity** demand, respectively, for **Pittsburgh.**
2. Compare two electricity forecasting models that predict **hourly electricity** demand for **Pittsburgh and Chicago** respectively.
3. Compare different **hourly electricity forecasting** model types for the city of **Pittsburgh**

Poll: Absolute vs. relative regression error metrics

Consider the following three situations: Which type of regression metric would you choose for evaluation and why?

1. Compare two electricity forecasting models which predict **daily and hourly electricity** demand respectively for the city of **Pittsburgh** **[Relative]**
2. Compare two electricity forecasting models which predict **hourly electricity** demand for **Pittsburgh and Chicago** respectively **[Relative]**
3. Compare different **hourly electricity forecasting** model types for the city of **Pittsburgh** **[Absolute and/or Relative]**

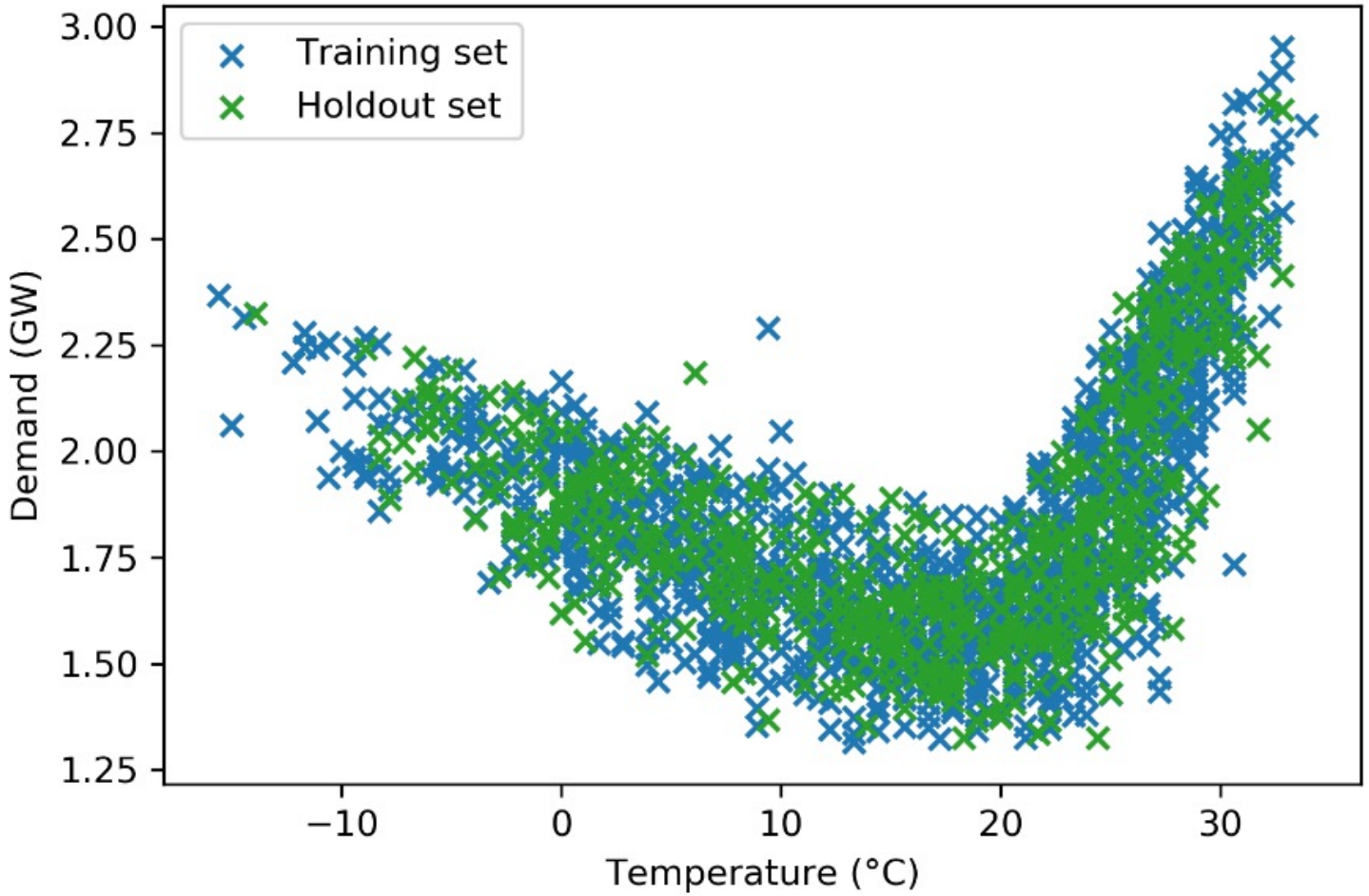
To determine prediction errors we use **cross-validation**



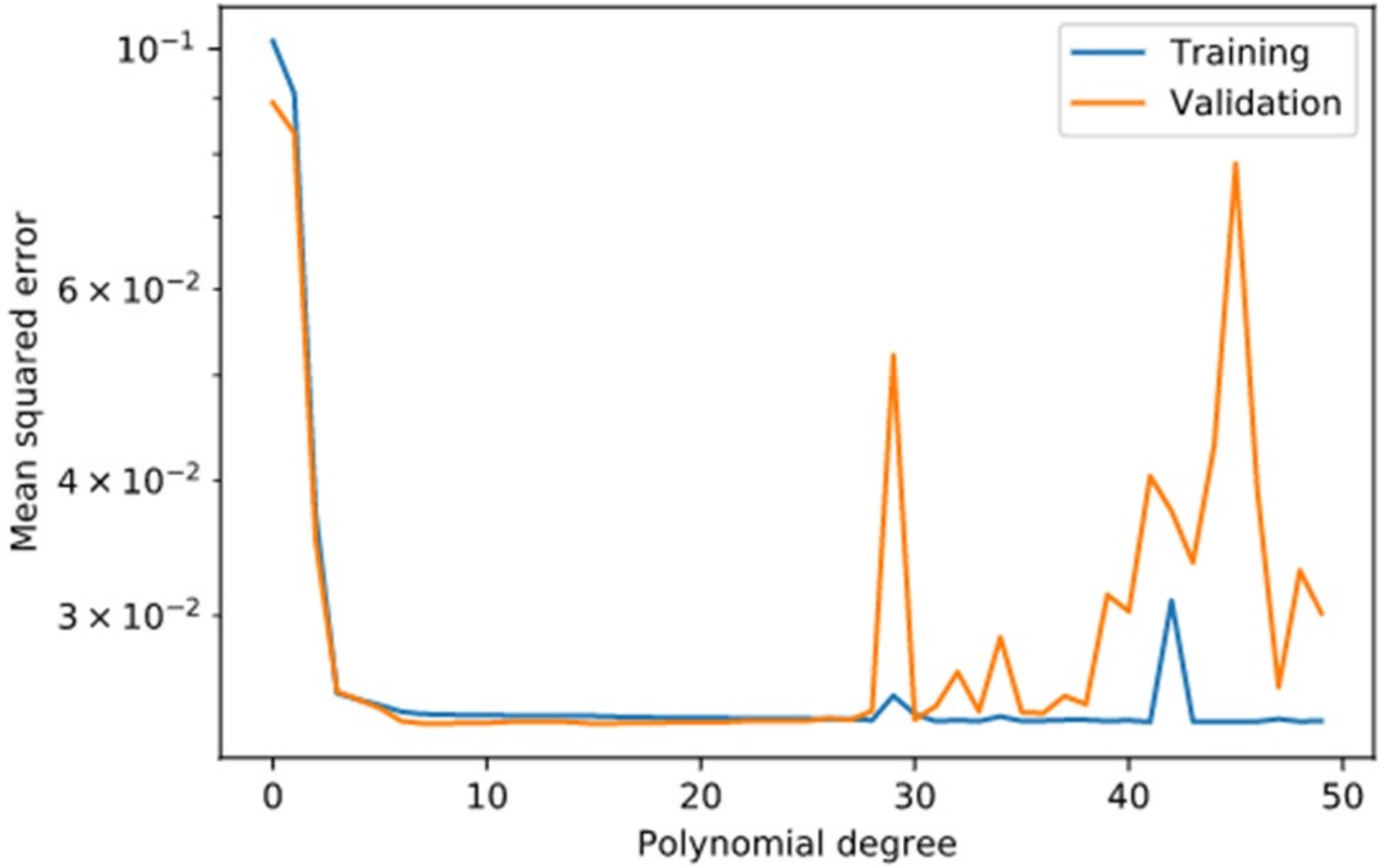
- Although it is difficult to quantify the true generalization error (i.e., the error of these algorithms over the *complete* distribution of possible examples), we can approximate it by **holdout cross-validation**
- Basic idea is to split the data set into a training set and a holdout set
- Train the algorithm on the training set and evaluate on the holdout set

Source: Bishop (2006)

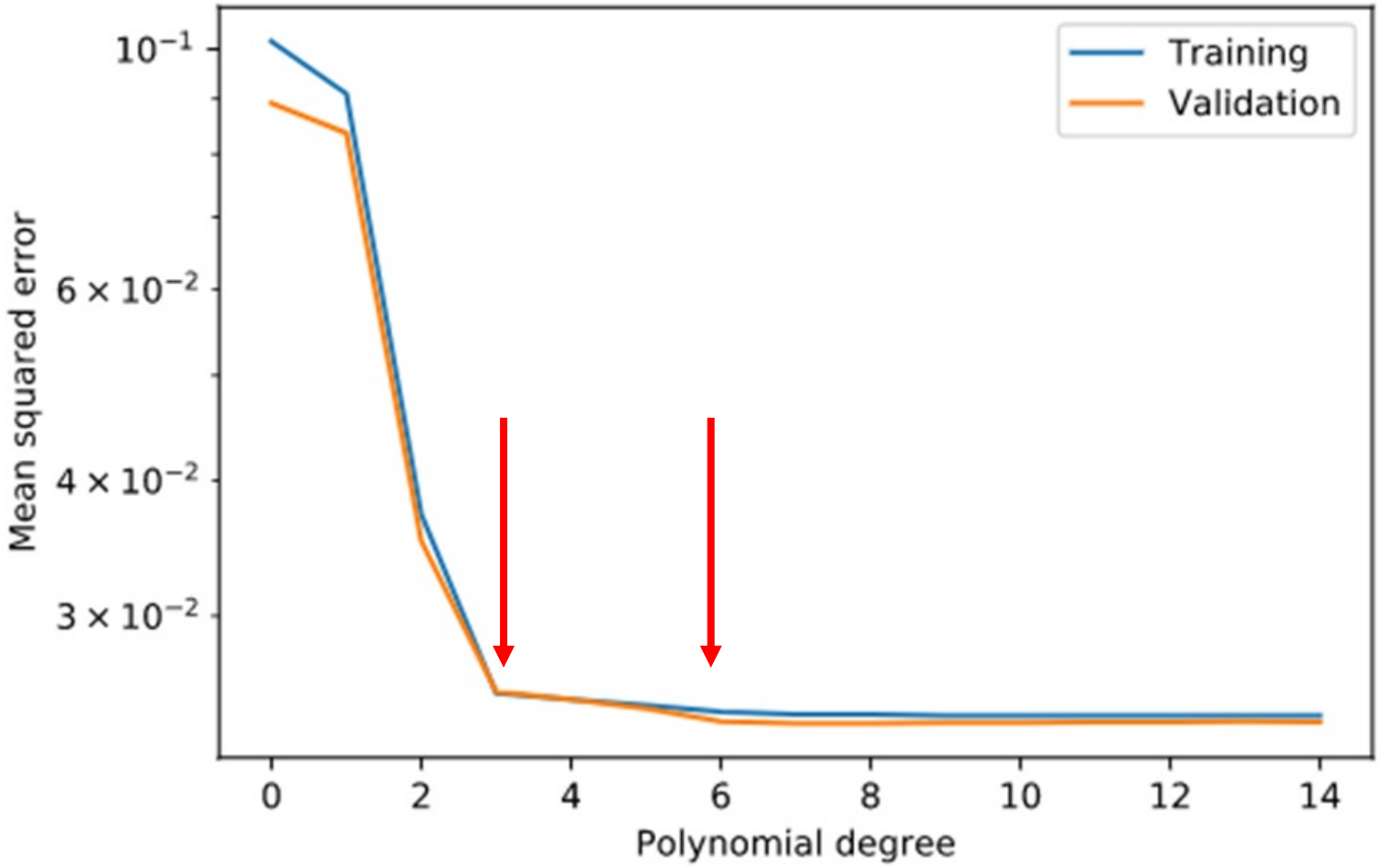
Illustrating cross-validation



Training and cross-validation loss by degree



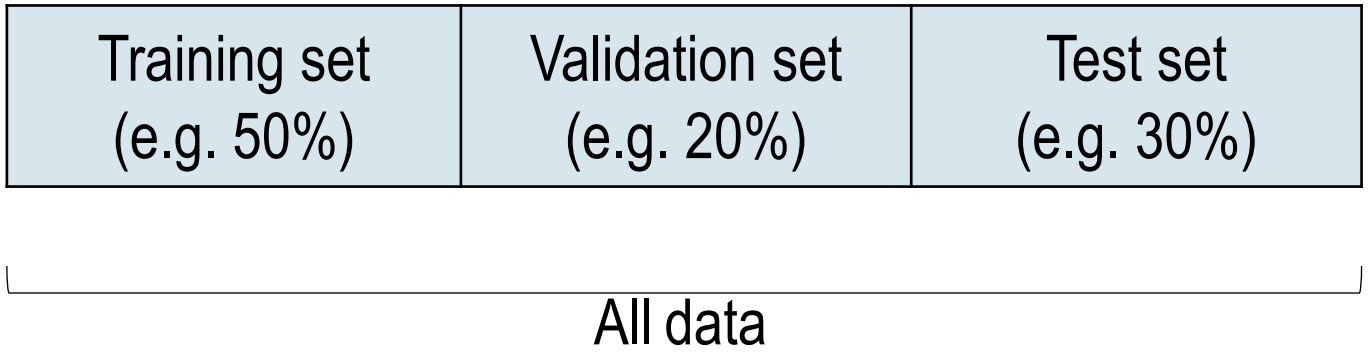
Training and cross-validation loss by degree



Issues with the presented cross-validation methods

- Even though we used a training/holdout split to fit the **parameters**, we are still effectively fitting the **hyperparameters** to the holdout set
  - **Parameters**: These are fitted by the model (in our example the θ)
  - **Hyperparameters**: These are set by the data scientist (in our example e.g. the degree of the polynomial)
- Imagine an algorithm that ignores the training set and makes random predictions; given a large enough hyperparameter search (e.g., over random seed), we could get perfect holdout performance

What to do instead? – Partition your data into training, validation and test set prior to learning a ML model



1. Divide data into training set, holdout set, and test set
2. Train algorithm on training set (i.e., to learn parameters), use holdout set to select hyperparameters
3. (Optional) retrain system on training + holdout
4. Evaluate performance on test set

Source: Bishop (2006)

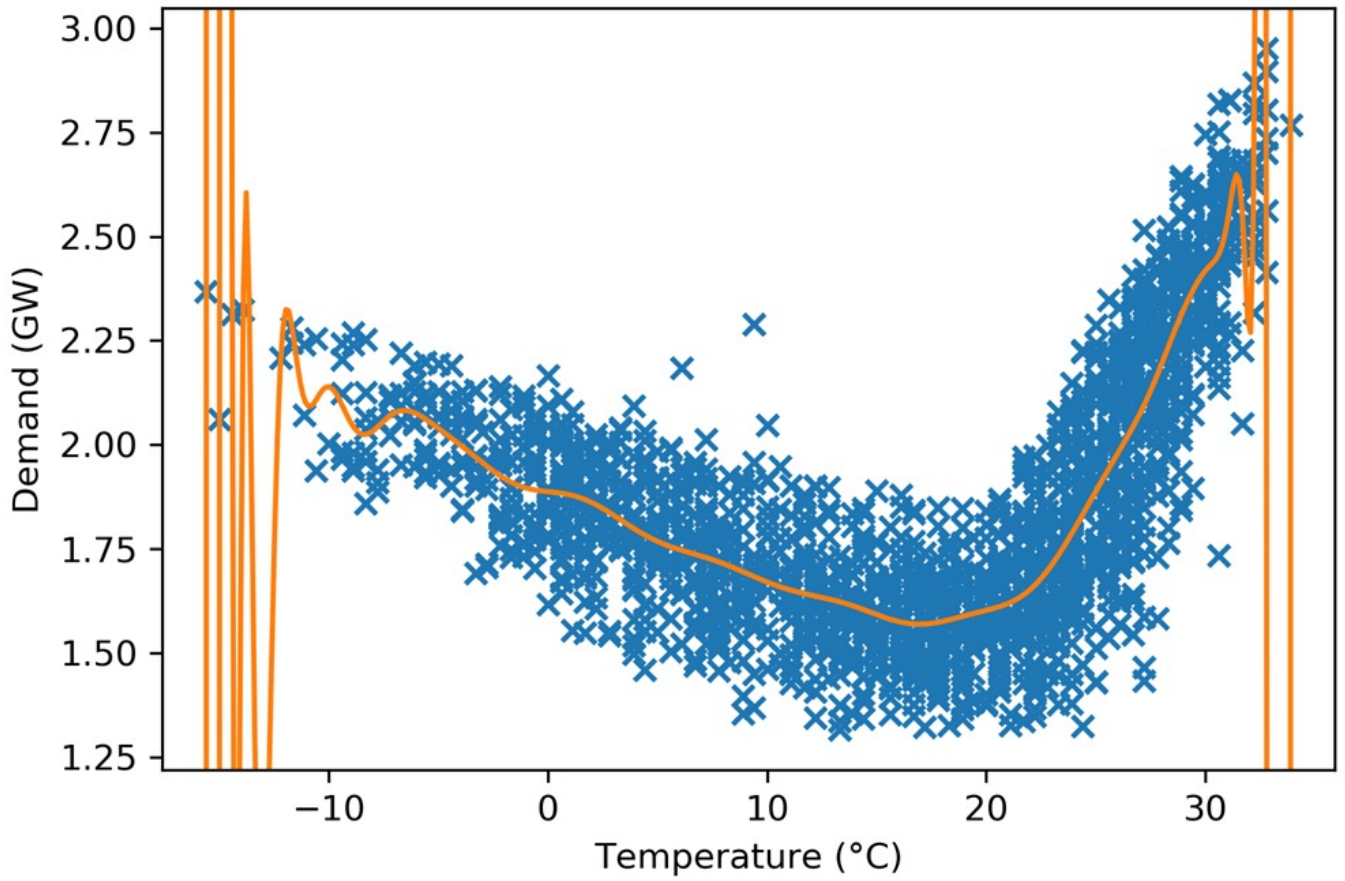
In practice…

- "Leakage" of test set performance into algorithm design decisions in almost always a reality when dealing with any fixed data set (in theory, as soon as you look at test set performance once, you have corrupted that data as a valid set set)
- This is true in research as well as in data science practice
- **The best solutions:** evaluate your system "in the wild" (where it will see truly novel examples) as often a possible; recollect data if you suspect overfitting to test set; look at test set performance sparingly
- An interesting and very active area of research: adaptive data analysis (differential privacy to theoretically guarantee no overfitting)

Agenda

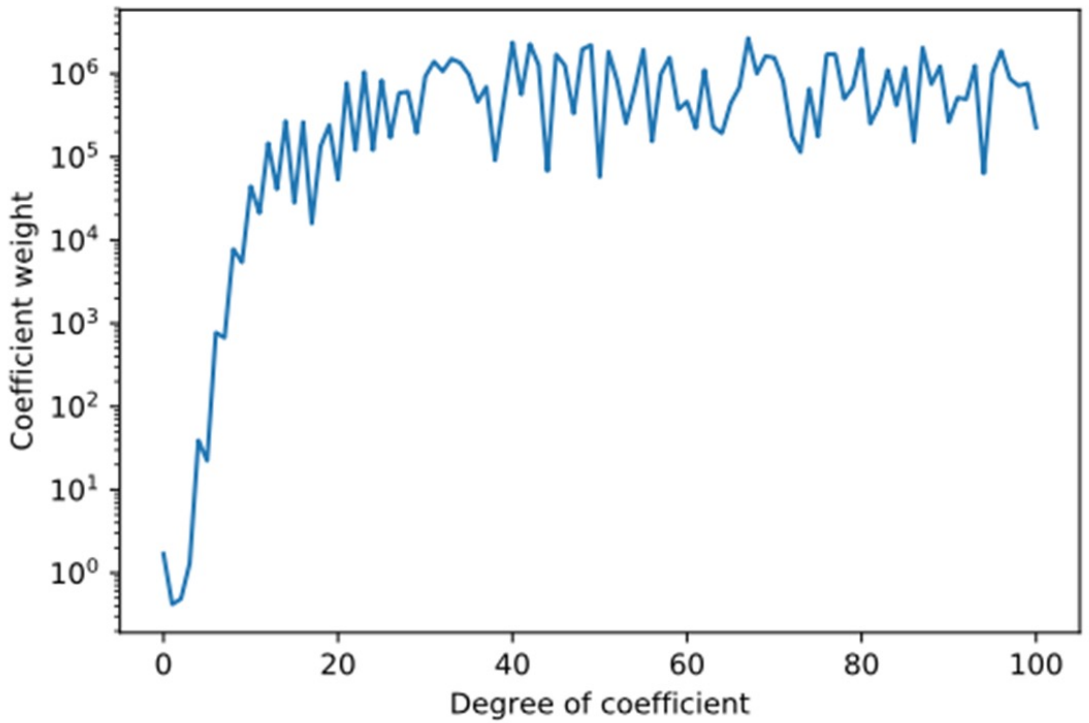

Evaluating Regression Model Performance
Regularization

Let us consider model complexity – Recall this example of our overfit 100 degree polynomial



- We have seen that the **degree of the polynomial** acts as a **natural measure of the "complexity"** of the model, higher degree polynomials are more complex (taken to the limit, we fit any finite data set exactly)

High model complexity comes with high coefficient weights – Controlling the size of parameters (regularization) as way to control model complexity



- But fitting these models also requires extremely large coefficients on these polynomials
- For 50 degree polynomial, the first few coefficients are:

$$\theta = 2.27\times10^4, 1.61\times10^5, 6.88\times10^4, -1.36\times10^5, \ldots$$

- This suggests an **alternative way to control model complexity**: **keep the weights small (regularization)**, i.e. control the magnitude of the model parameters!

How do we control the size of the parameters? – We add a term in our objective (loss) function

L2 regularization (ridge regression)

$$\underset{\theta}{Minimize}\ \frac{1}{m}\sum_{i=1}^{m}\ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) + \lambda\sum_{i=1}^{n}\theta^2$$

Note: The above regularization is referred to as L2 regularization (ridge regression), as we work with a squared regularization term

- This formulation trades off loss on the training set with a penalty on high values of the parameters (λ = **regularization parameter**).
- By varying λ from zero (**no regularization**) to infinity (**infinite regularization**, meaning parameters will all be zero), we can sweep out different sets of model complexity

Regularized least squares

- For least squares, there is a simple solution to the regularized loss minimization problem

$$\underset{\theta}{Minimize}\ \frac{1}{m}\sum_{i=1}^{m}\ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) + \lambda\sum_{i=1}^{n}\theta^2$$

- Taking gradients by the same rules as before gives:

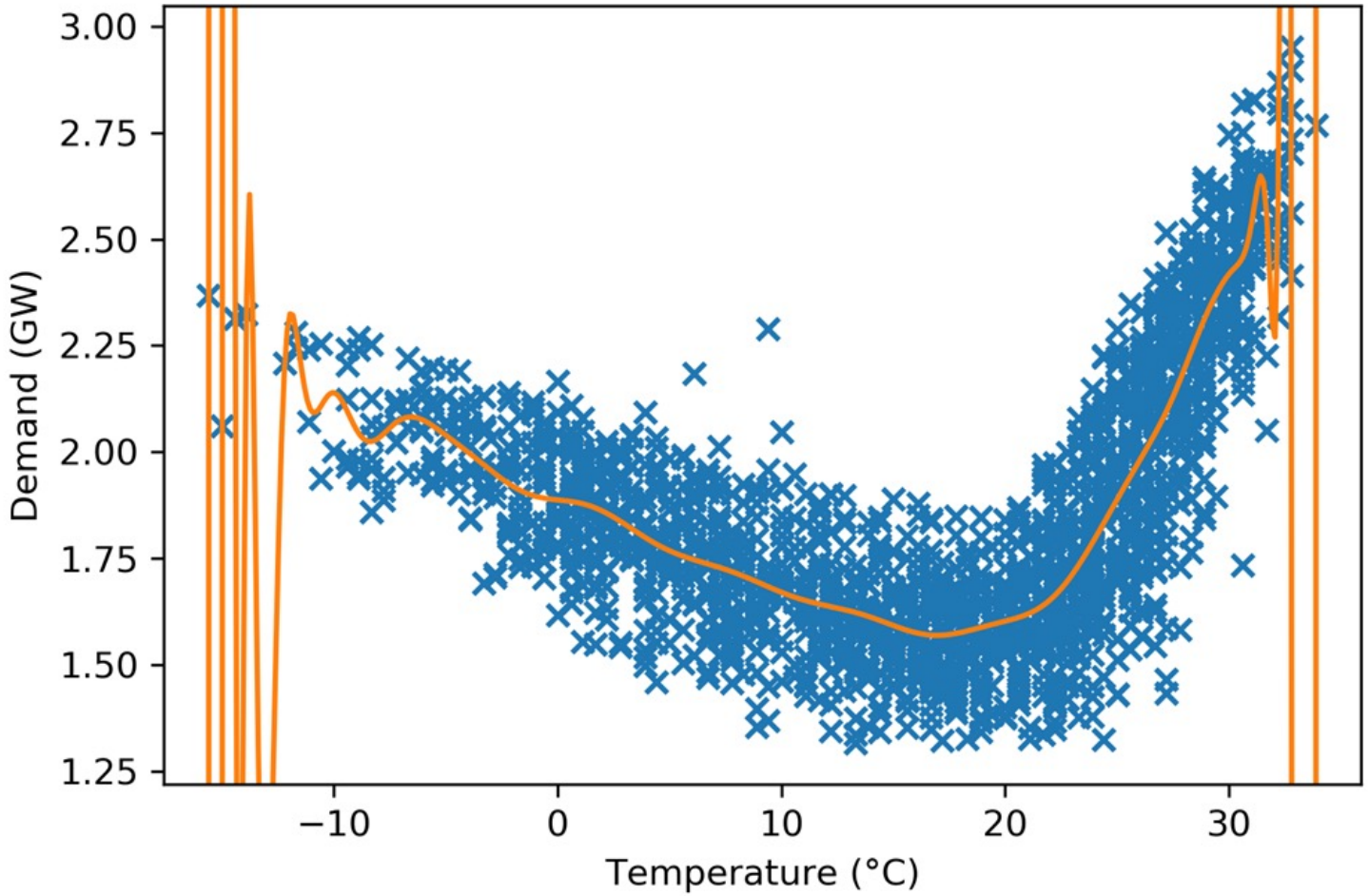
$$\nabla_\theta\left(\sum_{i=1}^{m}\left(\theta^T x^{(i)} - y^{(i)}\right)^2 + \lambda\sum_{i=1}^{n}\theta^2\right) = 2X^T(X\theta - y) + 2\lambda\theta$$

- Setting gradient equal to zero leads to the solution

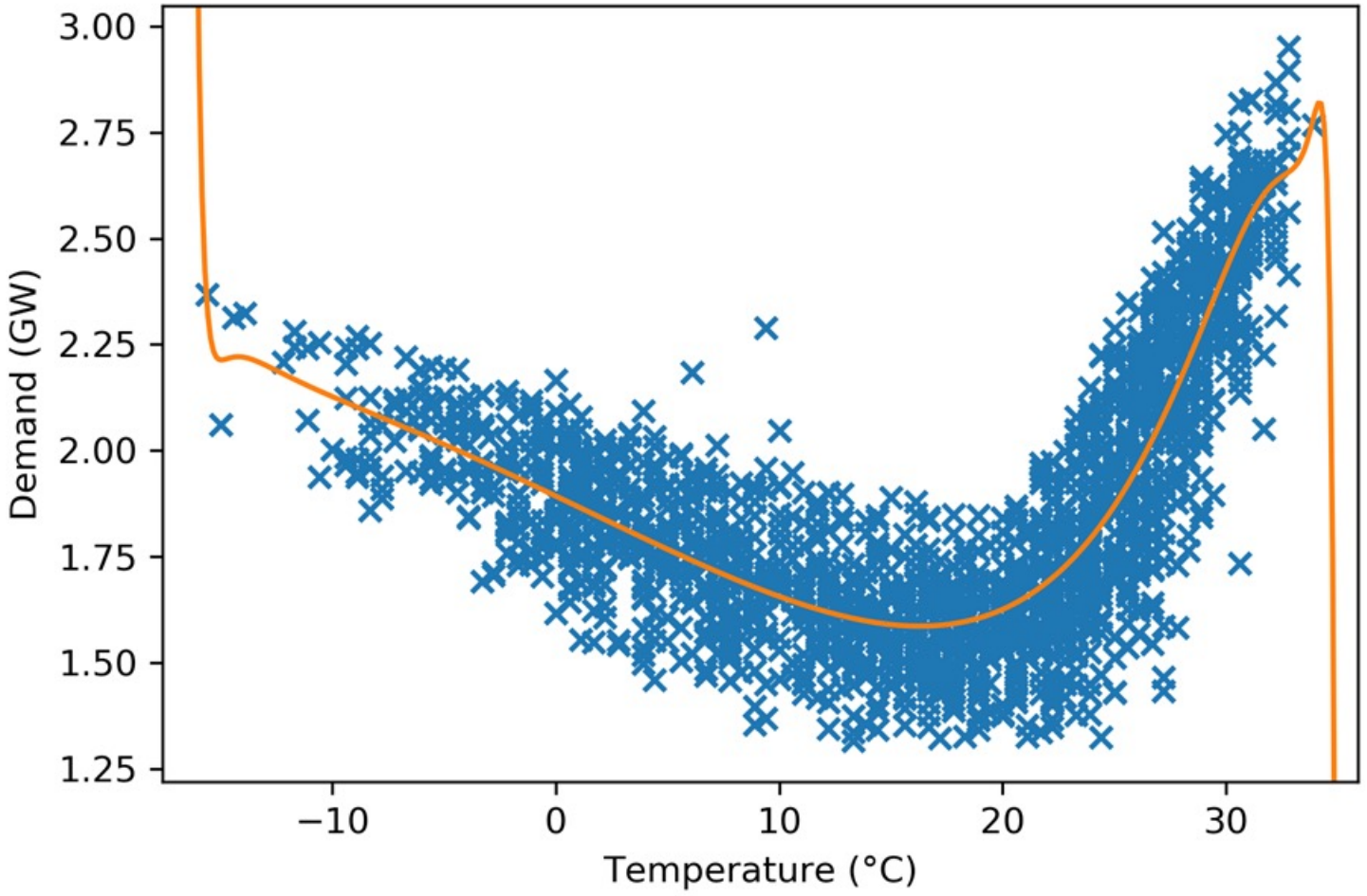
$$2X^TX\theta + 2\lambda\theta = 2X^Ty \Rightarrow \theta = (X^TX + \lambda I)^{-1}X^Ty$$

- Looks just like the normal equations but with an additional λI term

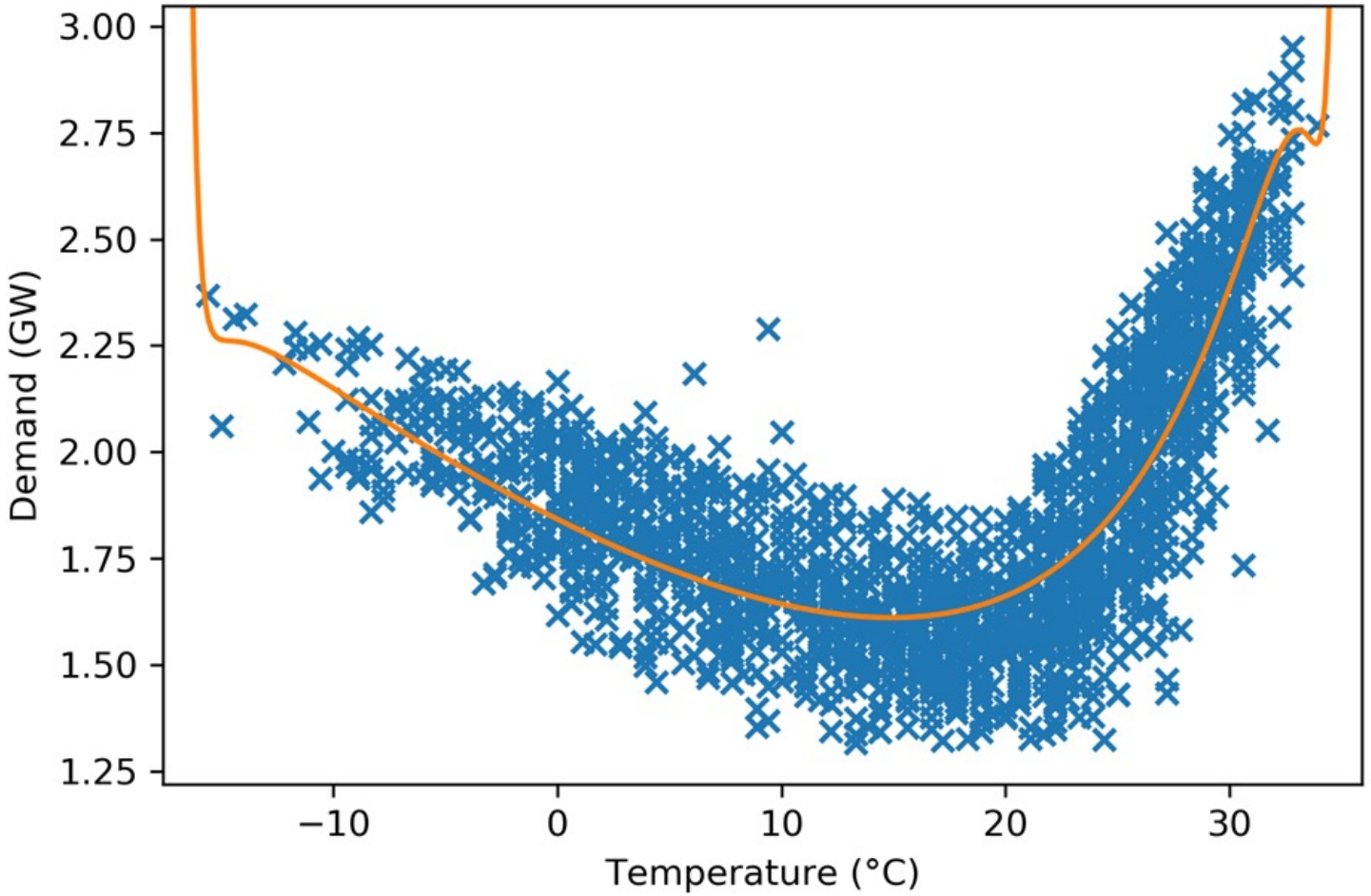
100 degree polynomial fit (λ = 0)



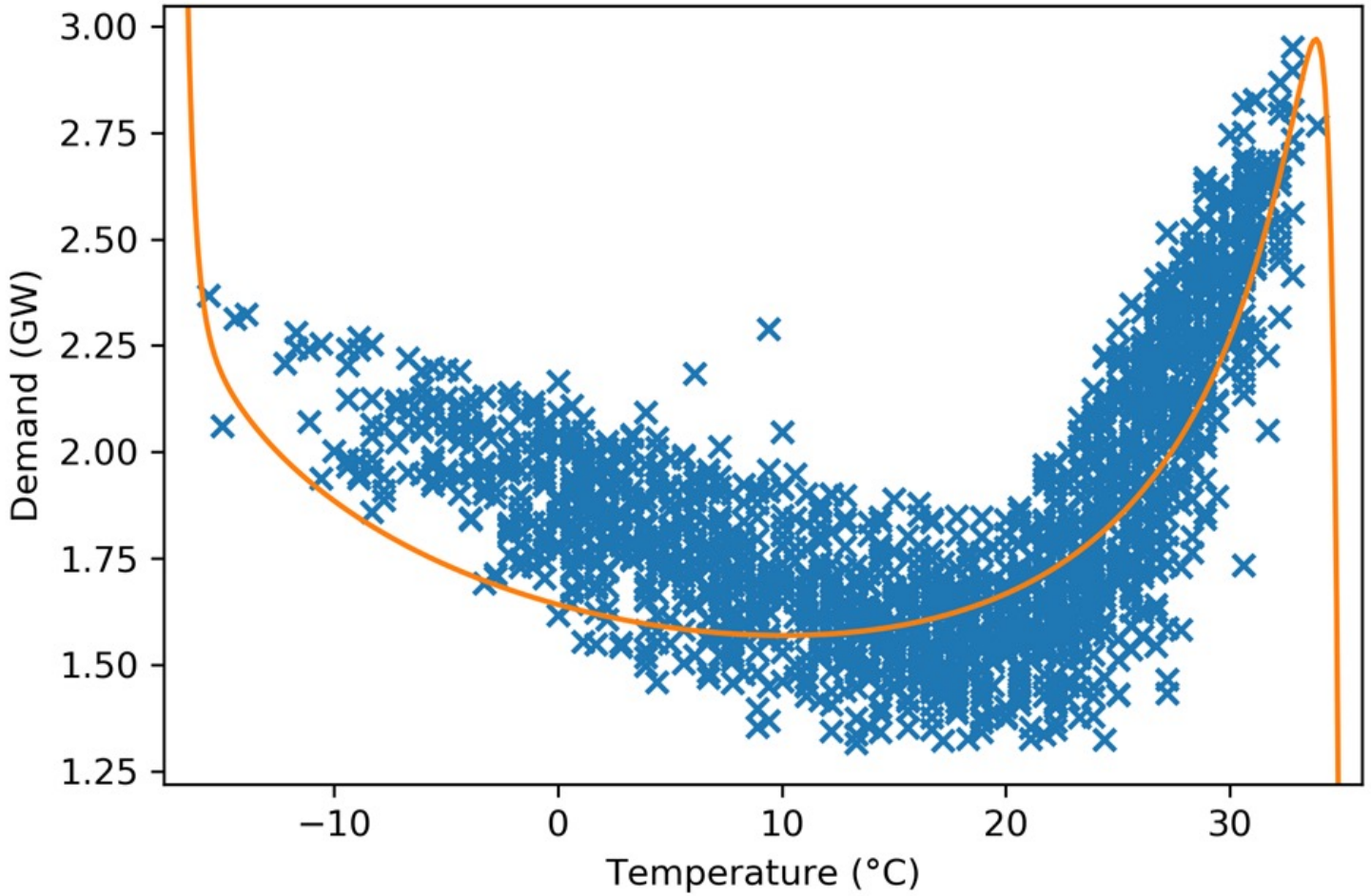
100 degree polynomial fit (using regularization with λ = 1)



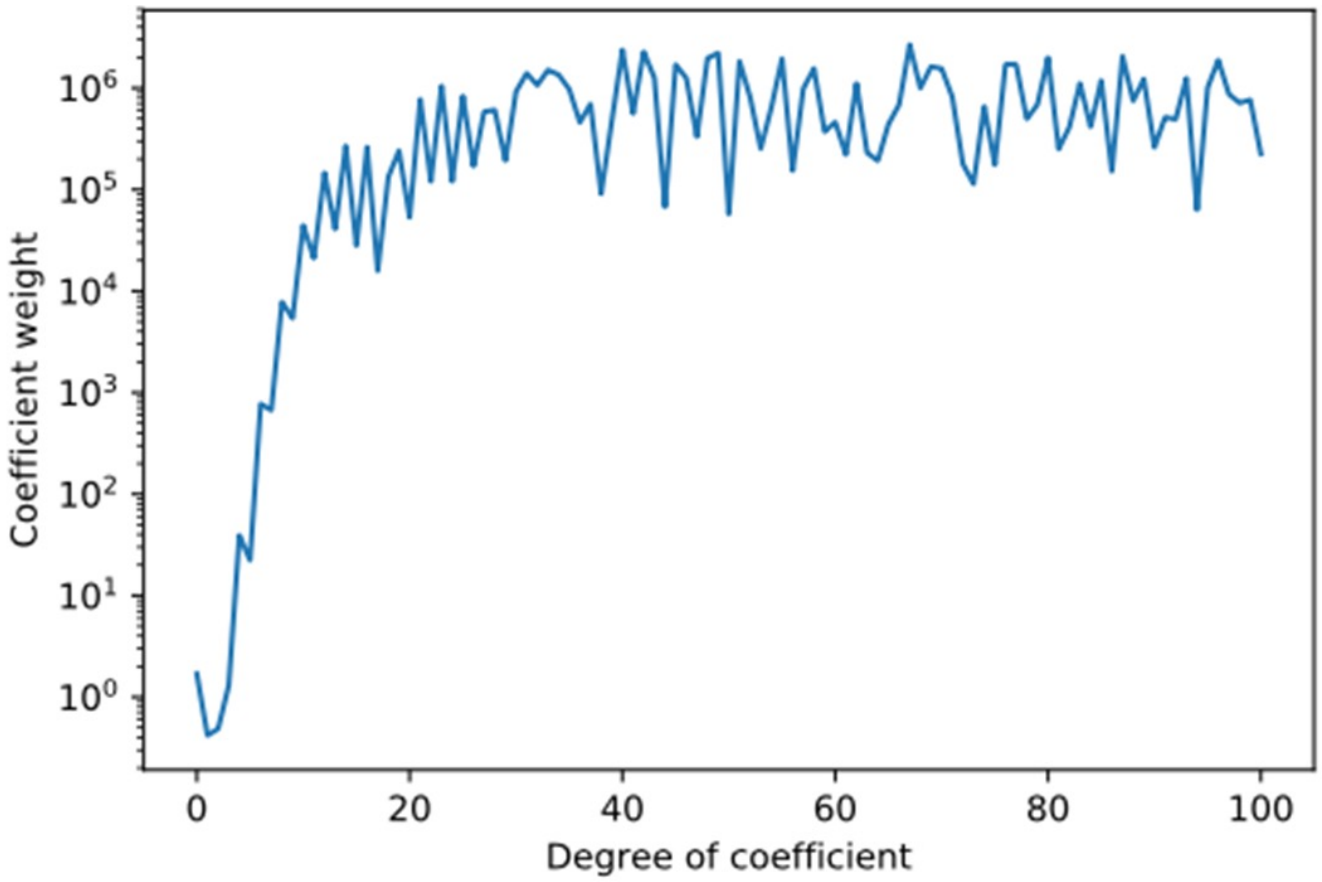
100 degree polynomial fit (using regularization with λ = 10)



100 degree polynomial fit (using regularization with λ = 100)

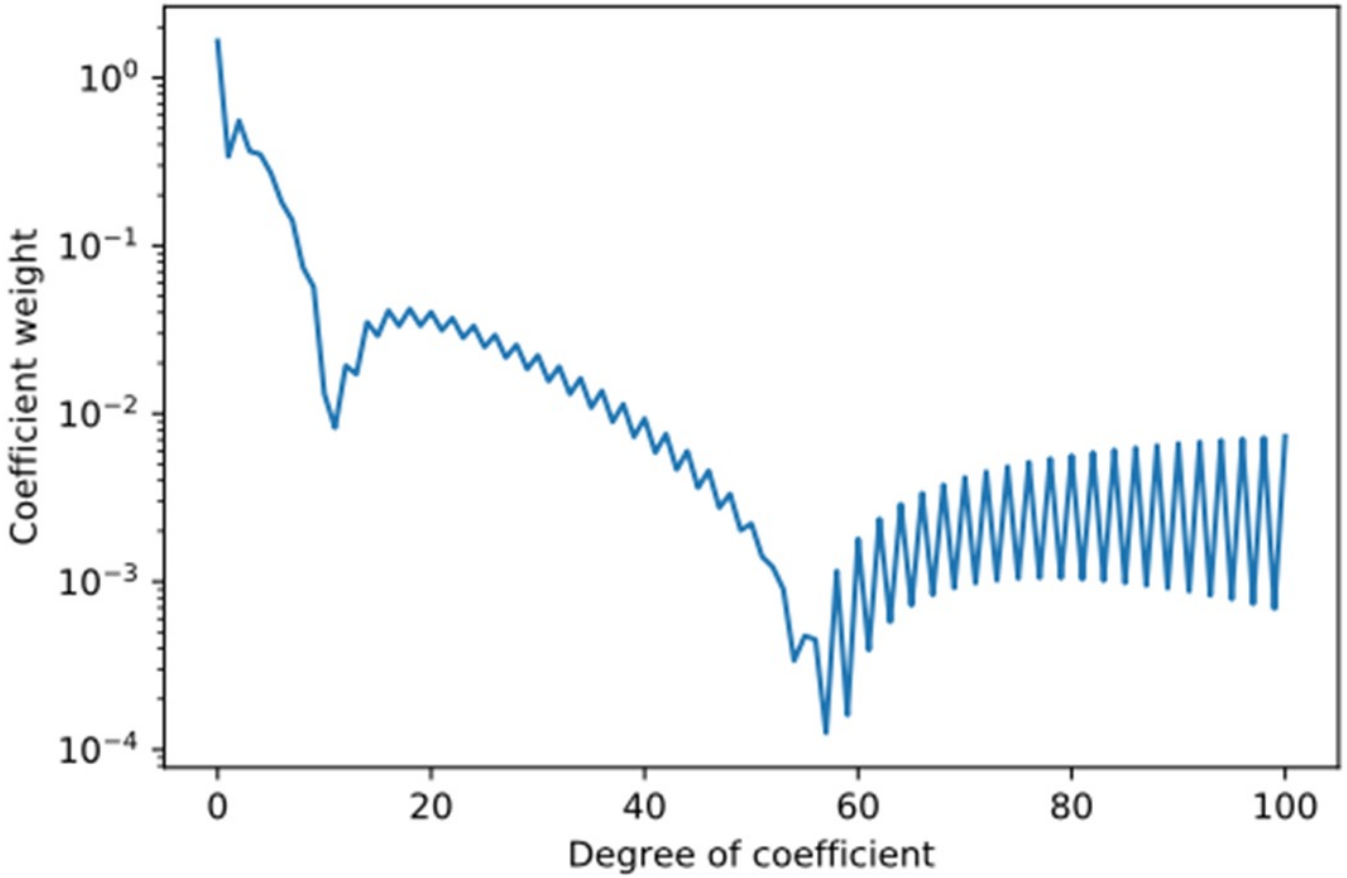


Let's consider the coefficient weights **without regularization**



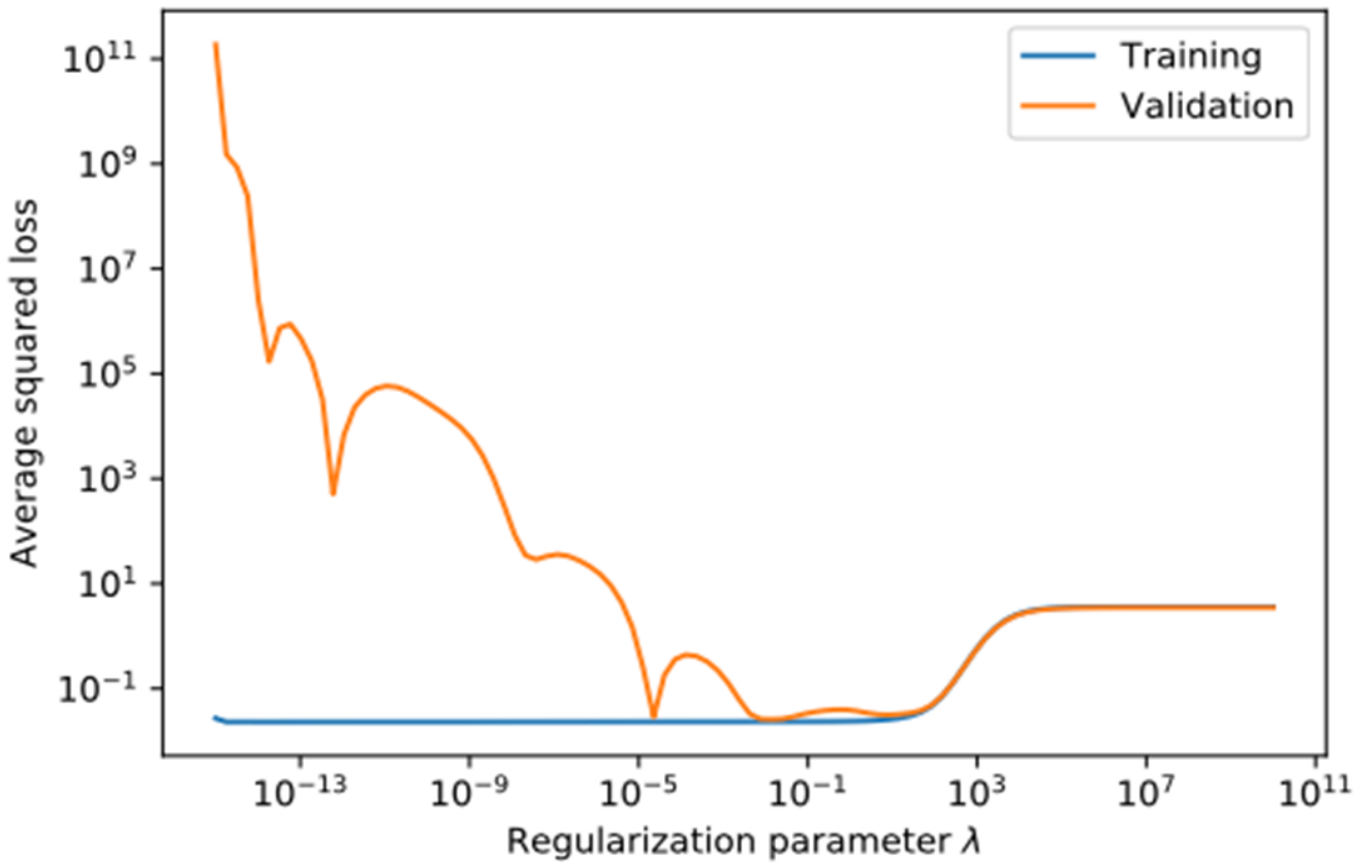
- First let's look at the magnitude of the coefficients (on a log scale), for the unregularized θ

… and **with regularization** ($\lambda = 1$)



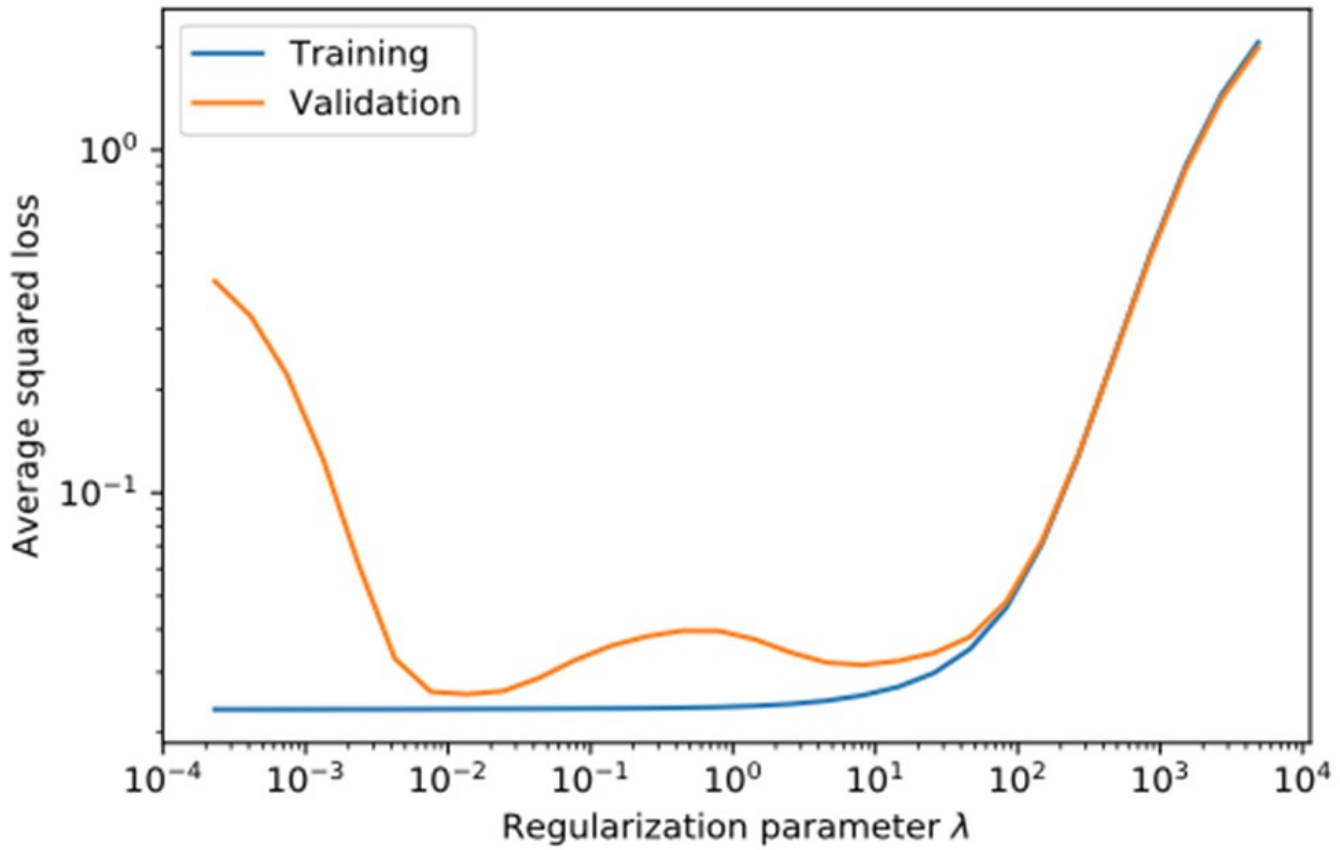
- After degree 8 or so, the model puts very little relative weight on any higher-degree coefficients
- This makes sense as we know from the previous lecture that we can fit the data well with a relatively small polynomial degree
- The regularization term thus leads us to rely more heavily on the lower order terms

Training/cross-validation loss by regularization



- Lower λ means less regularization, whereas large λ means more regularization (eventually just corresponding to all zero weights)
- Also note that we are using a logarithmic scale on the x-axis. This means that regularization typically works on a scale of orders of magnitude. If you **search over possible regularization terms**, you'll need to do this search **over a logarithmic space**, because you need very large changes to the magnitude of λ.

Training/cross-validation loss by regularization (zoom into good region)



- This plot suggests that regularization parameters between 10^{-2} and 10^{1} seem to work best for this problem
- (The middle bump between the two extremes is likely an artificat of the particular cross validation set, and the whole range still suffers relatively low error)

Poll: features and regularization

Suppose you run linear regression with polynomial features and some initial guess for d and λ. You find that your validation loss is much higher than your training loss. Which actions might be beneficial to take?

1. Decrease λ
2. Increase λ
3. Decrease d
4. Increase d

Answer: features and regularization

Suppose you run linear regression with polynomial features and some initial guess for d and λ.You find that your validation loss is much higher than you training loss. Which actions might be beneficial to take?

1. Decrease λ
2. **Increase λ**
3. **Decrease d**
4. Increase d

Both these ways decreases the validation loss. But the choice of each of these ways depends on your data.

Another option to control model complexity is L1 regularization (LASSO regression)

L1 regularization (LASSO regression)

$$\underset{\theta}{Minimize}\ \frac{1}{m}\sum_{i=1}^{m}\ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) + \lambda\sum_{i=1}^{n}|\theta|$$

Note: The above regularization is referred to as L1 regularization (LASSO regression), as we work with an unsquared regularization term

- This formulation trades off loss on the training set with a penalty on absolute values of the parameters (λ = **regularization parameter**).
- This type of regularization (L1) **can lead to zero coefficients** of particular features so LASSO also **helps in feature selection**

Comparison: L1 vs. L2 Regularization

Lasso (L1) Regularization

$$\min_{\theta} \frac{1}{m}\sum_{i=1}^{m} \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) + \lambda \sum_{i=1}^{n} |\theta|$$

- We penalize the (absolute) weight of coefficients to combat uneccesary degrees of freedom in the model, striking a trade-off between complexity and goodnes of fit.
- Essentially, this can be used as feature selection, because a θ of zero hints at a feature not being relevant.

Ridge Regression (L2 Regularization)

$$\min_{\theta} \frac{1}{m}\sum_{i=1}^{m} \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) + \lambda \sum_{i=1}^{n} \theta^2$$

- Here, the larger the coefficient, the more it get's penalized (by having an influence of λ on the loss), with a quadratic relationship.
- This distributes coefficient weight (i.e., penalty) between features, essentially "smoothing" influence of single features.
- In a way, L2 discourages setting θ to zero, because the quadratic term means diminishing returns for the penalty, therefore setting some coefficient to zero doesn't "help" the penalty as it doesn't significantly reduces it.

Contact

For general questions and enquiries on **research**, **teaching**, **job openings** and new **projects** refer to our website at www.is3.uni-koeln.de

For specific enquiries regarding this course contact us by sending an email to the **IS3 teaching** address at is3-teaching@wiso.uni-koeln.de and prefix the subject with [AA].