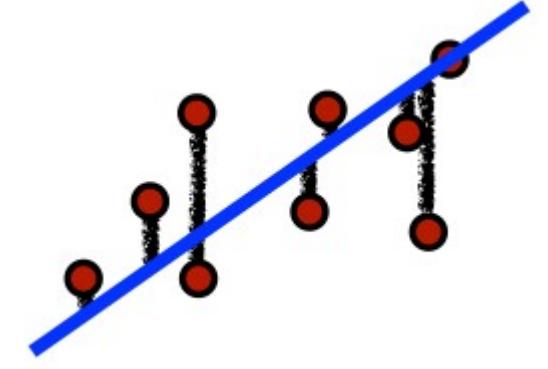


# Linear Regression

# Regression

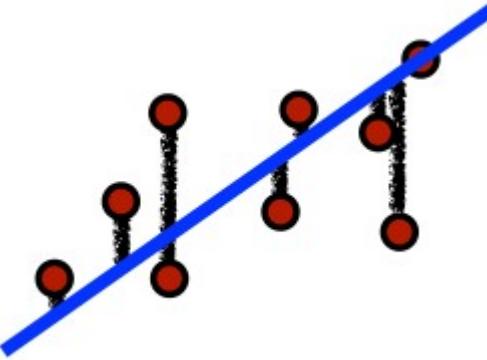


Goal: Learn a mapping from observations (features) to continuous labels given a training set (supervised learning)

Example Height, Gender, Weight → Shoe Size

- Audio features → Song year
- Processes, memory → Power consumption
- Historical financials → Future stock price
- Many more

# Linear Least Squares Regression



**Example:** Predicting shoe size from height, gender, and weight

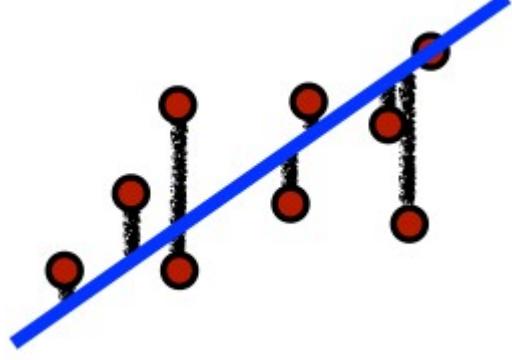
For each observation we have a feature vector,  $\mathbf{x}$ , and label,  $y$

$$\mathbf{x}^T = [x_1 \quad x_2 \quad x_3]$$

We assume a *linear* mapping between features and label:

$$y \approx w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

# Linear Least Squares Regression



**Example:** Predicting shoe size from height, gender, and weight

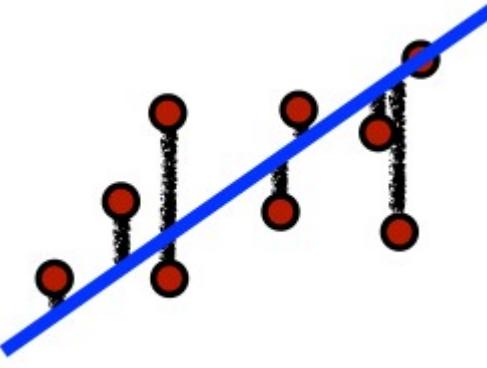
We can augment the feature vector to incorporate offset:

$$\mathbf{x}^\top = [1 \quad x_1 \quad x_2 \quad x_3]$$

We can then rewrite this linear mapping as scalar product:

$$y \approx \hat{y} = \sum_{i=0}^3 w_i x_i = \mathbf{w}^\top \mathbf{x}$$

# Why a Linear Mapping?

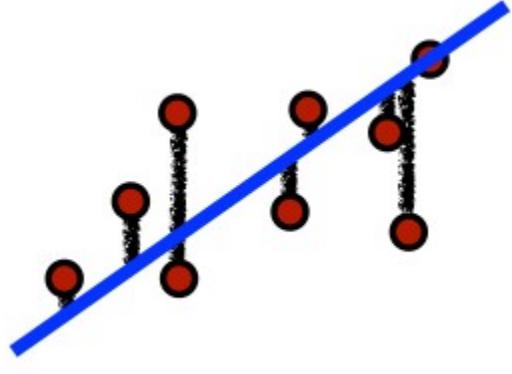


**Simple**

**Often works well in practice**

**Can introduce complexity via feature extraction**

# 1D Example



**Goal:** find the line of best fit

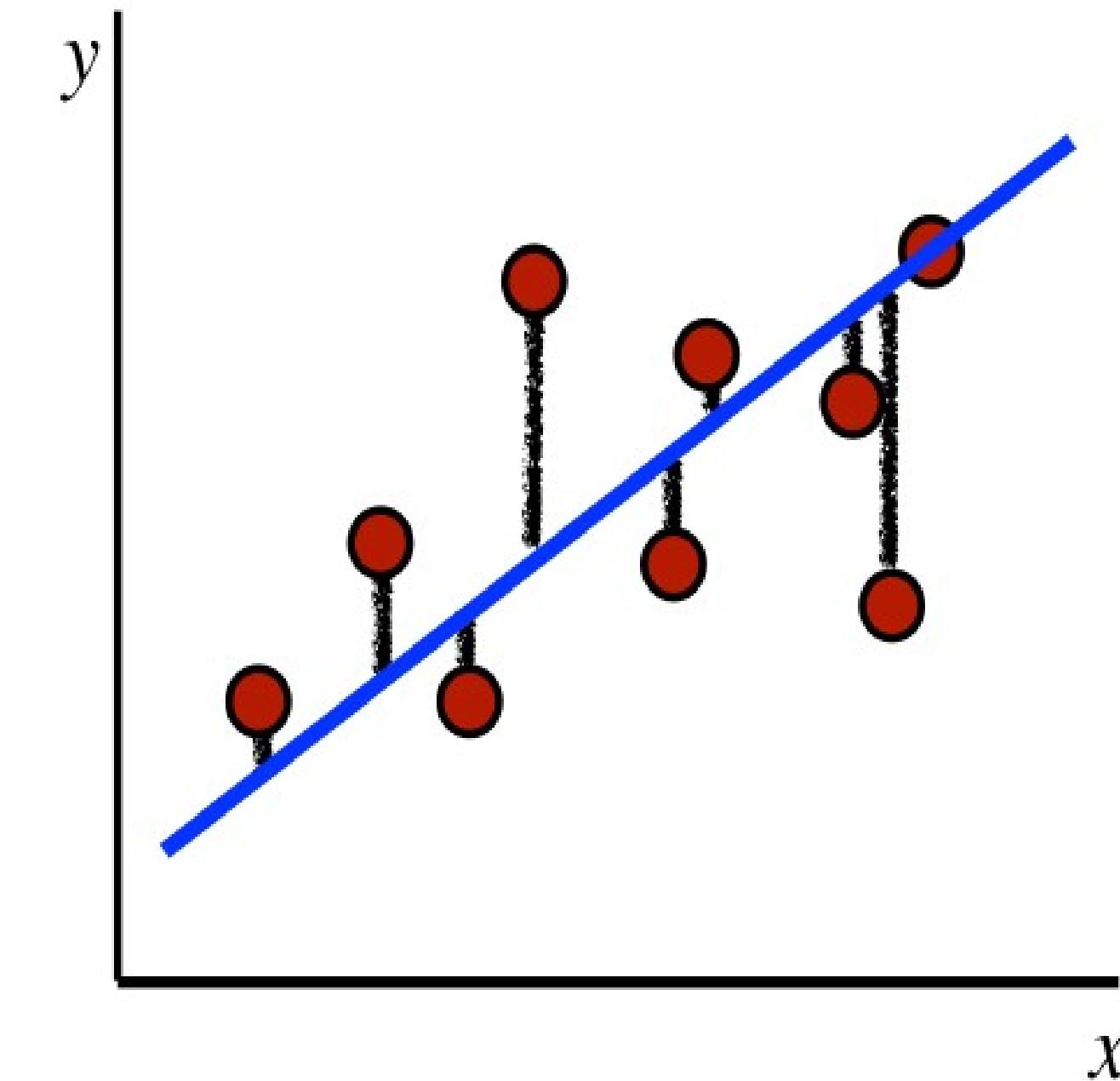
$x$  coordinate: features

$y$  coordinate: labels

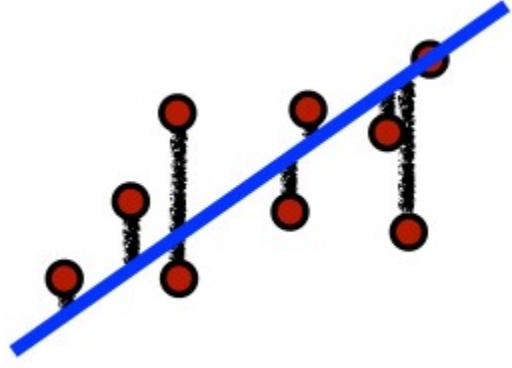
$$y \approx \hat{y} = w_0 + w_1 x$$

Intercept / Offset

Slope



# Evaluating Predictions



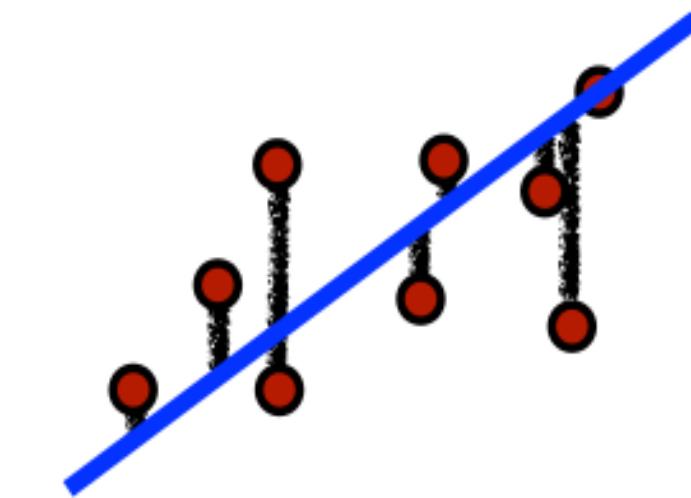
Can measure ‘closeness’ between label and prediction

- Shoe size: better to be off by one size than 5 sizes
- Song year prediction: better to be off by a year than by 20 years

What is an appropriate evaluation metric or ‘loss’ function?

- Absolute loss:  $|y - \hat{y}|$
- Squared loss:  $(y - \hat{y})^2$  ← Has nice mathematical properties

# How Can We Learn Model ( $\mathbf{w}$ )?



Assume we have  $n$  training points, where  $\mathbf{x}^{(i)}$  denotes the  $i$ th point

Recall two earlier points:

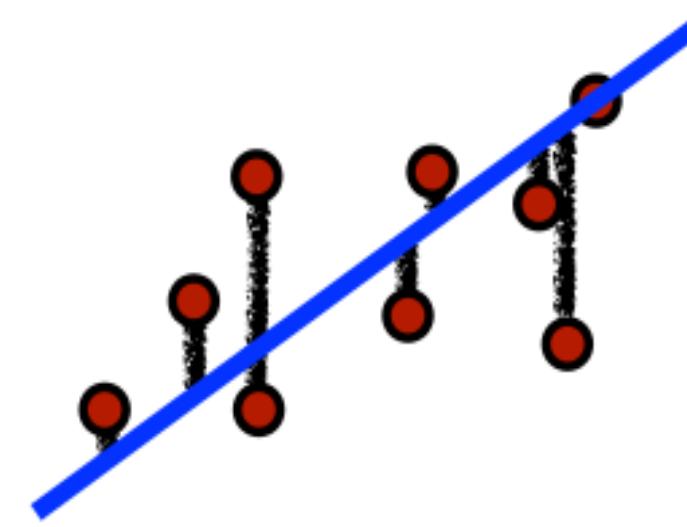
- *Linear assumption:*  $\hat{y} = \mathbf{w}^\top \mathbf{x}$
- We use *squared loss*:  $(y - \hat{y})^2$

Idea: Find  $\mathbf{w}$  that minimizes squared loss over training points:

$$\min_{\mathbf{w}} \sum_{i=1}^n (\underline{\mathbf{w}^\top \mathbf{x}^{(i)}} - \hat{y}^{(i)})^2$$

Given  $n$  training points with  $d$  features, we define:

- $\mathbf{X} \in \mathbb{R}^{n \times d}$ : matrix storing points
- $\mathbf{y} \in \mathbb{R}^n$ : real-valued labels
- $\hat{\mathbf{y}} \in \mathbb{R}^n$ : predicted labels, where  $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$
- $\mathbf{w} \in \mathbb{R}^d$ : regression parameters / model to learn



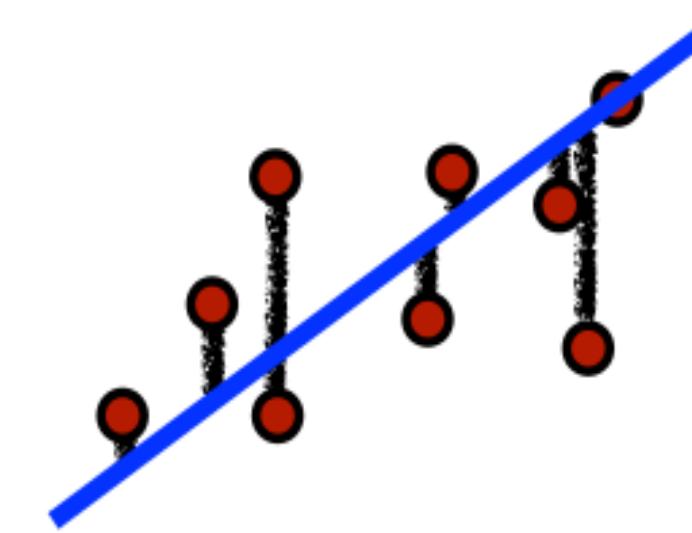
**Least Squares Regression:** Learn mapping ( $\mathbf{w}$ ) from features to labels that minimizes residual sum of squares:

$$\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

Equivalent  $\min_{\mathbf{w}} \sum_{i=1}^n (\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)})^2$  by definition of Euclidean norm

Find solution by setting derivative to zero

$$1D: f(w) = \|\mathbf{w}\mathbf{x} - \mathbf{y}\|_2^2 = \sum_{i=1}^n (wx^{(i)} - y^{(i)})^2$$



$$\frac{df}{dw}(w) = 2 \underbrace{\sum_{i=1}^n x^{(i)}(wx^{(i)} - y^{(i)})}_{w\mathbf{x}^\top \mathbf{x} - \mathbf{x}^\top \mathbf{y}} = 0 \iff w\mathbf{x}^\top \mathbf{x} - \mathbf{x}^\top \mathbf{y} = 0$$
$$\iff w = (\mathbf{x}^\top \mathbf{x})^{-1} \mathbf{x}^\top \mathbf{y}$$

**Least Squares Regression:** Learn mapping ( $\mathbf{w}$ ) from features to labels that minimizes residual sum of squares:

$$\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

Closed form solution:  $\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$  (if inverse exists)

# Overfitting and Generalization

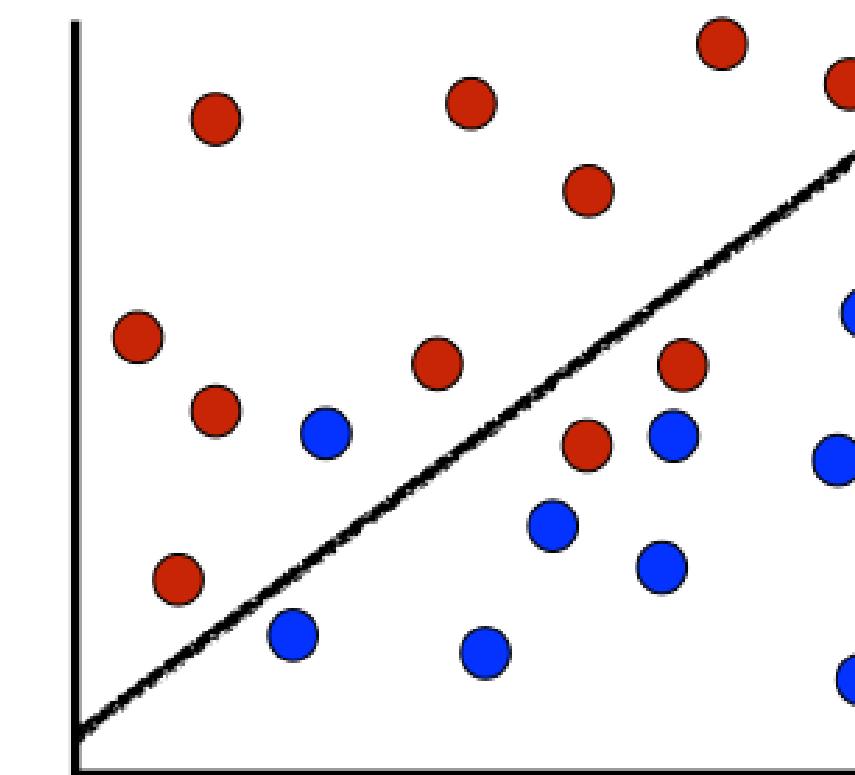
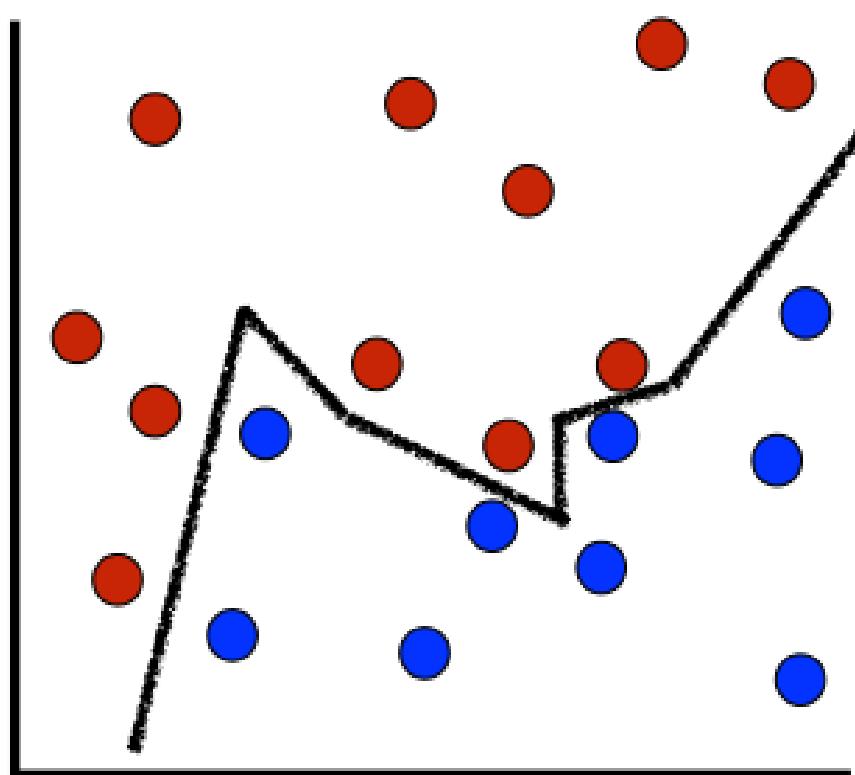
We want good predictions on new data, i.e., 'generalization'

Least squares regression minimizes training error, and could overfit

- Simpler models are more likely to generalize (Occam's razor)

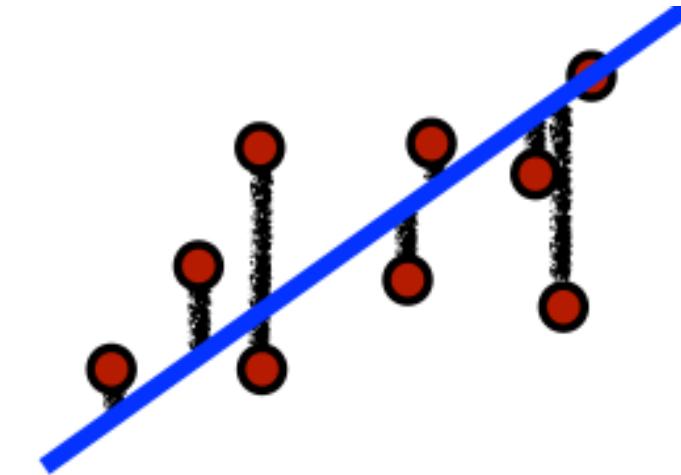
Can we change the problem to penalize for model complexity?

- Intuitively, models with smaller weights are simpler



Given  $n$  training points with  $d$  features, we define:

- $\mathbf{X} \in \mathbb{R}^{n \times d}$ : matrix storing points
- $\mathbf{y} \in \mathbb{R}^n$ : real-valued labels
- $\hat{\mathbf{y}} \in \mathbb{R}^n$ : predicted labels, where  $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$
- $\mathbf{w} \in \mathbb{R}^d$ : regression parameters / model to learn



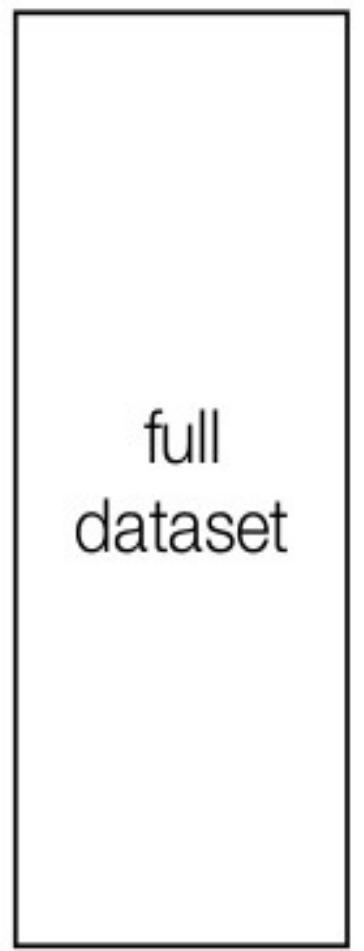
**Ridge Regression:** Learn mapping ( $\mathbf{w}$ ) that minimizes residual sum of squares along with a regularization term:

$$\min_{\mathbf{w}} \frac{\text{Training Error}}{||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2} + \lambda \frac{\text{Model Complexity}}{||\mathbf{w}||_2^2}$$

Closed-form solution:  $\mathbf{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_d)^{-1} \mathbf{X}^\top \mathbf{y}$

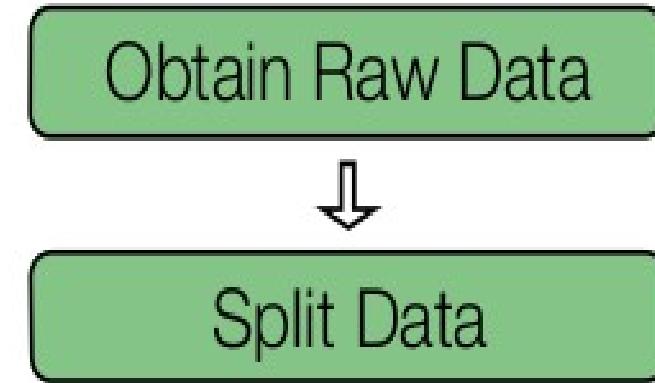
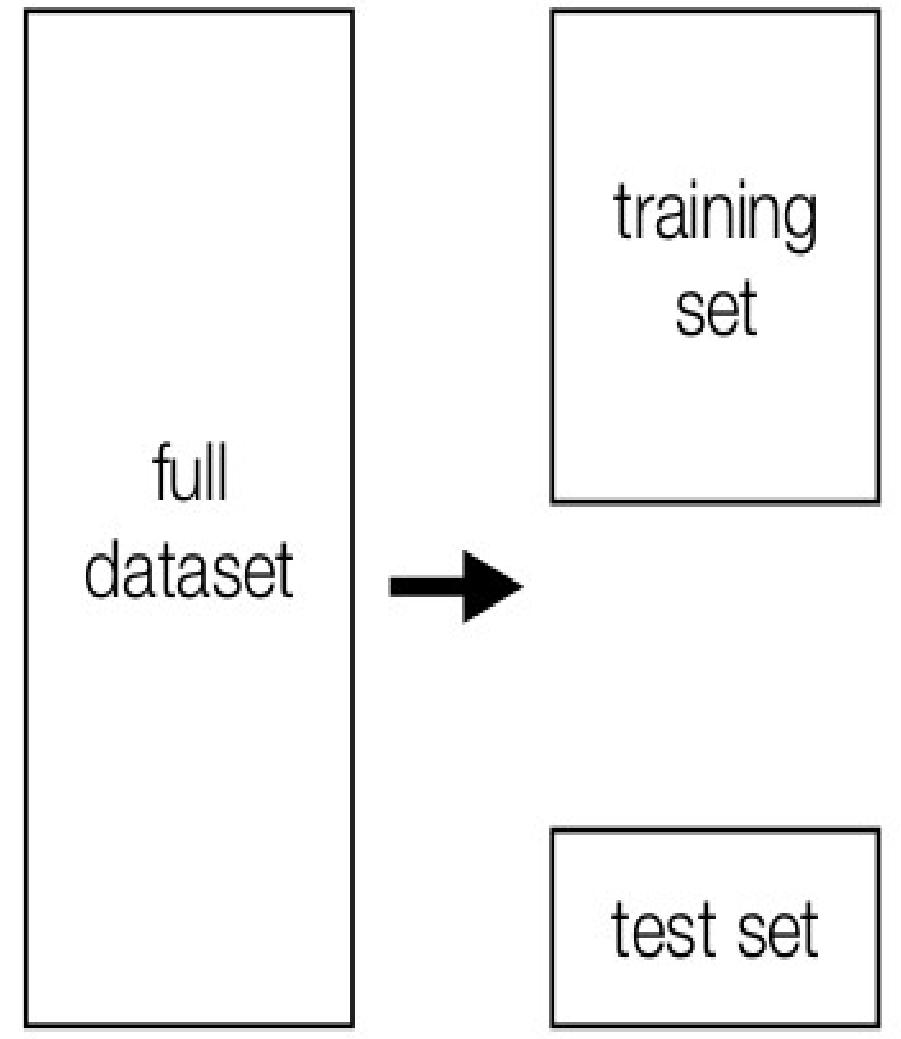
free parameter trades off  
between training error and  
model complexity

# **Millionsong Regression Pipeline**

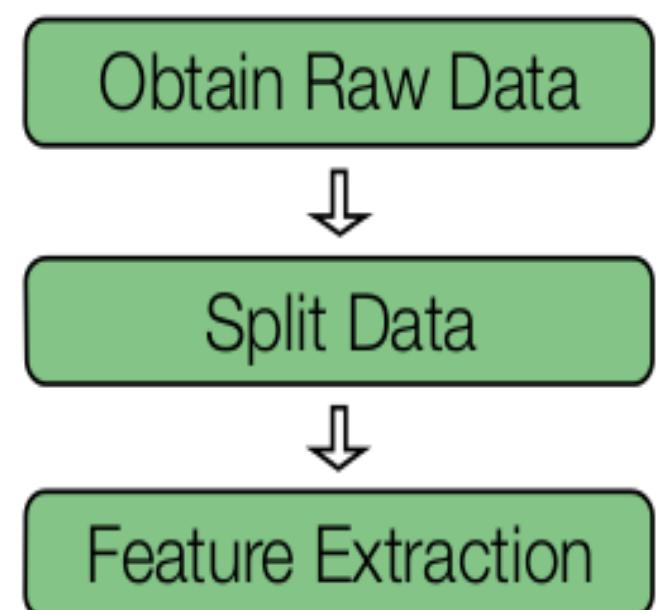
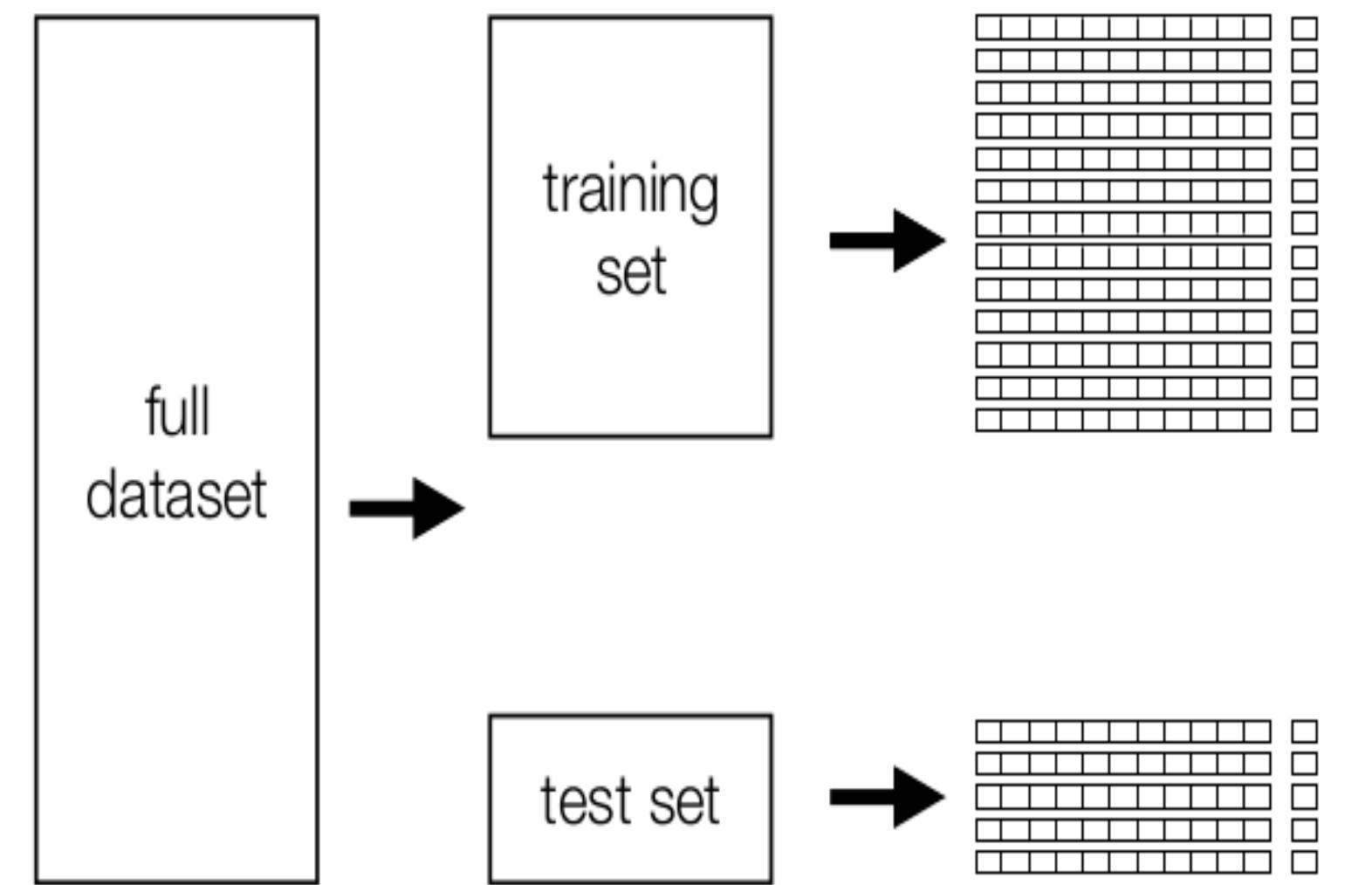


Obtain Raw Data

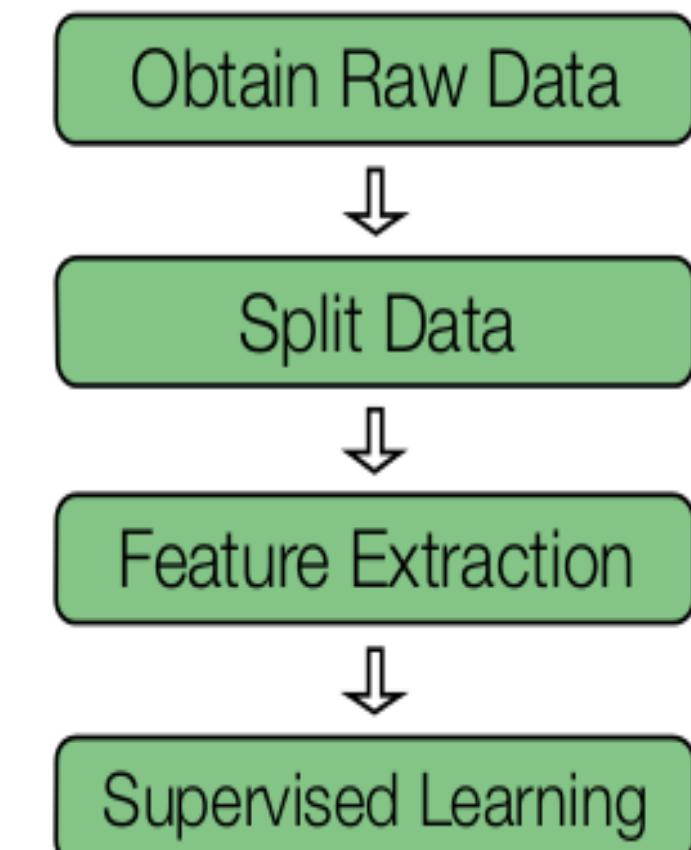
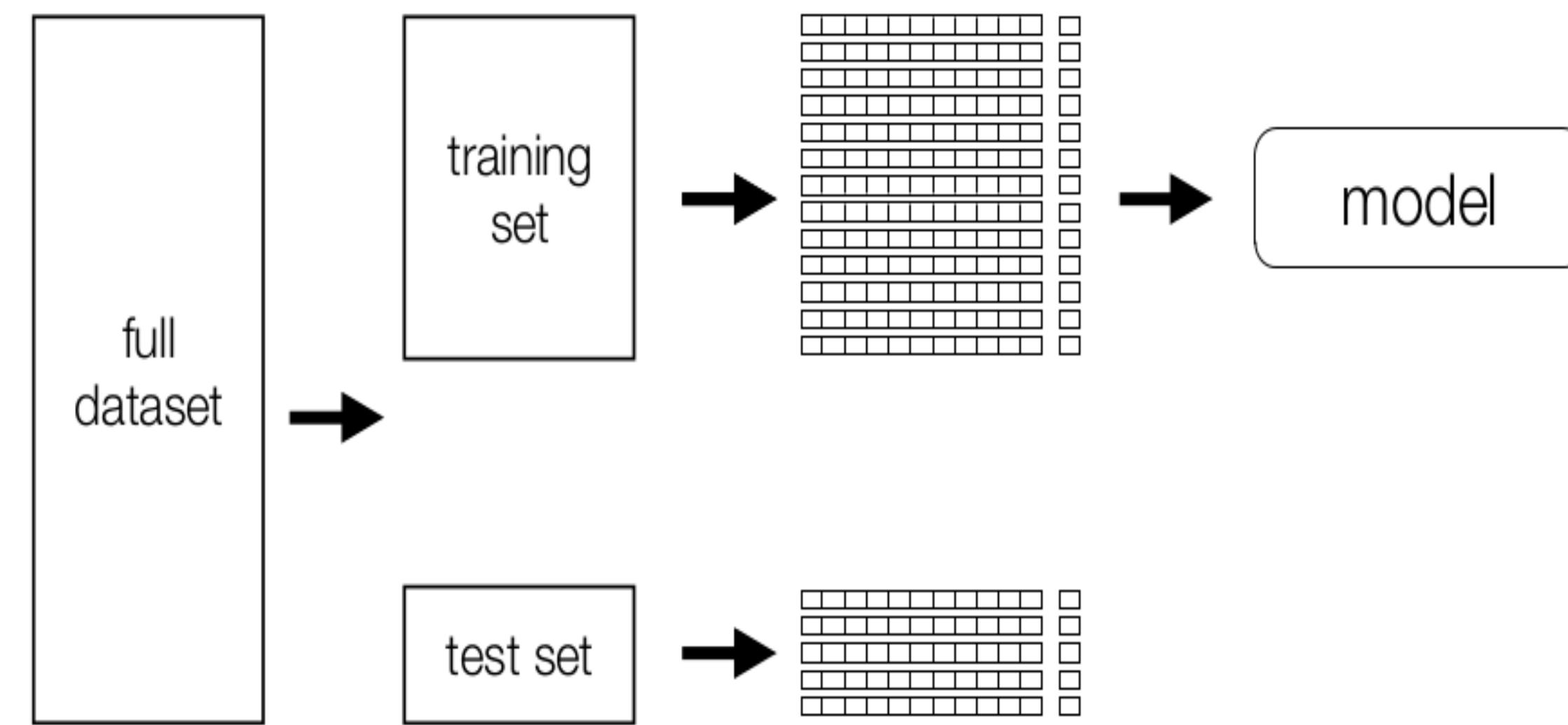
# Supervised Learning Pipeline



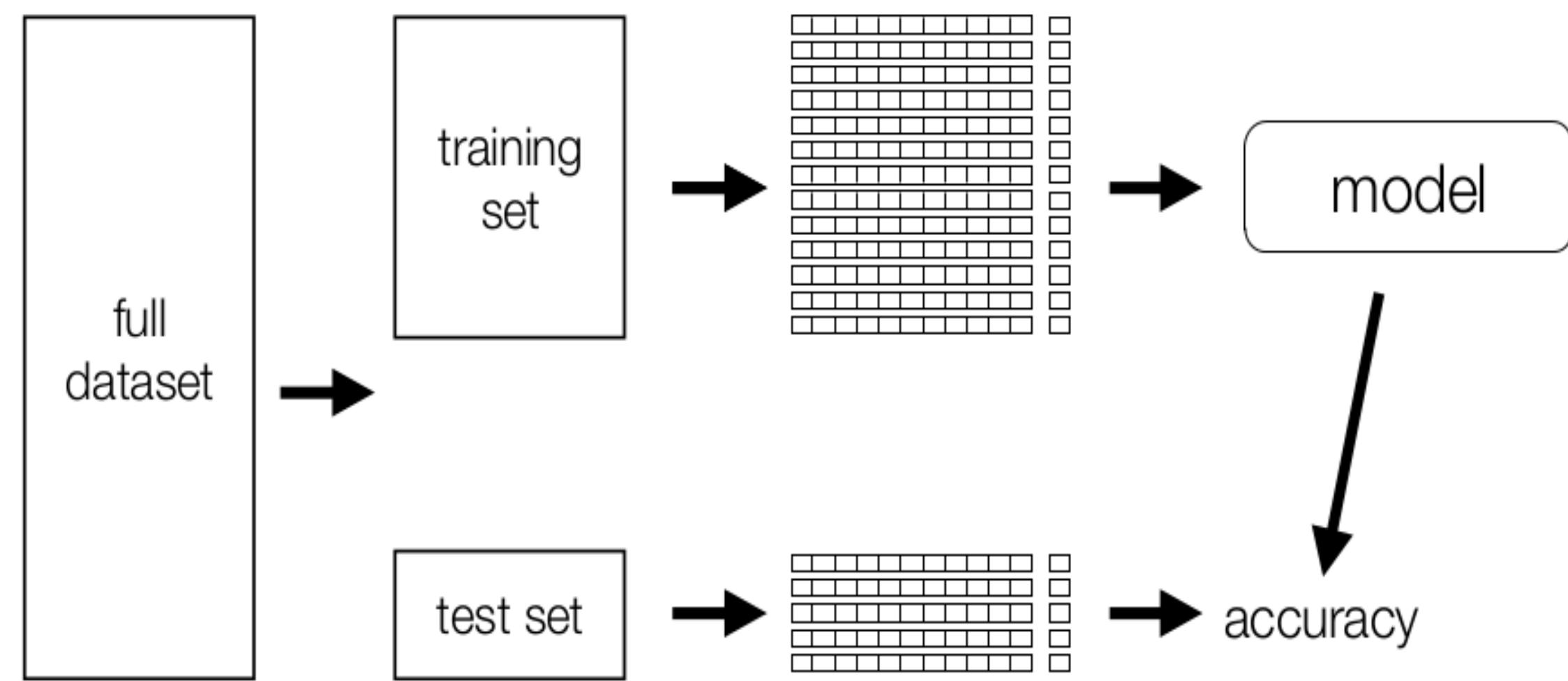
# Supervised Learning Pipeline



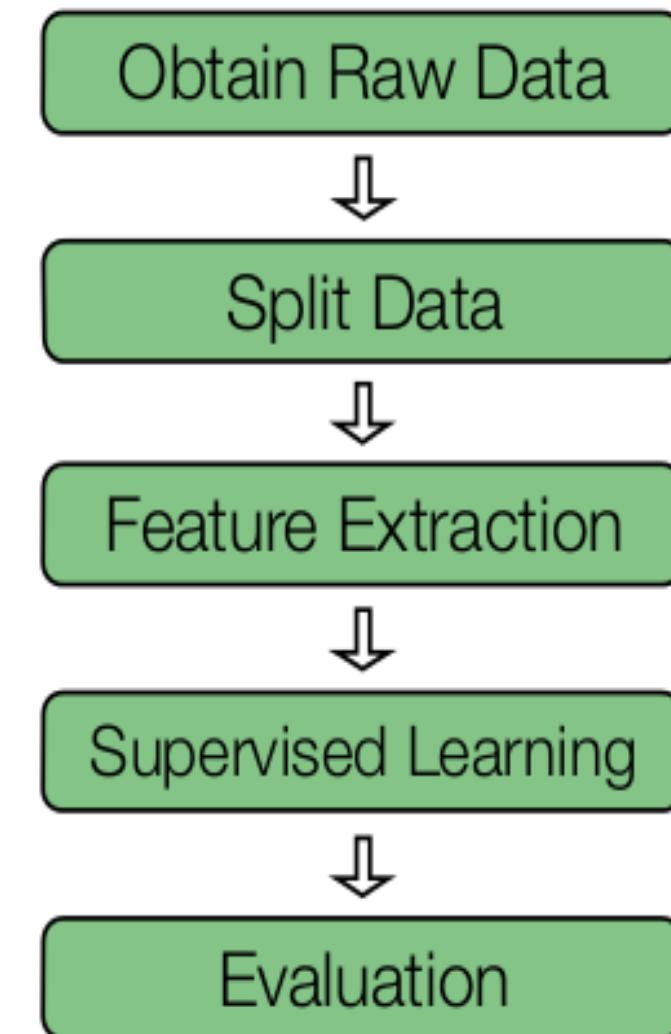
# Supervised Learning Pipeline

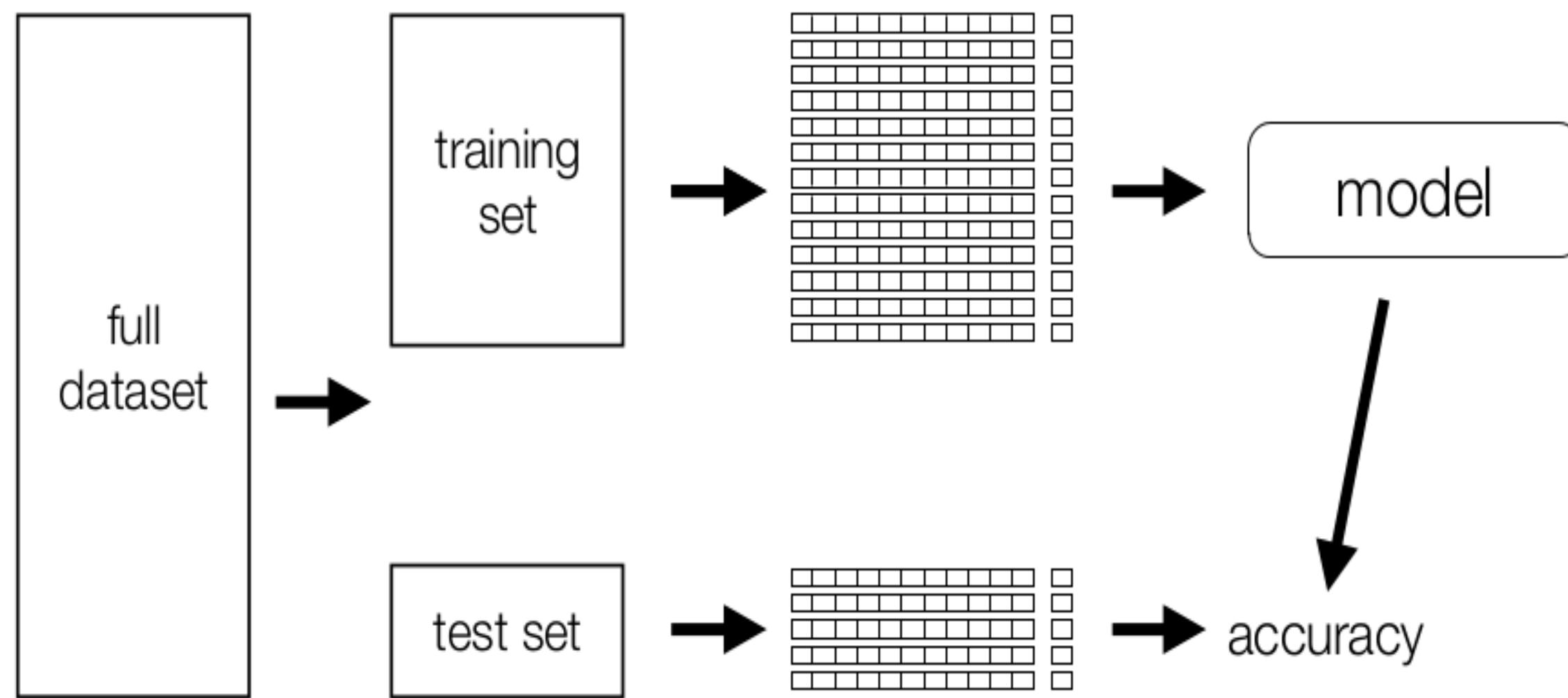


# Supervised Learning Pipeline

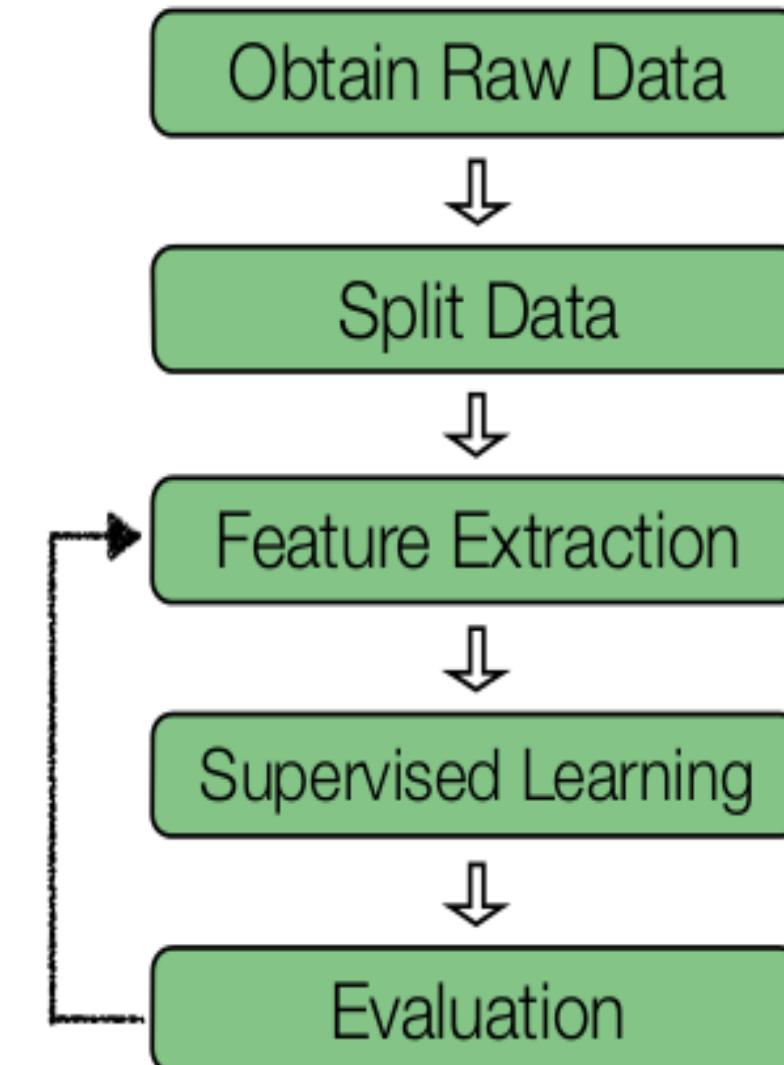


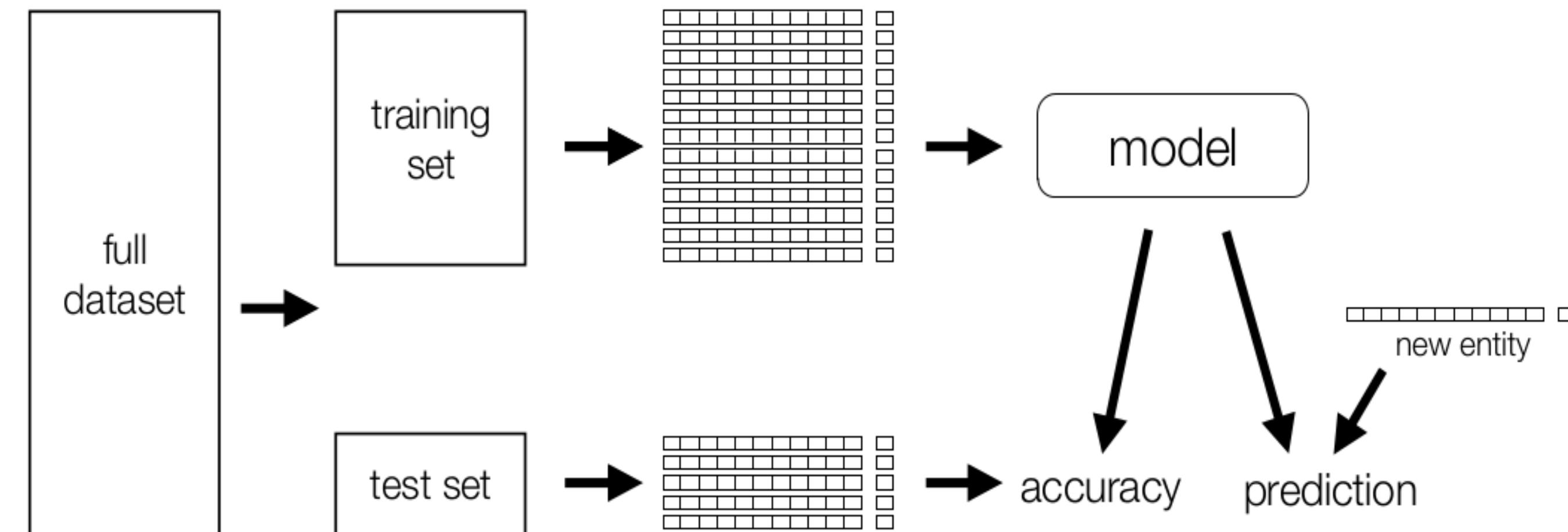
# Supervised Learning Pipeline



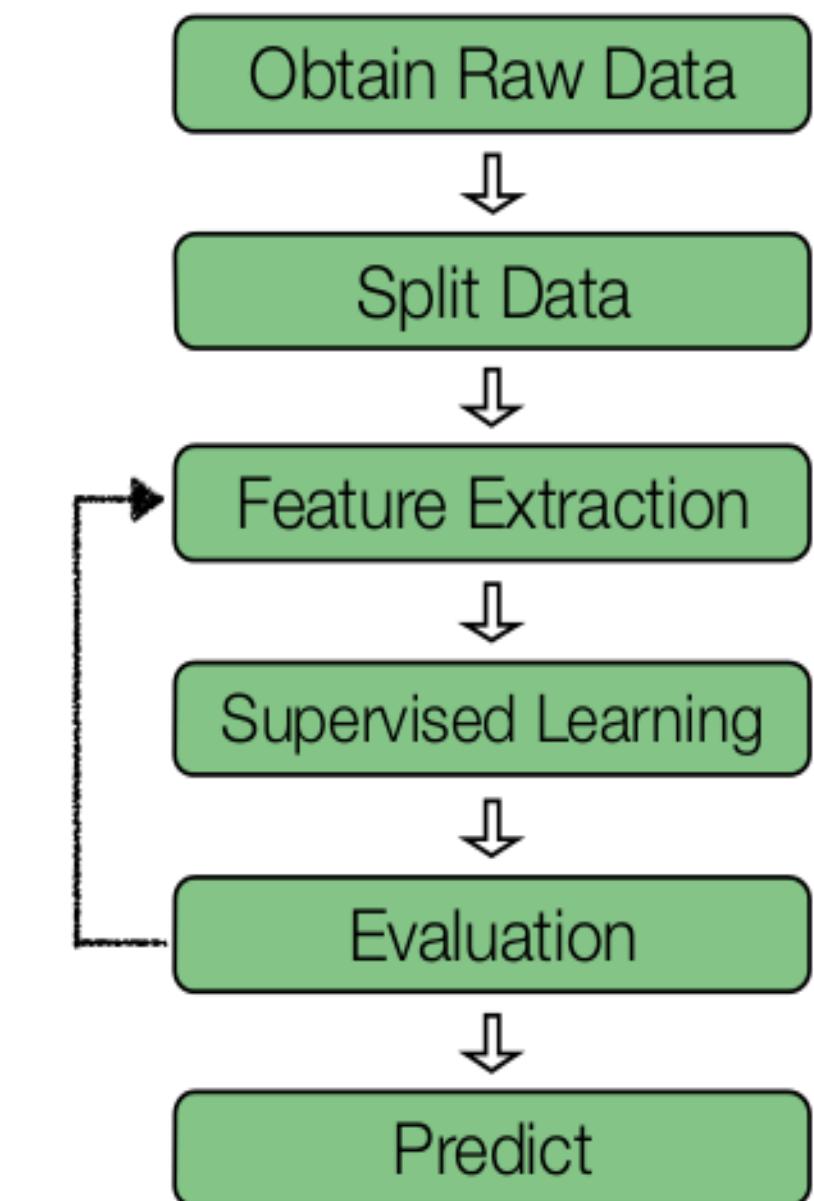


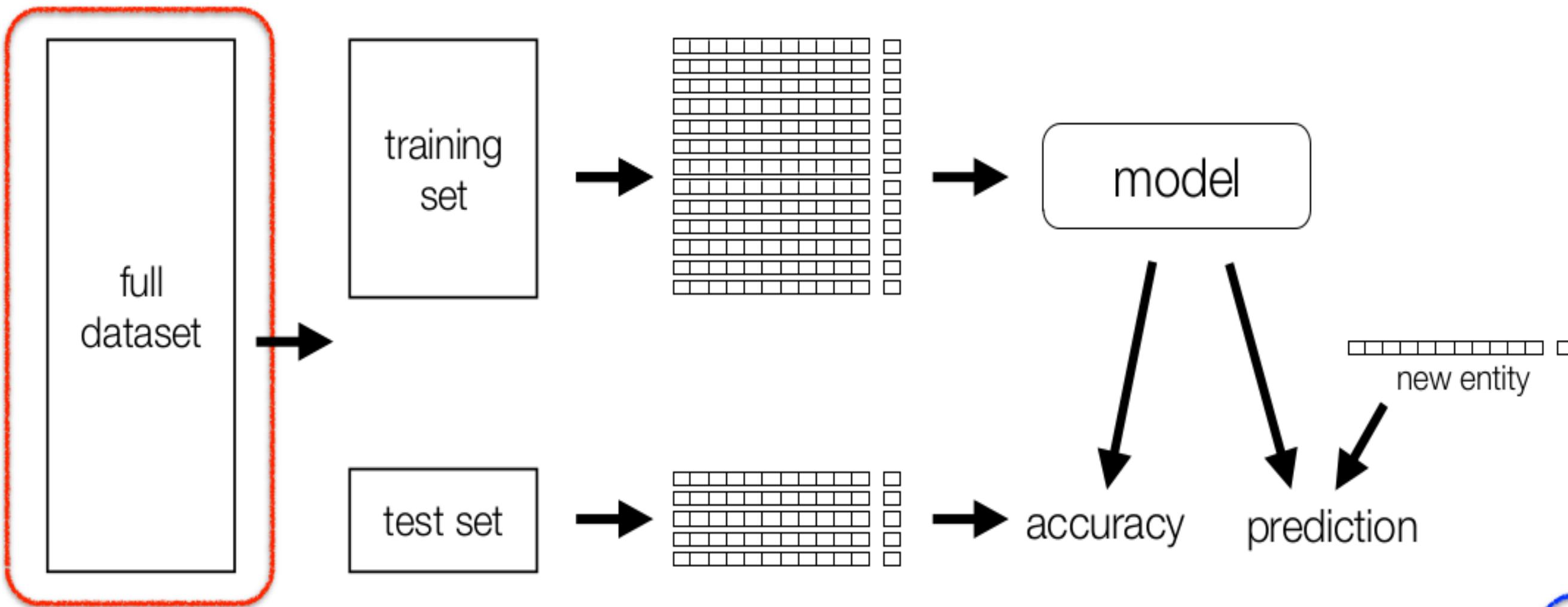
# Supervised Learning Pipeline





# Supervised Learning Pipeline

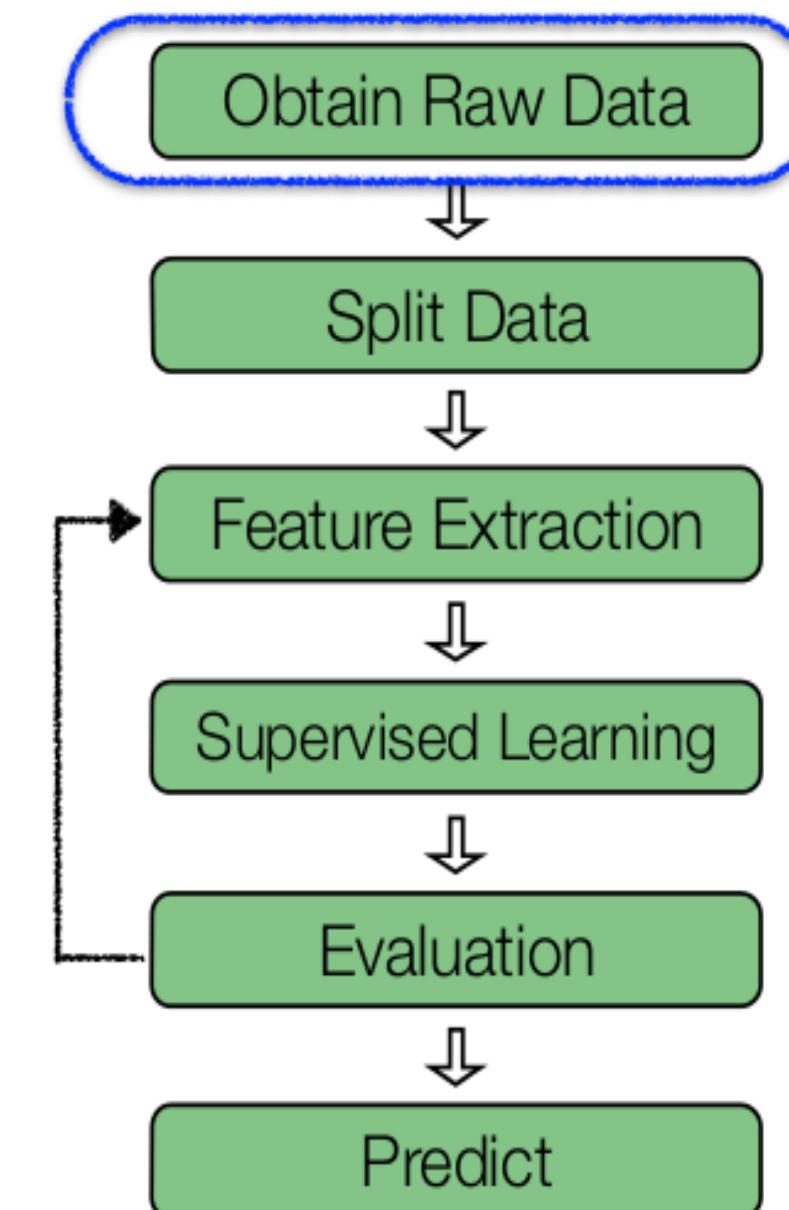


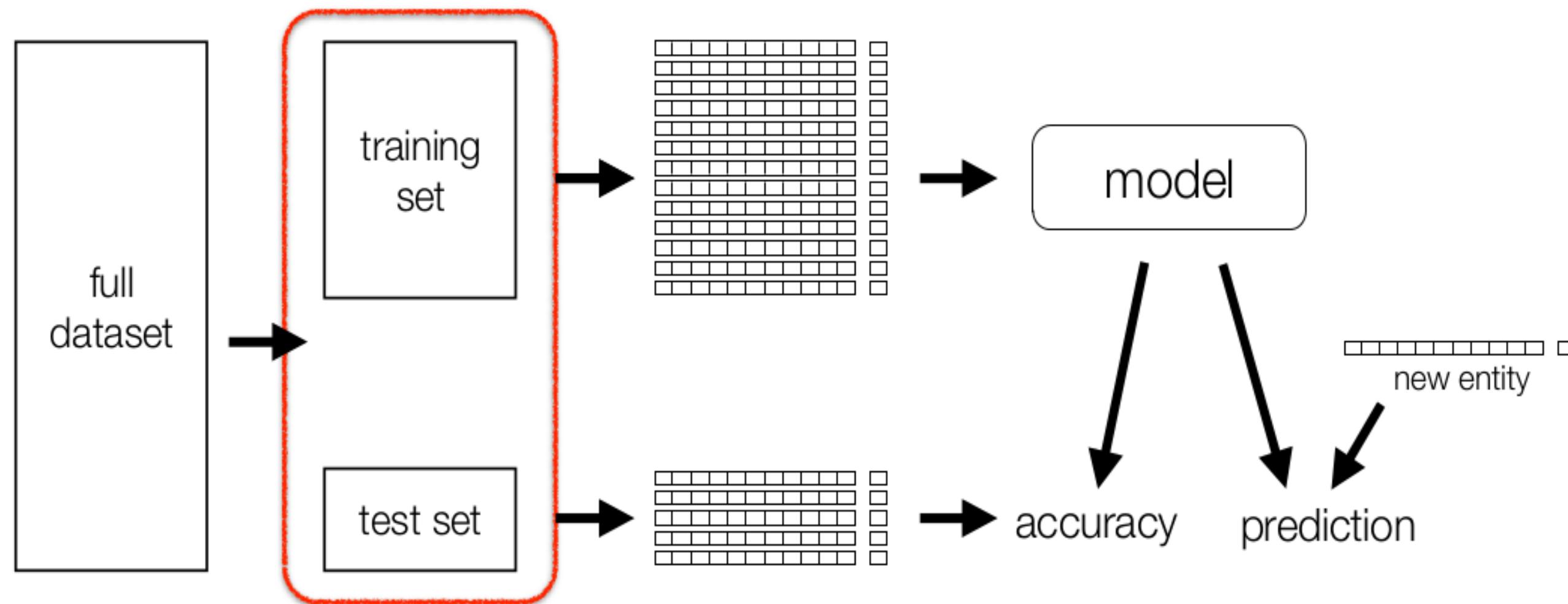


**Goal:** Predict song's release year from audio features

**Raw Data:** Millionsong Dataset from UCI ML Repository

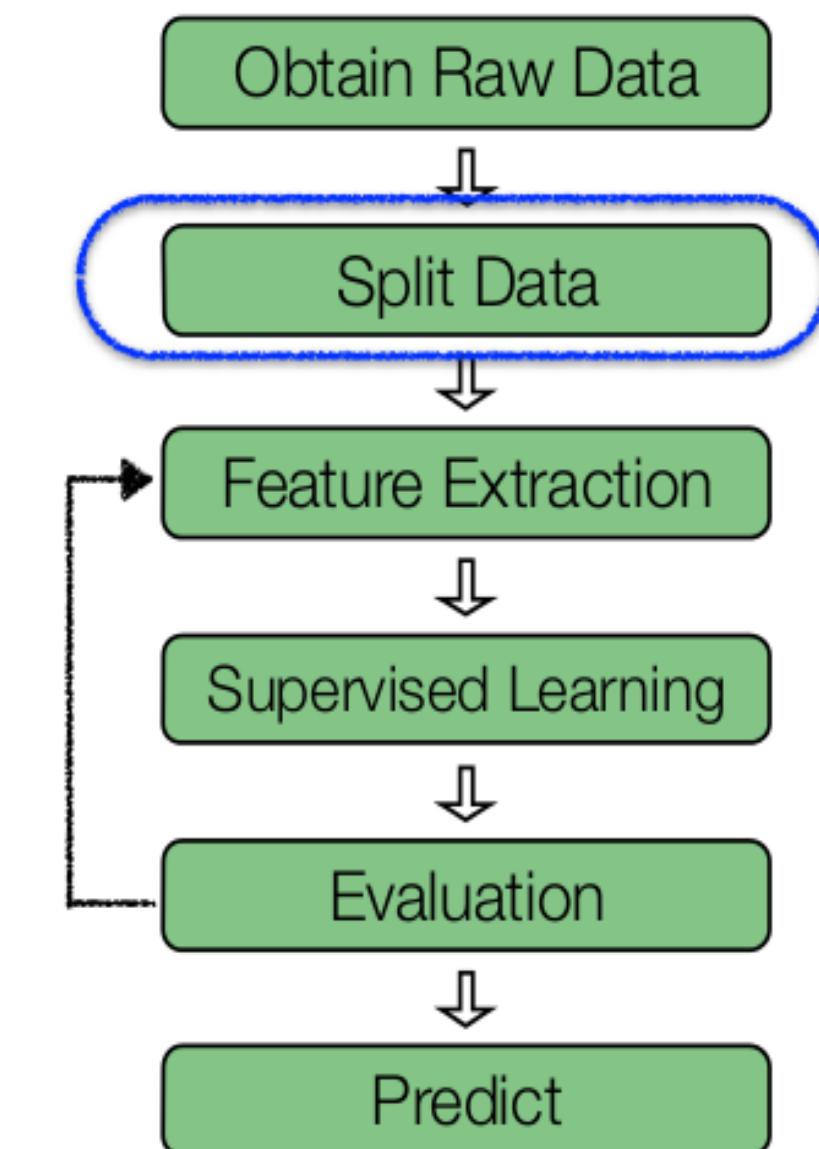
- Western, commercial tracks from 1980-2014
- 12 timbre averages (features) and release year (label)

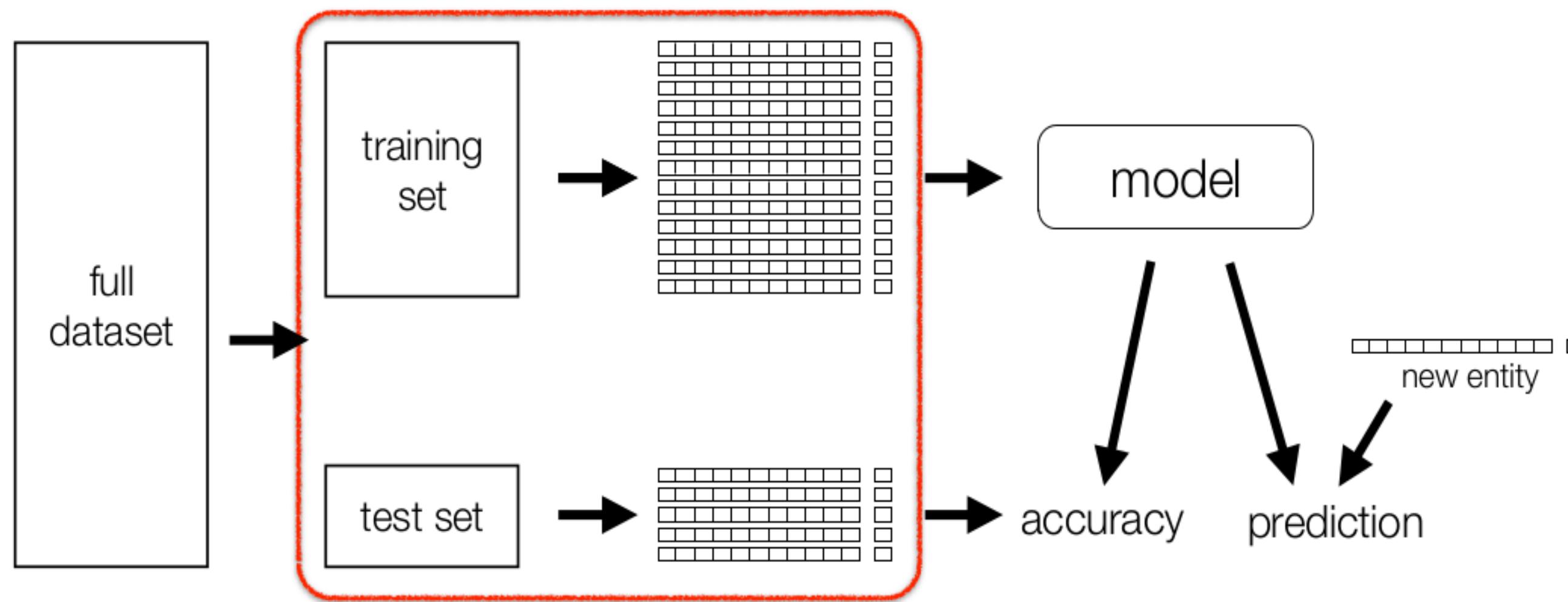




**Split Data:** Train on training set, evaluate with test set

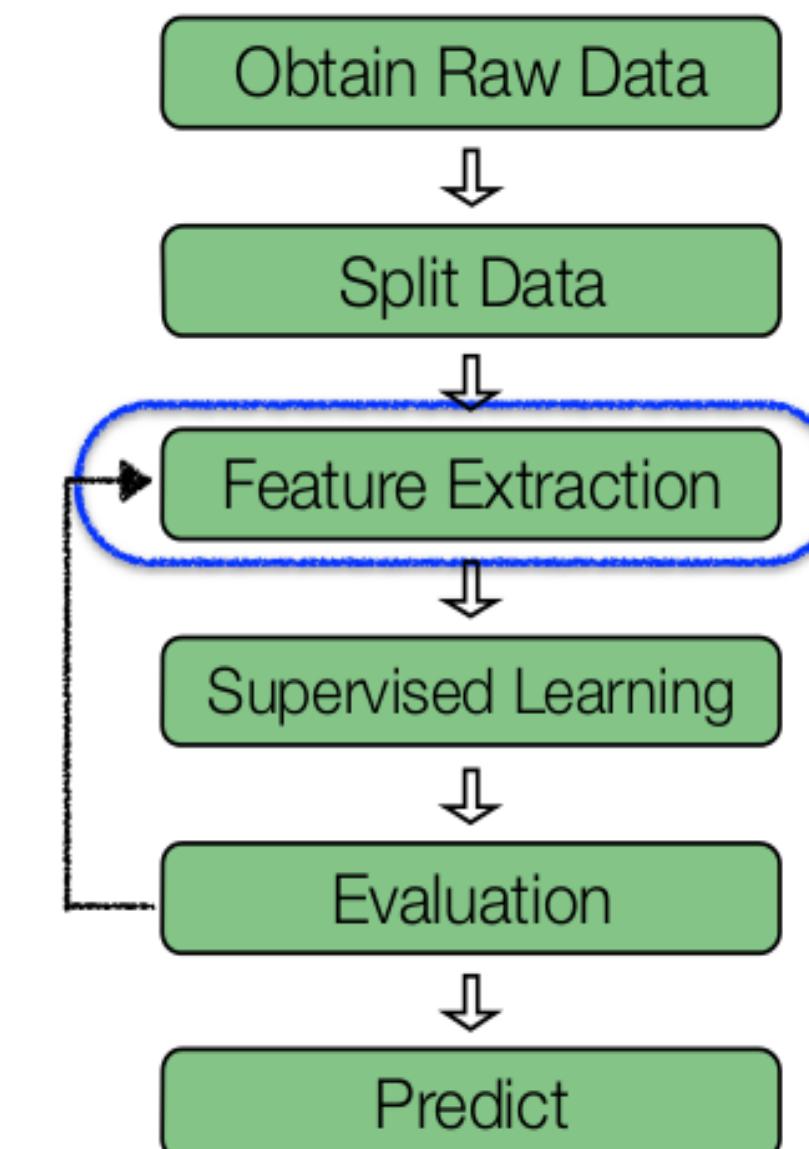
- Test set simulates unobserved data
- Test error tells us whether we've generalized well





## Feature Extraction: Quadratic features

- Compute pairwise feature interactions
  - Captures covariance of initial timbre features
  - Leads to a non-linear model relative to raw features



Given 2 dimensional data, quadratic features are:

$$\mathbf{x} = [x_1 \quad x_2]^\top \implies \Phi(\mathbf{x}) = [x_1^2 \quad x_1x_2 \quad x_2x_1 \quad x_2^2]^\top$$

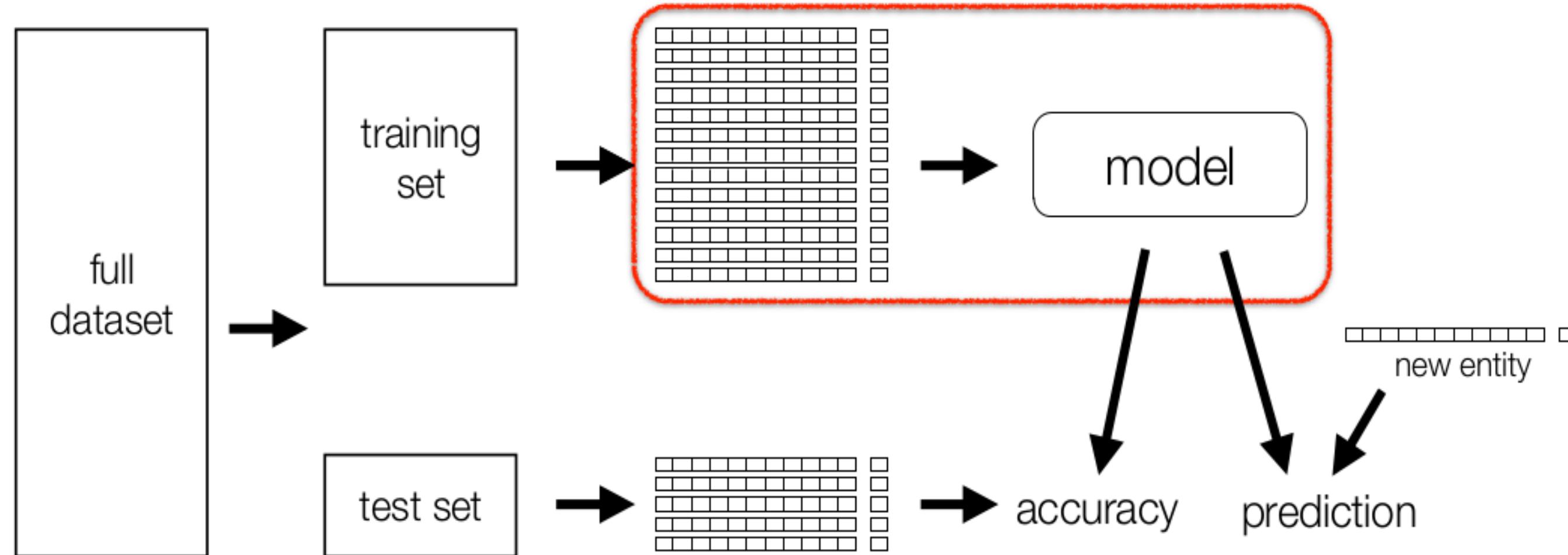
$$\mathbf{z} = [z_1 \quad z_2]^\top \implies \Phi(\mathbf{z}) = [z_1^2 \quad z_1z_2 \quad z_2z_1 \quad z_2^2]^\top$$

More succinctly:

$$\Phi'(\mathbf{x}) = [x_1^2 \quad \sqrt{2}x_1x_2 \quad x_2^2]^\top \quad \Phi'(\mathbf{z}) = [z_1^2 \quad \sqrt{2}z_1z_2 \quad z_2^2]^\top$$

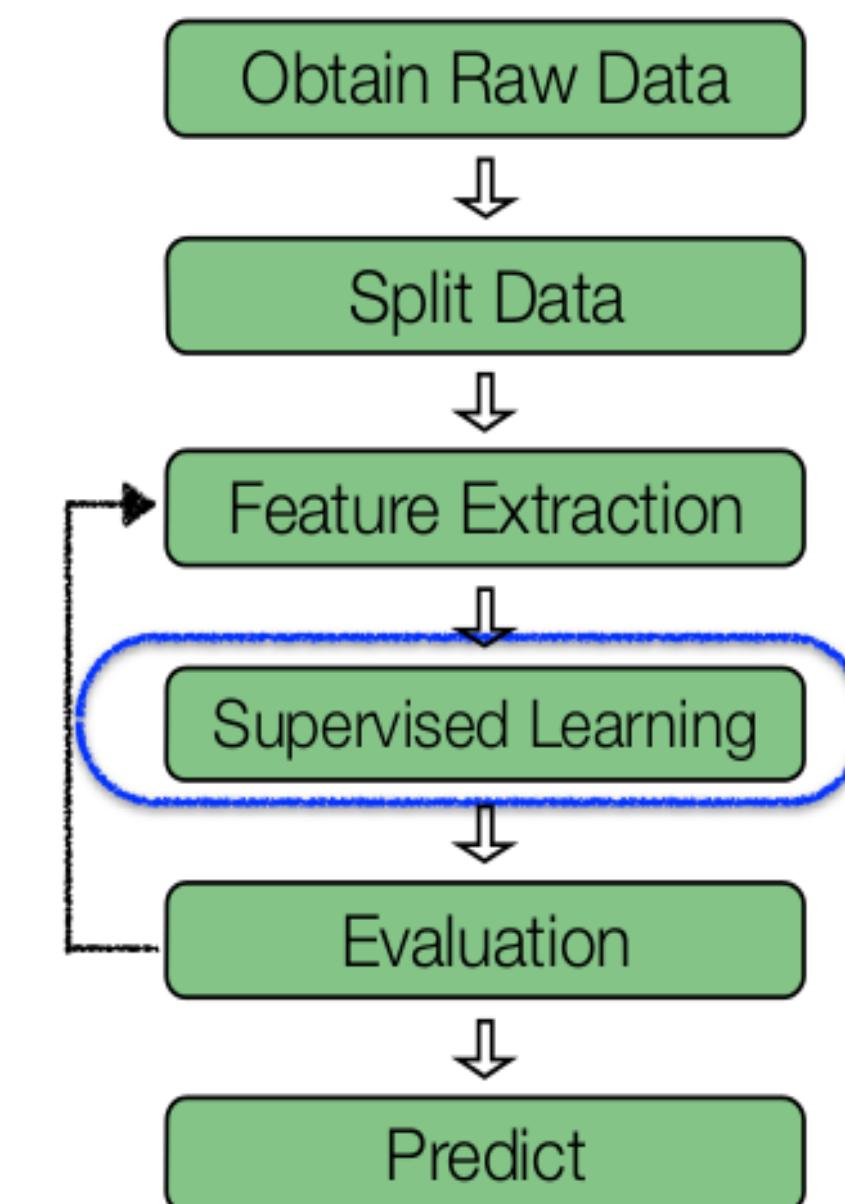
Equivalent inner products:

$$\Phi(\mathbf{x})^\top \Phi(\mathbf{z}) = x_1^2 z_1^2 + 2x_1 x_2 z_1 z_2 + x_2^2 z_2^2 = \Phi'(\mathbf{x})^\top \Phi'(\mathbf{z})$$



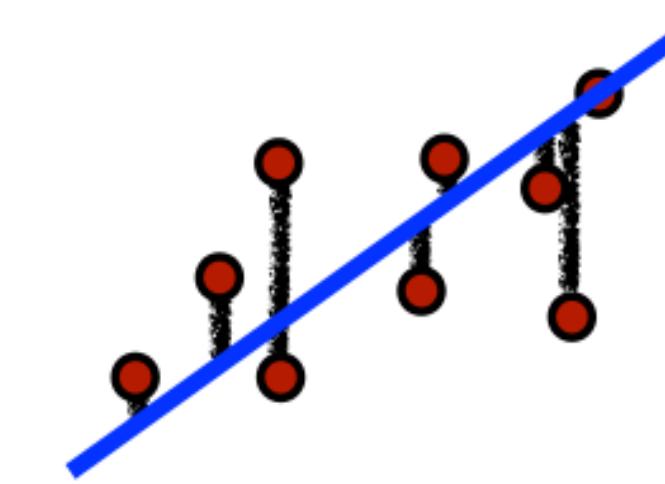
## Supervised Learning: Least Squares Regression

- Learn a mapping from entities to continuous labels given a training set
- Audio features → Song year



Given  $n$  training points with  $d$  features, we define:

- $\mathbf{X} \in \mathbb{R}^{n \times d}$ : matrix storing points
- $\mathbf{y} \in \mathbb{R}^n$ : real-valued labels
- $\hat{\mathbf{y}} \in \mathbb{R}^n$ : predicted labels, where  $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$
- $\mathbf{w} \in \mathbb{R}^d$ : regression parameters / model to learn



**Ridge Regression:** Learn mapping ( $\mathbf{w}$ ) that minimizes residual sum of squares along with a regularization term:

$$\min_{\mathbf{w}} \frac{\text{Training Error}}{\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2} + \frac{\text{Model Complexity}}{\lambda \|\mathbf{w}\|_2^2}$$

Closed-form solution:  $\mathbf{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_d)^{-1} \mathbf{X}^\top \mathbf{y}$

**Ridge Regression:** Learn mapping ( $\mathbf{w}$ ) that minimizes residual sum of squares along with a regularization term:

$$\min_{\mathbf{w}} \frac{\text{Training Error}}{\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2} + \frac{\text{Model Complexity}}{\lambda \|\mathbf{w}\|_2^2}$$

free parameter trades off between training error and model complexity

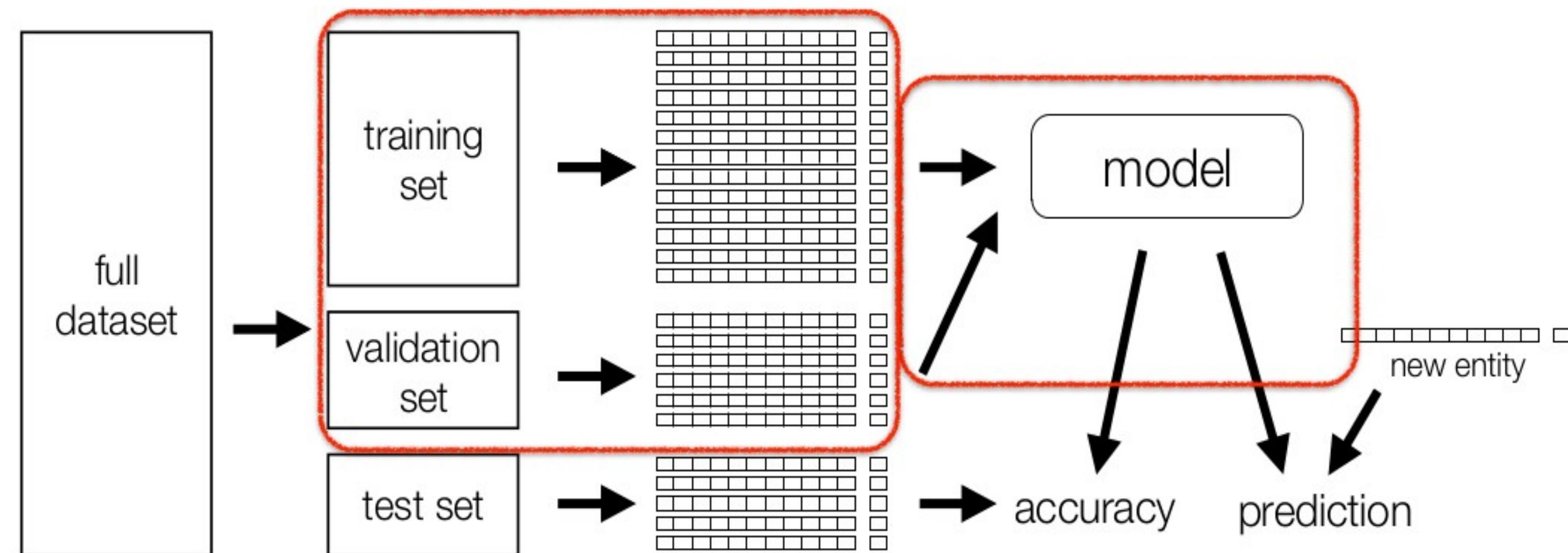
How do we choose a good value for this free parameter?

- Most methods have free parameters / ‘hyperparameters’ to tune

First thought: Search over multiple values, evaluate each on test set

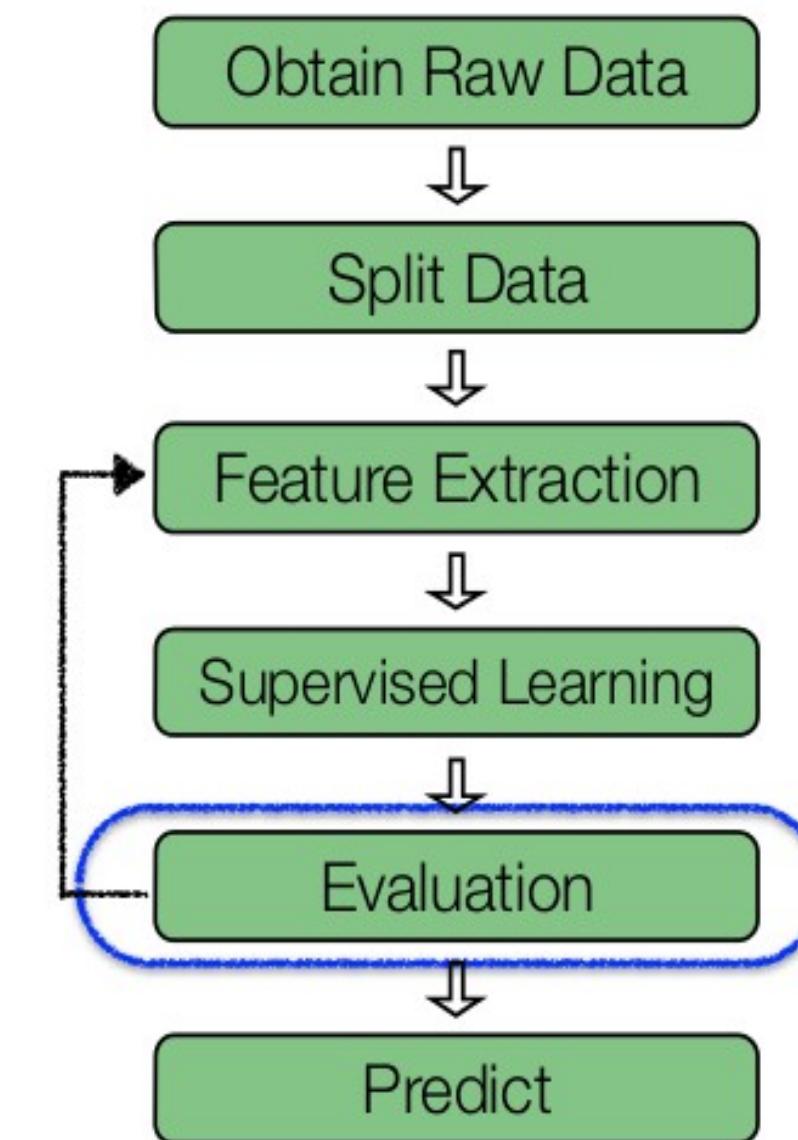
- But, goal of test set is to simulate unobserved data
- We may overfit if we use it to choose hyperparameters

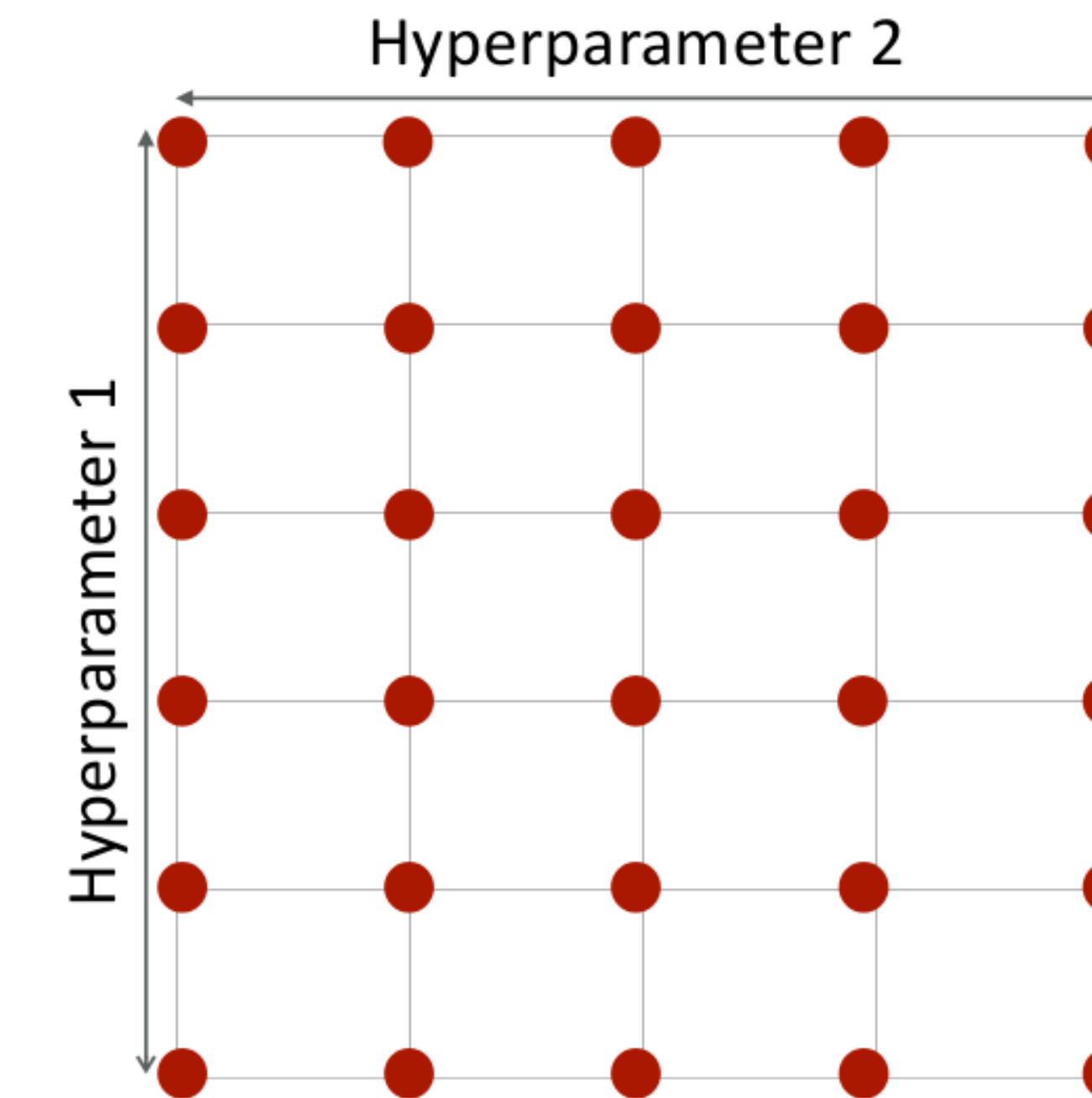
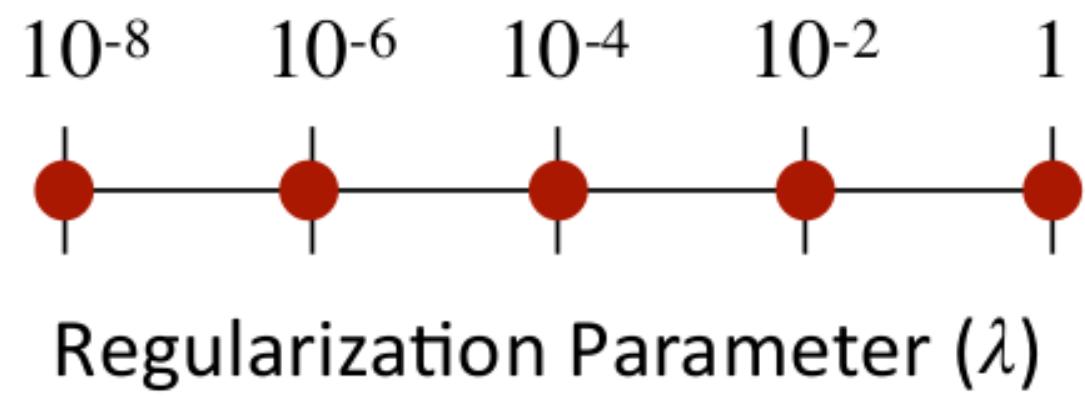
Second thought: **Create another hold out dataset for this search**



## Evaluation (Part 1): Hyperparameter tuning

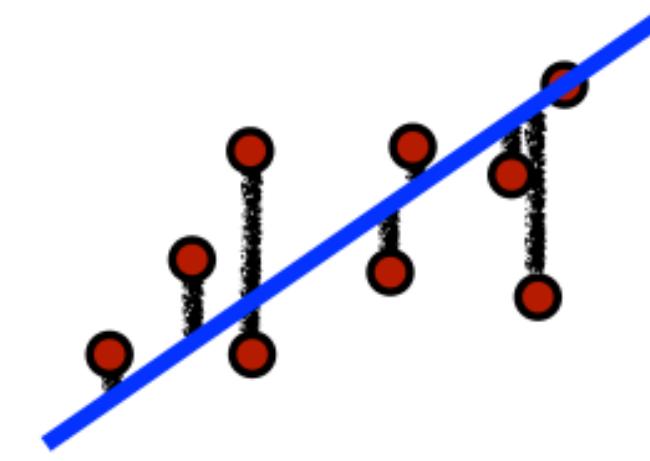
- *Training*: train various models
- *Validation*: evaluate various models (e.g., Grid Search)
- *Test*: evaluate final model's accuracy





- Grid Search:** Exhaustively search through hyperparameter space
- Define and discretize search space (linear or log scale)
  - Evaluate points via validation error

# Evaluating Predictions



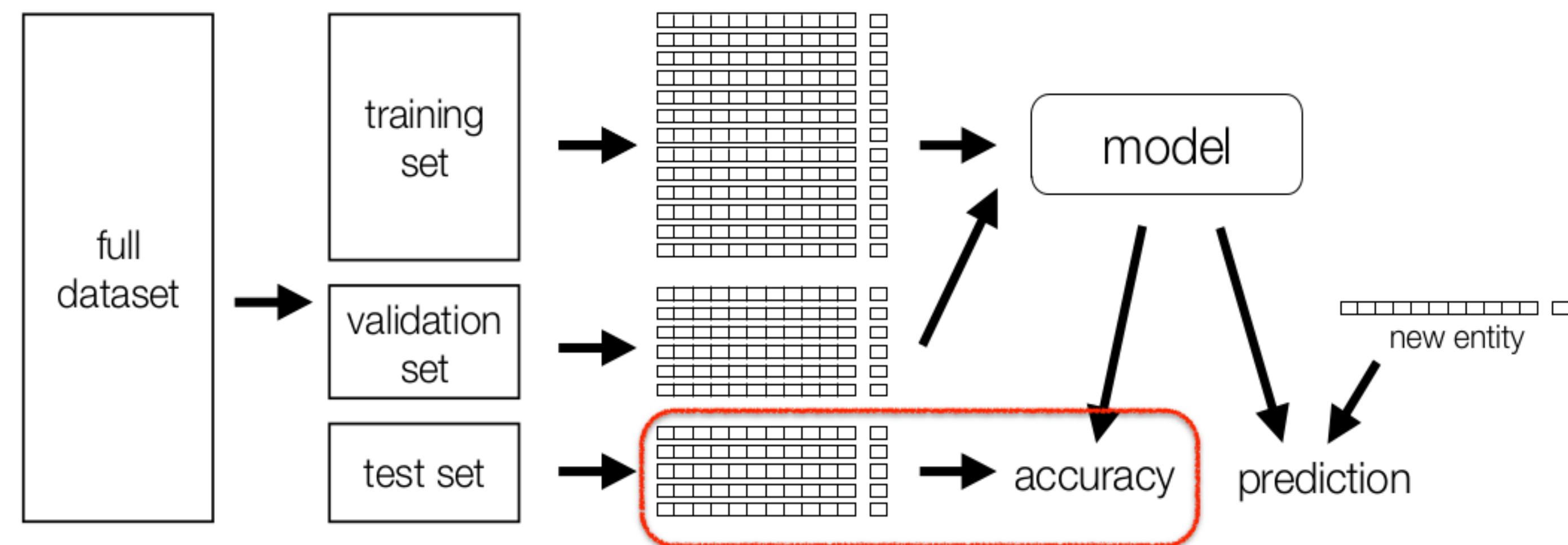
How can we compare labels and predictions for  $n$  validation points?

Least squares optimization involves squared loss,  $(y - \hat{y})^2$ , so it seems reasonable to use mean squared error (**MSE**):

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (\hat{y}^{(i)} - y^{(i)})^2$$

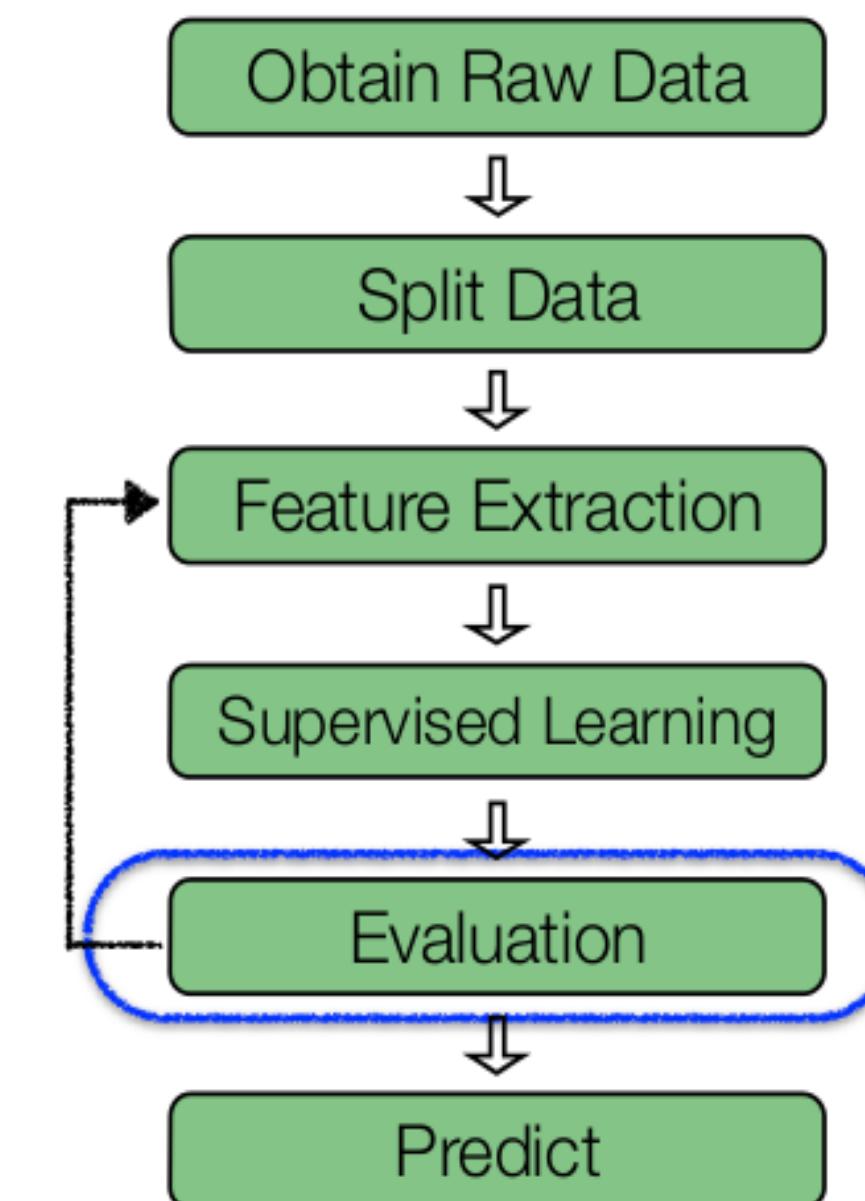
But MSE's unit of measurement is square of quantity being measured, e.g., "squared years" for song prediction

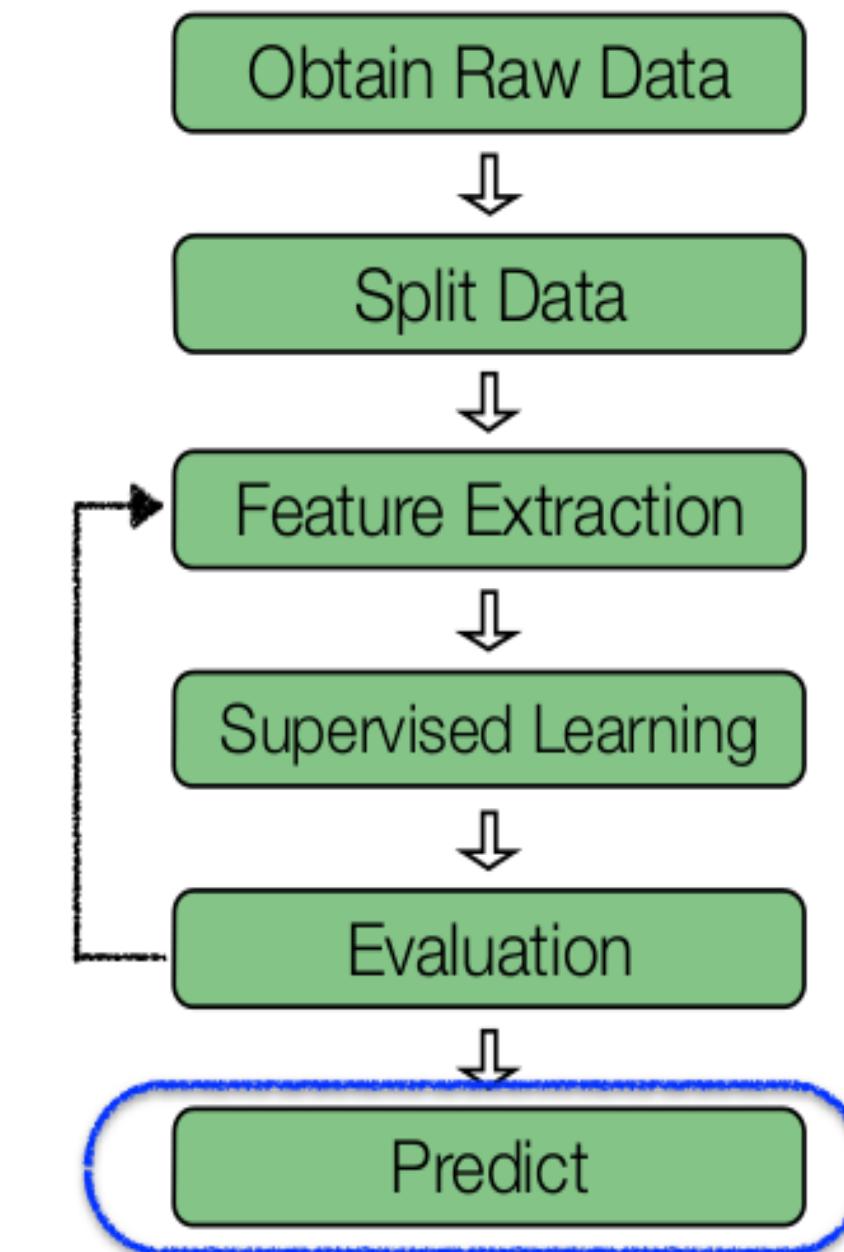
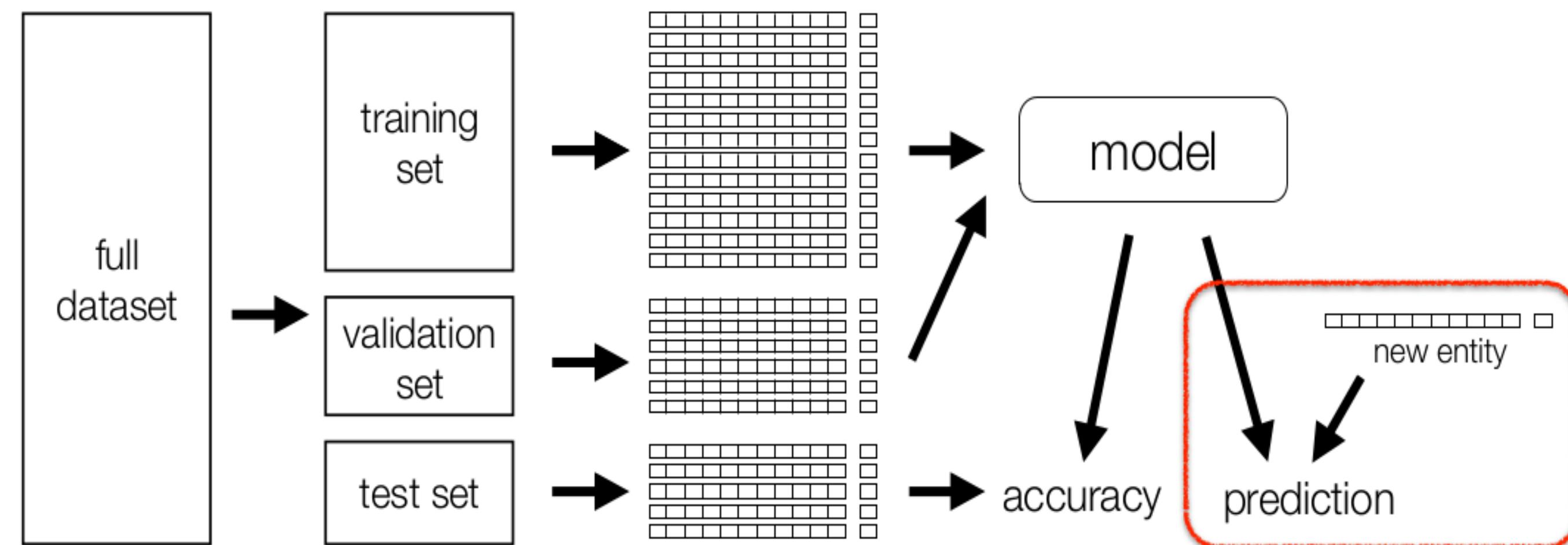
More natural to use root-mean-square error (**RMSE**), i.e.,  $\sqrt{\text{MSE}}$



## Evaluation (Part 2): Evaluate final model

- Training set: train various models
- Validation set: evaluate various models
- *Test set*: evaluate final model's accuracy



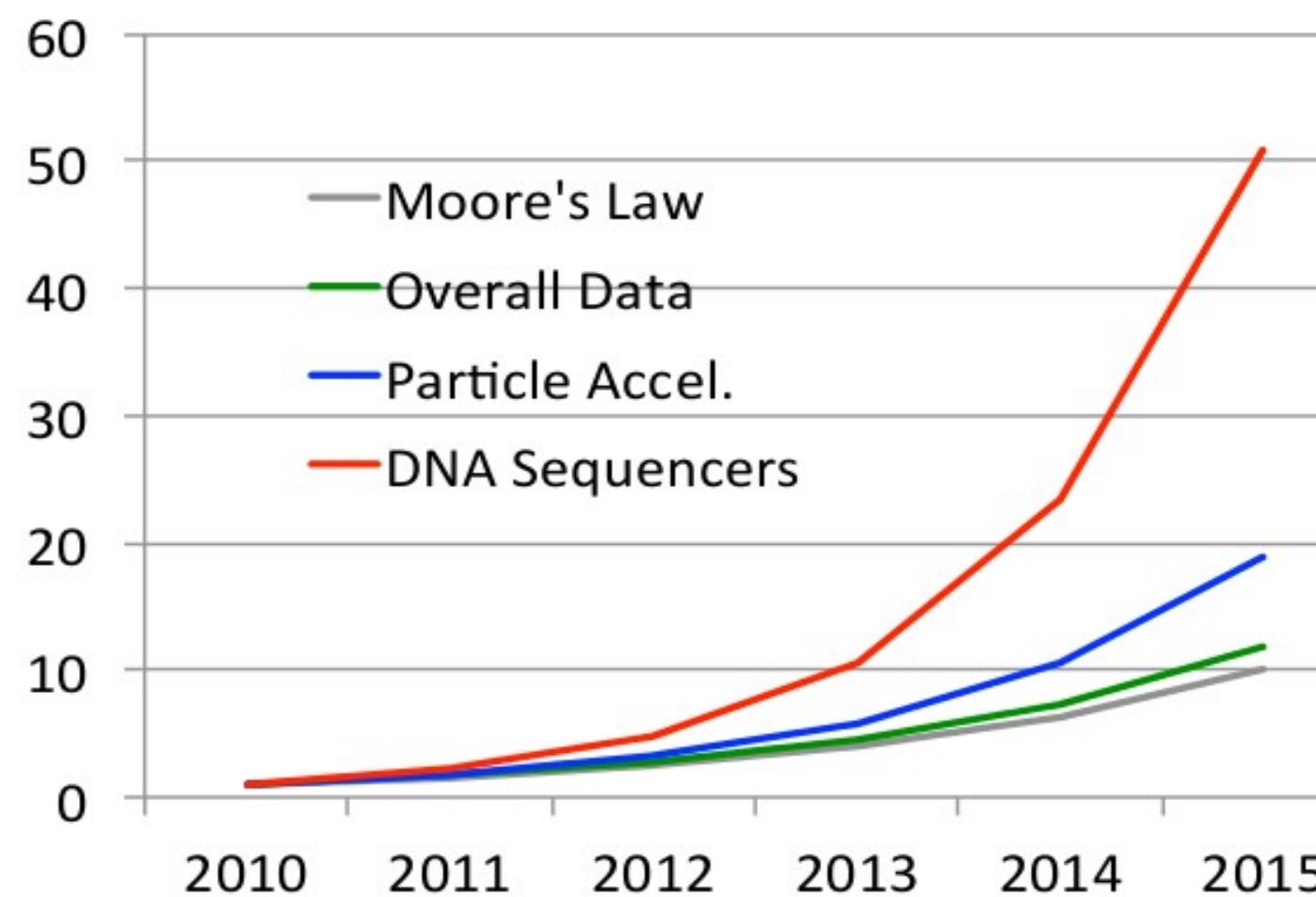


**Predict:** Final model can then be used to make predictions on future observations, e.g., new songs

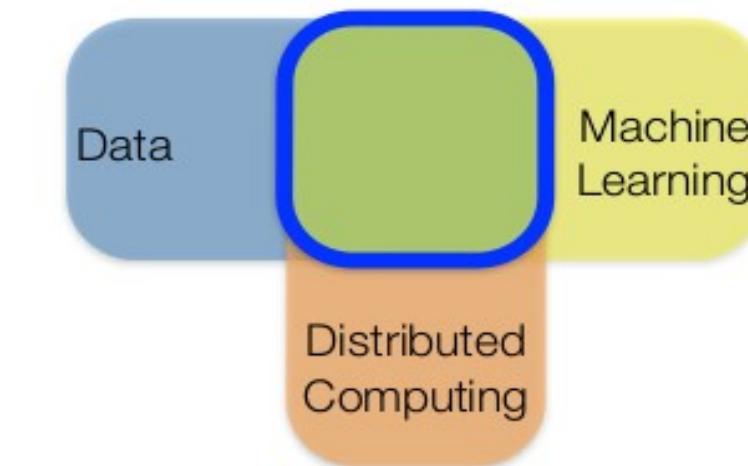
# **Distributed ML: Computation and Storage**

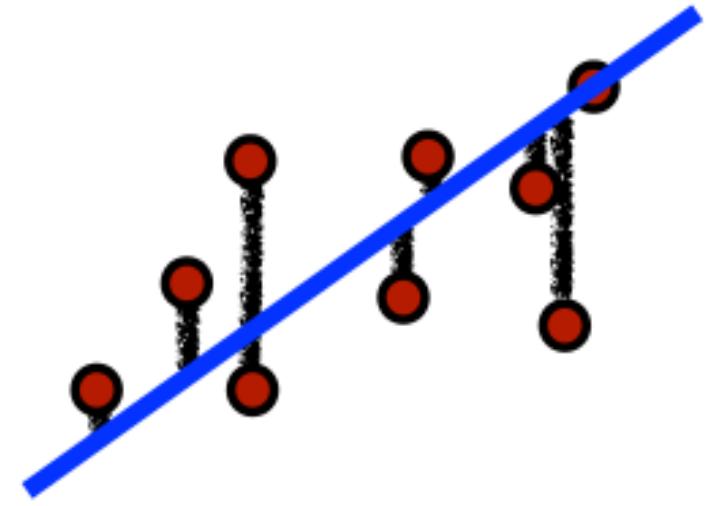
# Challenge: Scalability

Classic ML techniques are not always suitable for modern datasets



Data Grows Faster  
than Moore's Law  
[IDC report, Kathy Yellick, LBNL]





**Least Squares Regression:** Learn mapping ( $\mathbf{w}$ ) from features to labels that minimizes residual sum of squares:

$$\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

Closed form solution:  $\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$  (if inverse exists)

How do we solve this computationally?

- Computational profile similar for Ridge Regression

# Computing Closed Form Solution

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

**Computation:**  $O(nd^2 + d^3)$  operations

Consider number of arithmetic operations ( +, -,  $\times$ ,  $/$  )

Computational bottlenecks:

- Matrix multiply of  $\mathbf{X}^\top \mathbf{X}$  :  $O(nd^2)$  operations
- Matrix inverse:  $O(d^3)$  operations

Other methods (Cholesky, QR, SVD) have same complexity

# Storage Requirements

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

**Computation:**  $\mathcal{O}(nd^2 + d^3)$  operations

**Storage:**  $\mathcal{O}(nd + d^2)$  floats

Consider storing values as floats (8 bytes)

Storage bottlenecks:

- $\mathbf{X}^\top \mathbf{X}$  and its inverse:  $\mathcal{O}(d^2)$  floats
- $\mathbf{X}$  :  $\mathcal{O}(nd)$  floats

# Big $n$ and Small $d$

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

**Computation:**  $O(\underline{nd^2} + d^3)$  operations

**Storage:**  $O(\underline{nd} + d^2)$  floats

Assume  $O(d^3)$  computation and  $O(d^2)$  storage feasible on single machine

Storing  $\mathbf{X}$  and computing  $\mathbf{X}^\top \mathbf{X}$  are the bottlenecks

Can distribute storage and computation!

- Store data points (rows of  $\mathbf{X}$ ) across machines
- Compute  $\mathbf{X}^\top \mathbf{X}$  as a sum of outer products

# Matrix Multiplication via Inner Products

Each entry of output matrix is result of inner product of inputs matrices

$$\begin{bmatrix} 9 & 3 & 5 \\ 4 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & -5 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 28 \\ \end{bmatrix}$$

$$9 \times 1 + 3 \times 3 + 5 \times 2 = 28$$

# Matrix Multiplication via Inner Products

Each entry of output matrix is result of inner product of inputs matrices

$$\begin{bmatrix} 9 & 3 & 5 \\ 4 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 28 & 18 \end{bmatrix}$$

# Matrix Multiplication via Inner Products

Each entry of output matrix is result of inner product of inputs matrices

$$\begin{bmatrix} 9 & 3 & 5 \\ 4 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 28 & 18 \\ 11 & 9 \end{bmatrix}$$

# Matrix Multiplication via Outer Products

Output matrix is **sum of outer products** between corresponding rows and columns of input matrices

$$\begin{bmatrix} 9 \\ 4 \end{bmatrix} \begin{bmatrix} 3 & 5 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & -5 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} \quad & \quad \\ \quad & \quad \end{bmatrix}$$

$$\begin{bmatrix} 9 & 18 \\ 4 & 8 \end{bmatrix}$$

# Matrix Multiplication via Outer Products

Output matrix is **sum of outer products** between corresponding rows and columns of input matrices

$$\begin{bmatrix} 9 & \boxed{3} & 5 \\ 4 & \boxed{1} & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ \boxed{3} & -5 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} & \\ & \end{bmatrix}$$

$$\begin{bmatrix} 9 & 18 \\ 4 & 8 \end{bmatrix} + \begin{bmatrix} 9 & -15 \\ 3 & -5 \end{bmatrix}$$

# Matrix Multiplication via Outer Products

Output matrix is **sum of outer products** between corresponding rows and columns of input matrices

$$\begin{bmatrix} 9 & 3 & \boxed{5} \\ 4 & 1 & \boxed{2} \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & -5 \\ \boxed{2} & \boxed{3} \end{bmatrix} = \begin{bmatrix} \quad & \quad \\ \quad & \quad \end{bmatrix}$$

$$\begin{bmatrix} 9 & 18 \\ 4 & 8 \end{bmatrix} + \begin{bmatrix} 9 & -15 \\ 3 & -5 \end{bmatrix} + \begin{bmatrix} 10 & 15 \\ 4 & 6 \end{bmatrix}$$

# Matrix Multiplication via Outer Products

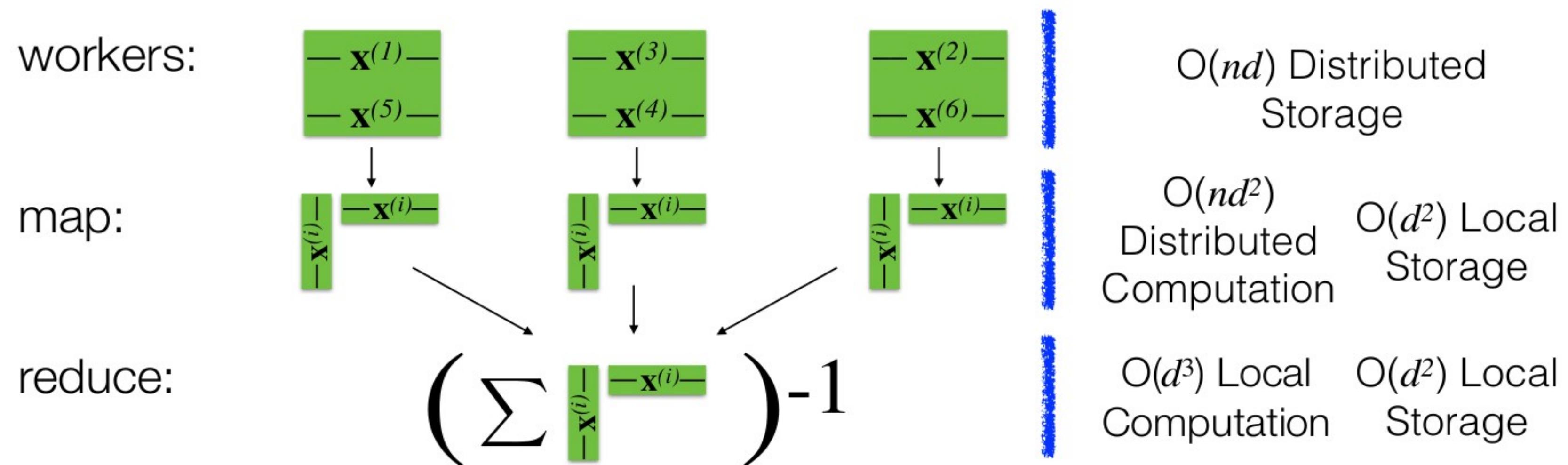
Output matrix is **sum of outer products** between corresponding rows and columns of input matrices

$$\begin{bmatrix} 9 & 3 & \boxed{5} \\ 4 & 1 & \boxed{2} \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & -5 \\ \boxed{2} & \boxed{3} \end{bmatrix} = \begin{bmatrix} 28 & 18 \\ 11 & 9 \end{bmatrix}$$

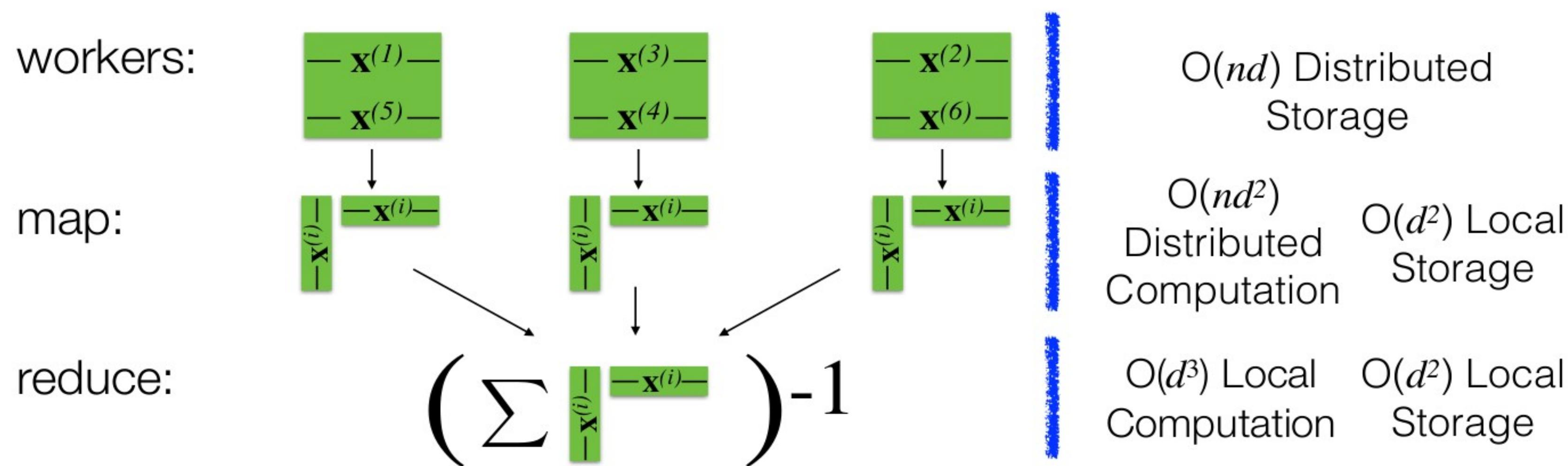
$$\begin{bmatrix} 9 & 18 \\ 4 & 8 \end{bmatrix} + \begin{bmatrix} 9 & -15 \\ 3 & -5 \end{bmatrix} + \begin{bmatrix} 10 & 15 \\ 4 & 6 \end{bmatrix}$$

$$\mathbf{X}^\top \mathbf{X} = \begin{matrix} & & n & & d \\ & \begin{matrix} \mathbf{x}^{(1)} \\ \vdots \\ \mathbf{x}^{(n)} \end{matrix} & \cdots & \begin{matrix} \mathbf{x}^{(1)} \\ \vdots \\ \mathbf{x}^{(n)} \end{matrix} & \end{matrix} = \sum_{i=1}^n \begin{matrix} & \mathbf{x}^{(i)} \\ \mathbf{x}^{(i)} \end{matrix}$$

Example:  $n = 6$ ; 3 workers



> `trainData.map(computeOuterProduct)  
 .reduce(sumAndInvert)`



# **Distributed ML: Computation and Storage, Part II**

# Big $n$ and Small $d$

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

**Computation:**  $O(\cancel{nd^2} + d^3)$  operations

**Storage:**  $O(\cancel{nd} + d^2)$  floats

Assume  $O(d^3)$  computation and  $O(d^2)$  storage feasible on single machine

Can distribute storage and computation!

- Store data points (rows of  $\mathbf{X}$ ) across machines
- Compute  $\mathbf{X}^\top \mathbf{X}$  as a sum of outer products

## Big $n$ and Small $d$

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

**Computation:**  $O(\cancel{nd^2} + d^3)$  operations

**Storage:**  $O(\cancel{nd} + d^2)$  floats

```
> trainData.map(computeOuterProduct)  
    .reduce(sumAndInvert)
```

# Big $n$ and Big $d$

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

**Computation:**  $O(\underline{nd^2} + \underline{d^3})$  operations

**Storage:**  $O(\underline{nd} + \underline{d^2})$  floats

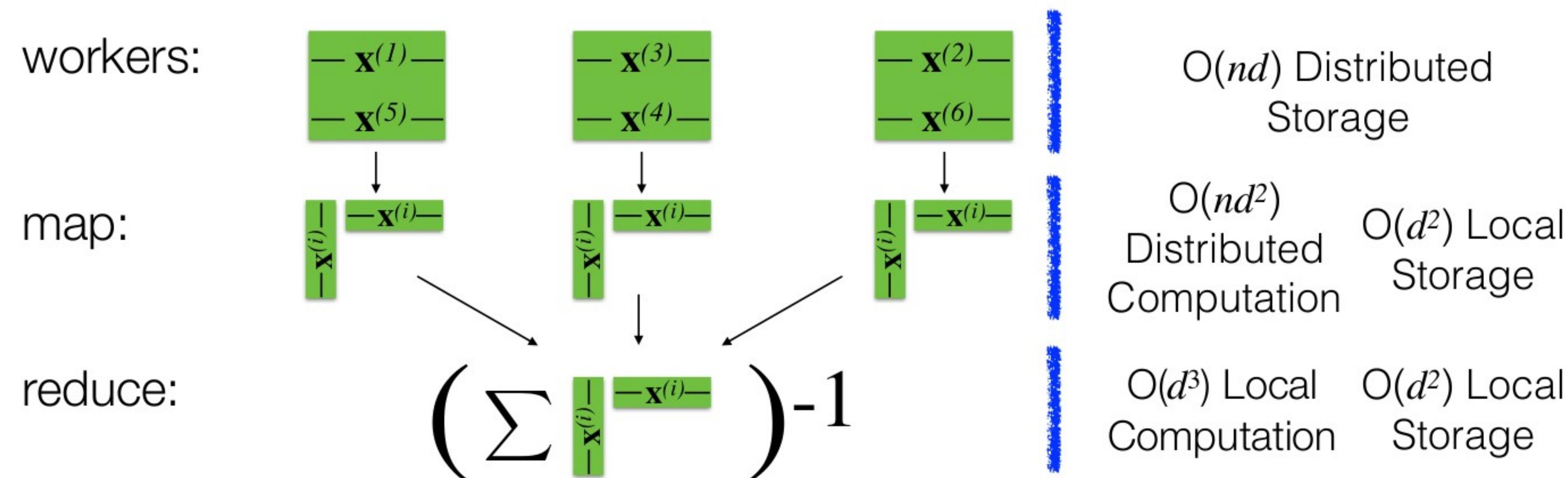
As before, storing  $\mathbf{X}$  and computing  $\mathbf{X}^\top \mathbf{X}$  are bottlenecks

Now, storing and operating on  $\mathbf{X}^\top \mathbf{X}$  is also a bottleneck

- Can't easily distribute!

$$\mathbf{X}^\top \mathbf{X} = \sum_{i=1}^n \begin{matrix} d \\ -\mathbf{x}^{(1)}- \\ -\mathbf{x}^{(2)}- \\ \cdots \\ -\mathbf{x}^{(n)}- \end{matrix} = \sum_{i=1}^n \begin{matrix} d \\ -\mathbf{x}^{(1)}- \\ -\mathbf{x}^{(2)}- \\ \vdots \\ -\mathbf{x}^{(n)}- \end{matrix}$$

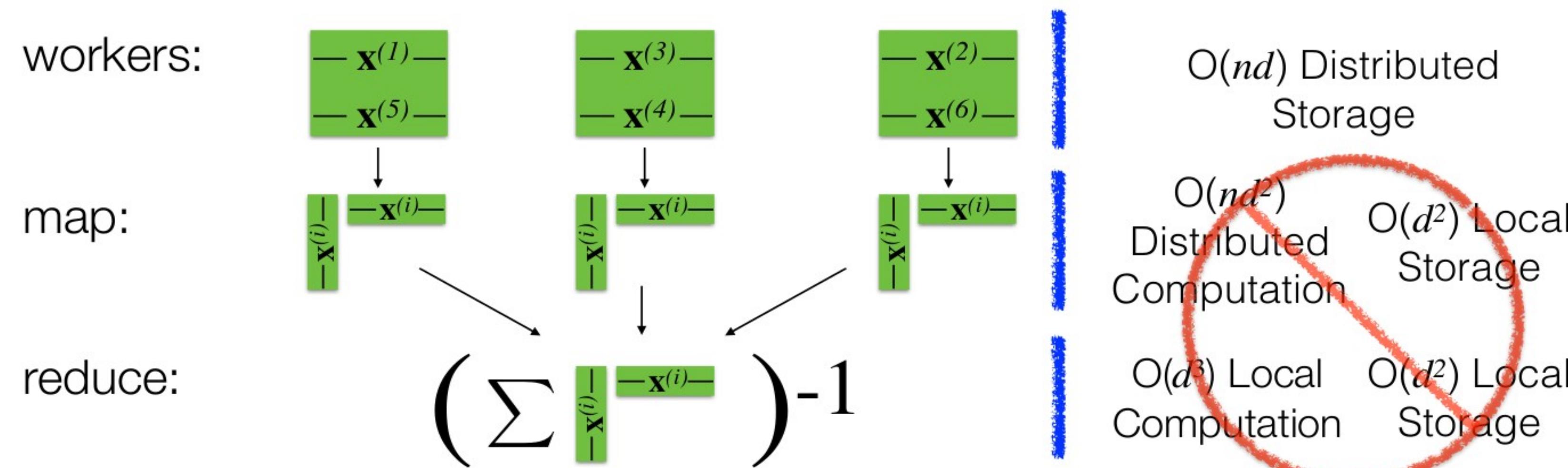
Example:  $n = 6$ ; 3 workers



$$\mathbf{X}^\top \mathbf{X} = \sum_{i=1}^n \begin{pmatrix} -\mathbf{x}^{(i)}- \\ -\mathbf{x}^{(i)}- \end{pmatrix}$$

$d$        $n$        $d$   
 $\mathbf{x}^{(1)}$      $\mathbf{x}^{(2)}$      $\cdots$      $\mathbf{x}^{(n)}$   
 $\vdots$   
 $\mathbf{x}^{(1)}$      $\mathbf{x}^{(2)}$   
 $\vdots$   
 $\mathbf{x}^{(n)}$

Example:  $n = 6$ ; 3 workers



# Big $n$ and Big $d$

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

**Computation:**  $O(\underline{nd^2} + \underline{d^3})$  operations

**Storage:**  $O(\underline{nd} + \underline{d^2})$  floats

As before, storing  $\mathbf{X}$  and computing  $\mathbf{X}^\top \mathbf{X}$  are bottlenecks

Now, storing and operating on  $\mathbf{X}^\top \mathbf{X}$  is also a bottleneck

- Can't easily distribute!

## 1st Rule of thumb

Computation and storage should be linear (in  $n, d$ )

# Big $n$ and Big $d$

We need methods that are linear in time and space

One idea: **Exploit sparsity**

- Explicit sparsity can provide orders of magnitude storage and computational gains

Sparse data is prevalent

- Text processing: bag-of-words, n-grams
- Collaborative filtering: ratings matrix
- Graphs: adjacency matrix
- Categorical features: one-hot-encoding
- Genomics: SNPs, variant calling

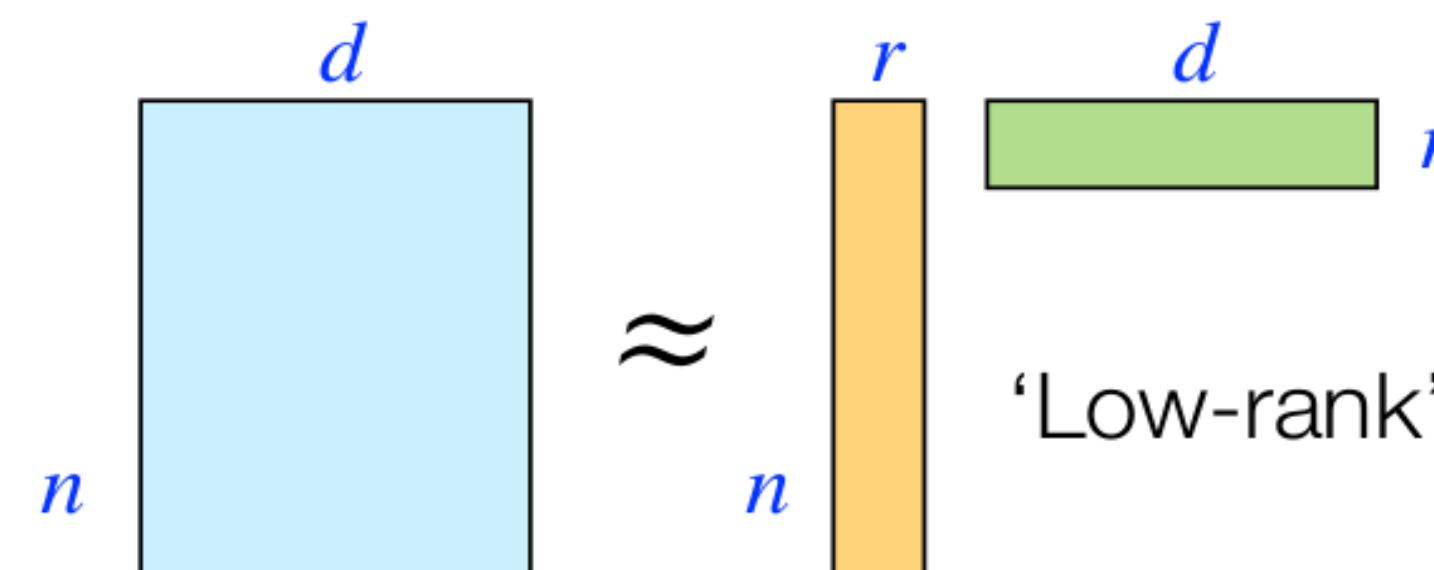
dense : 1. 0. 0. 0. 0. 0. 3.  
sparse :  $\begin{cases} \text{size} : 7 \\ \text{indices} : \underline{0} \underline{6} \\ \text{values} : \underline{1.} \underline{3.} \end{cases}$

# Big $n$ and Big $d$

We need methods that are linear in time and space

One idea: **Exploit sparsity**

- Explicit sparsity can provide orders of magnitude storage and computational gains
- Latent sparsity assumption can be used to reduce dimension, e.g., PCA, low-rank approximation (unsupervised learning)



# Big $n$ and Big $d$

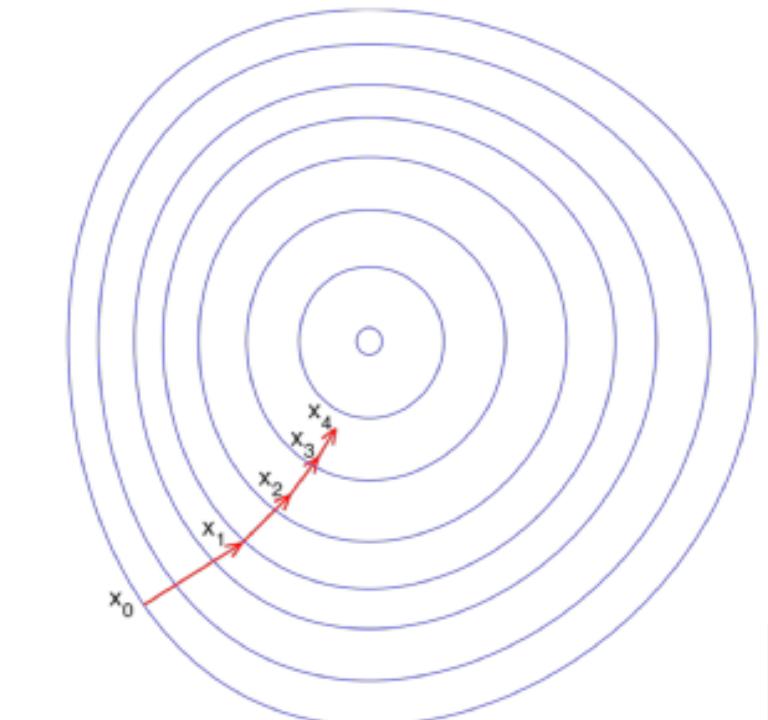
We need methods that are linear in time and space

One idea: **Exploit sparsity**

- Explicit sparsity can provide orders of magnitude storage and computational gains
- Latent sparsity assumption can be used to reduce dimension, e.g., PCA, low-rank approximation (unsupervised learning)

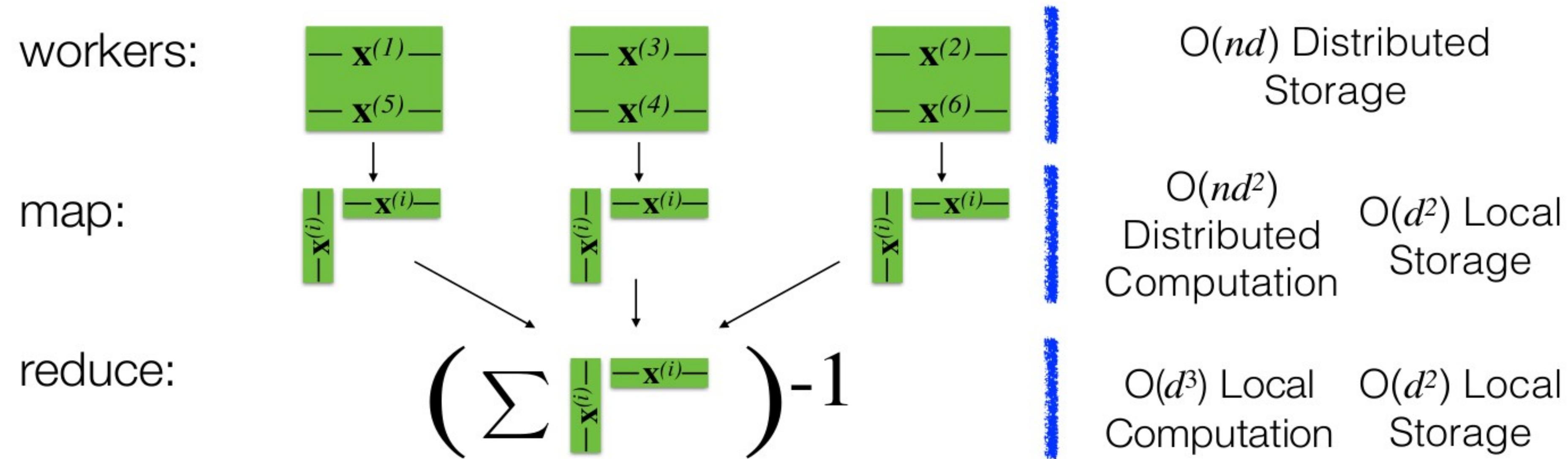
Another idea: **Use different algorithms**

- Gradient descent is an iterative algorithm that requires  $O(nd)$  computation and  $O(d)$  local storage per iteration

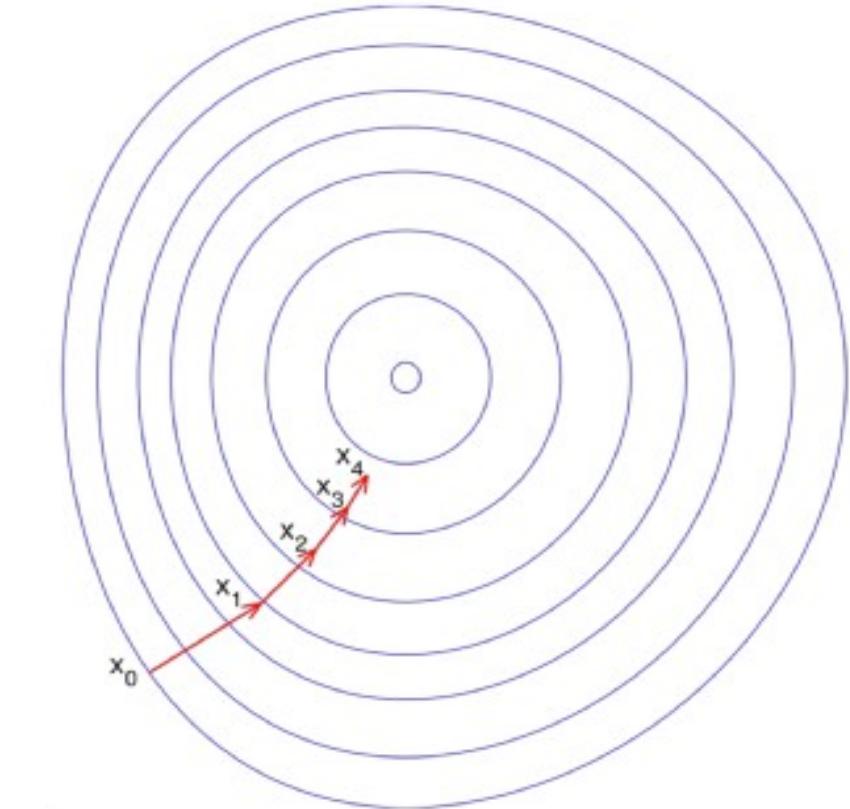


# Closed Form Solution for Big $n$ and Big $d$

Example:  $n = 6$ ; 3 workers



# Gradient Descent for Big $n$ and Big $d$



Example:  $n = 6$ ; 3 workers

**workers:**

—  $\mathbf{x}^{(1)}$  —  
—  $\mathbf{x}^{(5)}$  —

—  $\mathbf{x}^{(3)}$  —  
—  $\mathbf{x}^{(4)}$  —

—  $\mathbf{x}^{(2)}$  —  
—  $\mathbf{x}^{(6)}$  —

map:

—  $\mathbf{x}^{(i)}$  —  
—  $\mathbf{x}^{(i)}$  —

—  $\mathbf{x}^{(i)}$  —  
—  $\mathbf{x}^{(i)}$  —

—  $\mathbf{x}^{(i)}$  —  
—  $\mathbf{x}^{(i)}$  —

reduce:

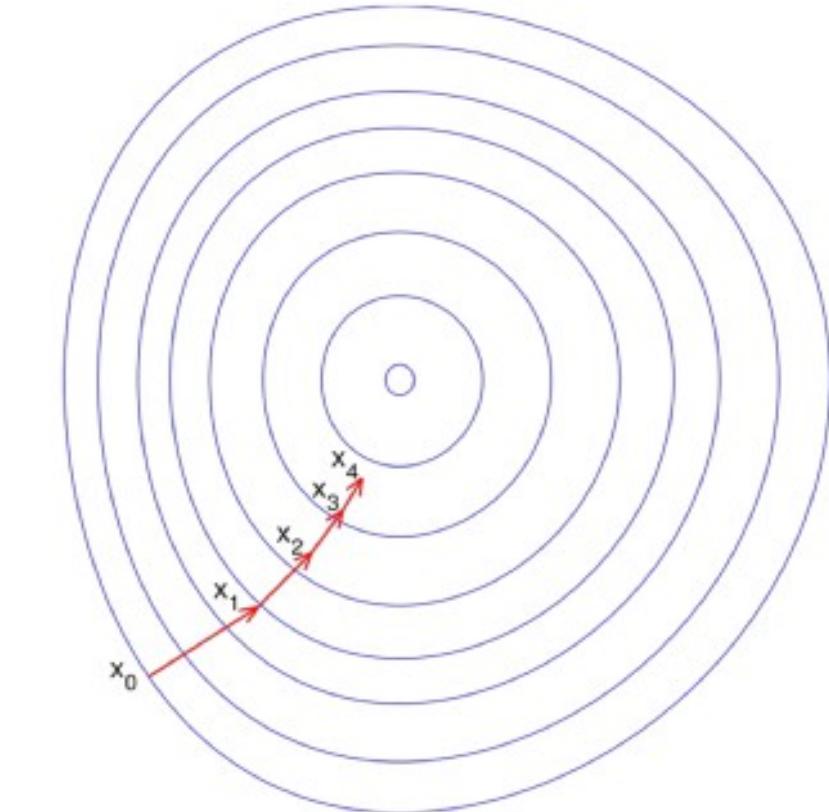
$$\left( \sum \begin{array}{|c|} \hline \mathbf{x}^{(i)} \\ \hline \end{array} \right) - 1$$

$O(nd)$  Distributed Storage

$O(nd^2)$  Distributed Computation       $O(d^2)$  Local Storage

$O(d^3)$  Local Computation       $O(d^2)$  Local Storage

# Gradient Descent for Big $n$ and Big $d$



Example:  $n = 6$ ; 3 workers

workers:

$$\begin{array}{|c|} \hline - \mathbf{x}^{(1)} - \\ \hline - \mathbf{x}^{(5)} - \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline - \mathbf{x}^{(3)} - \\ \hline - \mathbf{x}^{(4)} - \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline - \mathbf{x}^{(2)} - \\ \hline - \mathbf{x}^{(6)} - \\ \hline \end{array}$$

map:

?

?

?

reduce:

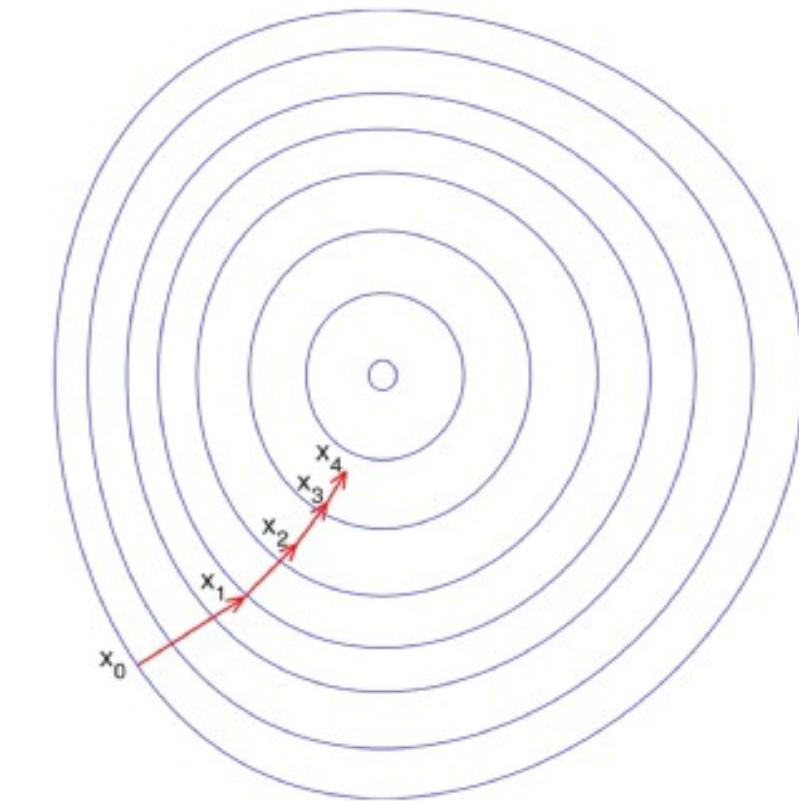
$$\left( \sum \begin{array}{|c|} \hline - \mathbf{x}^{(i)} - \\ \hline \end{array} \right) - 1$$

$O(nd)$  Distributed Storage

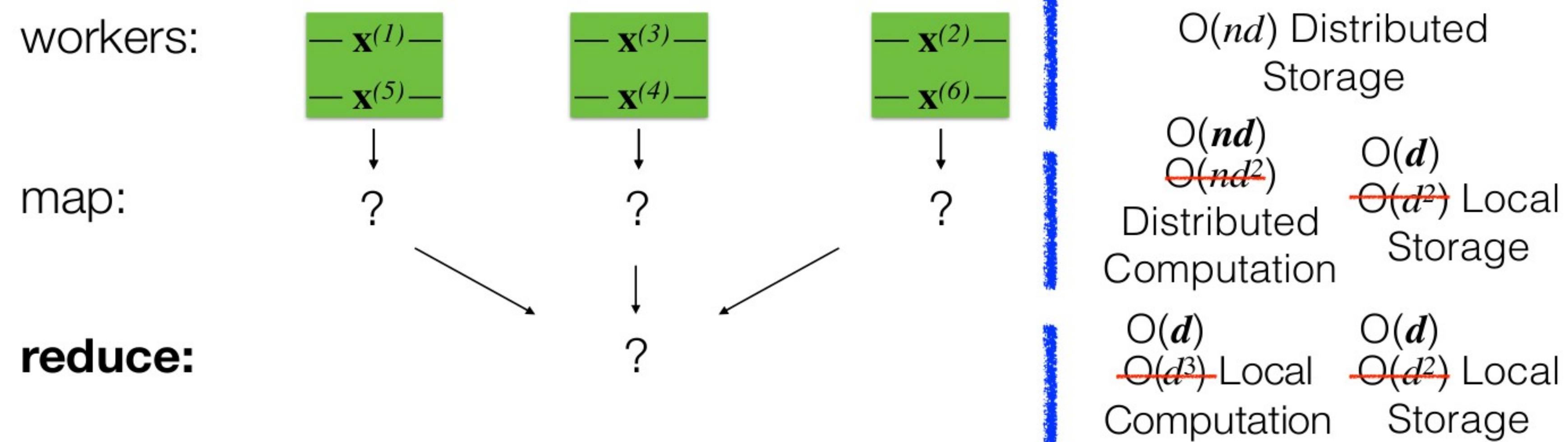
$O(nd)$   
 $O(nd^2)$  Distributed Computation       $O(d)$   
 $O(d^2)$  Local Storage

$O(d^3)$  Local Computation       $O(d^2)$  Local Storage

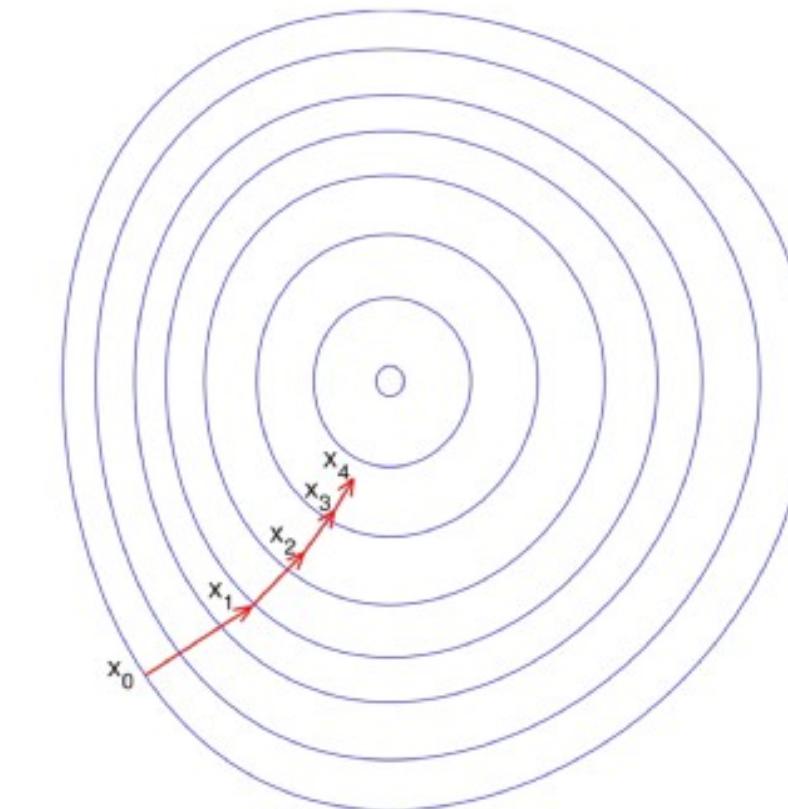
# Gradient Descent for Big $n$ and Big $d$



Example:  $n = 6$ ; 3 workers



# Gradient Descent for Big $n$ and Big $d$



Example:  $n = 6$ ; 3 workers

