Lecture 5 pre-video

Vector norms

How we measure distances between vectors

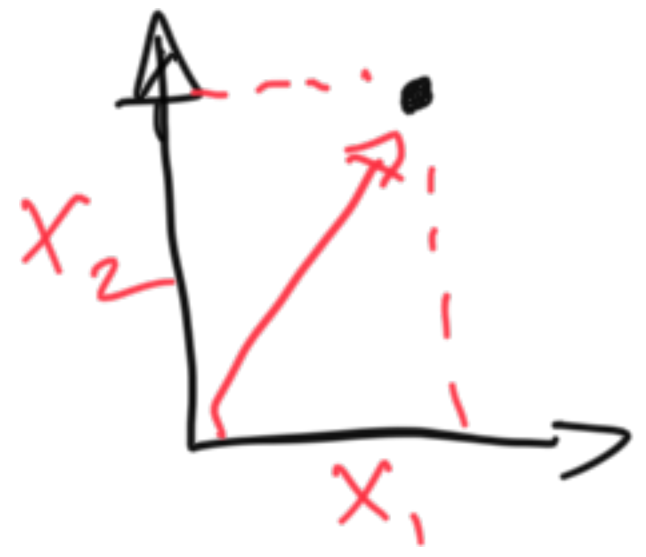
Vector norms

- Named vector norms L1, L2, … named after mathematician Henri Lebesgue (1875-1941)
- A vector norm $p : X \mapsto \mathbb{R}$ has the following properties (where X is a vector space)
    - Triangle inequality $p(x+y) \leq p(x) + p(y) \ \ \forall x, y \in X$
    - Absolute homogeneity $p(sx) = |s| p(x) \ \ \forall s \in \mathbb{R}, x \in X$
    - Positive definiteness $p(x) = 0 \iff x = 0$
    - Non-negativity $p(x) \geq 0 \ \ \forall x \in X$

Vector norms

L2: Euclidean

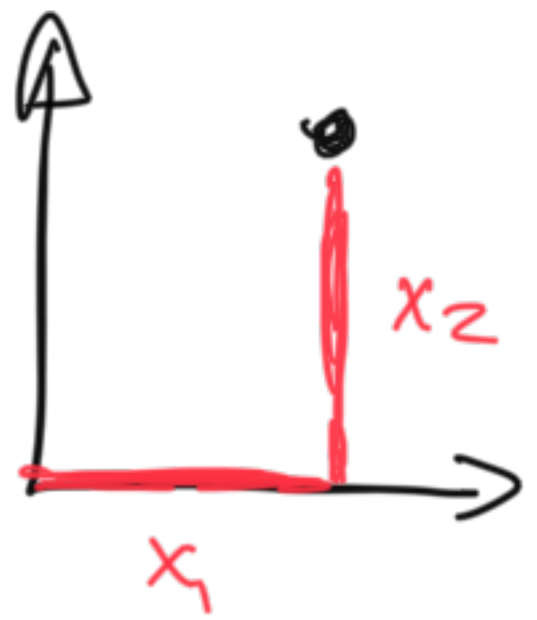
$$\|x\|_2 \Rightarrow \sqrt{x_1^2 + x_2^2 + \dots x_n^2}$$



Vector norms

L1: Absolute value / Manhattan distance

$$\|\mathbf{x}\|_1 = |x_1| + |x_2| + \dots |x_n|$$



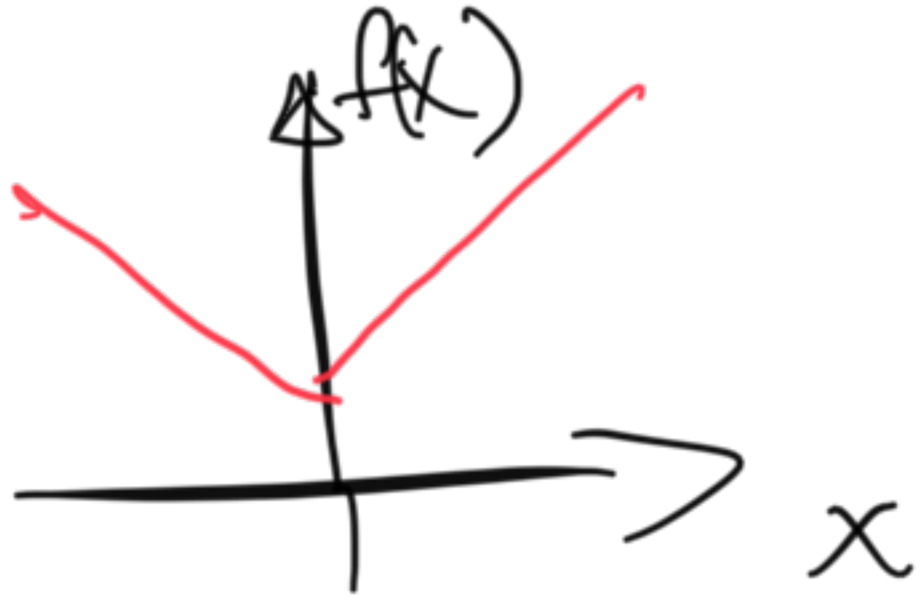
need abs otherwise violate non-negativity for vectors in lower/left quadrants

Vector norms

Application of L1 to an error function, and its derivative

for a linear fct $f(x)$

$$\| f(\mathbf{x}) \|_1$$



$$\frac{\partial \| f(\mathbf{x}) \|_1}{\partial \mathbf{x}} = \text{sign}(f(\mathbf{x})) \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$$

Vector norms

L∞

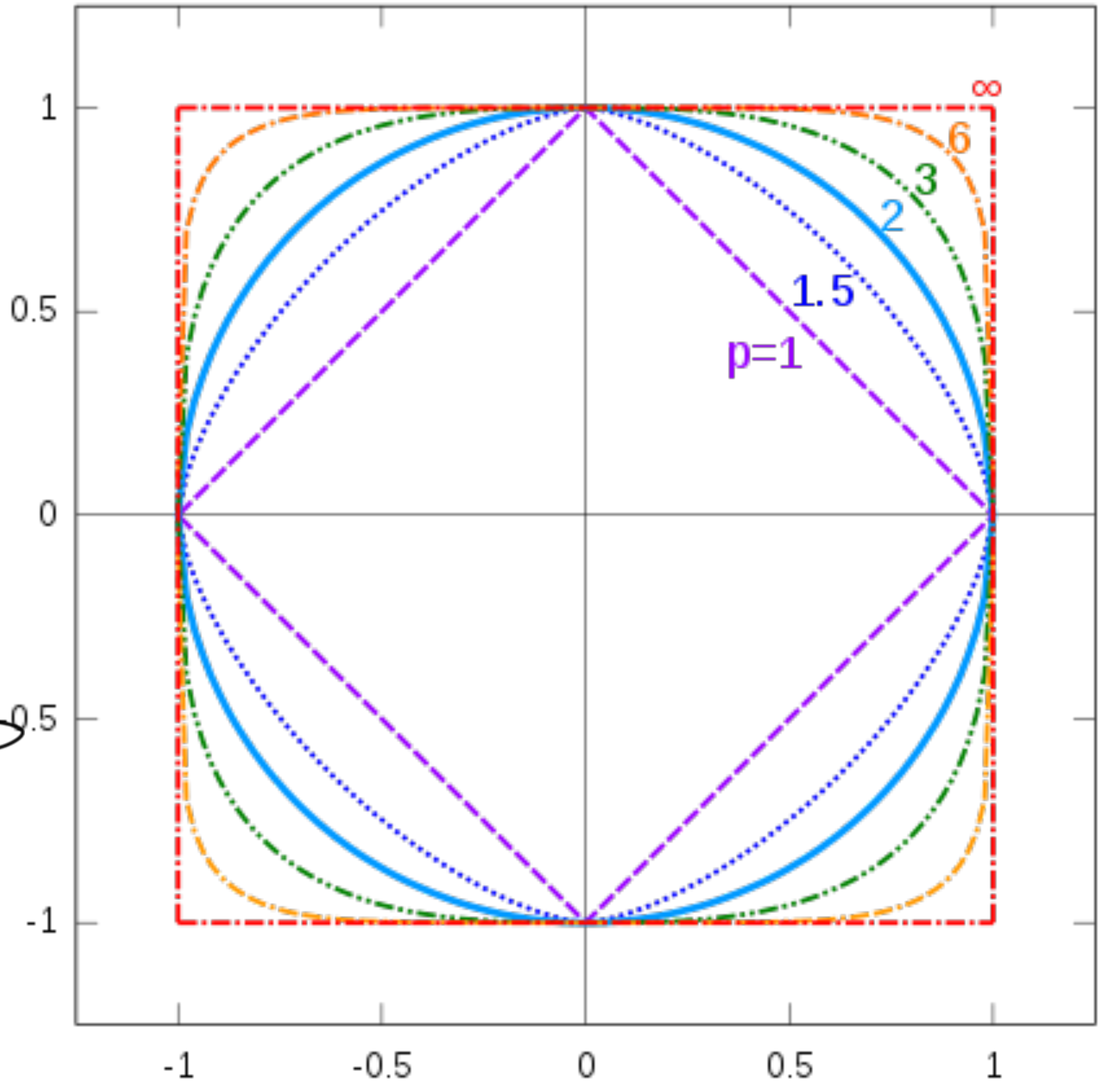
$$\|x\|_\infty = \max\left[x_1, x_2 \cdots x_n\right]$$

Vector norms

Lp

$$\|x\|_p = \left(x_1^p + x_2^p + \dots x_n^p\right)^{1/p}$$

for the $x \in \mathbb{R}^2$ where x_1, x_2 are in range $[0,1]$



Lecture #3

Regularization to prevent overfitting + robust regression to minimize outliers

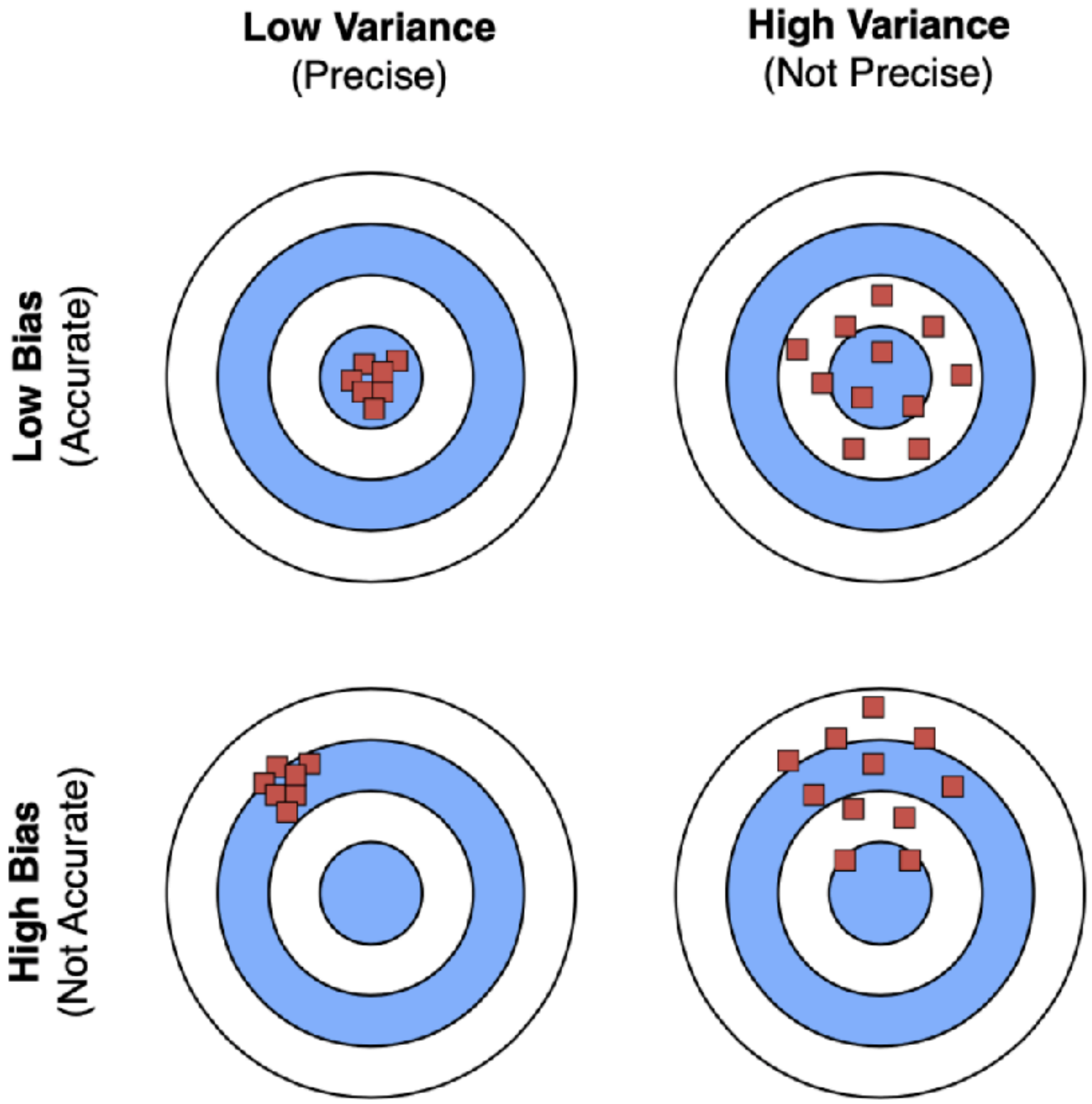
Jason G. Fleischer, Ph.D.
Asst. Teaching Professor
Department of Cognitive Science, UC San Diego

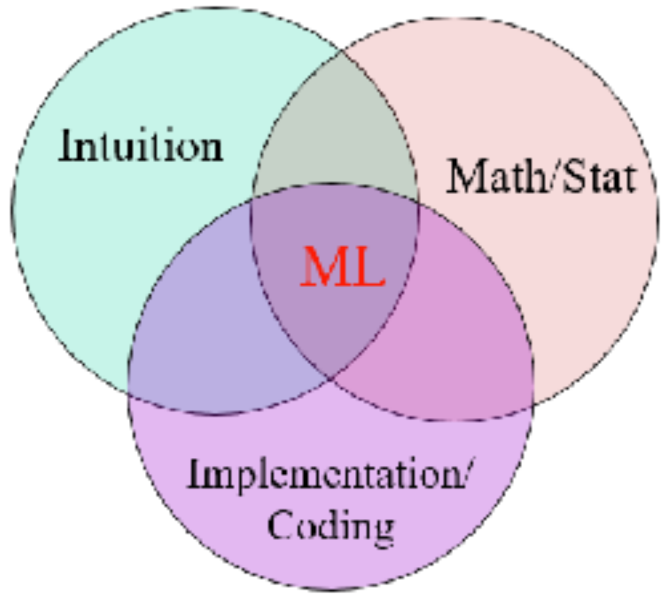
jfleischer@ucsd.edu

@jasongfleischer

https://jgfleischer.com

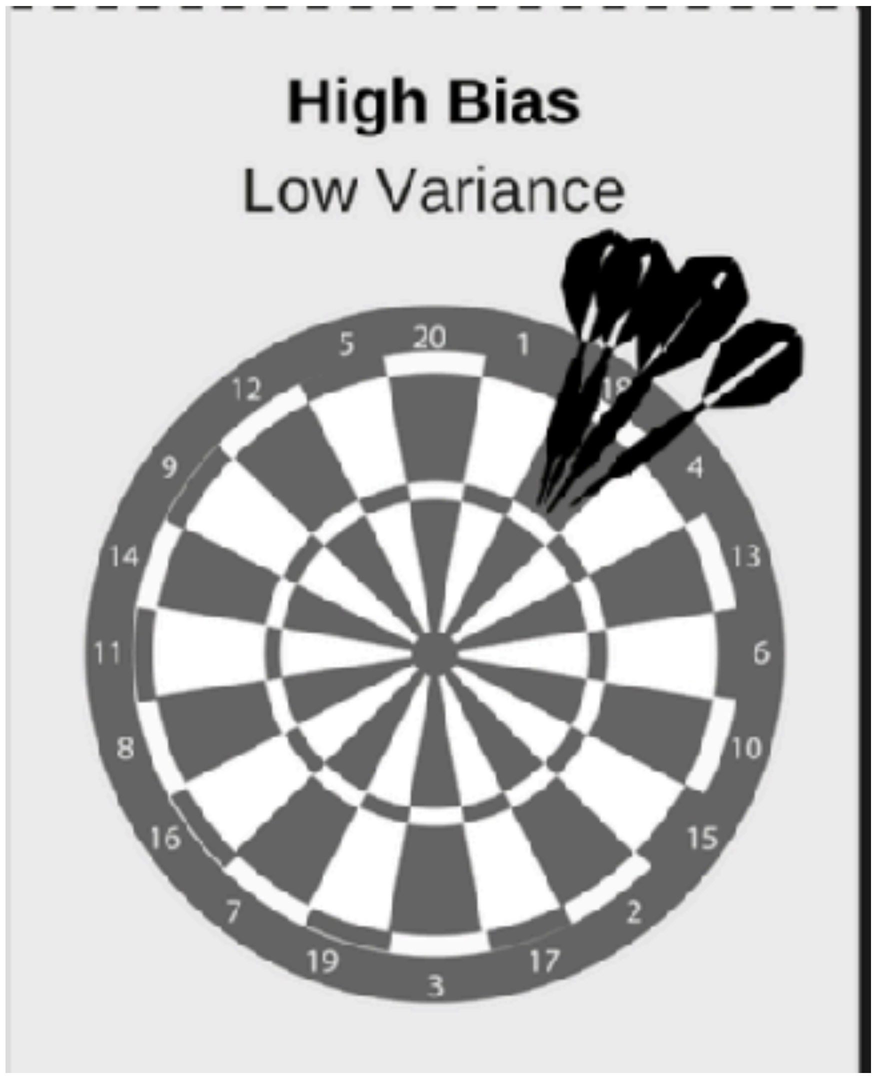
Bias Variance Tradeoff

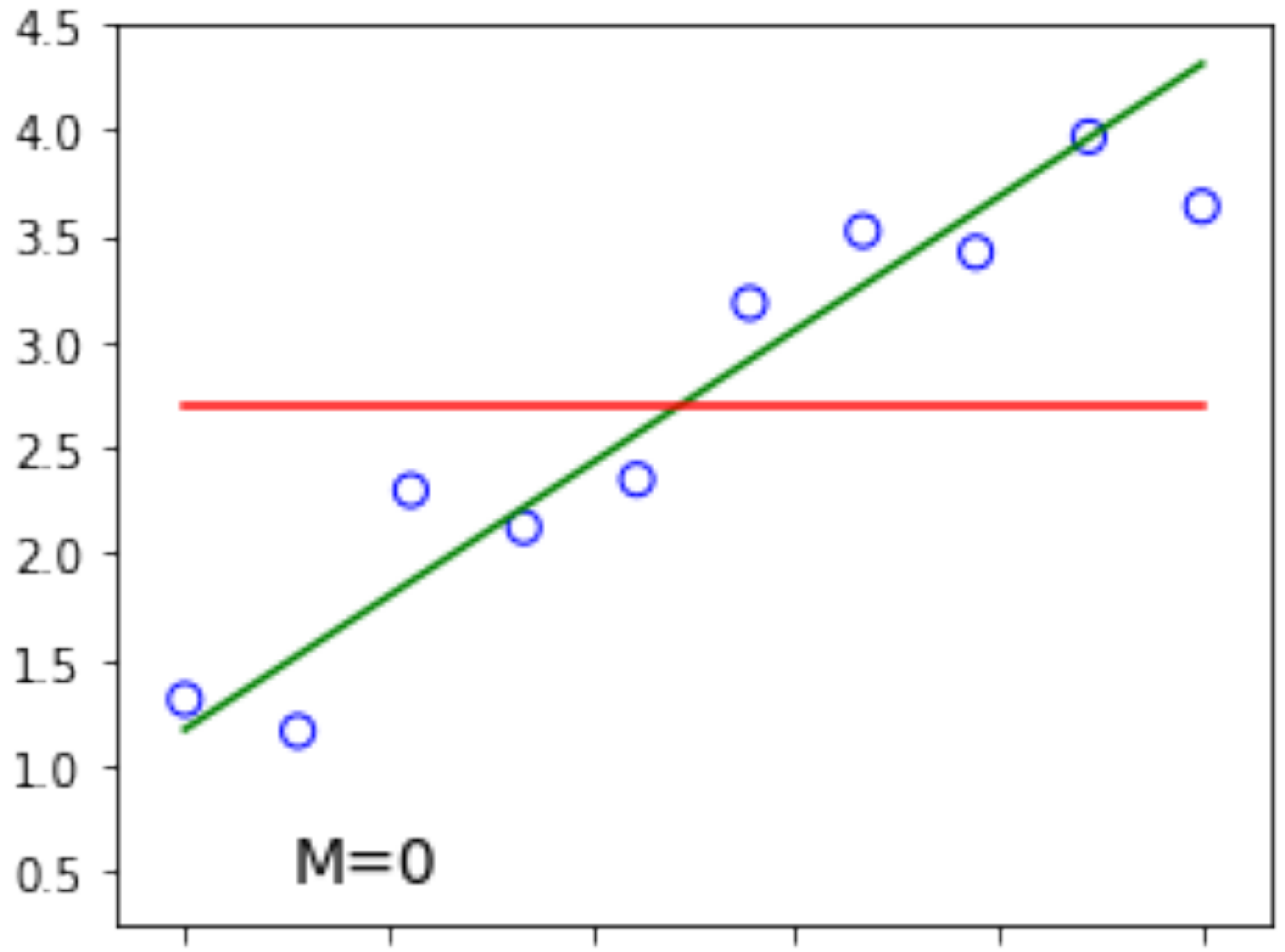




Intuition

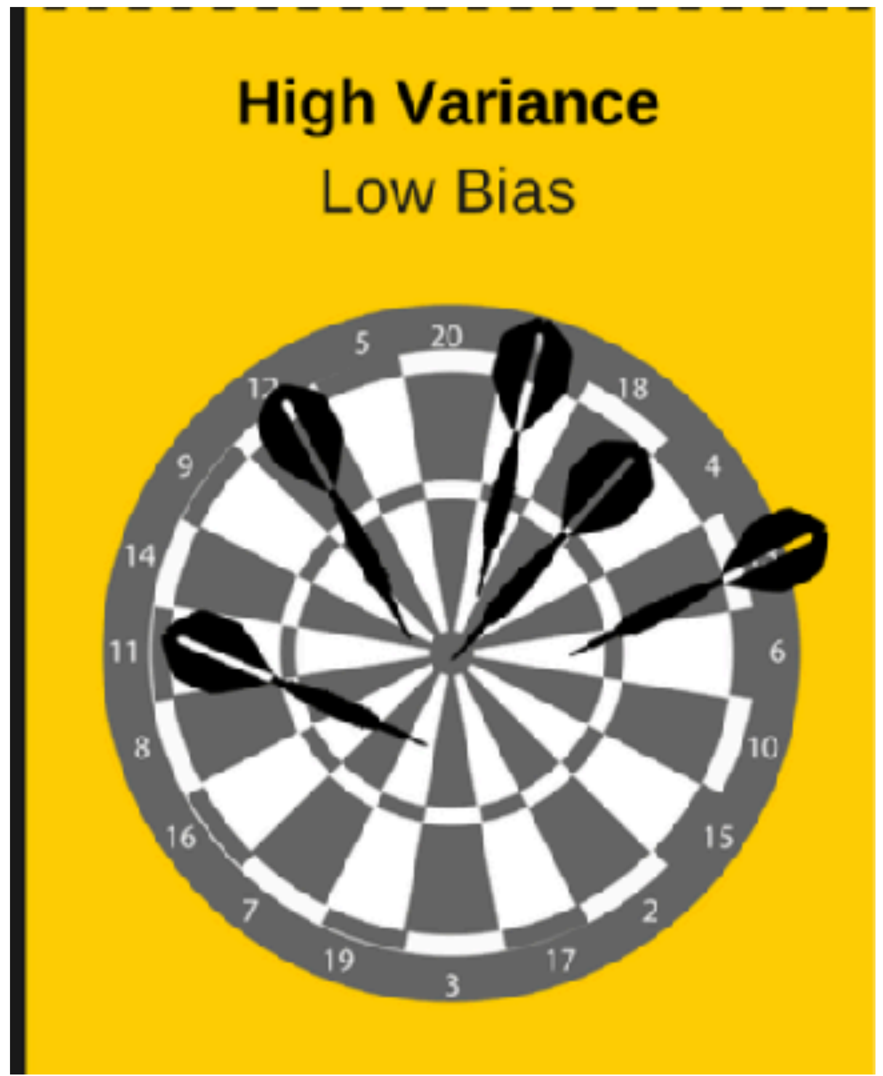
Bias-variance tradeoff in model complexity

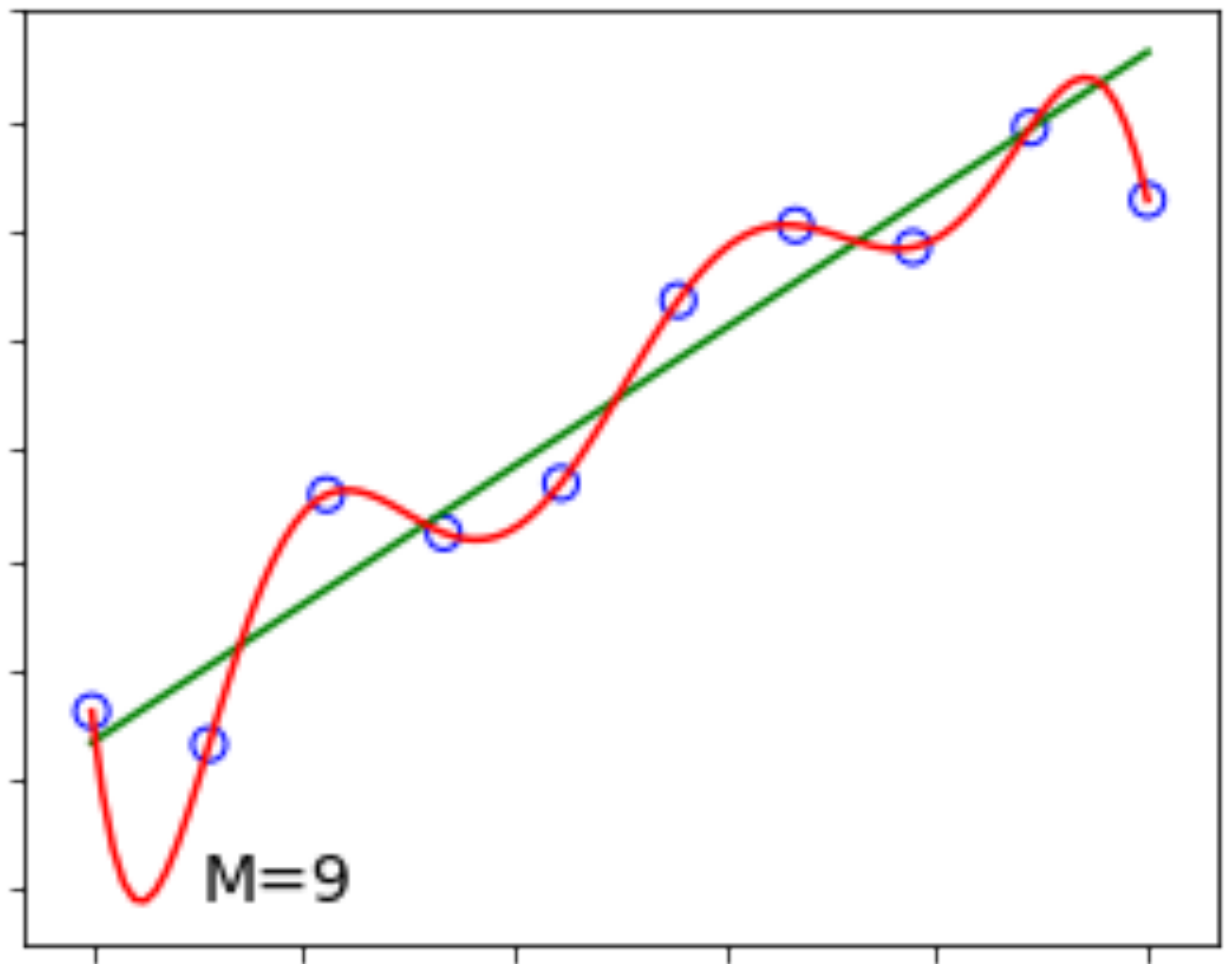




Underfitting

(Model is too simple!)



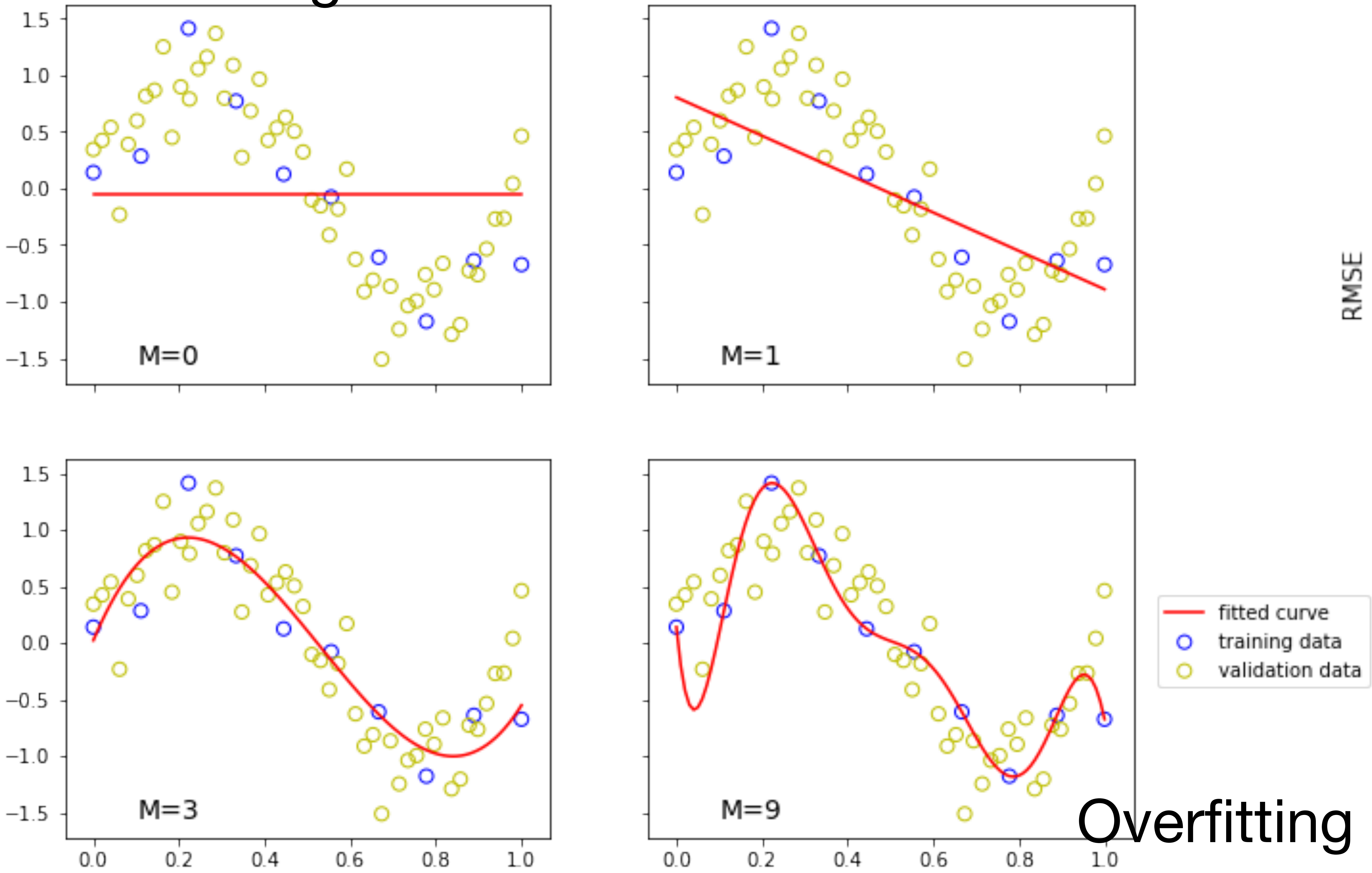


Overfitting

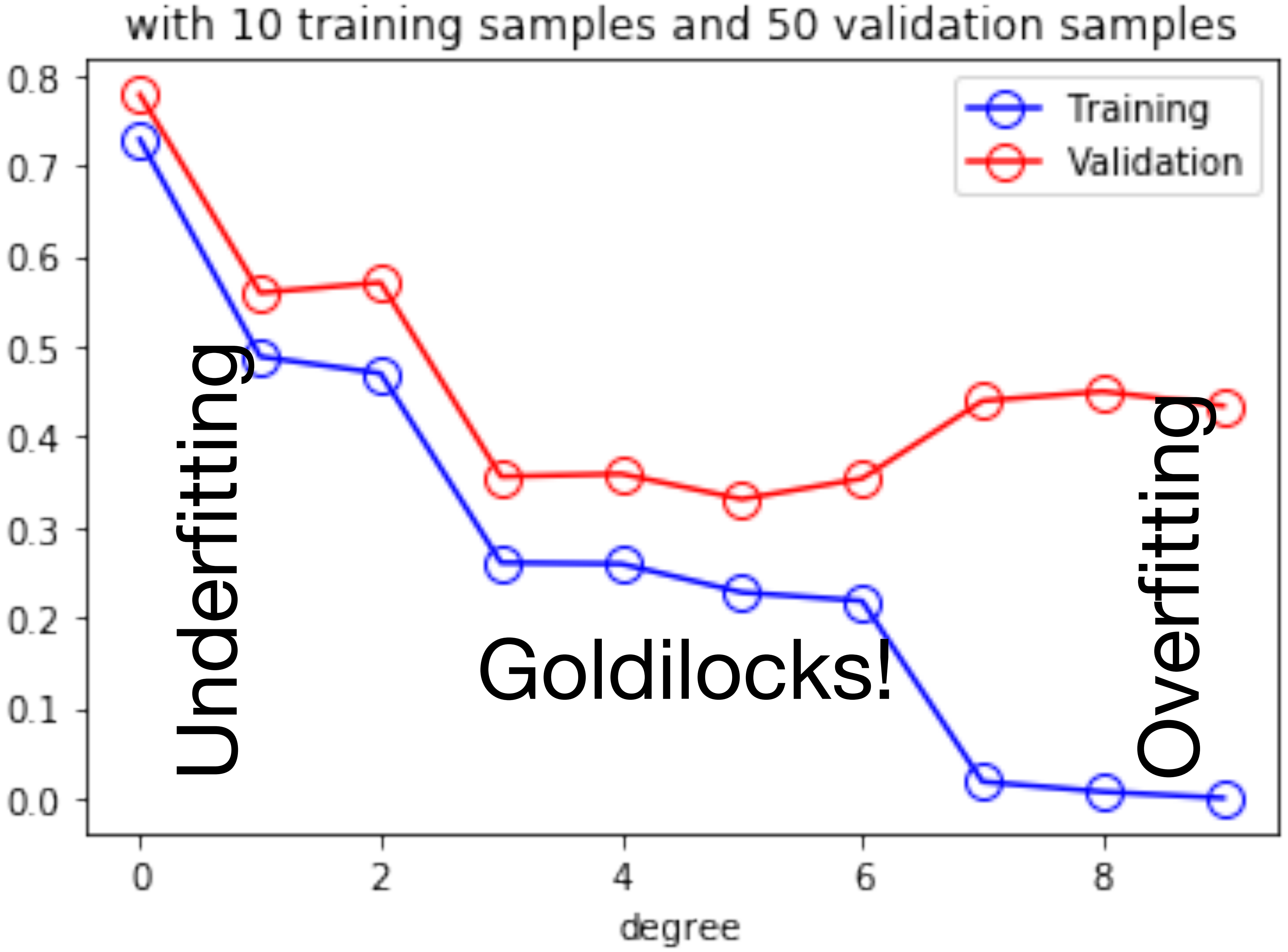
(Model is too complex!)

Training set -> set the parameters
Validation set -> try different models, select best
Test set -> how good is your chosen model

Underfitting



Overfitting



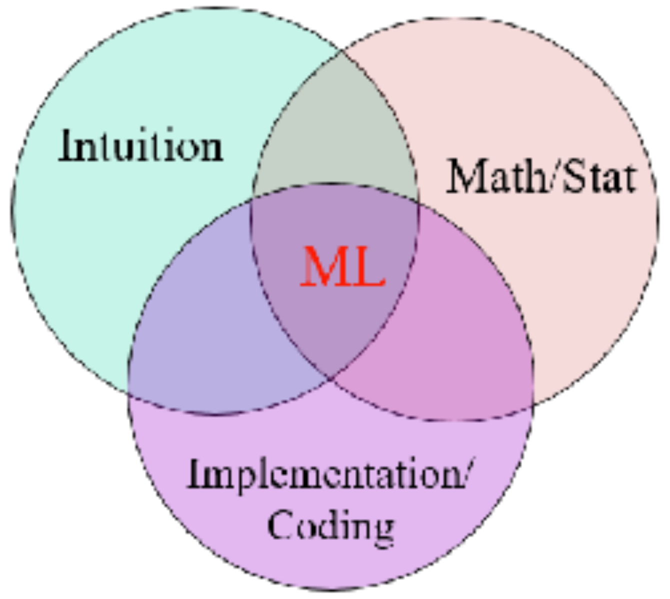
Underfitting

Goldilocks!

Overfitting

A “validation curve”

https://scikit-learn.org/stable/auto_examples/model_selection/plot_validation_curve.html#plotting-validation-curves



Implementation

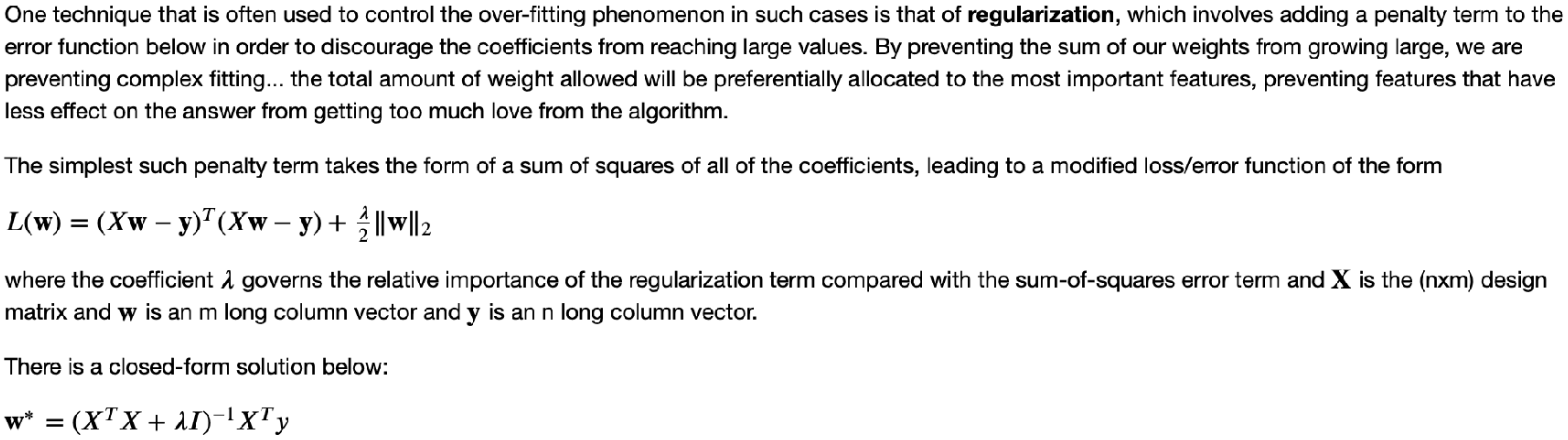
Linear Regression with Regularization

https://colab.research.google.com/github/COGS118A/demo_notebooks/blob/main/lecture_06_regularization.ipynb

https://github.com/COGS118A/demo_notebooks.git

Regularizations reduce overfitting to random noise

Penalize solutions that are too complex



One technique that is often used to control the over-fitting phenomenon in such cases is that of **regularization**, which involves adding a penalty term to the error function below in order to discourage the coefficients from reaching large values. By preventing the sum of our weights from growing large, we are preventing complex fitting... the total amount of weight allowed will be preferentially allocated to the most important features, preventing features that have less effect on the answer from getting too much love from the algorithm.

The simplest such penalty term takes the form of a sum of squares of all of the coefficients, leading to a modified loss/error function of the form

$$L(\mathbf{w}) = (X\mathbf{w} - \mathbf{y})^T(X\mathbf{w} - \mathbf{y}) + \frac{\lambda}{2}\|\mathbf{w}\|_2$$

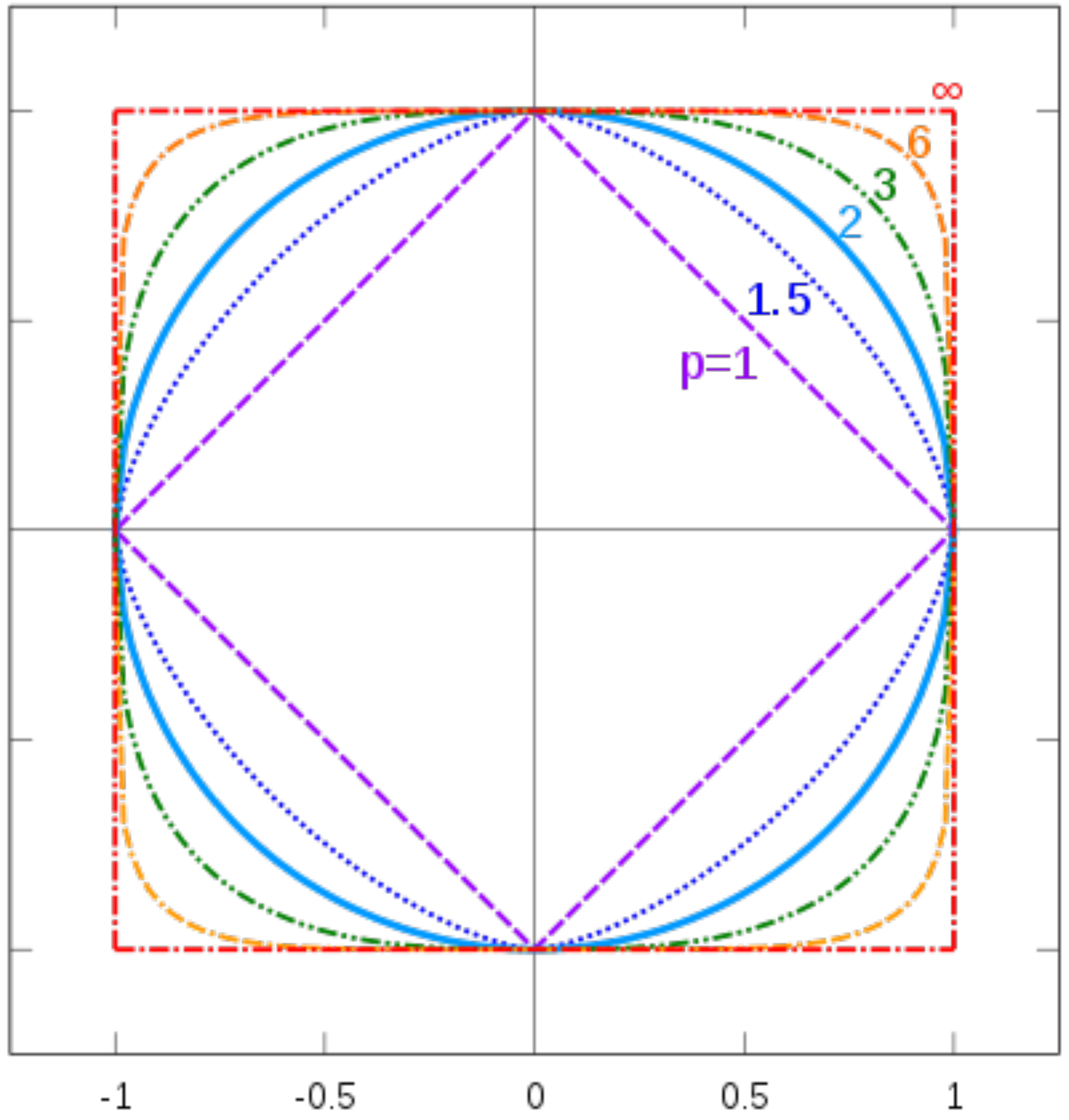
where the coefficient λ governs the relative importance of the regularization term compared with the sum-of-squares error term and $\mathbf{X}$ is the (nxm) design matrix and $\mathbf{w}$ is an m long column vector and $\mathbf{y}$ is an n long column vector.

There is a closed-form solution below:

$$\mathbf{w}^* = (X^T X + \lambda I)^{-1} X^T y$$

Regularizations reduce overfitting to random noise

Penalize solutions that are too complex



You can also do L1 regularization which is often called LASSO regression (least absolute shrinkage and selection operator)

$$L(\mathbf{w}) = (X\mathbf{w} - \mathbf{y})^T(X\mathbf{w} - \mathbf{y}) + \frac{\lambda}{2}\|\mathbf{w}\|_1$$

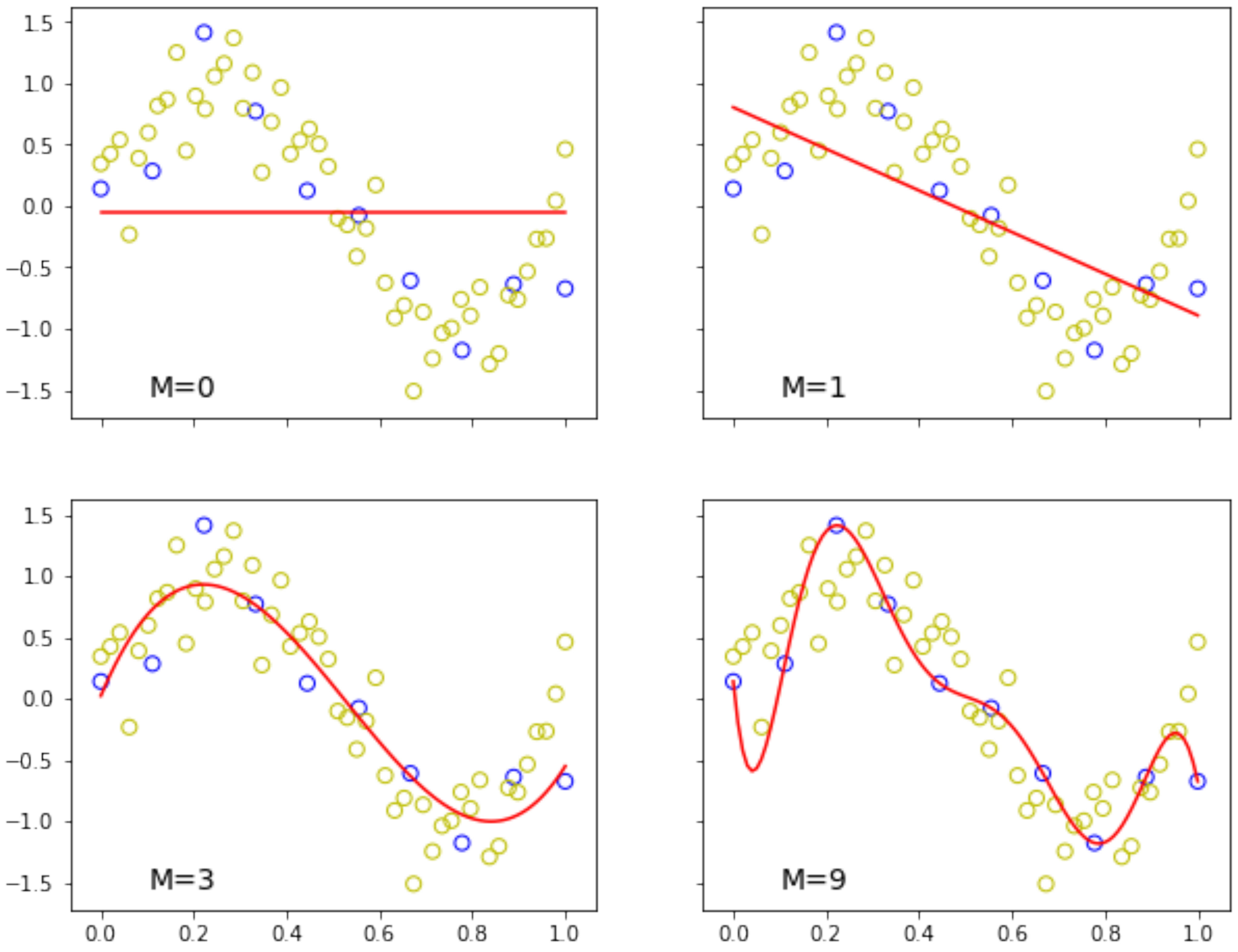
And you can combine the two together in a technique called ElasticNet

$$L(\mathbf{w}) = (X\mathbf{w} - \mathbf{y})^T(X\mathbf{w} - \mathbf{y}) + \frac{\lambda}{2}(\alpha\|\mathbf{w}\|_1 + (1-\alpha)\|\mathbf{w}\|_2)$$

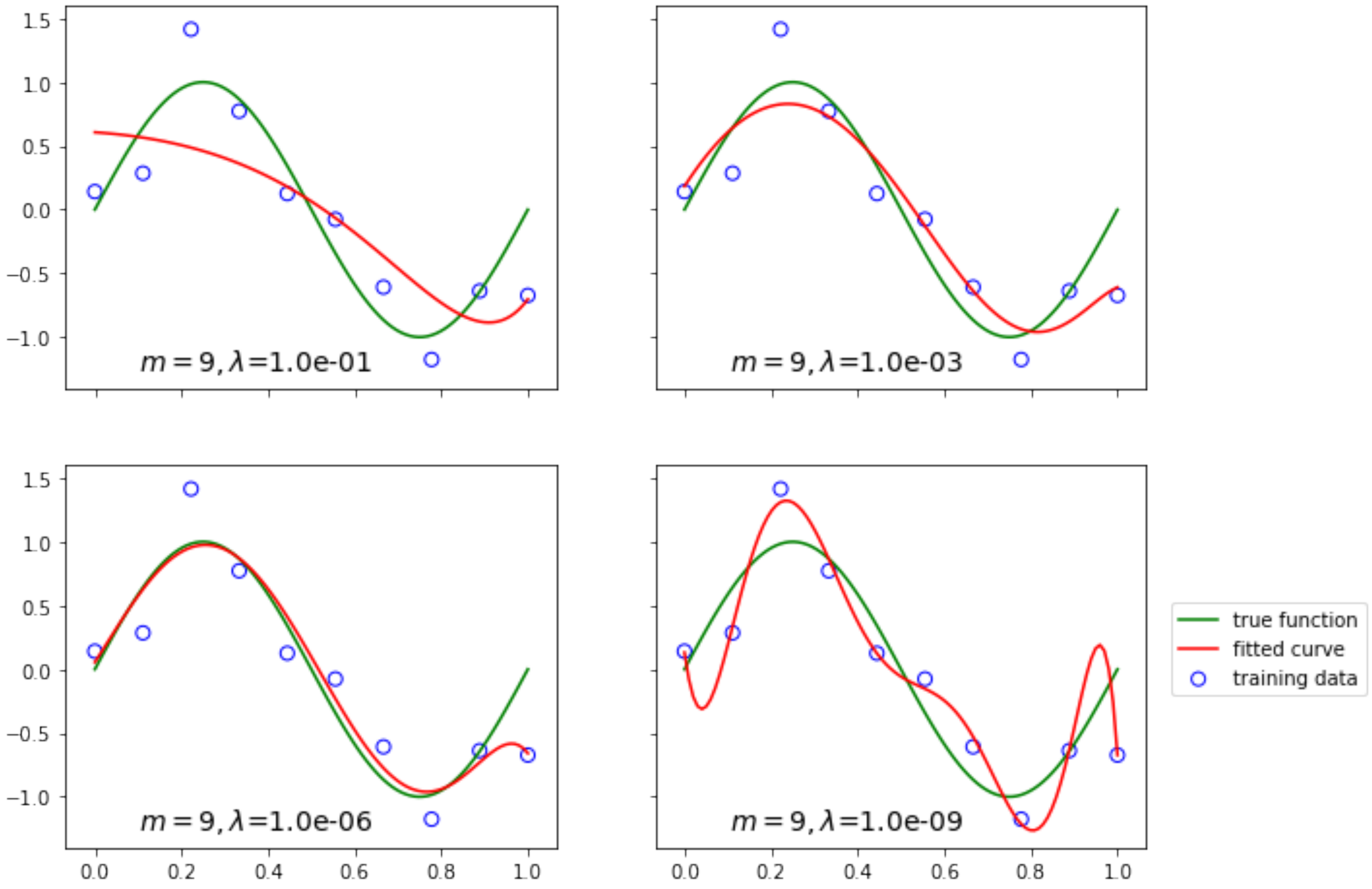
where $\alpha \in [0, 1]$ is a parameter dictating the proportion of L1 to L2 regularization.

Why all these different kinds of regularization? Well L1 tends to produce a *sparse* solution... many weights that are not important are driven towards 0. You use this technique when it seems appropriate to you that less-important factors have no influence on the solution. Whereas L2 limits the total amount of weight evenly, so less-important factors can continue to have less-influence-but-still-some-influence.

One application of L1 regularization: feature selection. Let's say you have data about the expression levels of ~27,000 protein coding genes across a few thousand humans, and you want to dermine if any of the genes have an effect on a disease. Since almost NONE of them will, it's good to use something like a heavy L1 penalty, which will prevent you from picking up too much on random chance associations that may exist in the data.



9th order polynomial regression + L2 regularization



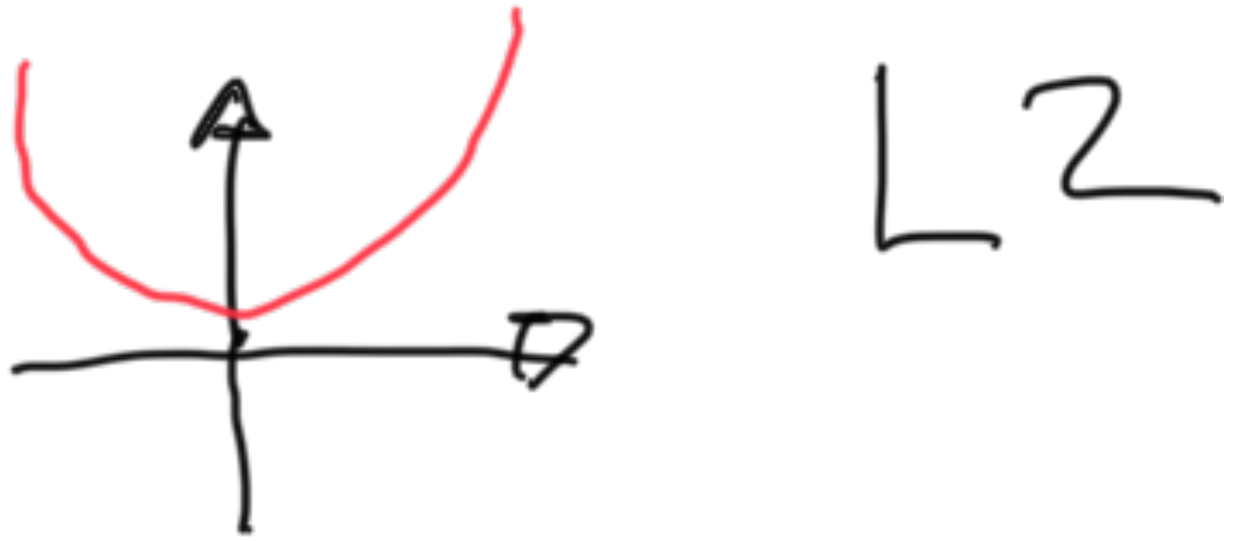
Robust estimation
to reject outliers

Estimation and optimization

You can have a different loss function than Residual Sum of Squares!

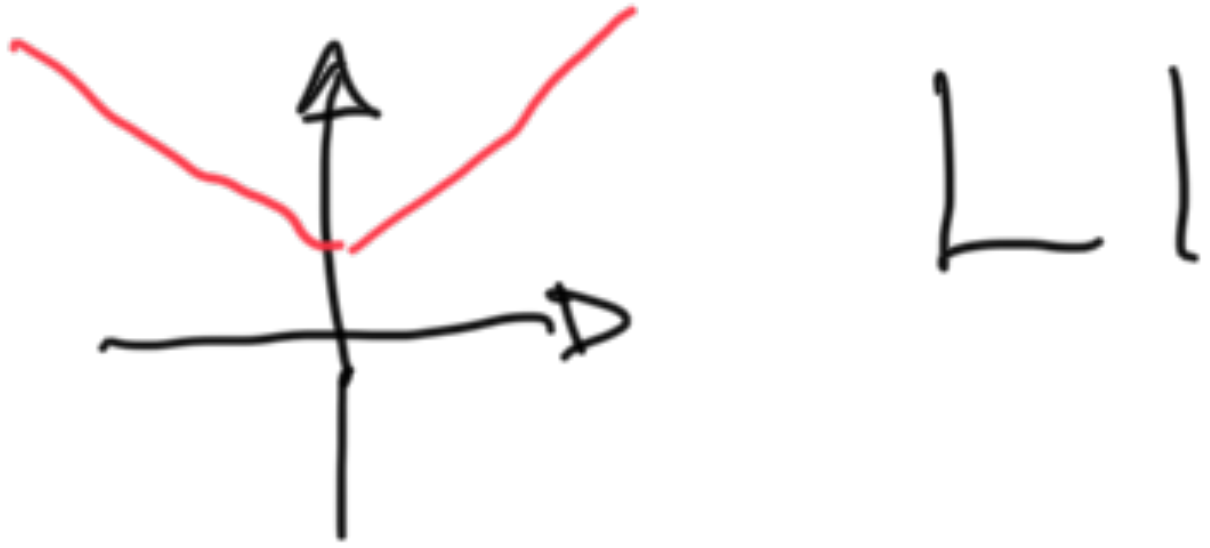
L2 norm: aka RSS!

$$e = \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i; \theta))^2$$



L1 norm:

$$e = \sum_{i=1}^{n} |y_i - f(\mathbf{x}_i; \theta)|$$



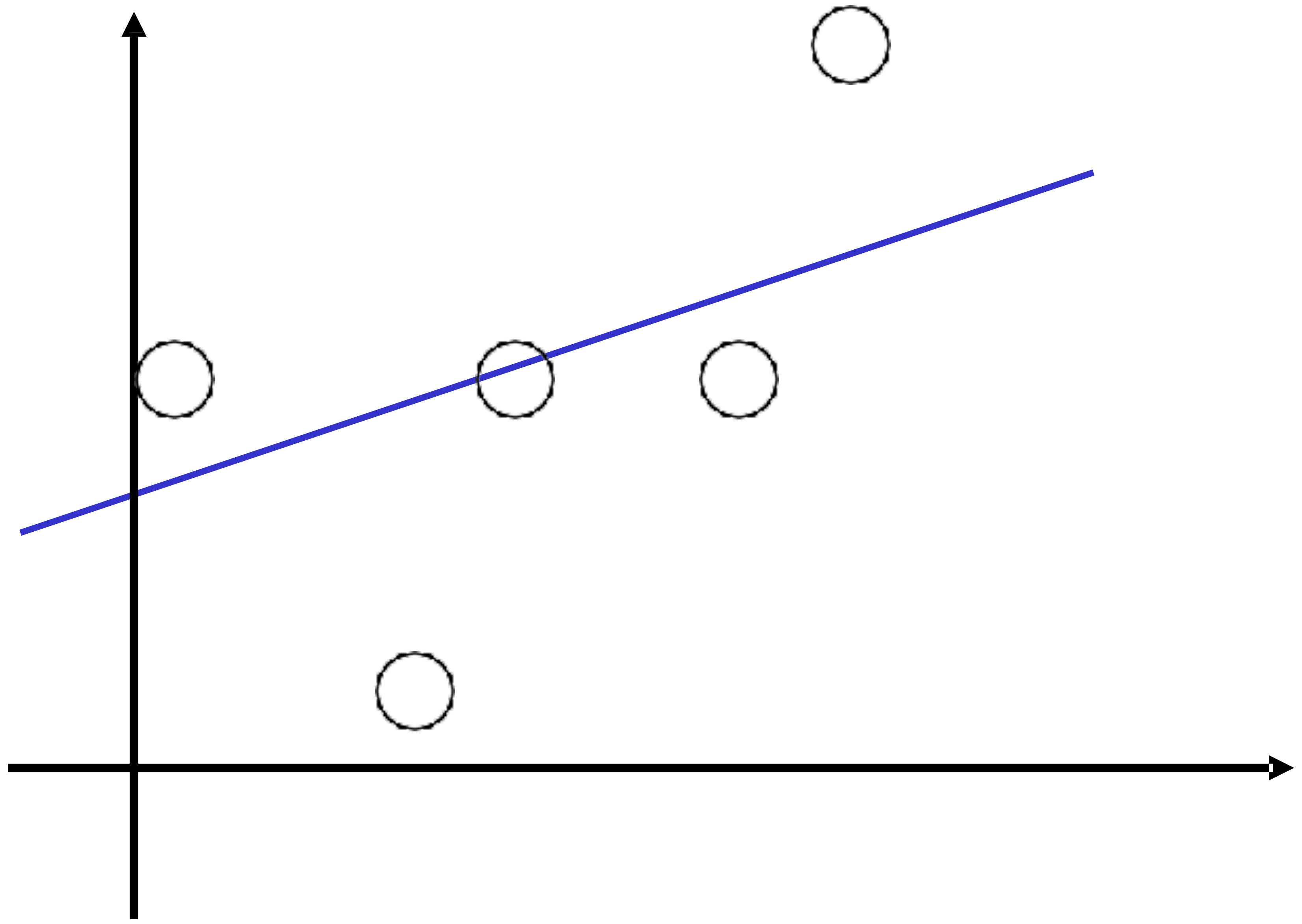
Estimation and optimization

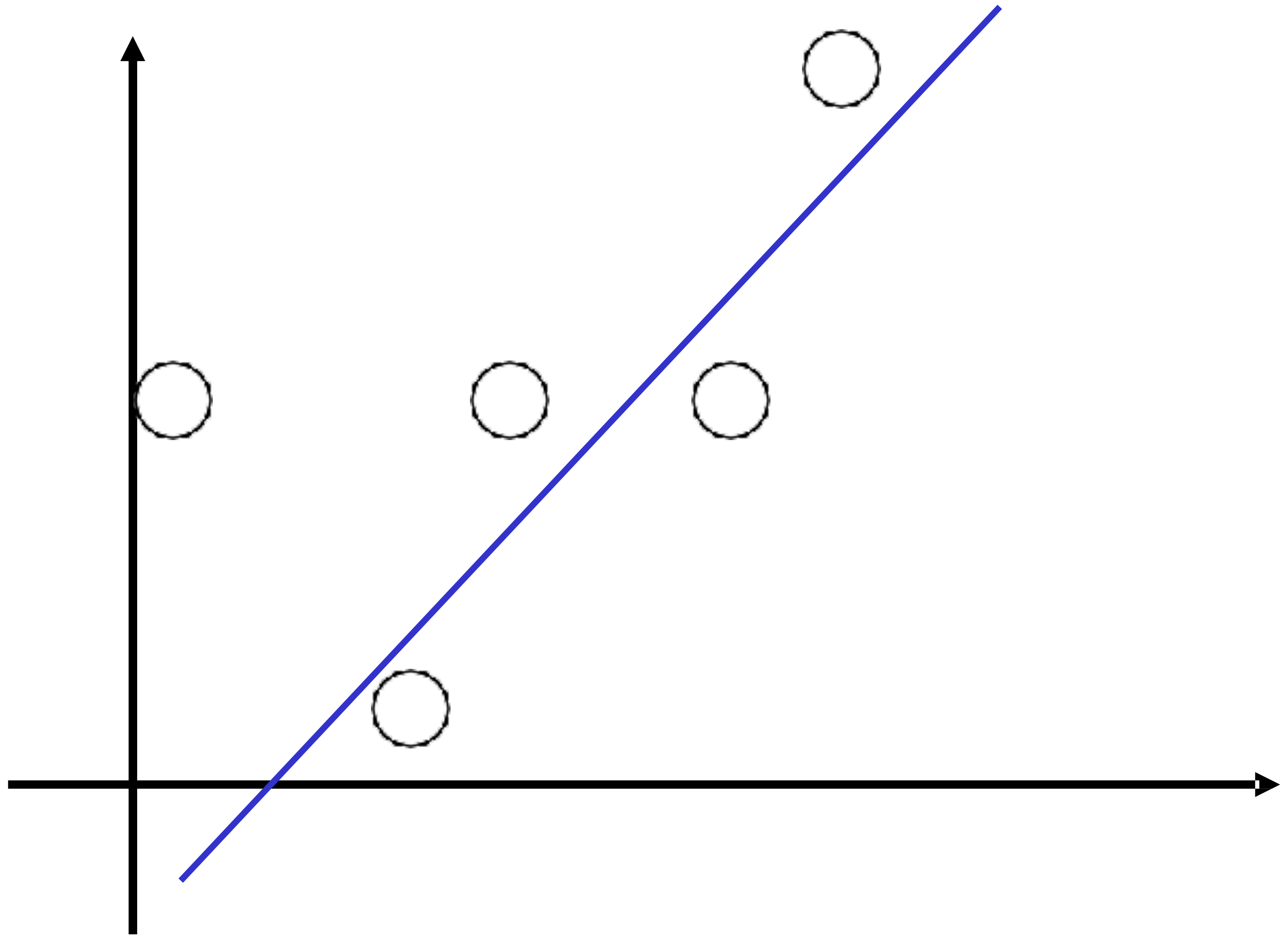
$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}; \mathbf{x}; \mathbf{y})$$

$$\begin{aligned}\mathcal{L}_{\text{OLS}}(\mathbf{w}; \mathbf{x}; \mathbf{y}) &= \left(\mathbf{y} - f(\mathbf{x}; \mathbf{w})\right)^{\text{T}} \left(\mathbf{y} - f(\mathbf{x}; \mathbf{w})\right) \\ &= \|\mathbf{y} - f(\mathbf{x}; \mathbf{w})\|_2^2\end{aligned}$$

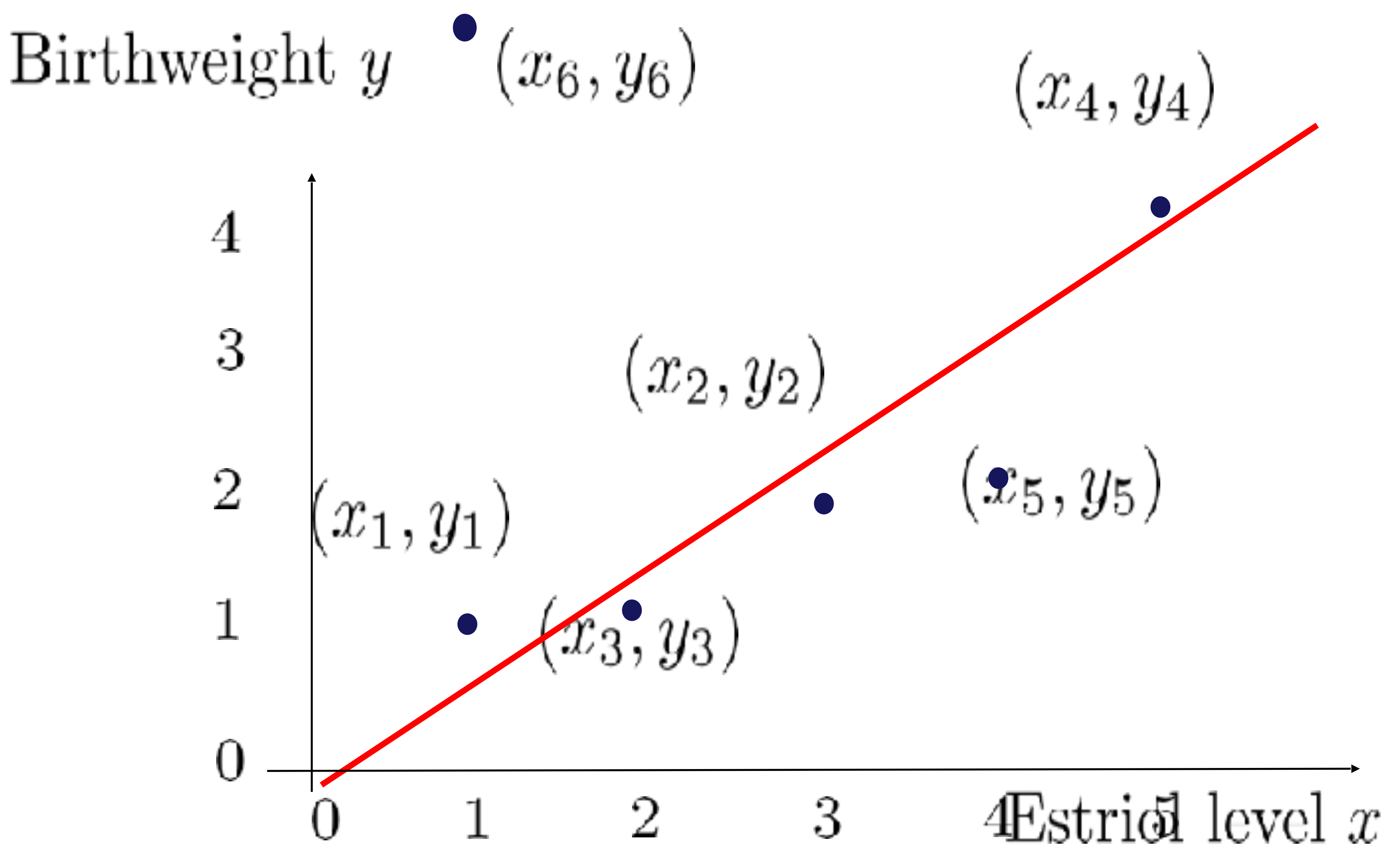
$$\begin{aligned}\mathcal{L}_{\text{robust}}(\mathbf{w}; \mathbf{x}; \mathbf{y}) &= \sum_{i=1}^{n} \left| y_i - f\left(\mathbf{x}_i; \mathbf{w}\right) \right| \\ &= \|\mathbf{y} - f(\mathbf{x}; \mathbf{w})\|_1\end{aligned}$$





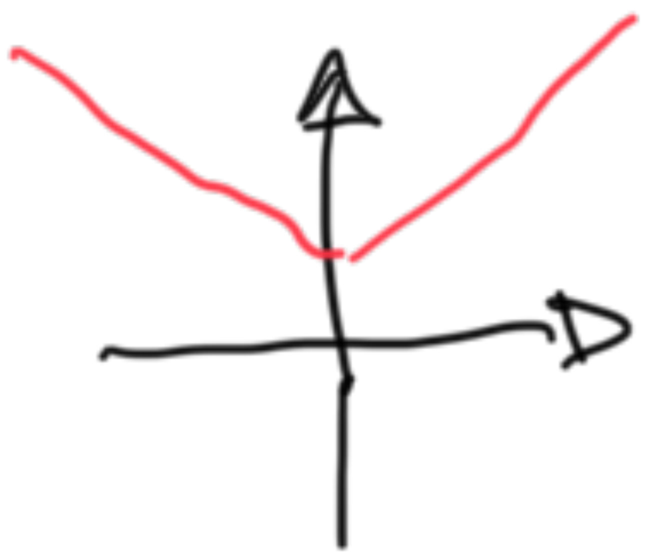
Robust estimation

$$\begin{array}{ccccc} Y & = & X & & W \\ \begin{pmatrix}1\\1.9\\1.05\\4.1\\2.1\\6.0\end{pmatrix} & & \begin{pmatrix}1,1\\1,3\\1,2\\1,5\\1,4\\1,1.1\end{pmatrix} & & \begin{pmatrix}w_0\\w_1\end{pmatrix}\end{array}$$



~~$W^* = (X^TX)^{-1}X^TY$~~

$$W^* = \arg min_W \sum_i (\mathbf{x}_i^T W - y_i)^2$$



L1

$$W^* = \arg min_W \sum_i |\mathbf{x}_i^T W - y_i|$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathscr{L}(\mathbf{w};\mathbf{x};\mathbf{y})$$

L1

$$S_{training} = \{(x_i, y_i), i = 1..n\}$$

E.g. $S_{training} = \{(x_i, y_i), i = 1..n\}$

$= \{(1,1),(3,1.9),(2,1.05),(5,4.1),(4,2.1)\}$

$$Y = \begin{pmatrix} 1 \\ 1.9 \\ 1.05 \\ 4.1 \\ 2.1 \end{pmatrix} \qquad X = \begin{pmatrix} 1,1 \\ 1,3 \\ 1,2 \\ 1,5 \\ 1,4 \end{pmatrix}$$

1. Loss (Cost) Function

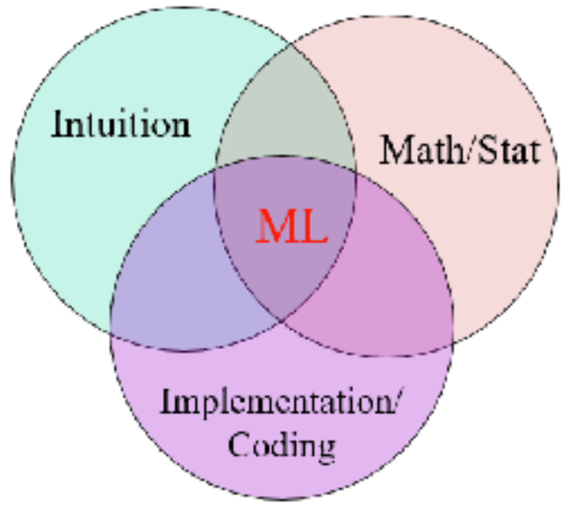
$$\mathscr{L}_{\text{robust}}(\mathbf{y};\mathbf{x};\mathbf{w}) = \sum_{i=1}^{n} \left| y_i - f\left(\mathbf{x}_i;\mathbf{w}\right) \right|$$
$$= \|\mathbf{y} - f(\mathbf{x};\mathbf{w})\|_1$$

2. Obtain the gradient

$$\frac{\partial \mathscr{L}(\mathbf{w})}{\partial \mathbf{w}} = \text{sign}\left(\mathbf{y} - f(\mathbf{x};\mathbf{w})\right)\mathbf{x}$$

3. Update parameter W

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \lambda_t \frac{\partial \mathscr{L}(\mathbf{w})}{\partial \mathbf{w}}$$



Recap: Robust estimation

Intuition: It's very easy to overfit to either random noise or outliers. Help your ML algorithm reject either form of perturbation by applying robust estimation methods like using an L1 loss function (helps reduce the influence of outliers) or by adding L1 or L2 regularization to any loss function (helps reduce overfitting to noise by penalizing complex and large parameter values)

Math:

L1 loss function
(robust regression)

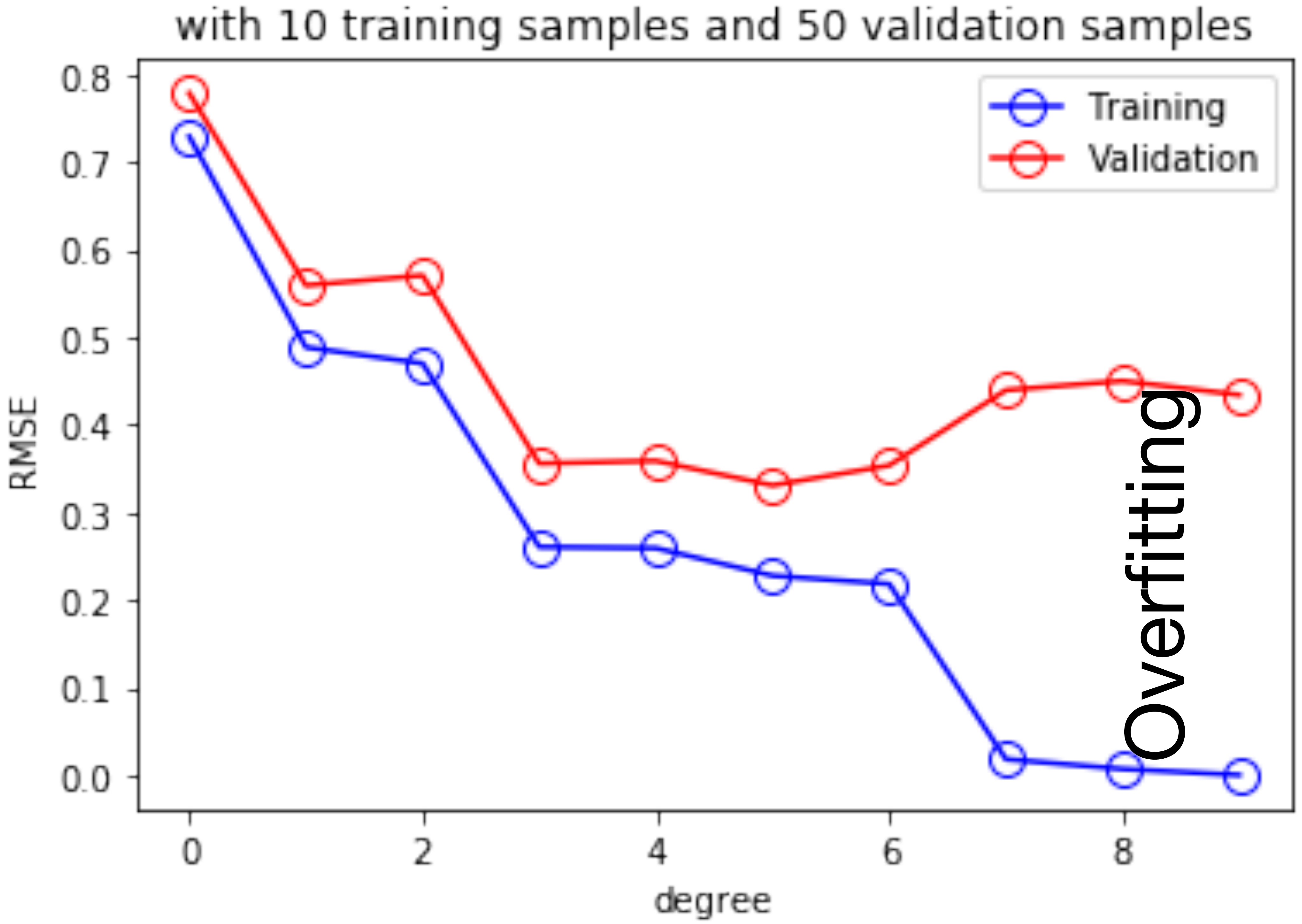
$$L(\mathbf{w}) = \frac{1}{2}\|(X\mathbf{w} - \mathbf{y})\|_1$$

L2 loss function + combo L1/L2 regularization
(ElasticNet)

$$L(\mathbf{w}) = \frac{1}{2}(X\mathbf{w} - \mathbf{y})^T(X\mathbf{w} - \mathbf{y}) + \frac{\lambda}{2}(\alpha\|\mathbf{w}\|_1 + (1-\alpha)\|\mathbf{w}\|_2)$$

Overfitting and underfitting

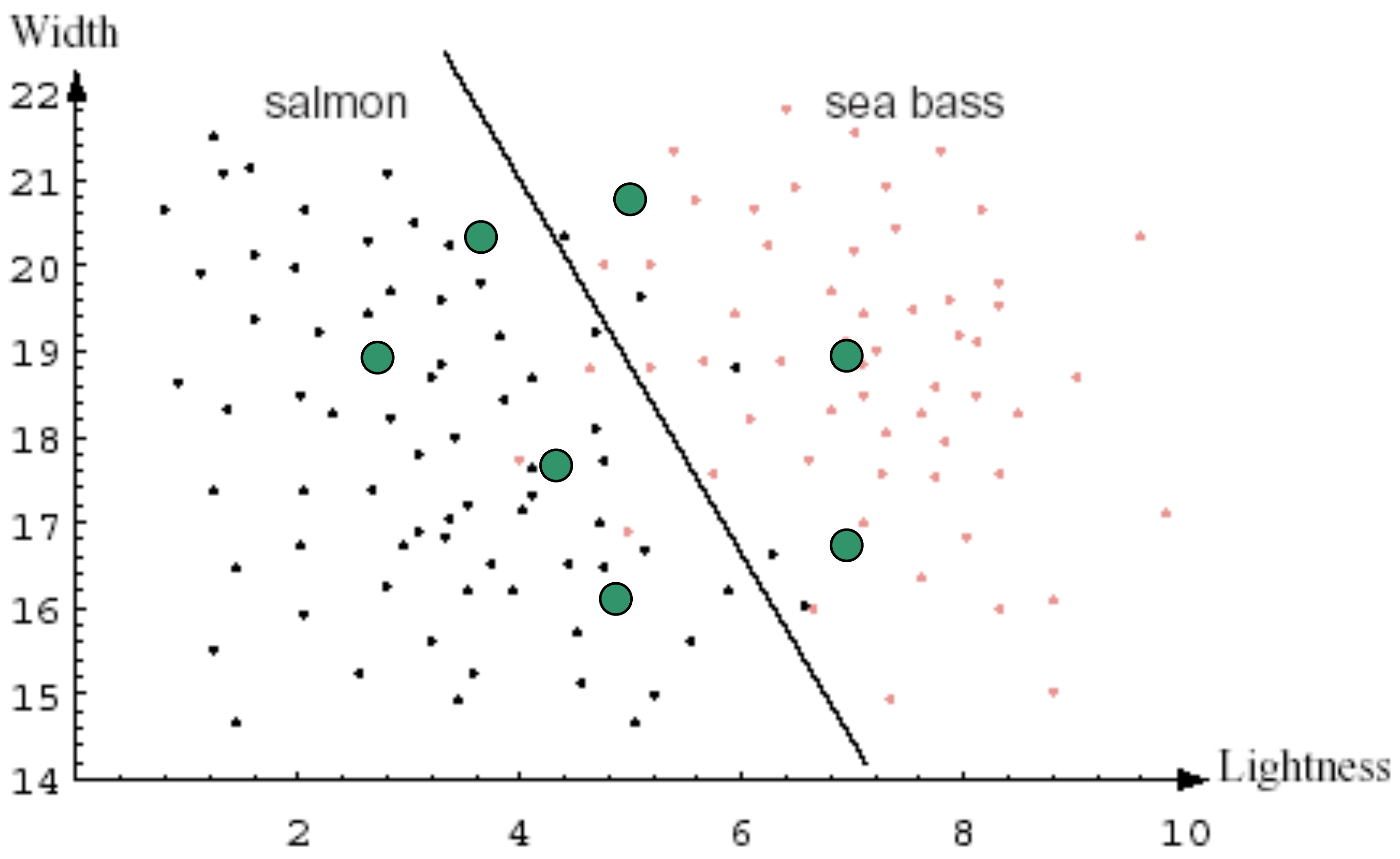
- Normally $e_{\text{train}} \leq e_{\text{test}}$
- $e_{\text{test}} = e_{\text{train}} + e_{\text{generalization}}$
- Overfit: when $e_{\text{train}} \ll e_{\text{test}}$
  - Generalization error gets big!
- Underfit:
  - $e_{\text{train}} \sim$ chance level,
  - e_{test} going down with additional model complexity



Testing error

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\} \qquad S_{testing} = \{(\mathbf{x}_i, y_i), i = 1..q\}$$

$$e_{testing} = e_{training} + generalization(f)$$



$$e_{testing} = \frac{1}{q}\sum_{i=1}^{q} \mathbf{1}(y_i \neq f(\mathbf{x}_i))$$

$$e_{testing} = 0.5?$$

1. $e_{testing} = 0.5 + 0.0$

2. $e_{testing} = 0.0 + 0.5$

Question: What are the two extreme cases leading to the same testing error=0.5 (Assume we have the same number of positives and negatives.)

Testing error

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\} \qquad S_{testing} = \{(\mathbf{x}_i, y_i), i = 1..q\}$$

$$\epsilon_{\text{testing}} = \epsilon_{\text{training}} + \epsilon_{\text{generalization}}$$

Case 1:

1. $e_{testing} = 0.5 + 0.0$

- We make a completely random guess even after training (didn't learn anything and attained an error of 0.5).
- A random guess is however highly generalizable, which doesn't incur any generalization error.

Case 2:

2. $e_{testing} = 0.0 + 0.5$

- We make perfect classifications on the training data (memorizing the labels for every training sample).
- Merely memorizing the training samples with their corresponding classification labels is however highly non-generalizable, incurring the largest generalization error (no identical training sample will appear in testing).

Both are extreme cases that lead to trivial ML models.

Testing error

$$\epsilon_{\text{testing}} = \epsilon_{\text{training}} + \epsilon_{\text{generalization}}$$

Case 1:

1. $e_{testing} = 0.5 + 0.0$

Case 2:

2. $e_{testing} = 0.0 + 0.5$

Both are extreme cases that lead to trivial models that we should avoid.

Ideally: $e_{testing} = 0.0 + 0.0$

A classification model that is perfect after training and makes no error in testing.

In practice: $e_{testing} = 0.05 + 0.1$

A classification model that does well after training and generalizes reasonably well in testing.