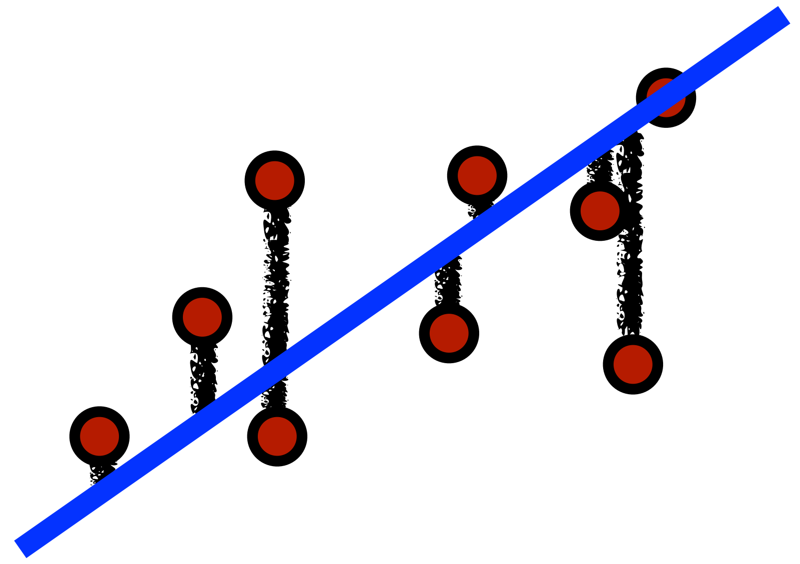
Linear Regression

Regression

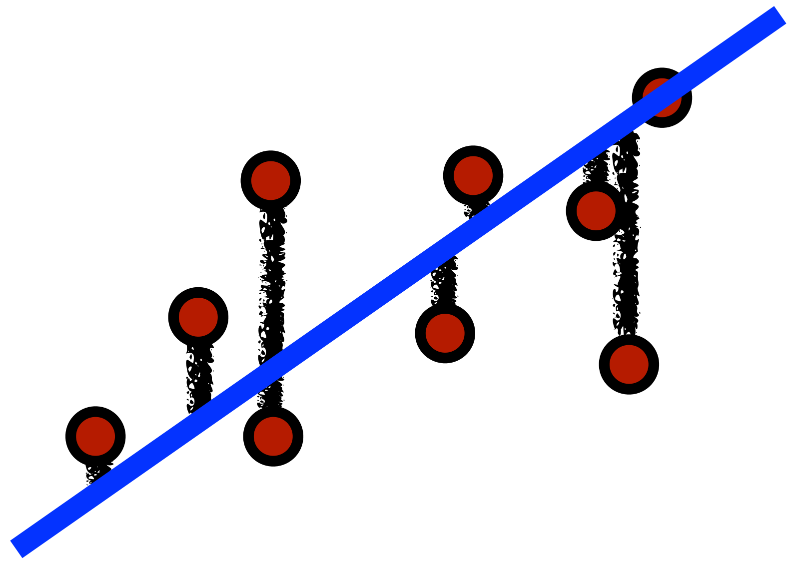


Goal: Learn a mapping from observations (features) to continuous labels given a training set (supervised learning)

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

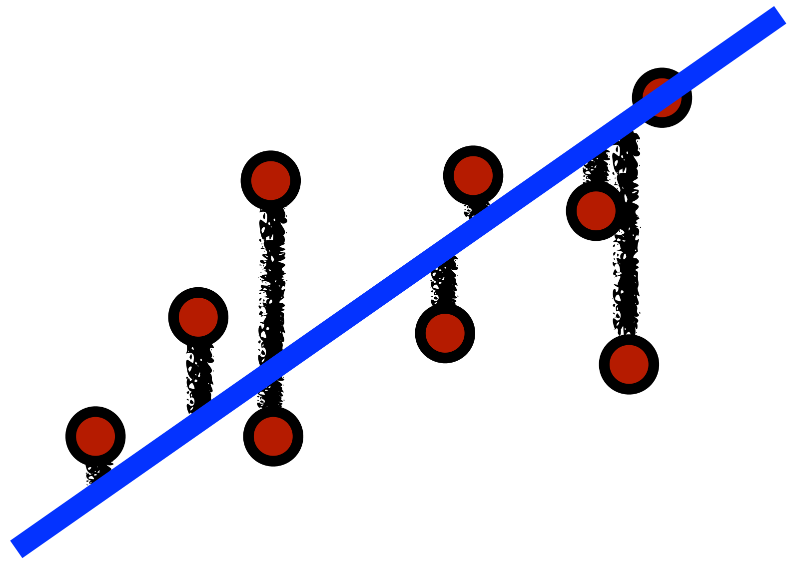
For each observation we have a feature vector, $\mathbf{x}$, and label, y

$$\mathbf{x}^\top = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}$$

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

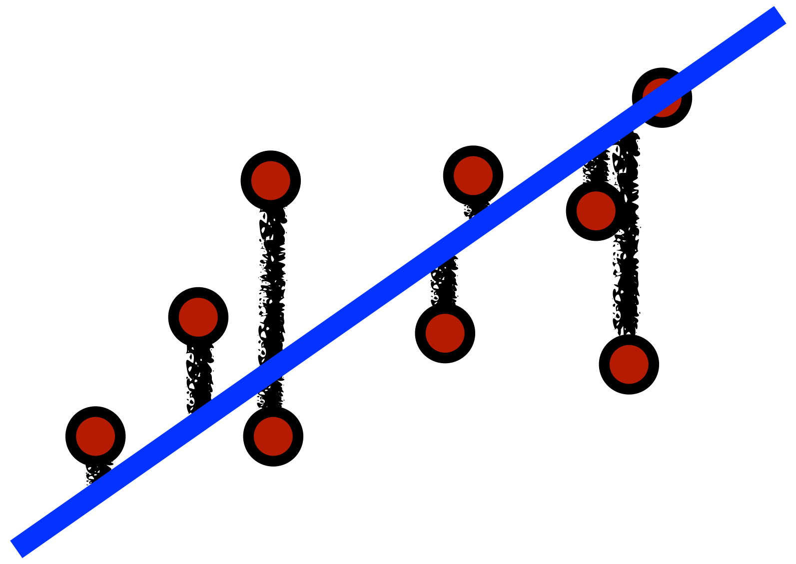
We can augment the feature vector to incorporate offset:

$$\mathbf{x}^\top = \begin{bmatrix} 1 & x_1 & x_2 & x_3 \end{bmatrix}$$

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

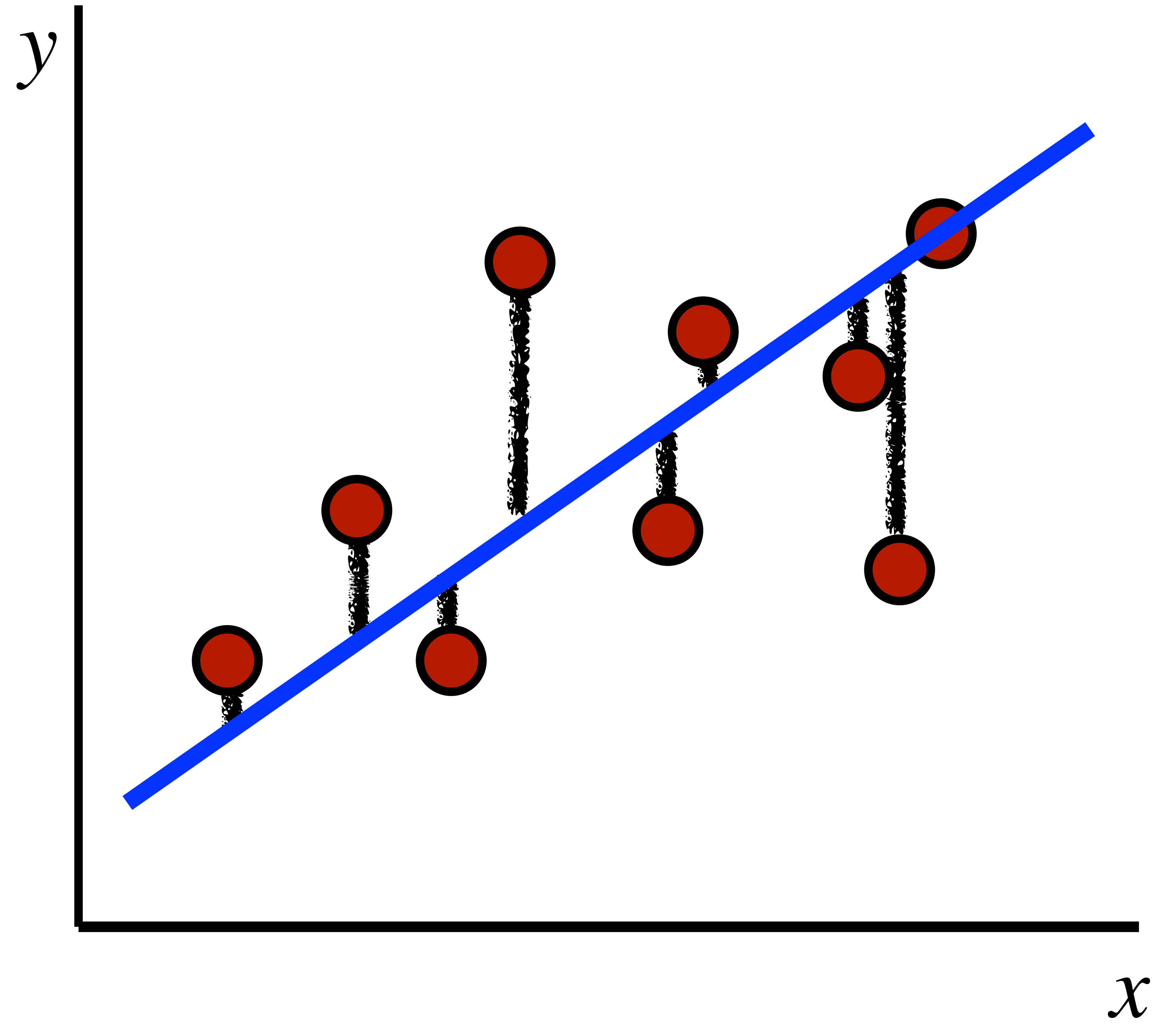
x coordinate: features

y coordinate: labels

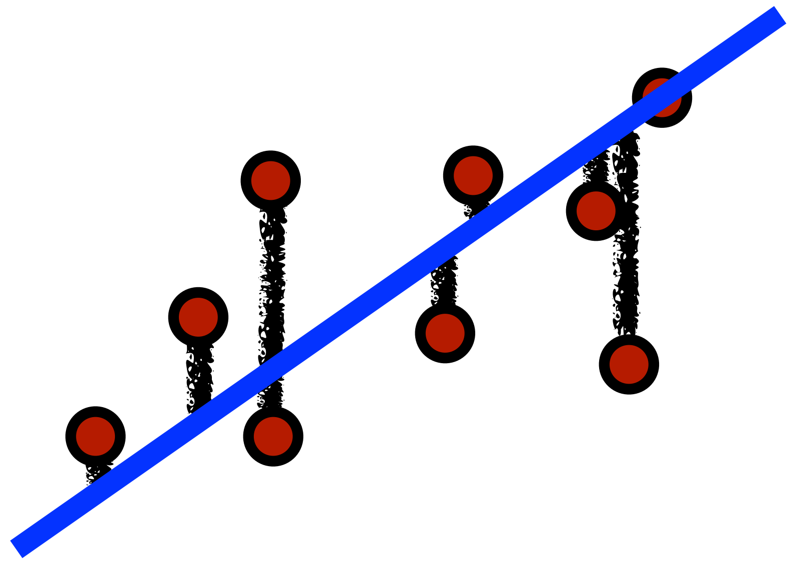
$$y \approx \hat{y} = w_0 + w_1 x$$

Intercept / Offset

Slope



Evaluating Predictions



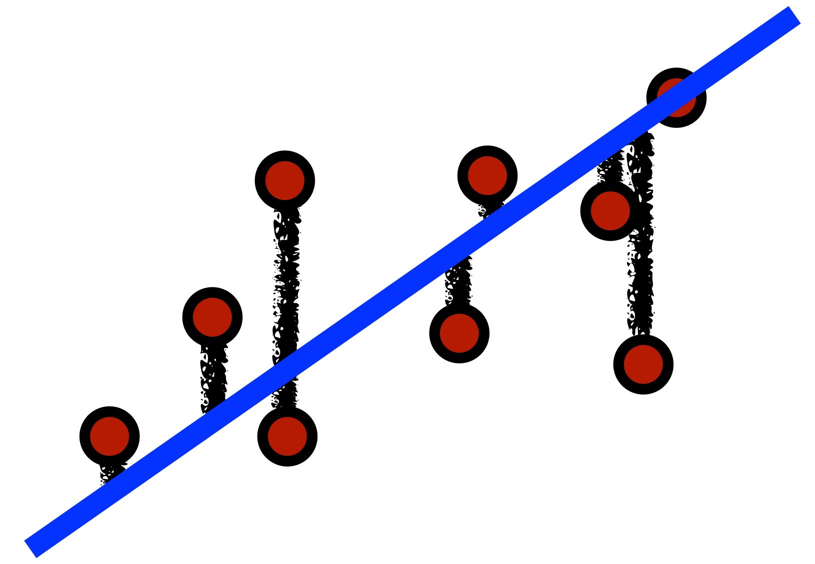
Can measure 'closeness' between label and prediction

- 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 ($\mathbf{w}$)?



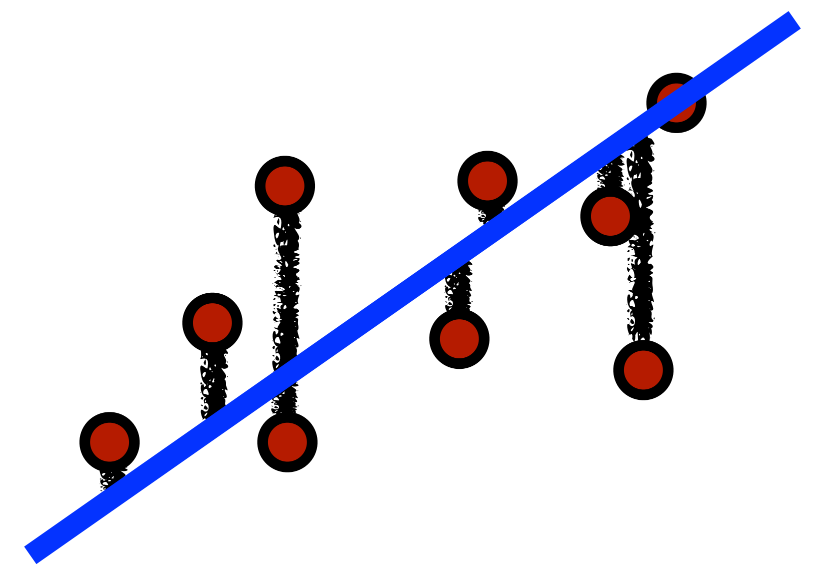
Assume we have n training points, where $\mathbf{x}^{(i)}$ denotes the ith point

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:

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



Given n training points with d features, we define:

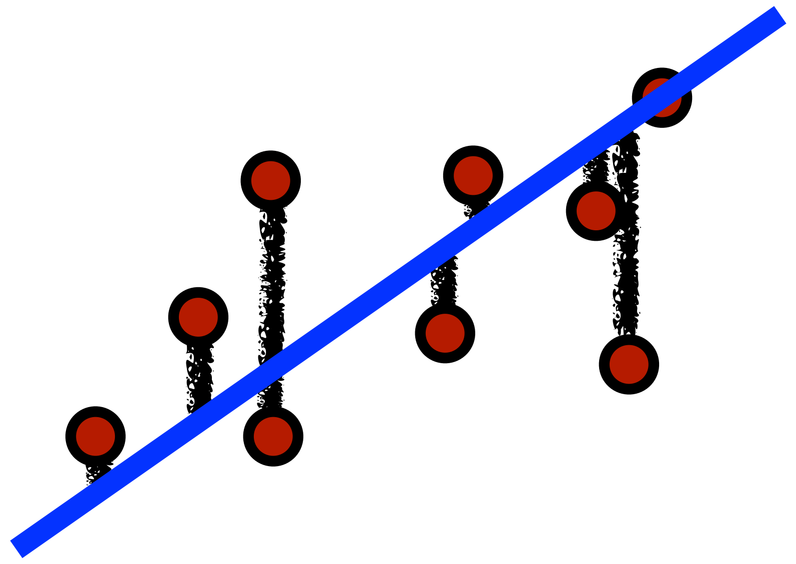
- $\mathbf{X} \in \mathbb{R}^{n \times d}$: matrix storing points
- $\mathbf{y} \in \mathbb{R}^{n}$: real-valued labels
- $\hat{\mathbf{y}} \in \mathbb{R}^{n}$: predicted labels, where $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$
- $\mathbf{w} \in \mathbb{R}^{d}$: regression parameters / model to learn

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

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

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

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

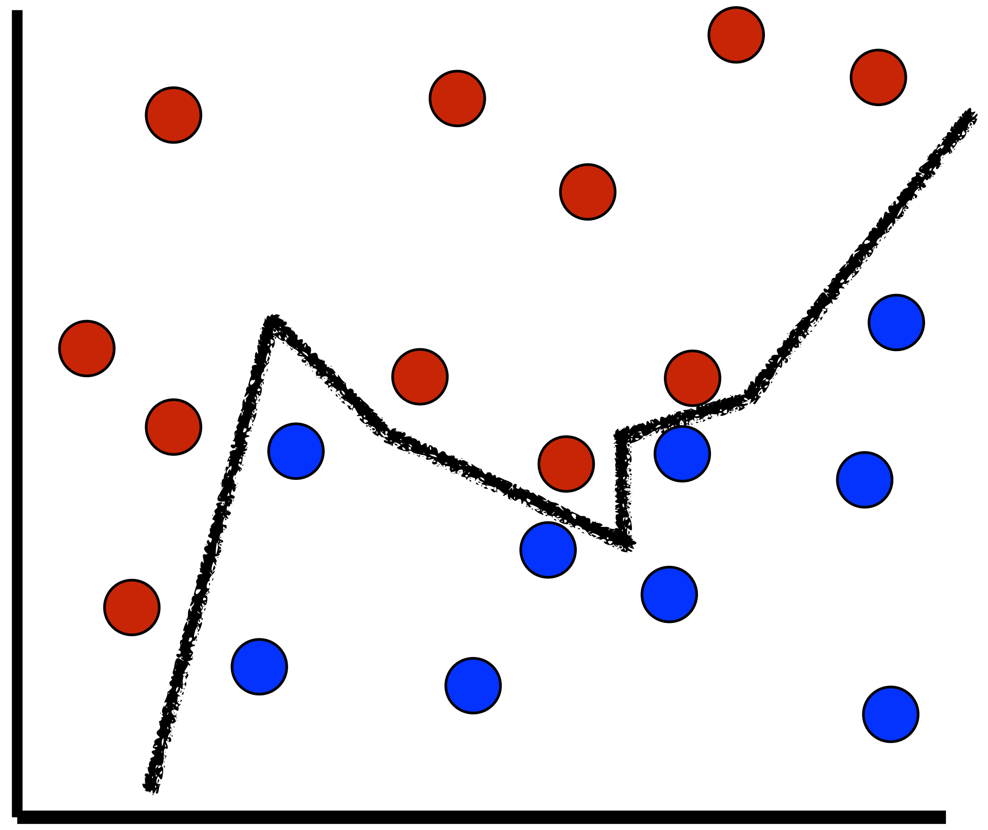
We want good predictions on new data, i.e., ’generalization’

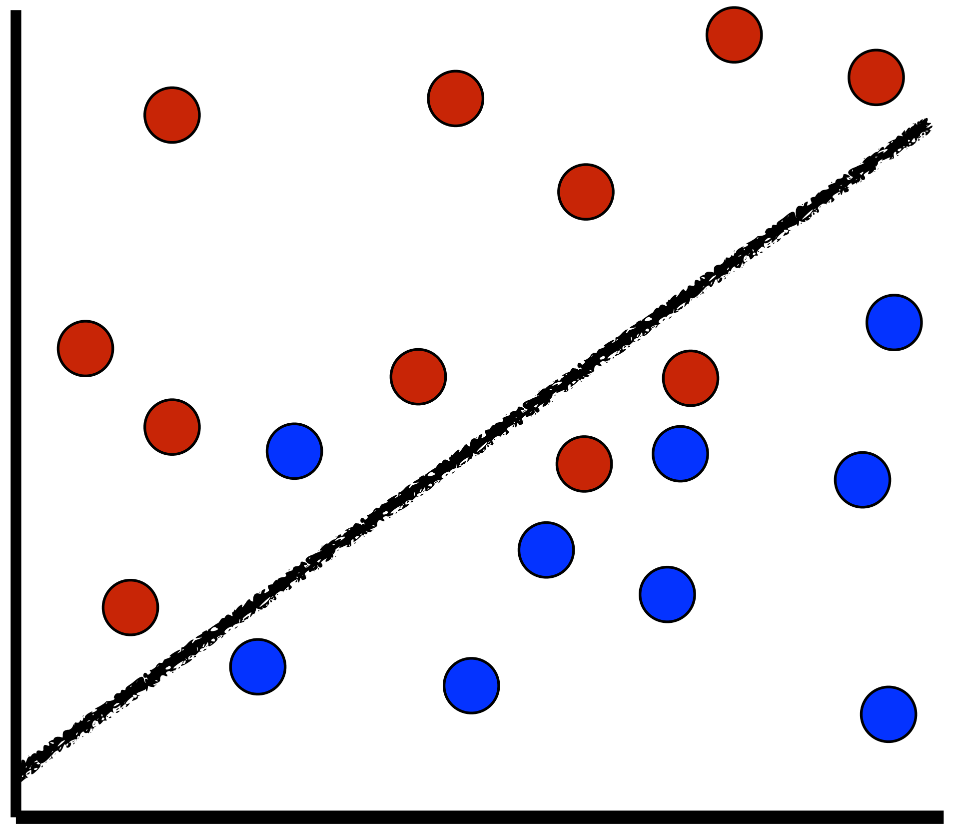
Least squares regression minimizes training error, and could overfit

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

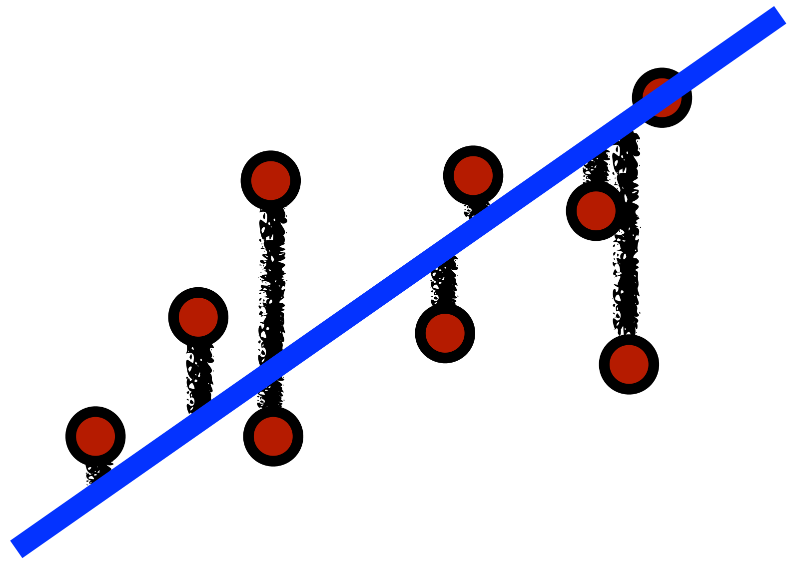
Can we change the problem to penalize for model complexity?

- 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

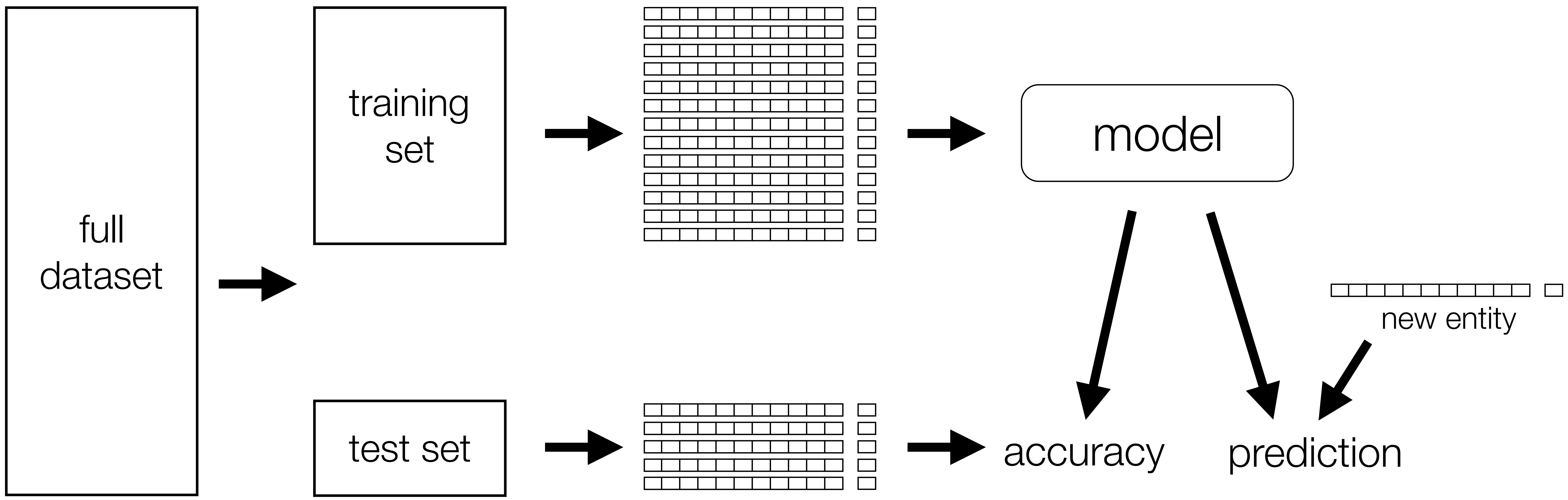
Ridge Regression: Learn mapping ($\mathbf{w}$) that minimizes residual sum of squares along with a regularization term:

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

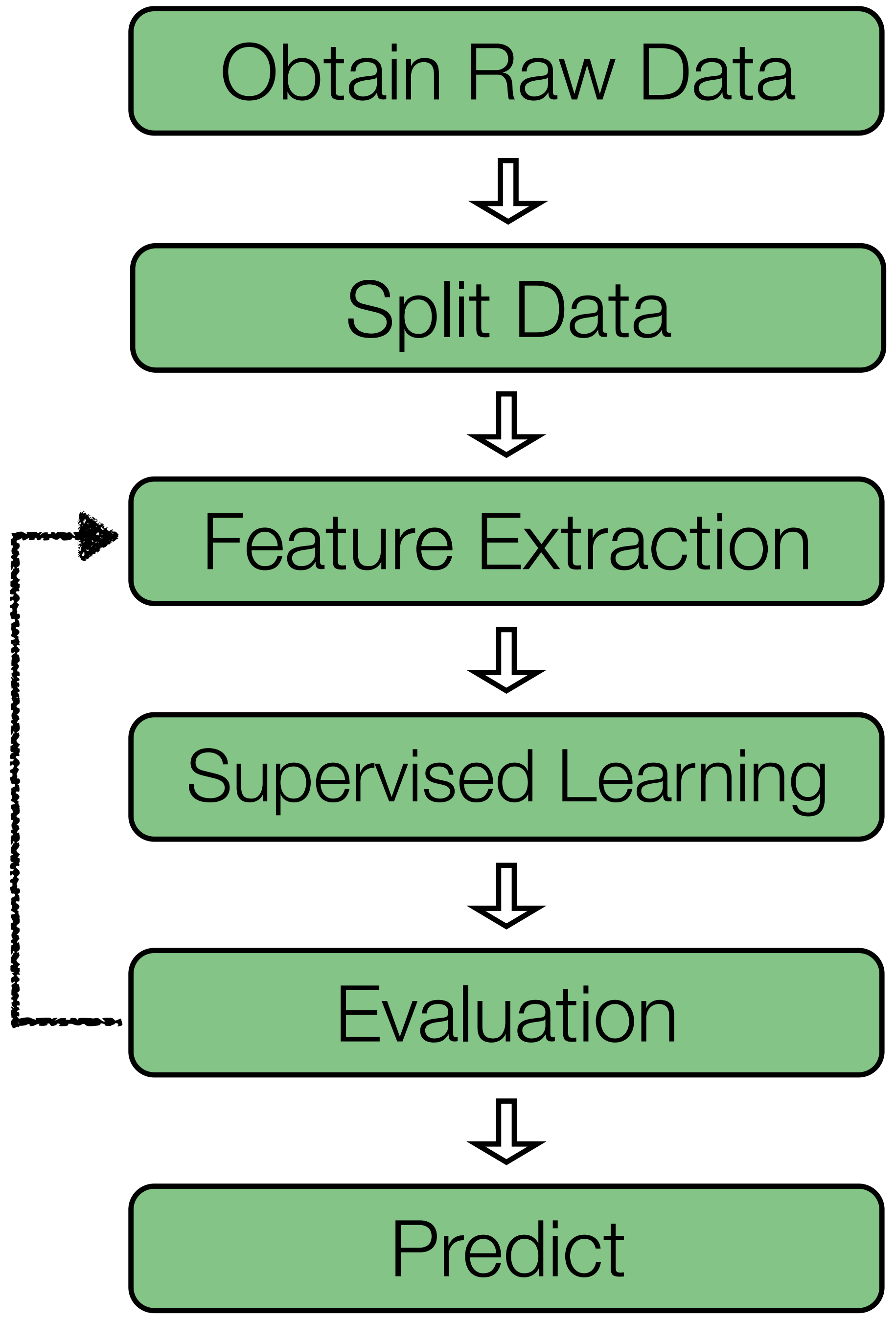
free parameter trades off between training error and model complexity

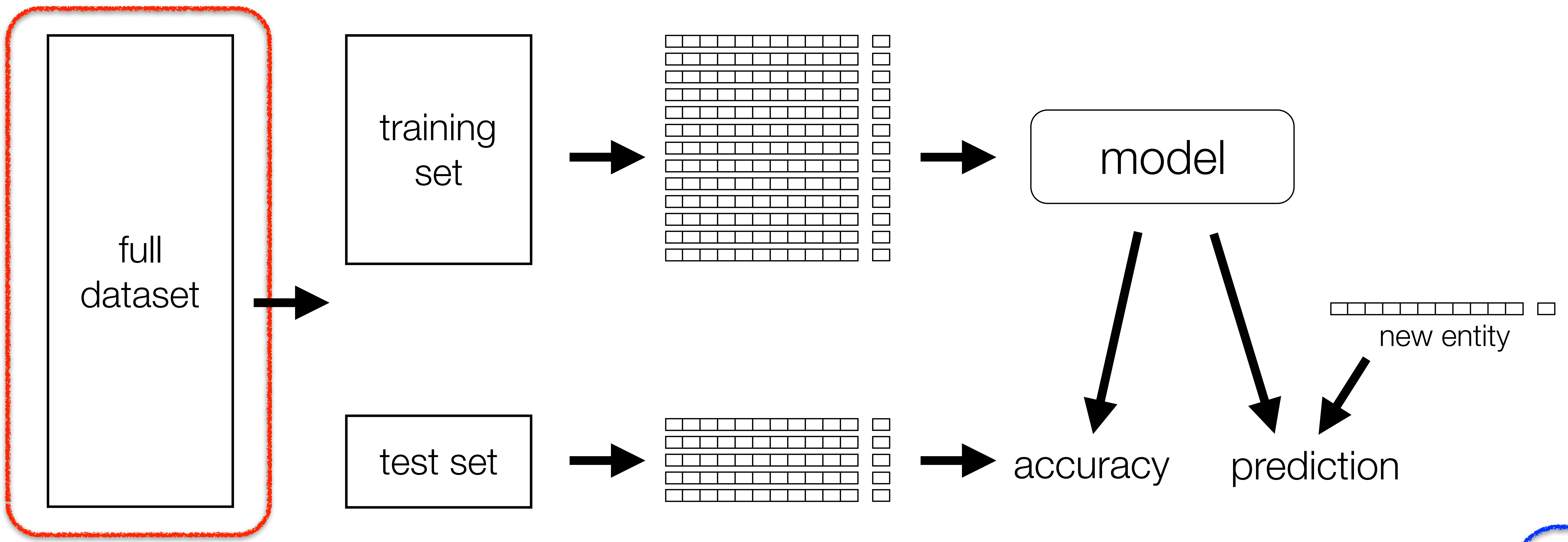
Closed-form solution: $\mathbf{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_d)^{-1} \mathbf{X}^\top \mathbf{y}$

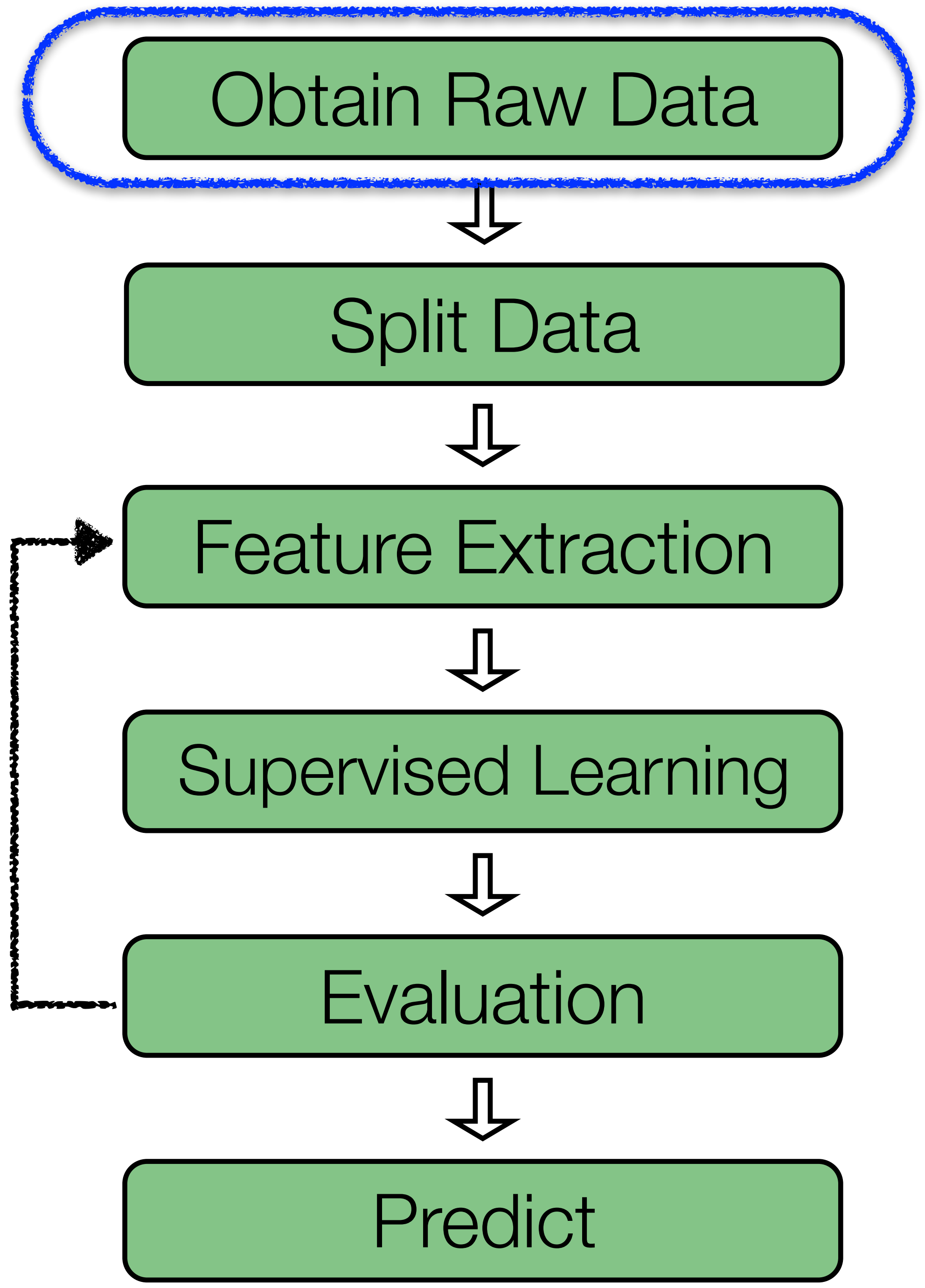
Millionsong
Regression Pipeline



Supervised Learning Pipeline



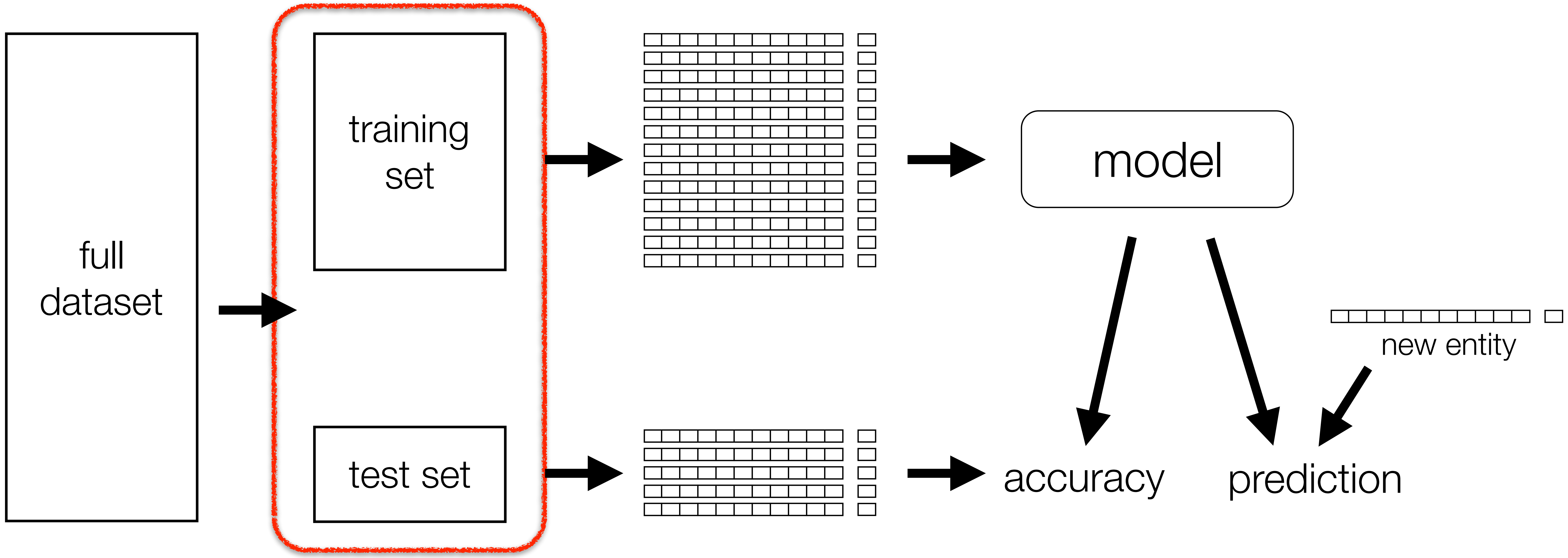




Goal: Predict song's release year from audio features

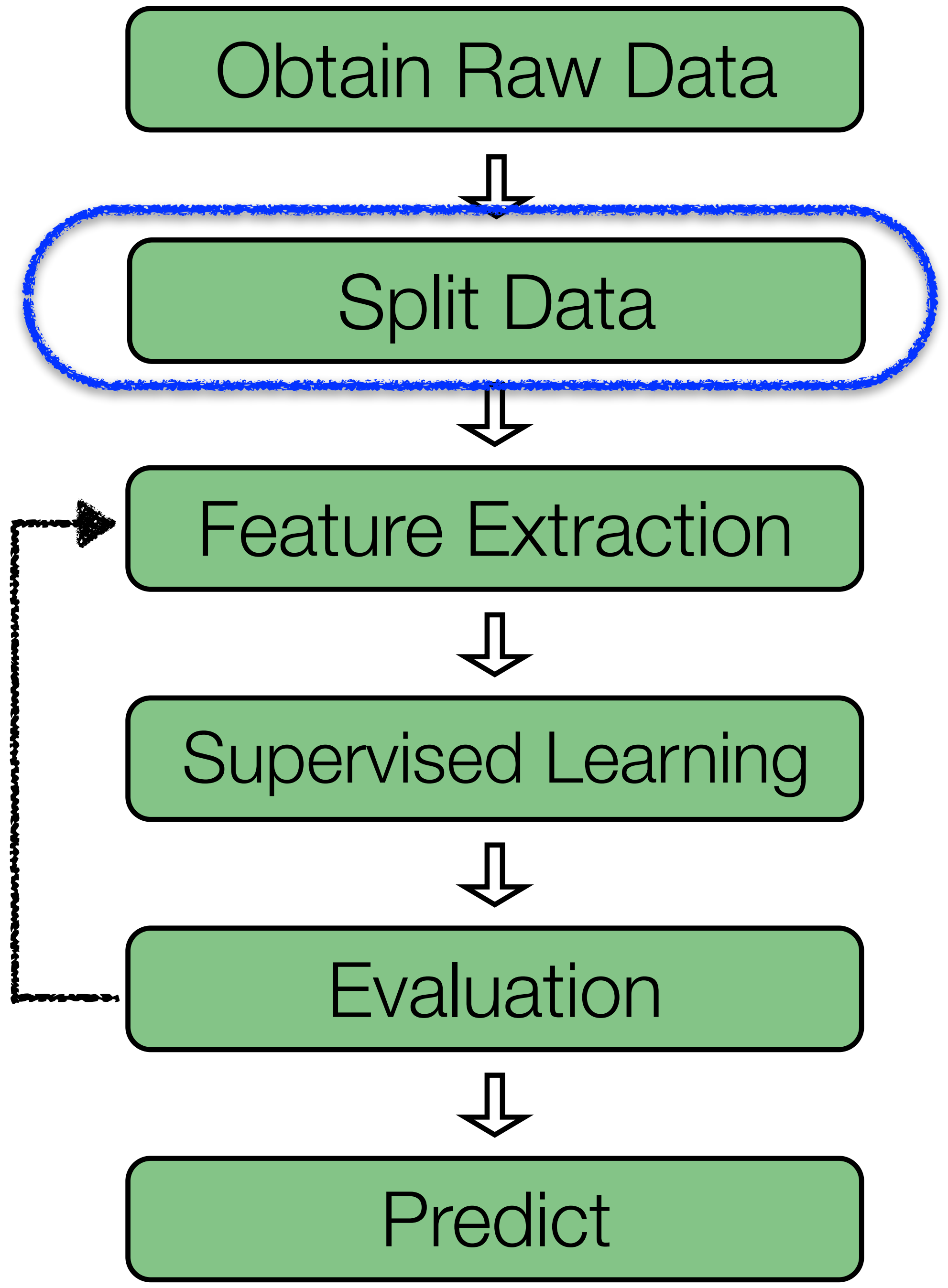
Raw Data: Millionsong Dataset from UCI ML Repository

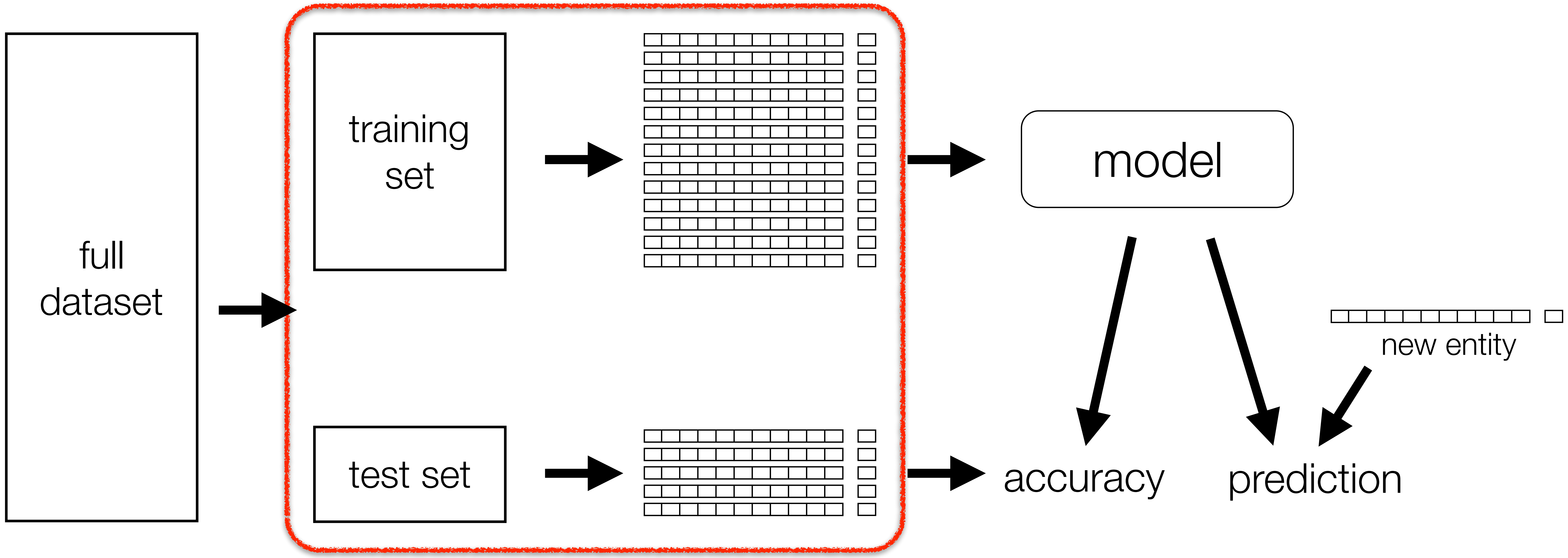
- Western, commercial tracks from 1980-2014
- 12 timbre averages (features) and release year (label)



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

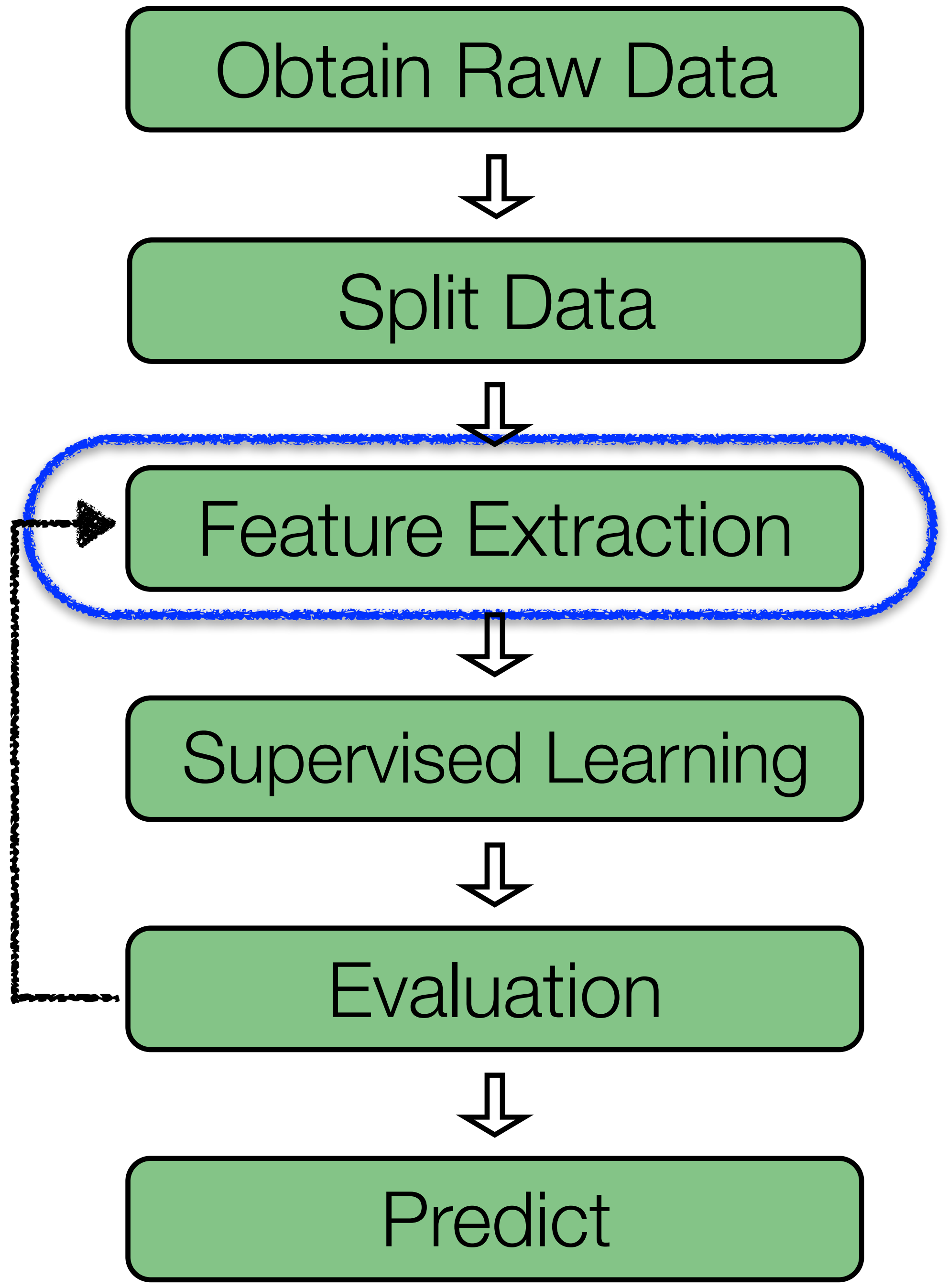
- Test set simulates unobserved data
- Test error tells us whether we've generalized well





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:

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^\top \implies \Phi(\mathbf{x}) = \begin{bmatrix} x_1^2 & x_1 x_2 & x_2 x_1 & x_2^2 \end{bmatrix}^\top$$

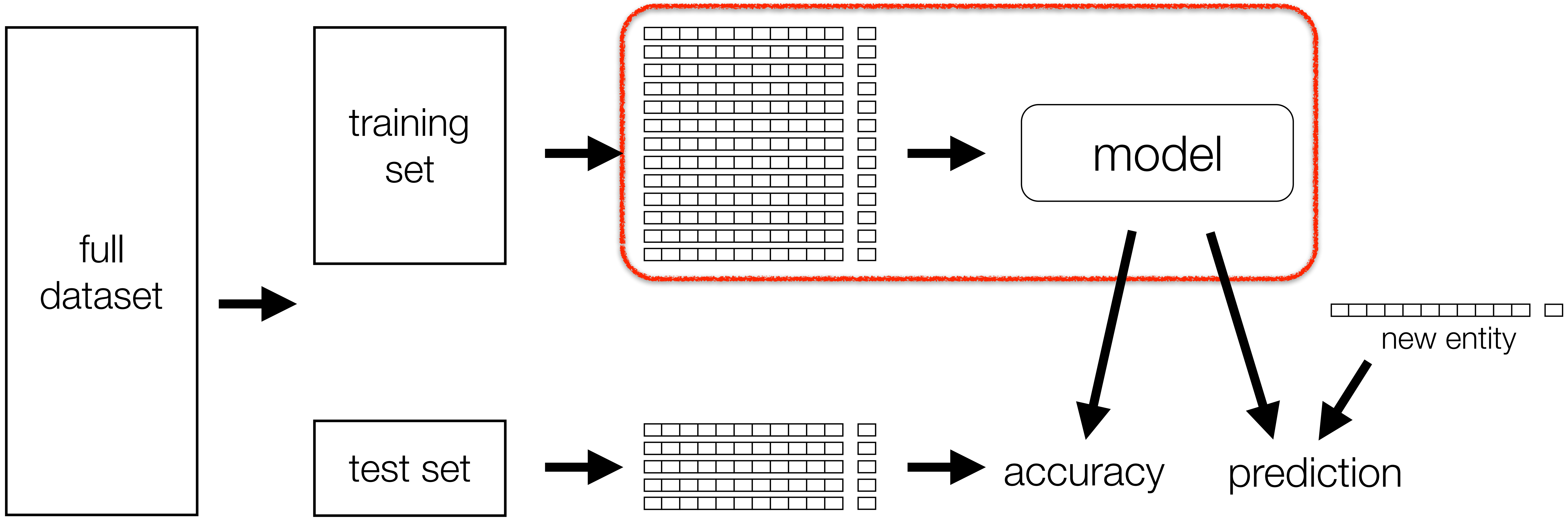
$$\mathbf{z} = \begin{bmatrix} z_1 & z_2 \end{bmatrix}^\top \implies \Phi(\mathbf{z}) = \begin{bmatrix} z_1^2 & z_1 z_2 & z_2 z_1 & z_2^2 \end{bmatrix}^\top$$

More succinctly:

$$\Phi'(\mathbf{x}) = \begin{bmatrix} x_1^2 & \sqrt{2} x_1 x_2 & x_2^2 \end{bmatrix}^\top \qquad \Phi'(\mathbf{z}) = \begin{bmatrix} z_1^2 & \sqrt{2} z_1 z_2 & z_2^2 \end{bmatrix}^\top$$

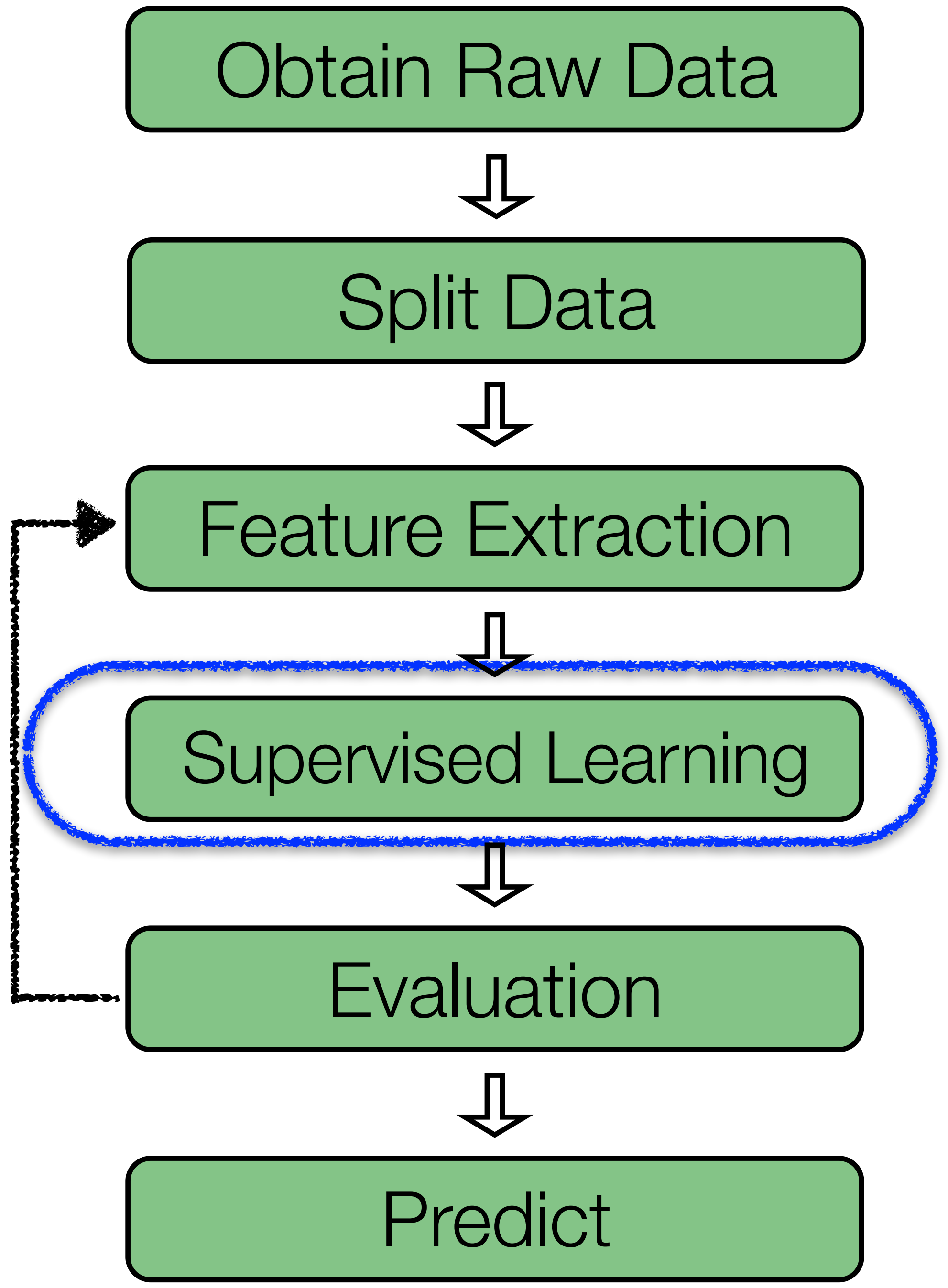
Equivalent inner products:

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

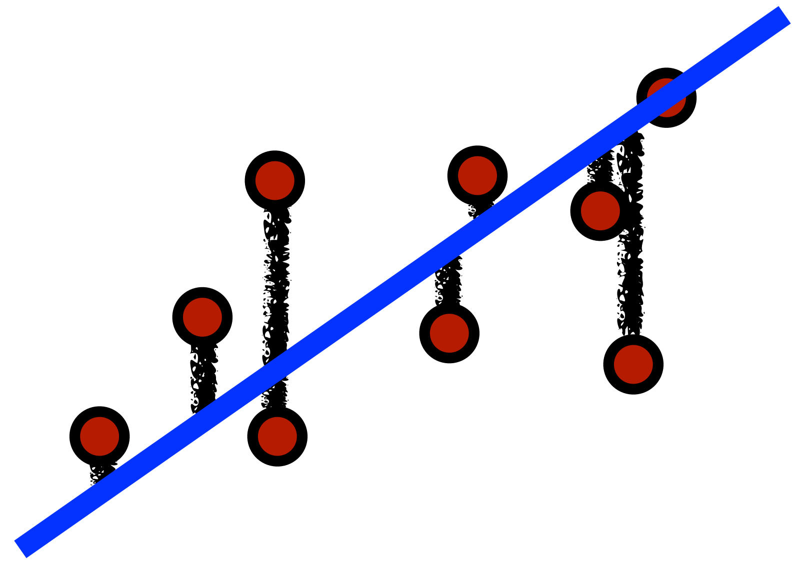


Supervised Learning: Least Squares Regression

- 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:

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

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:

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

free parameter trades off between training error and model complexity

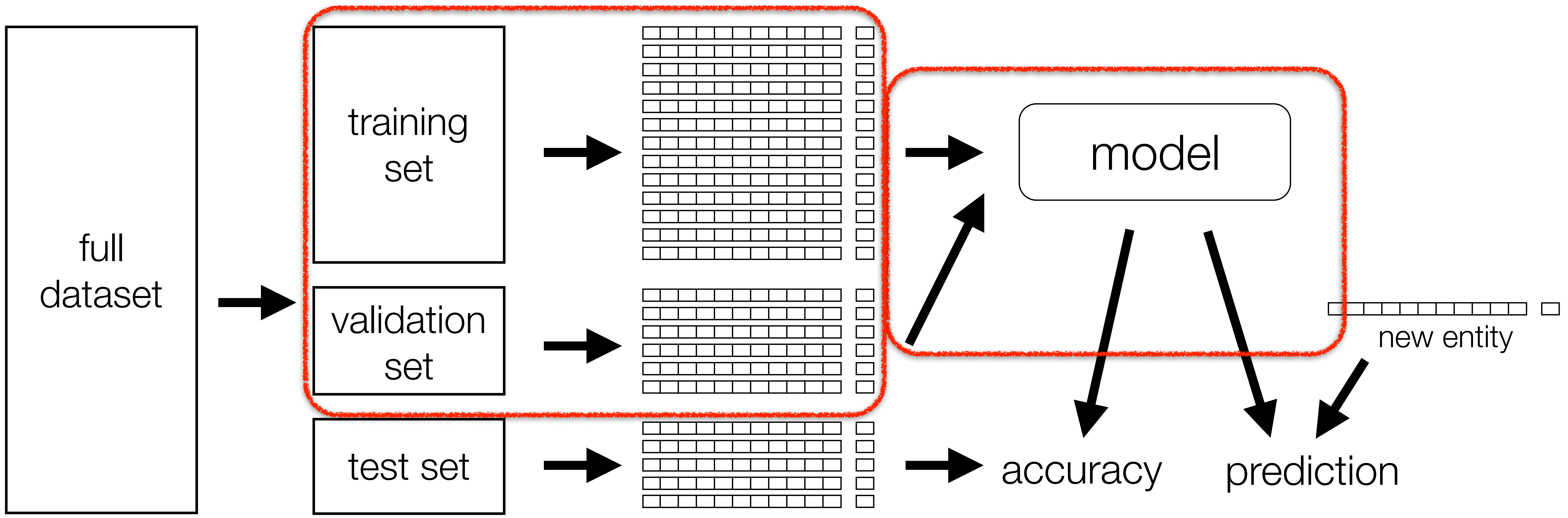
How do we choose a good value for this free parameter?

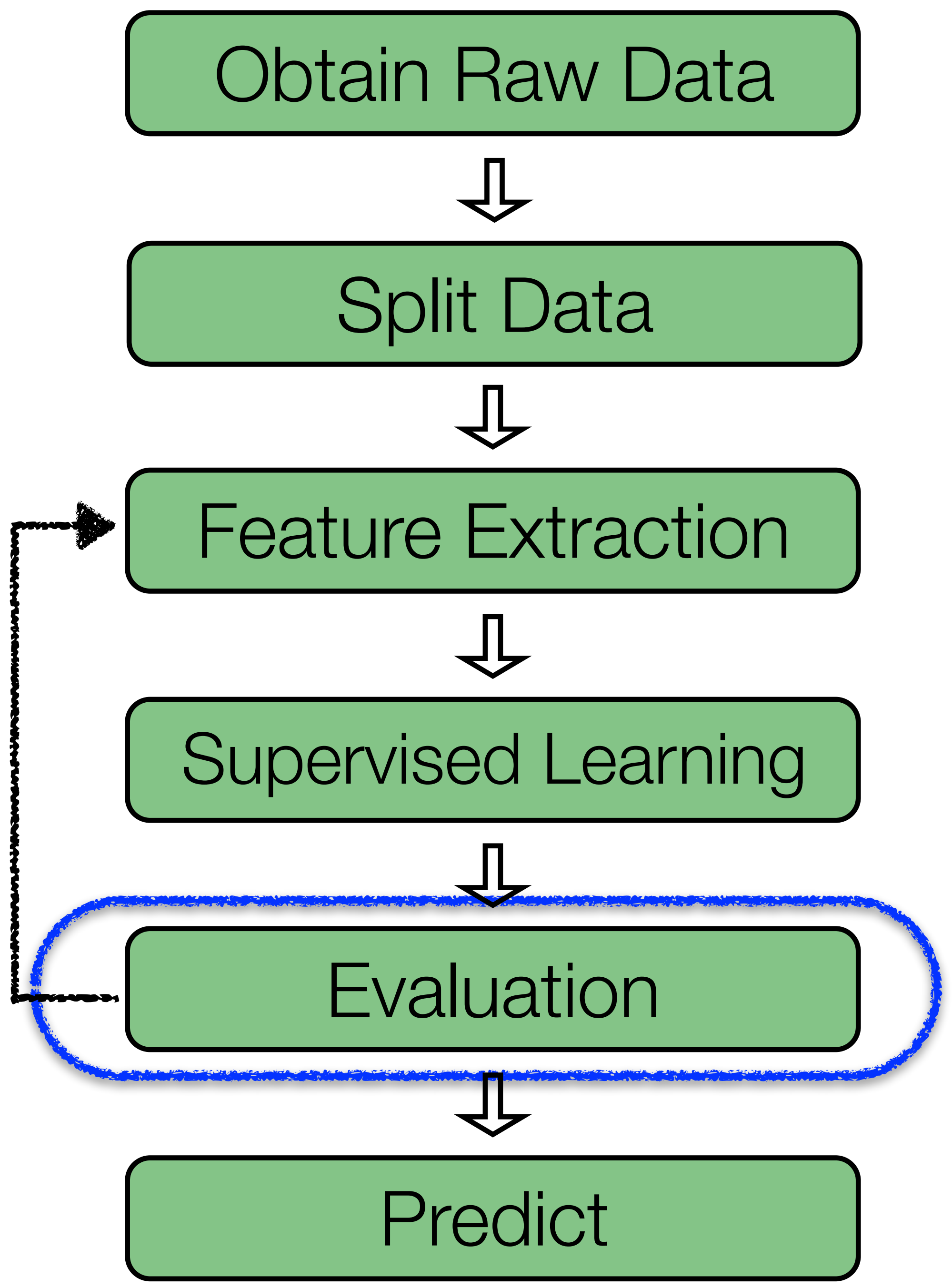
- Most methods have free parameters / ‘hyperparameters’ to tune

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

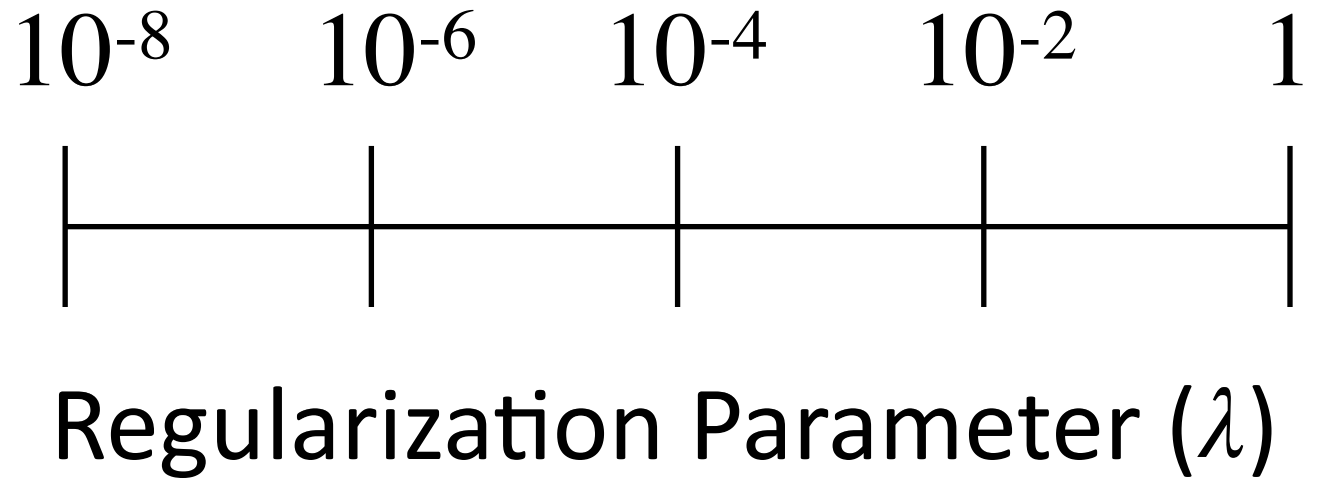
Second thought: **Create another hold out dataset for this search**





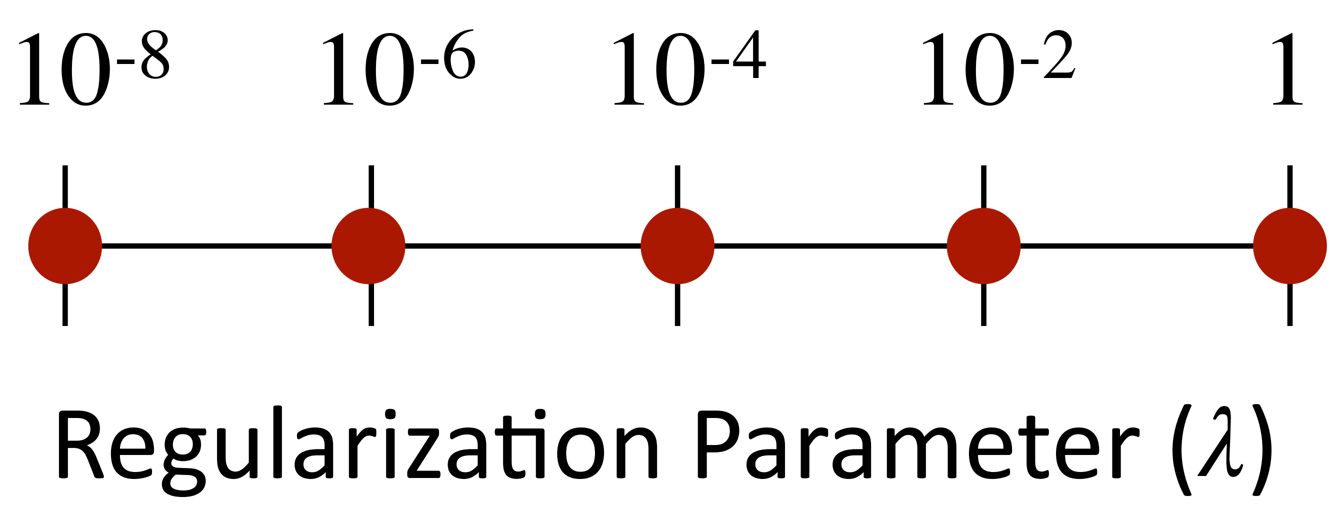
Evaluation (Part 1): Hyperparameter tuning

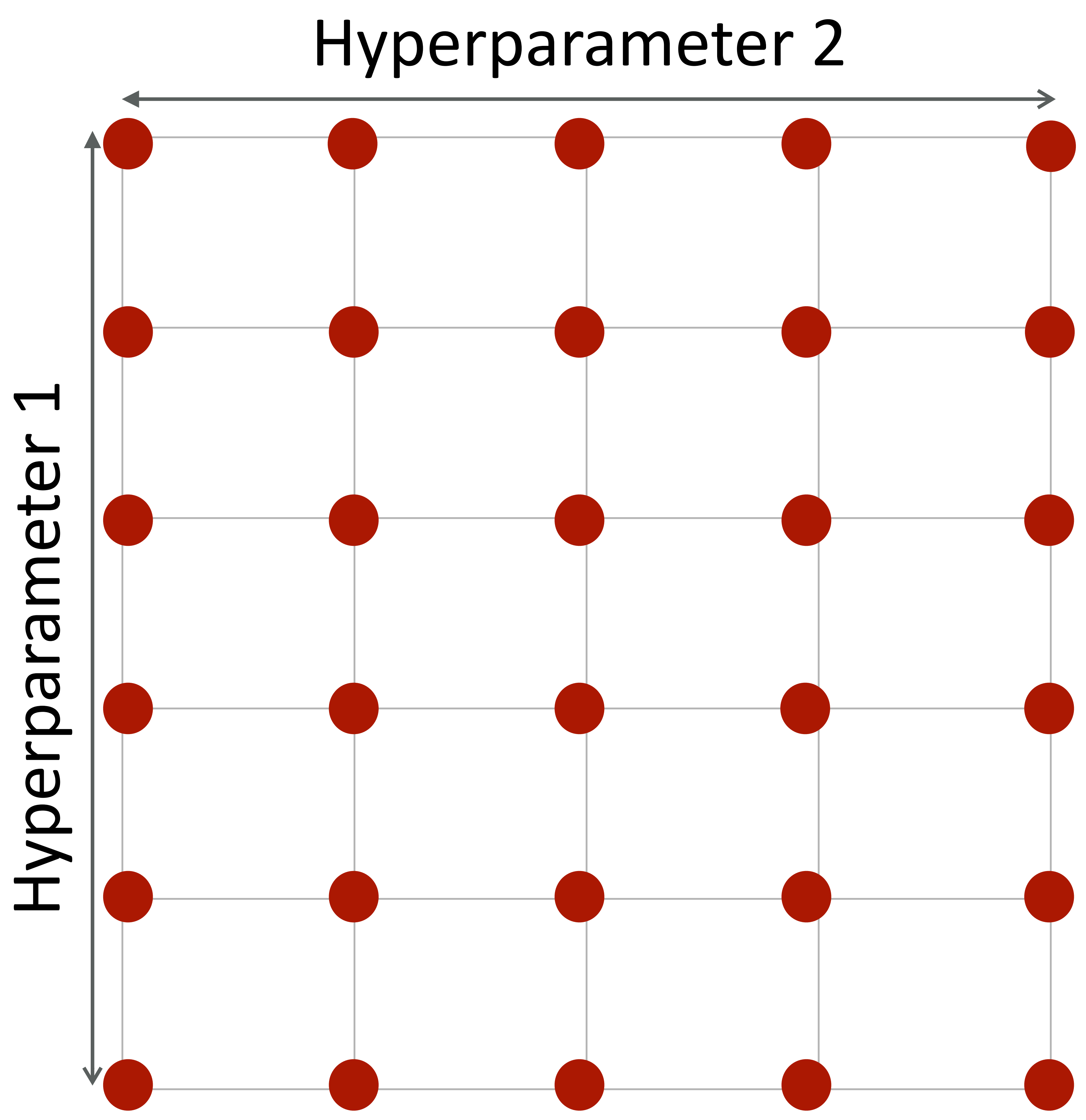
- *Training*: train various models
- *Validation*: evaluate various models (e.g., Grid Search)
- Test: evaluate final model's accuracy



Grid Search: Exhaustively search through hyperparameter space

- Define and discretize search space (linear or log scale)
- Evaluate points via validation error

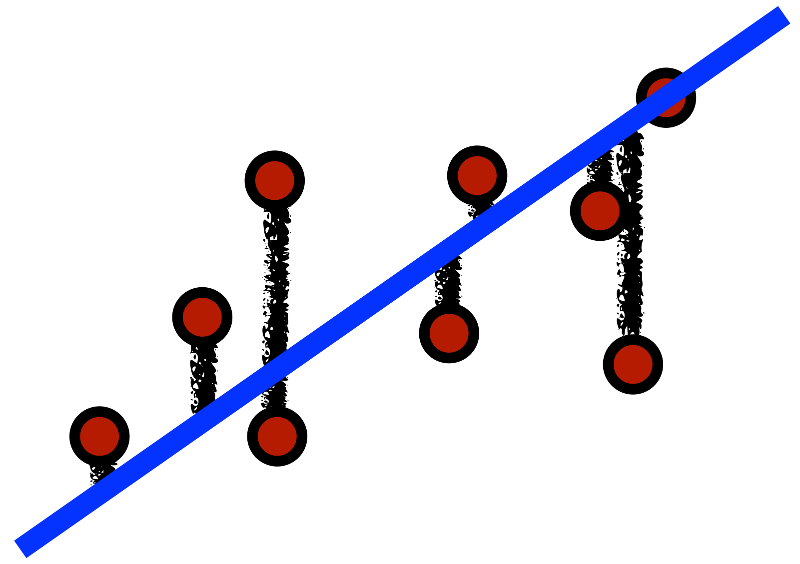




Grid Search: Exhaustively search through hyperparameter space

- Define and discretize search space (linear or log scale)
- Evaluate points via validation error

Evaluating Predictions



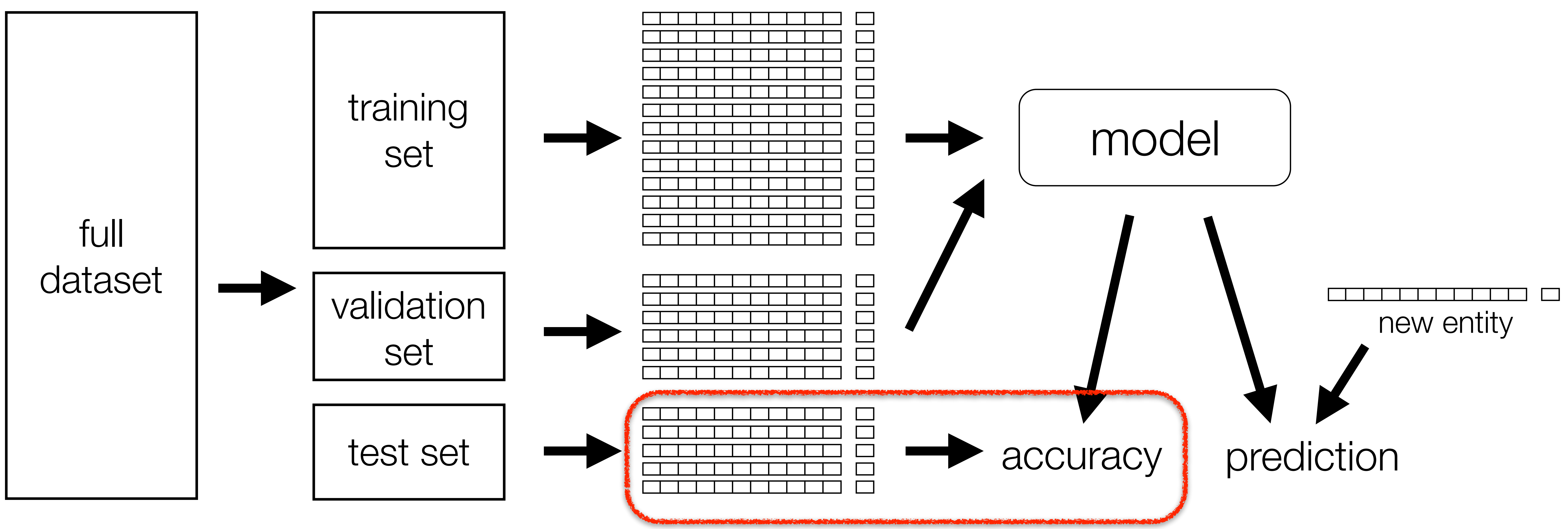
How can we compare labels and predictions for n validation points?

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

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

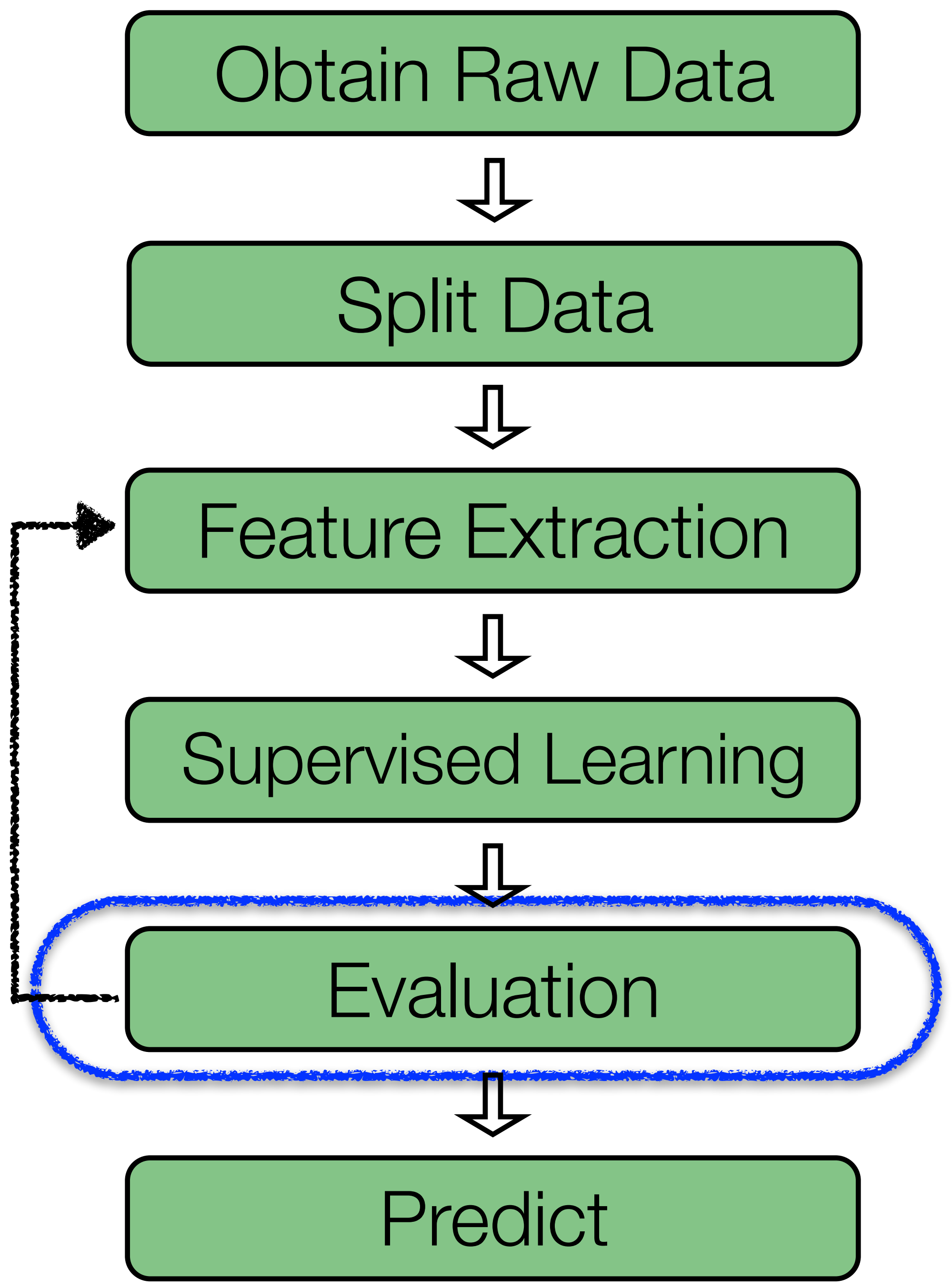
But MSE's unit of measurement is square of quantity being measured, e.g., "squared years" for song prediction

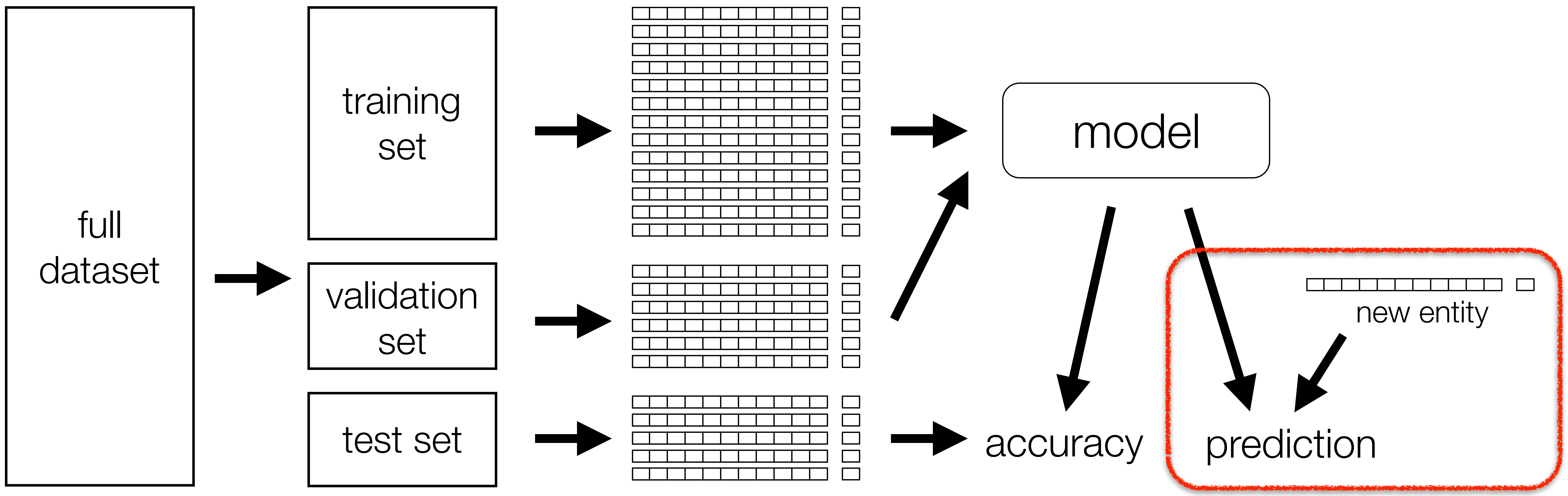
More natural to use root-mean-square error (**RMSE**), i.e., $\sqrt{\text{MSE}}$



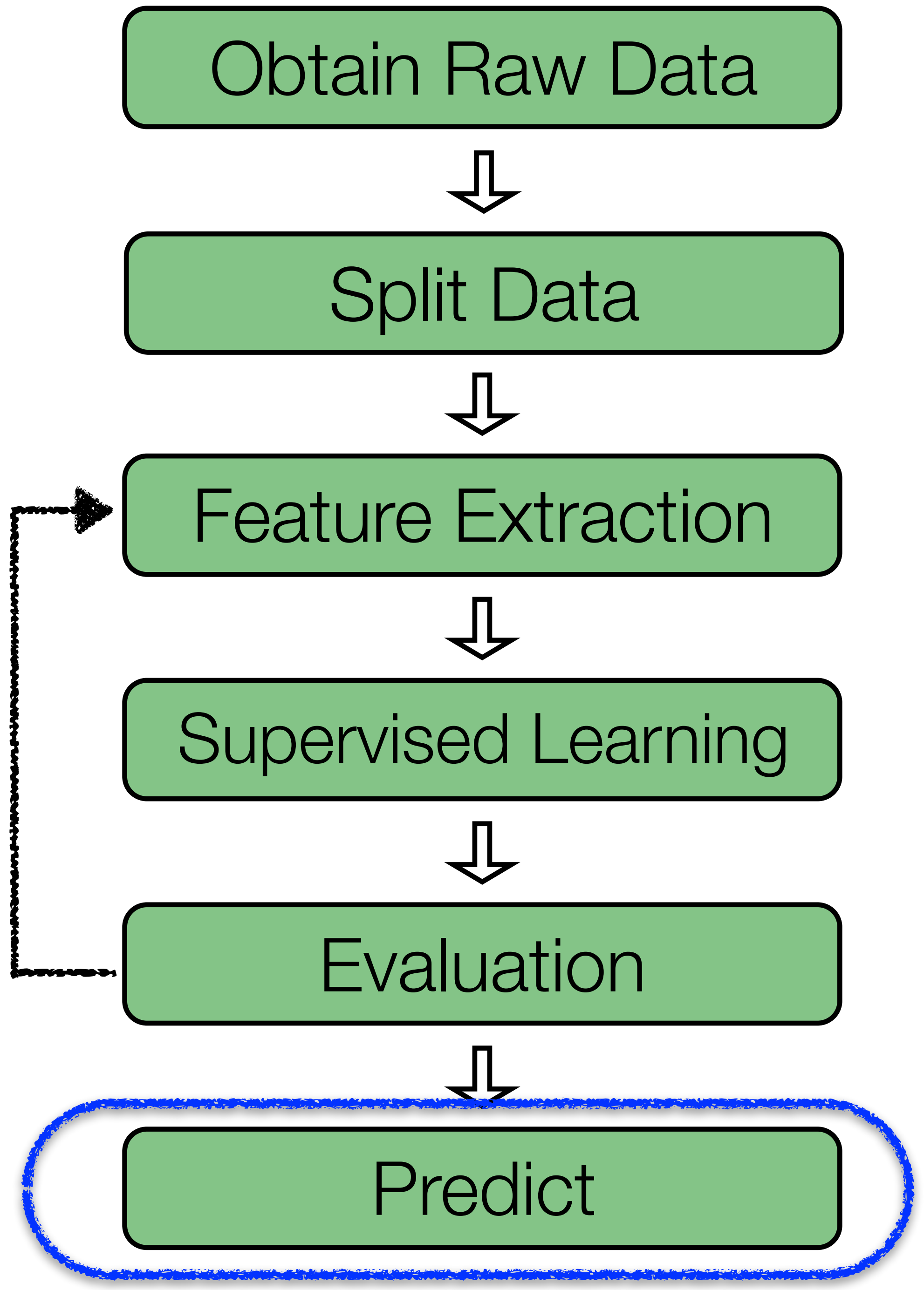
Evaluation (Part 2): Evaluate final model

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





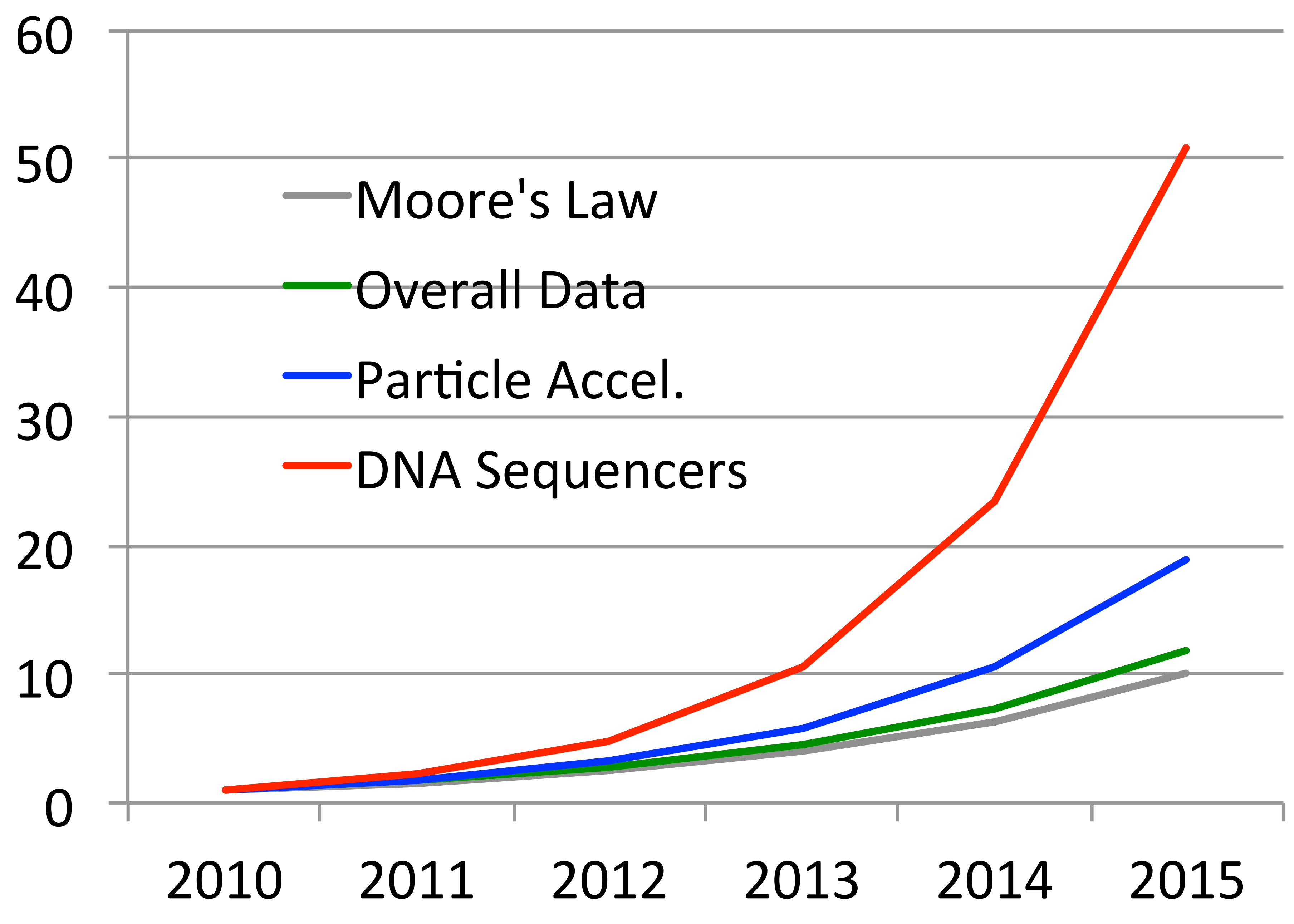
Predict: Final model can then be used to make predictions on future observations, e.g., new songs



Distributed ML: Computation and Storage

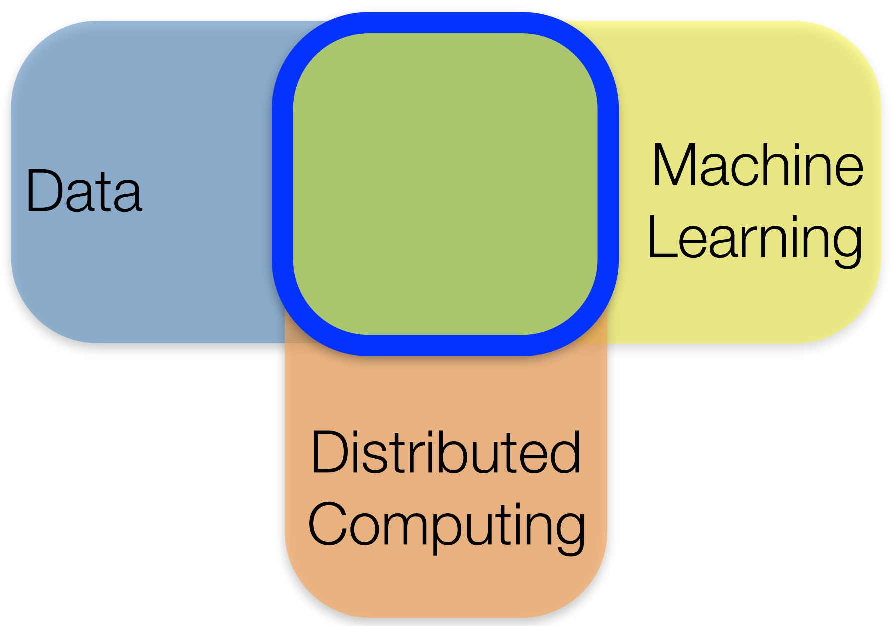
Challenge: Scalability

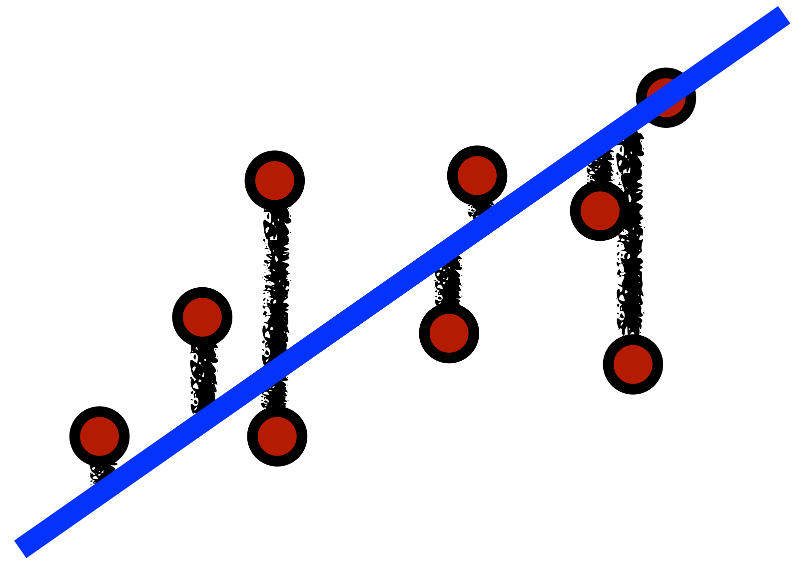
Classic ML techniques are not always suitable for modern datasets



Data Grows Faster than Moore's Law

[IDC report, Kathy Yelick, 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 (+, −, ×, /)

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

Storage: O($nd + d^2$) floats

Consider storing values as floats (8 bytes)

Storage bottlenecks:

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

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

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

Matrix Multiplication via Inner Products

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

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

Matrix Multiplication via Inner Products

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

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

Matrix Multiplication via Outer Products

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

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

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

Matrix Multiplication via Outer Products

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

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

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

Matrix Multiplication via Outer Products

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

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

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

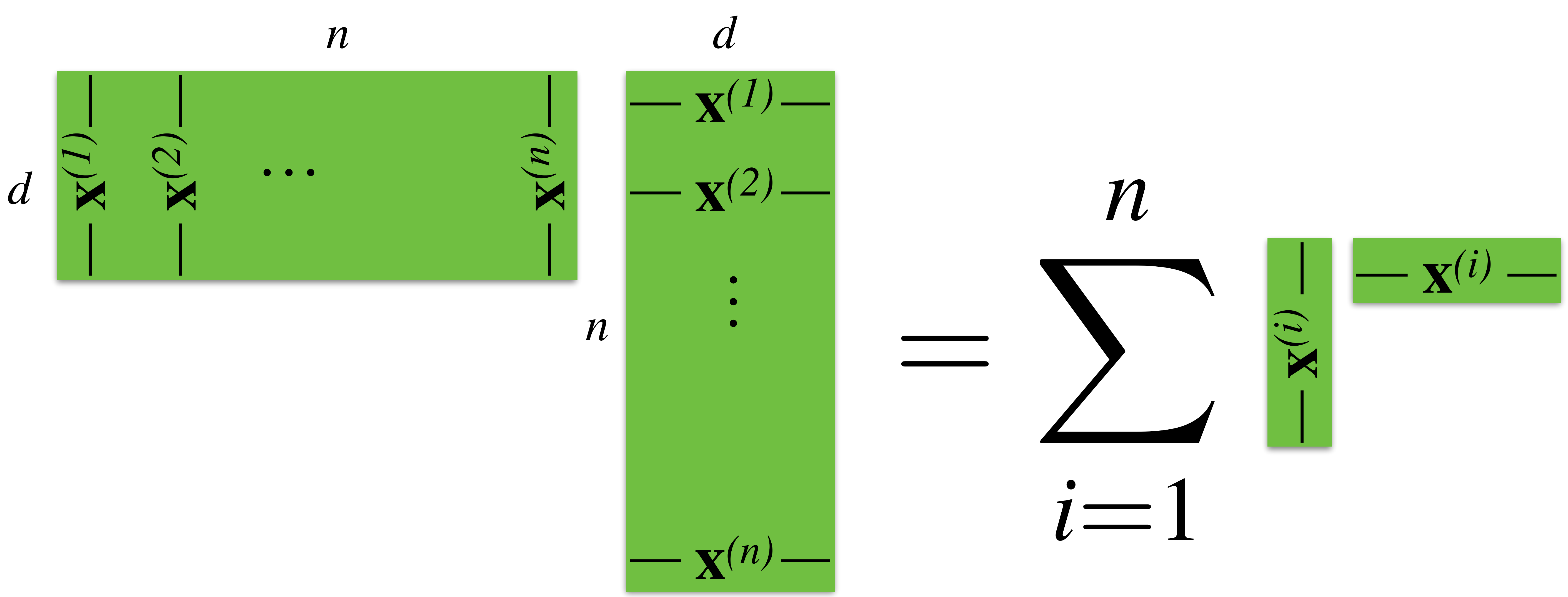
Matrix Multiplication via Outer Products

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

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

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

$$\mathbf{X}^\top \mathbf{X} =$$

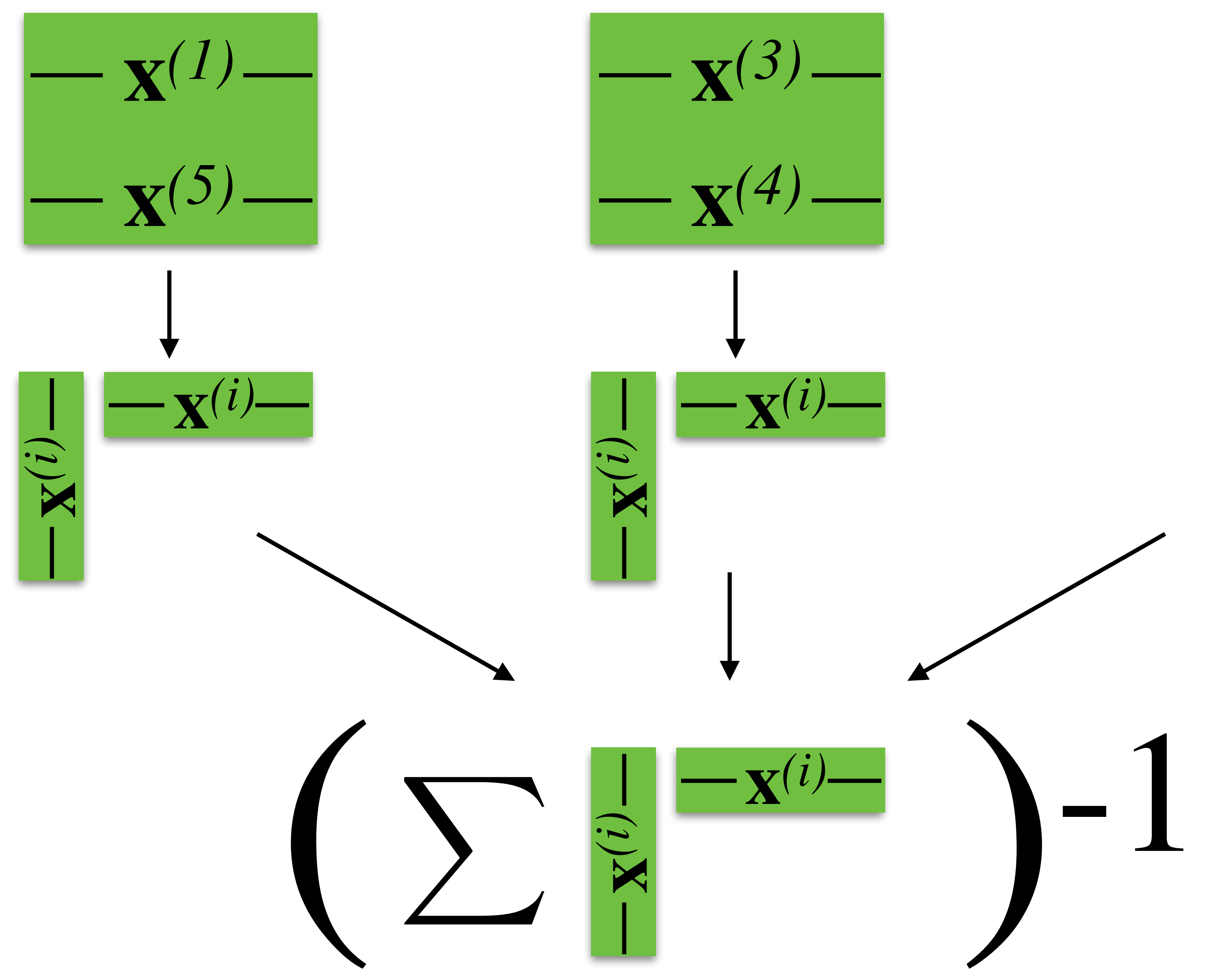


Example: n = 6; 3 workers

workers:

map:

reduce:



O(nd) Distributed Storage

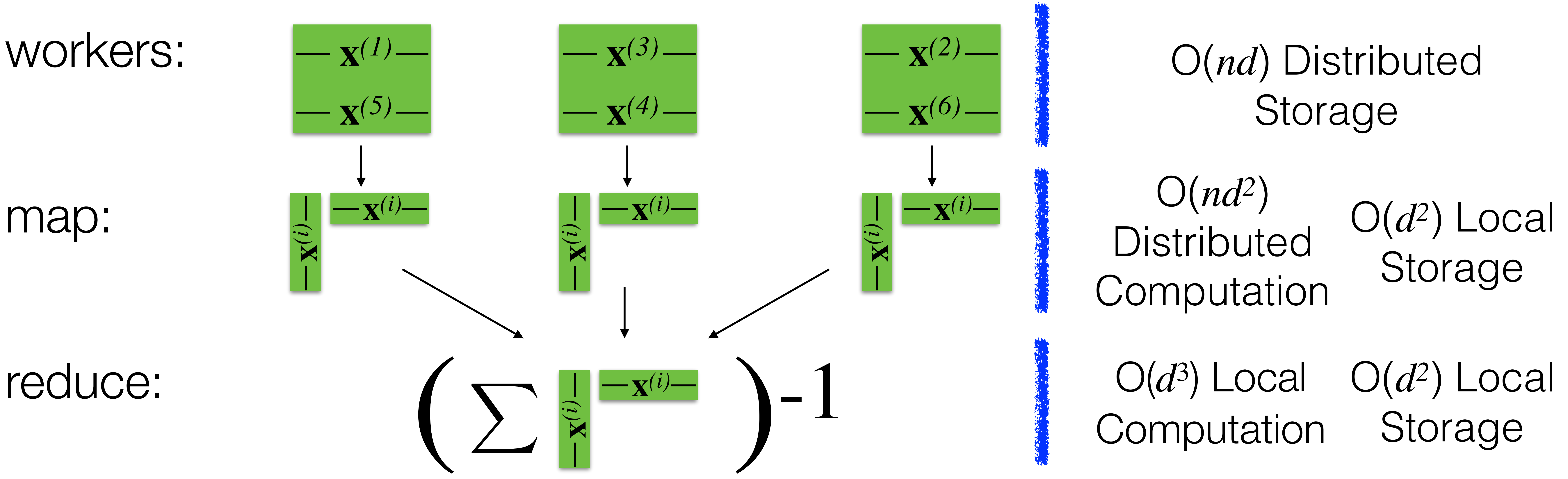
O(nd^2) Distributed Computation

O(d^2) Local Storage

O(d^3) Local Computation

O(d^2) Local Storage

```
> 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($nd^2 + d^3$) operations

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

Storage: O($nd + d^2$) floats

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

Big n and Big d

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

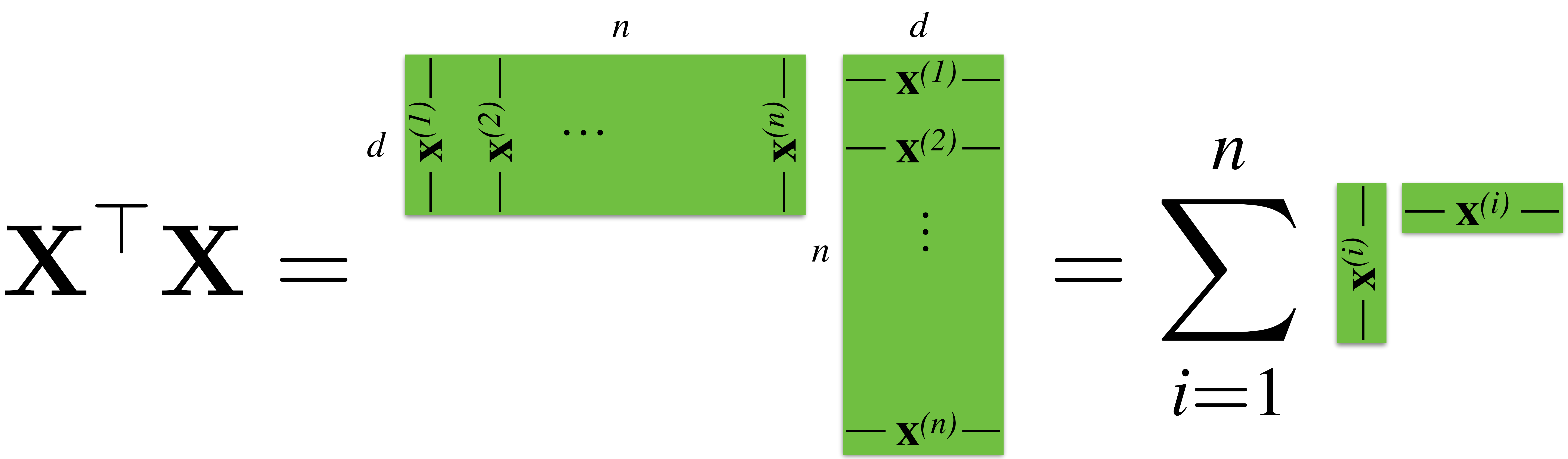
Computation: $O(\underline{nd^2} + \underline{d^3})$ operations

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

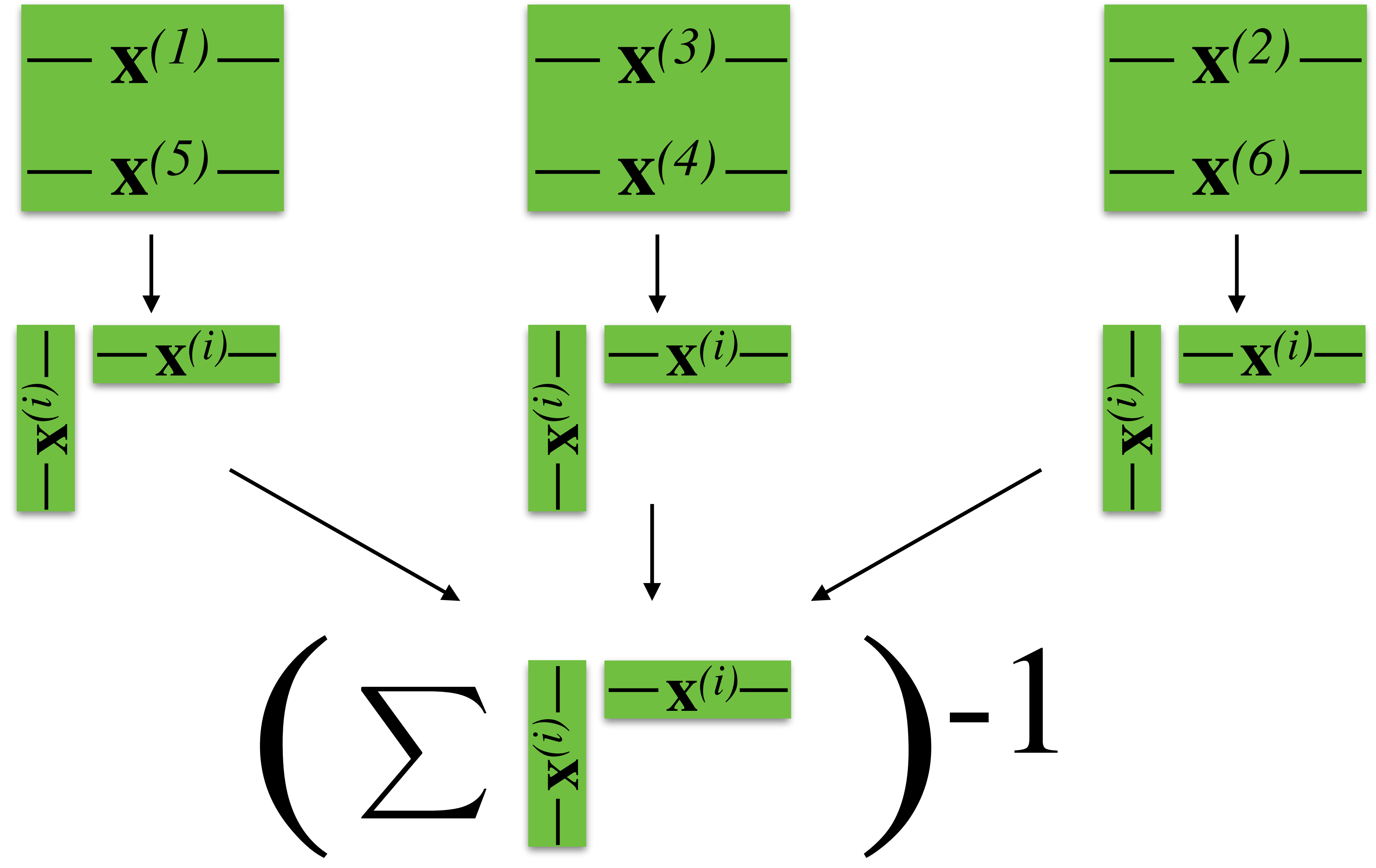


Example: n = 6; 3 workers

workers:

map:

reduce:



O(nd) Distributed Storage

O(nd^2) Distributed Computation

O(d^2) Local Storage

O(d^3) Local Computation

O(d^2) Local Storage

Big n and Big d

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

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

Storage: $O(nd + d^2)$ 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

$$\text{dense} : \; 1. \;\; 0. \;\; 0. \;\; 0. \;\; 0. \;\; 0. \;\; 3.$$

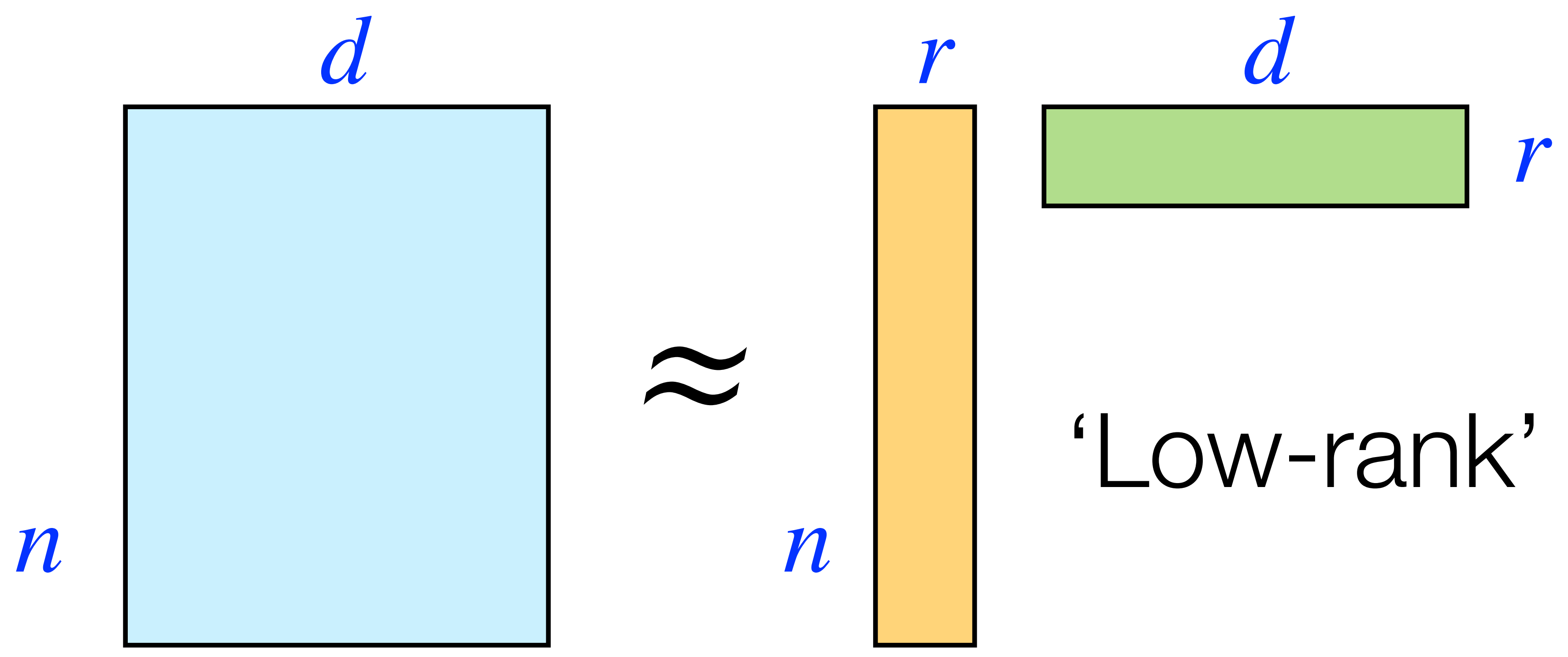
$$\text{sparse} : \begin{cases} \text{size} : 7 \\ \text{indices} : 0 \;\; 6 \\ \text{values} : 1. \;\; 3. \end{cases}$$

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

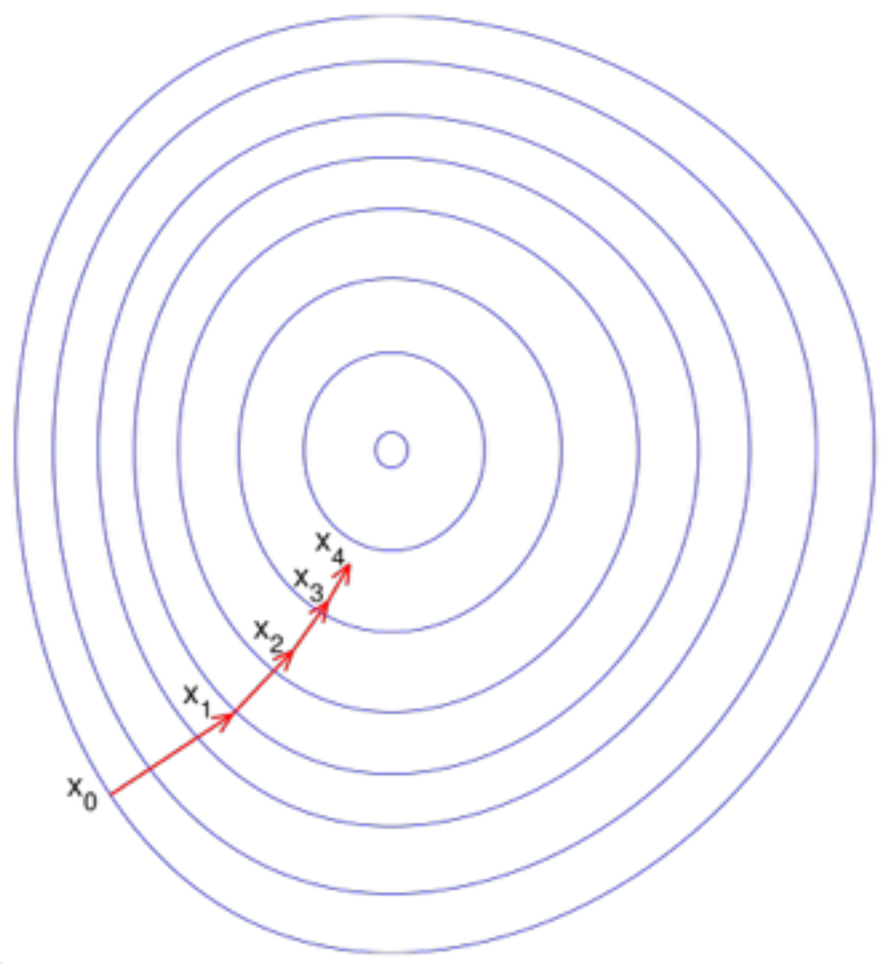
We need methods that are linear in time and space

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



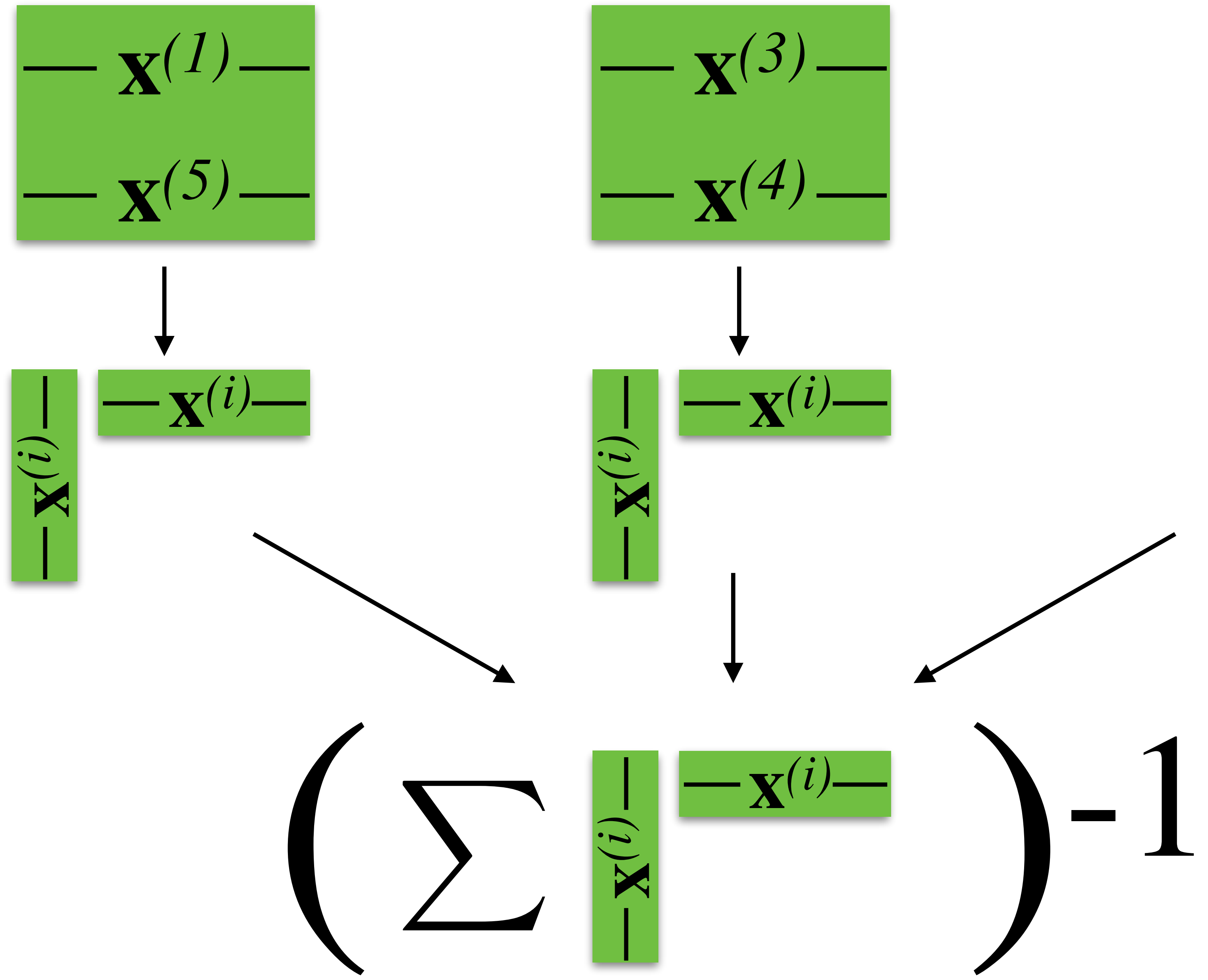
Closed Form Solution for Big n and Big d

Example: $n = 6$; 3 workers

workers:

map:

reduce:



O(nd) Distributed Storage

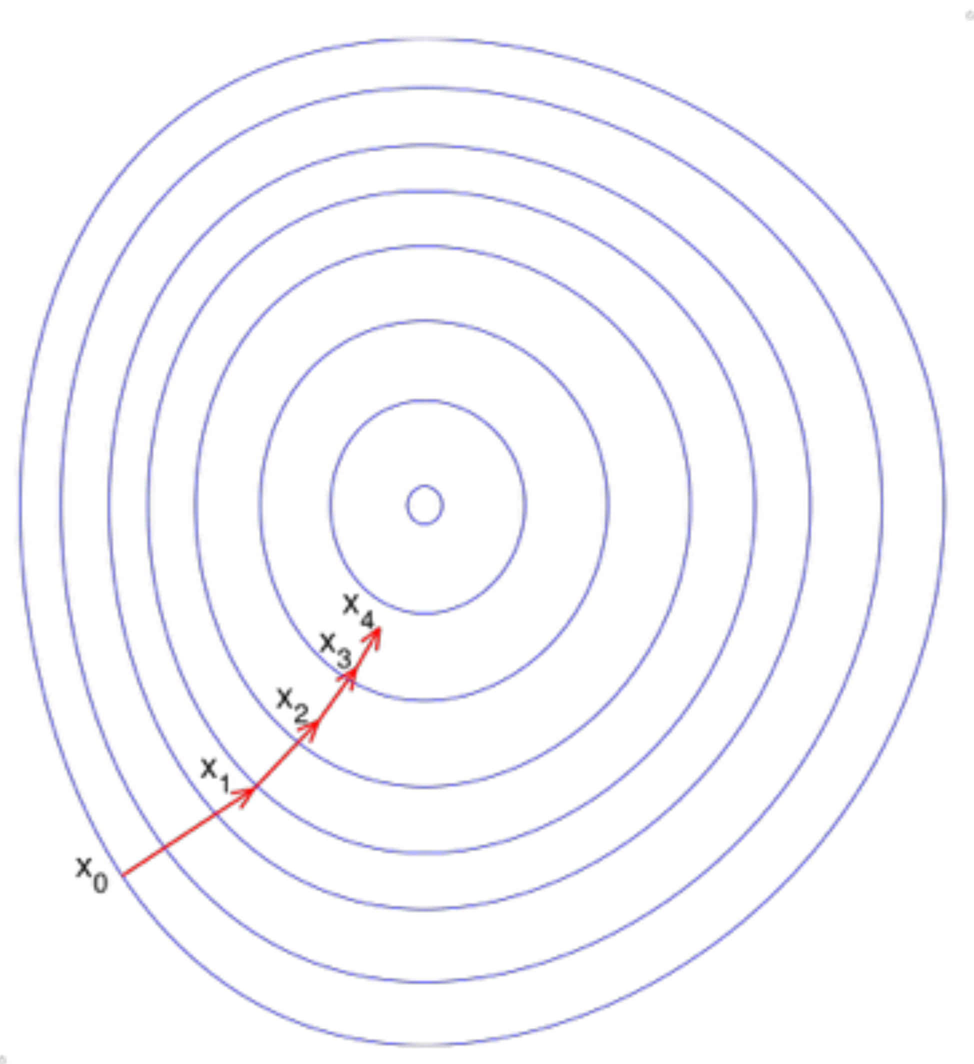
O(nd^2) Distributed Computation

O(d^2) Local Storage

O(d^3) Local Computation

O(d^2) Local Storage

Gradient Descent for Big n and Big d



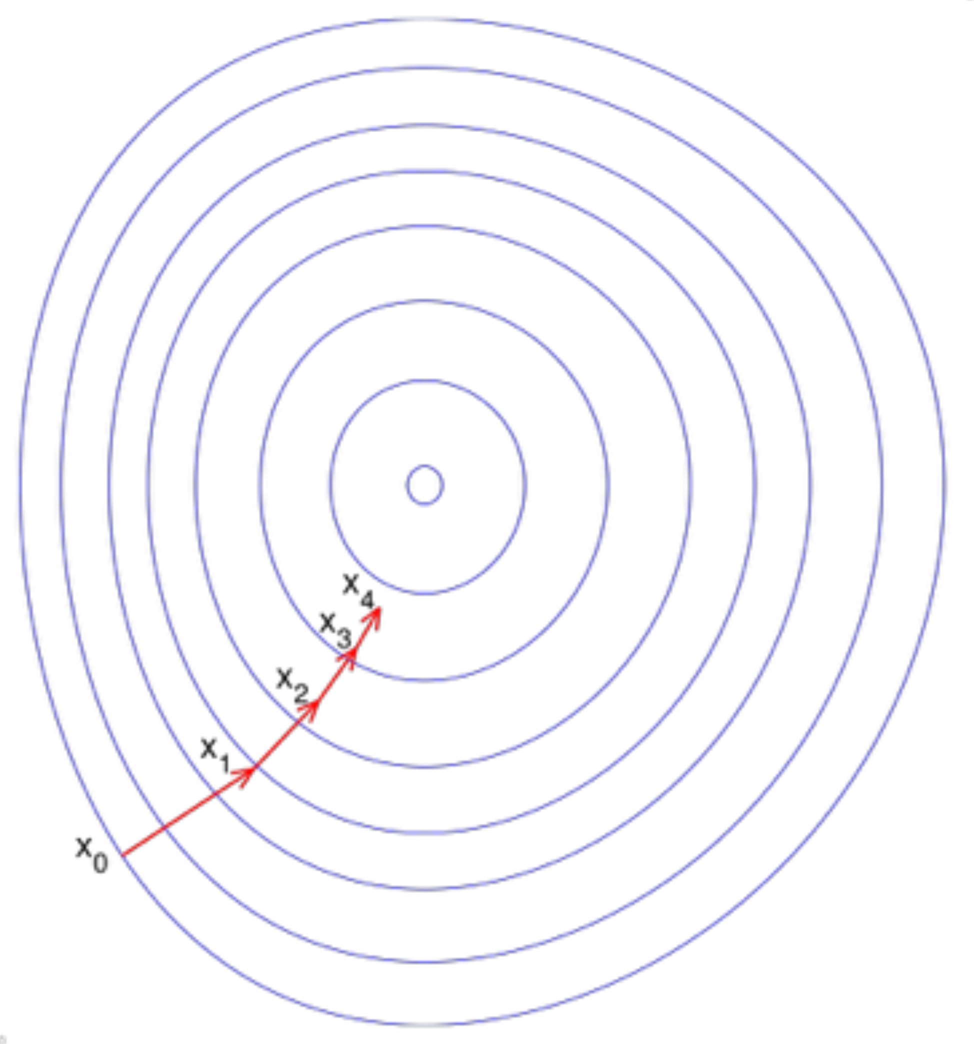
Example: n = 6; 3 workers

workers: $\begin{bmatrix} - \mathbf{x}^{(1)} - \\ - \mathbf{x}^{(5)} - \end{bmatrix}$ $\begin{bmatrix} - \mathbf{x}^{(3)} - \\ - \mathbf{x}^{(4)} - \end{bmatrix}$ $\begin{bmatrix} - \mathbf{x}^{(2)} - \\ - \mathbf{x}^{(6)} - \end{bmatrix}$ — $O(nd)$ Distributed Storage

map: $\mathbf{x}^{(i)} \, \mathbf{x}^{(i)\top}$ $\quad \mathbf{x}^{(i)} \, \mathbf{x}^{(i)\top}$ $\quad \mathbf{x}^{(i)} \, \mathbf{x}^{(i)\top}$ — $O(nd^2)$ Distributed Computation; $O(d^2)$ Local Storage

reduce: $\left(\sum \mathbf{x}^{(i)} \, \mathbf{x}^{(i)\top} \right)^{-1}$ — $O(d^3)$ Local Computation; $O(d^2)$ Local Storage

Gradient Descent for Big n and Big d



Example: $n = 6$; 3 workers

workers: — $\mathbf{x}^{(1)}$ — — $\mathbf{x}^{(5)}$ — | — $\mathbf{x}^{(3)}$ — — $\mathbf{x}^{(4)}$ — | — $\mathbf{x}^{(2)}$ — — $\mathbf{x}^{(6)}$ —

O(nd) Distributed Storage

map: ? ? ?

O($\boldsymbol{nd}$) ~~O(nd^2)~~ Distributed Computation

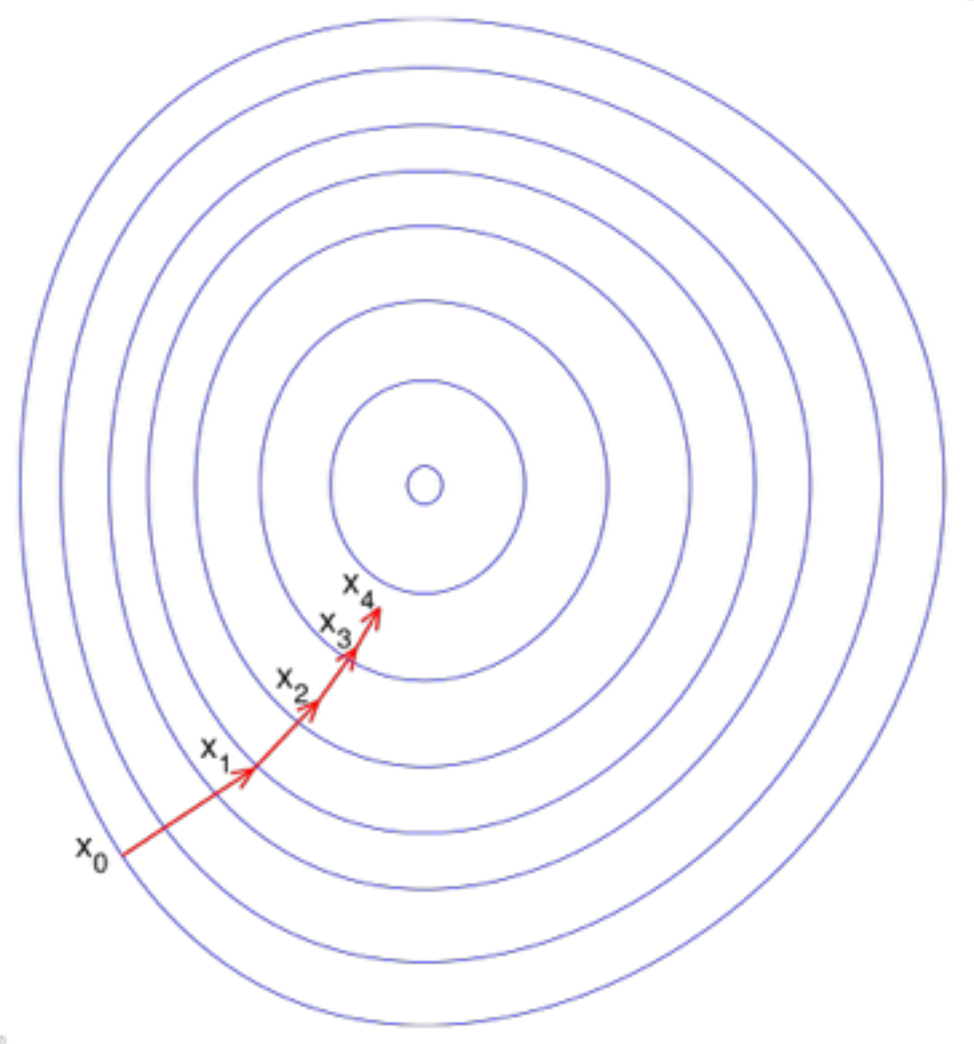
O($\boldsymbol{d}$) ~~O(d^2)~~ Local Storage

reduce: $\left(\sum \left[\,\mathbf{x}^{(i)}\,\right]\left[\,\mathbf{x}^{(i)}\,\right]\right)^{-1}$

O(d^3) Local Computation

O(d^2) Local Storage

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: ? ? ?

reduce: ?

O(nd) Distributed Storage

O($\boldsymbol{nd}$) $\cancel{O(nd^2)}$ Distributed Computation

O($\boldsymbol{d}$) $\cancel{O(d^2)}$ Local Storage

O($\boldsymbol{d}$) $\cancel{O(d^3)}$ Local Computation

O($\boldsymbol{d}$) $\cancel{O(d^2)}$ Local Storage