Linear Regression

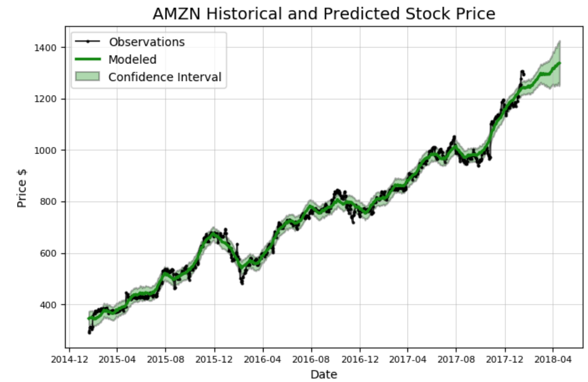
CSC 461: Machine Learning

Fall 2022

Prof. Marco Alvarez
University of Rhode Island

Continuous targets

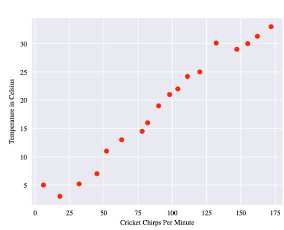
- Certain applications require the prediction of continuous values

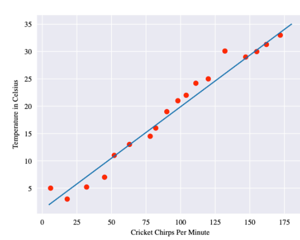


https://towardsdatascience.com/stock-prediction-in-python-b66555171a2

Linear model

- Assumes the output y is a linear function of the input $\boldsymbol{x}$
  - ✓ can use the function to make predictions, very simple approach, e.g. linear regression



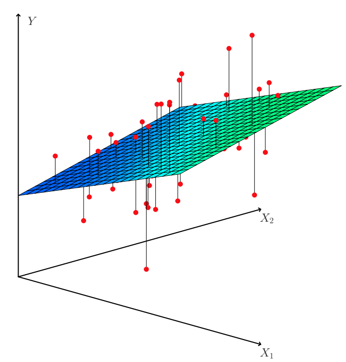


$$y = wx + b \quad w, x, b \in \mathbb{R}$$

Linear model

- What if we have d features?

why transpose?



$$y = \mathbf{w}^T\mathbf{x} + b \quad \mathbf{w}, \mathbf{x} \in \mathbb{R}^d \quad b \in \mathbb{R}$$

Figure from https://www.statlearning.com/

Linear model

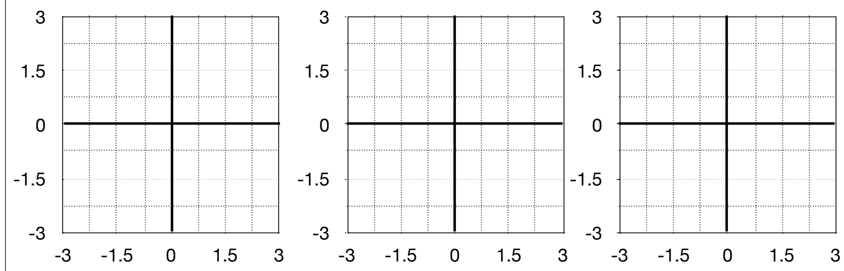
$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

hypothesis   weights   bias

The weights and bias are the model **parameters** which define the hypothesis and are used to make predictions

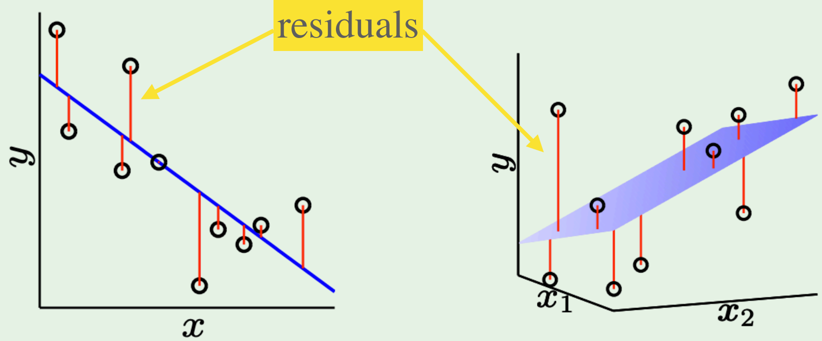
Draw the models

$$y = w_0 + w_1 x_1 \qquad w_0 = b$$



$\mathbf{w} = [0.5,0]$ $\mathbf{w} = [0,1.5]$ $\mathbf{w} = [0.5,0.5]$

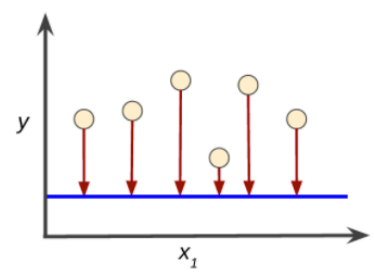
Residuals



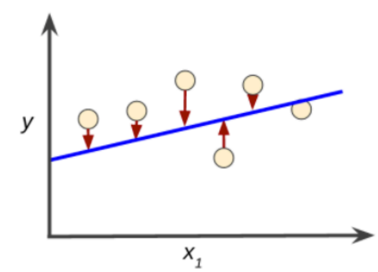
Want a linear function with small residuals

http://work.caltech.edu/slides/slides03.pdf

Residuals and Loss



High loss



Low loss

From https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss

Goal of learning

- Find a function that best approximates target function (minimize expected loss)

For $h \in \mathcal{H}$ and $\forall (\mathbf{x}^{(i)}, y^{(i)}) \sim P$, we want $h(\mathbf{x}^{(i)}) \approx f(\mathbf{x}^{(\mathbf{i})})$

- What is the expected loss?
  - ✓ cannot calculate, can **approximate** with empirical loss:

$$\mathbb{E}\left[l(h, \mathbf{x}^{(i)}, y^{(i)})\right]_{(\mathbf{x}^{(i)}, y^{(i)}) \sim P} \approx L(h, \mathcal{D})$$

Defining linear regression

- Data $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$

$\mathbf{x}^{(i)} \in \mathbb{R}^d, y^{(i)} \in \mathbb{R}$

- Loss Function: Squared Loss (MSE)

$$l_{sq}(h, \mathbf{x}, y) = \big(h(\mathbf{x}) - y\big)^2$$

$$L_{sq}(h, \mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} l_{sq}(h, \mathbf{x}^{(i)}, y^{(i)})$$

Alternative notation

X0	X1	X2	Y
1	0.5	0.1	0.25
1	0.3	0.9	0.5
1	0.3	0.875	1.15
1	0.45	0.15	2.13
...	...	...	...

$$h(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

$$= \sum_{i=1}^{d} w_i x_i + b$$

$x_0 = 1, \quad w_0 = -b$

$$= \sum_{i=0}^{d} w_i x_i$$

$$= \boxed{\mathbf{w}^T \mathbf{x}}$$

Alternative notation

$+1$

input: $(x_0, x_1, \ldots, x_d)$

model: $(w_0, w_1, \ldots, w_d)$

$$h(\mathbf{x}) = \sum_{i=0}^{d} w_i x_i = \mathbf{w}^T \mathbf{x}$$

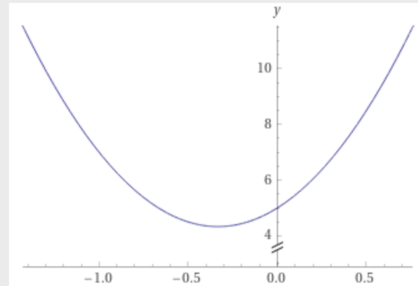
What is the hypothesis space?

all linear functions in $\mathbb{R}^{d+1}$

$\mathcal{H} = \{h_w : h_w(\mathbf{x}) = \mathbf{w}^T\mathbf{x},\ \mathbf{w} \in \mathbb{R}^{d+1}\}$

Closed-form solution

min vs. argmin



$$f(x) = 6x^2 + 4x + 5$$

$$\min f(x)\,?$$

$$\arg\min_{x} f(x)\,?$$

Solving linear regression (least squares)

find these parameters

$$\mathbf{w}^* = \arg\min_{h \in \mathscr{H}} \; \frac{1}{n}\sum_{i=1}^{n}\left(h(\mathbf{x}^{(i)}) - y^{(i)}\right)^2$$

given this objective function

unconstrained optimization

Solving linear regression (least squares)

$$\mathbf{w}^* = \arg\min_{h \in \mathscr{H}} \; \frac{1}{n}\sum_{i=1}^{n}\left(h(\mathbf{x}^{(i)}) - y^{(i)}\right)^2$$

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \; \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)}\right)^2$$

Understanding matrix form

$$\mathbf{y} = \mathbf{X}\mathbf{w}$$

input vector $\mathbf{x}^{(1)}$

$$\begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix} \approx \begin{bmatrix} \left(\mathbf{x}^{(1)}\right)^T \\ \vdots \\ \left(\mathbf{x}^{(n)}\right)^T \end{bmatrix} \cdot \begin{bmatrix} w_0 \\ \vdots \\ w_d \end{bmatrix}$$

Norms

- A **norm** is a function that assigns a strictly positive length to each vector in a vector space
  - ✓ except for the zero vector

ℓ_1-norm: $\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$

manhattan distance from origin

ℓ_2-norm: $\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} x_i^2\right)^{\frac{1}{2}}$

euclidean norm, euclidean distance from origin

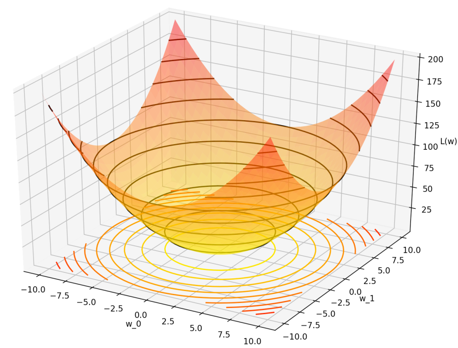
Using matrix notation

$$\arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \left(h(\mathbf{x}^{(i)}) - y^{(i)}\right)^2$$

$$\arg\min_{\mathbf{w}} \frac{1}{n} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

How to minimize it?

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{1}{n} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

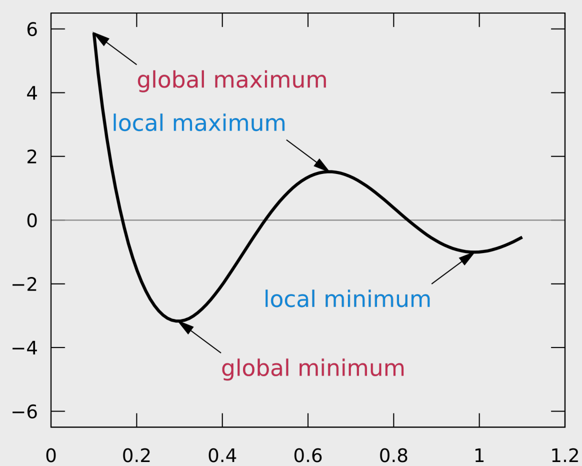


continuous, differentiable, **convex**

optimal solution

https://cnl.salk.edu/~schraudo/teach/NNcourse/linear1.html

(some) Critical points



https://en.wikipedia.org/wiki/Maxima_and_minima

Closed form solution

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\|\mathrm{X}\mathbf{w} - \mathbf{y}\|^2$$

$$\nabla E_{\text{in}}(\mathbf{w}) = \frac{2}{N}\mathrm{X}^{\mathsf{T}}(\mathrm{X}\mathbf{w} - \mathbf{y}) = \mathbf{0}$$

$$\mathrm{X}^{\mathsf{T}}\mathrm{X}\mathbf{w} = \mathrm{X}^{\mathsf{T}}\mathbf{y}$$

$$\mathbf{w} = \mathrm{X}^{\dagger}\mathbf{y} \quad \text{where} \quad \mathrm{X}^{\dagger} = (\mathrm{X}^{\mathsf{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathsf{T}}$$

$\mathrm{X}^{\dagger}$ is the 'pseudo-inverse' of X

There are other methods for finding the optimal solution
e.g. gradient descent, MLE

http://work.caltech.edu/slides/slides03.pdf

The algorithm

1: Construct the matrix X and the vector $\mathbf{y}$ from the data set $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$ as follows

$$\underbrace{\mathrm{X} = \begin{bmatrix} \text{—}\mathbf{x}_1^{\mathsf{T}}\text{—} \\ \text{—}\mathbf{x}_2^{\mathsf{T}}\text{—} \\ \vdots \\ \text{—}\mathbf{x}_N^{\mathsf{T}}\text{—} \end{bmatrix}}_{\text{input data matrix}}, \qquad \underbrace{\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\text{target vector}}.$$

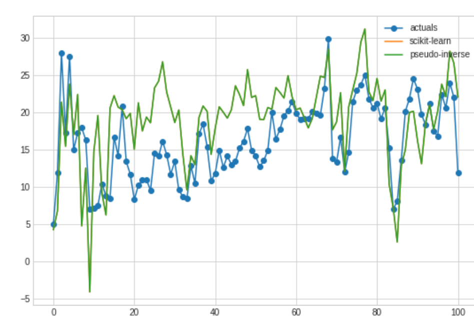
2: Compute the pseudo-inverse $\mathrm{X}^{\dagger} = (\mathrm{X}^{\mathsf{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathsf{T}}$.

3: Return $\mathbf{w} = \mathrm{X}^{\dagger}\mathbf{y}$.

http://work.caltech.edu/slides/slides03.pdf

Boston housing dataset

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33



https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html

Show me the code

```
w = np.linalg.pinv(Xtr).dot(Ytr)

pred = Xte.dot(w)

loss = np.mean((pred-Yte) ** 2)
```

vectorized computation

https://colab.research.google.com/drive/1GIovpb0ij4bSK1-jPKjNTlmXHSwTWwou

Computational complexity?

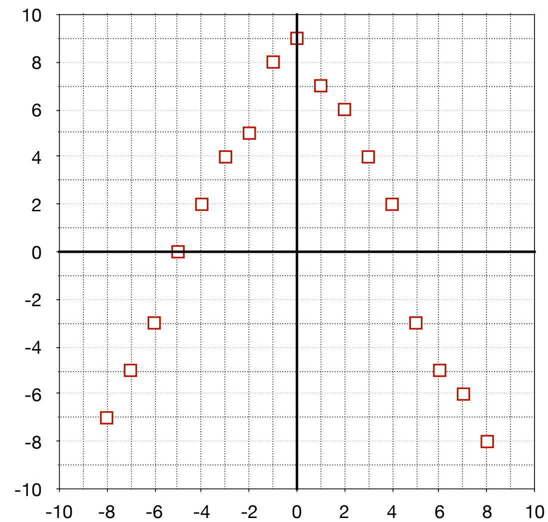
$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

$$O(nd^2 + d^3)$$

Training might be computationally expensive

Nonlinear features

Data is not always ‘linear’



Transforming the data

- Linear regression => **linear in the weights**
  - ✓ linear combination of the features
- Nonlinear functions
  - ✓ can transform the data nonlinearly using any feature transformations

$$\mathbf{x} = (x_0, \ldots x_d) \xrightarrow{\boldsymbol{\Phi}} \mathbf{z} = (x_0, \ldots z_{\tilde{d}})$$

input space $\mathcal{X} = \mathbb{R}^{d+1}$ feature space $\mathcal{Z} = \mathbb{R}^{\tilde{d}+1}$

Transforming the data

$$\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} \qquad \boldsymbol{\Phi}(\mathbf{x}) = \mathbf{z} = \begin{bmatrix} 1 \\ \Phi_1(\mathbf{x}) \\ \vdots \\ \Phi_{\tilde{d}}(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 1 \\ z_1 \\ \vdots \\ z_{\tilde{d}} \end{bmatrix}$$

$$h(\mathbf{x}) = \tilde{\mathbf{w}}^T \boldsymbol{\Phi}(\mathbf{x})$$

Polynomial models on one feature

- A **k-th** order polynomial model in one variable is defined as:

$$h(\mathbf{x}) = w_0 + w_1 x^1 + w_2 x^2 + \ldots + w_k x^k$$

$$\mathbf{x} = \begin{bmatrix} 1 \\ x \end{bmatrix} \qquad \boldsymbol{\Phi}(\mathbf{x}) = \mathbf{z} = \begin{bmatrix} 1 \\ \Phi_1(\mathbf{x}) \\ \vdots \\ \Phi_k(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 1 \\ x^1 \\ \vdots \\ x^k \end{bmatrix}$$

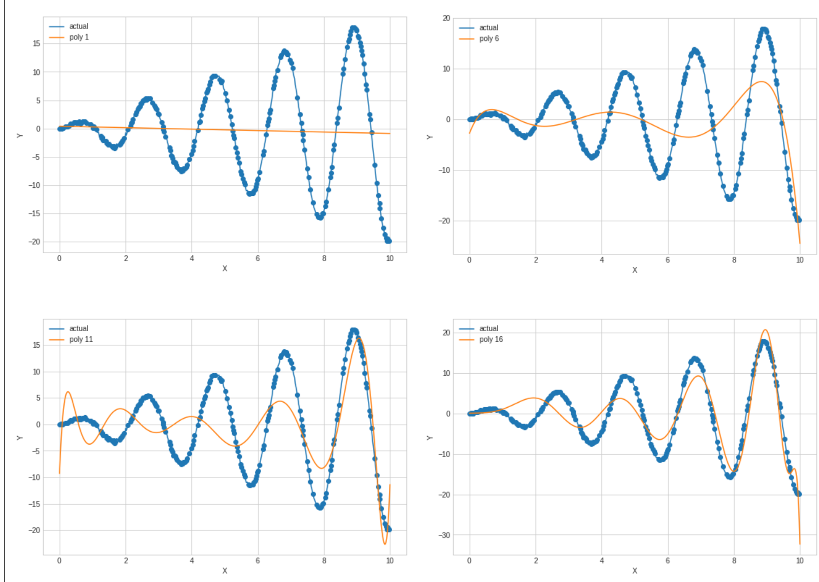
Polynomial models on two features

$$\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \qquad \boldsymbol{\Phi}(\mathbf{x}) = \mathbf{z} = \begin{bmatrix} 1 \\ \Phi_1(\mathbf{x}) \\ \Phi_2(\mathbf{x}) \\ \Phi_3(\mathbf{x}) \\ \Phi_4(\mathbf{x}) \\ \Phi_5(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \end{bmatrix}$$

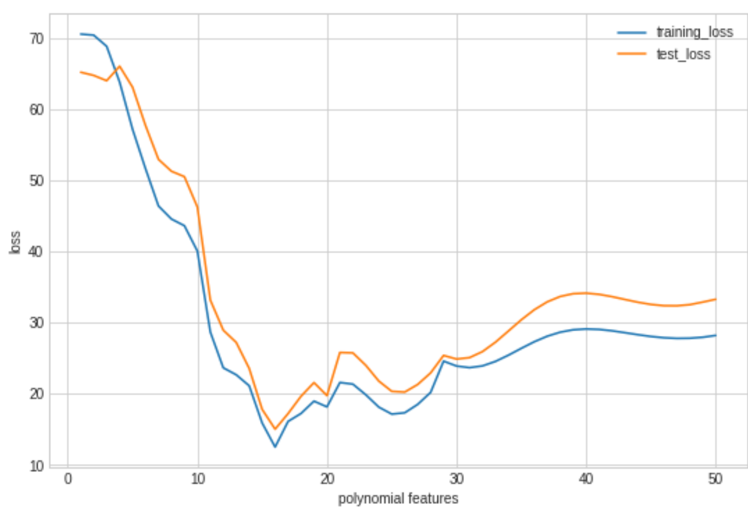
Show me the code

```python
# this function also adds the column of +1s
poly = PolynomialFeatures(p)

# transform data
_xtr = poly.fit_transform(Xtr)
_xte = poly.fit_transform(Xte)

# linear regression
w = np.linalg.pinv(_xtr).dot(Ytr)

# record losses
train_loss = np.mean((_xtr.dot(w)-Ytr)**2)
test_loss = np.mean((_xte.dot(w)-Yte)**2)
```

https://colab.research.google.com/drive/1W9kR_cbjYw0Ek2rsTO7_ojbfzxVN3pSJ#scrollTo=Wlm7SPzqhWnP



Trying a few transformations

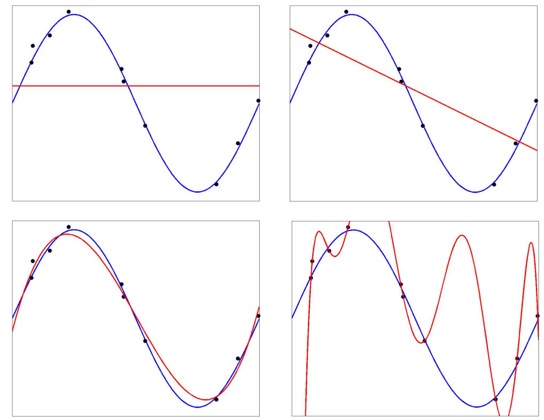


Polynomial models on more features

- `PolynomialFeatures` from *scikit-learn*
  - ✓ "all polynomial combinations of the features with degree less than or equal to the specified degree"
- Transformation function can be anything
  - ✓ choose transformation **before** looking into the data
  - ✓ use **cross-validation**
  - ✓ be aware of **computational cost**
  - ✓ be aware of **overfitting**

Overfitting and Regularization

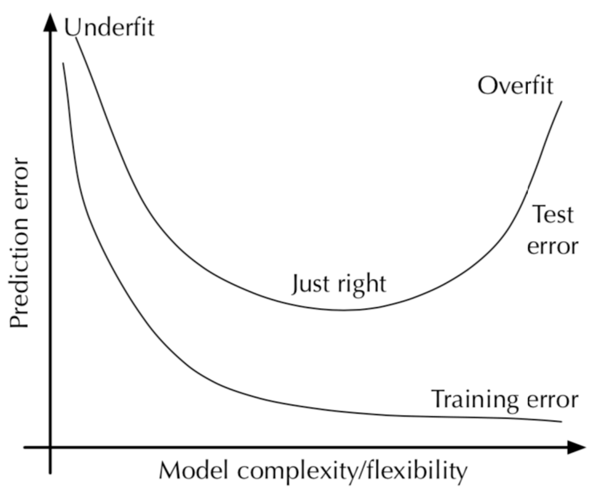
Lets talk about overfitting



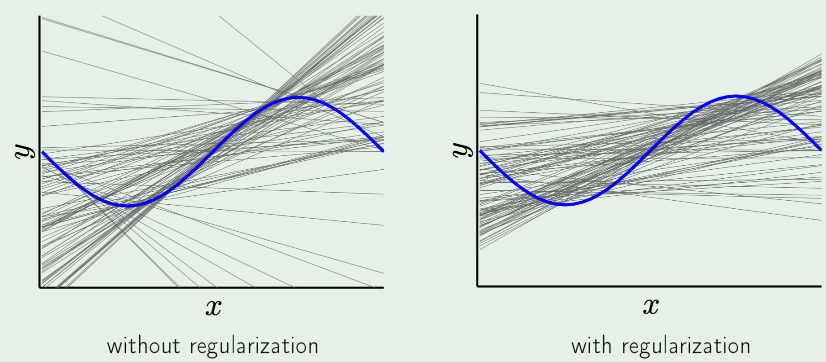
which one do you prefer?

Underfitting: model is too simple
Overfitting: model is too complex

Model complexity



What if we restrict the hypothesis space?



Regularization

- Adding a **penalty** to the weights to control the complexity of the model
  - ✓ usually penalizing higher weights (**except intercept**)
  - ✓ results in **simpler** or **more sparse** solutions
- Impact of regularization can be controlled by a parameter (*lambda*)

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{1}{n}\|\mathbf{X}\mathbf{w}-\mathbf{y}\|_2^2 + \lambda R(\mathbf{w})$$

L2 regularization

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{1}{n}\|\mathbf{X}\mathbf{w}-\mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_2^2$$

a.k.a. Ridge Regression

Can solve using matrix calculus again:

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

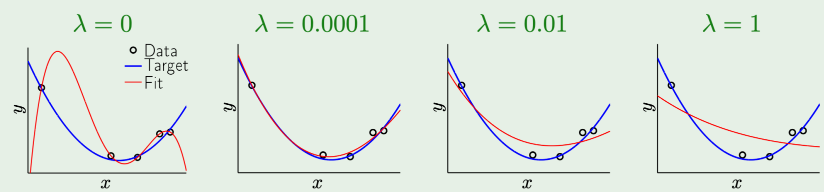
always invertible

L2 regularization

- If using the closed form solution for regularization the top-left corner of the identity matrix can be set to 0 (to handle intercept)

$$\begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

How does it work?



overfitting ⟶ ⟶ ⟶ ⟶ underfitting

http://work.caltech.edu/slides/slides12.pdf

L1 regularization

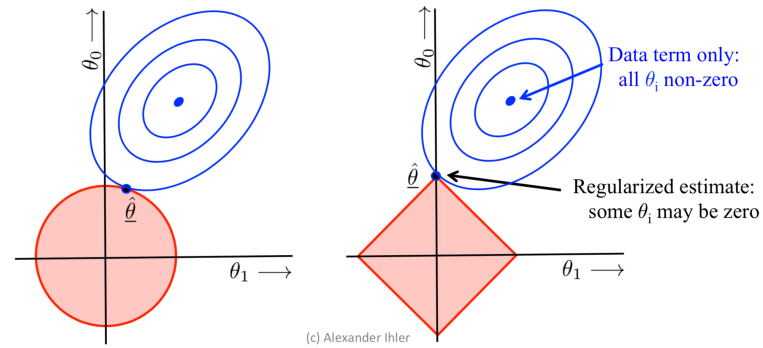
$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \quad \frac{1}{n}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_1$$

a.k.a. Lasso Regression

- Lasso does not have a **closed form solution**
  - can solve with quadratic programming or variants of gradient descent (subgradient methods)
- The regularization term is not differentiable

Comparison

- L1 regularization tends to generate **sparser** solutions



Final remarks

- Linear regression
  - ✓ solved by defining a hypothesis space and a loss function
  - ✓ essentially an optimization problem that can be solved directly (closed-form) or using other techniques, such as, gradient descent
- Important concepts
  - ✓ nonlinear features
  - ✓ overfitting/underfitting
  - ✓ regularization