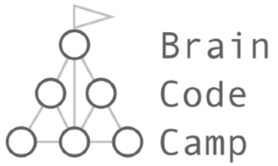


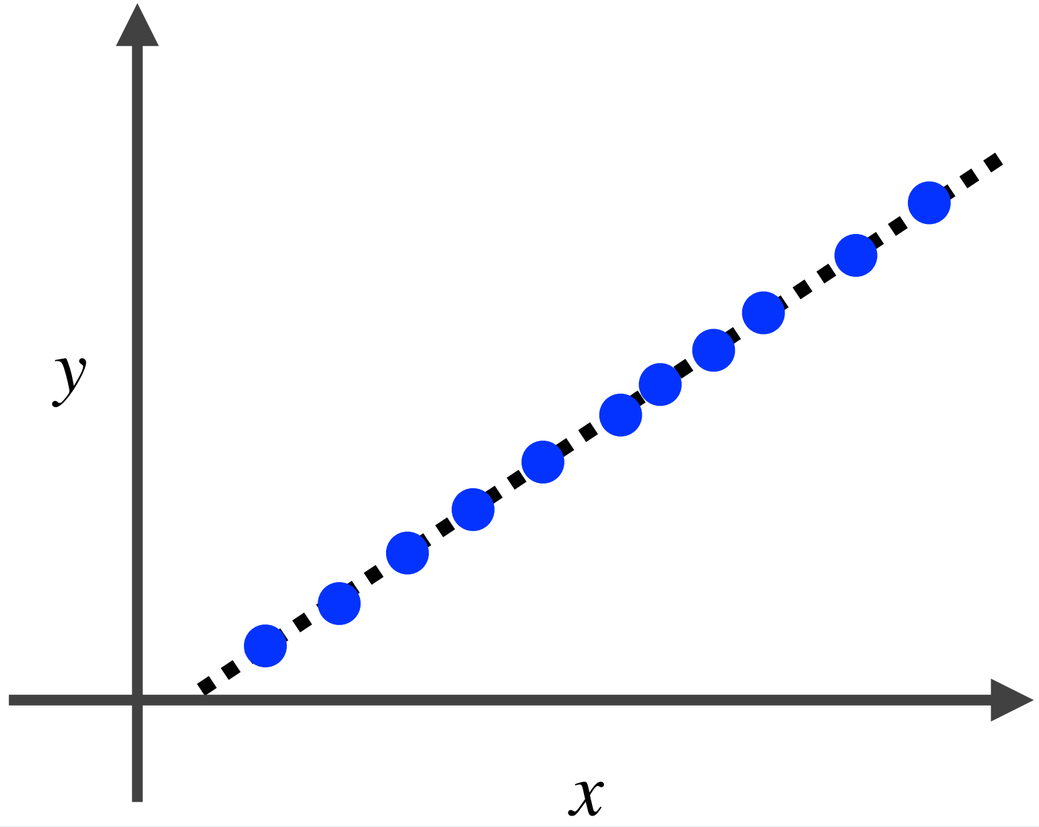

# Introduction to Modeling and Model Selection

Itthi Chatnuntawech

# Signal and Noise

## Underlying relationship

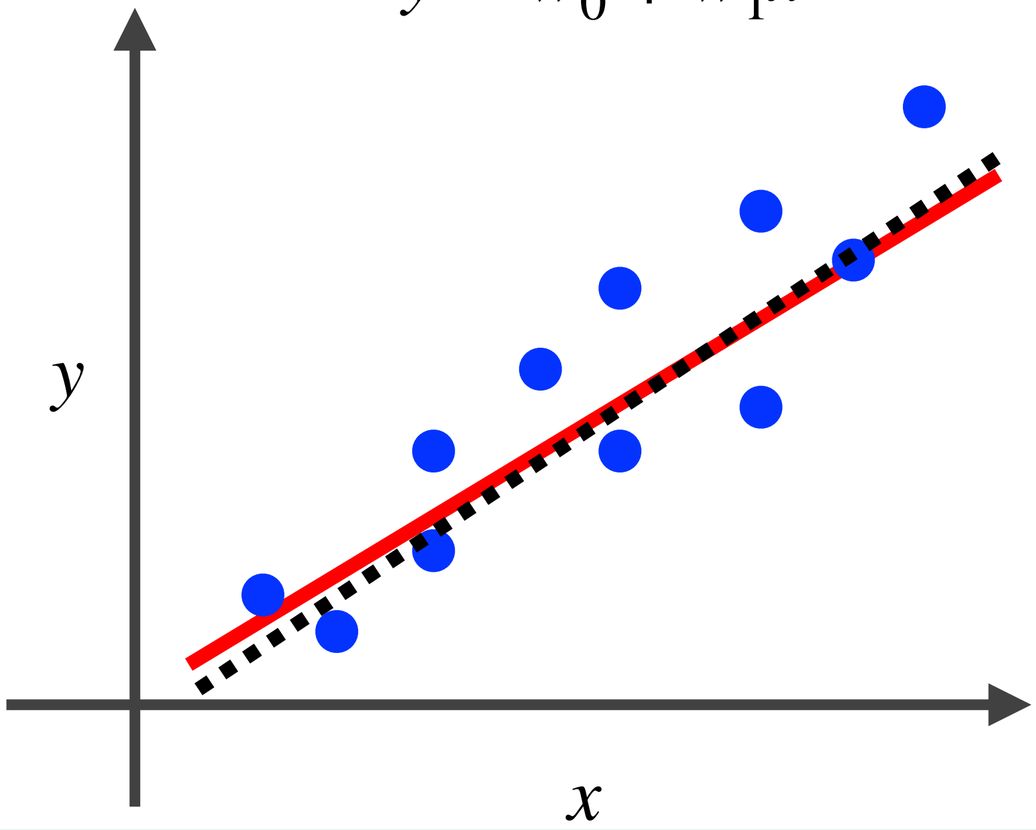
$$y = w_0 + w_1 x$$




## Observed data

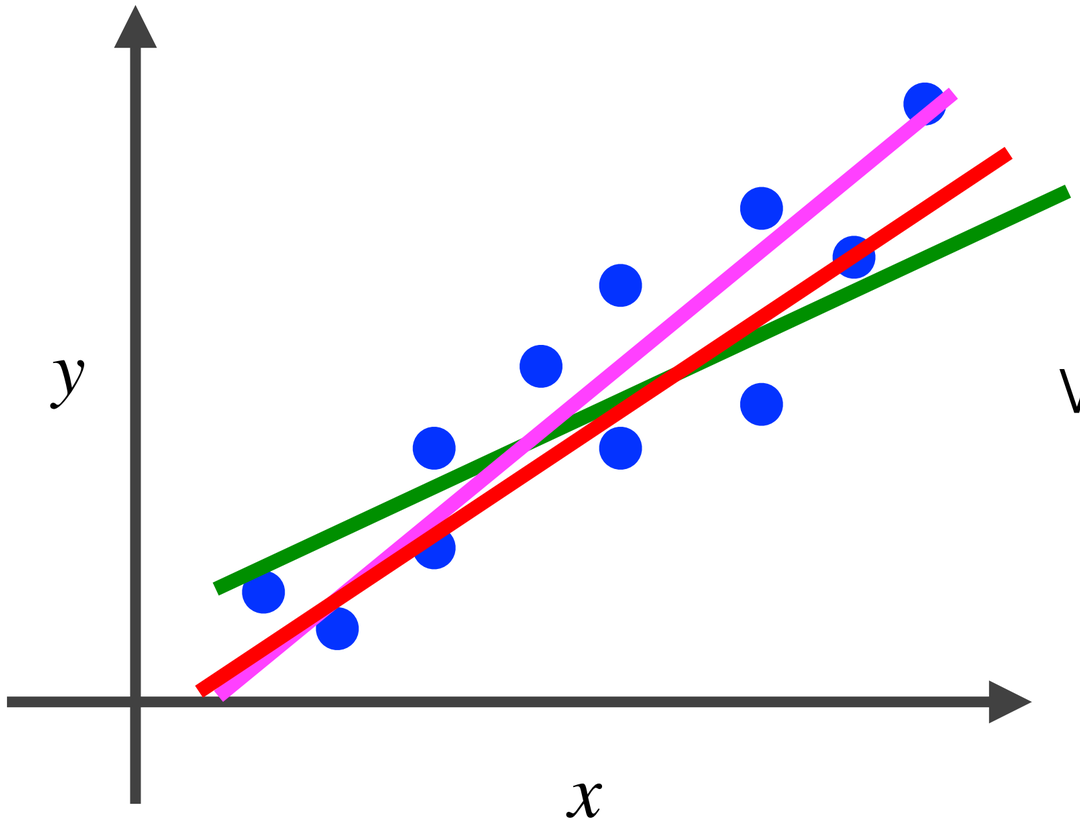
$$y = w_0 + w_1 x \ + \text{noise}$$

$$\hat{y} = \hat{w}_0 + \hat{w}_1 x$$



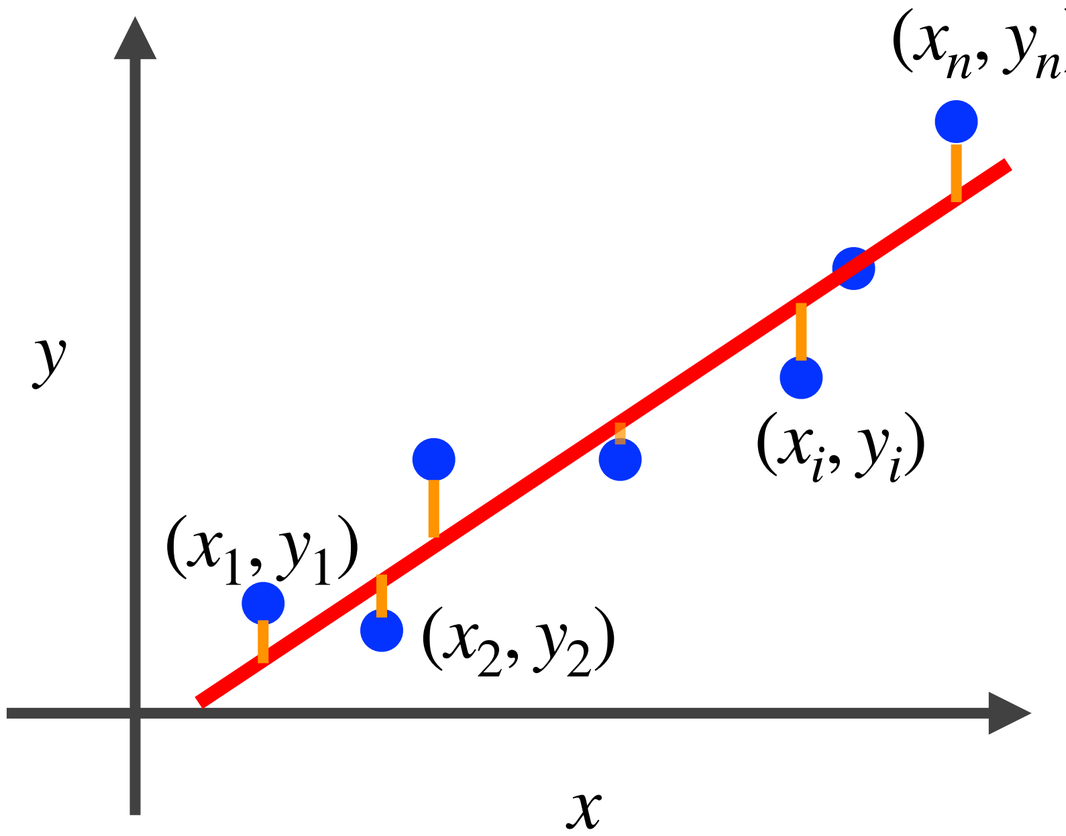
# Linear Model

$$\hat{y} = \hat{w}_0 + \hat{w}_1 x$$




Which one is the best?

# Mean-Squared Error (MSE)




Compute the error from sample i using the following loss function

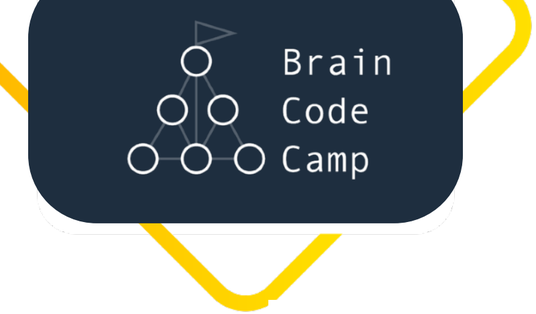
$$L(y_i, \hat{y}_i) = \left(y_i - \hat{y}_i\right)^2$$

Combine the errors from all samples

$$MSE(Y, \hat{Y}) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{y}_i)$$

$$Y : y_1, y_2, \ldots, y_i, \ldots, y_n$$

$$\hat{Y} : \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_i, \ldots, \hat{y}_n \quad = \frac{1}{n}\sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2$$




# Linear Regression

## sklearn.linear_model.LinearRegression

*class* `sklearn.linear_model.`**`LinearRegression`**(*, *fit_intercept=True*, *copy_X=True*, *n_jobs=None*, *positive=False*)

[source]

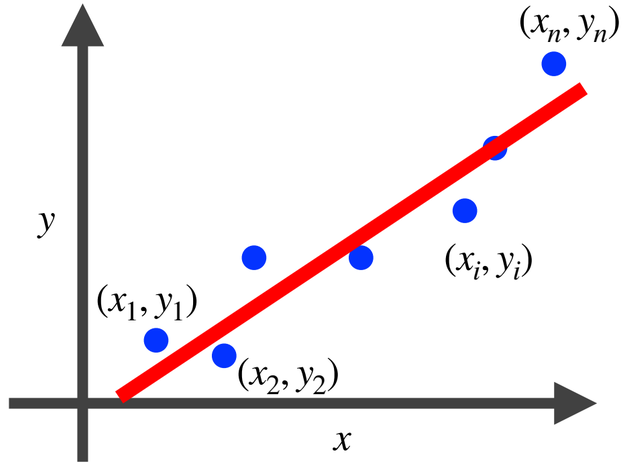
Ordinary least squares Linear Regression.

LinearRegression fits a linear model with coefficients w = (w1, ..., wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

# Linear Regression

Data

$$(x_1, y_1), (x_2, y_2), \ldots, (x_{100}, y_{100})$$




**x**

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{100} \end{bmatrix}$$

shape = (100, 1)

**y**

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{100} \end{bmatrix}$$

shape = (100, 1)

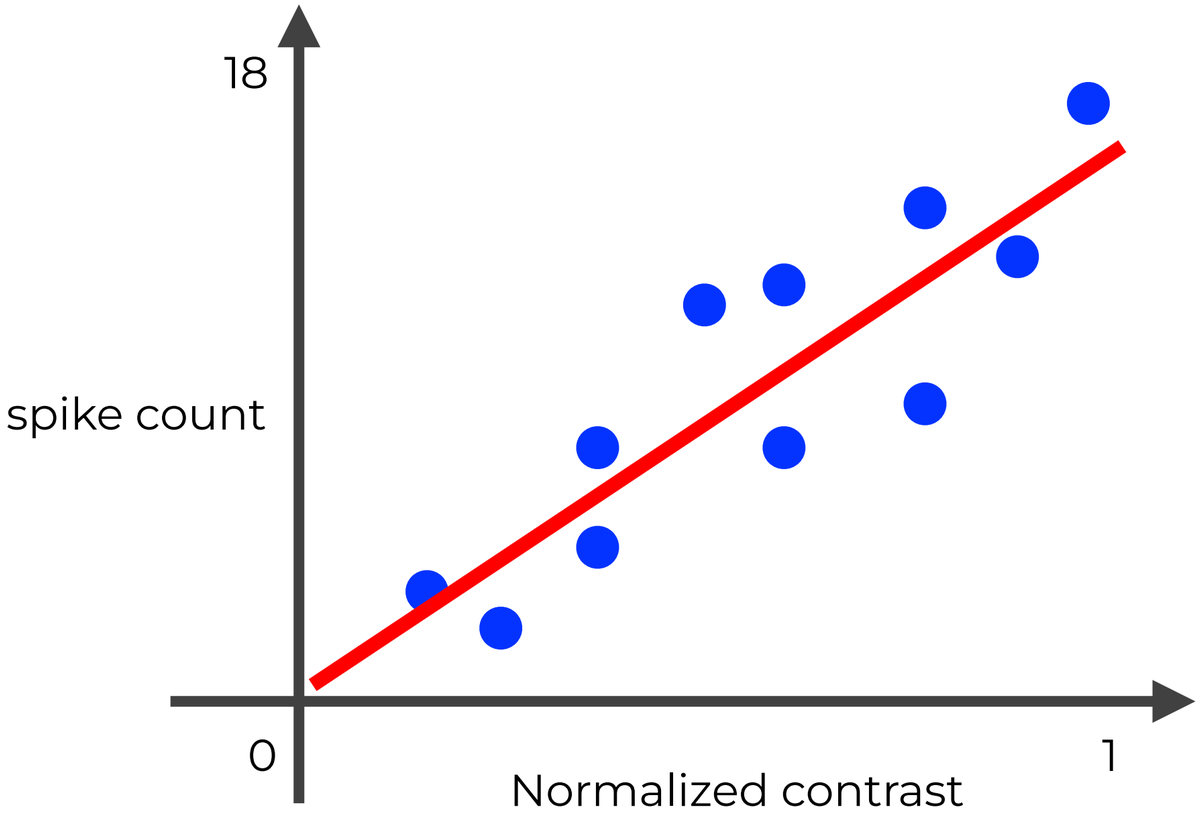
```
# Import a necessary module
from sklearn.linear_model import LinearRegression

# Create the model
model_linear = LinearRegression()

# Train the model
model_linear.fit(x, y)

# Make prediction
y_hat = model_linear.predict(x)
```

# Spike Count Prediction

$$\hat{y} = \hat{w}_0 + \hat{w}_1 x$$

1 feature

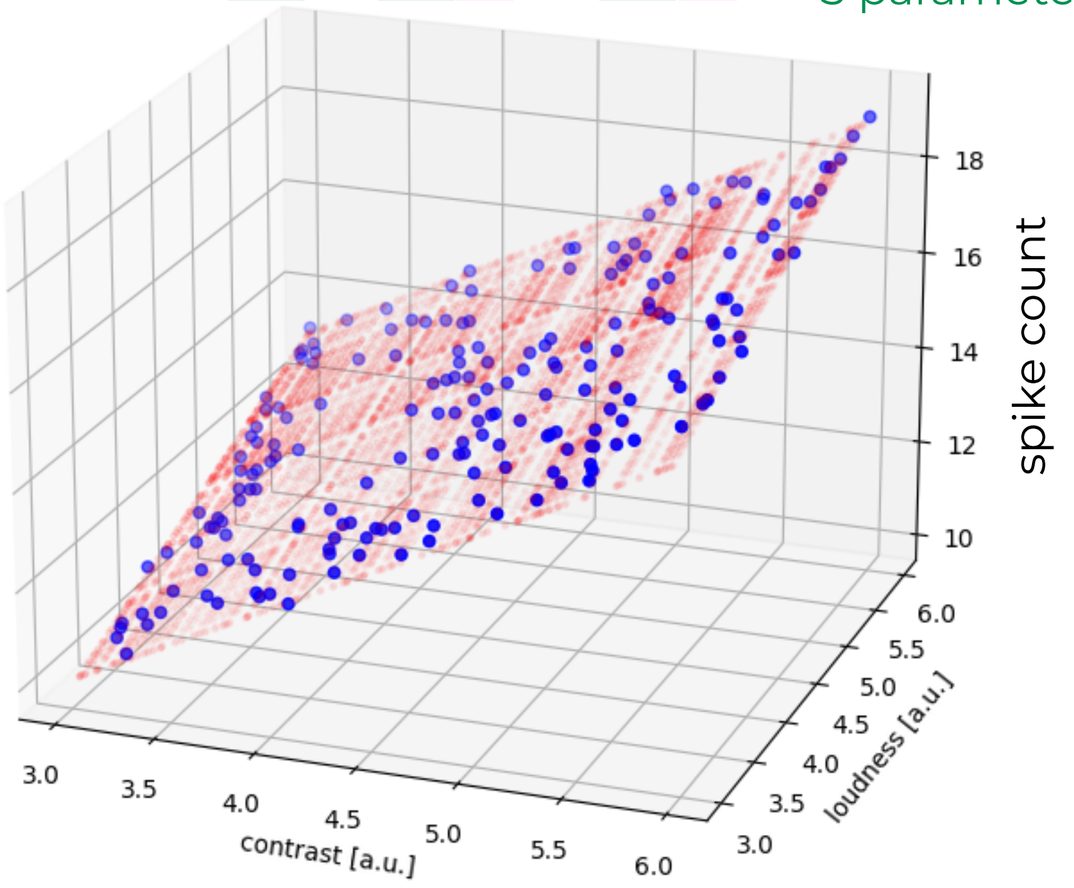
2 parameters




$$\hat{y} = \hat{w}_0 + \hat{w}_1 x_1 + \hat{w}_2 x_2$$

2 features

3 parameters



# Multiple Linear Regression

Let consider p features and n samples

Each datapoint is represented by a point in (p+1)-dimensional space

$$y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_p x_p$$

Find $\hat{w}_0, \hat{w}_1, \ldots, \hat{w}_p$ that minimize $MSE(Y, \hat{Y})$

feature 1 of sample i

feature p of sample i

$$\min_{\hat{w}_0,\ldots,\hat{w}_p} MSE(Y, \hat{Y}) = \min_{\hat{w}_0,\ldots,\hat{w}_p} \frac{1}{n} \sum_{i=1}^{n} \left( y_i - (\hat{w}_0 + \hat{w}_1 x_{i1} + \ldots + \hat{w}_p x_{ip}) \right)^2$$

**sklearn.linear_model.LinearRegression**

*class* sklearn.linear_model.**LinearRegression**(*, *fit_intercept=True*, *copy_X=True*, *n_jobs=None*, *positive=False*)

[source]

# Multiple Linear Regression

Data

$\mathbf{X}$

$$\begin{bmatrix} x_{11} & \cdots & x_{1p} \\ x_{21} & \cdots & x_{2p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{bmatrix}$$

feature 1 ... feature p

shape = (n, p)

$\mathbf{y}$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

shape = (n, 1)

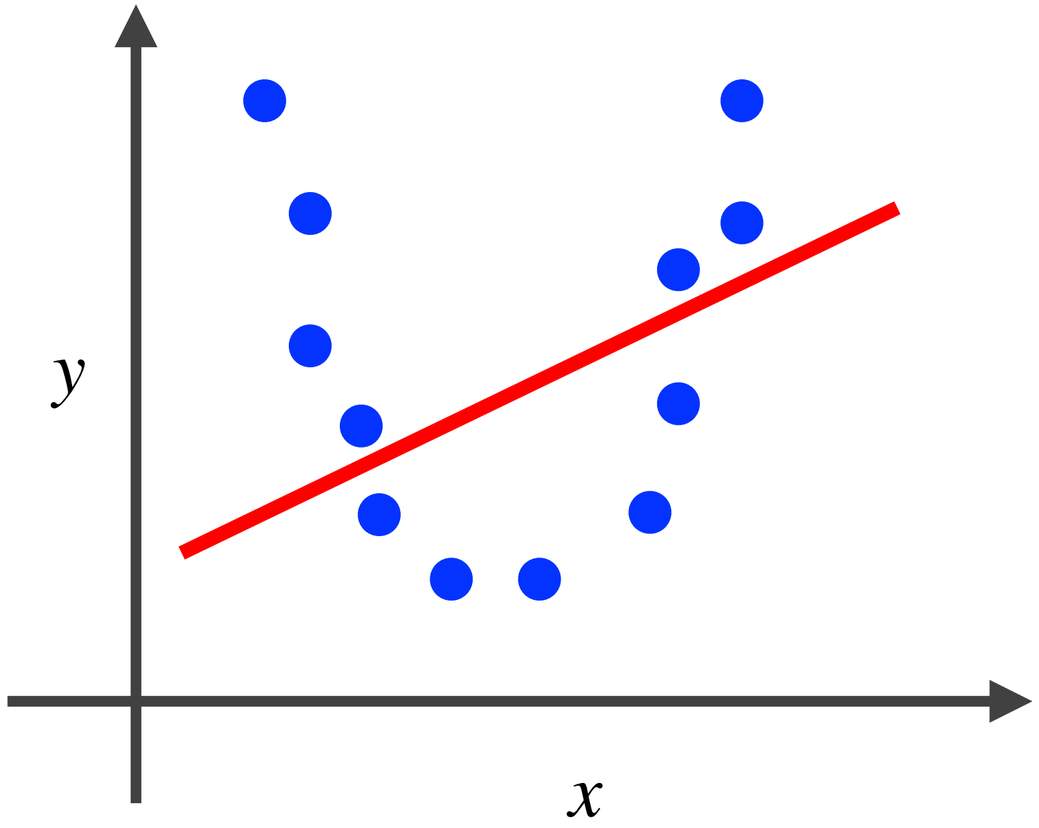
```
# Import a necessary module
from sklearn.linear_model import LinearRegression

# Create the model
model_linear = LinearRegression()

# Train the model
model_linear.fit(X, y)

# Make prediction
y_hat = model_linear.predict(x)
```

Brain Code Camp

# Nonlinear Relationship

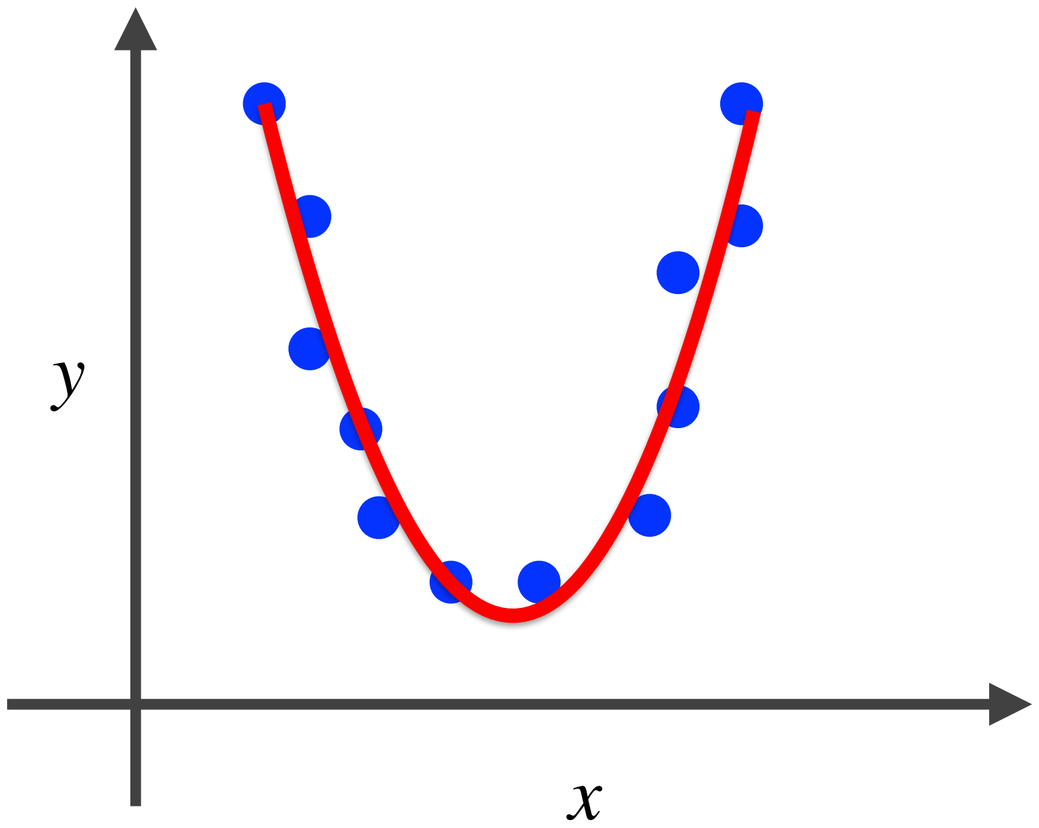
$y = mx + c$

$y = w_0 + w_1 x_1$




$y = ax^2 + bx + c$

$y = w_0 + w_1 x + w_2 x^2$



# Simple Polynomial

**A linear model**

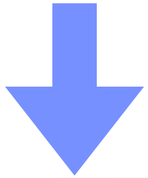
$$\hat{y} = \hat{w}_0 + \hat{w}_1 x_1 + \hat{w}_2 x_2 + \ldots + \hat{w}_p x_p$$

feature 1 feature 2 feature p

**A simple polynomial**

$$\hat{y} = \hat{w}_0 + \hat{w}_1 x + \hat{w}_2 x^2 + \ldots + \hat{w}_p x^p$$

Let $z_i = x^i$

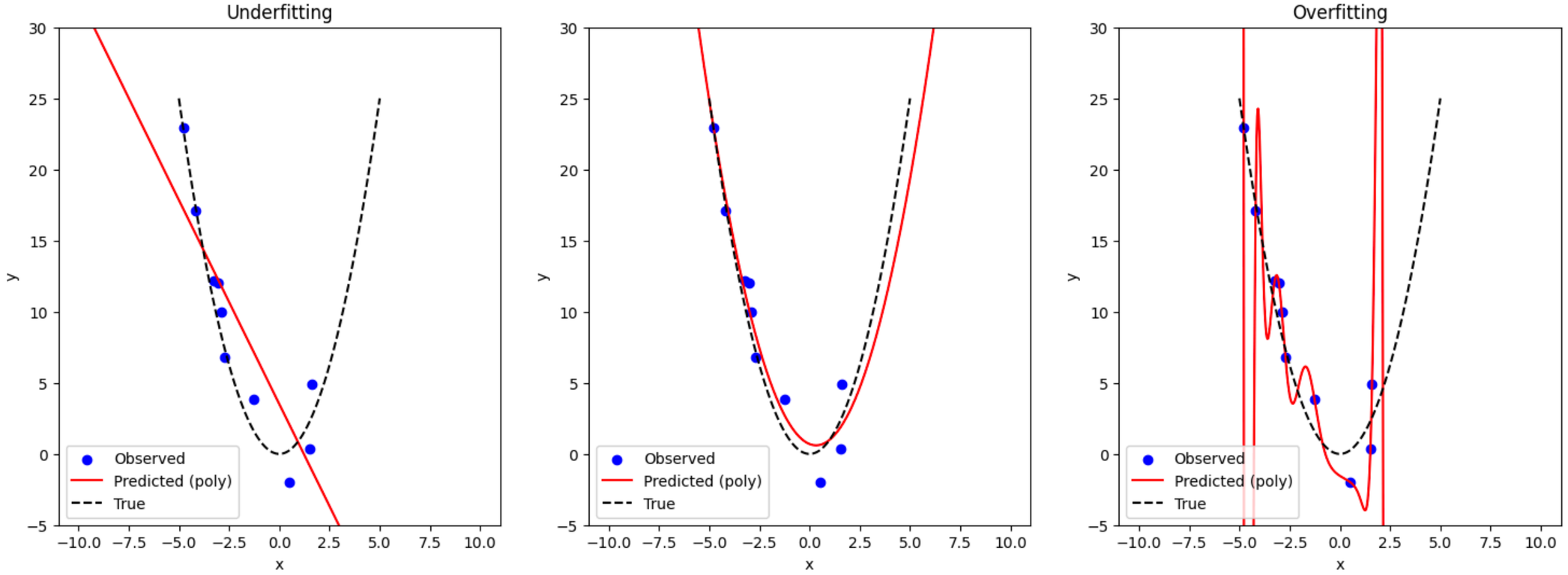


$$\hat{y} = \hat{w}_0 + \hat{w}_1 z_1 + \hat{w}_2 z_2 + \ldots + \hat{w}_p z_p$$

# Underfitting and Overfitting

Assume that the true relationship is $y = x^2$, and we have collected only 10 points

$$\hat{y} = \hat{w}_0 + \hat{w}_1 x + \hat{w}_2 x^2 + \ldots + \hat{w}_p x^p$$



Brain Code Camp

# L2 Regularization

training data  regularization term

$$\min_{\hat{w}_0,\ldots,\hat{w}_p} MSE(Y, \hat{Y}) + \lambda(\hat{w}_0^2 + \hat{w}_1^2 + \ldots + \hat{w}_p^2)$$

regularization parameter

Ensures that what we predict, $\hat{Y}$, matches what we have collected, $Y$

Ensures that the parameters do not become too large

$\lambda = 0$

$$\min_{\hat{w}_0,\ldots,\hat{w}_p} MSE(Y, \hat{Y}) + 0$$

no regularization

$\lambda$ large

$$\min_{\hat{w}_0,\ldots,\hat{w}_p} \text{small} + \lambda(\hat{w}_0^2 + \hat{w}_1^2 + \ldots + \hat{w}_p^2)$$

Do not care about the training data

A good $\lambda$ cares about the training data, while also pays attention to the regularization term

Brain Code Camp

# Linear regression

## sklearn.linear_model.LinearRegression

*class* sklearn.linear_model.**LinearRegression**(*, *fit_intercept=True*, *copy_X=True*, *n_jobs=None*, *positive=False*) [source]

**Linear regression with L2 regularization/ Tikhonov regularization**

## sklearn.linear_model.Ridge

*class* sklearn.linear_model.**Ridge**(*alpha=1.0*, *, *fit_intercept=True*, *copy_X=True*, *max_iter=None*, *tol=0.0001*, *solver='auto'*, *positive=False*, *random_state=None*) [source]

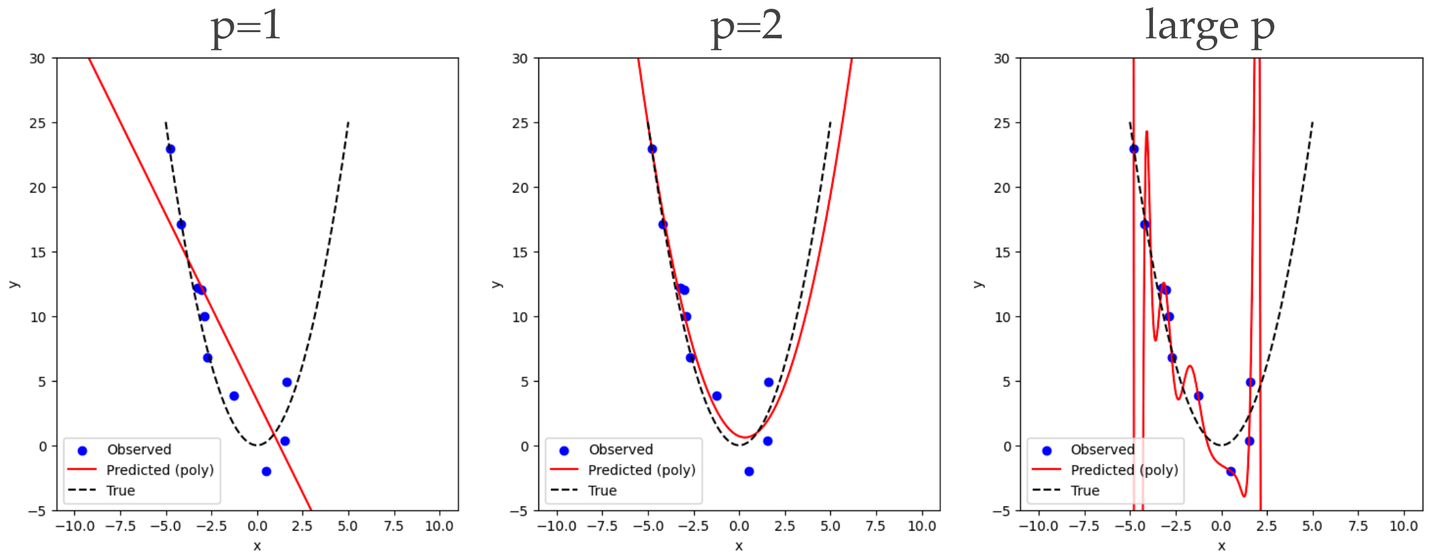
**Linear regression with L1 regularization**

## sklearn.linear_model.Lasso

*class* sklearn.linear_model.**Lasso**(*alpha=1.0*, *, *fit_intercept=True*, *precompute=False*, *copy_X=True*, *max_iter=1000*, *tol=0.0001*, *warm_start=False*, *positive=False*, *random_state=None*, *selection='cyclic'*) ¶ [source]

# Underfitting and Overfitting

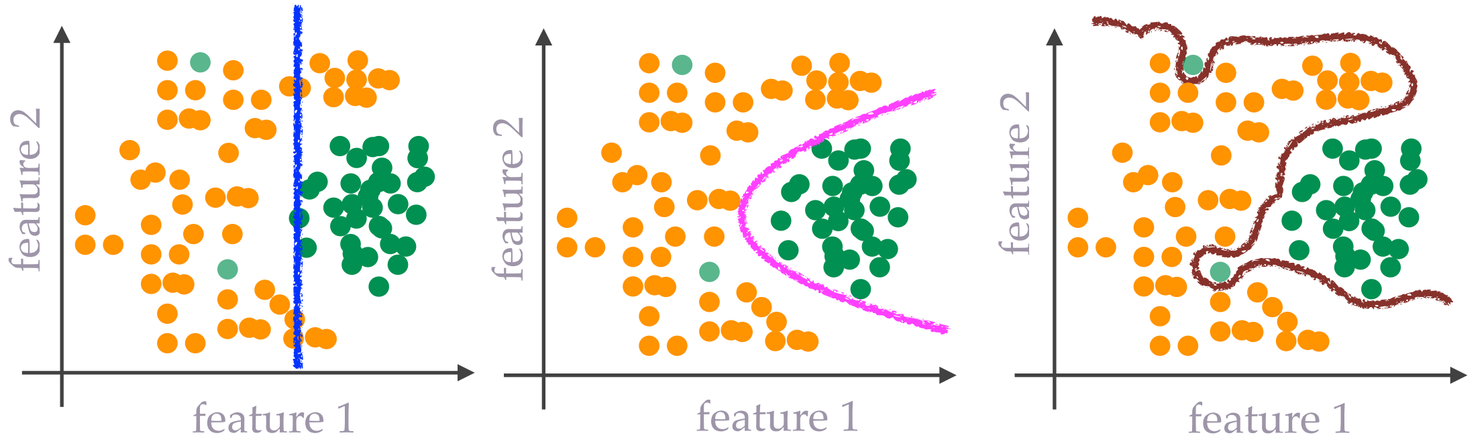
regression




How do we pick p?

Validation data!

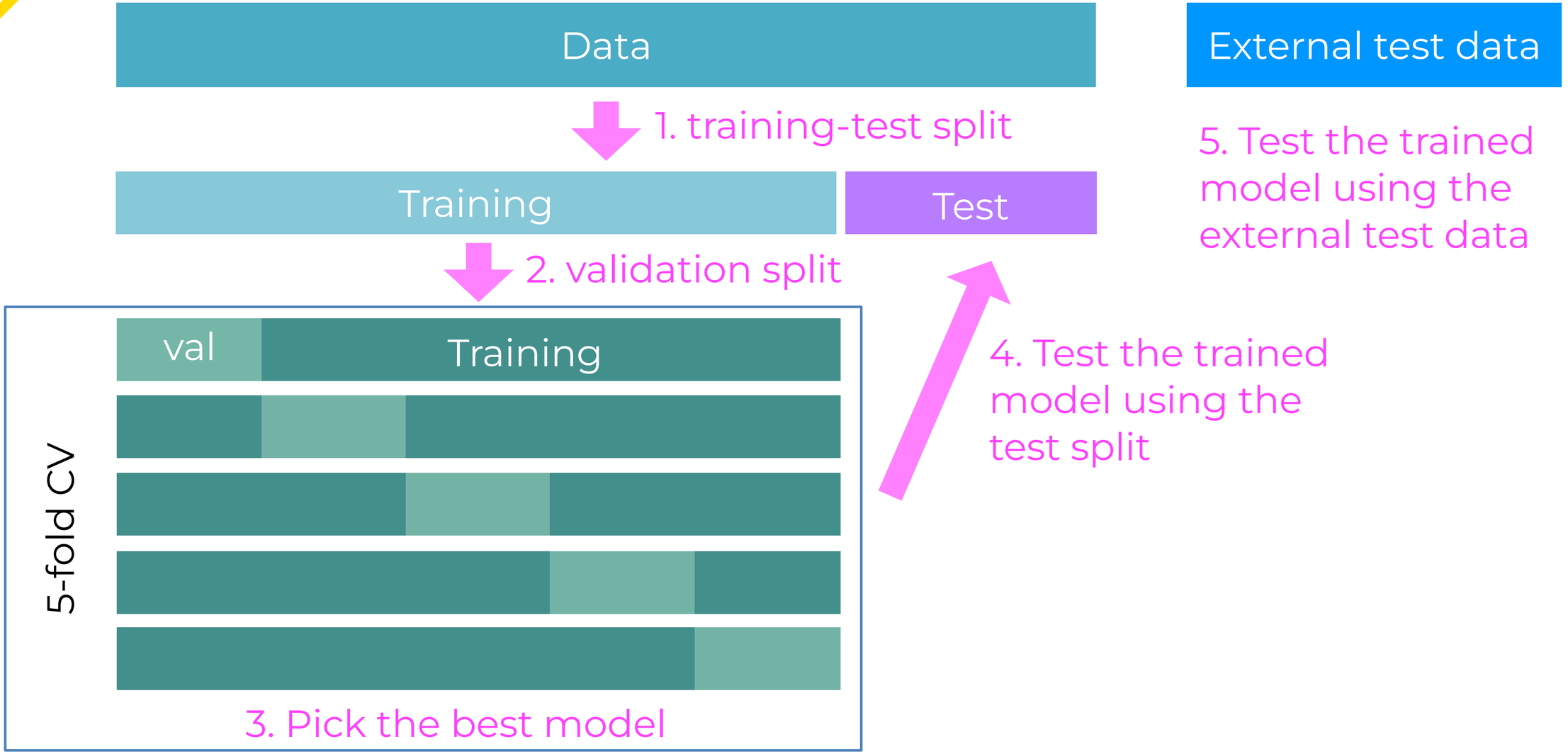
classification



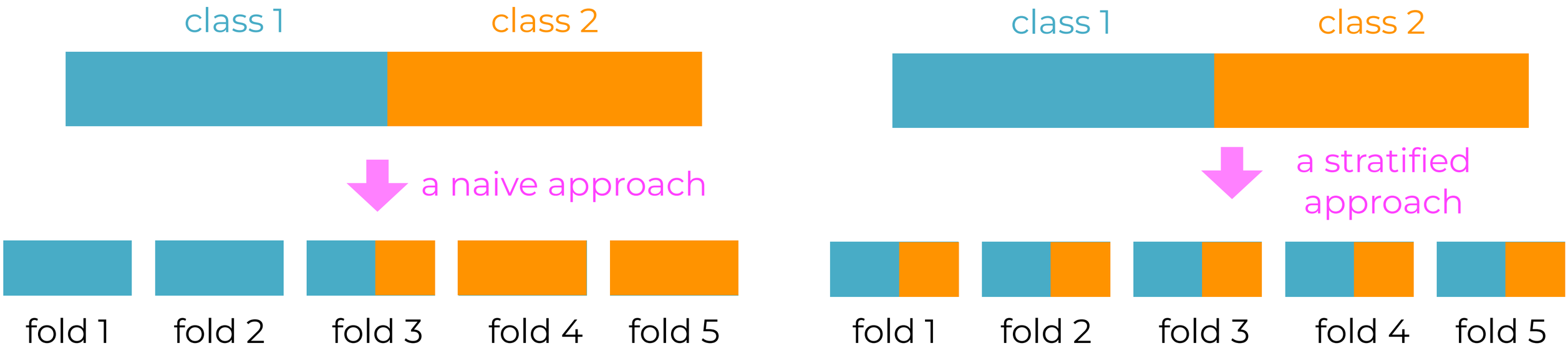

**Underfitting** **Overfitting**

Brain Code Camp

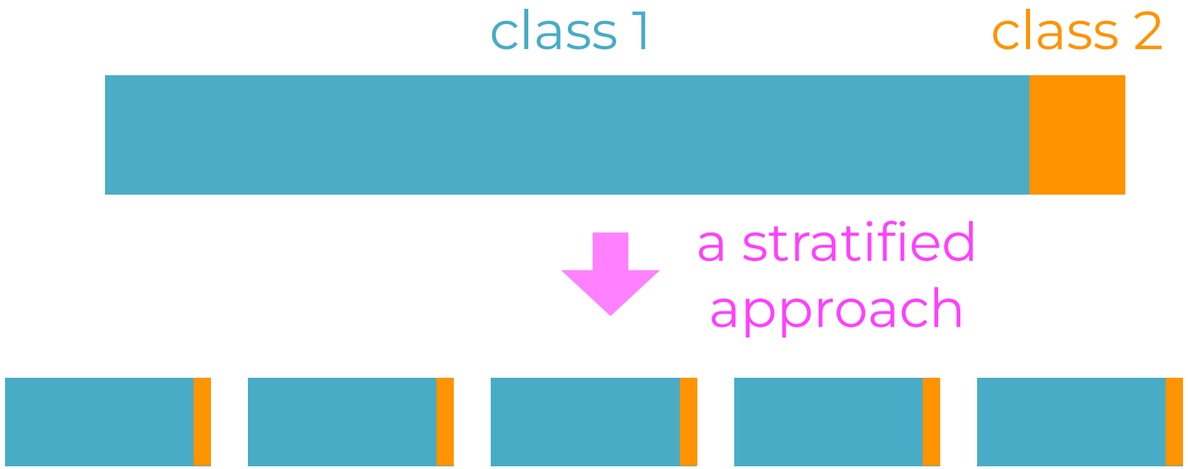
# A simple practical approach

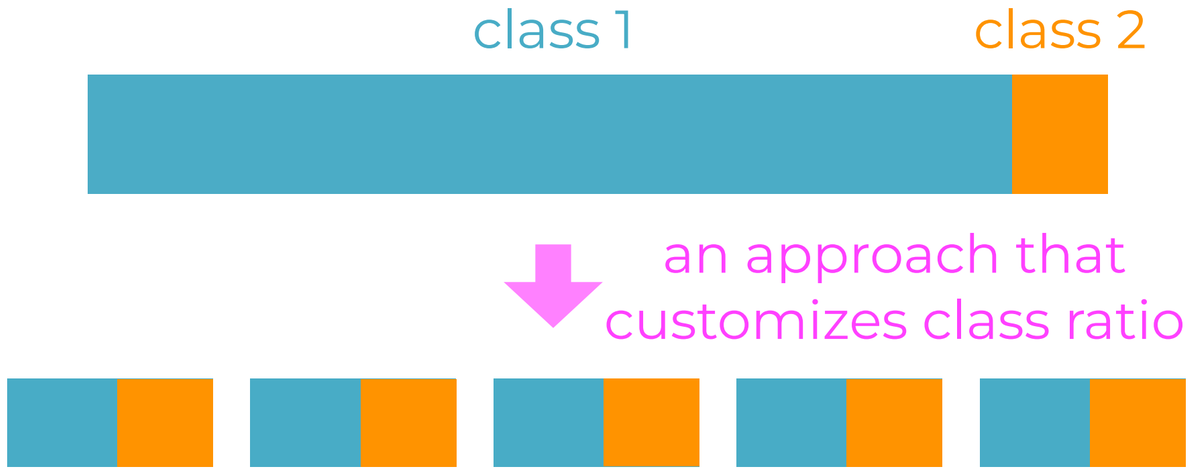


# Stratified CV



# Class imbalance

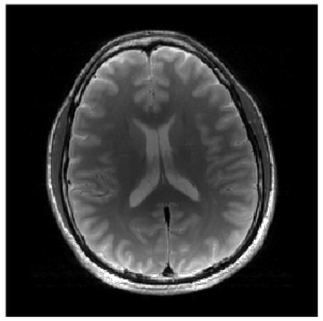


# Magnetic Resonance Imaging

unknown

Underlying image



Input

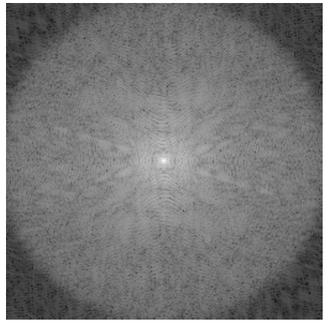
$x$

MRI machine

Approximately linear

$A$

Acquired k-space data



Acquired data

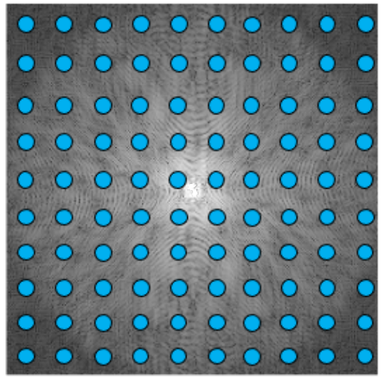
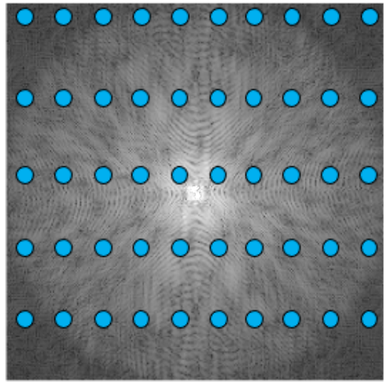
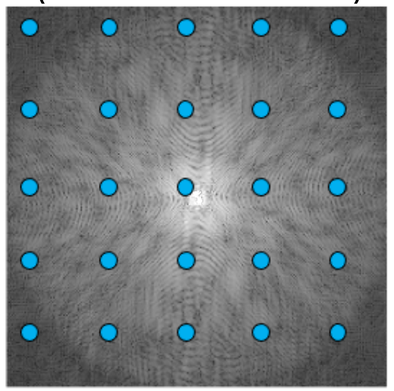
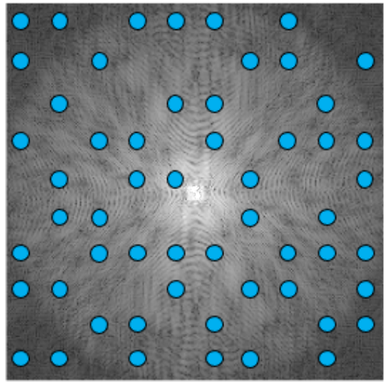
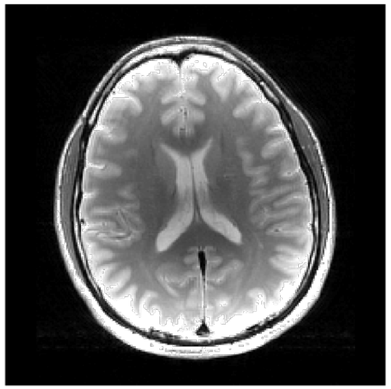
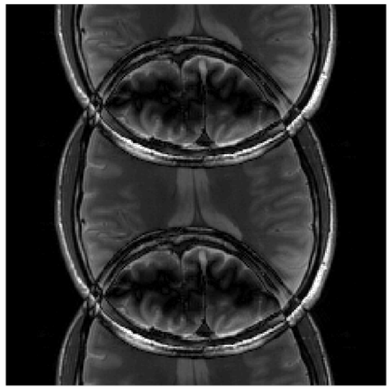
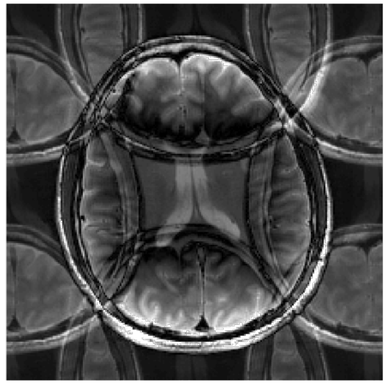
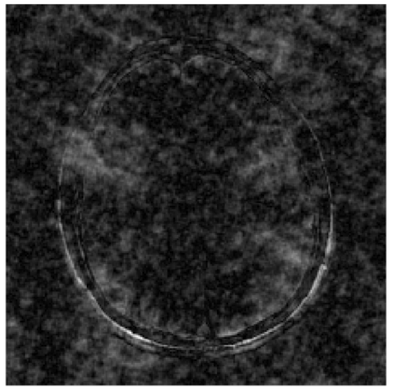
$y$

Want to find $x$ that makes $y \approx Ax$

$$\min_x \|y - Ax\|_2^2$$

$$x = A^{-1}y$$

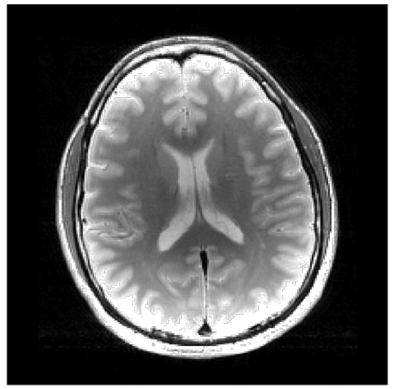
# Accelerated MRI

| | Fully sampled acquisition | Uniformly undersampled (one direction) | Uniformly undersampled (two directions) | Randomly undersampled |
|---|---|---|---|---|
| Acquired data (k-space) |  |  |  |  |
| | $x = A^{-1}y$ | $\min_x \|y - Ax\|_2^2$ | $\min_x \|y - Ax\|_2^2$ | $\min_x \|y - Ax\|_2^2$ |
| Reconstructed data (image-space) |  |  |  |  |
| | | Artifact (one direction) | Artifact (two directions) | Noise-like artifact |

# Accelerated MRI with Regularization

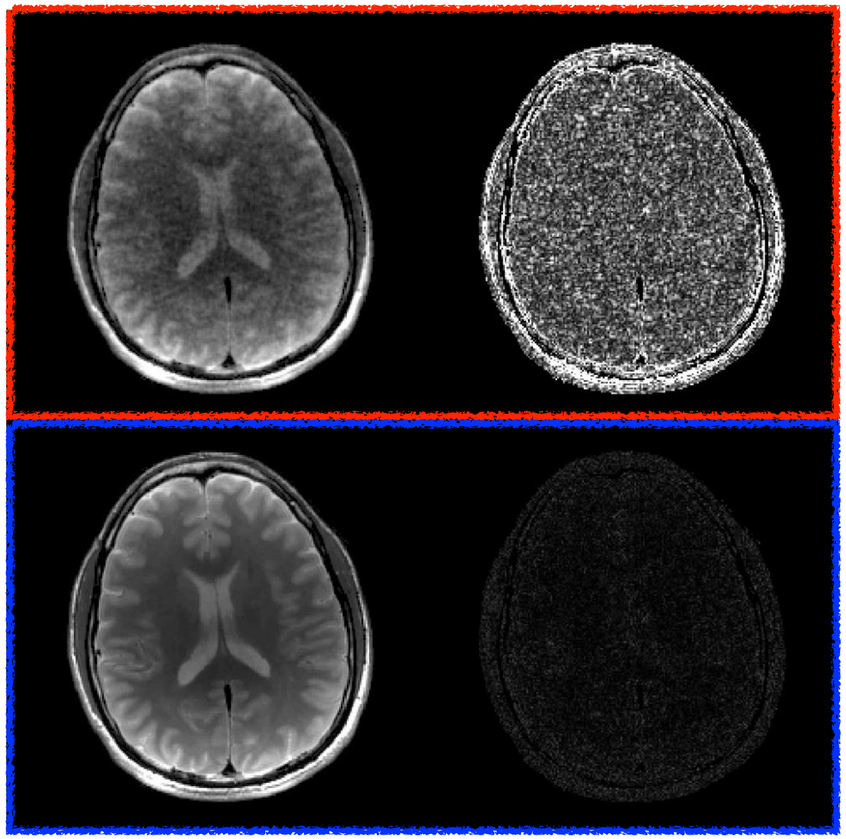
**Reduction factor = 2**

**True**

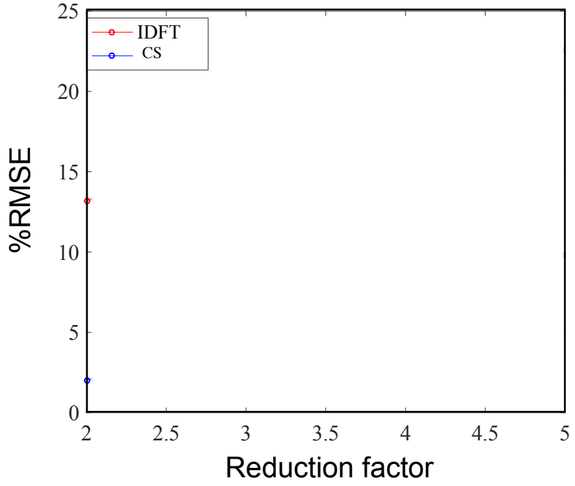


Reconstructed Image

Error (scaled 10x)



$$\hat{x} = \arg\min_x \frac{1}{2}\|Ax - y\|_2^2$$



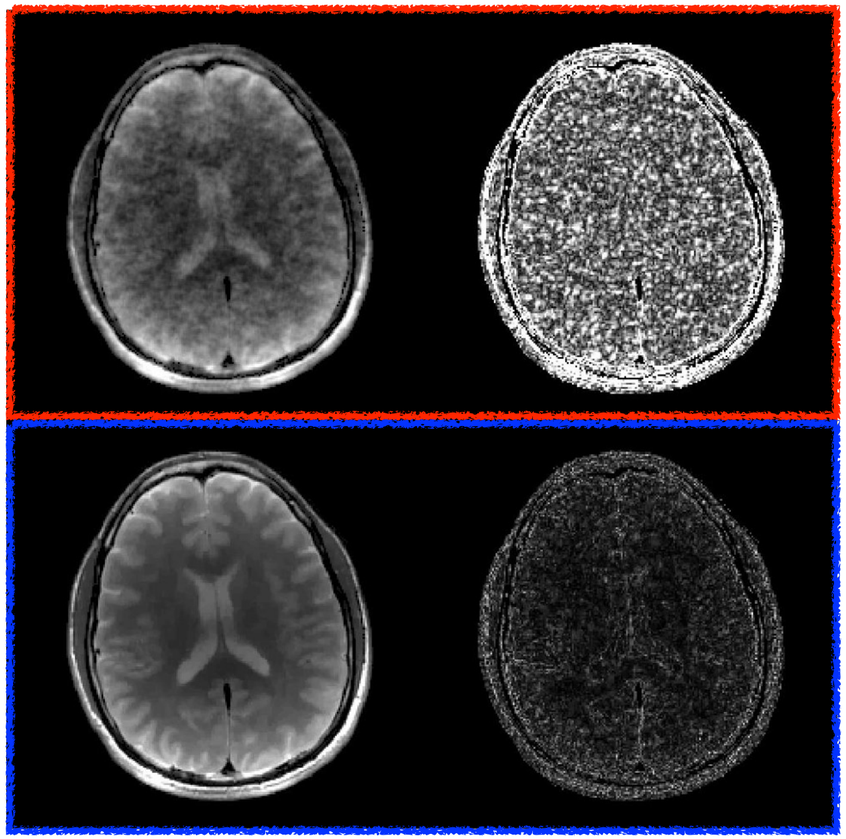

$$\hat{x} = \arg\min_x \frac{1}{2}\|Ax - y\|_2^2 + \lambda\|Gx\|_1$$

# Accelerated MRI with Regularization

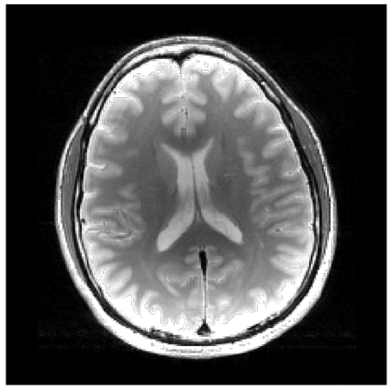
**Reduction factor = 3**

Reconstructed Image

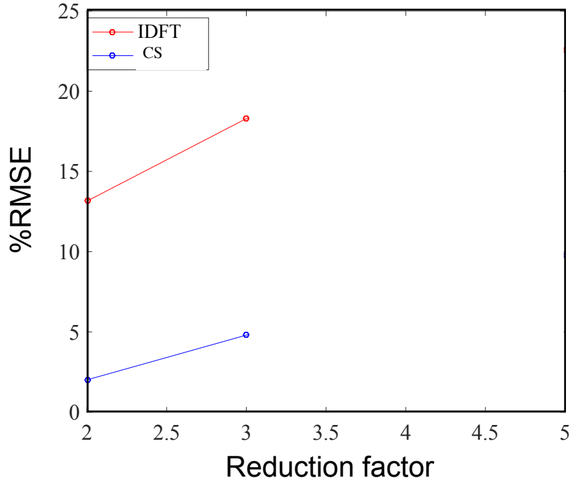
Error (scaled 10x)



**True**



$$\hat{x} = \arg\min_x \frac{1}{2}\|Ax - y\|_2^2$$



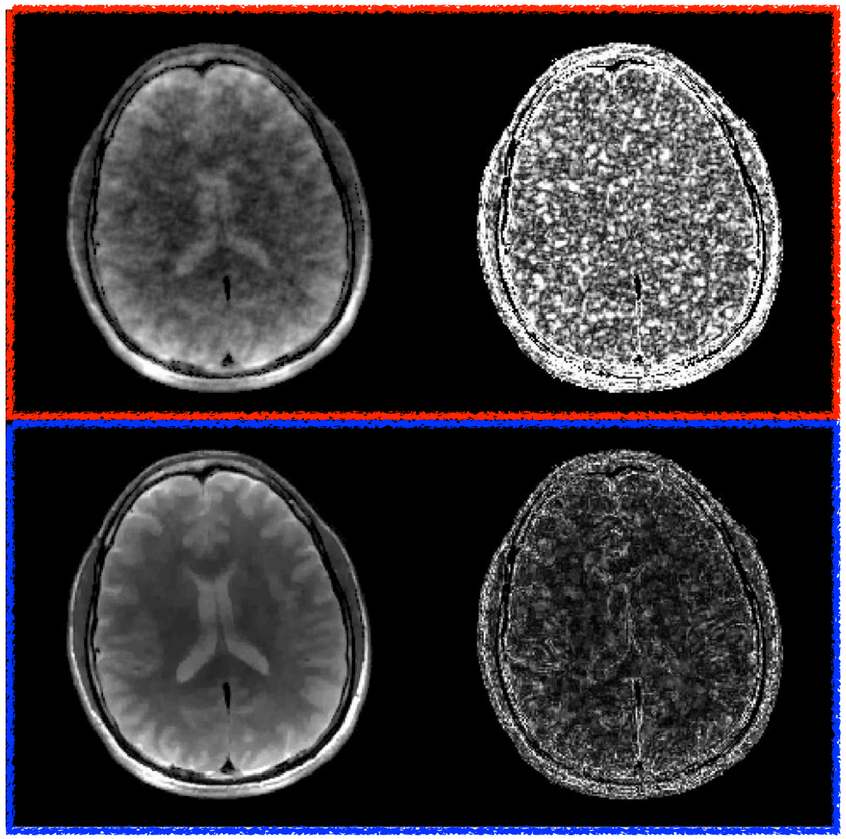

$$\hat{x} = \arg\min_x \frac{1}{2}\|Ax - y\|_2^2 + \lambda\|Gx\|_1$$

# Accelerated MRI with Regularization

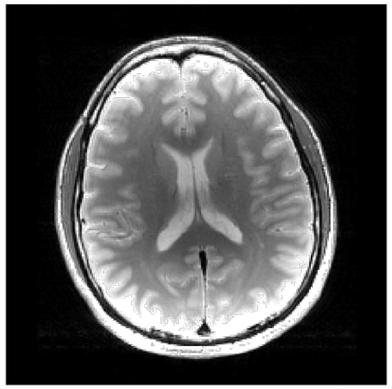
**Reduction factor = 4**

Reconstructed Image

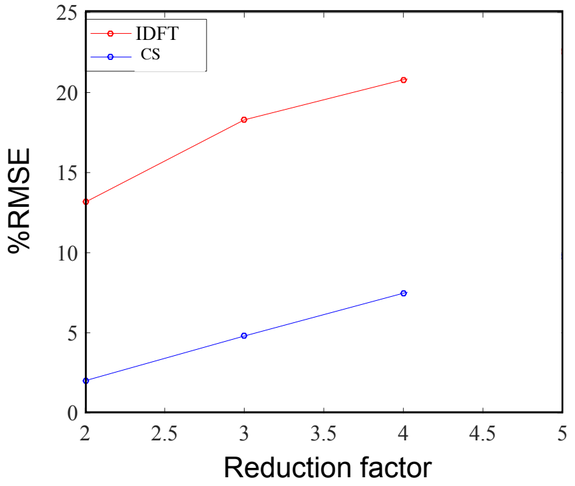
Error (scaled 10x)



**True**



$$\hat{x} = \arg\min_x \frac{1}{2}\|Ax - y\|_2^2$$




$$\hat{x} = \arg\min_x \frac{1}{2}\|Ax - y\|_2^2 + \lambda\|Gx\|_1$$

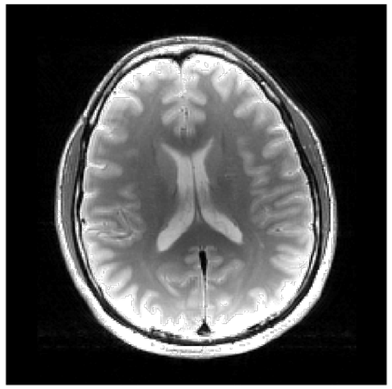
# Accelerated MRI with Regularization

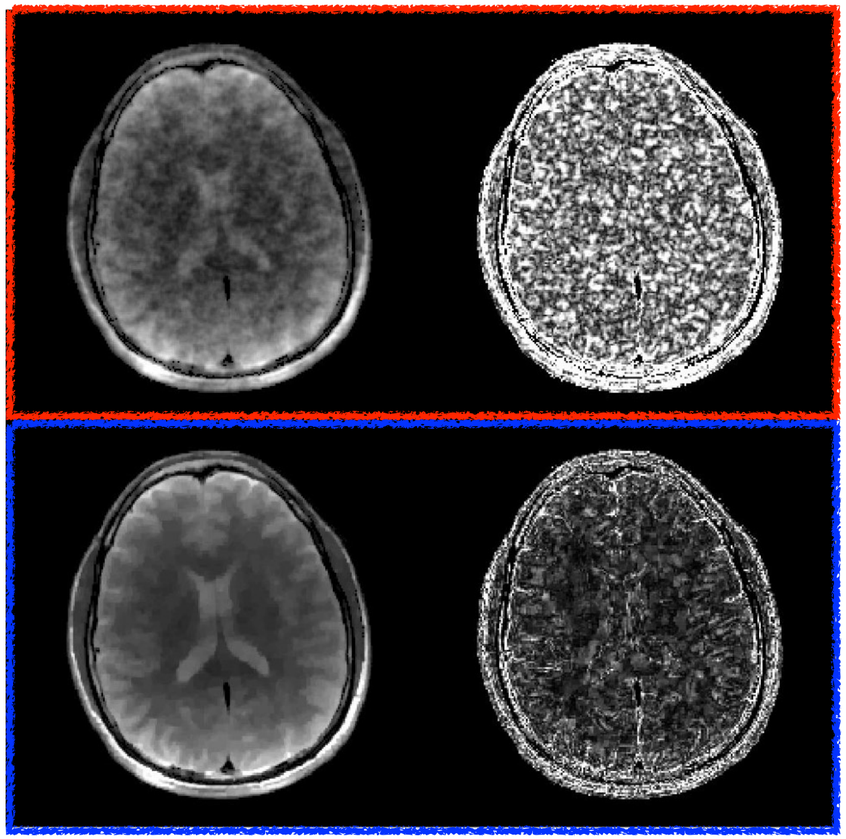
**Reduction factor = 5**

Reconstructed Image

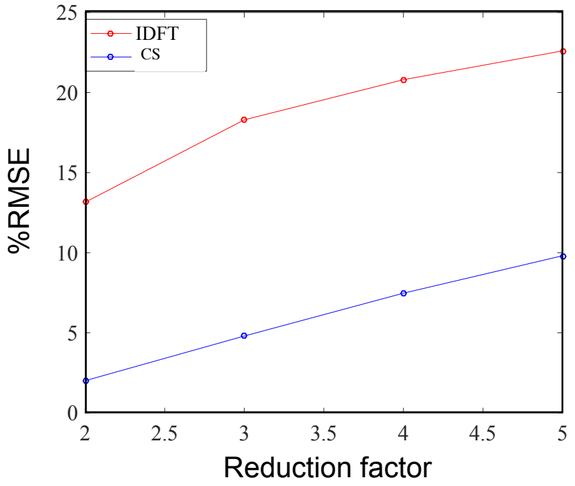
Error (scaled 10x)

**True**





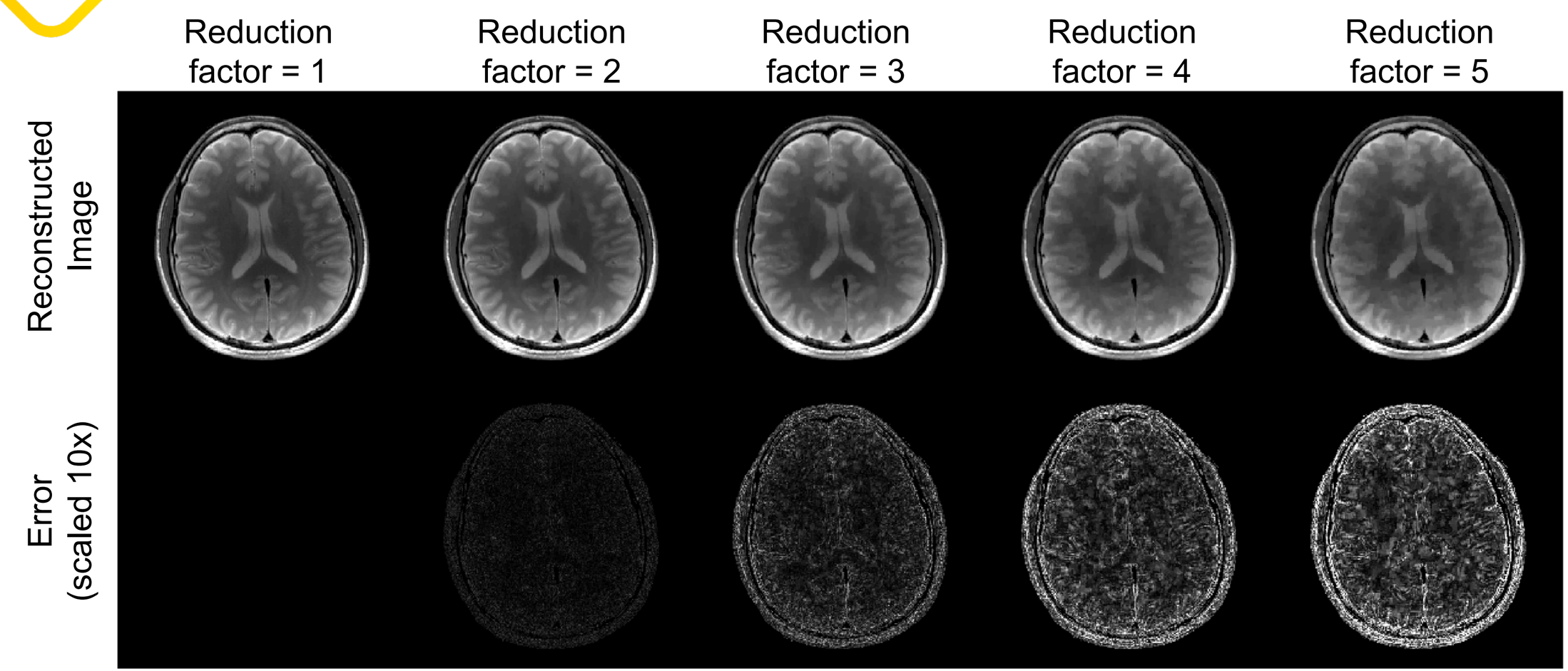
$$\hat{x} = \arg\min_x \frac{1}{2}\|Ax - y\|_2^2$$




$$\hat{x} = \arg\min_x \frac{1}{2}\|Ax - y\|_2^2 + \lambda\|Gx\|_1$$

Brain Code Camp

# Accelerated MRI with Regularization




$$\hat{x} = \arg\min_{x} \frac{1}{2}\|Ax - y\|_2^2 + \lambda\|Gx\|_1$$