# Lecture 4 – Supervised Learning

Regularization, Linear Classification

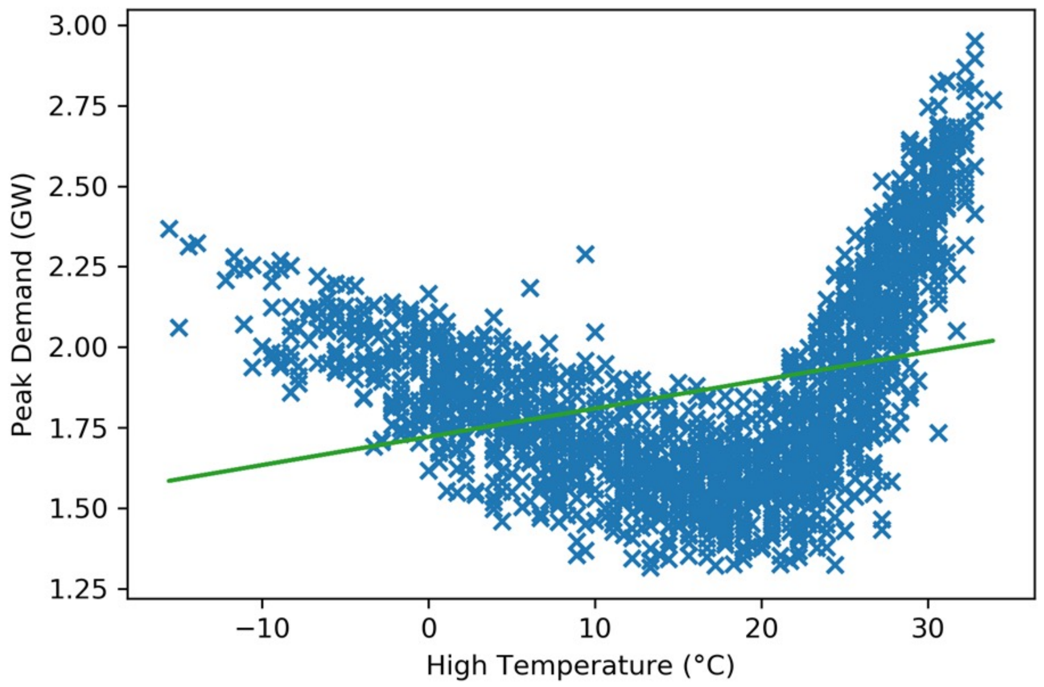
Information Systems for Sustainable Society (is3) | WiSo Faculty | Univ.-Prof. Dr. Wolfgang Ketter | 11/1/22

## Agenda

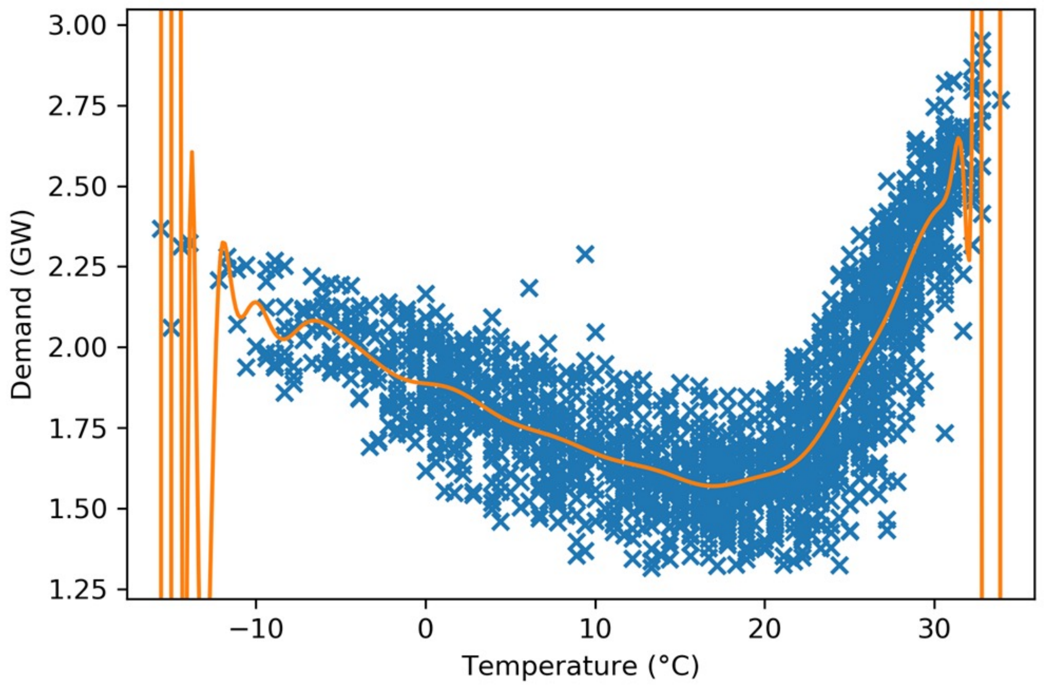
- Evaluating Regression Model Performance
- Regularization
- Linear Classification

## 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!

Goal: Minimize generalization error, not training error!

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

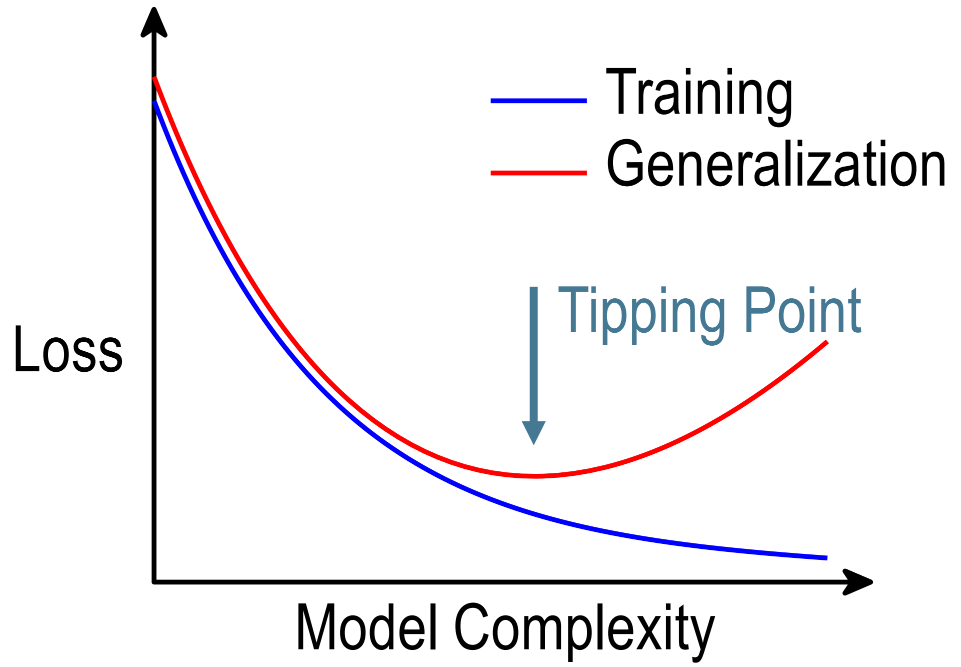
- Suppose we have $m$ examples in our data set
- Further suppose we have $n = m$ features (plus assumption that features are linearly independent)
- Then $X \in \mathbb{R}^{m \times n}$ is a square matrix, and least squares solution is:

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

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?

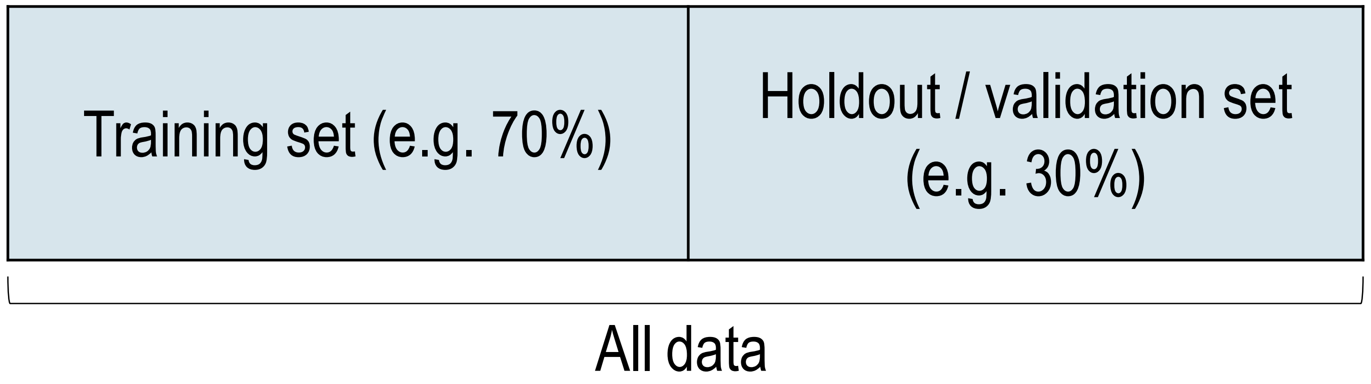
1. Compare two electricity forecasting models that predict **daily and hourly electricity** demand, respectively, for **Pittsburgh.**
2. Compare two electricity forecasting models that predict **hourly electricity** demand for **Pittsburgh and Chicago** respectively.
3. Compare different **hourly electricity 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]

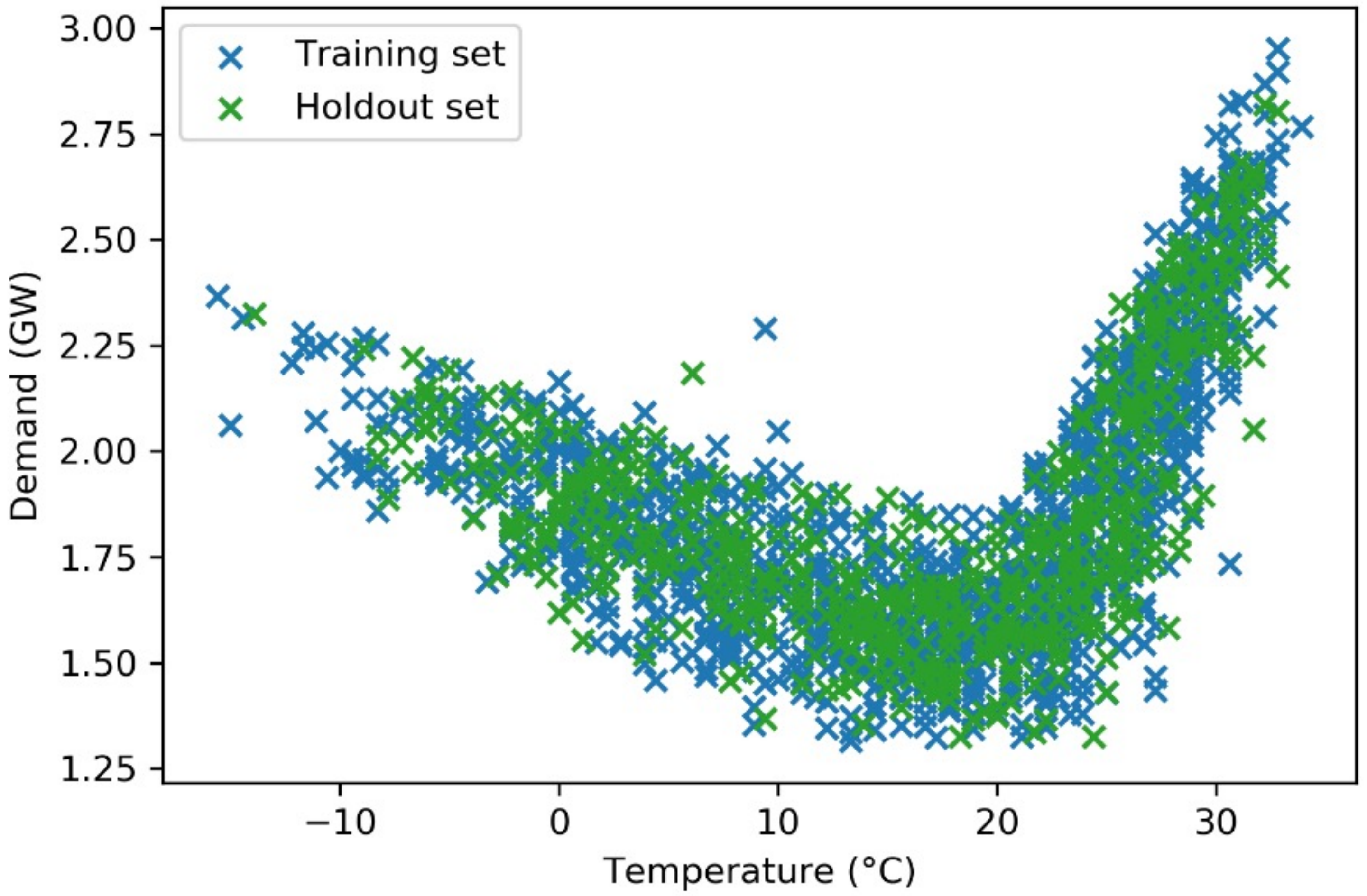
## To determine prediction errors we use **cross-validation**



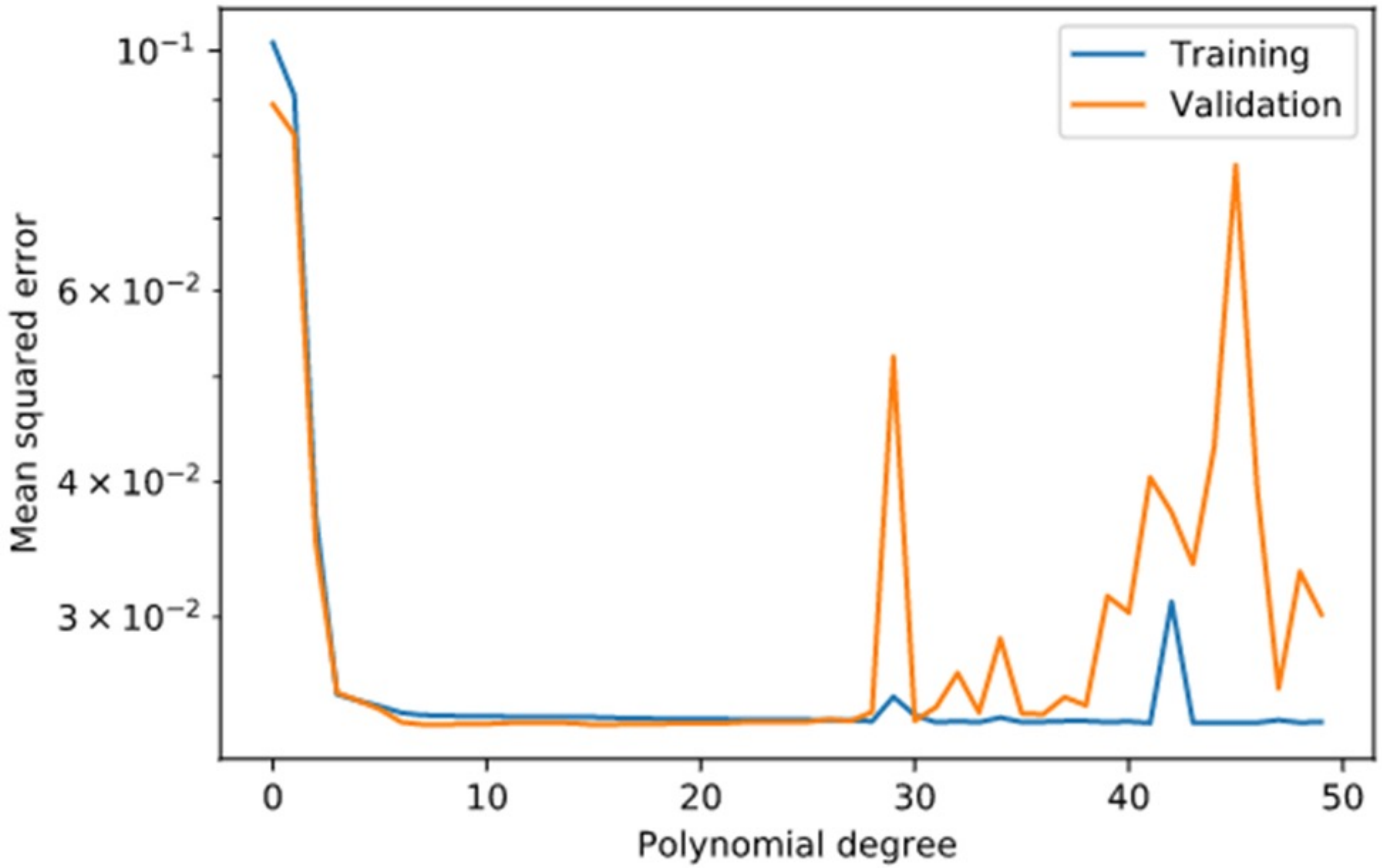
- Although it is difficult to quantify the true generalization error (i.e., the error of these algorithms over the *complete* distribution of possible examples), we can approximate it by **holdout cross-validation**
- Basic idea is to split the data set into a training set and a holdout set
- Train the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

Source: Bishop (2006)

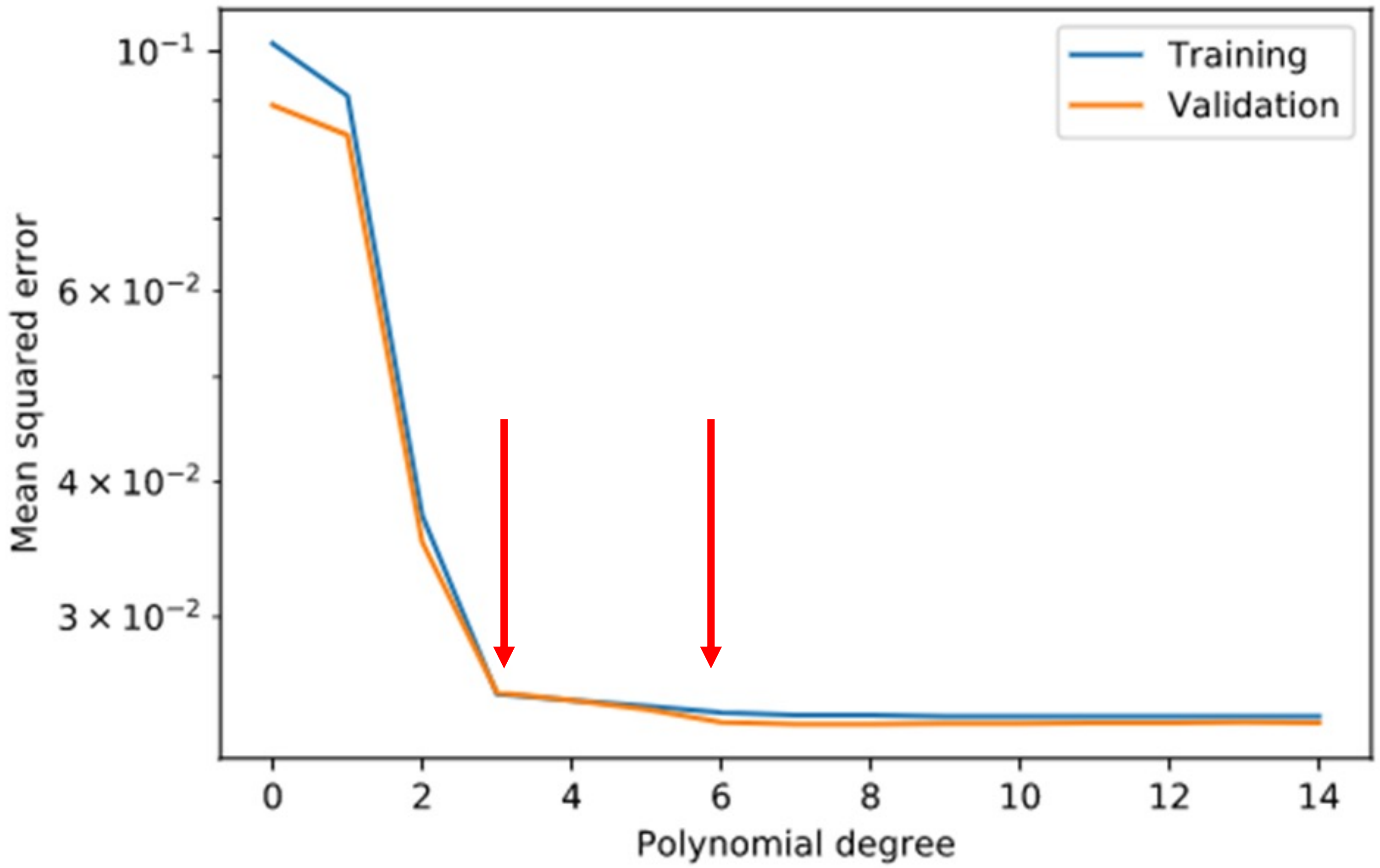
## Illustrating cross-validation



## Training and cross-validation loss by degree



## Training and cross-validation loss by degree

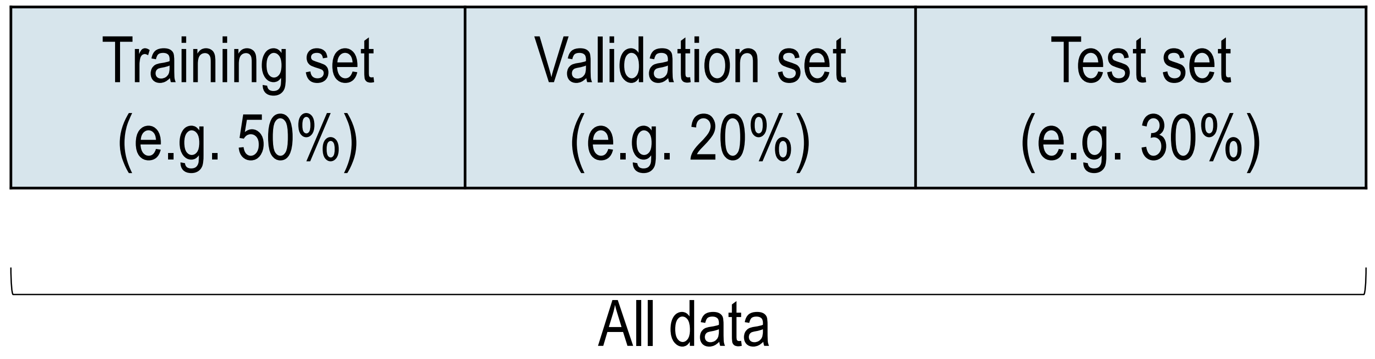


Universität zu Köln

## 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 $\theta$s)
  - **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)

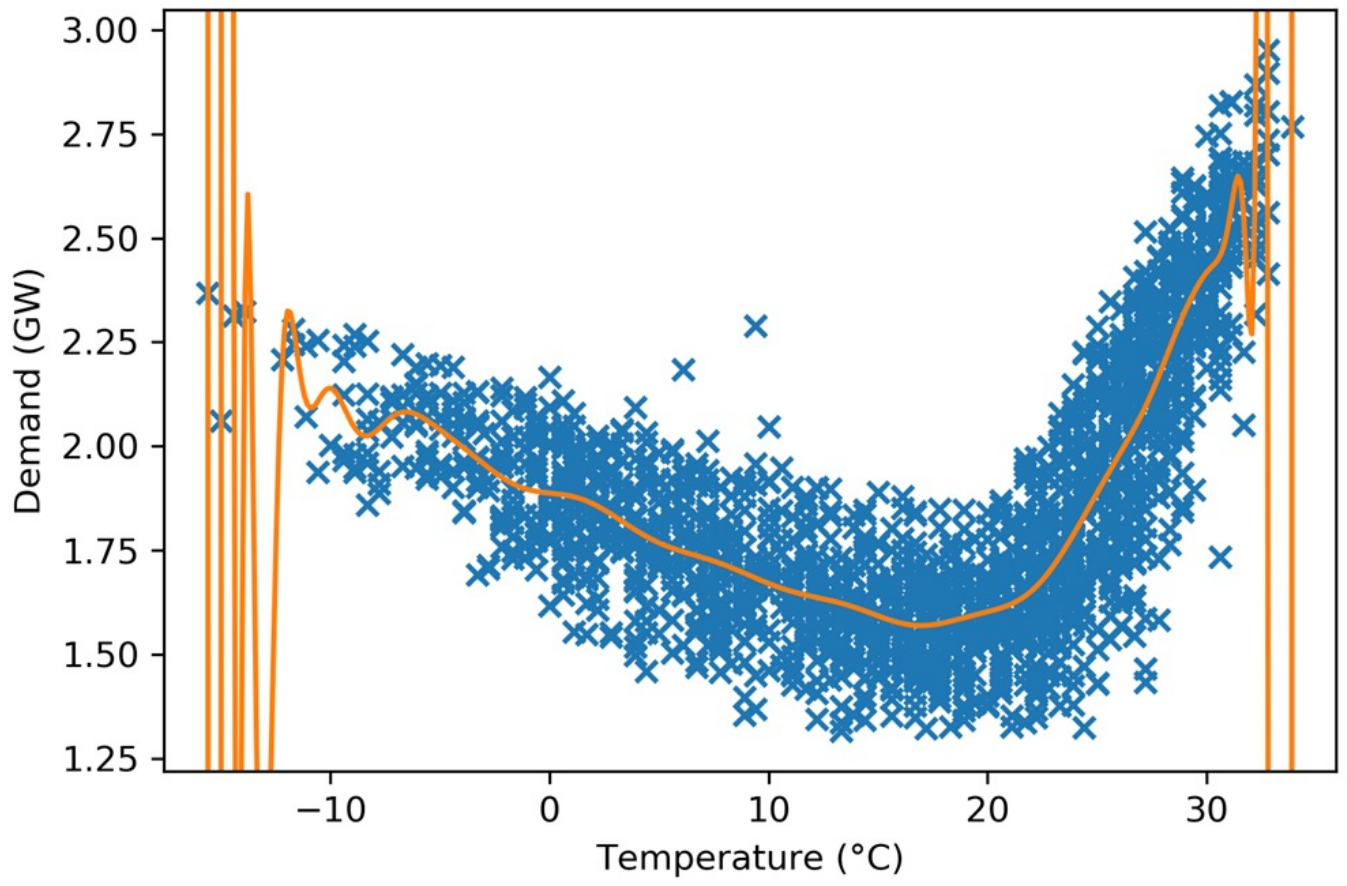
## In practice…

- “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

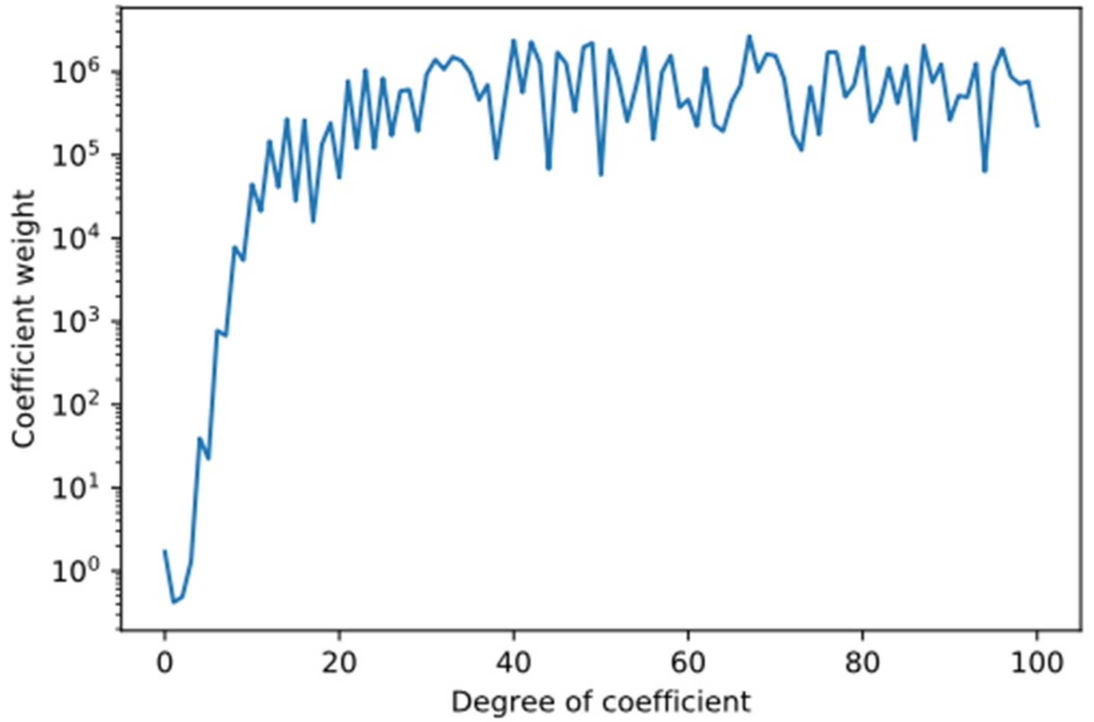
- Evaluating Regression Model Performance
- **Regularization**
- Linear Classification

## 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:

$$\theta = 2.27{\times}10^4, 1.61{\times}10^5, 6.88{\times}10^4, -1.36{\times}10^5, \ldots$$

- 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)**

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

**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 ($\lambda$ = **regularization parameter**).
- By varying $\lambda$ 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:

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

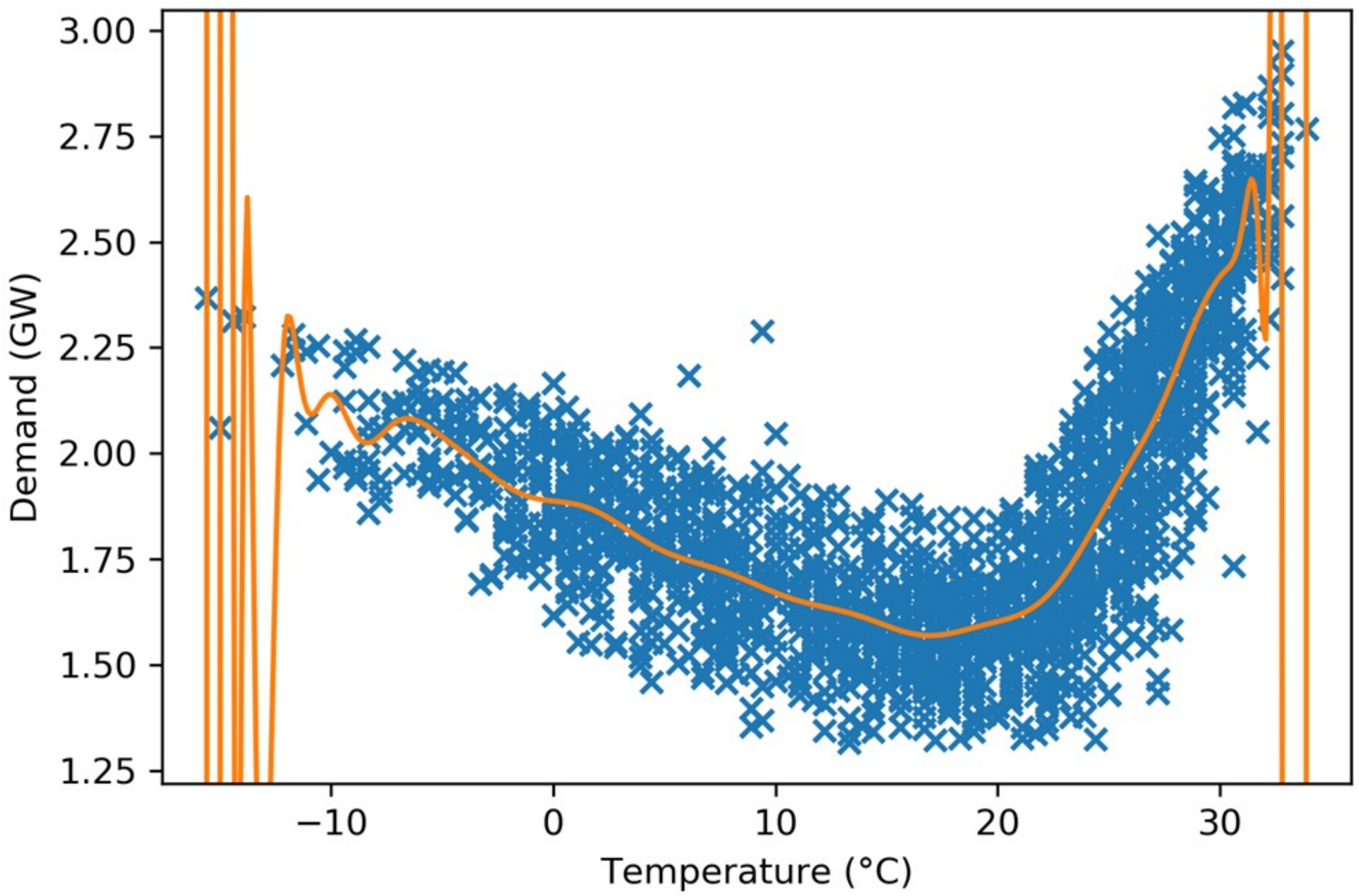
- Setting gradient equal to zero leads to the solution

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

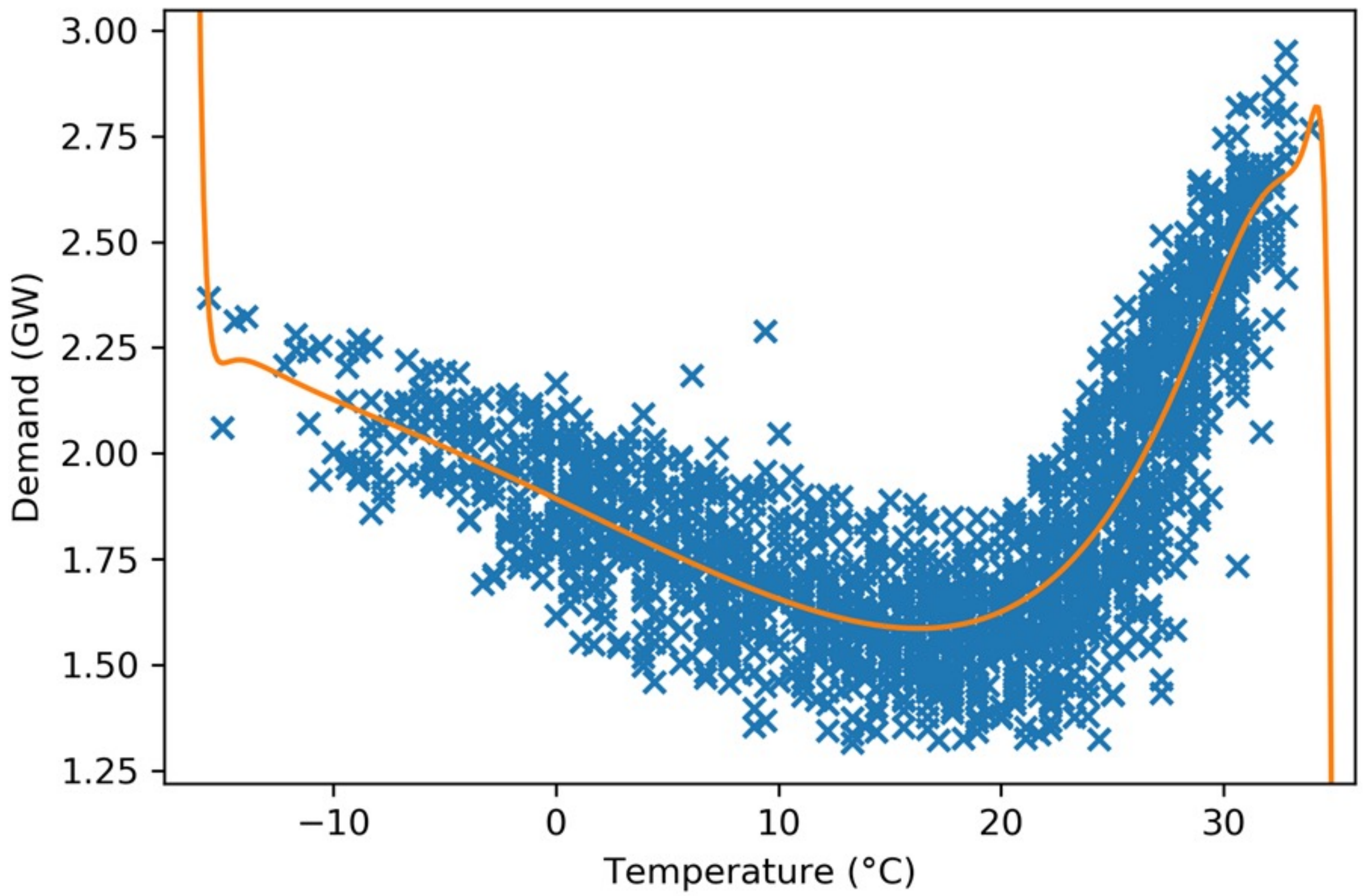
- Looks just like the normal equations but with an additional $\lambda I$ term

Universität zu Köln

## 100 degree polynomial fit ($\lambda = 0$)

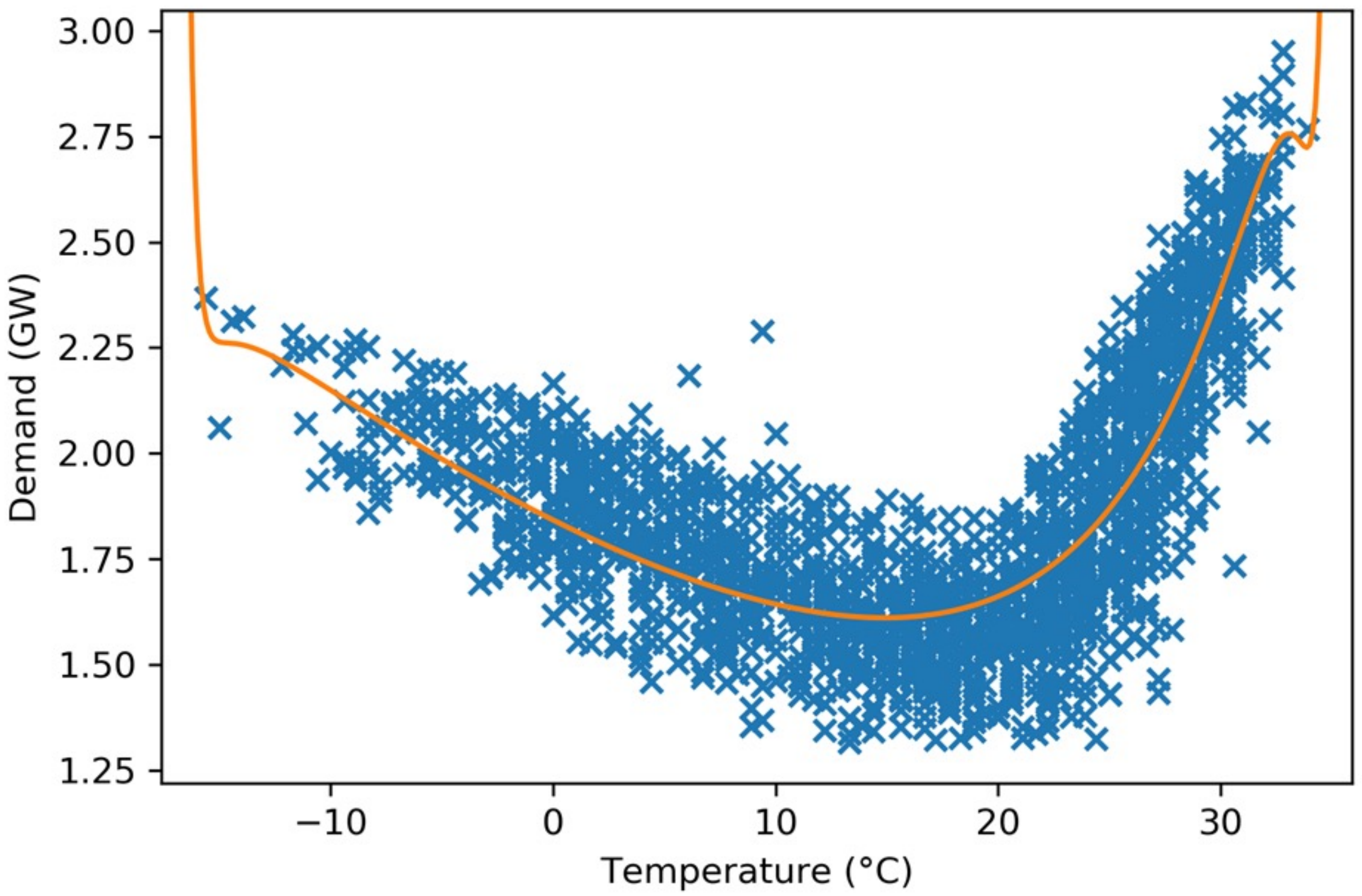


## 100 degree polynomial fit (using regularization with $\lambda = 1$)

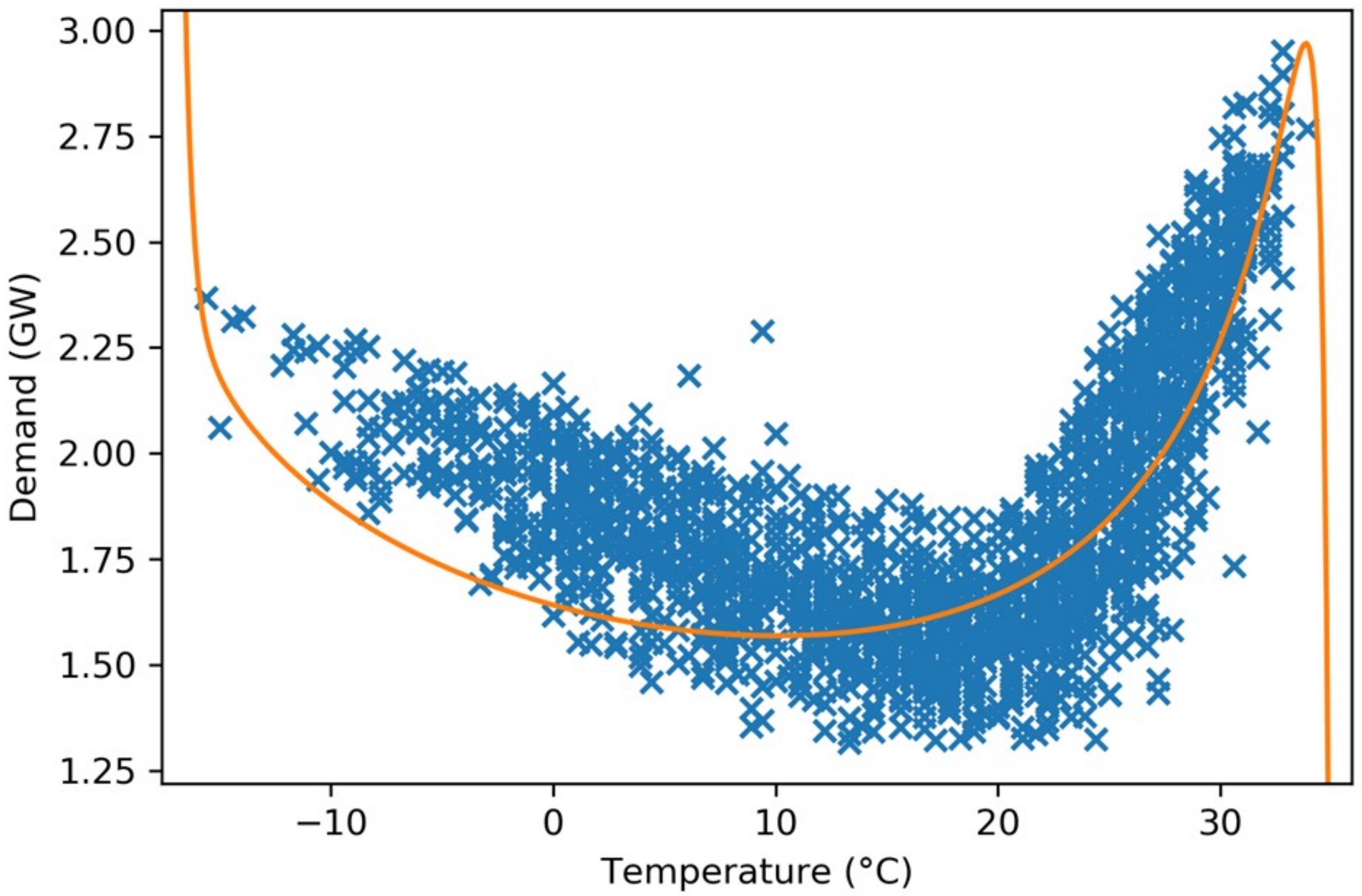


Universität zu Köln

## 100 degree polynomial fit (using regularization with $\lambda$ = 10)

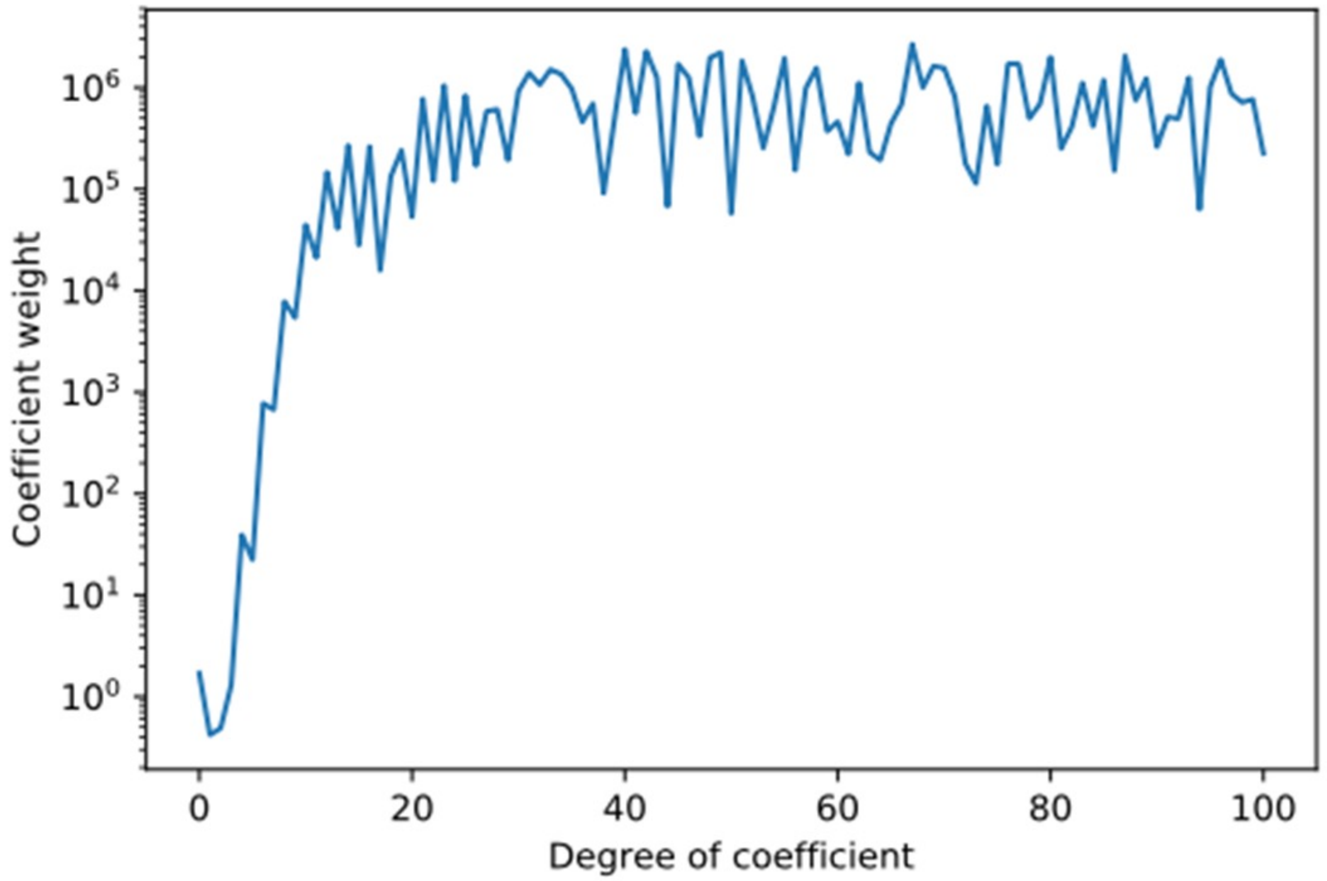


## 100 degree polynomial fit (using regularization with $\lambda$ = 100)



Universität zu Köln

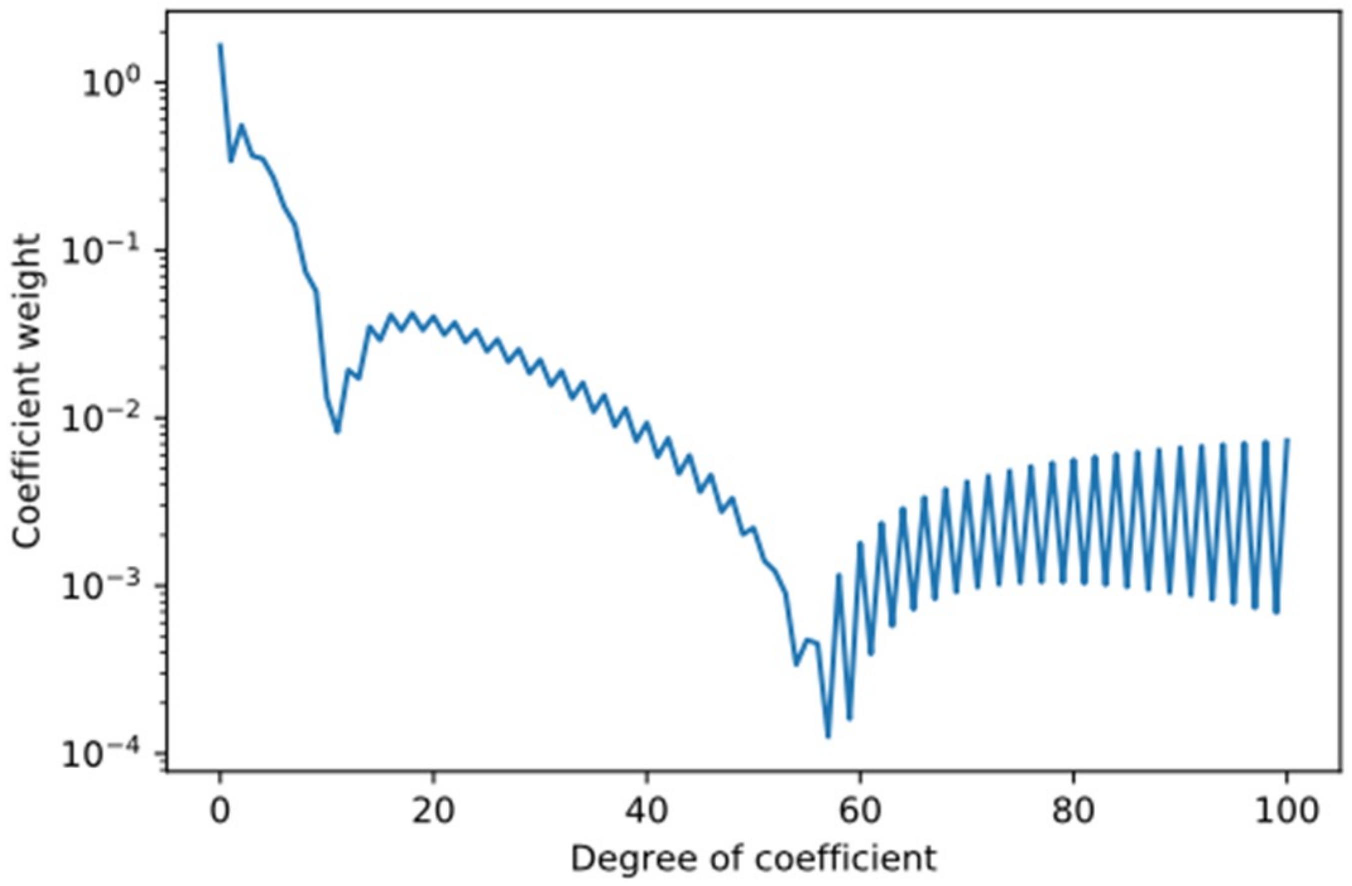
## Let’s consider the coefficient weights **without regularization**



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

Universität zu Köln

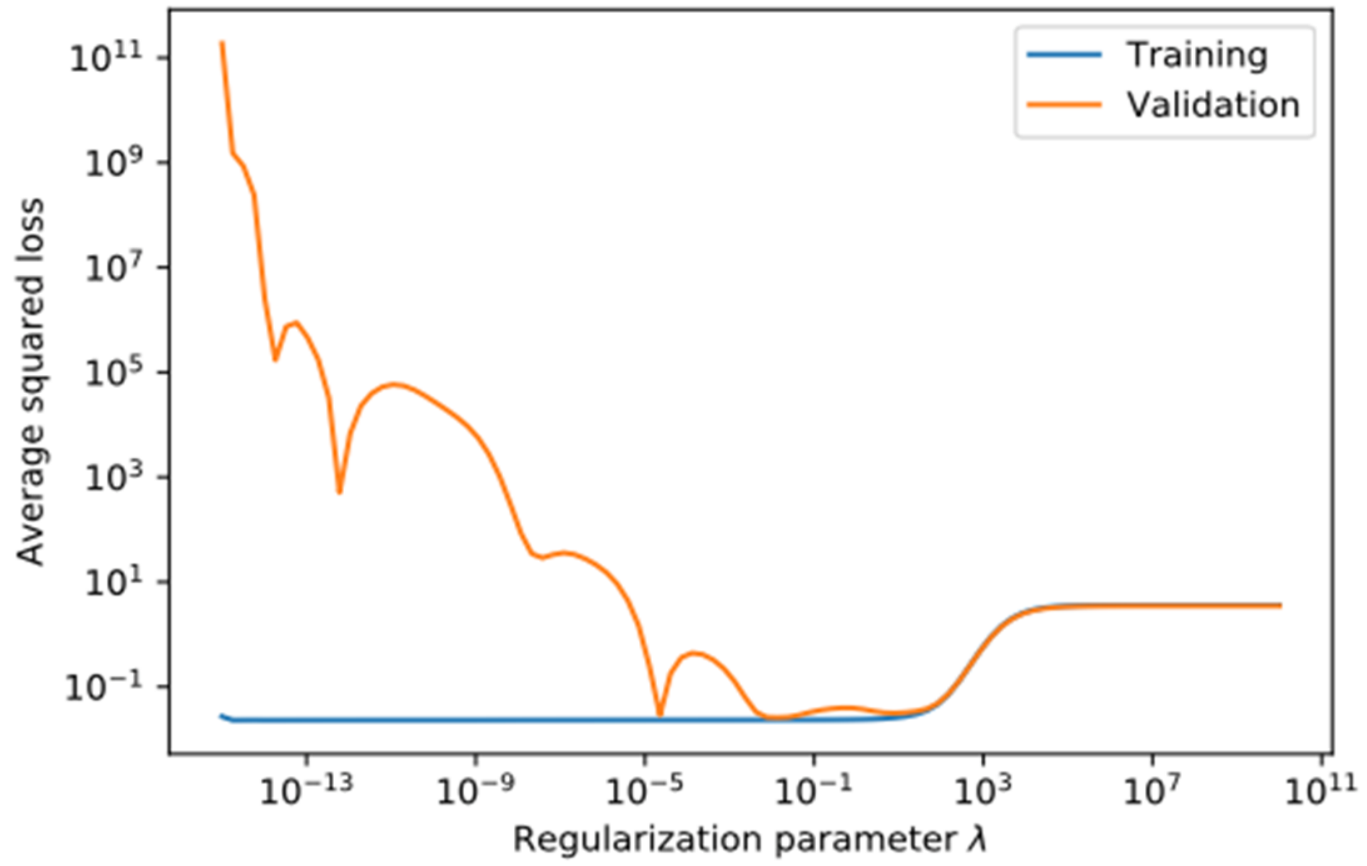
## … and **with regularization** ($\lambda = 1$)



- 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

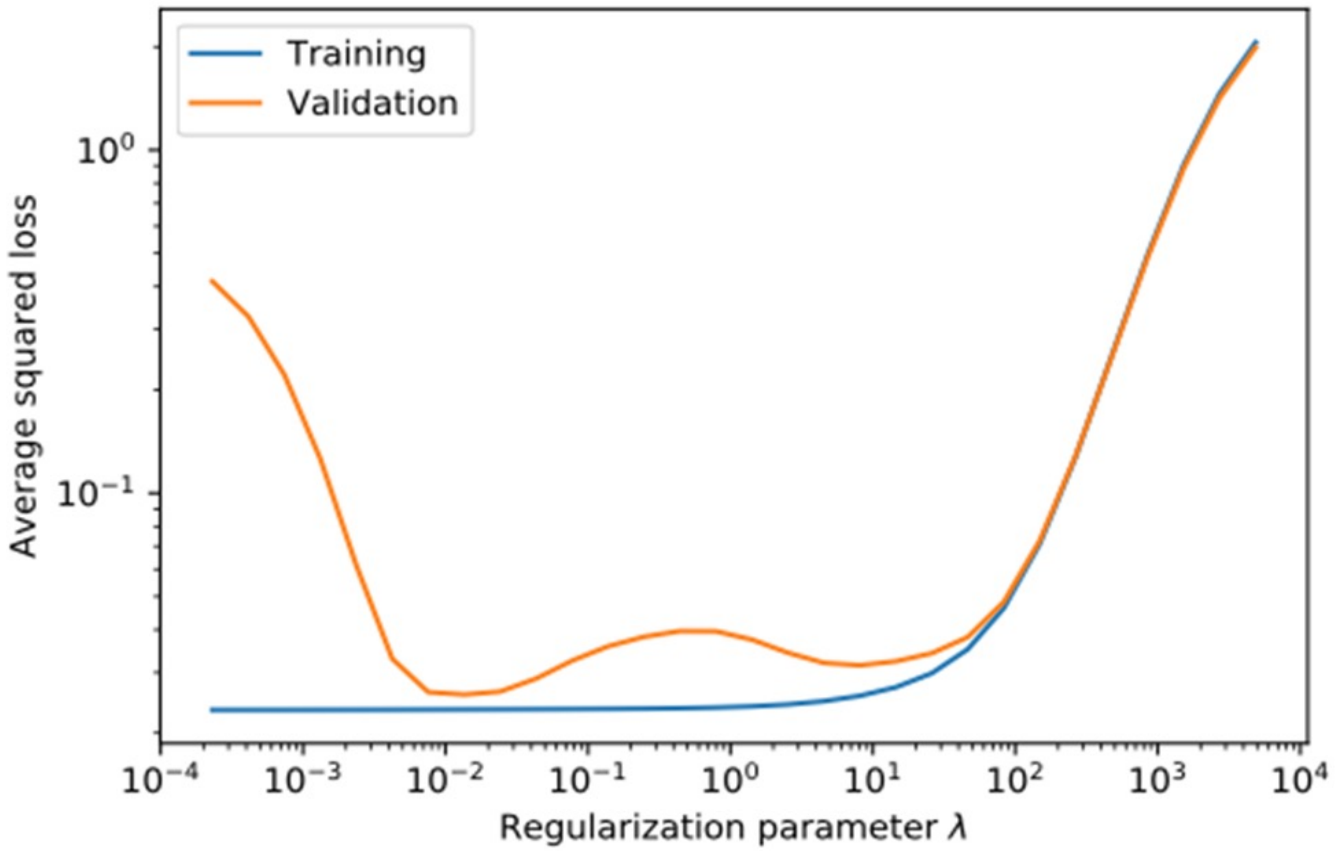
Universität zu Köln

## 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)

Universität zu Köln

## Poll: features and regularization

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

1. Decrease $\lambda$
2. [✓ Increase $\lambda$]
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 $\lambda$.You find that your validation loss is much higher than you training loss. Which actions might be beneficial to take?

1. Decrease $\lambda$
2. **Increase $\lambda$**
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 ($\lambda$ = **regularization parameter**).
- This type of regularization (L1) **can lead to zero coefficients** of particular features so LASSO also **helps in feature selection**

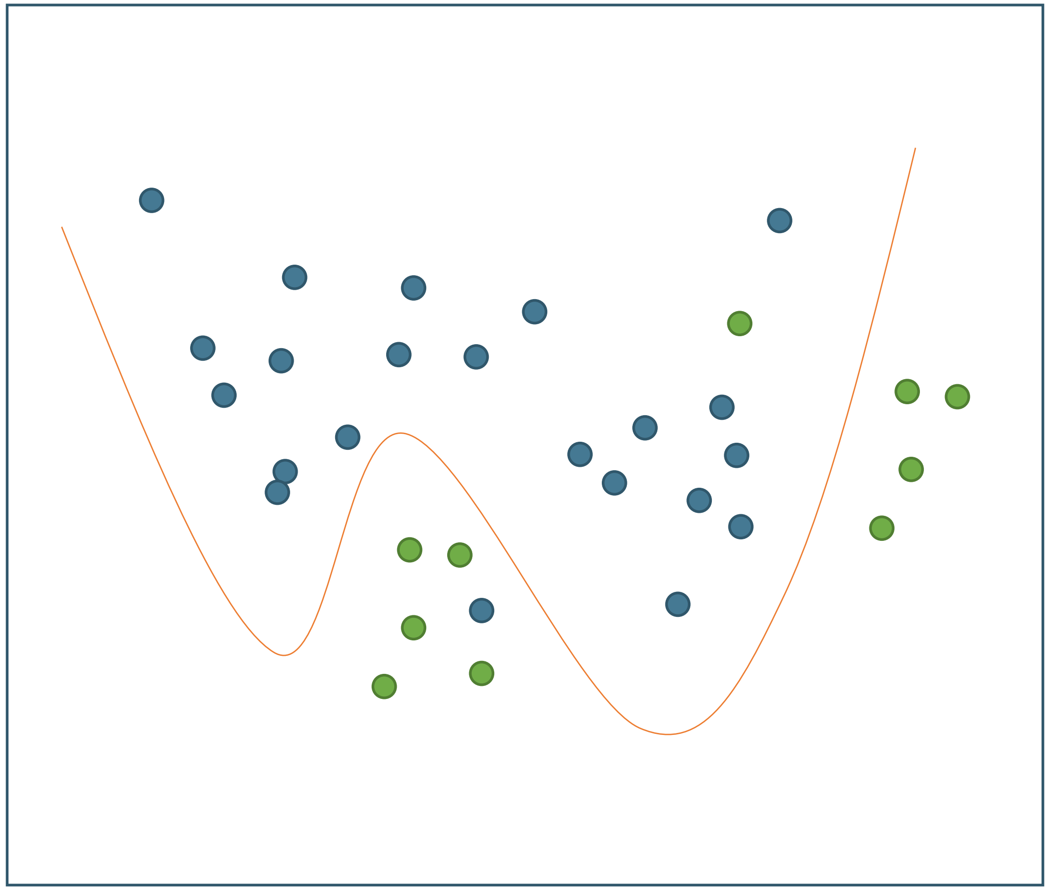
## Agenda

Evaluating Regression Model Performance

Regularization

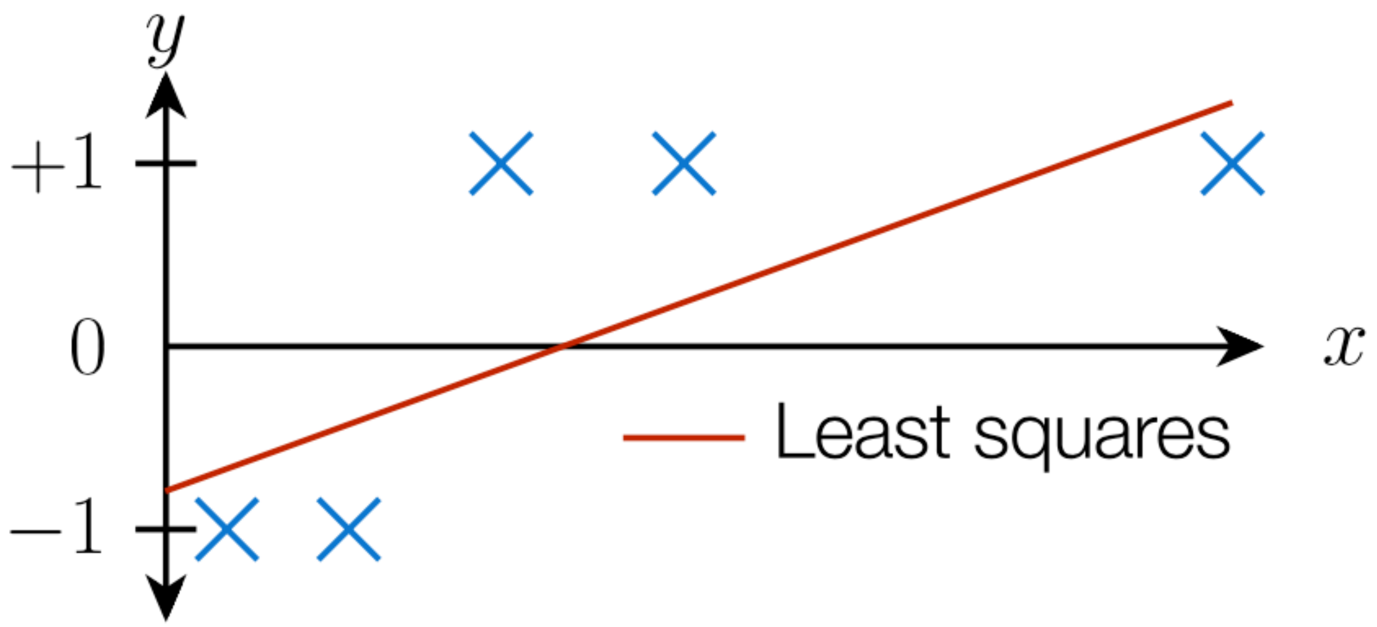
Linear Classification

## In this lecture we will move on from regression and focus on classification – Learning from labeled data to predict categorical outputs



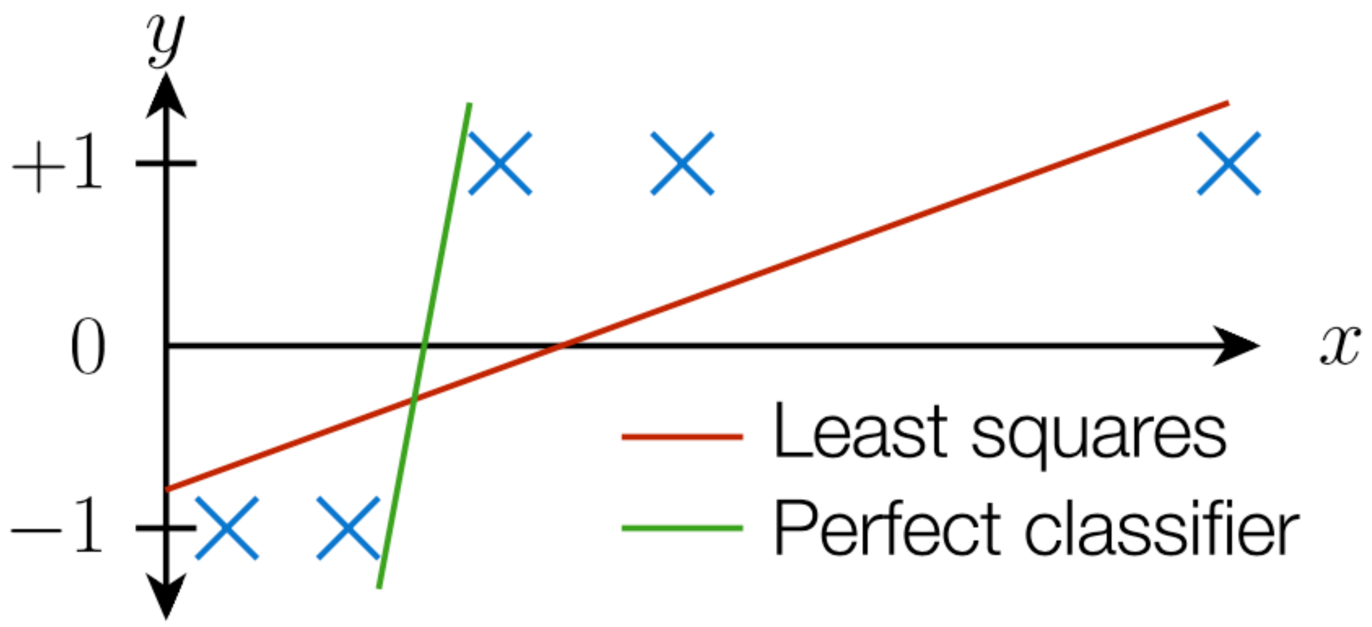
- Predict discrete-valued quantity $y$
  - **Binary classification:** $y \in \{-1, +1\}$
  - **Multi-class classification**: $y \in \{1,2, \ldots, k\}$

## Loss functions for classification



- How do we define a loss function $\ell: \mathbb{R} \times \{-1,+1\} \rightarrow \mathbb{R}_{+}$?
- What about just using squared loss?
  - We would be predicting a positive class wherever the line is positive, and vice versa
  - But this means that we actually predict incorrectly on the right-most positive point

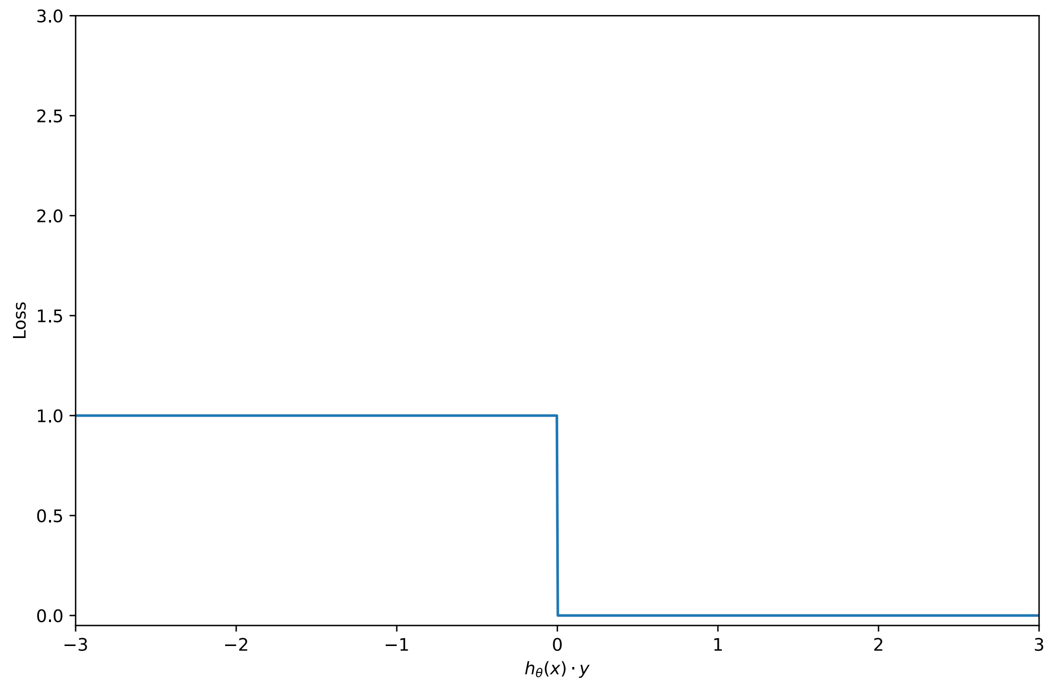
## Loss functions for classification



- It is of course completely possible to classify the data perfectly with a linear classifier
- Least-squares loss function applied to classification aims at predicting exactly +1 or −1 on each data point
- Numbers much larger than one for a positive example will add to the loss

## Since least-square seems inadequate in a classification setting, let us review alternative loss functions: 0/1 loss

### Illustration of 0/1 loss function



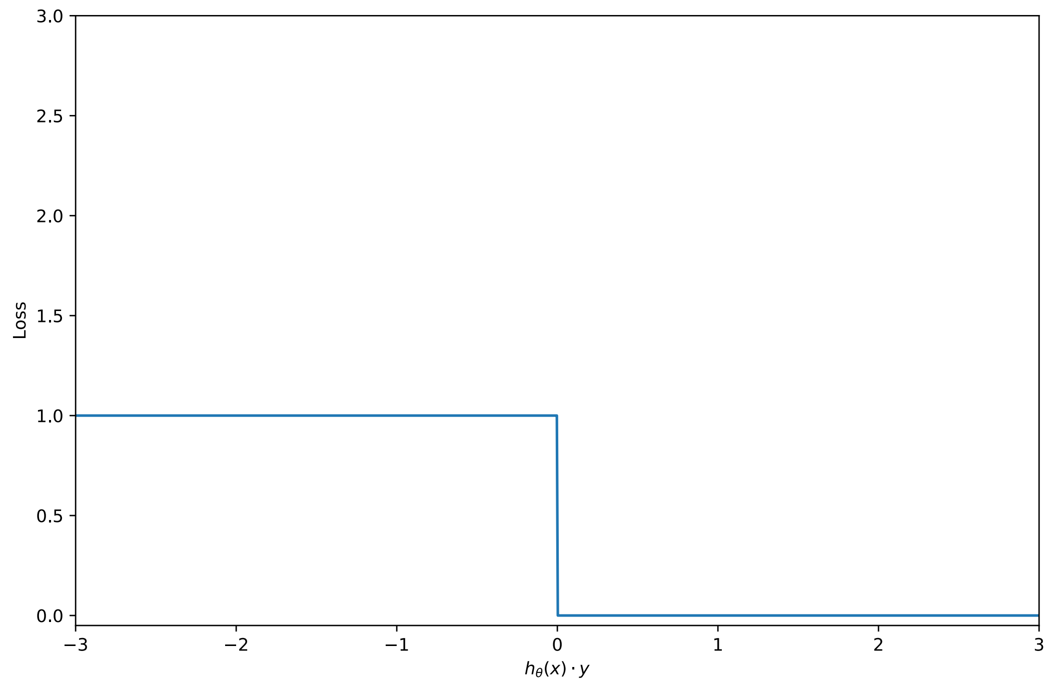
- 0/1 loss is **essentially an indicator function** that returns 1 if the item is misclassified and 0 if not – It is a **measure of classifier accuracy**
- If $h_\theta(x)$ and y have the same sign (either positive or negative), then $h_\theta(x) \cdot y$ will be positive, whereas if $h_\theta(x)$ and y have difference signs, then $h_\theta(x) \cdot y$ will be negative, and so incur a loss of one
- Mathematically,

$$\ell_{0/1}(h_\theta(x), y) = \begin{cases} 0\ if sign(h_\theta(x)) = y \\ 1\ otherwise \end{cases}$$
$$= 1\{y\, h_\theta(x) \leq 0\}$$

Universität zu Köln

## But how would you optimize 0/1 loss? – Due to the non-smoothness of 0/1 loss it cannot easily be differentiated

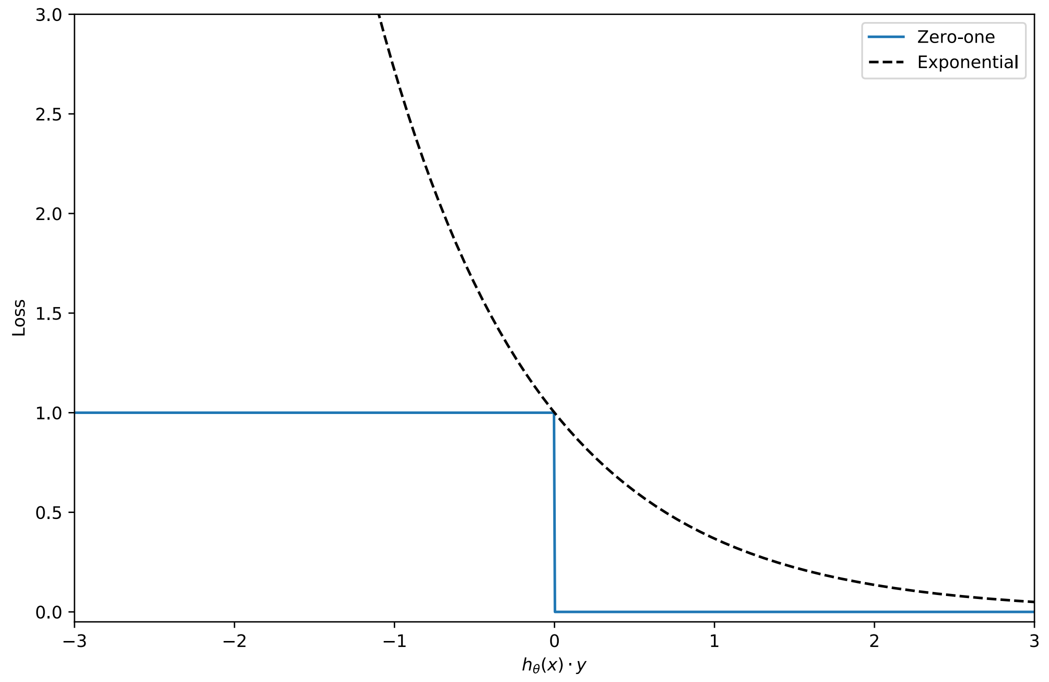
### Illustration of 0/1 loss function



- Unfortunately 0/1 loss is hard to optimize
  - 0/1 loss is non-smooth, and has derivative equal to zero everywhere except for the zero point, where the derivative is undefined
  - (NP-hard to find classifier with minimum 0/1 loss, relates to a property called convexity of the function)

## Therefore, alternative classification losses have been proposed – **Example Exponential Loss**

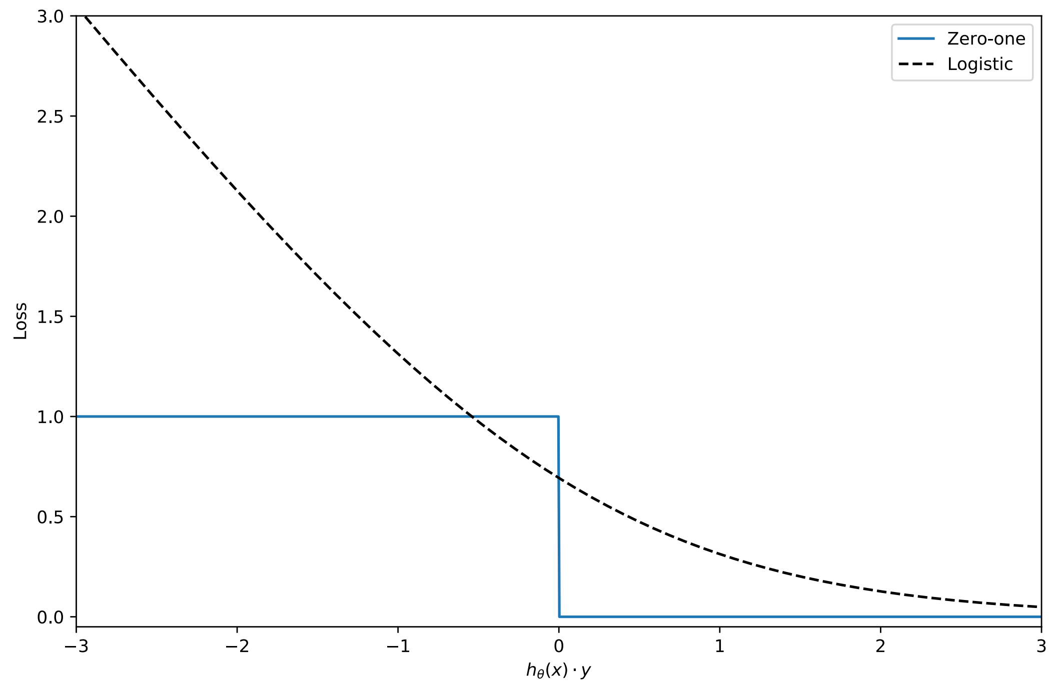
### Illustration of Exponential Loss function



- $\ell_{\text{exp}} = \exp(-y\,h_\theta(x))$
- This loss will go to zero for large $h_\theta(x) \cdot y$
- For negative $h_\theta(x) \cdot y$ the loss increases very quickly (exponentially)

# Therefore, alternative classification losses have been proposed – **Example Logistic Loss**

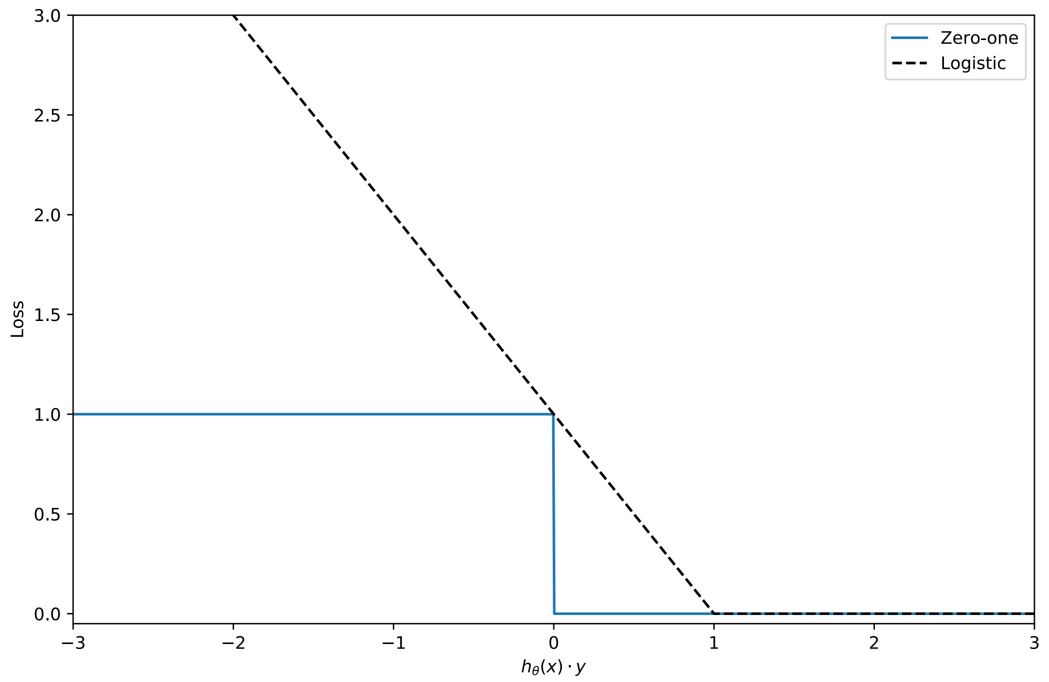
## Illustration of Logistic Loss function



- $\ell_{\text{logistic}} = \log\big(1 + \exp\big(-y\, h_\theta(x)\big)\big)$
- For large positive values of $h_\theta(x) \cdot y$ (i.e. prediction and true value are of the same class) loss will be very close to zero
- For large negative values of $h_\theta(x) \cdot y$ (i.e. large missclassification) the loss increases approximately linearly

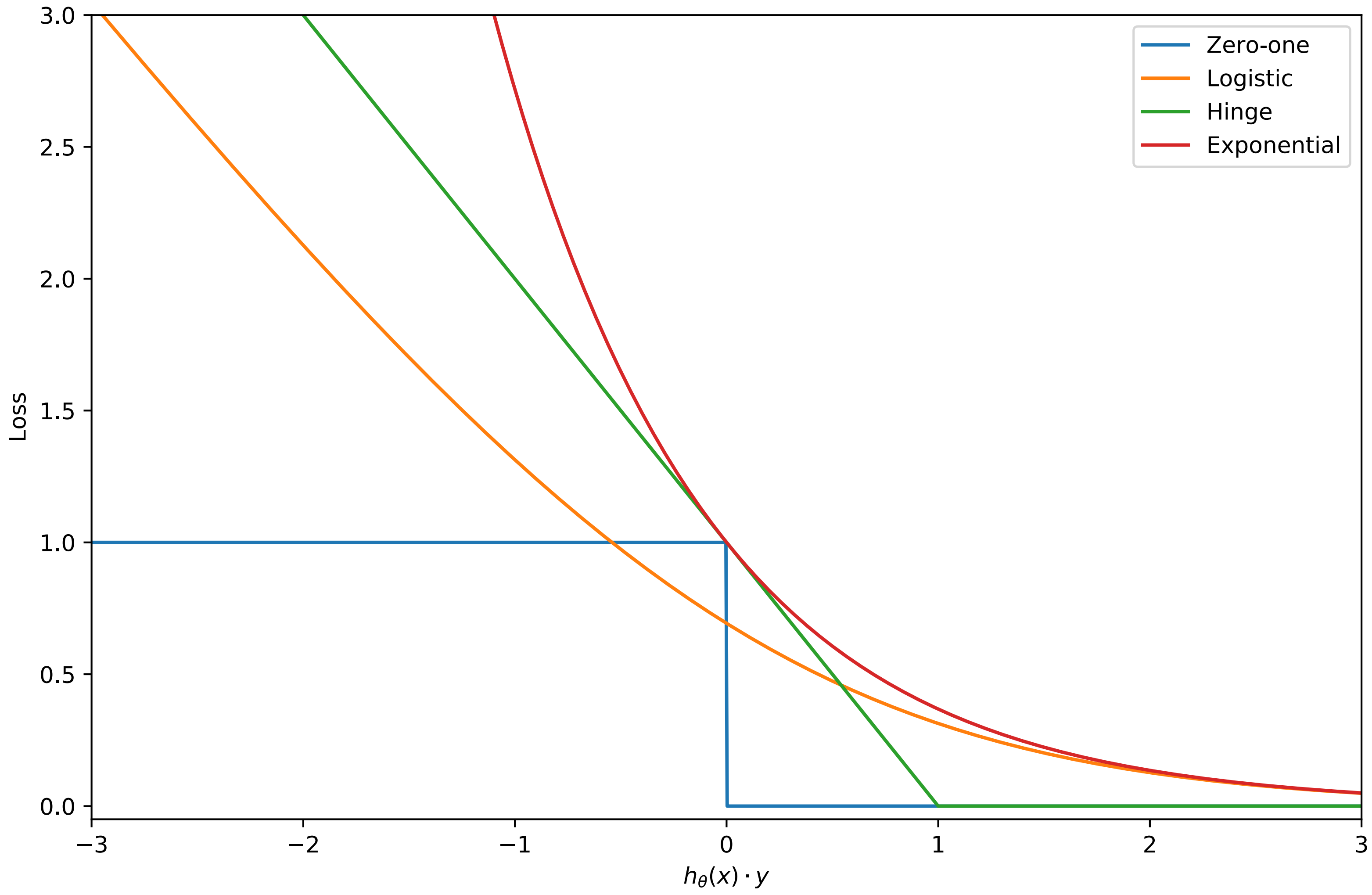
## Therefore, alternative classification losses have been proposed – **Example Hinge Loss**

### Illustration of Hinge Loss function



- $\ell_{\text{hinge}} = \max\{1 - y\, h_\theta(x), 0\}$
- As long as $h_\theta(x) \cdot y \geq 1$, this loss will be zero, whereas it will increase linearly for negative $h_\theta(x) \cdot y \geq 1x$.

## Summary: Three common differentiable loss functions for classification problems

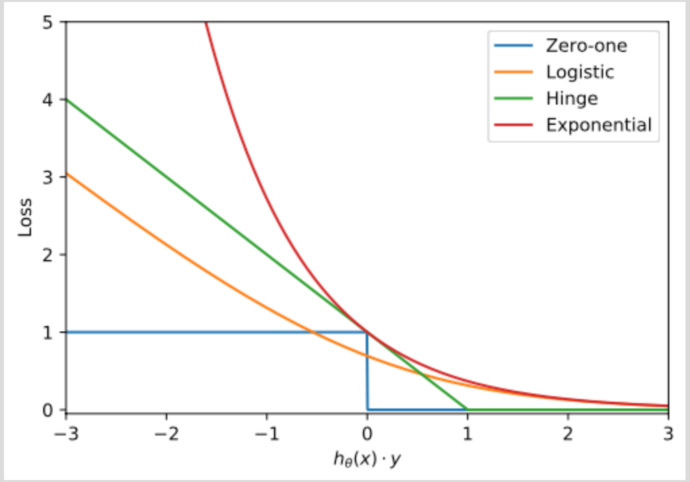


Universität zu Köln

## Poll: sensitivity to outliers

How sensitive would you estimate each of the following losses to outliers (i.e., points typically heavily misclassified)?

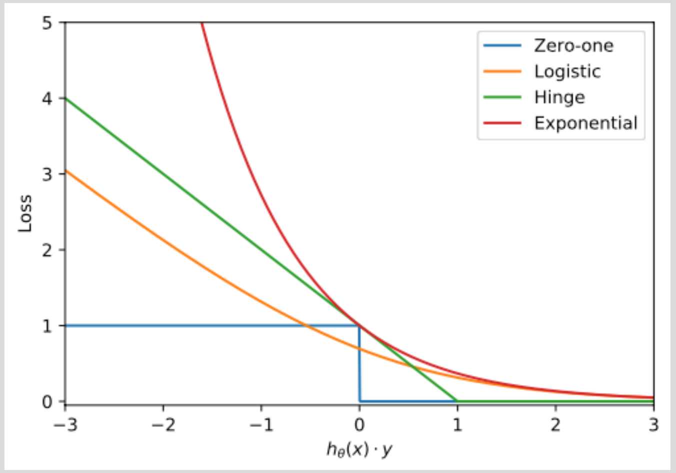
1. 0/1 < Exp < {Hinge, Logistic}
2. Exp < Hinge < Logistic < 0/1
3. Hinge< 0/1 < Logistic < Exp
[✓ 4.] 0/1 < {Hinge, Logistic} < Exp



## Poll: sensitivity to outliers

How sensitive would you estimate each of the following losses to be to outliers (i.e., points typically heavily misclassified)?

1. 0/1 < Exp < {Hinge, Logistic}
2. Exp < Hinge < Logistic < 0/1
3. Hinge< 0/1 < Logistic < Exp
4. **0/1 < {Hinge, Logistic} < Exp**

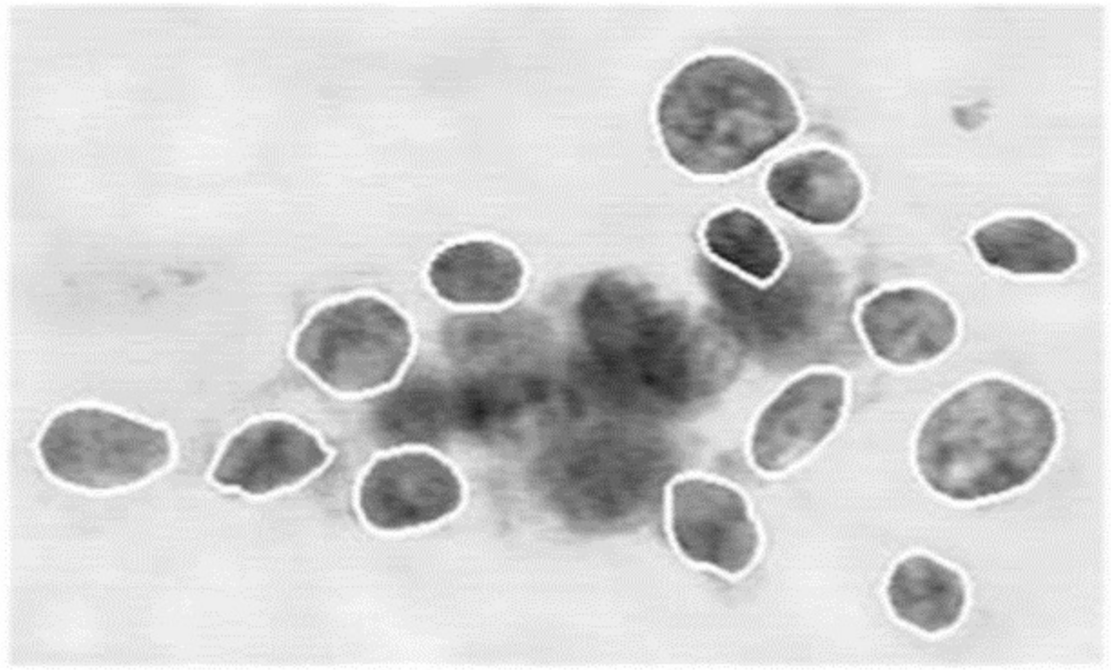


# To develop an understanding of classification we will work with a concrete example -**Breast cancer dataset**

## Breast cancer dataset

- Setting: researchers took **569 images of cancerous cells**, under a microscope, and manually selected the outlines of the different cells (this step is the kind of thing that would ideally be replaced by automatic computer vision architectures in current systems)
- Researchers then considered **10 different features of each cell**, of instance the area, perimeter, texture, number of concave points (i.e., indentations), variance of grayscale color, and some others

## Extract



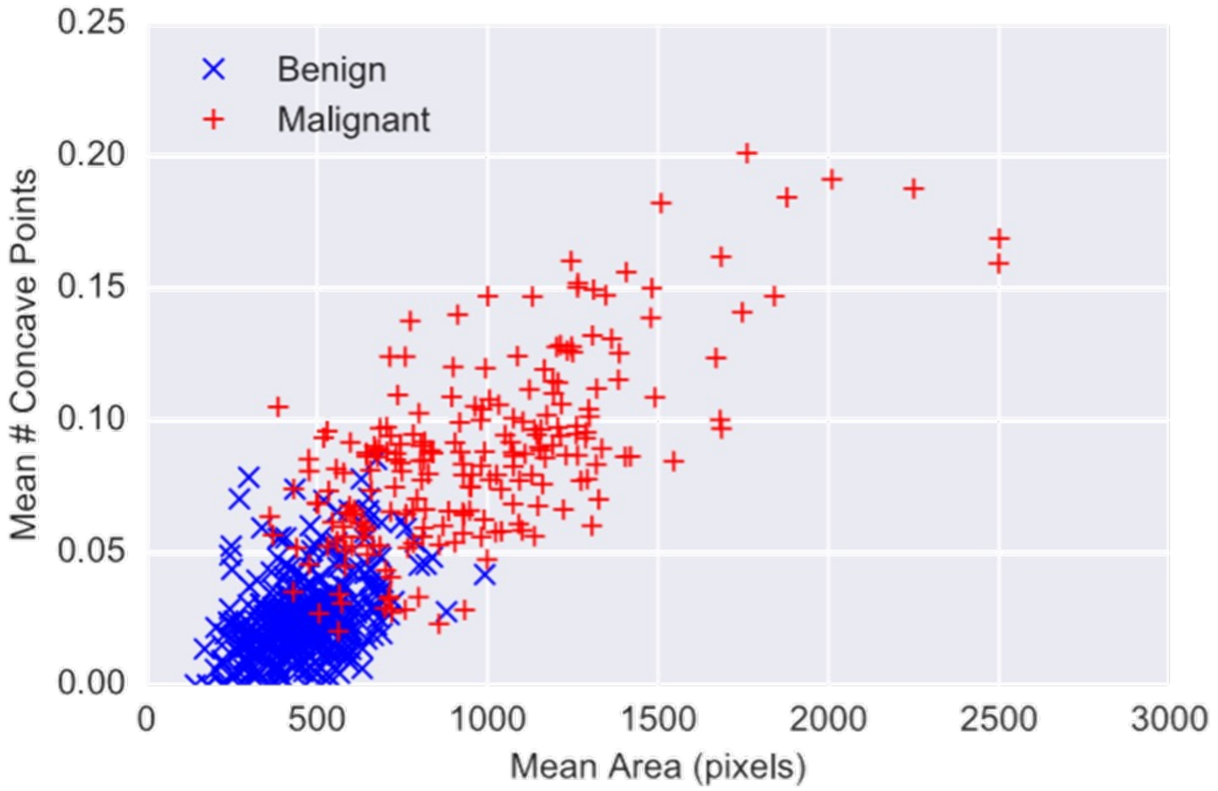
## Let us again use our common ML terminology to formally define the classification setting

| | **Term** | **Instantiation in cancer domain** |
|---|---|---|
| **Input features** | $x^{(i)} \in \mathbb{R}^n, i = 1, \dots, m$ | $x^{(i)} = \begin{bmatrix} Mean - Area^{(i)} \\ Mean - Concave - Points^{(i)} \\ 1 \end{bmatrix}$ |
| **Outputs** | $y^{(i)} \in \mathcal{Y}, i = 1, \dots, m$ | $y^{(i)} \in \{-1\ (benign), +1\ (malignant)\}$ |
| **Model parameters** | $\theta \in \mathbb{R}^n$ | $\theta = (\theta_1, \theta_2)$ |
| **Hypothesis function** | $h_\theta: \mathbb{R}^n \rightarrow \mathbb{R}$ | $h_\theta(x) = \theta^T x; \quad \hat{y} = sign(h_\theta(x))$ |
| **Loss function** | $\ell: \hat{\mathcal{Y}} \times \mathcal{Y} \rightarrow R_+$ | $\ell = \exp(-y\, h_\theta(x))$ |

## Solving classification tasks: Machine learning optimization

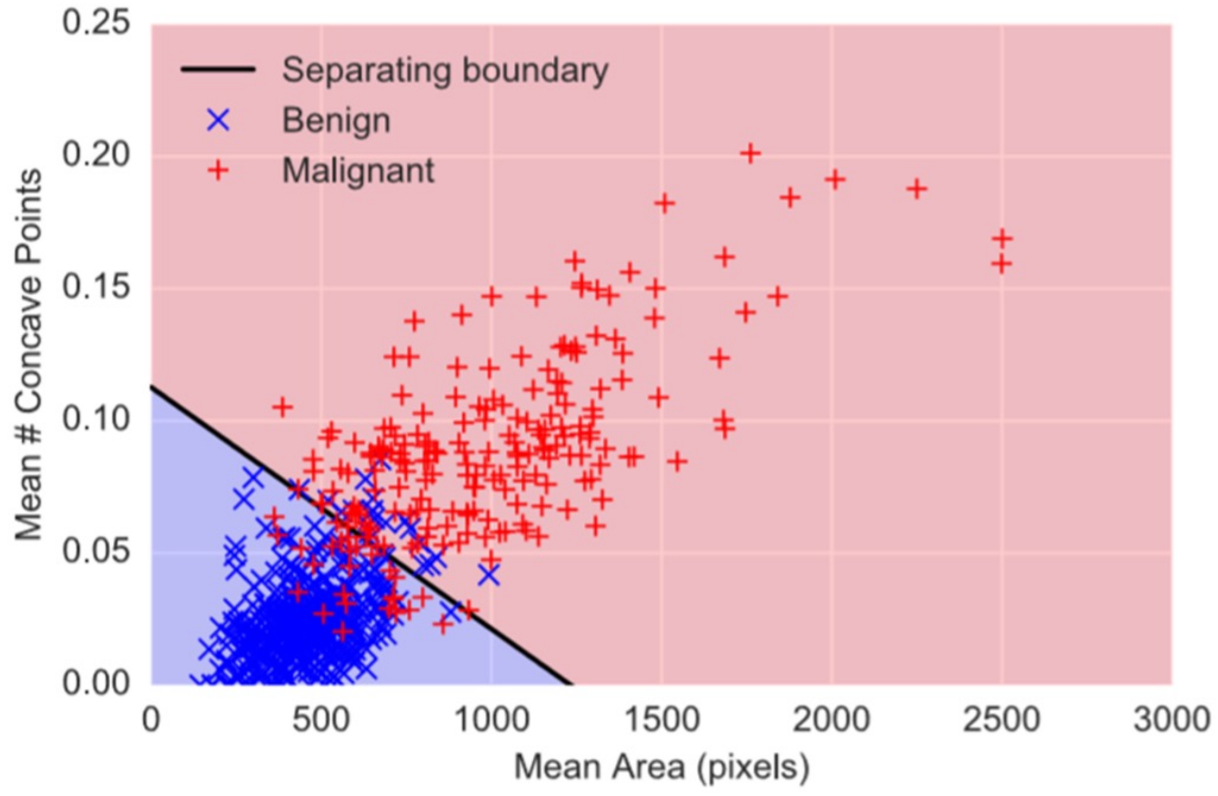
- With this notation, the "canonical" machine learning problem is written in the exact same way
  - $\underset{\theta}{Minimize} \sum_{i=1}^{m} \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$
- Unlike least squares, there is not an analytical solution to the zero gradient condition for most classification losses
- Instead, we solve these optimization problems using gradient descent (or a alternative optimization method, but we'll only consider gradient descent here)
  - $Repeat\colon\ \theta := \theta - \alpha \sum_{i=1}^{m} \nabla_\theta \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$

## Example: breast cancer classification



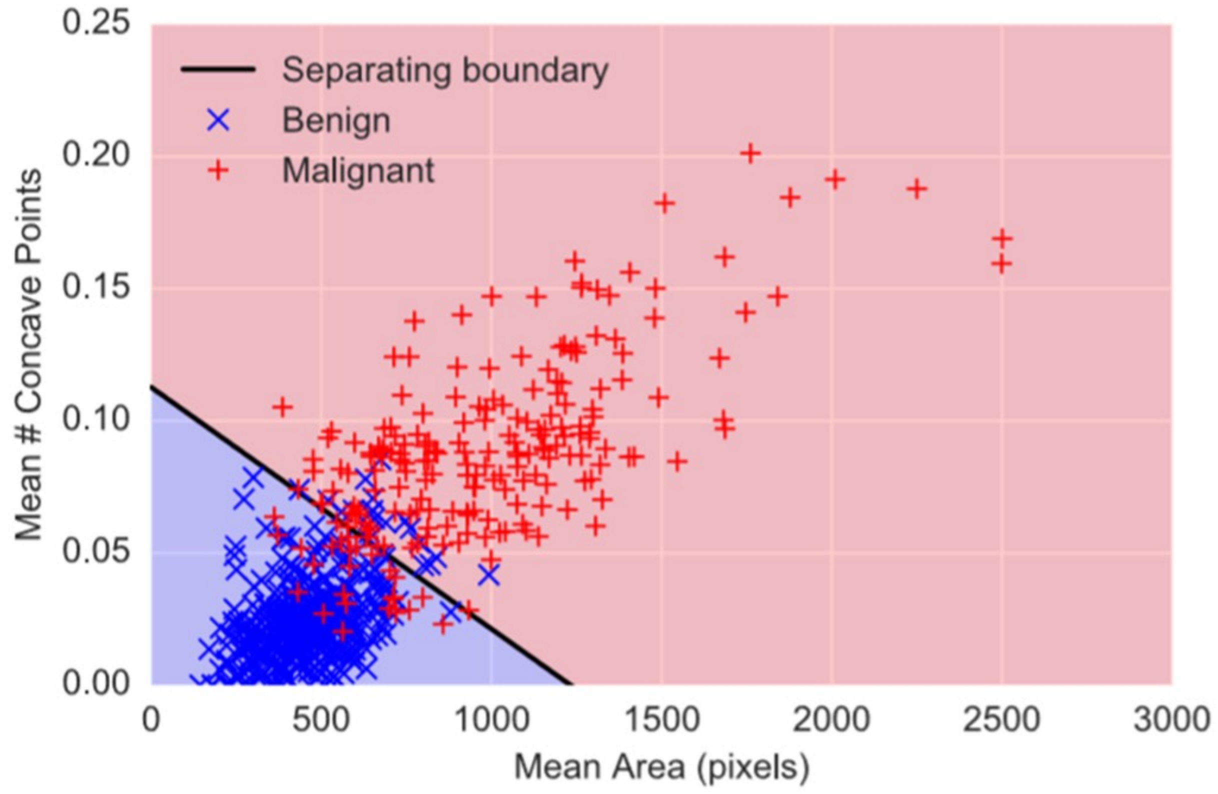
- Plot of two features: mean area vs. mean concave points, for two classes
- There is obviously some structure to the data: cells with greater average area and greater numbers of concave points are more likely to be malignant

# Linear classification example



- Linear regression ≡ "fitting a line to the data"
- **Linear classification ≡ "separating the classes with a line"**

## Understanding linear classification diagrams

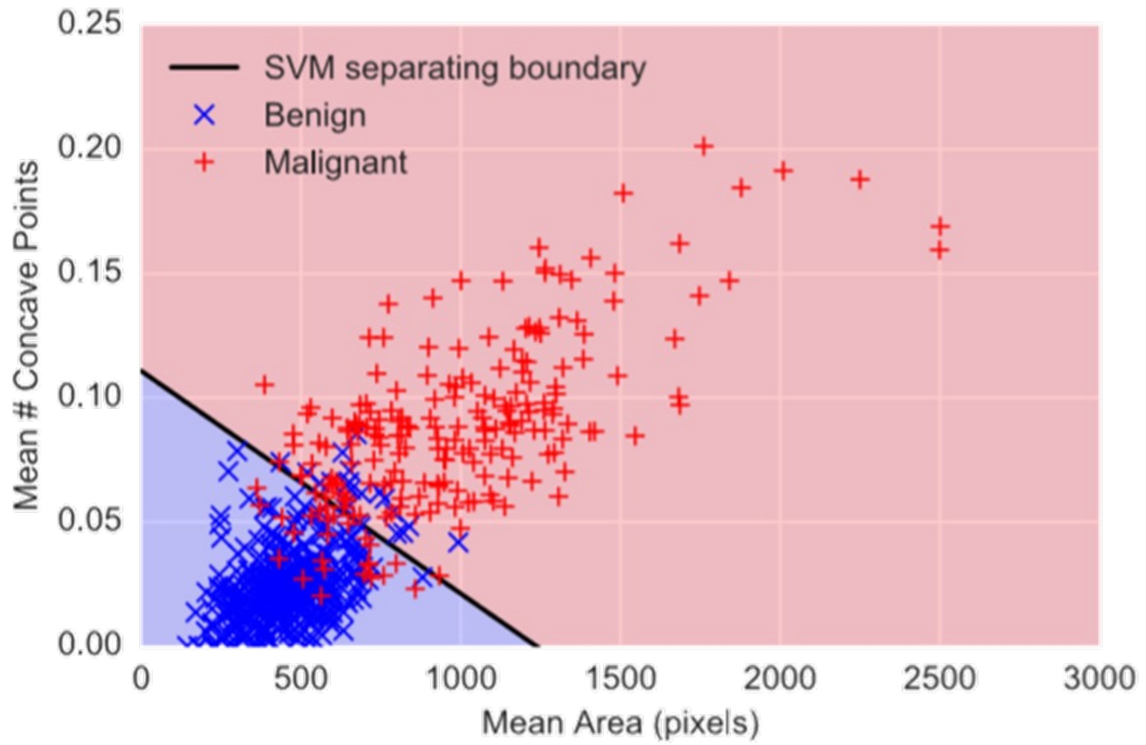


- Color shows region where the $h_\theta(x)$ is positive
- Separating boundary is given by the equation $h_\theta(x) = 0$

## Support vector machine (SVM) – Linear hypothesis function with hinge loss

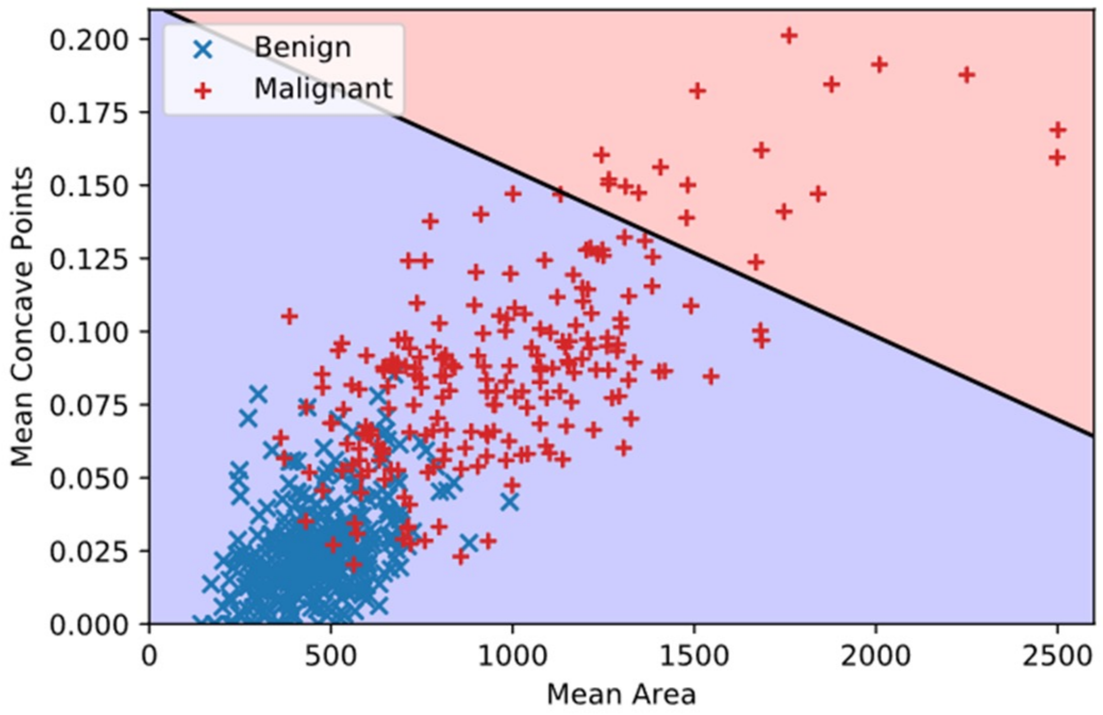
- A (linear) support vector machine (SVM) just solves the canonical machine learning optimization problem using **hinge loss** and **linear hypothesis**, **plus** an additional **regularization** term
  - $\underset{\theta}{Minimize} \sum_{i=1}^{m} max\left\{1 - y^{(i)}\theta^T x^{(i)}, 0\right\} + \frac{\lambda}{2} \|\theta\|_2^2$
- Even more precisely, the “standard” SVM does not actually regularize the "$\theta_i$" (corresponding to the constant feature, but we’ll ignore this here)
- Updates using gradient descent:
  - $\theta := \theta - \alpha \sum_{i=1}^{m} -y^{(i)} x^{(i)} 1 \left\{y^{(i)}\theta^T x^{(i)} \leq 1\right\} - \alpha\lambda\theta$

## Support vector machine example



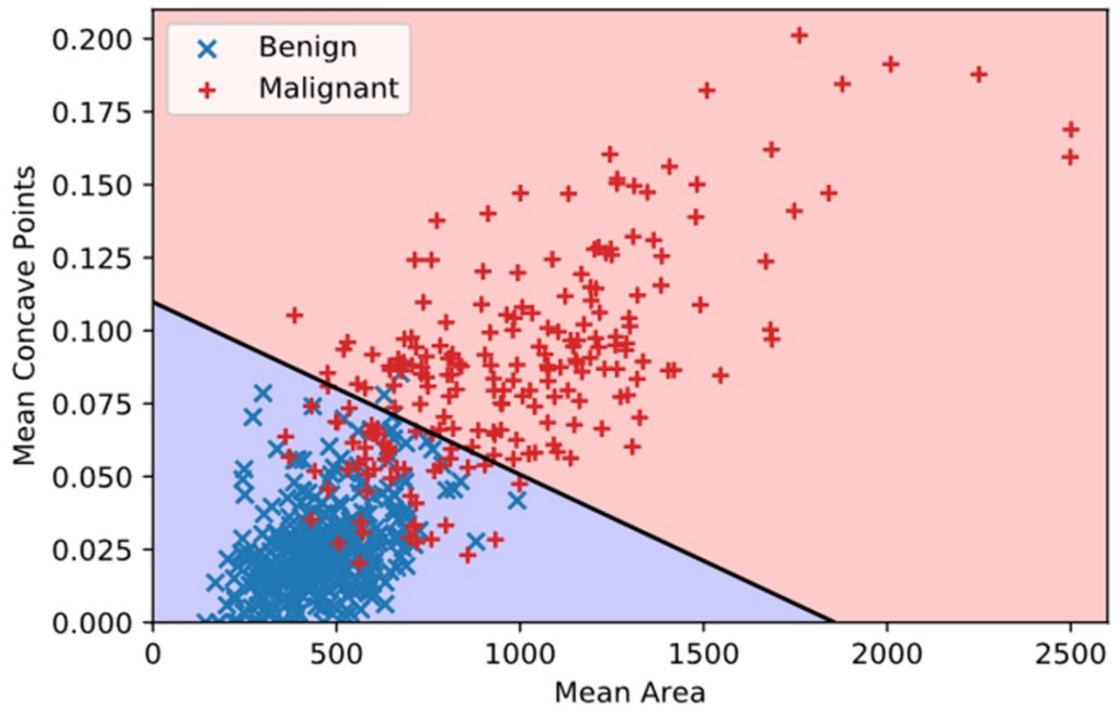
- Running support vector machine on cancer dataset, with small regularization parameter (effectively zero)
  - $\theta = \begin{bmatrix} 1.456 \\ 1.848 \\ -0.189 \end{bmatrix}$

## Support vector machine example



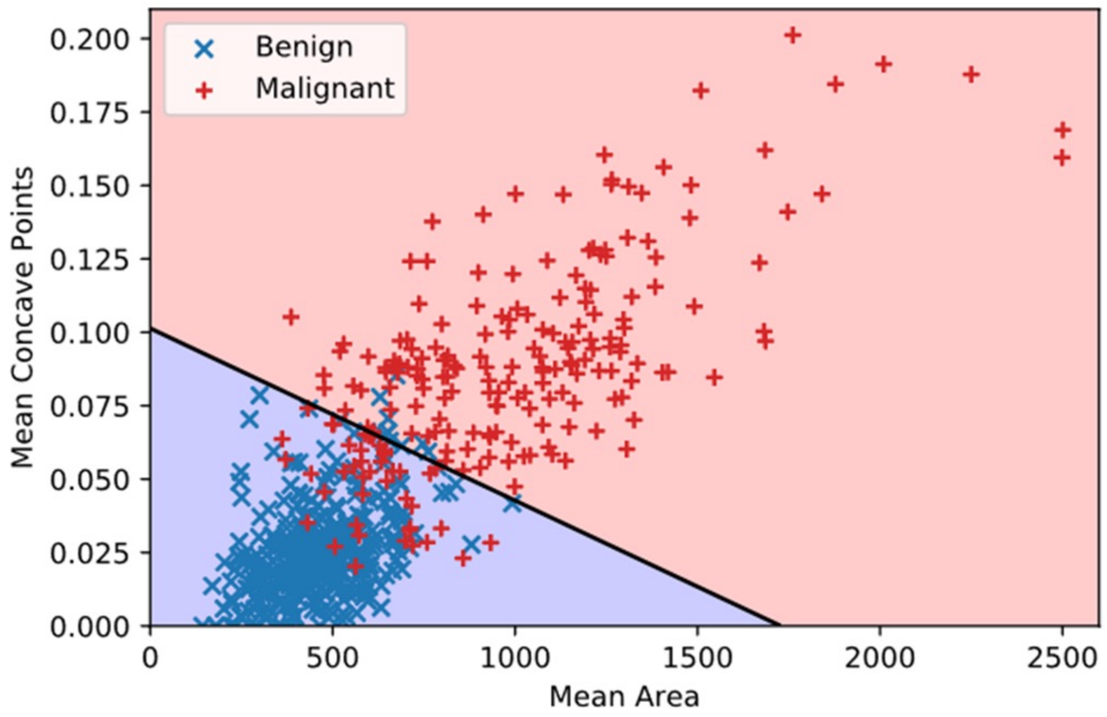
- Iterations: 10

## Support vector machine example



- Iterations: 50

# Support vector machine example

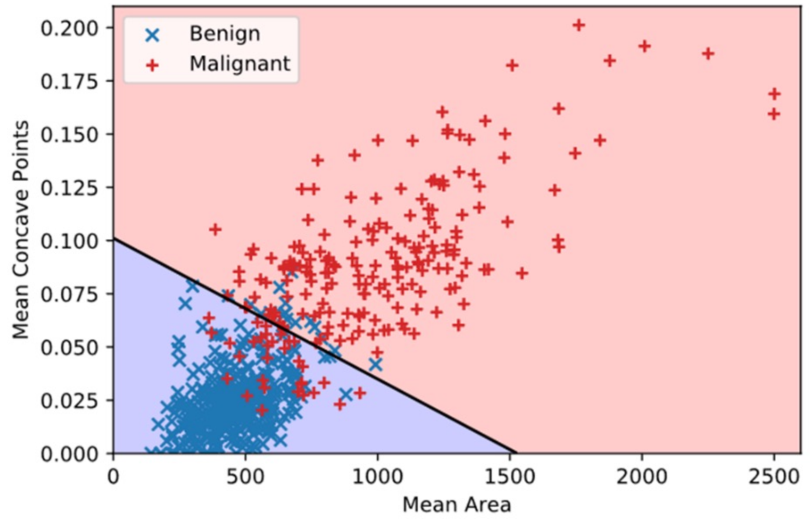


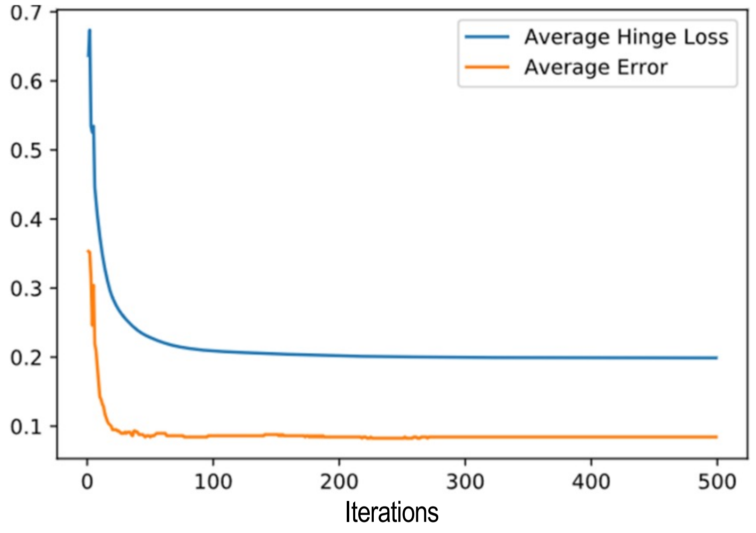
- Iterations: 100

## How to pre-process the data for classification? – A look at some alternative normalization techniques

- Thus far, we have been normalizing all columns of the data (except for the constant feature), to lie in the range [0,1]. Just to highlight two common alternatives:
  1. **Normalize features to lie in the range [−1,1]**
  2. **Normalize them to have zero mean and unit variance**
- Let’s look at the resulting behavior of the classifier for these two alternatives

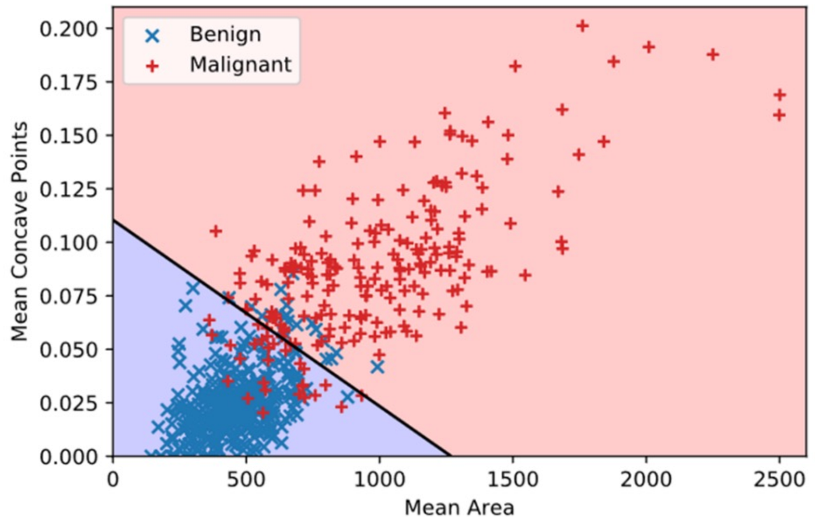
## [−1,1] feature re-scaling

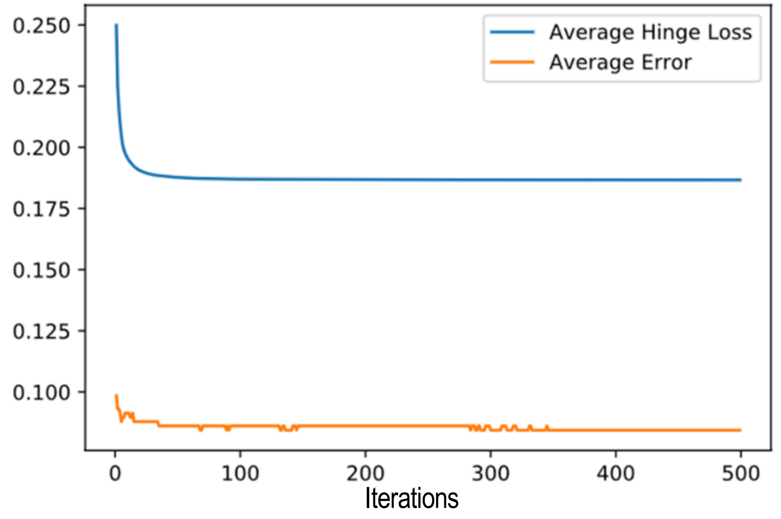




- [-1,1] is another common form of re-scaling method
- Let us look at what this means for the performance of SVM classification (hinge loss and classification error)
- Note: classification error is the ratio of wrongly classified samples vs. the full sample size

## Zero mean and unit variance re-scaling





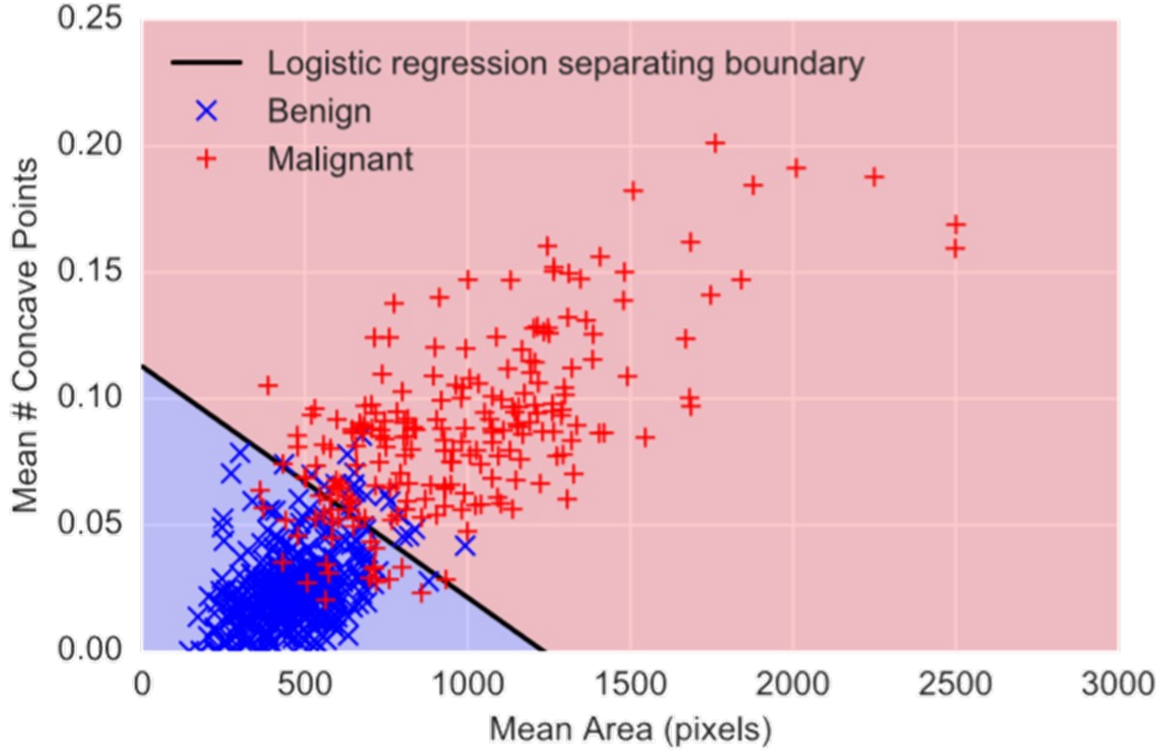
- Normalizing features to have zero mean is another common re-scaling technique for classification
- Indeed, it looks much better in this case – After just one gradient descent step, our classifier already has less than 10% error
- While the best re-scaling method is ultimately data-dependent, this strategy (normalizing columns to have zero mean and unit variance), is the most common strategy used in practice, and should be the default strategy you attempt when needing to normalize features for classification

## Logistic regression – Linear hypothesis function and logistic loss

- Logistic regression just solves this problem using logistic loss and linear hypothesis function
  - $\underset{\theta}{Minimize} \sum_{i=1}^{m} \log(1+\exp(-y^{(i)}\theta^T x^{(i)}))$
- Gradient descent updates (can you derive these?):
  - $\theta := \theta - \alpha \sum_{i=1}^{m} -y^{(i)} x^{(i)} \frac{1}{1+\exp\left(y^{(i)}\theta^T x^{(i)}\right)}$
- Logistic regression also has a nice probabilistic interpretation: certain quantities give the *probability*, under a particular model, of an example being positive or negative

We will consider this probabilistic setting more in the next lecture

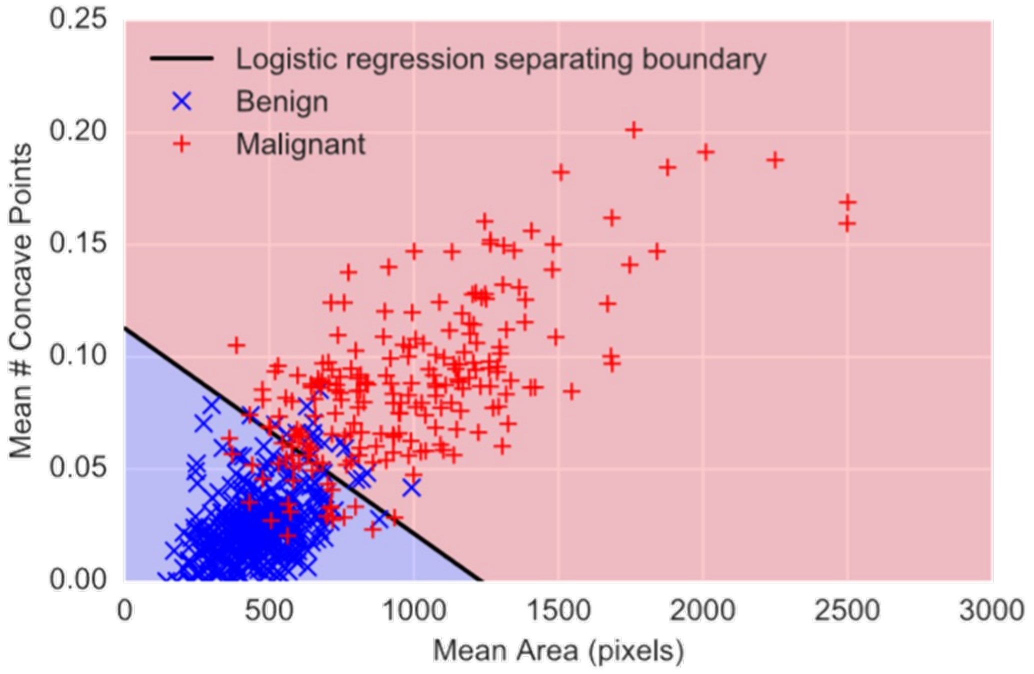
# Logistic regression example



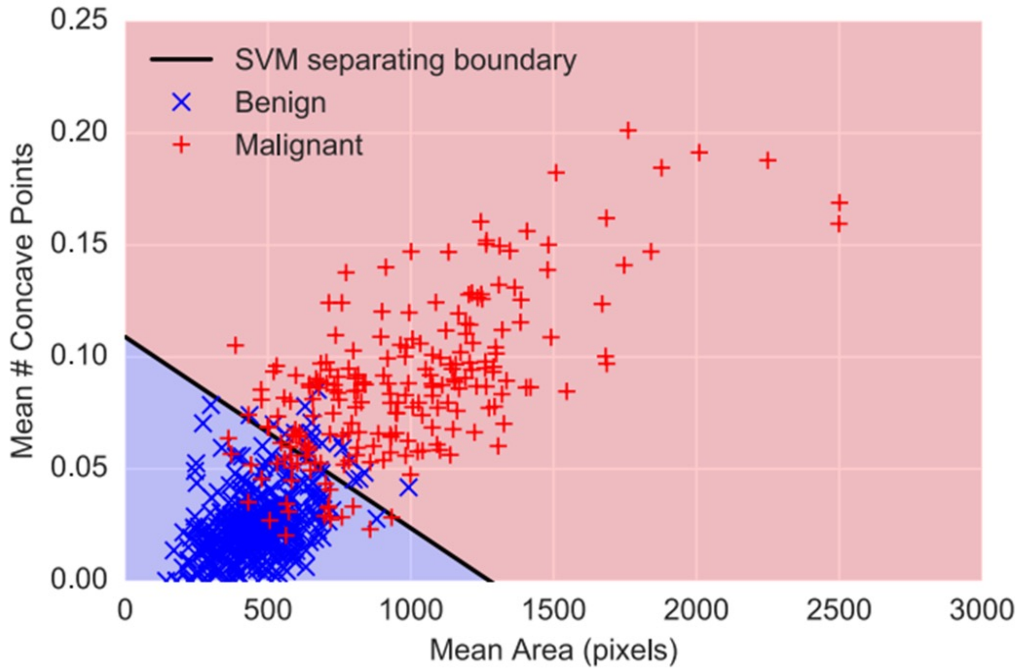
- Running logistic regression on cancer data set (small regularization)

# Logistic regression and SVM regression yield very similar results – Why?

## Logistic Regression

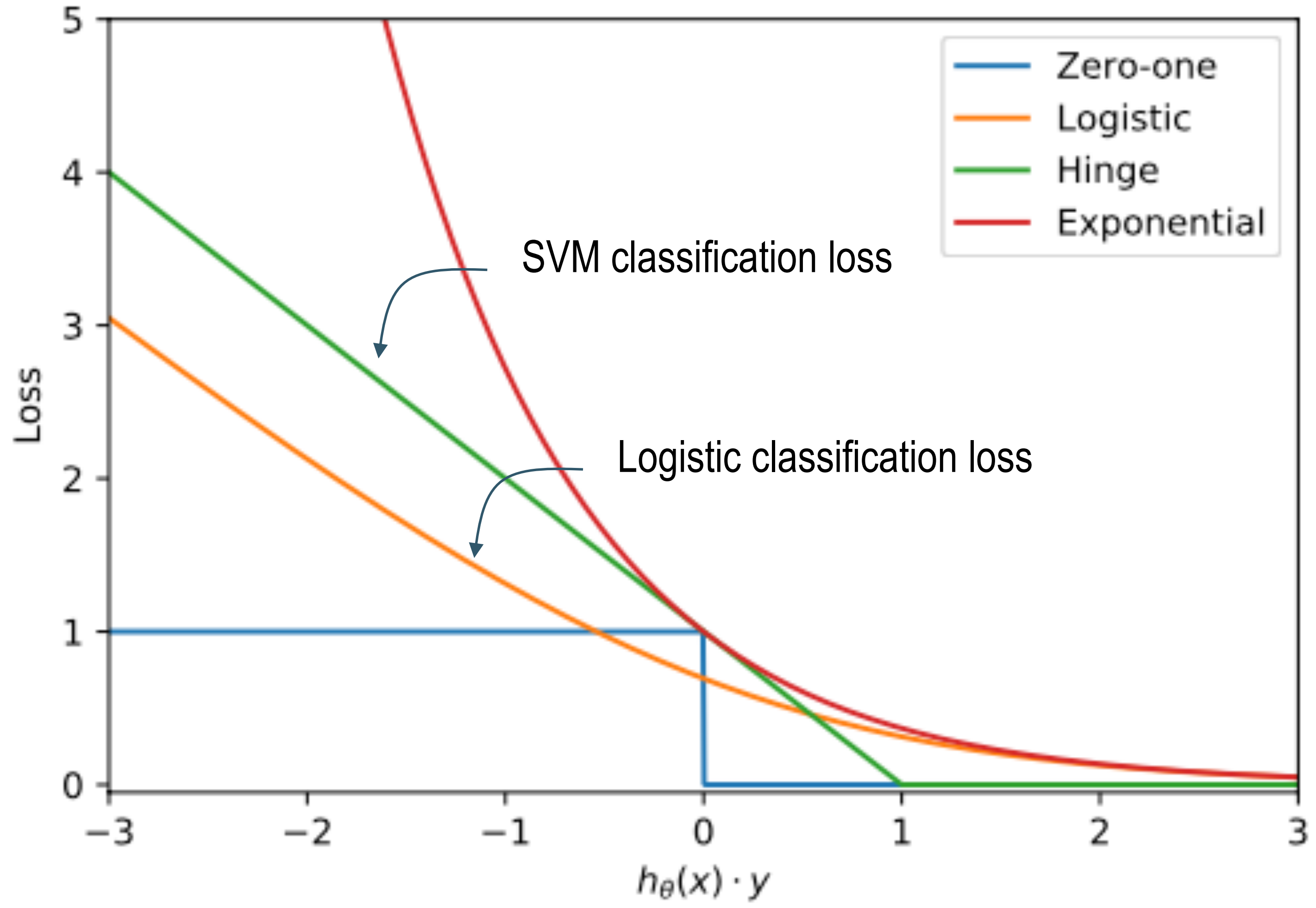


## SVM



Why do logistic regression and SVM regression yield very similar results on the breast cancer dataset?

## Logistic regression and SVM regression yield very similar results – This is because of the similar functional form of the



- SVM uses hinge loss
- Logistic regression uses logistic loss
- Both loss functions share similar properties: they both approach zero for $h_\theta(x)\cdot y$ large positive (hinge loss actually attains the zero value), and they both are approximately linear for $h_\theta(x)\cdot y$ large negative

!

Universität zu Köln

## 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