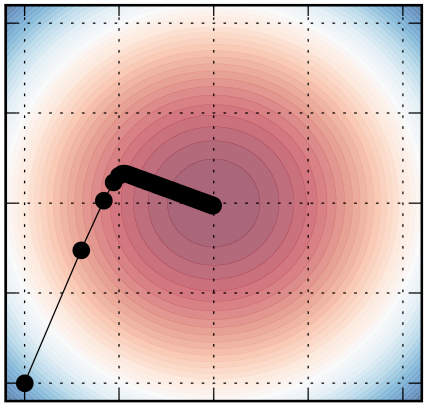
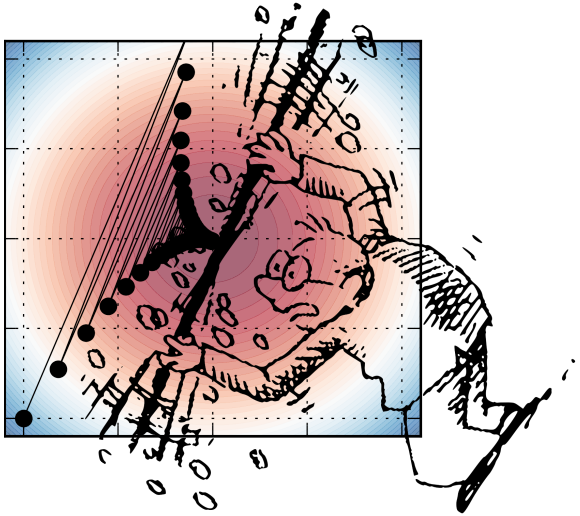
Doctoral School 2023

Machine Learning in Geomechanics

ALERT Geomaterials

Introduction to regression methods





Filippo Masi

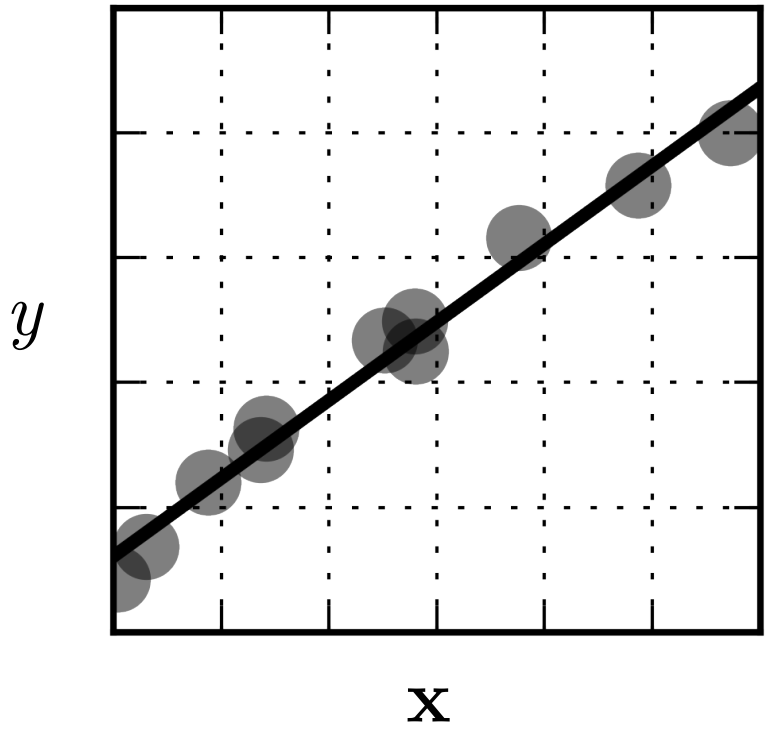
The University of Sydney

THE UNIVERSITY OF SYDNEY

Regression in a nutshell

Statistical technique to explore and quantify the relationship between **independent variables** $\mathbf{x}$ and **dependent variables** y.

$$y = f_{\boldsymbol{\theta}}(\mathbf{x}) + \epsilon$$

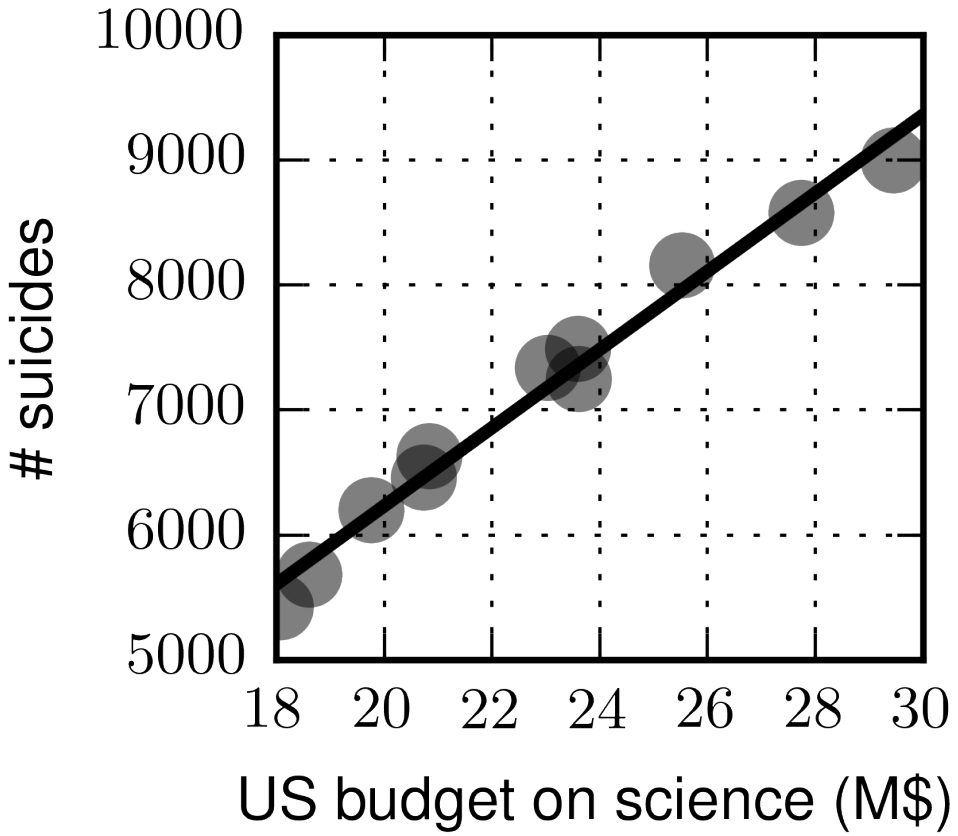


1) Identify a model that can be used for predictions and forecasting
2) Infer causal relationships between variables, under certain circumstances.

Regression in a nutshell

Statistical technique to explore and quantify the relationship between **independent variables** $\mathbf{x}$ and **dependent variables** y.

$$y = f_{\boldsymbol{\theta}}(\mathbf{x}) + \epsilon$$



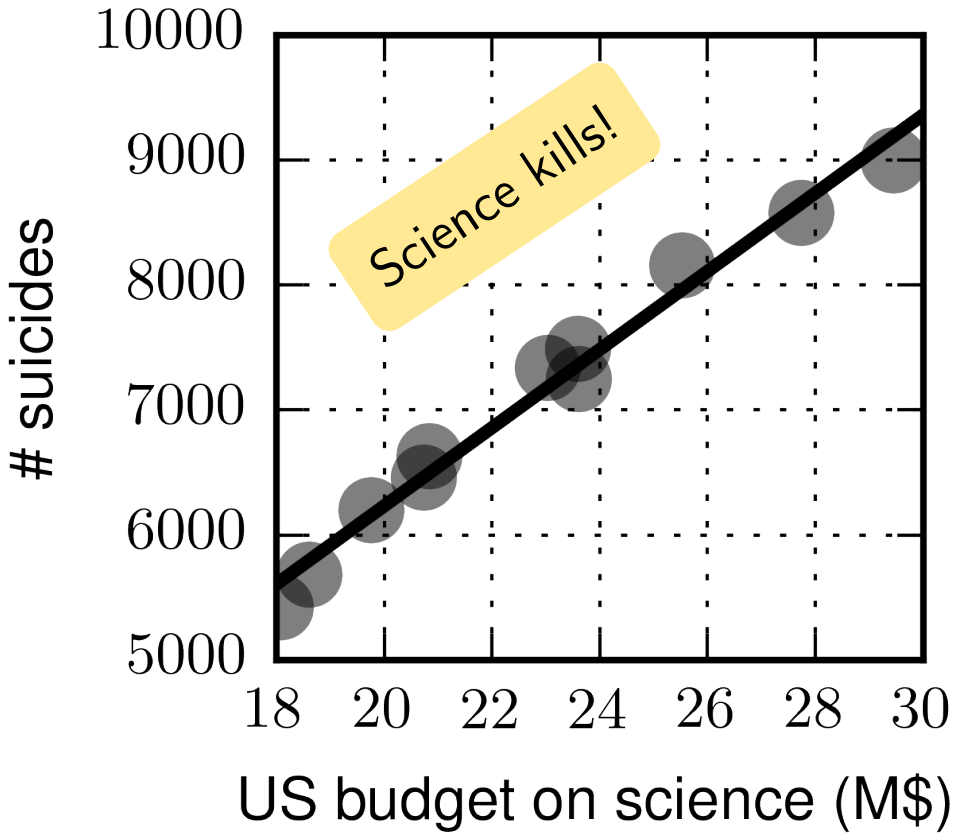
Office of Management and Budget, Centers for Disease Control & Prevention (1999-2009)

1) Identify a model that can be used for predictions and forecasting
2) Infer causal relationships between variables, under certain circumstances.

Regression in a nutshell

Statistical technique to explore and quantify the relationship between **independent variables** $\mathbf{x}$ and **dependent variables** y.

$$y = f_{\boldsymbol{\theta}}(\mathbf{x}) + \epsilon$$



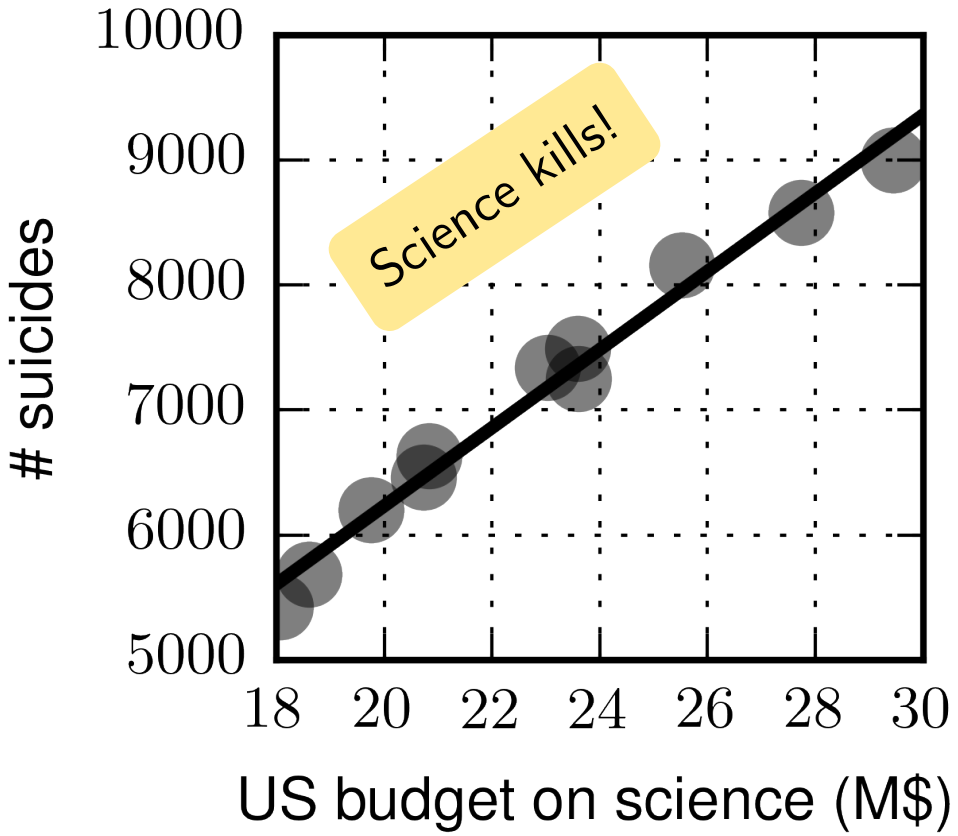
Office of Management and Budget, Centers for Disease Control & Prevention (1999-2009)

1) Identify a model that can be used for predictions and forecasting
2) Infer causal relationships between variables, under certain circumstances.

Regression in a nutshell

Statistical technique to explore and quantify the relationship between **independent variables** $\mathbf{x}$ and **dependent variables** y.

$$y = f_{\boldsymbol{\theta}}(\mathbf{x}) + \epsilon$$



Office of Management and Budget, Centers for Disease Control & Prevention (1999-2009)

Correlation does not imply causation

Linear regression

Linear regression

Consider a training data set $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$

m independent variables

$$\mathbf{X} \equiv \begin{bmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \\ \cdots \\ \mathbf{x}^{(n)} \end{bmatrix}$$

1 dependent variable

$$\mathbf{y} \equiv \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \cdots \\ y^{(n)} \end{bmatrix}$$

n snapshots

$$y^{(k)} \in \mathbb{R},$$
$$\mathbf{x}^{(k)} \in \mathbb{R}^m,$$
$$k \in [1, n]$$

Linear hypothesis function

$$\hat{y} = f_{\boldsymbol{\theta}}(\mathbf{x}) \equiv \boldsymbol{\theta}^{\mathrm{T}} \mathbf{x}^* = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_m x_m$$

parameters: *weights + bias* $\quad \boldsymbol{\theta} = \left[\theta_0, \theta_1, \theta_2, \cdots, \theta_m\right]^{\mathrm{T}}$

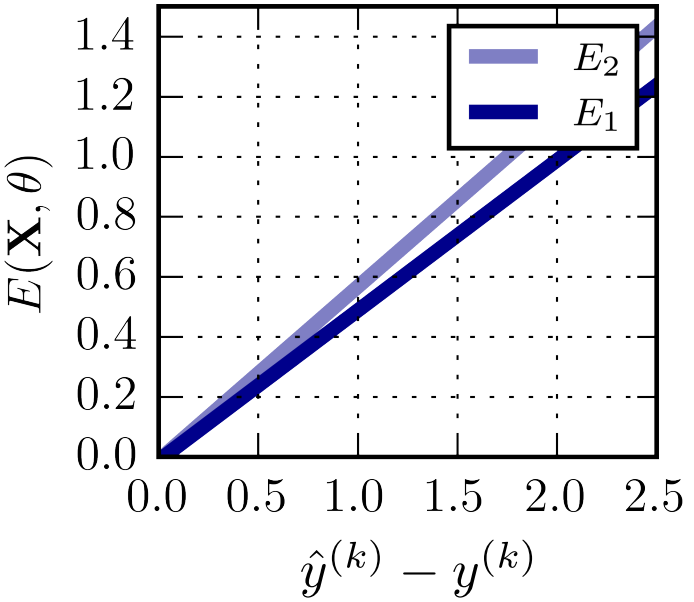
augmented features vector $\quad \mathbf{x}^* = \left[1, x_1, x_2, \cdots, x_m\right]^{\mathrm{T}}$

Linear regression

Error metric

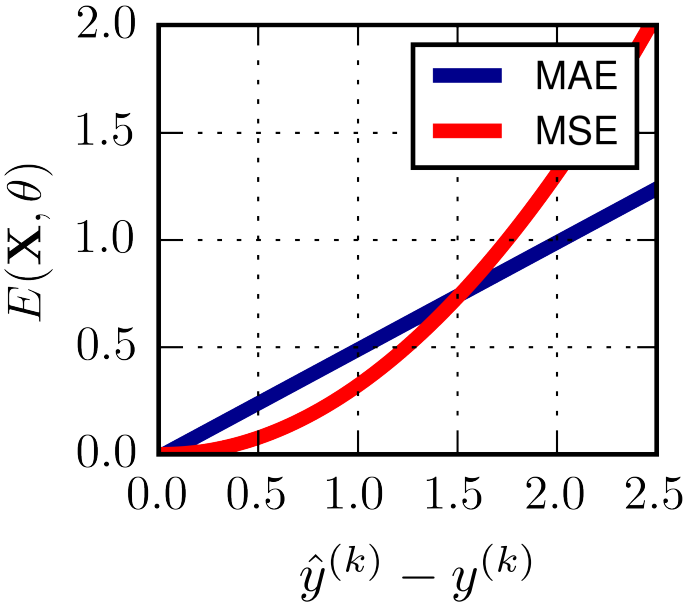
- Mean error based on the ℓ_r-norm

$$E_r\left(\mathbf{X}, f_{\boldsymbol{\theta}}\right) \equiv \left(\frac{1}{n}\sum_{k=1}^{n}\left|f_{\boldsymbol{\theta}}(\mathbf{x}^{(k)}) - y^{(k)}\right|^r\right)^{1/r}$$



- Mean squared error

$$\mathrm{MSE}\left(\mathbf{X}, f_{\boldsymbol{\theta}}\right) = \frac{1}{n}\sum_{k=1}^{n}\left(\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}^{(k)} - y^{(k)}\right)^2$$



The best fit model intrinsically depends on r

Linear regression

Training

find $\hat{\boldsymbol{\theta}}$ s.t. $\dfrac{\partial \mathrm{MSE}}{\partial \hat{\boldsymbol{\theta}}}\left(\mathbf{X}, f_{\hat{\boldsymbol{\theta}}}\right) = \dfrac{\partial}{\partial \hat{\boldsymbol{\theta}}}\left(\dfrac{1}{n}\left(\boldsymbol{\theta}^{\mathrm{T}}\mathbf{X} - \mathbf{y}\right)^{\mathrm{T}}\left(\boldsymbol{\theta}^{\mathrm{T}}\mathbf{X} - \mathbf{y}\right)\right)\Bigg|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} = 0$

The (unique) solution is given by the *normal equation*

$$\frac{2}{n}\mathbf{X}^{\mathrm{T}}\left(\mathbf{X}\hat{\boldsymbol{\theta}}^{\mathrm{T}} - \mathbf{y}\right) = 0$$

$$\hat{\boldsymbol{\theta}}^{\mathrm{T}} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

$\backsim \mathcal{O}\left(m^{2.5 \div 3}\right)$

Linear regression

Training

find $\hat{\boldsymbol{\theta}}$ s.t. $\dfrac{\partial \mathrm{MSE}}{\partial \hat{\boldsymbol{\theta}}}\left(\mathbf{X}, f_{\hat{\boldsymbol{\theta}}}\right) = \dfrac{\partial}{\partial \hat{\boldsymbol{\theta}}}\left(\dfrac{1}{n}\left(\boldsymbol{\theta}^{\mathrm{T}}\mathbf{X} - \mathbf{y}\right)^{\mathrm{T}}\left(\boldsymbol{\theta}^{\mathrm{T}}\mathbf{X} - \mathbf{y}\right)\right)\Bigg|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} = 0$

The (unique) solution is given by the *normal equation*

$$\frac{2}{n}\mathbf{X}^{\mathrm{T}}\left(\mathbf{X}\hat{\boldsymbol{\theta}}^{\mathrm{T}} - \mathbf{y}\right) = 0$$

$$\hat{\boldsymbol{\theta}}^{\mathrm{T}} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

$\backsim \mathcal{O}\left(m^{2.5 \div 3}\right)$

Alternatively $\hat{\boldsymbol{\theta}}^{\mathrm{T}}\mathbf{X} = \mathbf{y} \iff \hat{\boldsymbol{\theta}} = \mathbf{X}^{+}\mathbf{y}$

$\backsim \mathcal{O}\left(m^{2}\right)$

$$\underset{(n \times m)}{\mathbf{X}} = \underset{(n \times k)}{\mathbf{U}}\ \underset{(k \times k)}{\boldsymbol{\Sigma}}\ \underset{(k \times m)}{\mathbf{V}^{\mathrm{T}}}$$

Singular value decomposition $\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}} \Rightarrow \mathbf{X}^{+} = \mathbf{V}\boldsymbol{\Sigma}^{+}\mathbf{U}^{\mathrm{T}}$

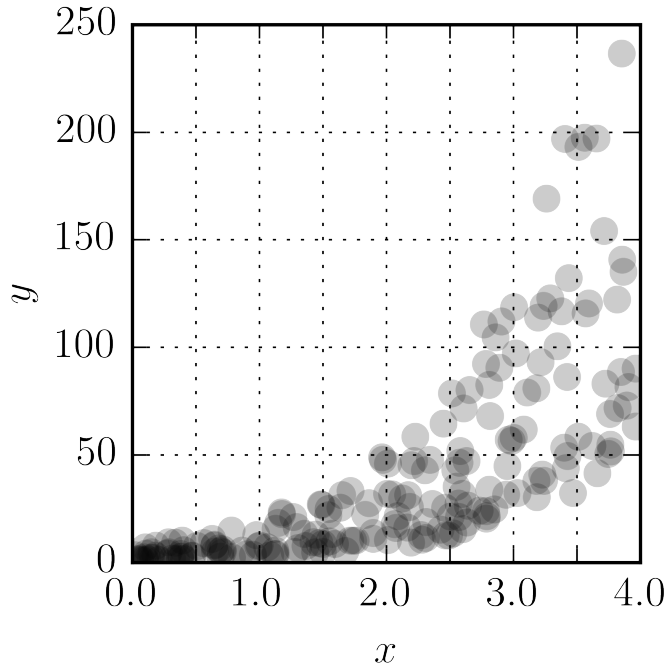
Example

https://qrco.de/regr

Example

Find the linear model that best fits the data set

$$\mathbf{y} = \exp\left(1 + \mathbf{x} + \mathcal{U}\left(-1, 1\right)\right)$$




```
import numpy as np
np.random.seed(42)
n_snapshots = 200 # set number of snapshots: n
a_1 = 1.; a_2 = np.exp(1) # set coefficients
noise = np.random.uniform(-1,1,(n_snapshots,1)) # uniform noise
X = np.random.uniform(0,4.,(n_snapshots,1)) # independent variable X
y = a_2*np.exp(a_1*X+noise) # dependent variable y
```

Example

Normal equation

$$\hat{\boldsymbol{\theta}}^{\mathrm{T}} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

gives

$$\hat{y} \approx -16.9 + 28x$$

```
X_p = np.c_[np.ones((n_snapshots, 1)), X] # add x0 = 1 to each instance
theta_best = np.linalg.inv(X_p.T.dot(X_p)).dot(X_p.T).dot(y)
print(theta_best) # parameters
```

```
array([[-16.93873601],[28.05702557]])
```

Example

Normal equation

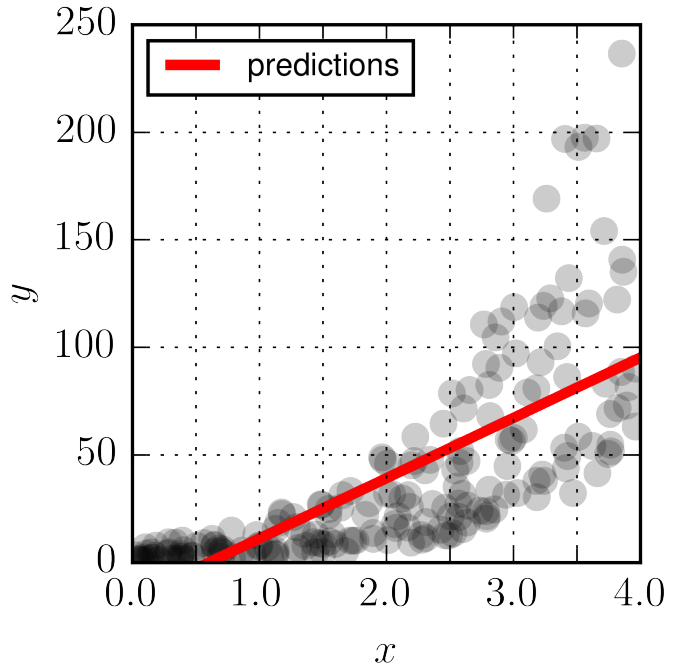
$$\hat{\boldsymbol{\theta}}^{\mathrm{T}} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

gives

$$\hat{y} \approx -16.9 + 28x$$

```
X_p = np.c_[np.ones((n_snapshots, 1)), X] # add x0 = 1 to each instance
theta_best = np.linalg.inv(X_p.T.dot(X_p)).dot(X_p.T).dot(y)
print(theta_best) # parameters
```

```
array([[-16.93873601],[28.05702557]])
```



... total failure!

```
X_new = np.array([[0],[4]])
X_b = np.c_[np.ones((2,1)), X_new] # add x0 = 1 to each instance
y_predict = X_b.dot(theta_best)
```

Example

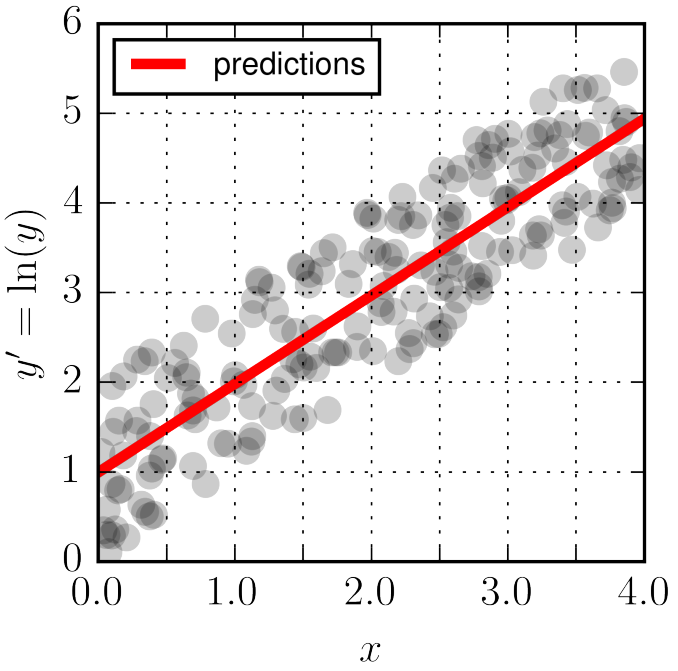
Data set transformation $y' = \ln y \rightarrow \mathbf{y}' = 1 + \mathbf{x} + \mathcal{U}(-1, 1)$

And magically:

```
y_prime = np.log(y) # change of coordinate, y' = ln(y)
X_b = np.c_[np.ones((n_snapshots, 1)), X] # add x0 = 1 to each instance
theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y_prime)
X_new = np.array([[0],[4]])
X_b = np.c_[np.ones((2,1)), X_new] # add x0 = 1 to each instance
y_predict = X_b.dot(theta_best) # parameters
print(theta_best)
```

```
array([[0.99530983],[0.98646971]])
```

$$\hat{y} \approx 0.98 + 0.99x$$



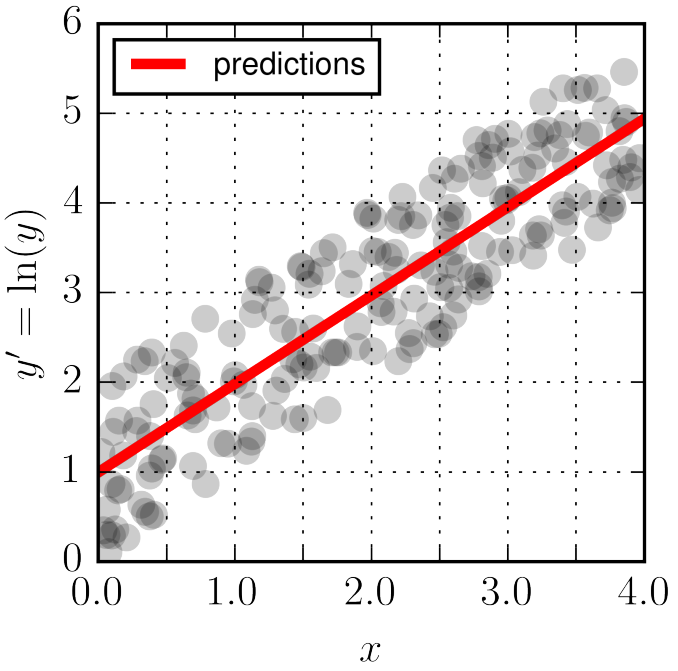
Example

Alternative way (via pseudoinverse):

```
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
regr = linear_model.LinearRegression() # create linear regression object
regr.fit(X, y_prime) # train the model
y_predict = regr.predict(X) # make predictions
```

```
Parameters: [0.99530983] [[0.98646971]]
Mean squared error: 0.3459
```

$$\hat{y} \approx 0.98 + 0.99x$$



Example

Alternative way (via pseudoinverse):

```
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
regr = linear_model.LinearRegression() # create linear regression object
regr.fit(X, y_prime) # train the model
y_predict = regr.predict(X) # make predictions
```

```
Parameters: [0.99530983] [[0.98646971]]
Mean squared error: 0.3459
```

Pros

- ✔ Analytically tractable, best-fit solutions
- ✔ Reduced computational complexity

Cons

- ✗ Impossible to fit nonlinear functions

Gradient descent

Gradient descent

Assume a general nonlinear hypothesis function $f_{\boldsymbol{\theta}}(\mathbf{x})$

$$\text{find } \hat{\boldsymbol{\theta}} \text{ s.t. } \frac{\partial \mathrm{MSE}}{\partial \hat{\boldsymbol{\theta}}}\left(\mathbf{X}, f_{\hat{\boldsymbol{\theta}}}\right) = 0 \implies \frac{1}{n}\sum_{k=1}^{n}\left(f_{\boldsymbol{\theta}}(\mathbf{x}^{(k)}) - y^{(k)}\right)\frac{\partial f_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} = 0.$$

Gradient descent

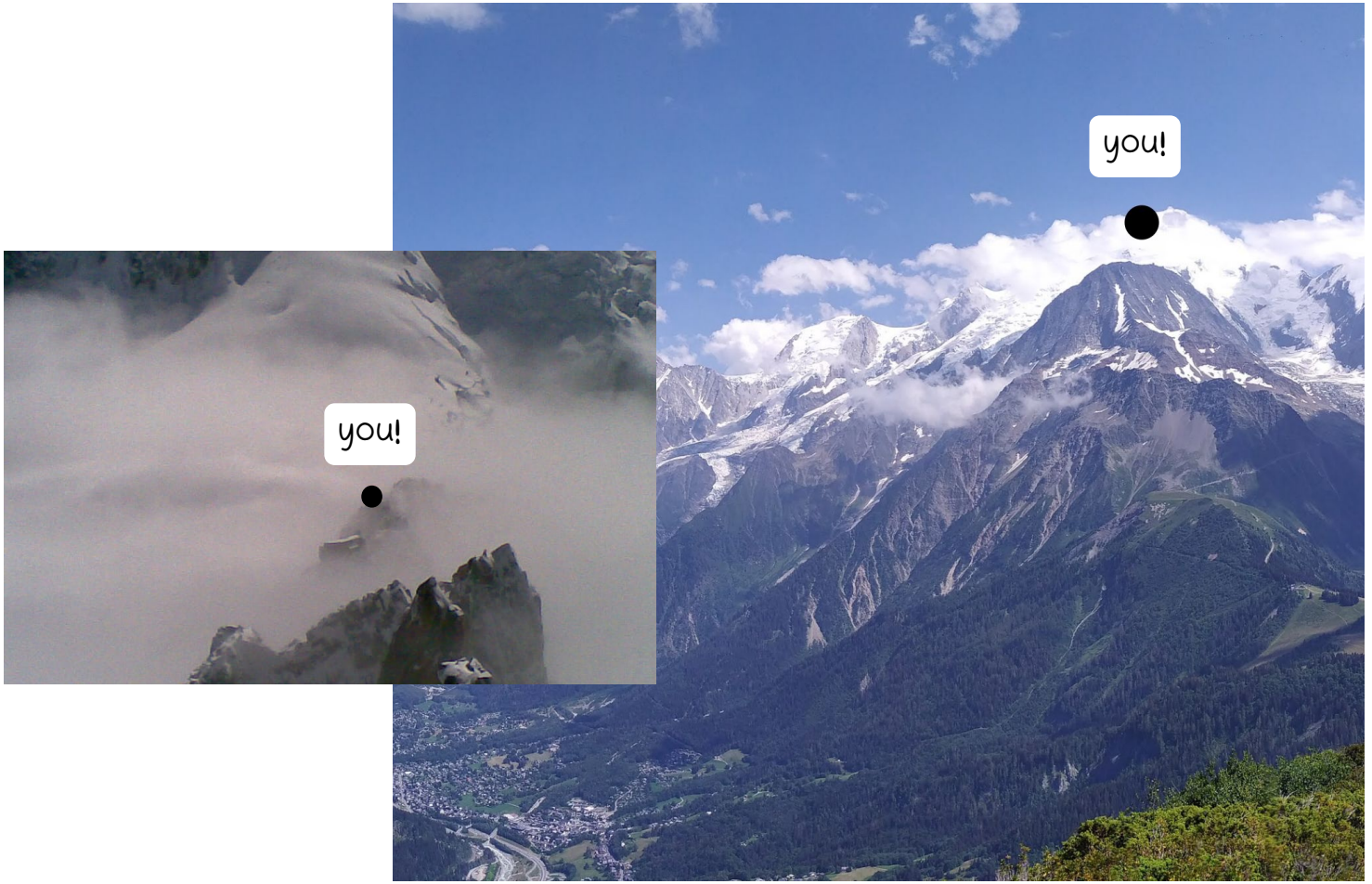
Assume a general nonlinear hypothesis function $f_{\boldsymbol{\theta}}(\mathbf{x})$

$$\text{find } \hat{\boldsymbol{\theta}} \text{ s.t. } \frac{\partial \text{MSE}}{\partial \hat{\boldsymbol{\theta}}}\left(\mathbf{X}, f_{\hat{\boldsymbol{\theta}}}\right) = 0 \implies \frac{1}{n}\sum_{k=1}^{n}\left(f_{\boldsymbol{\theta}}(\mathbf{x}^{(k)}) - y^{(k)}\right)\frac{\partial f_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} = 0.$$

Gradient descent

Assume a general nonlinear hypothesis function $f_{\boldsymbol{\theta}}(\mathbf{x})$

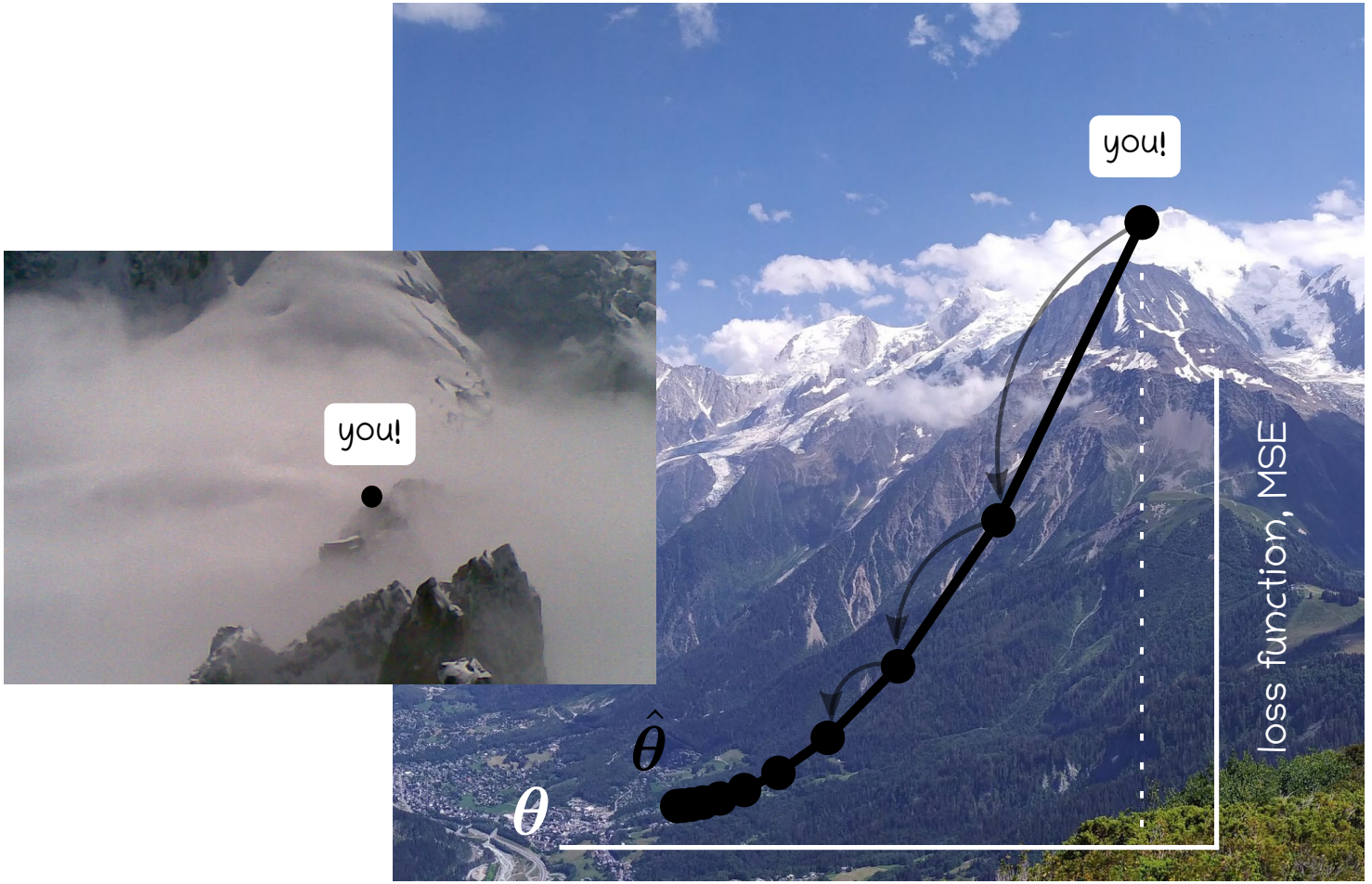
$$\text{find } \hat{\boldsymbol{\theta}} \text{ s.t. } \frac{\partial \text{MSE}}{\partial \hat{\boldsymbol{\theta}}}\left(\mathbf{X}, f_{\hat{\boldsymbol{\theta}}}\right) = 0 \implies \frac{1}{n}\sum_{k=1}^{n}\left(f_{\boldsymbol{\theta}}(\mathbf{x}^{(k)}) - y^{(k)}\right)\frac{\partial f_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} = 0.$$



Gradient descent

Assume a general nonlinear hypothesis function $f_{\boldsymbol{\theta}}(\mathbf{x})$

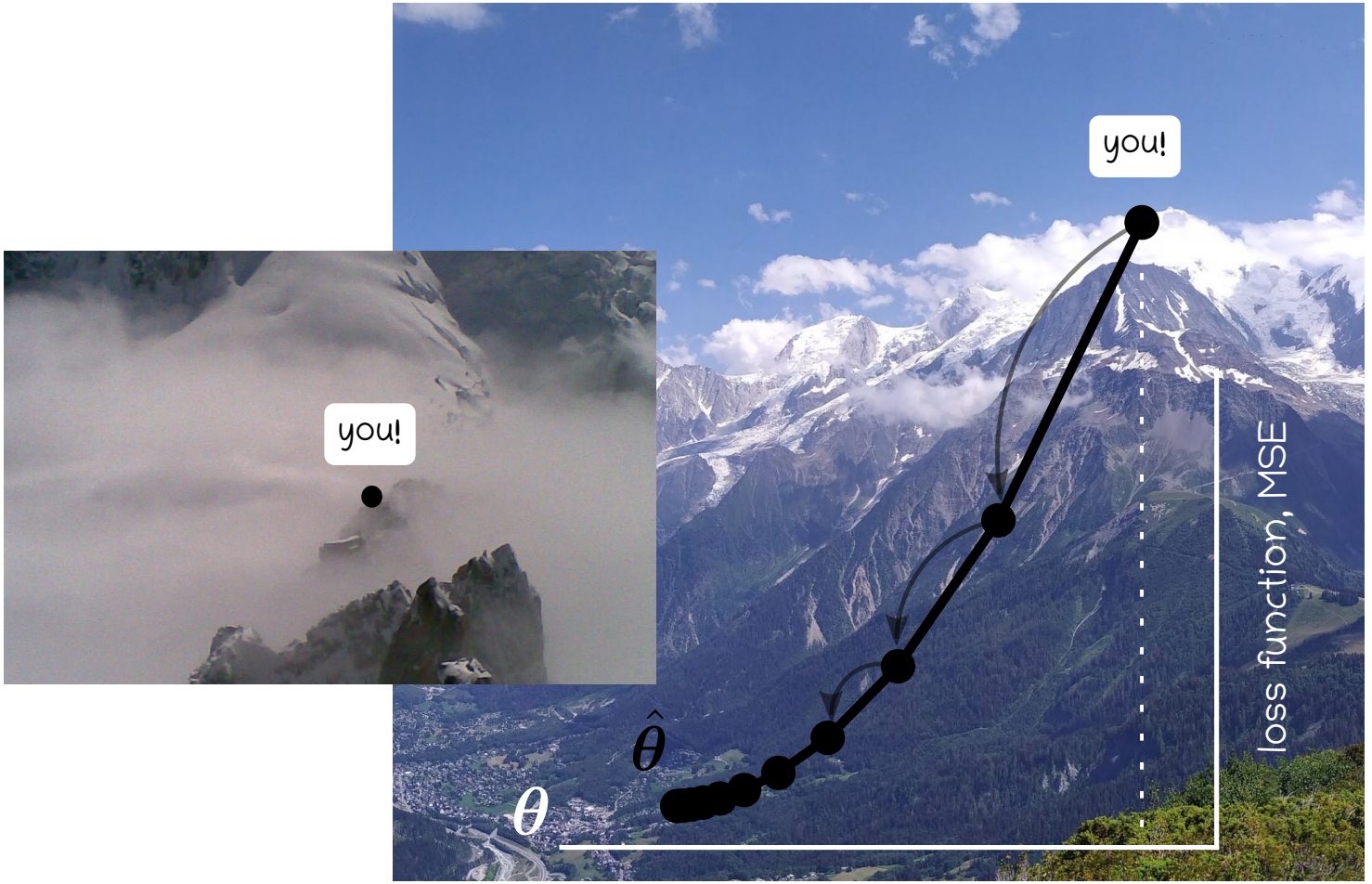
$$\boldsymbol{\theta}^{\text{ next step}} = \boldsymbol{\theta} - \frac{\partial \text{MSE}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}, f_{\boldsymbol{\theta}})$$



Gradient descent

Assume a general nonlinear hypothesis function $f_{\boldsymbol{\theta}}(\mathbf{x})$

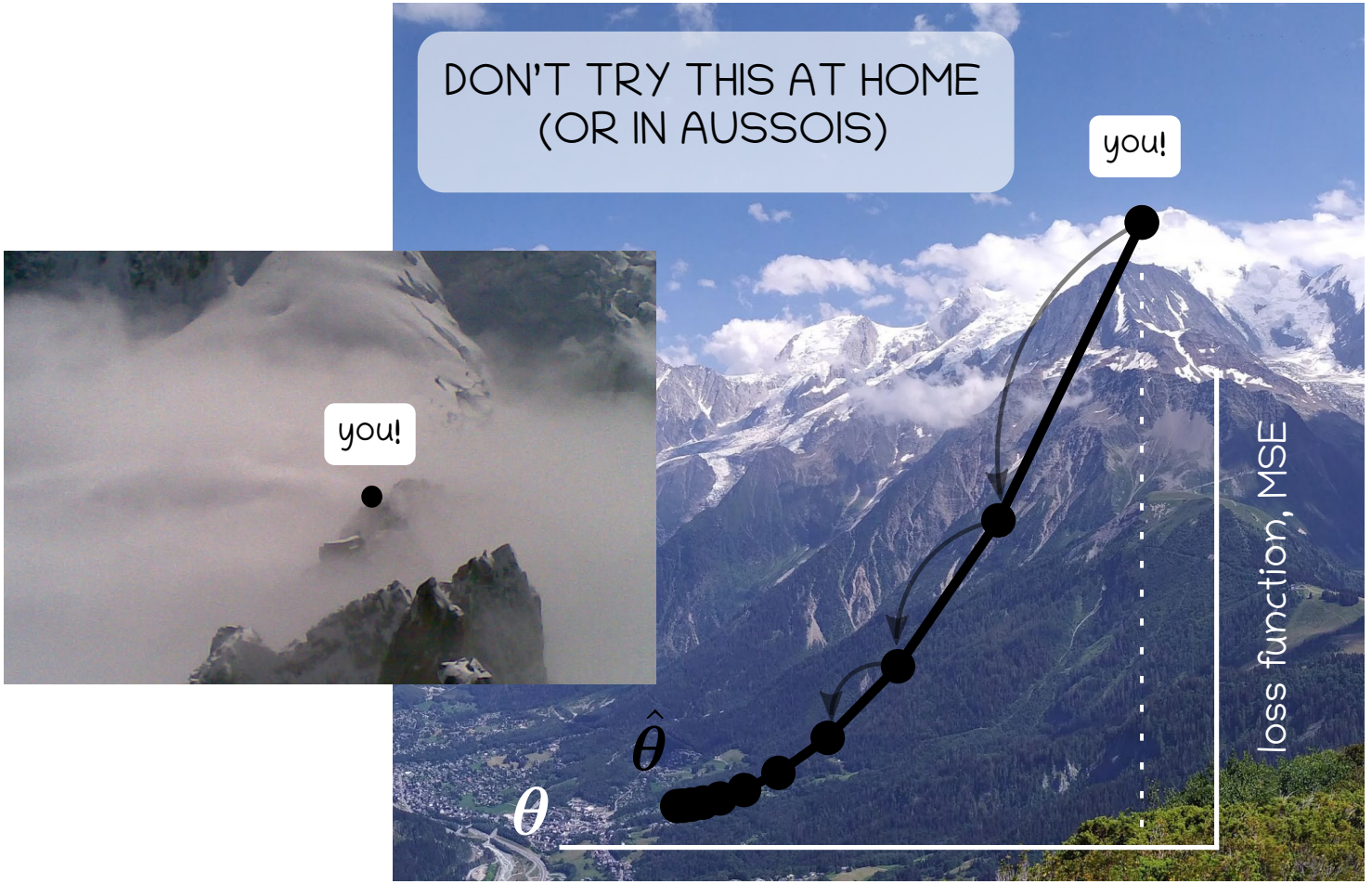
$$\boldsymbol{\theta}^{\text{ next step}} = \boldsymbol{\theta} - \eta \frac{\partial \text{MSE}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}, f_{\boldsymbol{\theta}})$$



Gradient descent

Assume a general nonlinear hypothesis function $f_{\boldsymbol{\theta}}(\mathbf{x})$

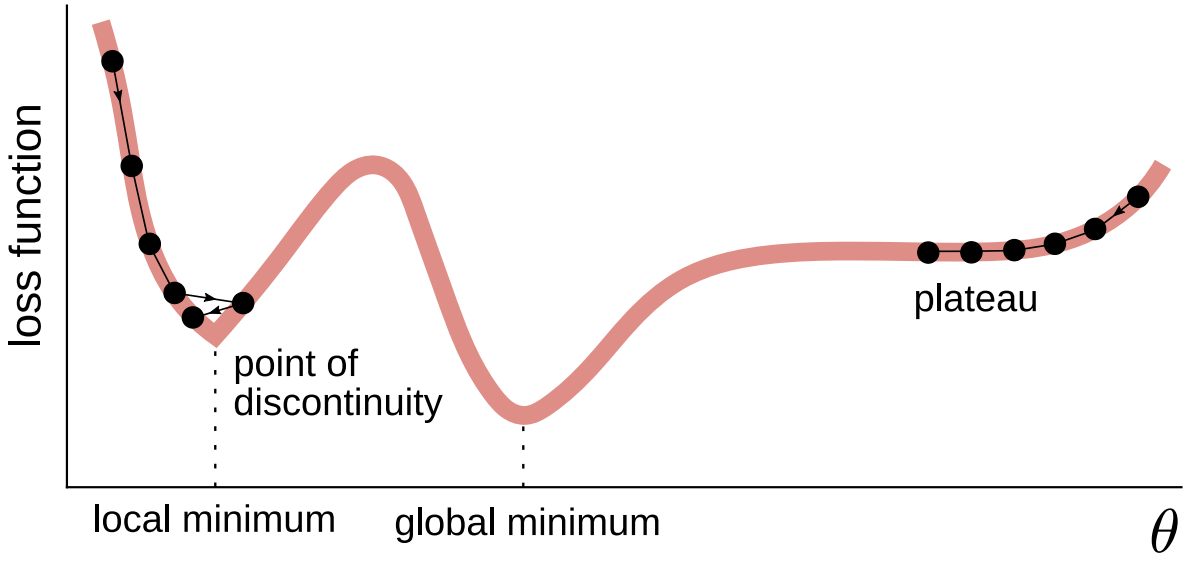
$$\boldsymbol{\theta}^{\text{ next step}} = \boldsymbol{\theta} - \eta \frac{\partial \text{MSE}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}, f_{\boldsymbol{\theta}})$$



Gradient descent

Optimizer equation $$\boldsymbol{\theta}^{\text{next step}} = \boldsymbol{\theta} - \eta \frac{\partial \text{MSE}}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}, f_{\boldsymbol{\theta}})$$

Learning rate η is a hyperparameter (!) — loss function and nonlinear hypothesis



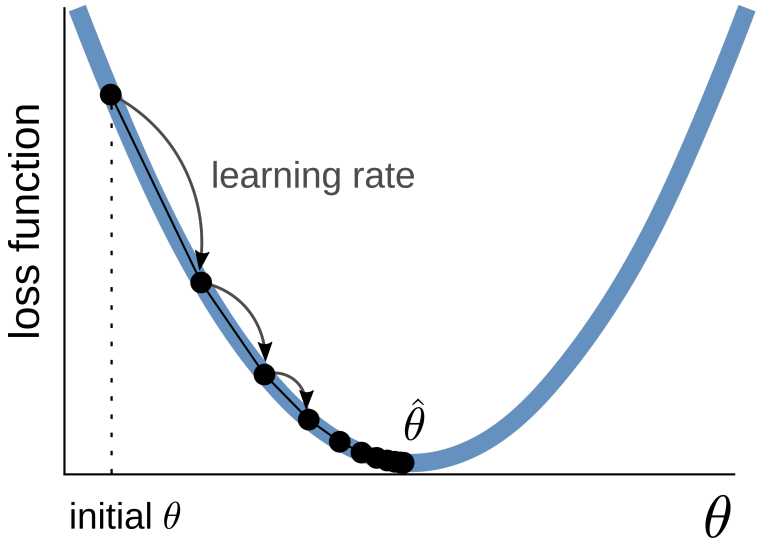
Gradient descent implementations:
batch, stochastic, mini-batch, etc. ... *tomorrow's lecture*

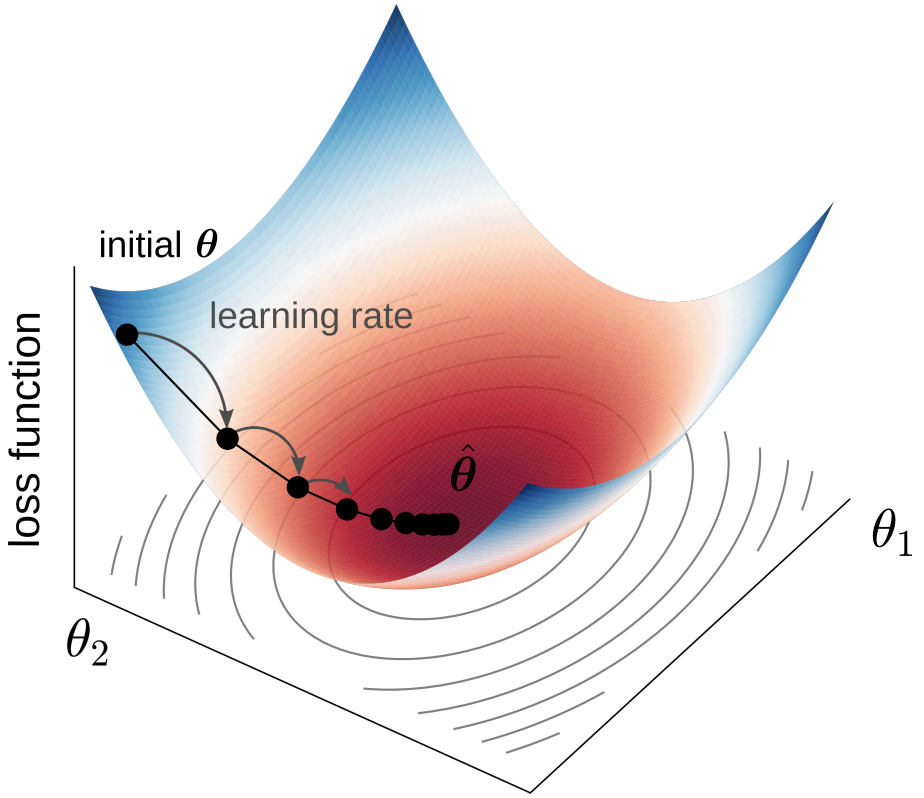
Batch GD

Use the full training set $\mathbf{X}$ at every step

$$\boldsymbol{\theta}^{\text{next step}} = \boldsymbol{\theta} - \eta \frac{\partial}{\partial \boldsymbol{\theta}} \text{MSE}\left(\mathbf{X}, f_{\boldsymbol{\theta}}\right)$$

$$\frac{\partial}{\partial \boldsymbol{\theta}} \text{MSE}\left(\mathbf{X}, f_{\boldsymbol{\theta}}\right) = \frac{2}{n} \mathbf{X}^{\text{T}} \left(\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\right)$$





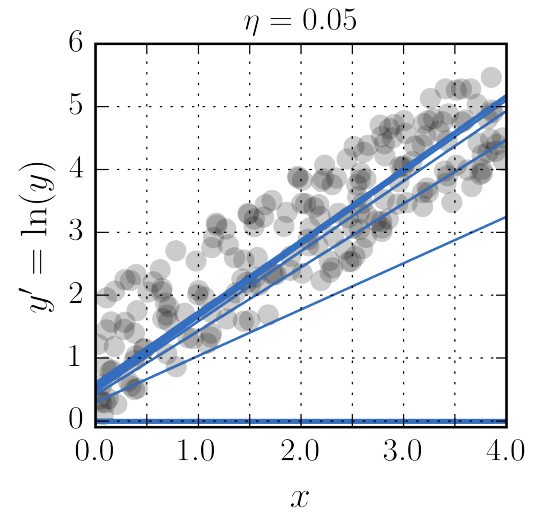
Batch GD > SVD if m large (features)

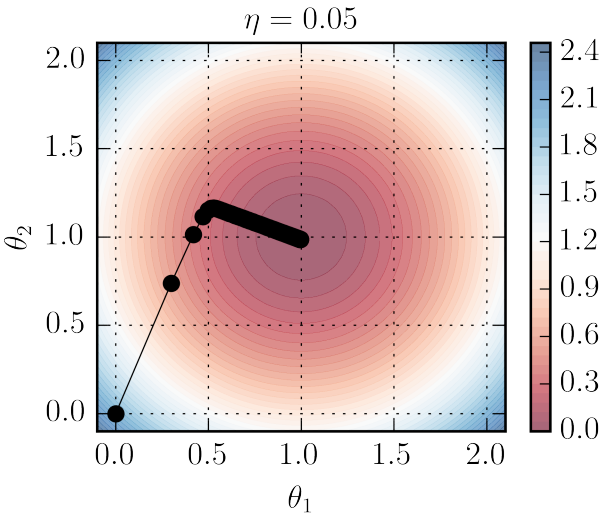
Batch GD – learning rate

```
eta = 0.05 # learning rate
n_epochs = 500 # no epochs
theta = np.array([[0.],[0.]]) # initialization
for epoch in range(n_epochs):
   gradients = 2/n_snapshots * X_p.T.dot(X_p.dot(theta) - y_prime)
   theta = theta - eta * gradients # optimizer
```

```
[[0.99530187] [0.98647279]]
```

waiting time $\backsim \mathcal{O}(1/\epsilon)$



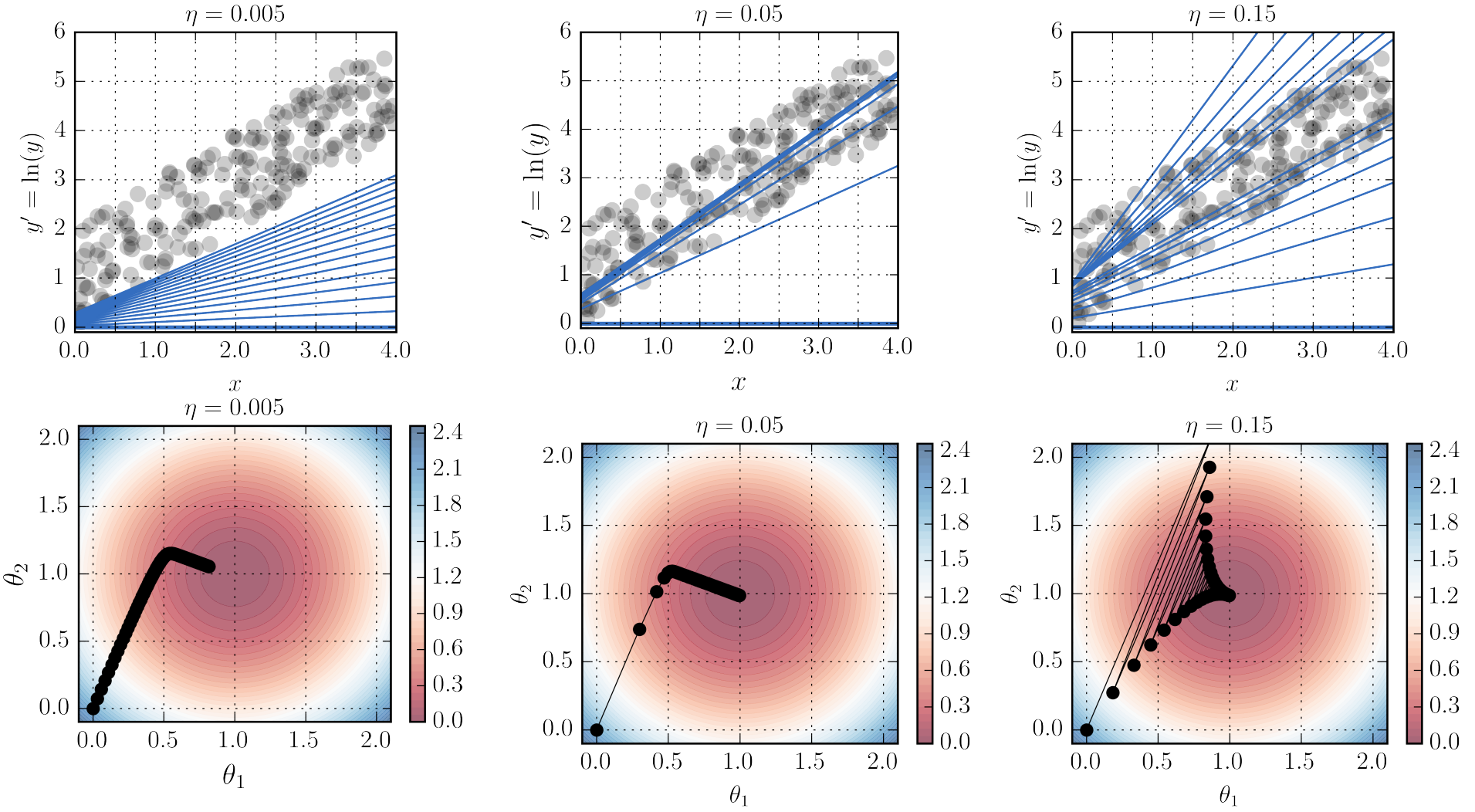


Batch GD – learning rate

```
eta = 0.05 # learning rate
n_epochs = 500 # no epochs
theta = np.array([[0.],[0.]]) # initialization
for epoch in range(n_epochs):
   gradients = 2/n_snapshots * X_p.T.dot(X_p.dot(theta) - y_prime)
   theta = theta - eta * gradients # optimizer
```

```
[[0.99530187] [0.98647279]]
```

waiting time $\backsim \mathcal{O}(1/\epsilon)$



Stochastic GD

Use a single (random) data point from the training set $\mathbf{X}$

- efficient memory allocation (> batch GD)
- nonsmooth behavior (< batch GD)

$$\boldsymbol{\theta}^{\text{next step}} = \boldsymbol{\theta} - \eta \frac{\partial}{\partial \boldsymbol{\theta}} \text{MSE}\left(\mathbf{X}, f_{\boldsymbol{\theta}}\right)$$

Stochastic GD

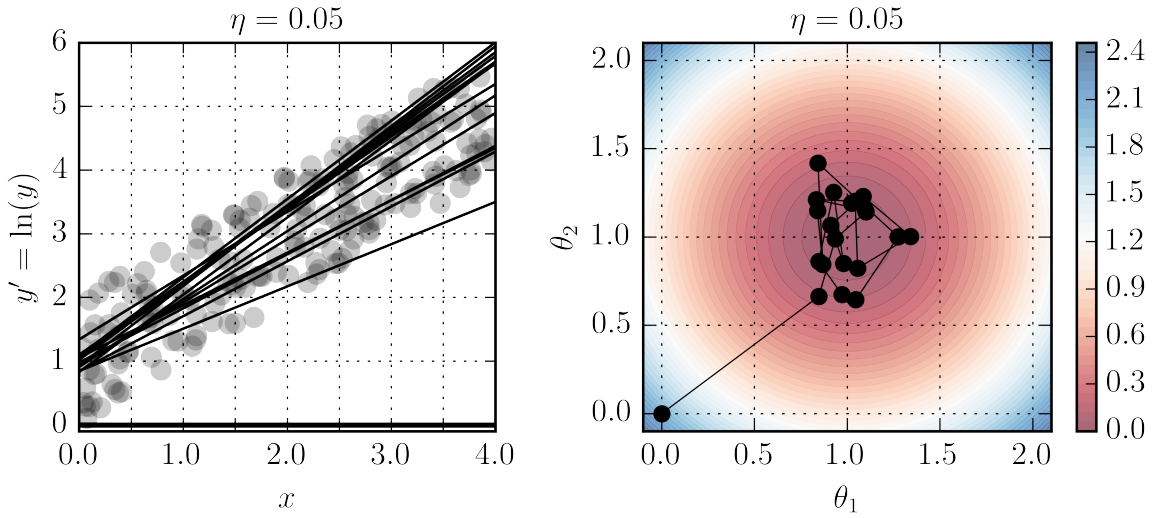
Use a single (random) data point from the training set $\mathbf{X}$

- efficient memory allocation (> batch GD)
- nonsmooth behavior (< batch GD)

$$\boldsymbol{\theta}^{\text{next step}} = \boldsymbol{\theta} - \eta \frac{\partial}{\partial \boldsymbol{\theta}} \text{MSE}\left(\mathbf{X}, f_{\boldsymbol{\theta}}\right)$$

```
eta = 0.05 #learning rate
n_epochs = 20 # no epochs
theta = np.array([[0.],[0.]]) # initialization
for epoch in range(n_epochs):
  for i in range(n_snapshots): # iterate over each snapshot
    random_index = np.random.randint(n_snapshots) # pick random snapshot
    xi = X_p[random_index:random_index+1]
    yi = y_prime[random_index:random_index+1]
    gradients = 2 * xi.T.dot(xi.dot(theta) - yi)
    theta = theta - eta * gradients
```

```
[[0.83905266] [1.15041967]]
```



Stochastic GD

Use a single (random) data point from the training set $\mathbf{X}$

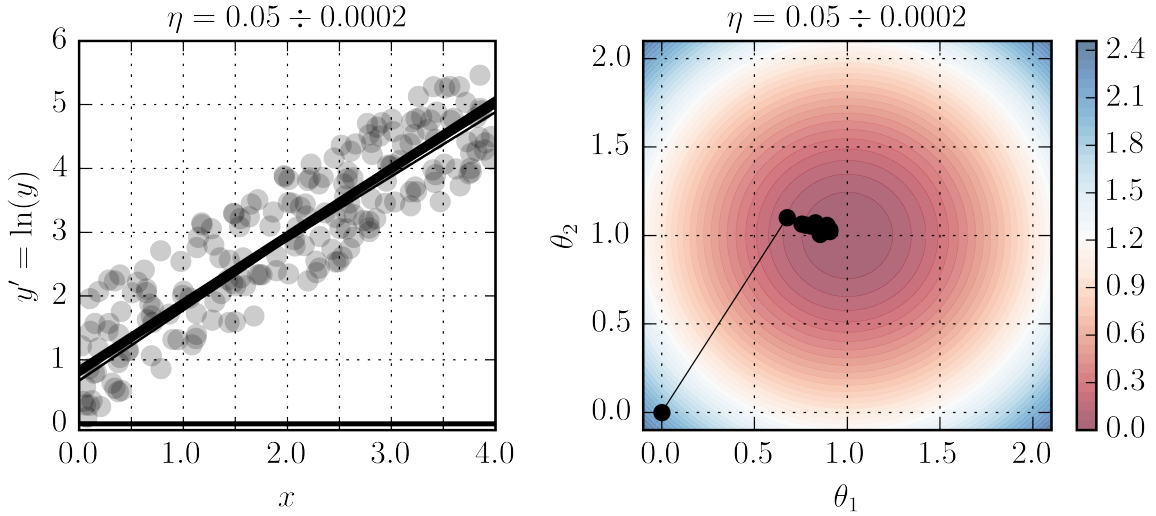
- efficient memory allocation (> batch GD)
- nonsmooth behavior (< batch GD)

$$\boldsymbol{\theta}^{\text{next step}} = \boldsymbol{\theta} - \eta \frac{\partial}{\partial \boldsymbol{\theta}} \text{MSE}\left(\mathbf{X}, f_{\boldsymbol{\theta}}\right)$$

highly dependent on learning rate and initialization

```
eta = 0.05 #learning rate
n_epochs = 20 # no epochs
theta = np.array([[0.],[0.]]) # initialization
t0, t1 = 1, 100 # learning schedule hyperparameters
def learning_schedule(t):
  return t0 / (t + t1)
for epoch in range(n_epochs):
  for i in range(n_snapshots): # iterate over each snapshot
    random_index = np.random.randint(n_snapshots) # pick random snapshot
    xi = X_p[random_index:random_index+1]
    yi = y_prime[random_index:random_index+1]
    eta = learning_schedule(epoch * n_snapshots + i) # schedule
    gradients = 2 * xi.T.dot(xi.dot(theta) - yi)
    theta = theta - eta * gradients
```

```
[[0.90607535] [1.02276101]]
```



Mini-batch GD

Use random subsets of b data from the training set $\mathbf{X}$ **(mini-batches)**

- optimized matrix ops, smoother behavior (> stochastic, batch GD)

$$\boldsymbol{\theta}^{\text{next step}} = \boldsymbol{\theta} - \eta \frac{\partial}{\partial \boldsymbol{\theta}} \text{MSE}\left(\mathbf{X}, f_{\boldsymbol{\theta}}\right)$$

Mini-batch GD

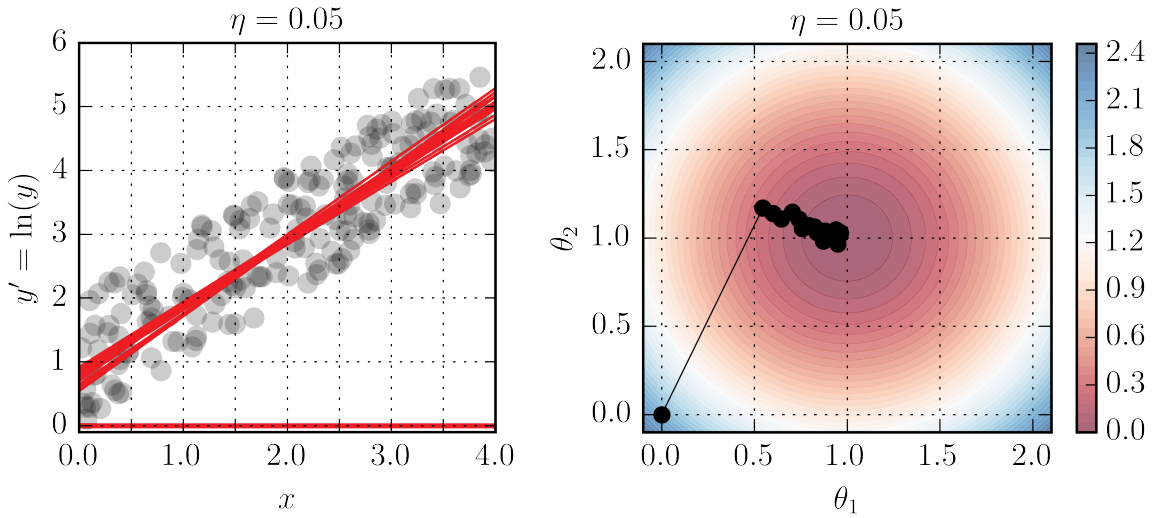
Use random subsets of b data from the training set $\mathbf{X}$ **(mini-batches)**

- optimized matrix ops, smoother behavior (> stochastic, batch GD)

$$\boldsymbol{\theta}^{\text{next step}} = \boldsymbol{\theta} - \eta \frac{\partial}{\partial \boldsymbol{\theta}} \text{MSE}\left(\mathbf{X}, f_{\boldsymbol{\theta}}\right)$$

```
eta = 0.05 #learning rate
theta = np.array([[0.],[0.]]) #initialization
batch_size = 30 # define mini-batches size
for epoch in range(n_epochs):
  for batch in iterate_minibatches(X_p, y_prime, batch_size, shuffle=True):
    x_batch, y_batch = batch
    gradients = 2/batch_size * x_batch.T.dot(x_batch.dot(theta) - y_batch)
    theta = theta - eta * gradients
```

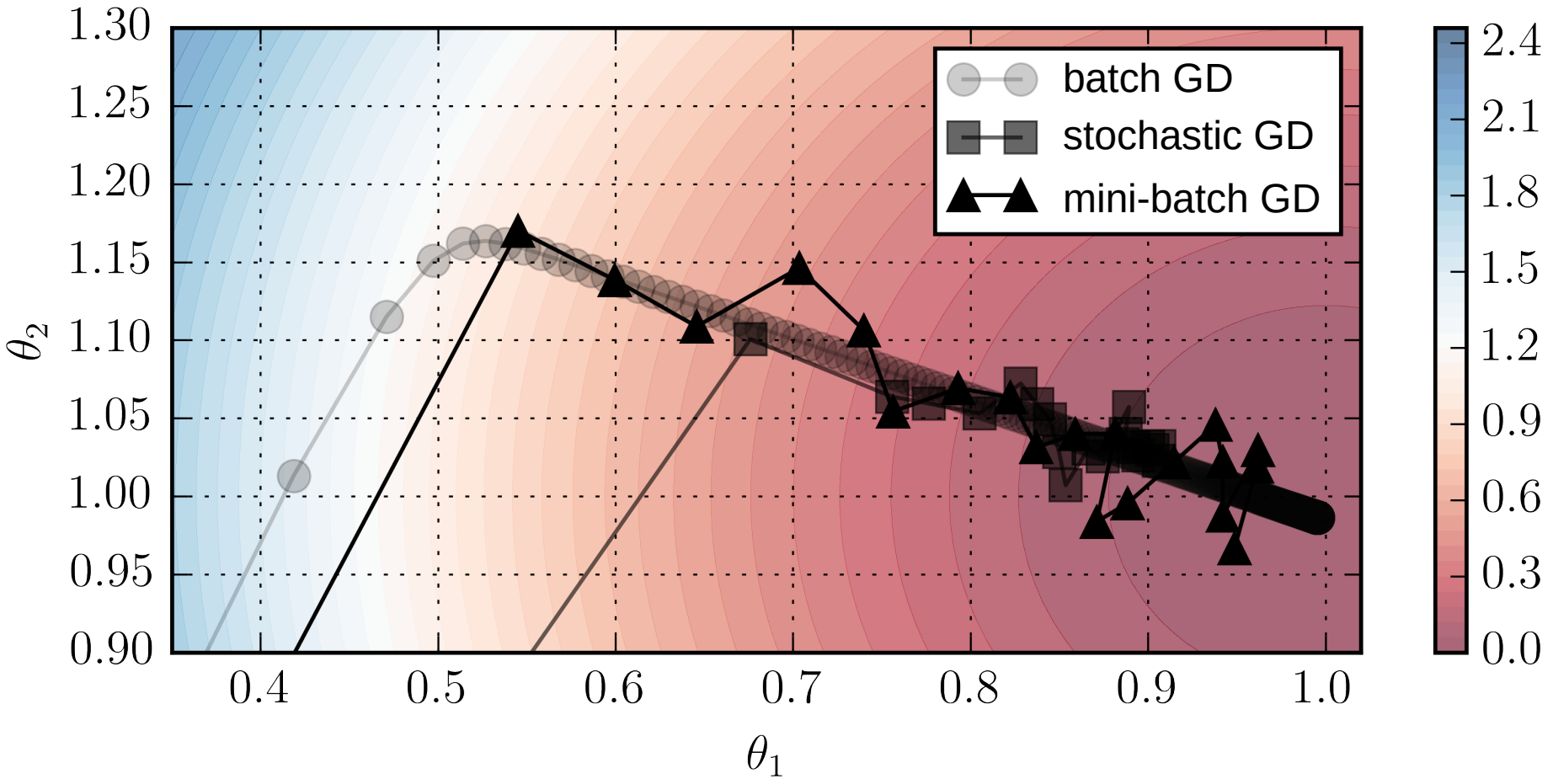
```
[[0.94862053] [0.96679924]]
```



Batch vs stochastic vs mini-batch GD

Mini-batch > stochastic/batch GD

Rule of thumb: mini-batch size $b = 1$ (stochastic) $b = n$ (batch)

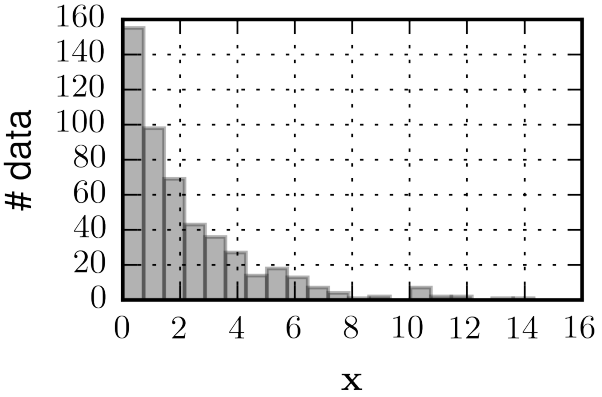


Features scaling and model validation

Feature scaling

Optimization algorithms struggle with features of different scales

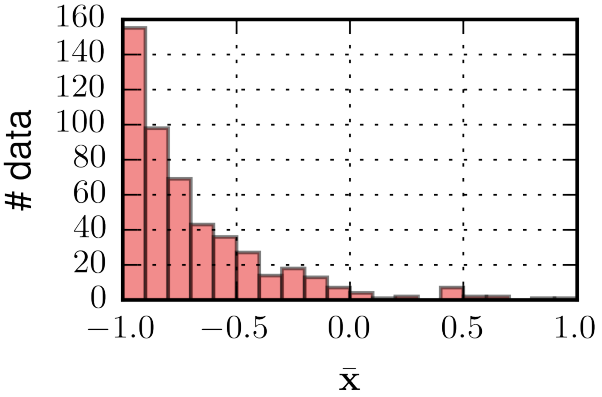
$$\underbrace{\bar{\mathbf{x}}}_{\text{scaled feature}} = \frac{1}{\alpha}\left(\underbrace{\mathbf{x}}_{\text{feature}} - \beta\right)$$



Normalization

$$\alpha \equiv \frac{1}{2}\left(\max\left(\mathbf{x}\right) - \min\left(\mathbf{x}\right)\right)$$

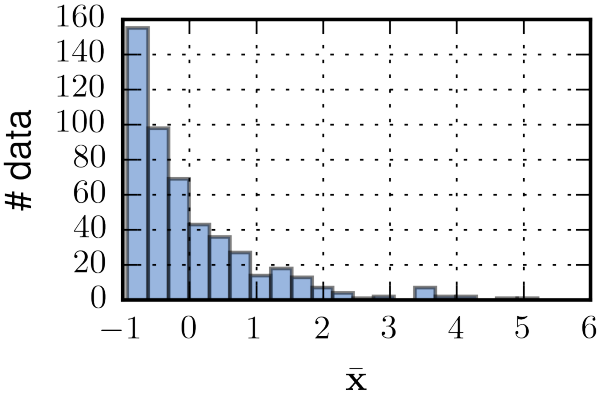
$$\beta \equiv \alpha + \min\left(\mathbf{x}\right)$$



Standardization

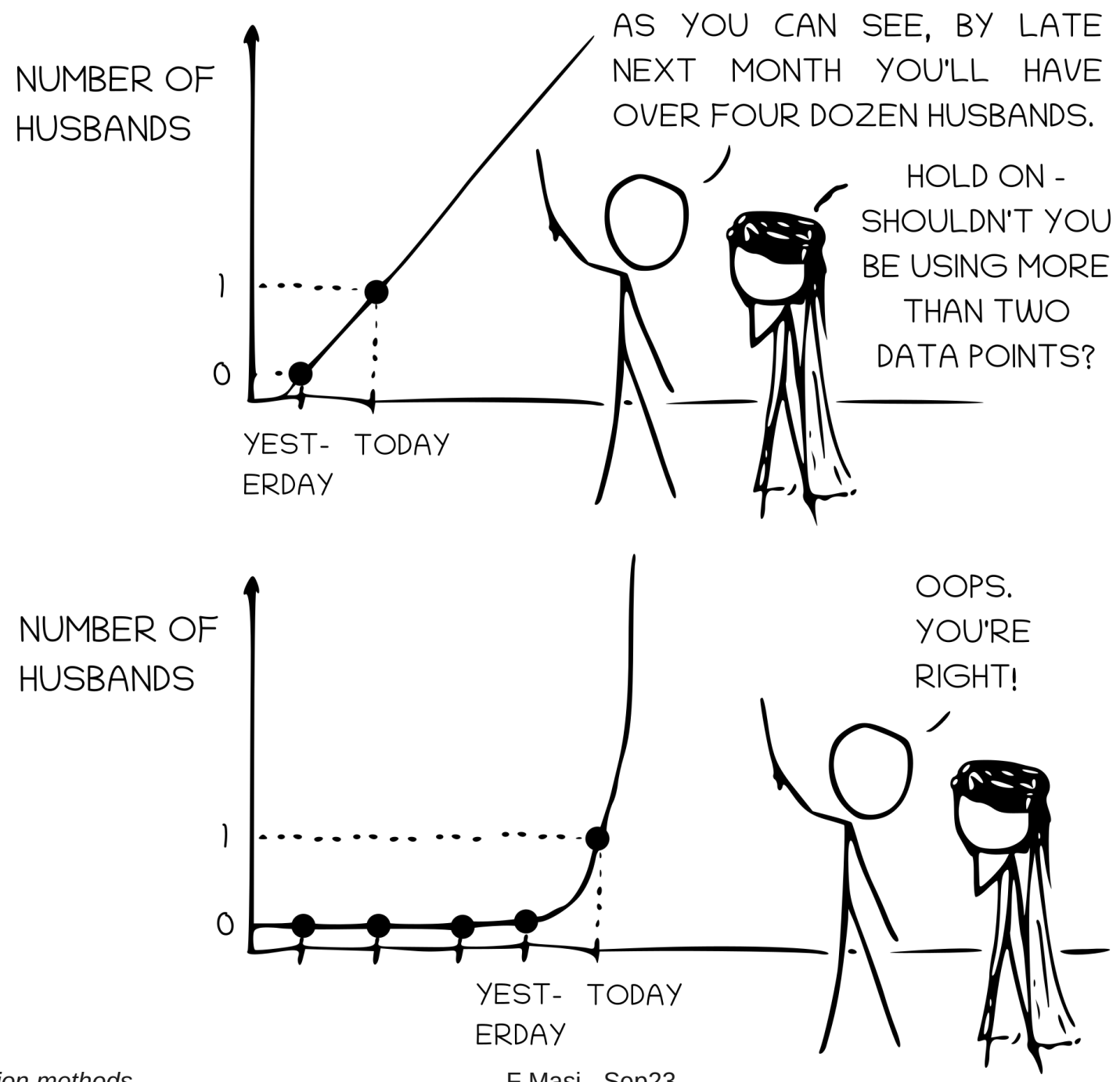
$$\alpha \equiv \sigma\left(\mathbf{x}\right)$$

$$\beta \equiv \mu\left(\mathbf{x}\right)$$

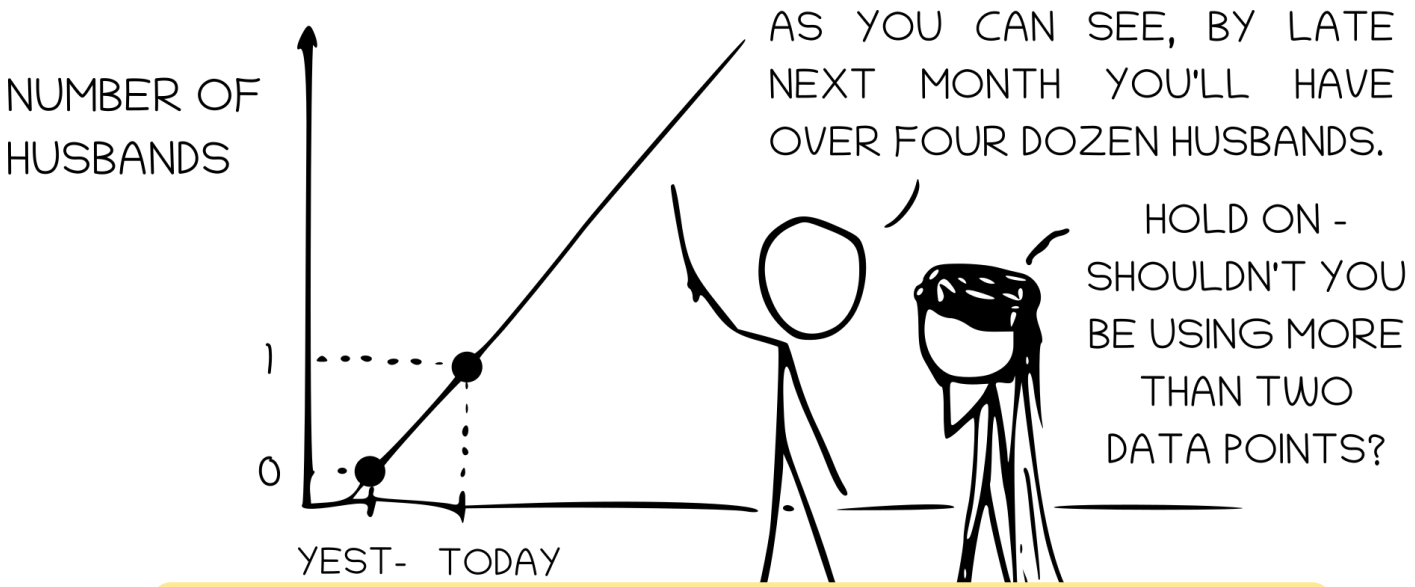


μ mean, σ standard deviation

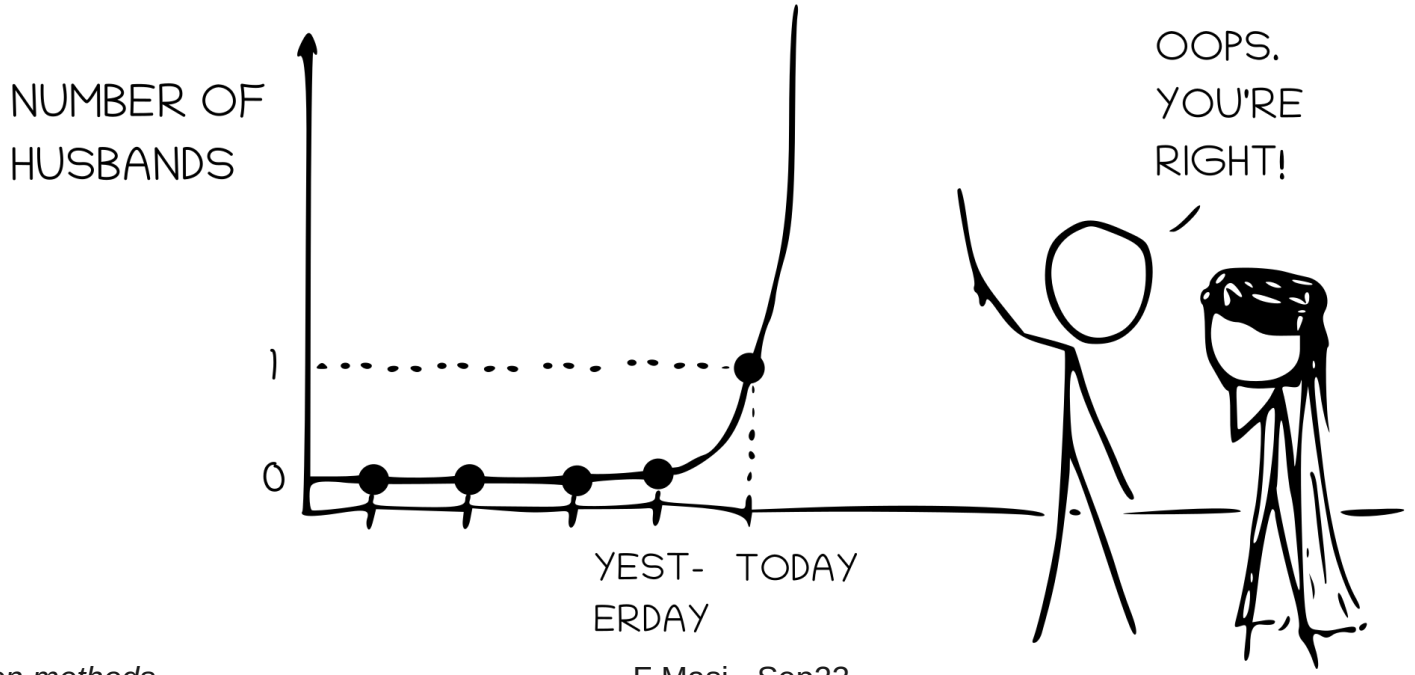
Test and validation of a model



Test and validation of a model



ML is all about having models that generalize well



Test and validation of a model

Interpolation: ability to predict values within the range of observed data points.

Generalization: ability to make predictions for new, previously unseen data, drawn from the same distribution as the ones used to train the model.

Extrapolation: ability to predict values beyond the range of the training set.

Test and validation of a model

Interpolation: ability to predict values within the range of observed data points.

Generalization: ability to make predictions for new, previously unseen data, drawn from the same distribution as the ones used to train the model.

Extrapolation: ability to predict values beyond the range of the training set.

...always test models to quantify their generalization capabilities!

Test and validation of a model

Interpolation: ability to predict values within the range of observed data points.

Generalization: ability to make predictions for new, previously unseen data, drawn from the same distribution as the ones used to train the model.

Extrapolation: ability to predict values beyond the range of the training set.

...always test models to quantify their generalization capabilities!

Test: $$\{\mathbf{X}, \mathbf{y}\} \leftarrow \mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}}$$

Test and validation of a model

Interpolation: ability to predict values within the range of observed data points.

Generalization: ability to make predictions for new, previously unseen data, drawn from the same distribution as the ones used to train the model.

Extrapolation: ability to predict values beyond the range of the training set.

...always test models to quantify their generalization capabilities!

Test: $$\{\mathbf{X}, \mathbf{y}\} \leftarrow \mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}}$$

Hold-out validation: $$\{\mathbf{X}, \mathbf{y}\} \leftarrow \underbrace{\mathcal{D}}_{100\%} = \underbrace{\mathcal{D}_{\text{train}}}_{60\%} \cup \underbrace{\mathcal{D}_{\text{val}}}_{20\%} \cup \underbrace{\mathcal{D}_{\text{test}}}_{20\%}$$

Test and validation of a model

Interpolation: ability to predict values within the range of observed data points.

Generalization: ability to make predictions for new, previously unseen data, drawn from the same distribution as the ones used to train the model.

Extrapolation: ability to predict values beyond the range of the training set.

...always test models to quantify their generalization capabilities!

Test: $$\{\mathbf{X}, \mathbf{y}\} \leftarrow \mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}}$$

Hold-out validation: $$\{\mathbf{X}, \mathbf{y}\} \leftarrow \underbrace{\mathcal{D}}_{100\%} = \underbrace{\mathcal{D}_{\text{train}}}_{60\%} \cup \underbrace{\mathcal{D}_{\text{val}}}_{20\%} \cup \underbrace{\mathcal{D}_{\text{test}}}_{20\%}$$

***k*-fold cross-validation:** $$\mathcal{D}_{\text{train+val}} = \bigcup_{l=1}^{k} \mathcal{D}^{(l)} \Rightarrow \quad \hat{y}_{(l)} = f_{\boldsymbol{\theta}_{(l)}}(\mathbf{x}_{(l)})$$

$$\boldsymbol{\theta} = 1/k \sum_{l=1}^{k} \boldsymbol{\theta}_{(l)}$$


my simple
linear
regression
model


my simple linear regression model
outliers
correlated features


my simple linear regression model
outliers
correlated features
non-linearity

Nonlinear regression

Nonlinear regression

We can actually fit nonlinear data with a linear model!

$$\mathbf{X} \leftarrow \boldsymbol{\phi}(\mathbf{x}) = \underbrace{\begin{bmatrix} \boldsymbol{\phi}_1(\mathbf{x}) \\ \boldsymbol{\phi}_2(\mathbf{x}) \\ \vdots \\ \boldsymbol{\phi}_l(\mathbf{x}) \end{bmatrix}}_{\text{basis functions}}$$

(polynomials, trigonometric,
radial basis functions, etc.)

$$\Downarrow$$

$$\hat{y} = \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})$$

$$\Downarrow$$

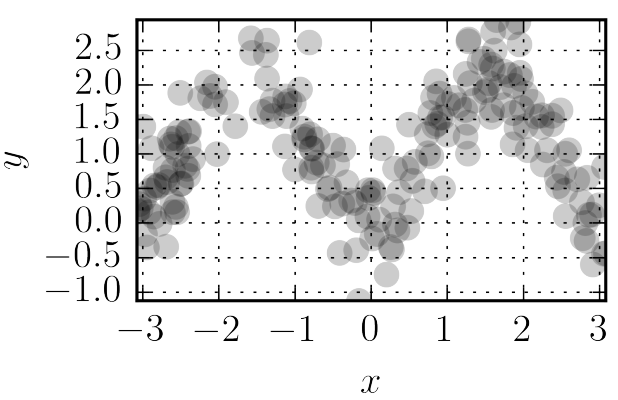
... exactly as before!

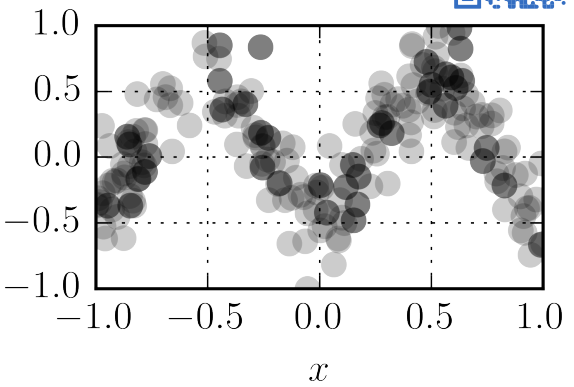
End-to-end example

$$y = \sin(|x|) + \sin(x^2) + \mathcal{N}(0, \sigma)$$

```
from sklearn.model_selection import
train_test_split
X_train,X_test,y_train,y_test =
train_test_split(X,y,test_size=0.2)

param_x = scaling(X_train,fit=True)
param_y = scaling(y_train,fit=True)
# scale X,y (training and test sets)
norm_X_train = scaling(X_train,
                       transform=True,
                       param=param_x)
norm_y_train = scaling(y_train,
                       transform=True,
                       param=param_y)
```






```
def scaling(x, fit=False, transform=False, inverse_transform=False,
            norm=True,param=None):
  if fit==True:
    if norm==True:
      min_ = np.amin(x); max_ = np.amax(x)
      a = 0.5*(max_-min_)
      b = 0.5*(max_+min_)
    else:
      a = np.std(x)
      b = np.mean(x)
    return [a,b]
  elif transform==True:
     return np.divide(x-param[1],param[0])
  elif inverse_transform==True:
     return np.multiply(x,param[0])+param[1]
```

or use Scikit-learn's `preprocessing` module.

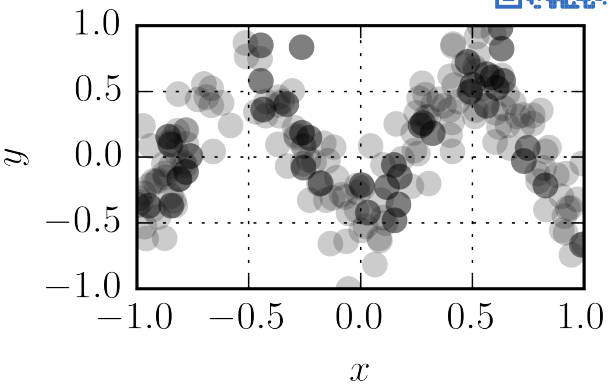
End-to-end example

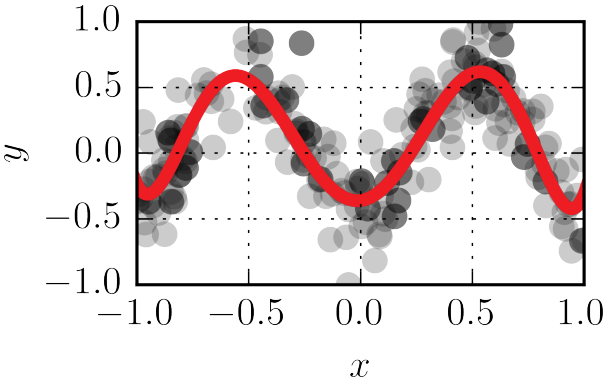
Pick polynomial basis functions (degree=6)

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
poly_features = PolynomialFeatures(degree=6, include_bias=False)
norm_X_poly = poly_features.fit_transform(norm_X_train)
lin_reg = LinearRegression()
lin_reg.fit(norm_X_poly,norm_y_train) # perform linear regression
lin_reg.intercept_, lin_reg.coef_ # theta_0 and theta_k
```

```
(array([-0.35987146]),
 array([[ 0.20041976,    0.48763306,   7.23777176,   -0.72575099,
          -16.10685101,  8.95138004]]))
```

$$\hat{y} = -0.36 + 0.20x + 7.24x^2 - 0.73x^3 - 16.11x^4 + 0.49x^5 + 8.95x^6$$






```
x = np.expand_dims(np.linspace(-np.pi,np.pi,200),1)
norm_x = scaling(x,transform=True,param=param_x)
norm_x_poly = poly_features.fit_transform(norm_x)
norm_y_predicted = lin_reg.predict(norm_x_poly)
from sklearn.metrics import mean_squared_error
norm_X_test_poly = poly_features.fit_transform(norm_X_test)
norm_y_predicted_train = lin_reg.predict(norm_X_poly)
norm_y_predicted_test = lin_reg.predict(norm_X_test_poly)
print('MSE on train set: ',
mean_squared_error(norm_y_predicted_train,norm_y_train))
print('MSE on test set: ',
mean_squared_error(norm_y_predicted_test,norm_y_test))
```

```
MSE on training data set: 0.05173368072737507
MSE on test data set: 0.05142192371233674
```

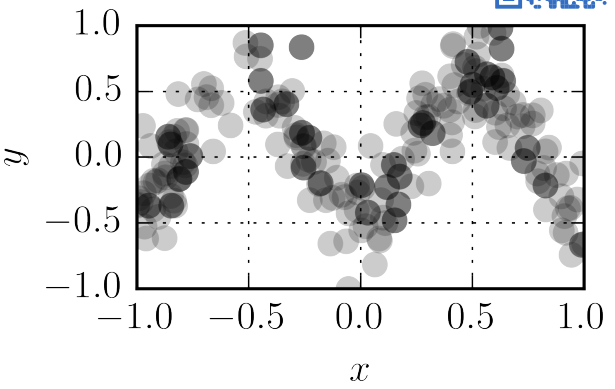
End-to-end example

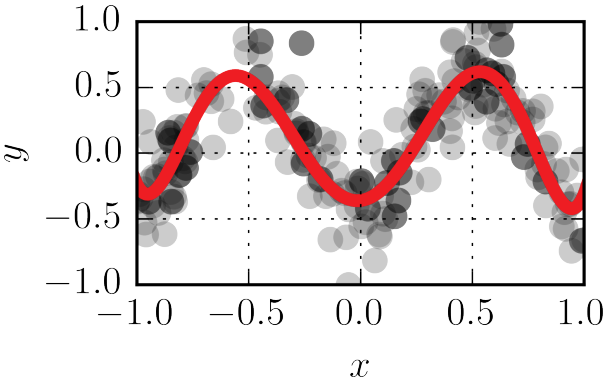
Pick polynomial basis functions (degree=6)

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
poly_features = PolynomialFeatures(degree=6, include_bias=False)
norm_X_poly = poly_features.fit_transform(norm_X_train)
lin_reg = LinearRegression()
lin_reg.fit(norm_X_poly,norm_y_train) # perform linear regression
lin_reg.intercept_, lin_reg.coef_ # theta_0 and theta_k
```

```
(array([-0.35987146]),
 array([[ 0.20041976,    0.48763306,   7.23777176,   -0.72575099,
          -16.10685101,  8.95138004]]))
```

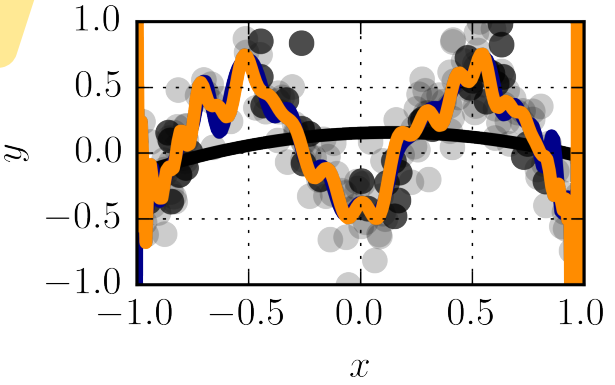
$$\hat{y} = -0.36 + 0.20x + 7.24x^2 - 0.73x^3 - 16.11x^4 + 0.49x^5 + 8.95x^6$$





hyperparameters selection

```
degree = [2,6,50,100] # different polynomial degree
color = ['black','red','darkblue','darkorange']
fig = plt.figure(figsize=(3., 2.))
plt.plot(norm_X_train,norm_y_train,'ko',alpha=0.2)
plt.plot(norm_X_test,norm_y_test,'ko')
for i in range(len(degree)):
  poly_features = PolynomialFeatures(degree=degree[i],
                                     include_bias=False)
  norm_X_poly = poly_features.fit_transform(norm_X_train)
  lin_reg = LinearRegression() # linear regression model
  lin_reg.fit(norm_X_poly,norm_y_train) # fit model
  norm_x_poly = poly_features.fit_transform(norm_x)
  norm_y_predicted = lin_reg.predict(norm_x_poly)
  plt.plot(norm_x,norm_y_predicted,'-',color=color[i],linewidth=3,
           label=str(degree[i]))
```

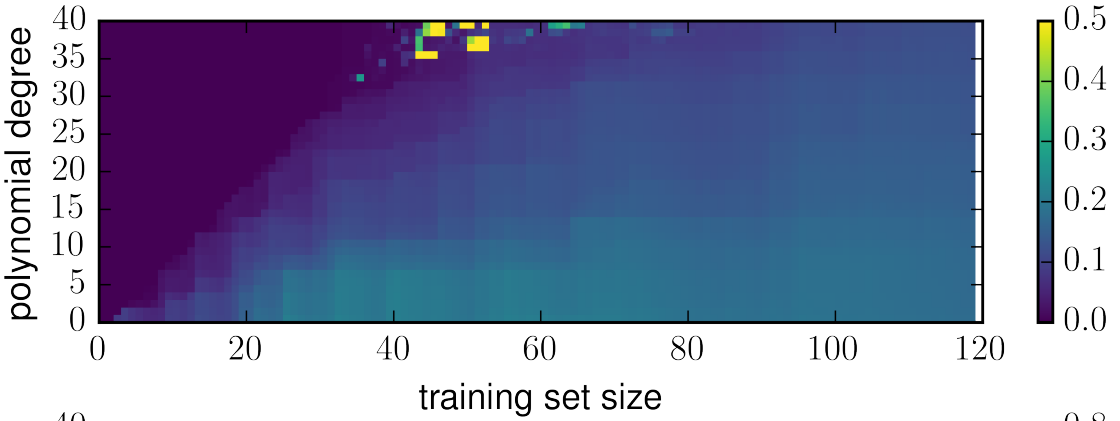


End-to-end example

Model validation

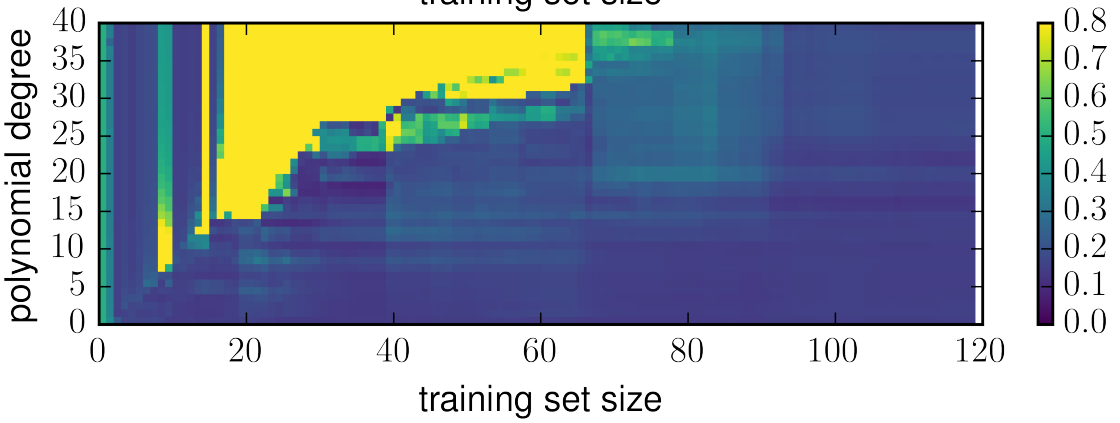
model training

$\mathcal{D}_{\text{train}}$



model selection

$\mathcal{D}_{\text{val}}$

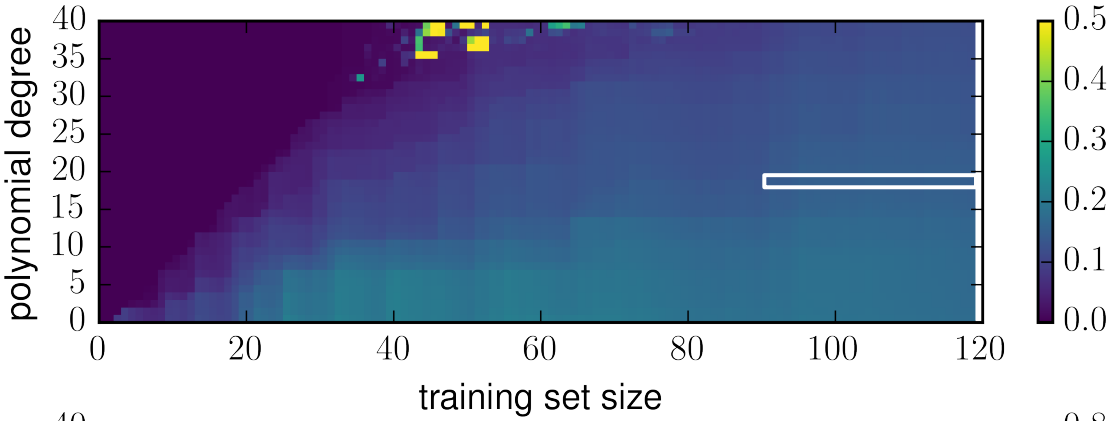


End-to-end example

Model validation

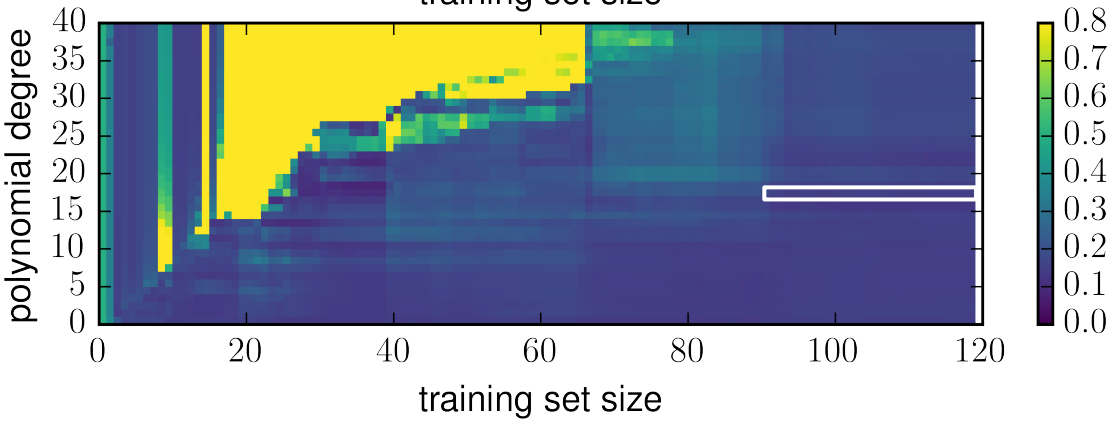
model training

$\mathcal{D}_{\text{train}}$



model selection

$\mathcal{D}_{\text{val}}$

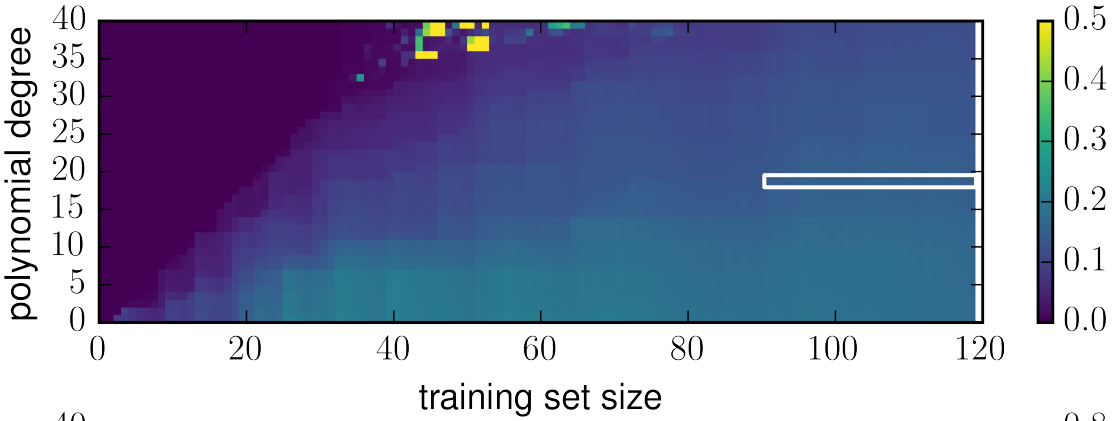


End-to-end example

Model validation

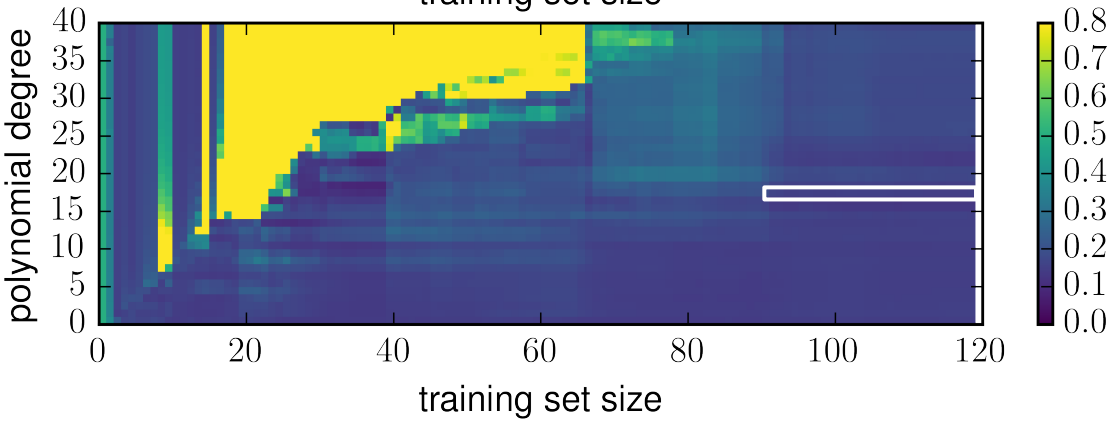
model training

$\mathcal{D}_{\text{train}}$



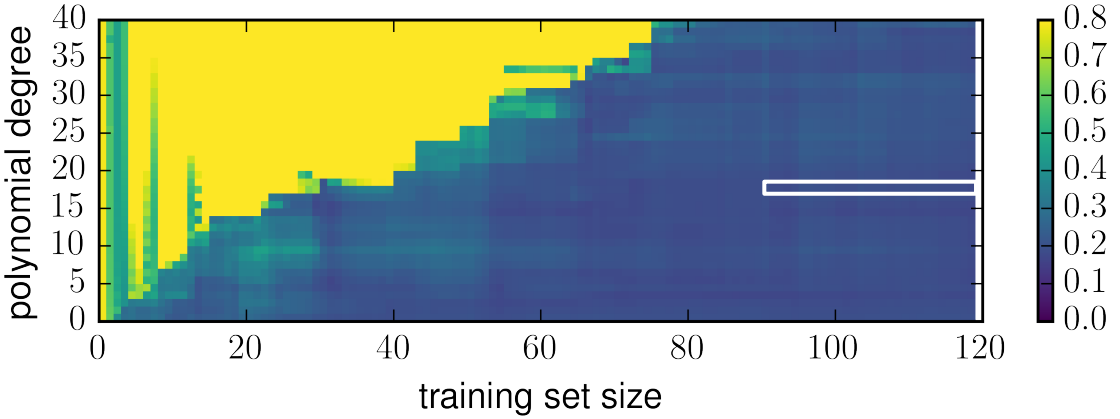
model selection

$\mathcal{D}_{\text{val}}$

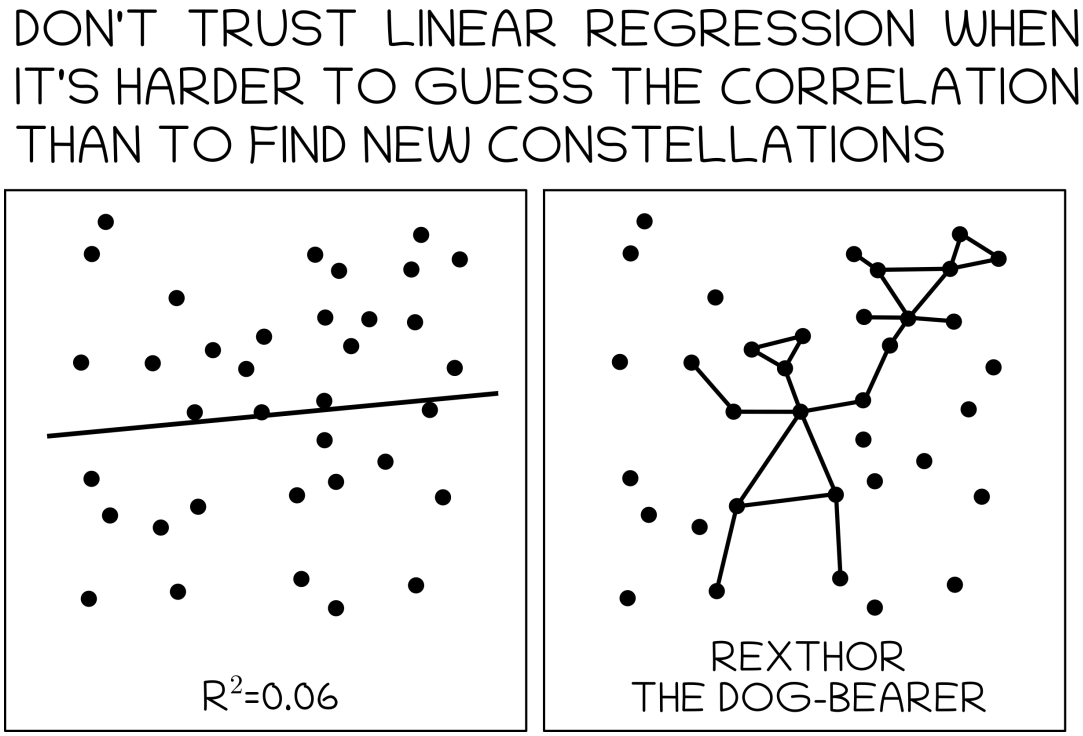


deployment

$\mathcal{D}_{\text{test}}$



Regularized techniques



Over- and under-determined systems

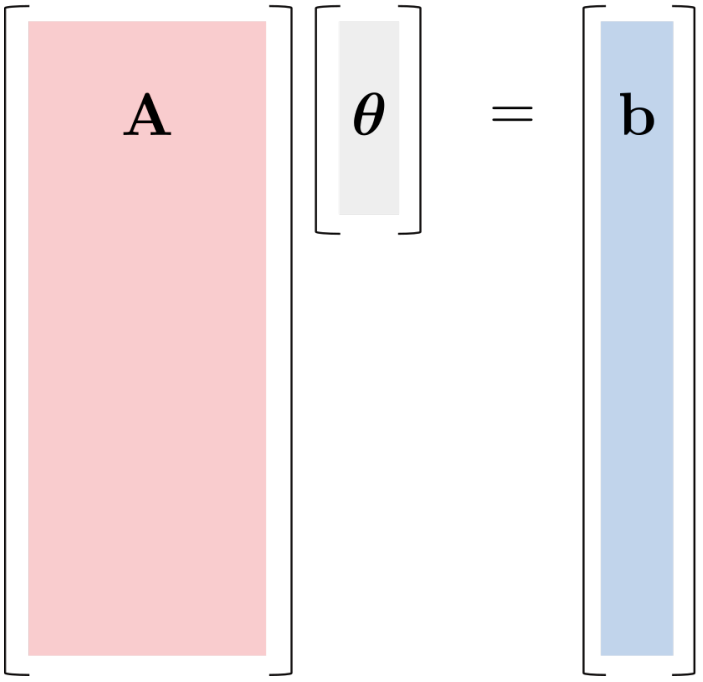
Regression (linear/nonlinear):

$$\underbrace{\mathbf{A}}_{\mathbf{X}^{\mathrm{T}}}\boldsymbol{\theta} = \underbrace{\mathbf{b}}_{\mathbf{y}^{\mathrm{T}}}$$

$n \times m \qquad n \times p$

∄ solution

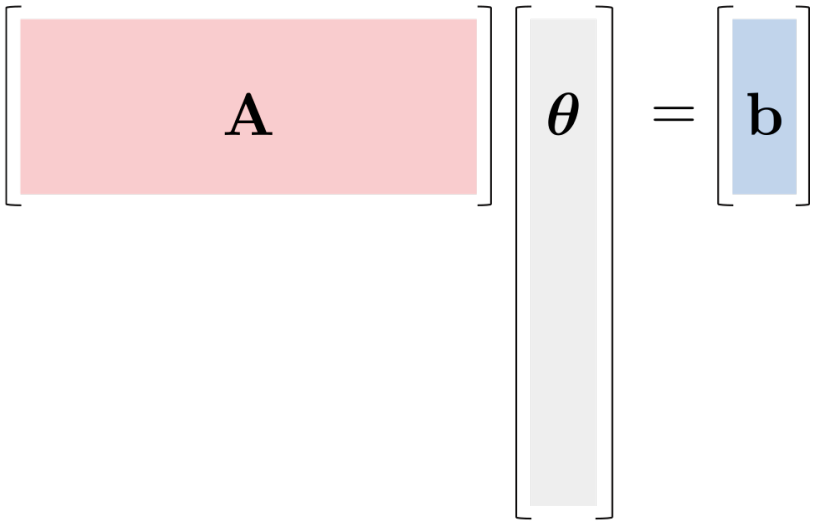
overdetermined sys: $n > m$



(regression)

∞ solutions

underdetermined sys: $n < m$



(overparametrized neural nets)

Over- and under-determined systems

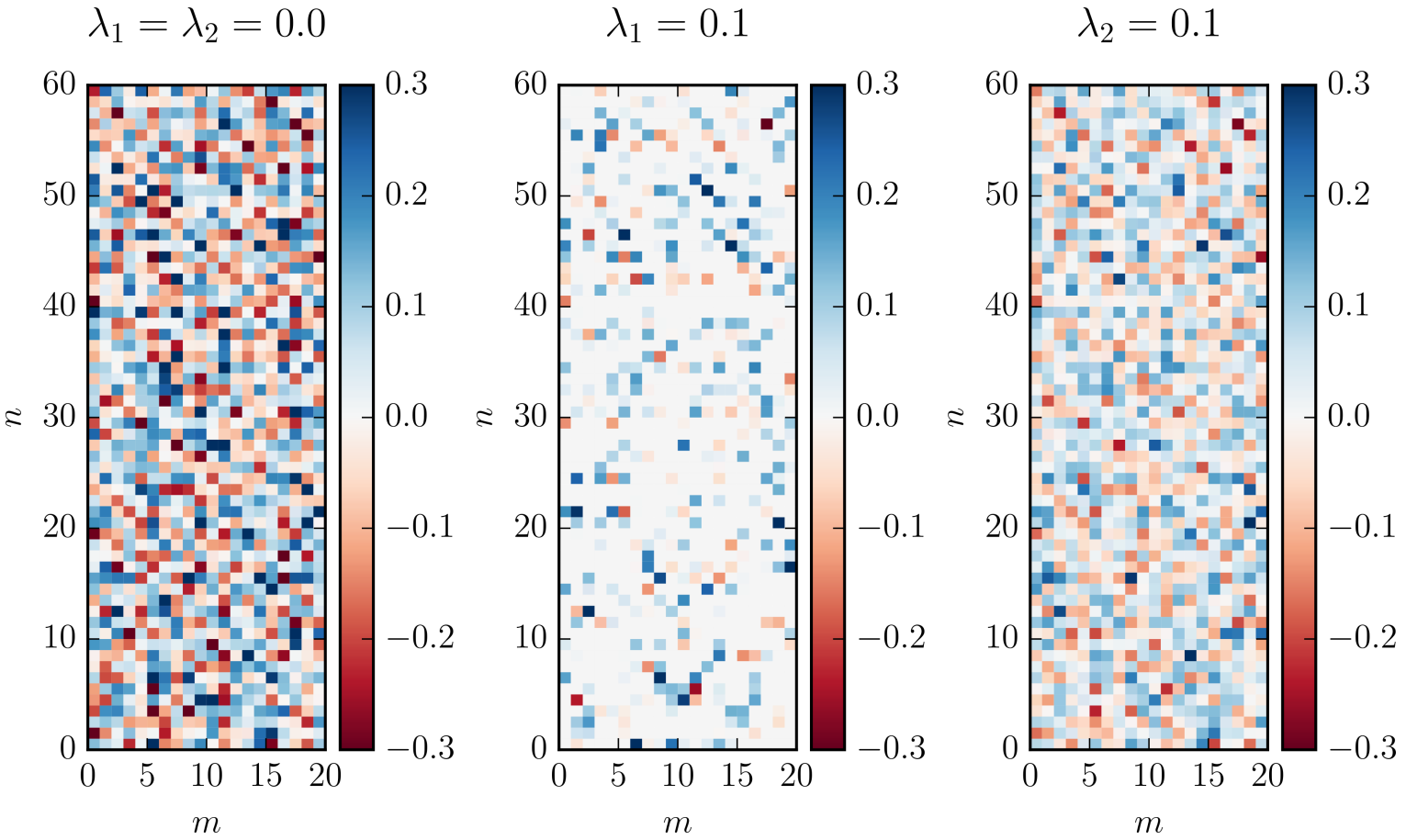
Overdetermined systems: penalty

Underdetermined systems: constraints

$$\underbrace{\mathbf{A}}_{\mathbf{X}^{\mathrm{T}}}\boldsymbol{\theta} = \underbrace{\mathbf{b}}_{\mathbf{y}^{\mathrm{T}}}$$

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left(\operatorname{MSE}(\mathbf{A}\boldsymbol{\theta} - \mathbf{b}) + \lambda_1 ||\boldsymbol{\theta}||_1 + \lambda_2 ||\boldsymbol{\theta}||_2 \right)$$

$\hat{\boldsymbol{\theta}} =$



Regularized regression

Regularized regression

LASSO (Least Absolute Shrinkage and Selection Operator) regression

$$\mathcal{L}^{\text{LASSO}}(\mathbf{X}, f_{\boldsymbol{\theta}}) = \text{MSE}(\mathbf{X}, f_{\boldsymbol{\theta}}) + \lambda_1 ||\boldsymbol{\theta}||_1$$

only GD

non differentiable at $\boldsymbol{\theta} = \mathbf{0}$: $\text{grad}\left(\boldsymbol{\theta}, \mathcal{L}^{\text{LASSO}}\right) \mapsto \frac{\partial}{\partial \boldsymbol{\theta}} \text{MSE}(\mathbf{X}, f_{\boldsymbol{\theta}}) + \text{sign}(\boldsymbol{\theta})$

Regularized regression

LASSO (Least Absolute Shrinkage and Selection Operator) regression

$$\mathcal{L}^{\text{LASSO}}(\mathbf{X}, f_{\boldsymbol{\theta}}) = \text{MSE}(\mathbf{X}, f_{\boldsymbol{\theta}}) + \lambda_1 ||\boldsymbol{\theta}||_1$$

only GD

non differentiable at $\boldsymbol{\theta} = \mathbf{0}$: $\text{grad}(\boldsymbol{\theta}, \mathcal{L}^{\text{LASSO}}) \mapsto \frac{\partial}{\partial \boldsymbol{\theta}} \text{MSE}(\mathbf{X}, f_{\boldsymbol{\theta}}) + \text{sign}(\boldsymbol{\theta})$

Ridge regression

$$\mathcal{L}^{\text{Ridge}}(\mathbf{X}, f_{\boldsymbol{\theta}}) = \text{MSE}(\mathbf{X}, f_{\boldsymbol{\theta}}) + \lambda_2 ||\boldsymbol{\theta}||_2^2$$

normal equation or GD

Regularized regression

LASSO (Least Absolute Shrinkage and Selection Operator) regression

$$\mathcal{L}^{\text{LASSO}}(\mathbf{X}, f_{\boldsymbol{\theta}}) = \text{MSE}(\mathbf{X}, f_{\boldsymbol{\theta}}) + \lambda_1 ||\boldsymbol{\theta}||_1$$

only GD

non differentiable at $\boldsymbol{\theta} = \mathbf{0}$: $\text{grad}\left(\boldsymbol{\theta}, \mathcal{L}^{\text{LASSO}}\right) \mapsto \frac{\partial}{\partial \boldsymbol{\theta}} \text{MSE}(\mathbf{X}, f_{\boldsymbol{\theta}}) + \text{sign}(\boldsymbol{\theta})$

Ridge regression

$$\mathcal{L}^{\text{Ridge}}(\mathbf{X}, f_{\boldsymbol{\theta}}) = \text{MSE}(\mathbf{X}, f_{\boldsymbol{\theta}}) + \lambda_2 ||\boldsymbol{\theta}||_2^2$$

normal equation or GD

Elastic net regression

$$\mathcal{L}^{\text{Elastic net}}(\mathbf{X}, f_{\boldsymbol{\theta}}) = \text{MSE}(\mathbf{X}, f_{\boldsymbol{\theta}}) + \varrho\lambda ||\boldsymbol{\theta}||_1 + \frac{1-\varrho}{\varrho}\lambda ||\boldsymbol{\theta}||_2^2$$

Regularized regression

LASSO (Least Absolute Shrinkage and Selection Operator) regression

$$\mathcal{L}^{\text{LASSO}}(\mathbf{X}, f_{\boldsymbol{\theta}}) = \text{MSE}(\mathbf{X}, f_{\boldsymbol{\theta}}) + \lambda_1 ||\boldsymbol{\theta}||_1$$

only GD

non differentiable at $\boldsymbol{\theta} = \mathbf{0}$: $\text{grad}(\boldsymbol{\theta}, \mathcal{L}^{\text{LASSO}}) \mapsto \frac{\partial}{\partial \boldsymbol{\theta}} \text{MSE}(\mathbf{X}, f_{\boldsymbol{\theta}}) + \text{sign}(\boldsymbol{\theta})$

Ridge regression

$$\mathcal{L}^{\text{Ridge}}(\mathbf{X}, f_{\boldsymbol{\theta}}) = \text{MSE}(\mathbf{X}, f_{\boldsymbol{\theta}}) + \lambda_2 ||\boldsymbol{\theta}||_2^2$$

normal equation or GD

Elastic net regression

$$\mathcal{L}^{\text{Elastic net}}(\mathbf{X}, f_{\boldsymbol{\theta}}) = \text{MSE}(\mathbf{X}, f_{\boldsymbol{\theta}}) + \varrho\lambda ||\boldsymbol{\theta}||_1 + \frac{1-\varrho}{\varrho}\lambda ||\boldsymbol{\theta}||_2^2$$

Implementation

```
from sklearn.linear_model import Lasso, Ridge, ElasticNet
lambda_1 = 0.0001
lasso_reg = Lasso(alpha=lambda_1,max_iter=100000)
poly_features = PolynomialFeatures(degree=100, include_bias=False)
lasso_reg.fit(norm_X_poly,norm_y_train) # fit LASSO reg model

ridge_reg = Ridge(alpha=lambda_1,max_iter=100000)
ridge_reg.fit(norm_X_poly,norm_y_train) # fit Ridge reg model

net_reg = ElasticNet(alpha=2*lambda_2,l1_ratio=1.0, max_iter=100000) #l1_ratio = \varrho
net_reg.fit(norm_X_poly,norm_y_train) # fit Elastic Net reg mode
```

Regularized regression

LASSO (Least Absolute Shrinkage and Selection Operator) regression

$$\mathcal{L}^{\text{LASSO}}(\mathbf{X}, f_{\boldsymbol{\theta}}) = \text{MSE}(\mathbf{X}, f_{\boldsymbol{\theta}}) + \lambda_1 ||\boldsymbol{\theta}||_1$$

only GD

non differentiable at $\boldsymbol{\theta} = \mathbf{0}$: $\text{grad}(\boldsymbol{\theta}, \mathcal{L}^{\text{LASSO}}) \mapsto \frac{\partial}{\partial \boldsymbol{\theta}} \text{MSE}(\mathbf{X}, f_{\boldsymbol{\theta}}) + \text{sign}(\boldsymbol{\theta})$

Ridge regression

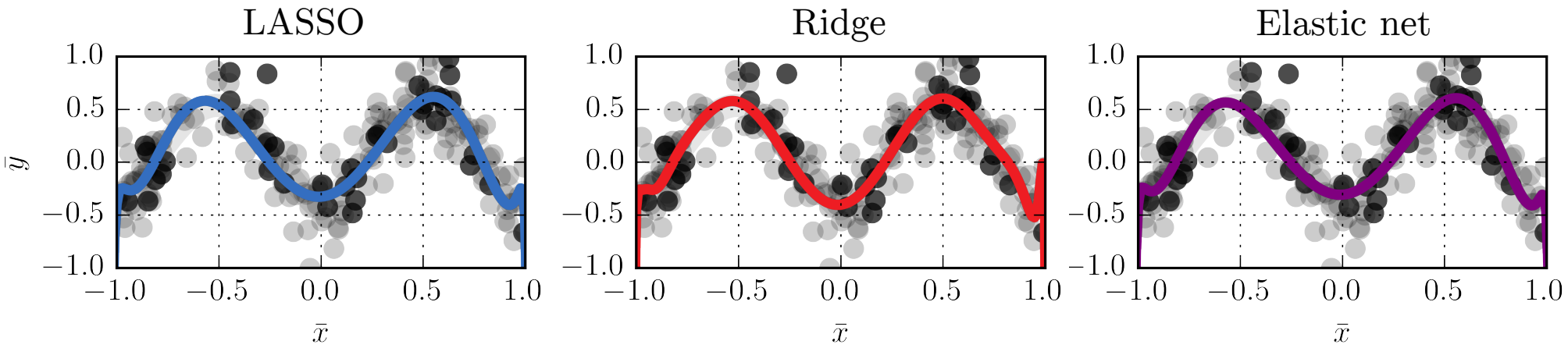
$$\mathcal{L}^{\text{Ridge}}(\mathbf{X}, f_{\boldsymbol{\theta}}) = \text{MSE}(\mathbf{X}, f_{\boldsymbol{\theta}}) + \lambda_2 ||\boldsymbol{\theta}||_2^2$$

normal equation or GD

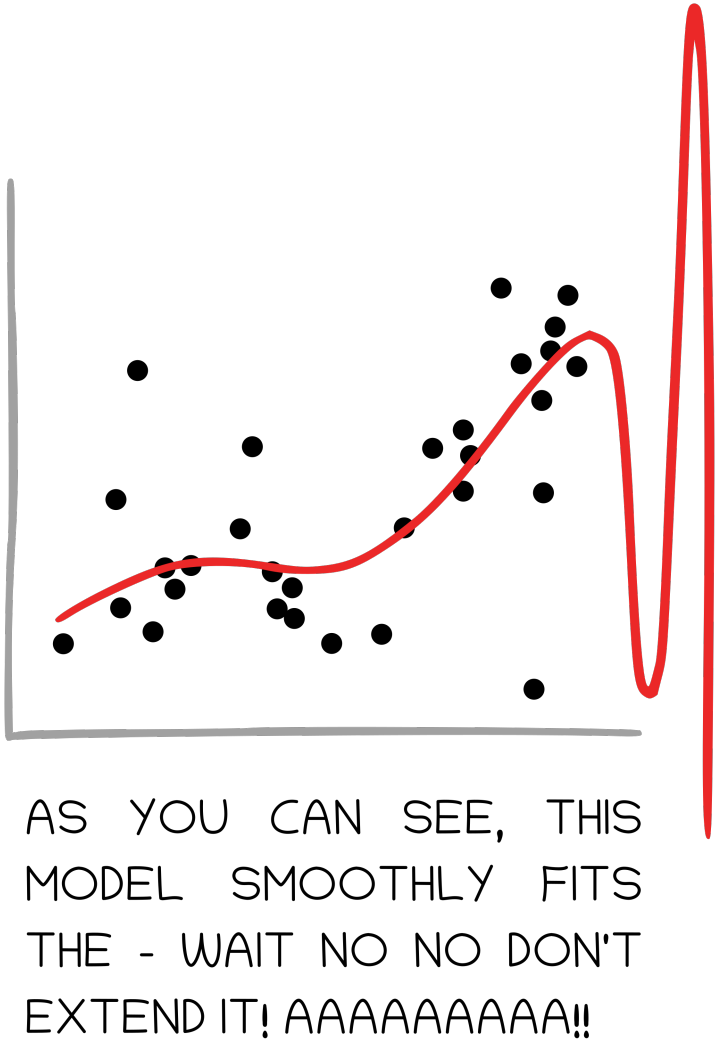
Elastic net regression

$$\mathcal{L}^{\text{Elastic net}}(\mathbf{X}, f_{\boldsymbol{\theta}}) = \text{MSE}(\mathbf{X}, f_{\boldsymbol{\theta}}) + \varrho\lambda ||\boldsymbol{\theta}||_1 + \frac{1-\varrho}{\varrho}\lambda ||\boldsymbol{\theta}||_2^2$$

Implementation



Interpretable models and *where* to find them



Interpretable models

Interpretable models are capable of explaining why data behaves in a certain manner.

lex parsimoniae

"Entia non sunt multiplicanda praeter necessitatem."

"Entities must not be multiplied beyond necessity."

Interpretable models

Interpretable models are capable of explaining why data behaves in a certain manner.

lex parsimoniae

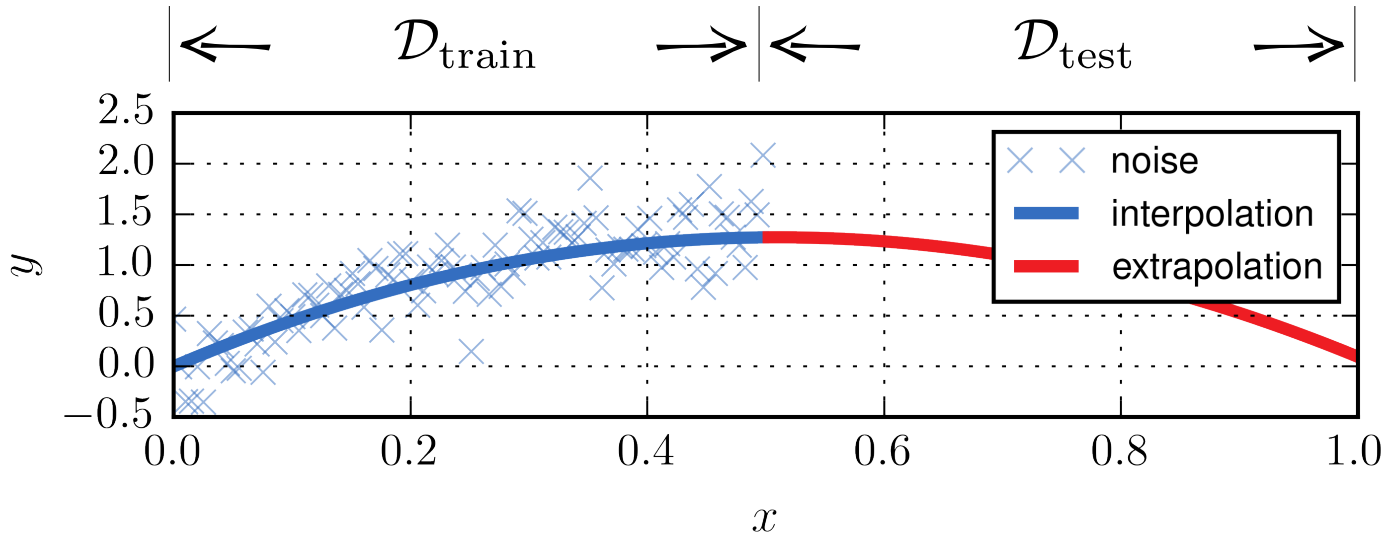
"Entia non sunt multiplicanda praeter necessitatem."

"Entities must not be multiplied beyond necessity."

Where to find them

Discover interpretable models (governing equations): ballistic!

$$\mathrm{ODESolve}\Big(\ddot{z}(t) + g = 0\Big) \implies z(t) = z^{(0)} + \dot{z}^{(0)}t - \frac{1}{2}gt^2$$

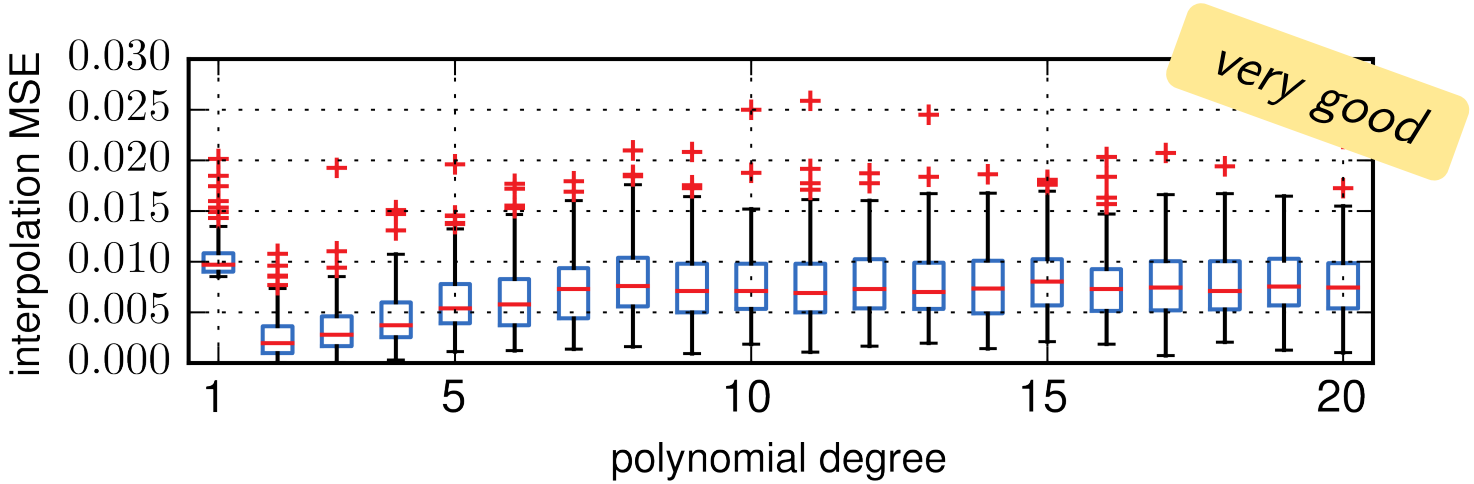


Where to find them

Discover interpretable models (governing equations): ballistic!

$$\text{ODESolve}\Big(\ddot{z}(t) + g = 0\Big) \implies z(t) = z^{(0)} + \dot{z}^{(0)}t - \frac{1}{2}gt^2$$

interpolation

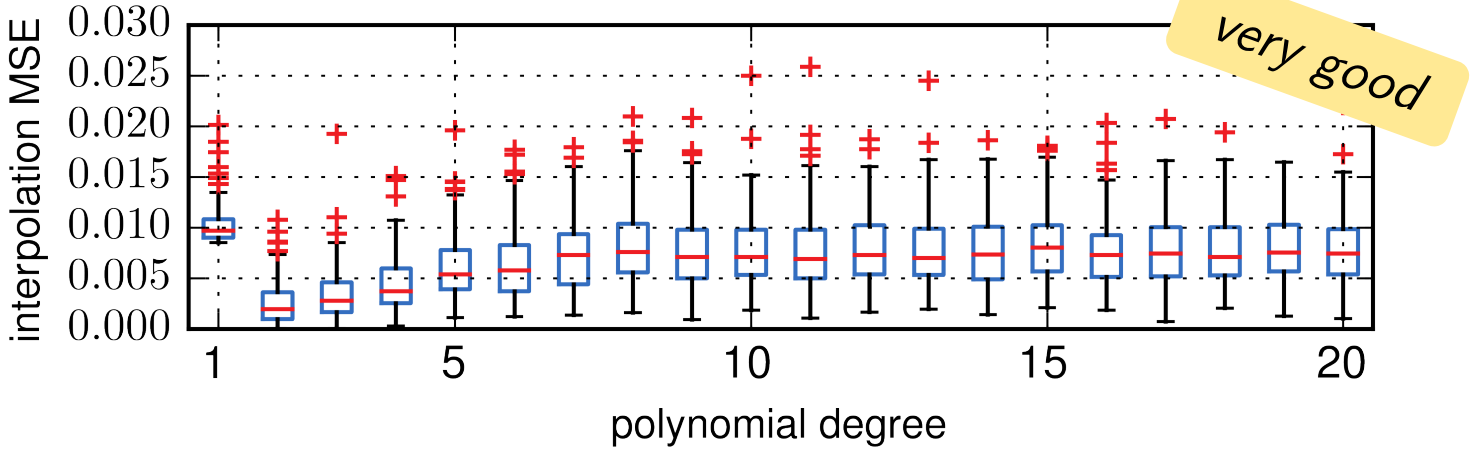


Where to find them

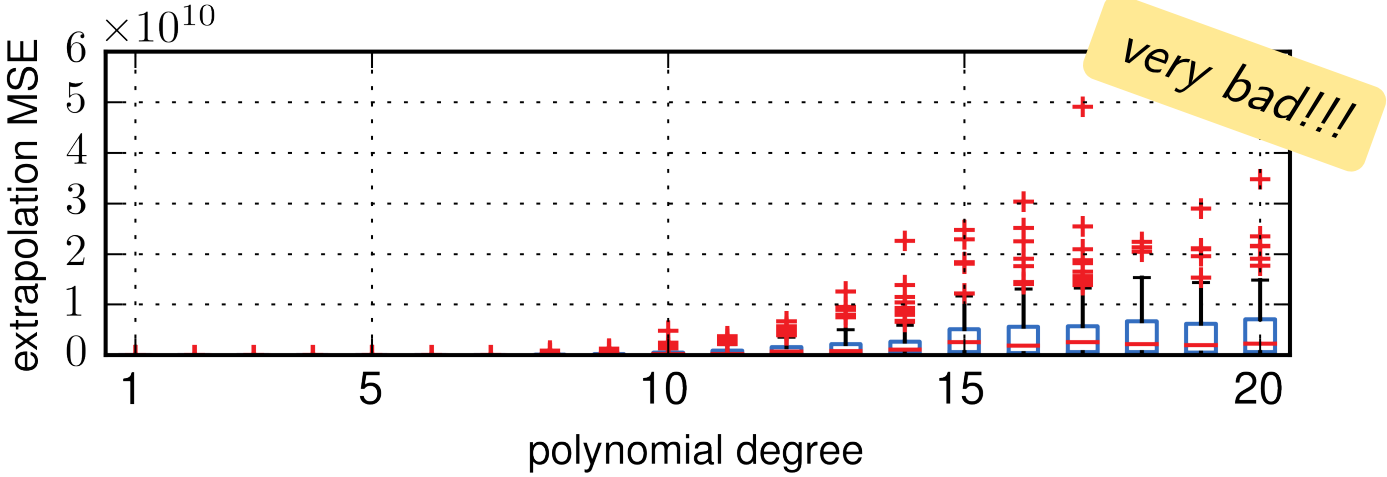
Discover interpretable models (governing equations): ballistic!

$$\text{ODESolve}\Big(\ddot{z}(t) + g = 0\Big) \implies z(t) = z^{(0)} + \dot{z}^{(0)}t - \frac{1}{2}gt^2$$

interpolation



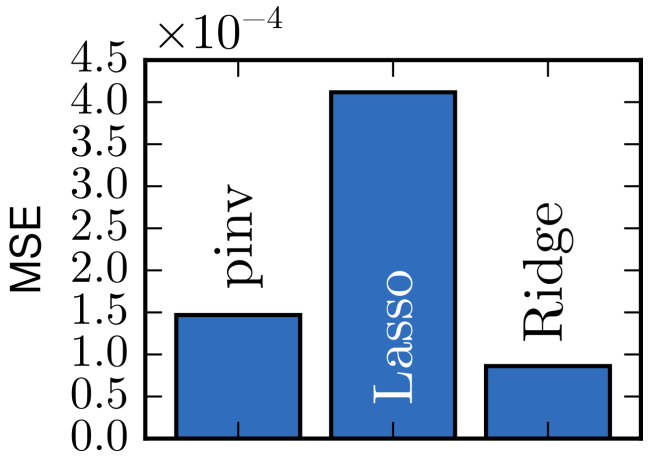
extrapolation



Where to find them

Overfitting: k-fold cross validation ($k = 100$)

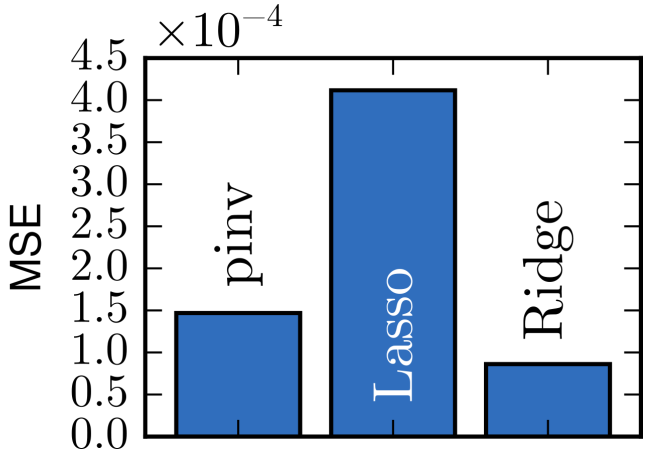
interpolation



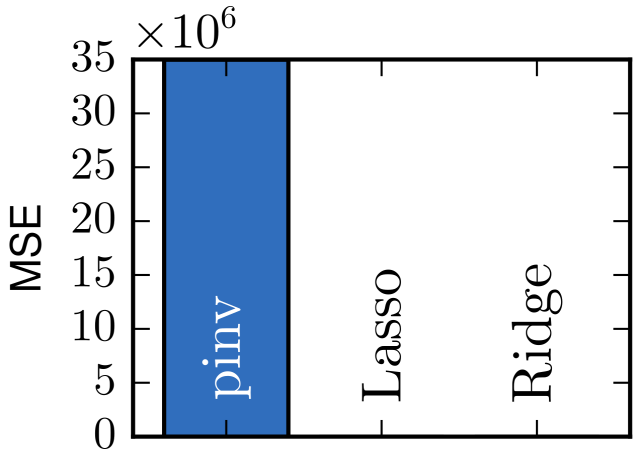
Where to find them

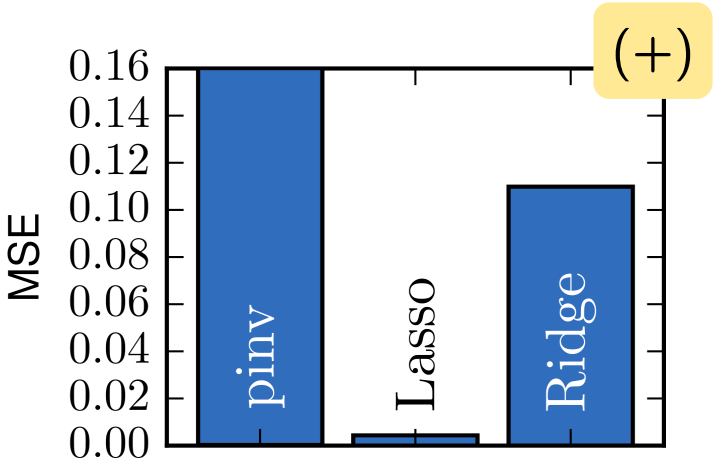
Overfitting: k-fold cross validation ($k = 100$)

interpolation



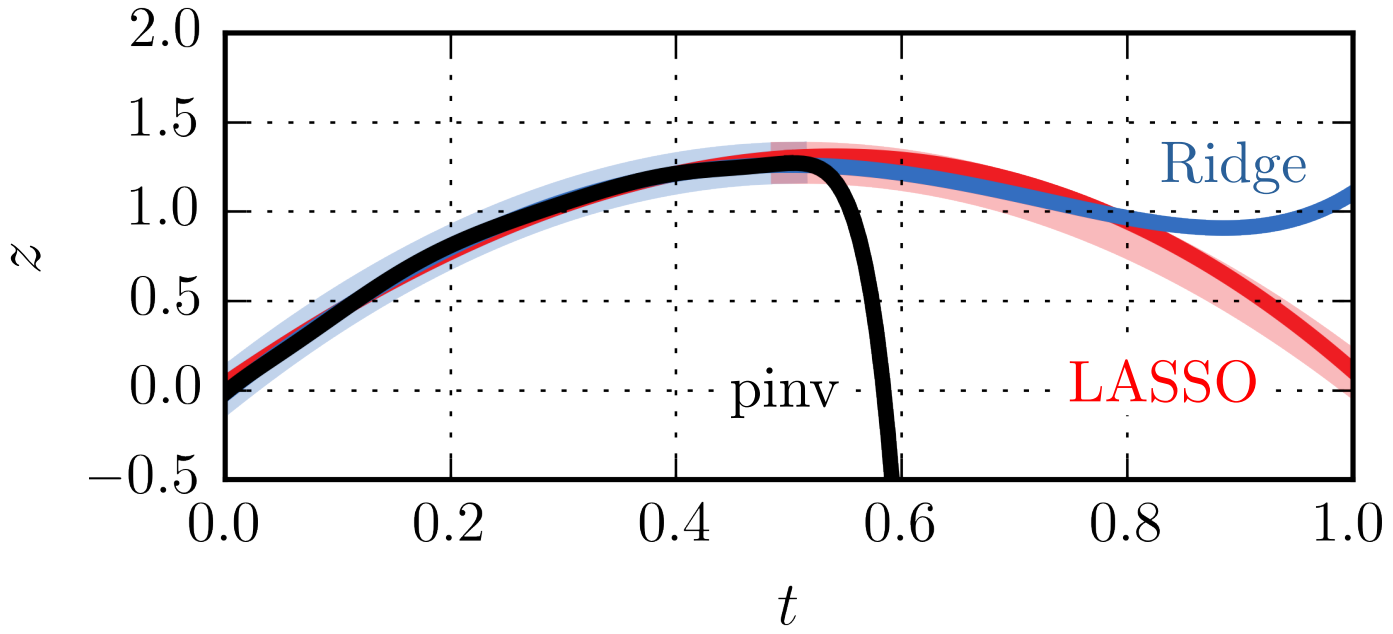
extrapolation





Where to find them

LASSO ≫ Ridge ≫ pinv

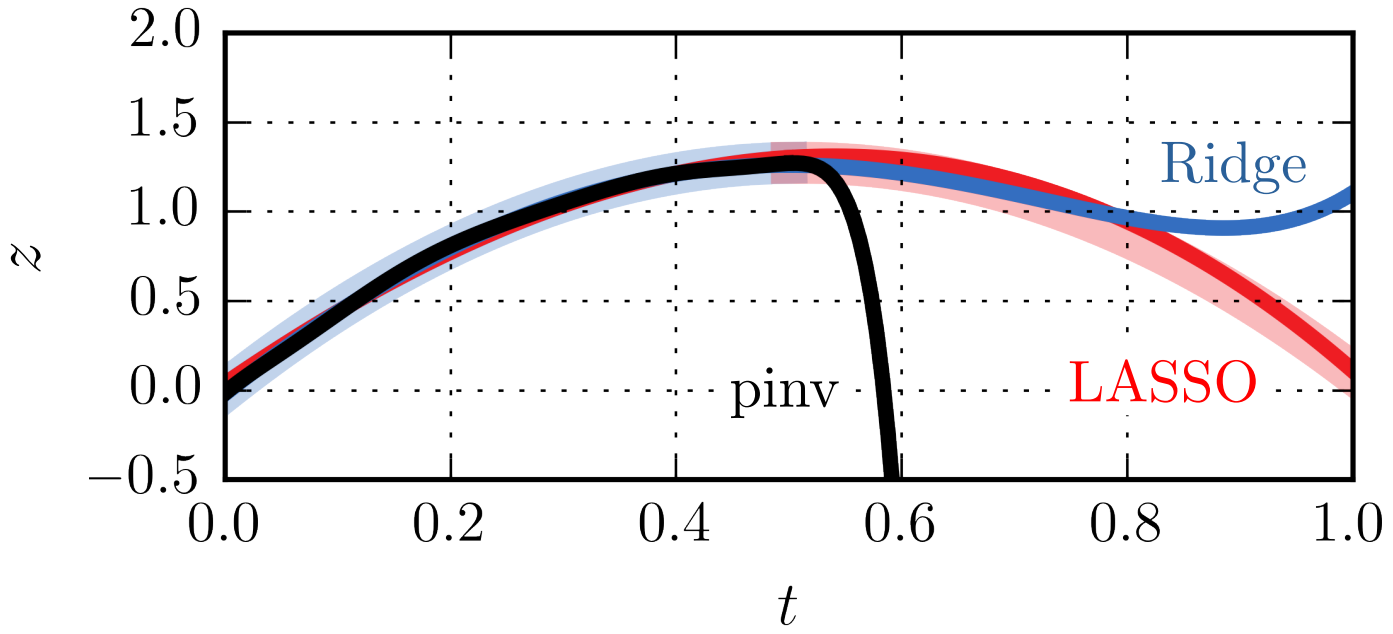


$$\text{Physical model: } z(t) = 5.0t - 4.9t^2$$

$$\text{LASSO model: } \hat{z}(t) = 4.3t - 2.9t^2 + 1.3t^3$$

Where to find them

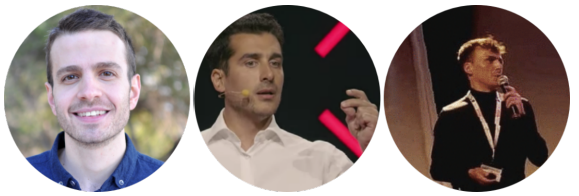
LASSO ≫ Ridge ≫ pinv



$$\text{Physical model: } z(t) = 5.0t - 4.9t^2$$

$$\text{LASSO model: } \hat{z}(t) = 4.3t - 2.9t^2 + 1.3t^3$$

... wait for the next lectures



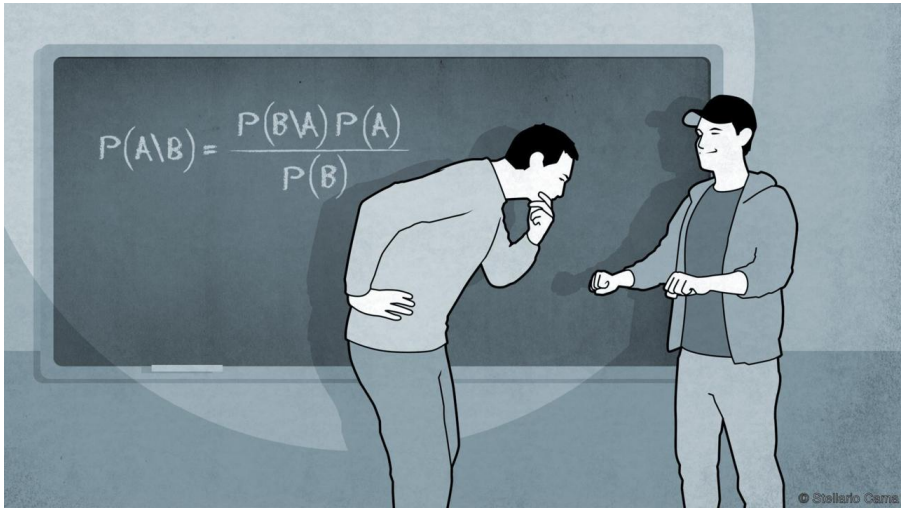
Bayesian regression

Bayesian versus *frequentist*

All a matter of how we interpret probability!

The Frequentist

- Frequency of occurrence of an event
- There are "true" values of the parameters
- Compute point estimates of the model parameters



Bayesian versus *frequentist*

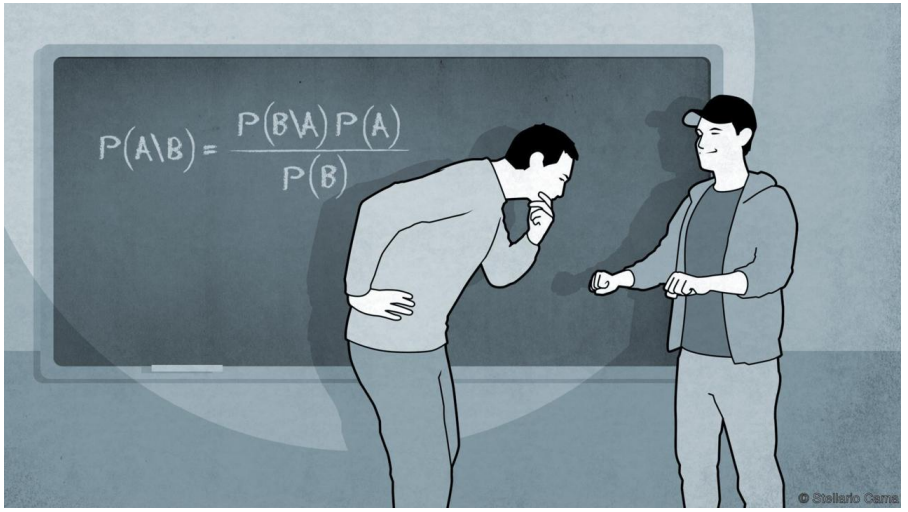
All a matter of how we interpret probability!

The Bayesian

- Degree of belief on the occurrence of an event
- No true values of parameters, just more probable values
- Compute model parameters as inferred probability distributions

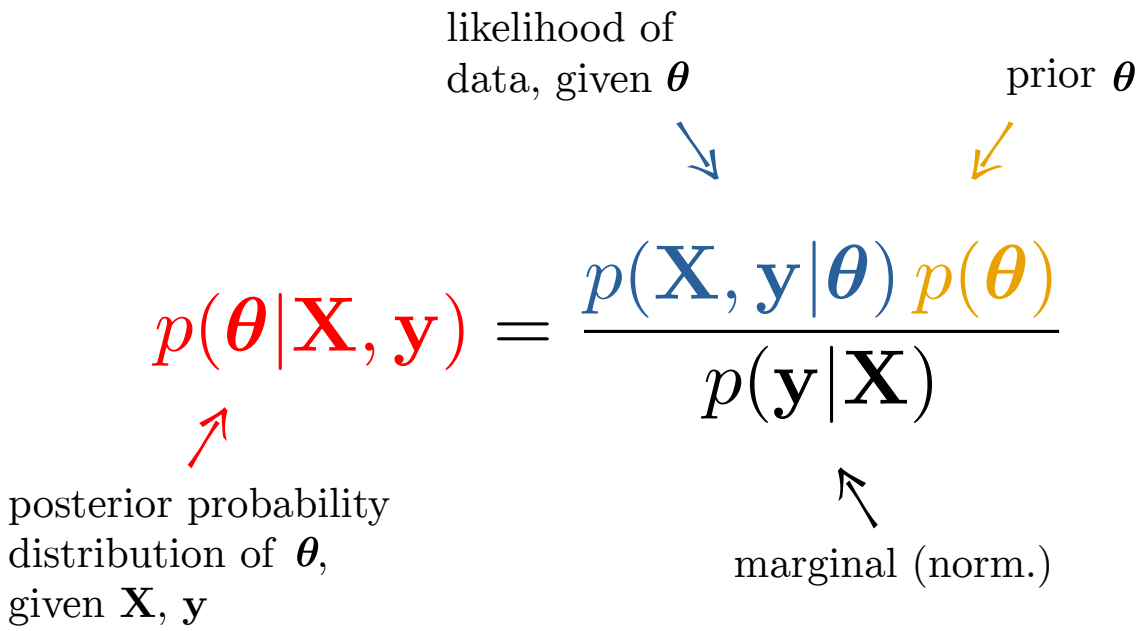
The Frequentist

- Frequency of occurrence of an event
- There are "true" values of the parameters
- Compute point estimates of the model parameters



Bayes theorem

The probability of an event, based on prior knowledge of conditions that might be related to the event



Maximum likelihood estimation (MLE)

Frequentist approach

$$\mathbf{y} = \boldsymbol{\theta}^{\mathrm{T}}\mathbf{X} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \backsim \mathcal{N}\left(\mathbf{0}, \sigma^2\mathbf{I}\right)$$

$$p(\mathbf{X}, \mathbf{y}|\boldsymbol{\theta}) = \prod_{k=1}^{n} \left(\frac{1}{\sigma\sqrt{2\pi}} \exp\left\{ -\left(\frac{y^{(k)} - \boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}^{(k)}}{\sqrt{2}\sigma} \right)^2 \right\} \right)$$

Maximum likelihood estimation (MLE)

Frequentist approach

$$\mathbf{y} = \boldsymbol{\theta}^{\mathrm{T}}\mathbf{X} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \backsim \mathcal{N}\left(\mathbf{0}, \sigma^2\mathbf{I}\right)$$

$$p(\mathbf{X}, \mathbf{y}|\boldsymbol{\theta}) = \prod_{k=1}^{n} \left(\frac{1}{\sigma\sqrt{2\pi}} \exp\left\{ -\left(\frac{y^{(k)} - \boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}^{(k)}}{\sqrt{2}\sigma} \right)^2 \right\} \right)$$

$$\begin{aligned}
\log p(\mathbf{X}, \mathbf{y}|\boldsymbol{\theta}) &= \log \prod_{k=1}^{n} p(\mathbf{x}^{(k)}, y^{(k)}|\boldsymbol{\theta}) \\
&= \sum_{k=1}^{n} \log p(\mathbf{x}^{(k)}, y^{(k)}|\boldsymbol{\theta}) \\
&= -\frac{n}{2}\log 2\pi - n\log\sigma - \frac{1}{2\sigma^2}\sum_{k=1}^{n}\left(y^{(k)} - \boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}^{(k)}\right)^2
\end{aligned}$$

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}}\Big(\log p(\mathbf{X}, \mathbf{y}|\boldsymbol{\theta}) \Big) = \arg\min_{\boldsymbol{\theta}} \left(\mathbf{y} - \boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}\right)^2$$

Maximum a posterior (MAP)

Semi-frequentist approach

$$\mathbf{y} = \boldsymbol{\theta}^{\mathrm{T}}\mathbf{X} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \backsim \mathcal{N}\left(\mathbf{0}, \sigma^2\mathbf{I}\right)$$

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}}\Big(p(\boldsymbol{\theta}|\mathbf{X},\mathbf{y})\Big) \propto \arg\max_{\boldsymbol{\theta}}\Big(p(\mathbf{X},\mathbf{y}|\boldsymbol{\theta})\,p(\boldsymbol{\theta})\Big)$$

$$\propto \arg\min_{\boldsymbol{\theta}}\left(\left(\frac{\mathbf{y}-\boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}}{\sigma}\right)^2 + \log\boldsymbol{\theta}\right)$$

Maximum a posterior (MAP)

Semi-frequentist approach

$$\mathbf{y} = \boldsymbol{\theta}^{\mathrm{T}}\mathbf{X} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \backsim \mathcal{N}\left(\mathbf{0}, \sigma^2\mathbf{I}\right)$$

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}}\Big(p(\boldsymbol{\theta}|\mathbf{X},\mathbf{y})\Big) \propto \arg\max_{\boldsymbol{\theta}}\Big(p(\mathbf{X},\mathbf{y}|\boldsymbol{\theta})\,p(\boldsymbol{\theta})\Big)$$

$$\propto \arg\min_{\boldsymbol{\theta}}\left(\left(\frac{\mathbf{y}-\boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}}{\sigma}\right)^2 + \log\boldsymbol{\theta}\right)$$

Prior probability distributions:

$$\text{if } \boldsymbol{\theta} \backsim \mathcal{N}\left(0, \frac{2}{\lambda}\mathbf{I}\right) \;\Rightarrow\; \hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}}\left(\left(\mathbf{y}-\boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}\right)^2 + \lambda\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\theta}\right)$$

Ridge

Maximum a posterior (MAP)

Semi-frequentist approach $\quad \mathbf{y} = \boldsymbol{\theta}^{\mathrm{T}}\mathbf{X} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \backsim \mathcal{N}\left(\mathbf{0}, \sigma^2\mathbf{I}\right)$

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}}\Big(p(\boldsymbol{\theta}|\mathbf{X},\mathbf{y})\Big) \propto \arg\max_{\boldsymbol{\theta}}\Big(p(\mathbf{X},\mathbf{y}|\boldsymbol{\theta})\,p(\boldsymbol{\theta})\Big)$$

$$\propto \arg\min_{\boldsymbol{\theta}}\left(\left(\frac{\mathbf{y}-\boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}}{\sigma}\right)^2 + \log\boldsymbol{\theta}\right)$$

Prior probability distributions:

$$\text{if } \boldsymbol{\theta} \backsim \mathcal{N}\left(0, \frac{2}{\lambda}\mathbf{I}\right) \Rightarrow \hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}}\left(\left(\mathbf{y}-\boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}\right)^2 + \lambda\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\theta}\right)$$

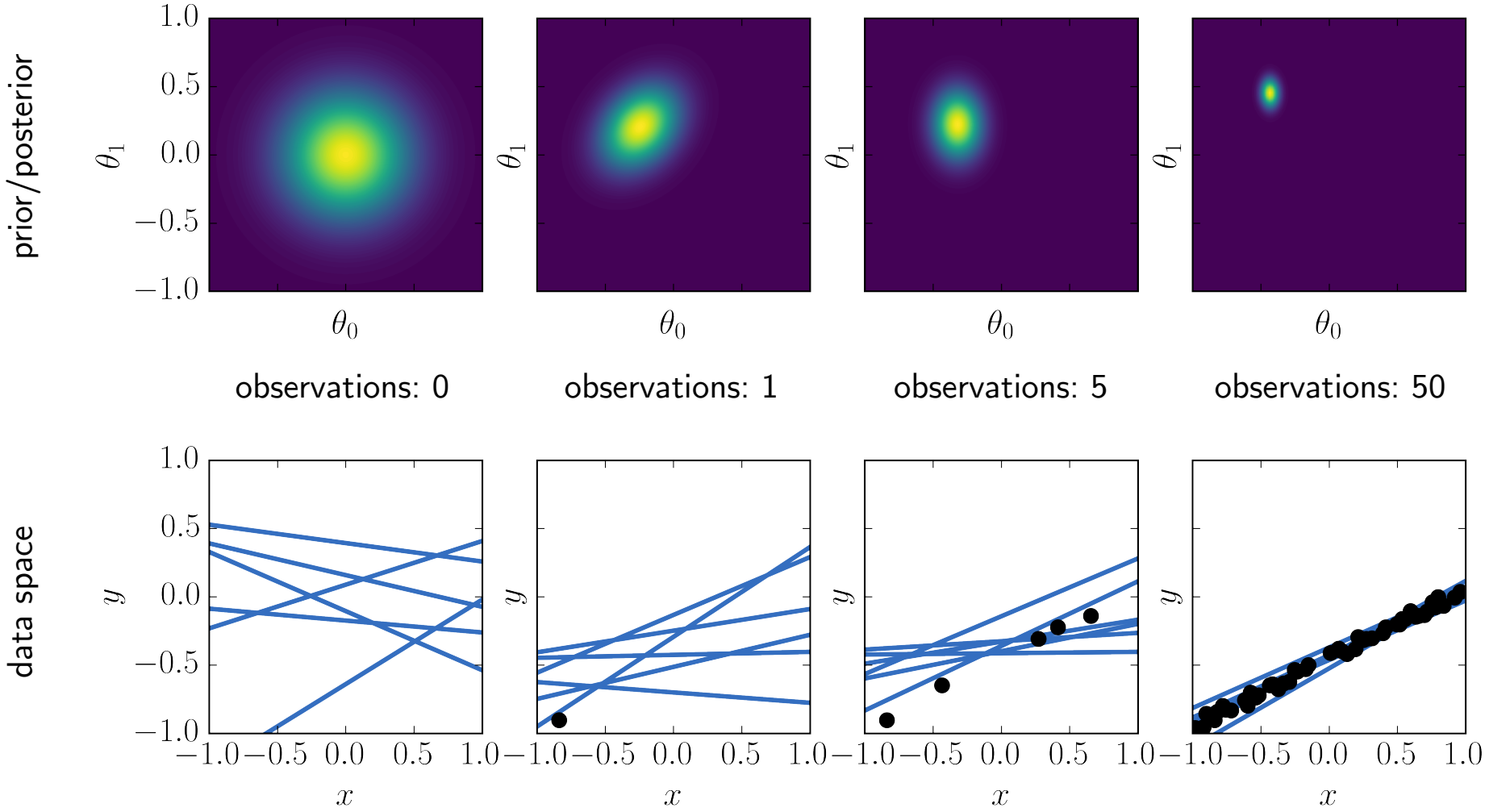
Ridge

$$\text{if } \boldsymbol{\theta} \backsim \mathcal{L}\left(0, \frac{\lambda}{2}\mathbf{I}\right) \Rightarrow \hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}}\left(\left(\mathbf{y}-\boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}\right)^2 + \lambda|\boldsymbol{\theta}|\right)$$

LASSO

Bayesian regression

Sampling from unknown posterior distribution given a prior



Bayesian inference involves tracing the full posterior of the parameters, given updated observations (cf. *Markov Chain Monte Carlo*)

Nonparametric models

Parametric versus non-parametric models

- hypothesis function $f_{\boldsymbol{\theta}}$ well-defined + $\boldsymbol{\theta} \in \mathbb{R}^m, m$ finite
- no predetermined $f_{\boldsymbol{\theta}}$ (need prior)

Nonparametric models

Parametric versus non-parametric models

- hypothesis function $f_{\boldsymbol{\theta}}$ well-defined + $\boldsymbol{\theta} \in \mathbb{R}^m, m$ finite
- no predetermined $f_{\boldsymbol{\theta}}$ (need prior)

def. Gaussian process (GP):

collection of random variables, any finite number of which has a Gaussian distribution

- characterized by a mean function: $\mu(\mathbf{x})$, and covariance function: $k(\mathbf{x}, \mathbf{x})$

Nonparametric models

Parametric versus non-parametric models

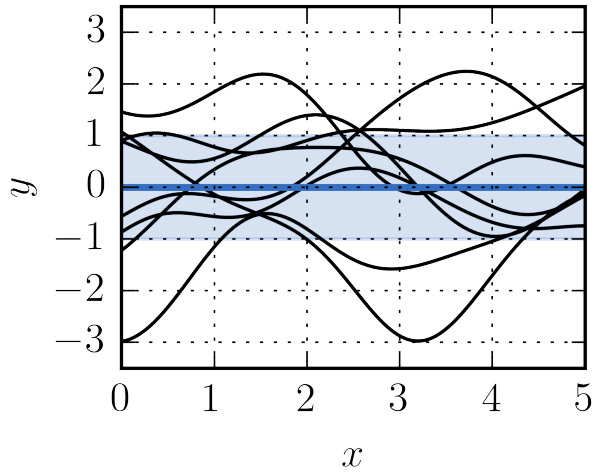
- hypothesis function $f_{\boldsymbol{\theta}}$ well-defined + $\boldsymbol{\theta} \in \mathbb{R}^m, m$ finite
- no predetermined $f_{\boldsymbol{\theta}}$ (need prior)

def. Gaussian process (GP):

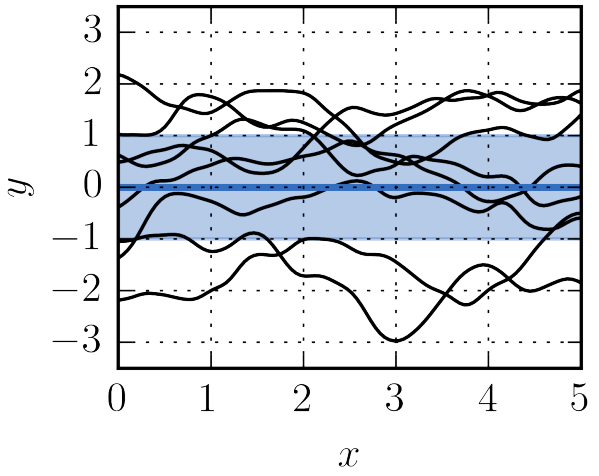
collection of random variables, any finite number of which has a Gaussian distribution

- characterized by a mean function: $\mu(\mathbf{x})$, and covariance function: $k(\mathbf{x}, \mathbf{x})$

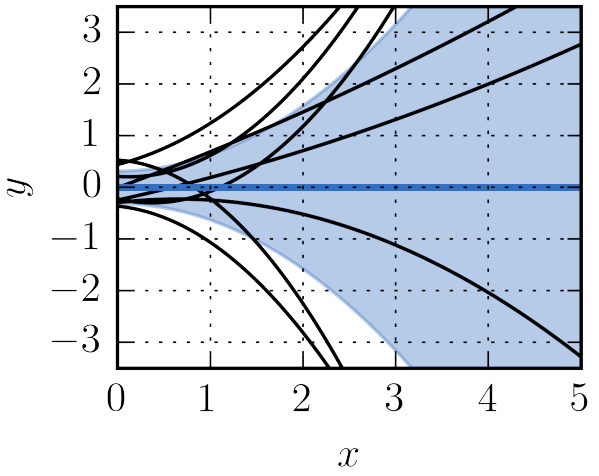
$$\exp\left(-\frac{d(\mathbf{x},\mathbf{x})^2}{2l^2}\right)$$



$$\left(1+\frac{d(\mathbf{x},\mathbf{x})^2}{2\alpha l^2}\right)^{-\alpha}$$



$$\sigma_0 + \mathbf{x}^{\mathrm{T}}\mathbf{x}$$



GP regression

Estimate the function $f(\mathbf{X})$ that minimizes $\mathbf{r} = \mathbf{y} - f(\mathbf{X})$ by placing a GP over it.

GP regression

Estimate the function $f(\mathbf{X})$ that minimizes $\mathbf{r} = \mathbf{y} - f(\mathbf{X})$ by placing a GP over it.

$$\mathbf{y} = f(\mathbf{X}) \sim \mathcal{N}(\mathbf{0}, \mathbf{K})$$

$$\mathbf{K} \equiv k(\mathbf{X}, \mathbf{X}) \quad \text{covariance}$$

kernel k!

Inference: Predict new $\mathbf{y}_*$, given new $\mathbf{x}_*$
that is, estimate the conditional distribution $p(\mathbf{y}_* | \mathbf{x}_*, \mathbf{X})$

$$f\left(\begin{bmatrix}\mathbf{x}\\ \mathbf{x}_*\end{bmatrix}\right) \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix}\mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^{\mathrm{T}} & \mathbf{K}_{**}\end{bmatrix}\right), \; \mathbf{K}_* \equiv k(\mathbf{X}, \mathbf{x}_*), \; \mathbf{K}_{**} \equiv k(\mathbf{x}_*, \mathbf{x}_*)$$

$$\Downarrow$$

$$\hat{\mathbf{y}}_* = f(\mathbf{x}_*) = \begin{cases} \mu\big(f(\mathbf{x}_*)\big) & = \mathbf{K}_*^{\mathrm{T}} \mathbf{K}^{-1} \mathbf{y}, \\ \mathrm{Cov}\big(f(\mathbf{x}_*)\big) & = \mathbf{K}_{**} - \mathbf{K}_* \mathbf{K}^{-1} \mathbf{K}_*^{\mathrm{T}} \end{cases}$$

GP regression with noise

Estimate the function $f(\mathbf{X})$ that minimizes $\mathbf{r} = \mathbf{y} - f(\mathbf{X})$ by placing a GP over it.

$$\mathbf{y} = f(\mathbf{X}) \sim \mathcal{N}(\mathbf{0}, \mathbf{K}) + \boldsymbol{\epsilon},\ \boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$$

$$\mathbf{K} \equiv k(\mathbf{X}, \mathbf{X}) \quad \text{covariance}$$

kernel k!

Inference: Predict new $\mathbf{y}_*$, given new $\mathbf{x}_*$
that is, estimate the conditional distribution $p(\mathbf{y}_* | \mathbf{x}_*, \mathbf{X})$

$$f\left(\begin{bmatrix}\mathbf{x}\\ \mathbf{x}_*\end{bmatrix}\right) \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix}\mathbf{K} + \sigma^2\mathbf{I} & \mathbf{K}_*\\ \mathbf{K}_*^{\mathrm{T}} & \mathbf{K}_{**}\end{bmatrix}\right),\ \mathbf{K}_* \equiv k(\mathbf{X}, \mathbf{x}_*),\ \mathbf{K}_{**} \equiv k(\mathbf{x}_*, \mathbf{x}_*)$$

$$\Downarrow$$

$$\hat{\mathbf{y}}_* = f(\mathbf{x}_*) = \begin{cases}\mu\big(f(\mathbf{x}_*)\big) & = \mathbf{K}_*^{\mathrm{T}} \left(\mathbf{K} + \sigma^2\mathbf{I}\right)^{-1} \mathbf{y},\\ \mathrm{Cov}\big(f(\mathbf{x}_*)\big) & = \mathbf{K}_{**} - \mathbf{K}_* \left(\mathbf{K} + \sigma^2\mathbf{I}\right)^{-1} \mathbf{K}_*^{\mathrm{T}} + \sigma^2\end{cases}$$

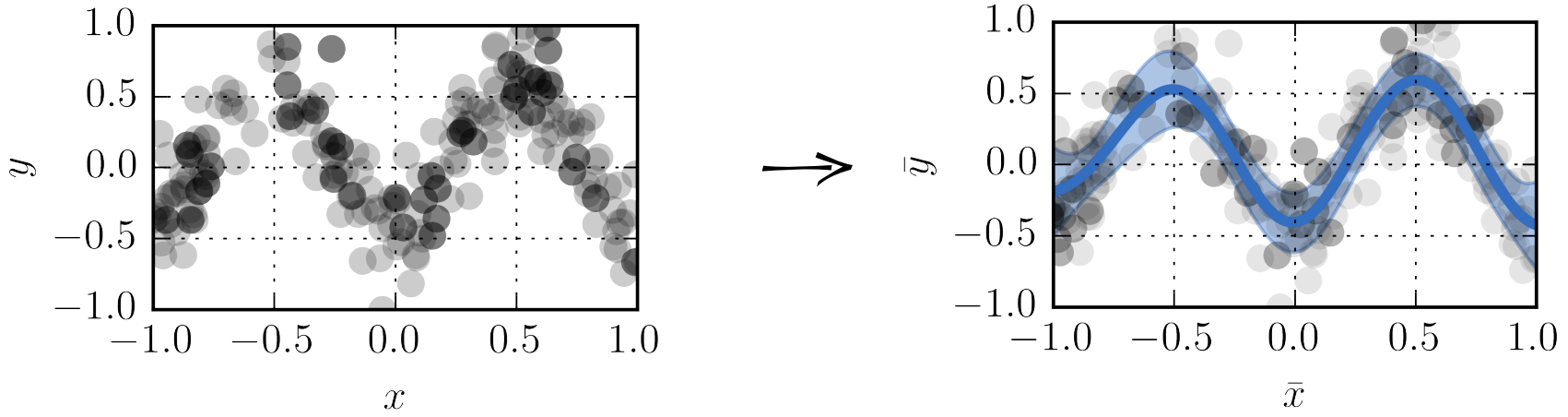
GP regression with noise

Example

```
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
# define a radial basis function kernel
kernel = 1 * RBF(length_scale=1.0, length_scale_bounds=(0.001, 10))
GPR = GaussianProcessRegressor(kernel=kernel,alpha=0.5**2) # noise term
GPR.fit(norm_X_train, norm_y_train) # find mean and covariance matrix
# posterior distribution
norm_y_mean, norm_y_std = GPR.predict(norm_x, return_std=True)
```

RBF kernel: $k(\mathbf{x}, \mathbf{x}) = \exp\left(-\frac{d(\mathbf{x}, \mathbf{x})^2}{2l^2}\right),\ l = 1.0$

noise term: $\sigma^2 = 0.5^2$



Tree-based regression methods

Decision trees

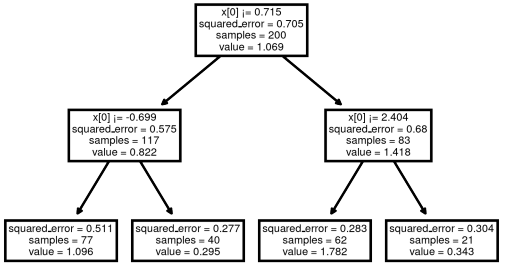
Non-parametric model: recursive partitioning of the feature space to group samples with similar targets

given training set $\{\mathbf{x}^{(k)}, y^{(k)}\}_{k=1}^{n}$

for each candidate split $\theta = \{i, t_l\}$ partition data Q_l into

$$Q_l^-(\theta) = \{(x, y) | x_i \leq t_l\}$$
$$Q_l^+(\theta) = Q_l / Q^-(\theta)$$

$n_l = 2$



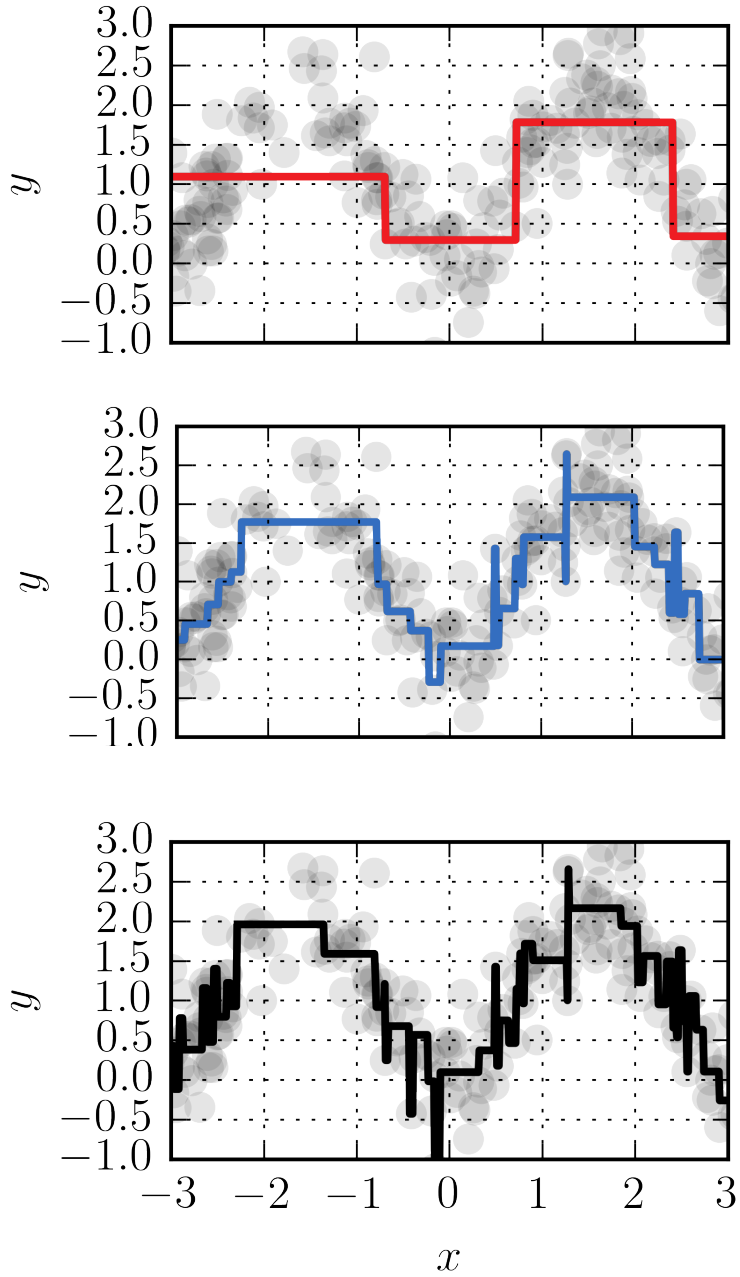
select parameters

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\Big(G(Q_l, \theta)\Big) = \underset{\theta}{\operatorname{argmin}}\left(\frac{n_l^-}{n_l} H\left(Q_l^-(\theta)\right) + \frac{n_l^+}{n_l} H\left(Q_l^+(\theta)\right)\right)$$

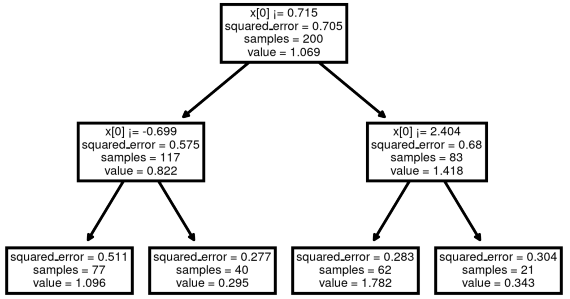
where $H(Q_l) \equiv \frac{1}{n_l} \sum_{y \in Q_l} \Big(y - \mu(y_l)\Big)^2$

$\backsim \mathcal{O}\Big(nm \log(n)\Big)$

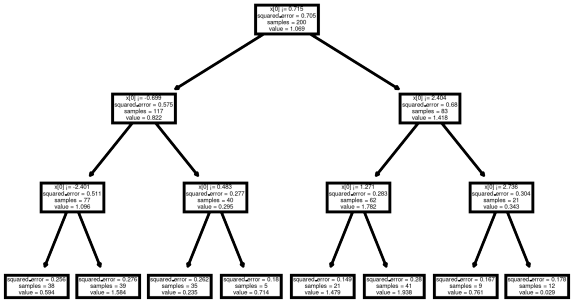
Decision trees



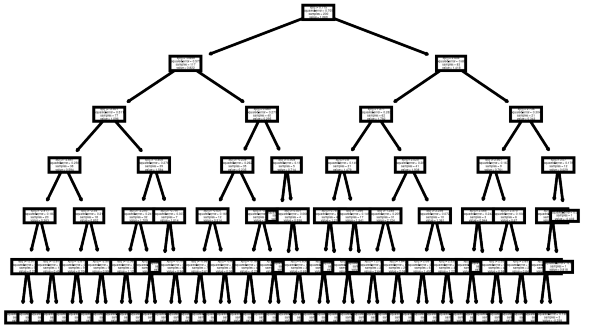
$n_l = 2$



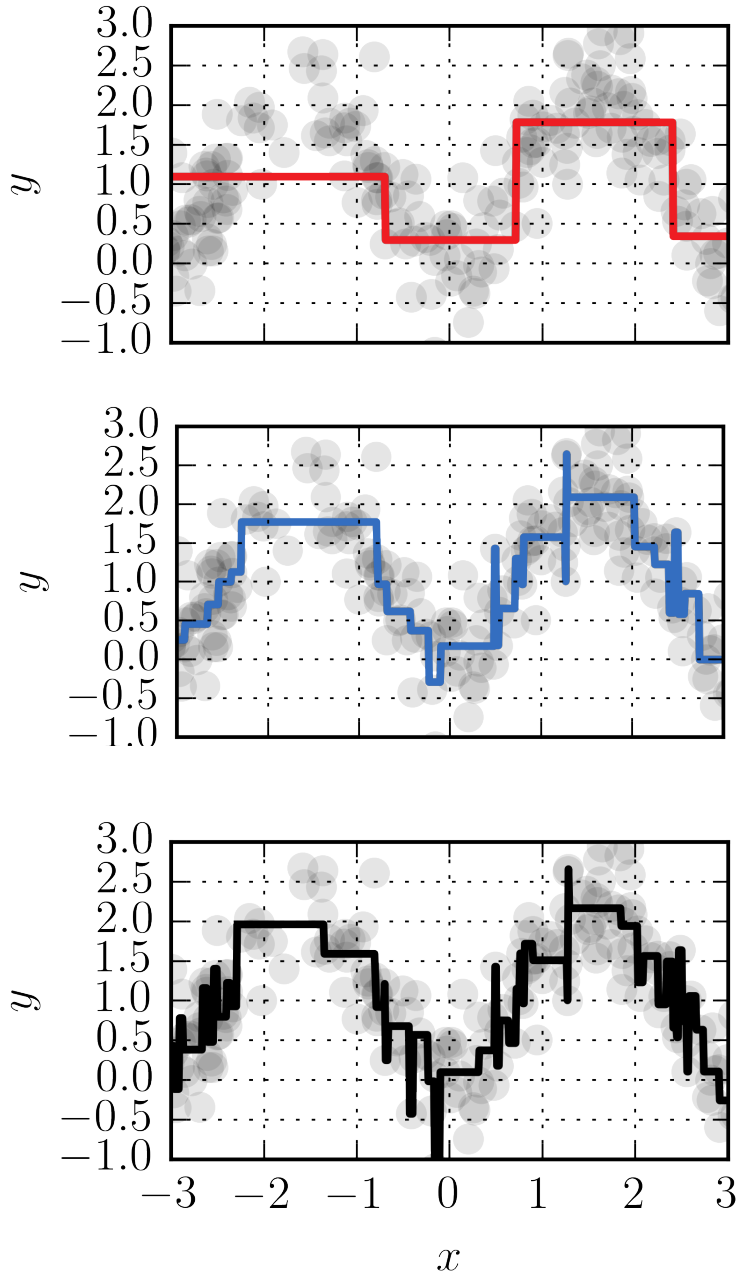
$n_l = 3$



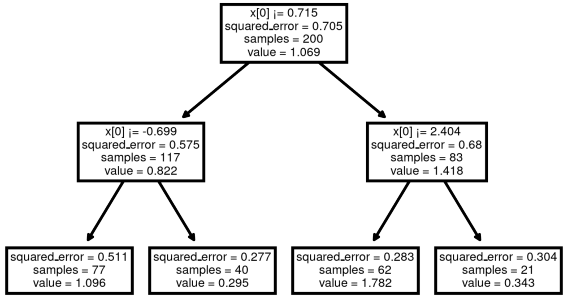
$n_l = 6$



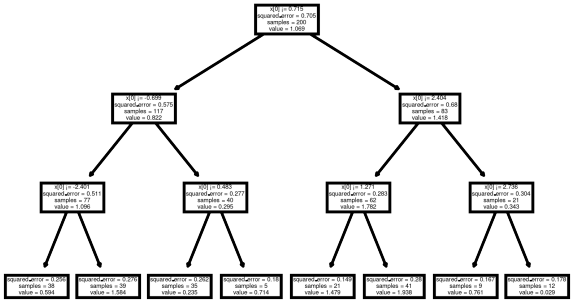
Decision trees



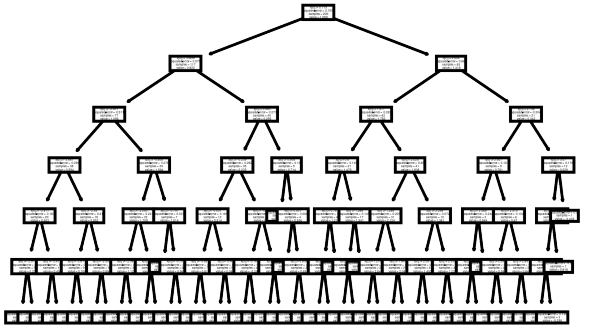
$n_l = 2$



$n_l = 3$



$n_l = 6$



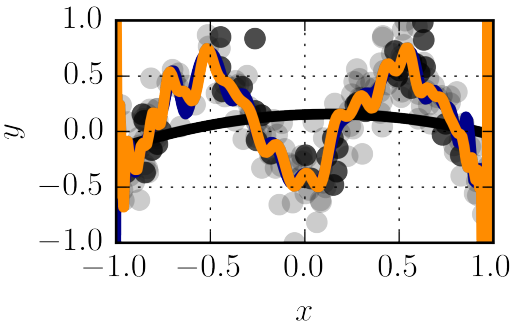
... support vector machines, random forests, etc.

In summary

In summary

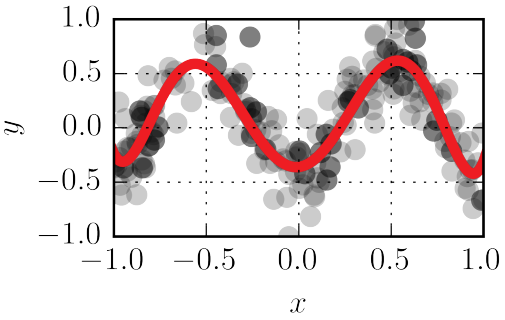
REGRESSION :

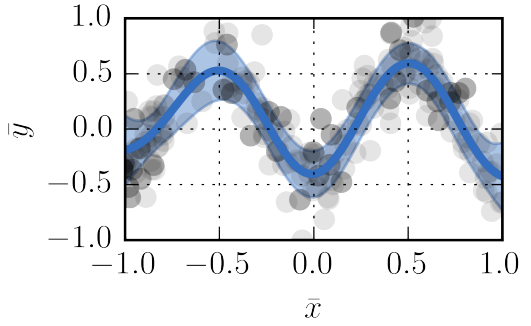
$$y = f_{\boldsymbol{\theta}}(\mathbf{x}) + \epsilon$$



FREQUENTIST APPROACH

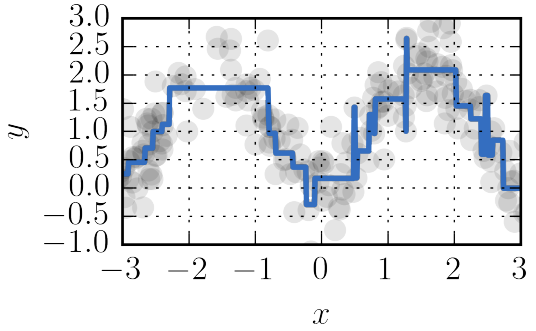
- Linear regression (normal equation/gradient descent)
- Nonlinear regression (polynomial, etc.)
- Regularized regression (LASSO, Ridge, Elastic net)
- Decision trees





BAYESIAN APPROACH

- Bayesian (non)linear regression
- Gaussian process regression (RBF kernel, etc.)

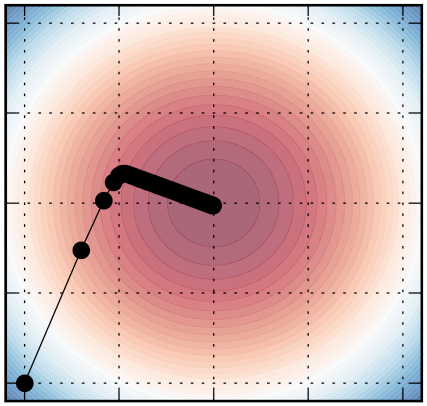


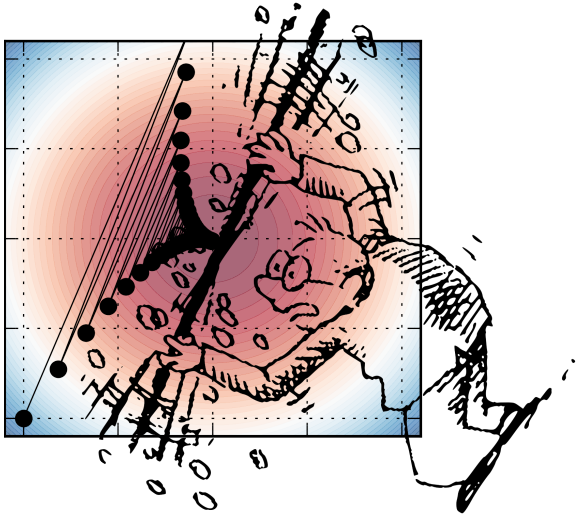
Doctoral School 2023

Machine Learning in Geomechanics

ALERT Geomaterials

Introduction to regression methods





Filippo Masi

The University of Sydney

THE UNIVERSITY OF SYDNEY