Announcements

- PS-1 due today.
- PS-2 will be out on Thursday.
- Questions about submission process, problem sets - on Piazza.
- If you know the answer to someone's questions - respond.

No laptops, screens during the class.

Last class

- Gradient descent
- Decision tree introduction
- Quantifying Entropy

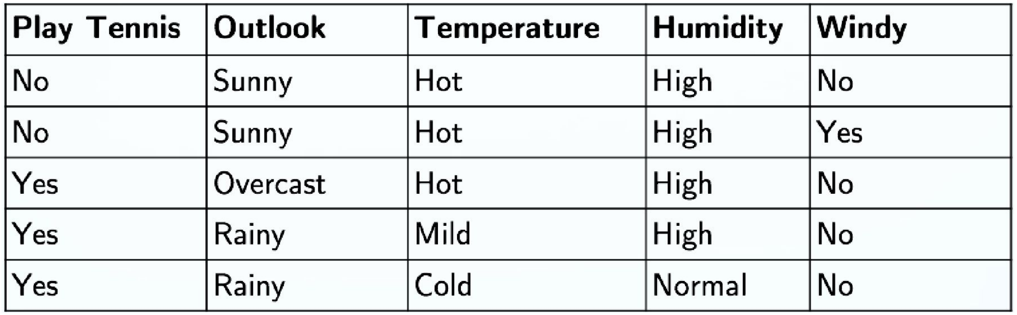
Gradient Descent

- Start somewhere = random initialization.
- Compute slope = compute gradient of the cost function wrt parameters.
- Take a step towards steepest direction
- Repeat, for certain steps, stopping criteria.

Summary: Algorithm for Decision Trees

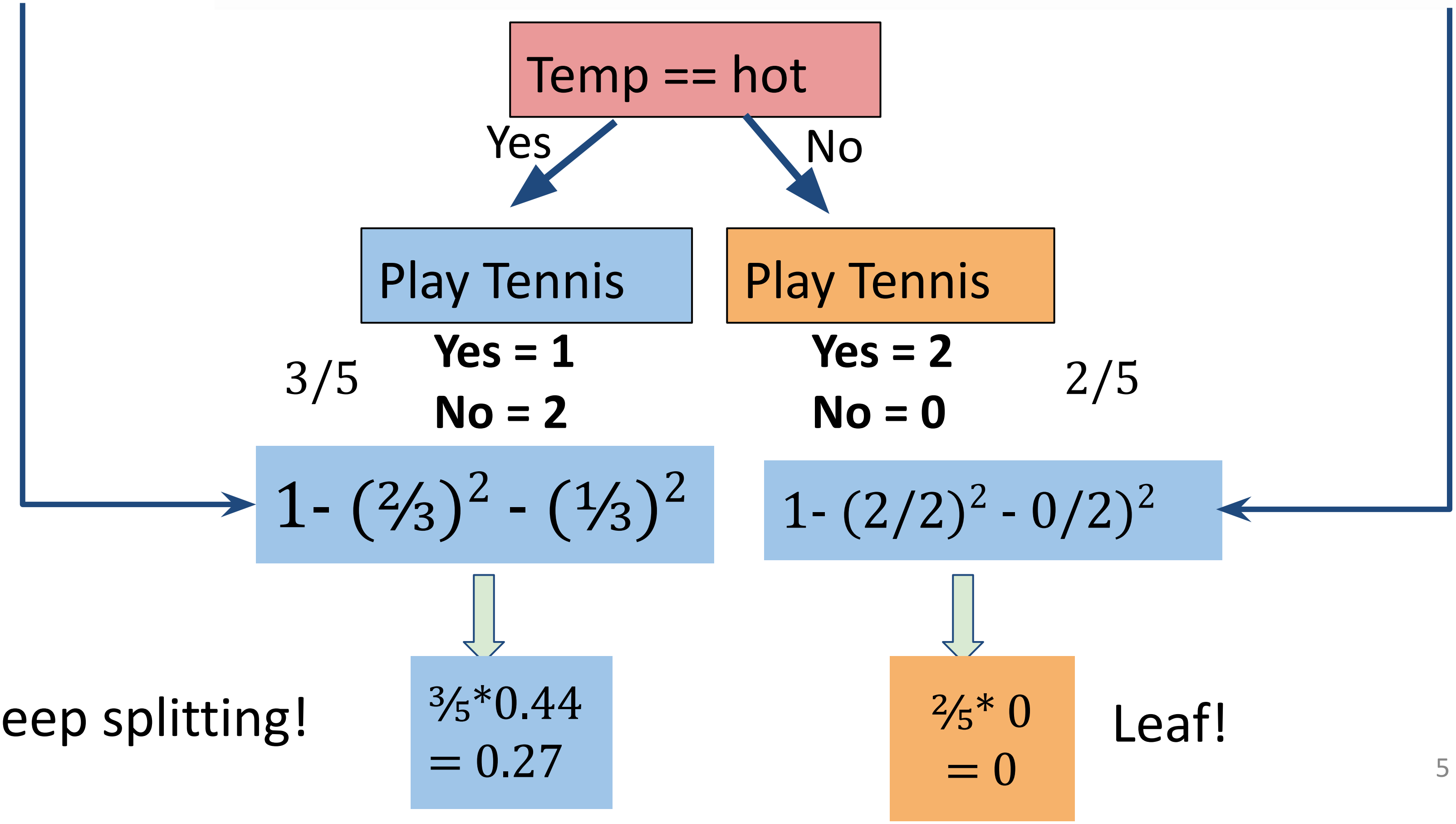
1. Start with an empty decision tree (undivided feature space).
2. Choose the ‘optimal’ variable on which to split and choose the ‘optimal’ threshold value for splitting.
    1. Often happens by going through all variables.
3. Recurse on each new node until stopping condition is met
4. For classification, we label each region in the model with the label of the class to which the plurality of the points within the region belong.

How to choose which variable to split on?



Play Tennis	Outlook	Temperature	Humidity	Windy
No	Sunny	Hot	High	No
No	Sunny	Hot	High	Yes
Yes	Overcast	Hot	High	No
Yes	Rainy	Mild	High	No
Yes	Rainy	Cold	Normal	No

Gini Impurity = 1 - (the probability of “Yes”)^2 - (the probability of “No”)^2



Temp == hot

Yes

No

Play Tennis

Yes = 1

No = 2

3/5

Play Tennis

Yes = 2

No = 0

2/5

$1 - (2/3)^2 - (1/3)^2$

$1 - (2/2)^2 - 0/2)^2$

$3/5 * 0.44 = 0.27$

$2/5 * 0 = 0$

Keep splitting!

Leaf!

Question from last class: why is entropy defined as log base 2

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

- No special reason - can be converted from log base 2 to log base 10 or vice-versa.

- Origins of entropy : field of communication and information theory.
  - Data was represented in terms of bits, hence base 2.

Today

- **Bagging and Random Forests**
- Feature normalization
- Curse of dimensionality

When to stop training decision trees?

- All data in a given node has the same label.
  - Return that label
- Data in a given node has too few examples
  - Return majority label of all data
- The tree has reached a predefined maximum depth.
  - A given node may not have all data-points belonging to the same class.
  - Some nodes may have a lot of examples.

Disadvantages of Decision Trees

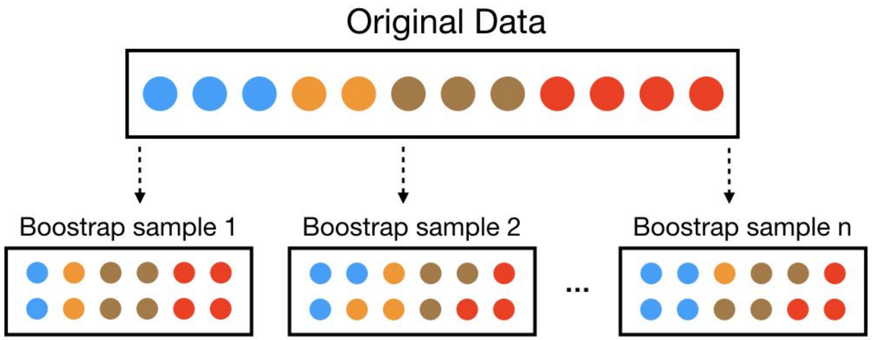
- Very sensitive to
  - noise in the data.
  - Skewed dataset.
- Prone to overfitting

Two ways to reduce model complexity

- Build out the tree - the prune in a post-hoc manner
- Early stopping *while* building the tree.

Bootstrap Aggregating: wisdom of the crowd

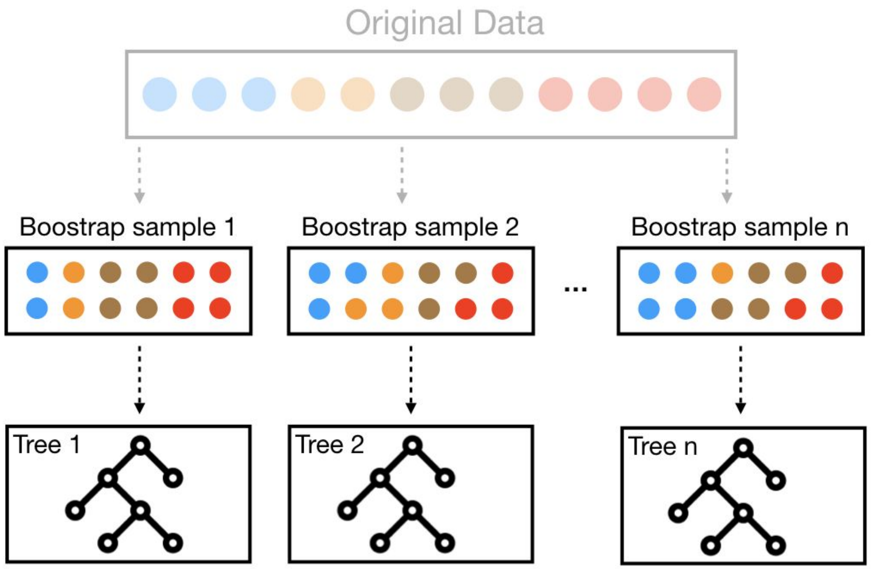
1. Sample with replacement (aka “bootstrap” the training data)



Bootstrap Aggregating: wisdom of the crowd

1. Sample with replacement (aka “bootstrap” the training data)

2. Fit an overgrown tree to each resampled data set

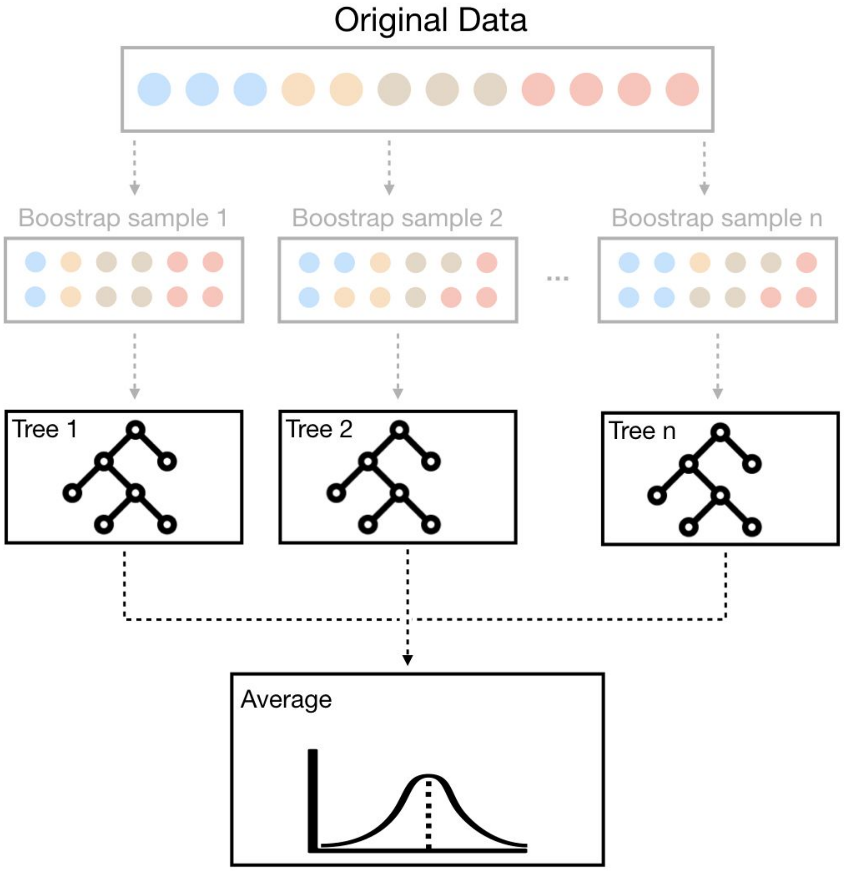


Bootstrap Aggregating: wisdom of the crowd

1. Sample with replacement (aka "bootstrap" the training data)

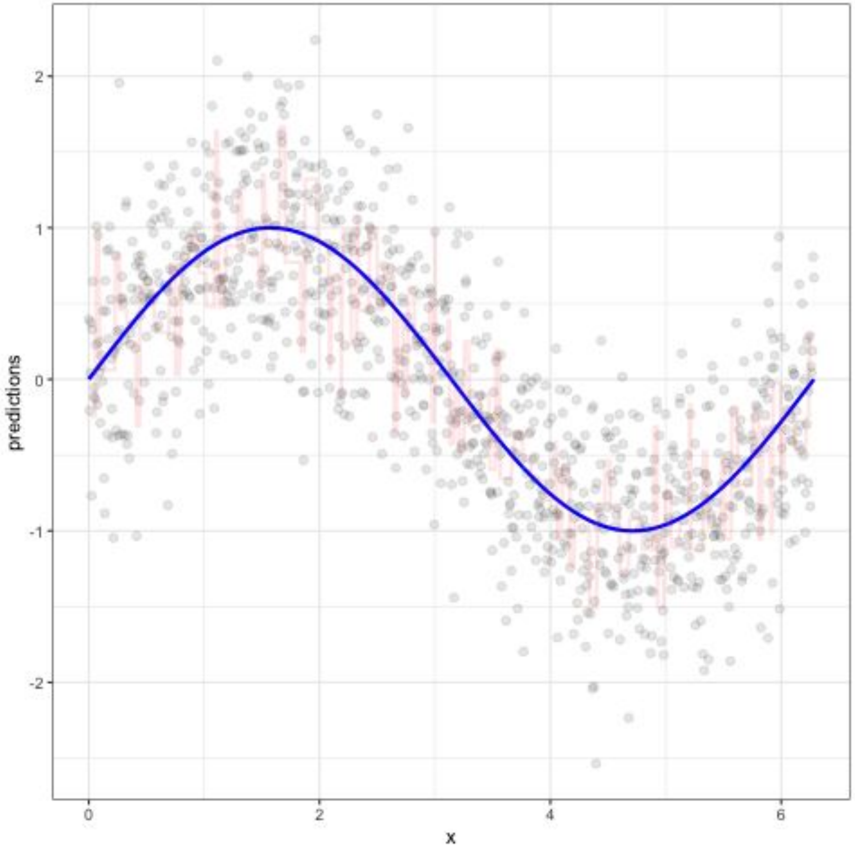
2. Fit an overgrown tree to each resampled data set

3. Average predictions

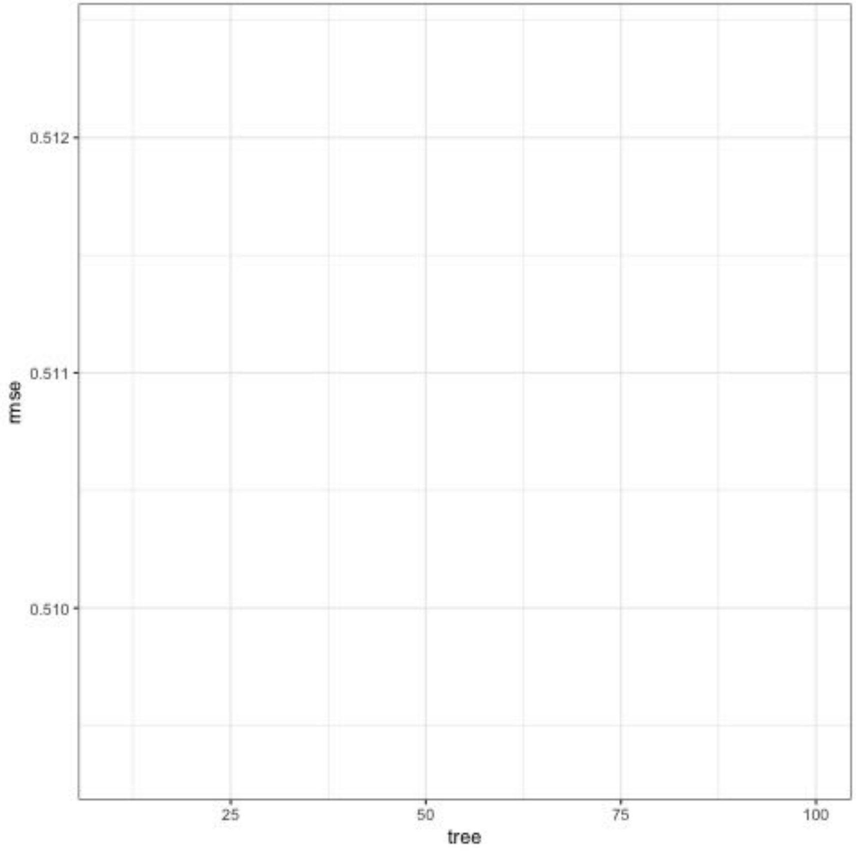


Bootstrap Aggregating: wisdom of the crowd

As more trees are added….



prediction error reduces

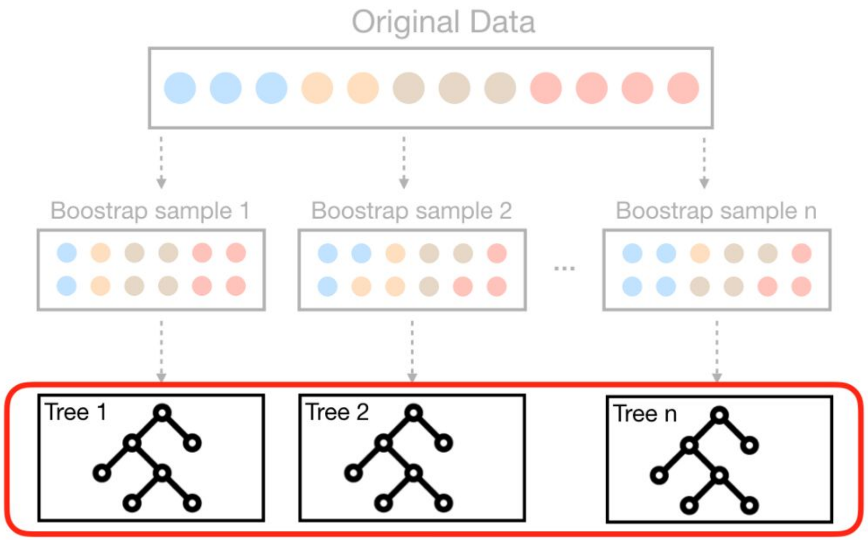


Pros of Bagging

- Reduces overfitting
  a. Makes model more generalizable
- Reduced impact of missing values

Random Forests

- Follow a similar bagging process but…

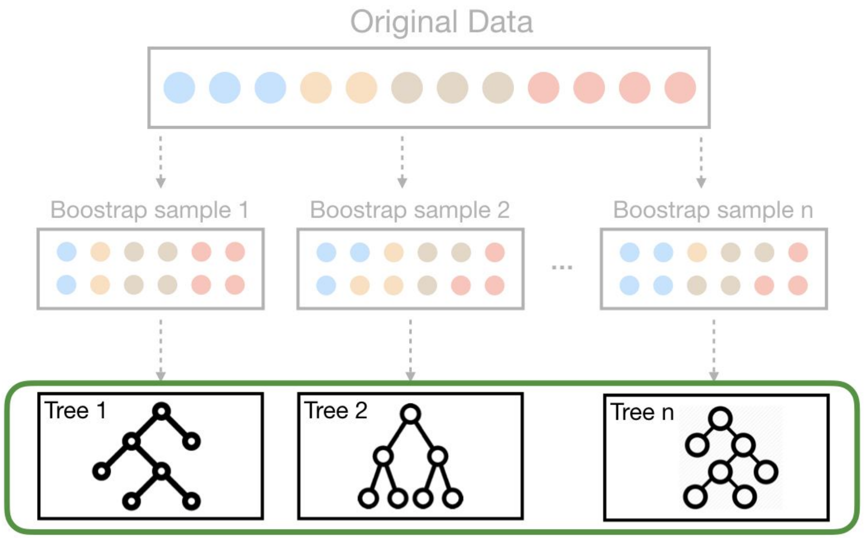


Bagging may produce many correlated trees

Random Forests

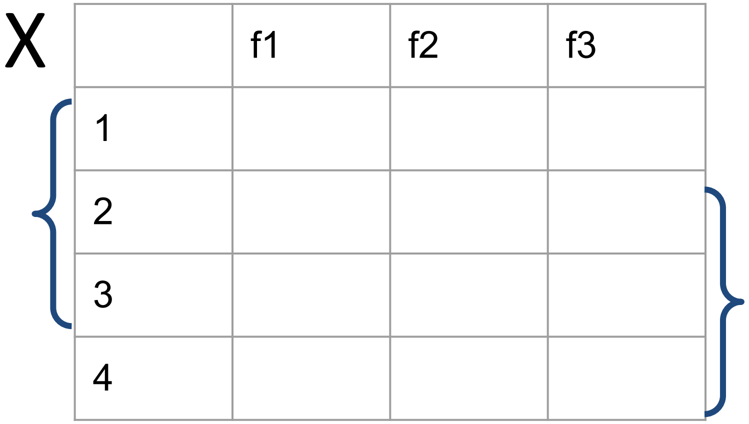
- Follow a similar bagging process but…

- For each split, the search for the split variable is ***restricted to a random subset of features***



Random Forests produce many unique trees

Bagging v/s Random Forests



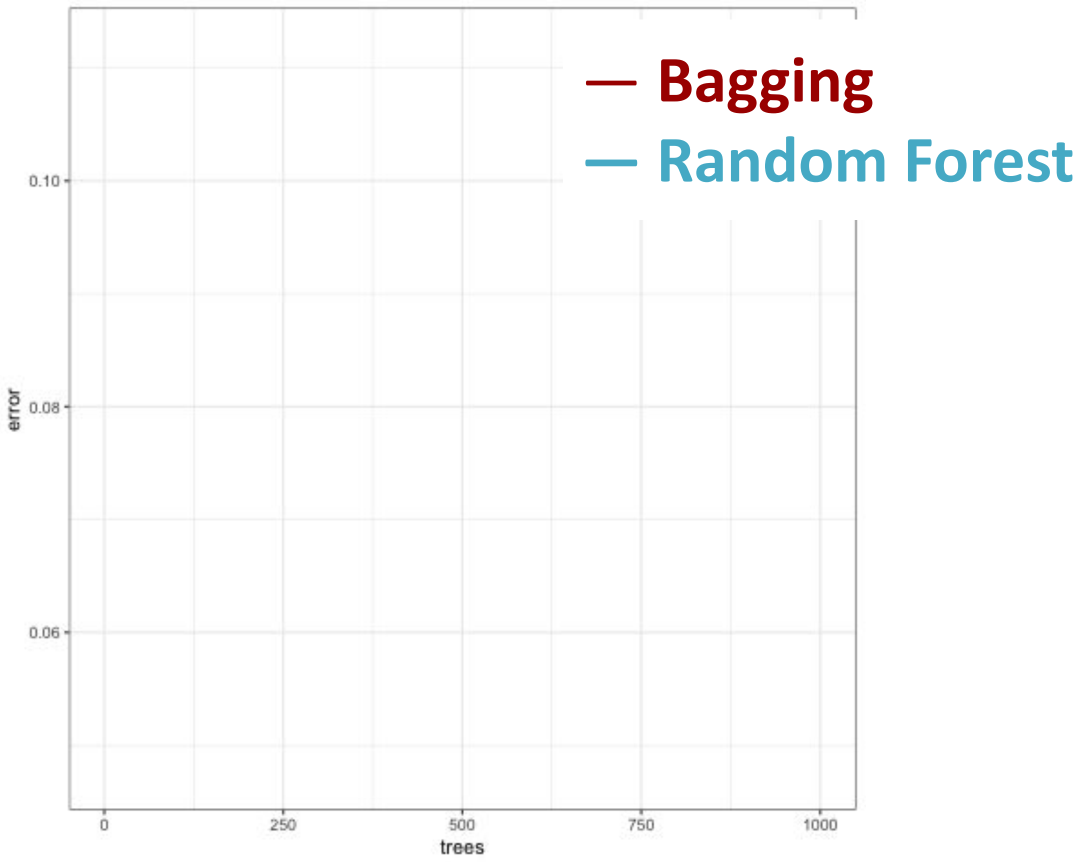
X

	f1	f2	f3
1			
2			
3			
4			

- Bagging is built on random sampling of the rows of the data

- Random forests is built on random sampling of the rows and columns of the data

Bagging v/s Random Forests



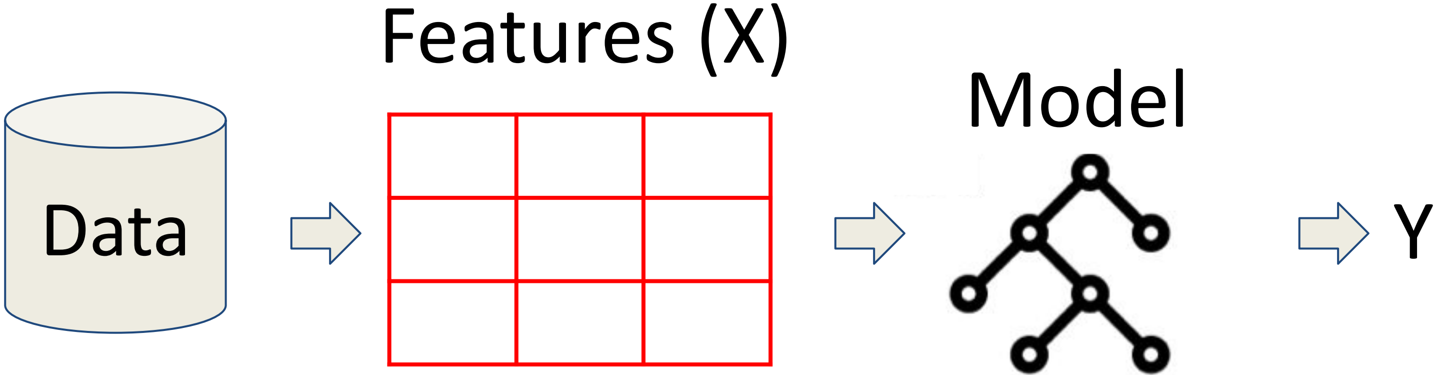
Summary

- Bagging ensembles many decision trees to reduce variance

- Bagging can result in many correlated trees

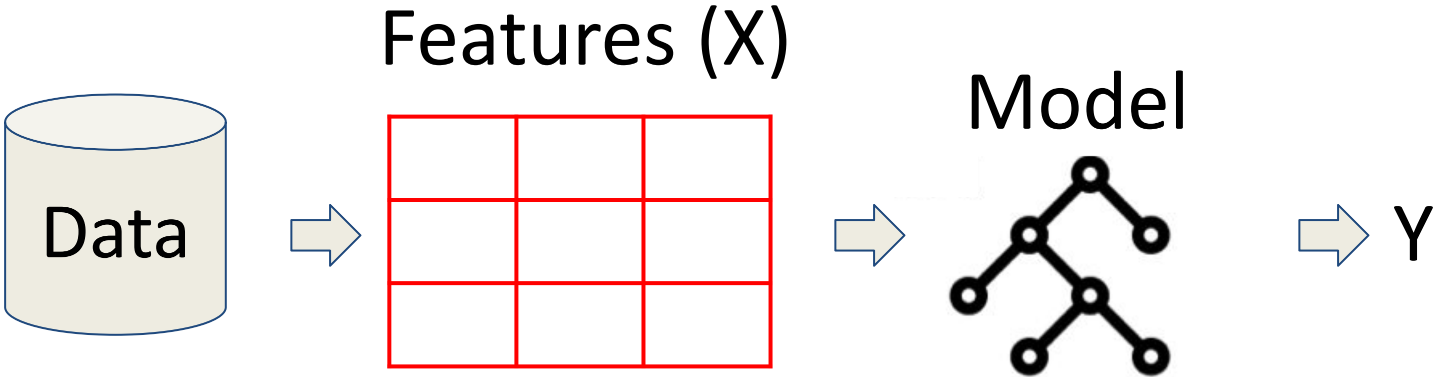
- Random forests address this by building their trees on a random subset of the features

Enemies of Machine Learning



- Some contributing factors:
  - Features have different scales.
  - Too many features
  - Too few features → simplify the model, feature engineering.

Enemies of Machine Learning



Features (X)

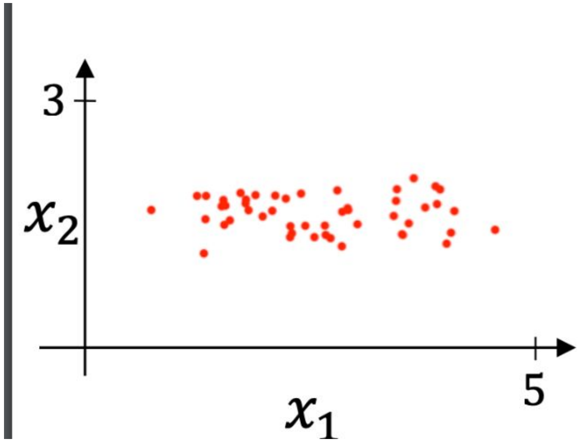
Model

Data

Y

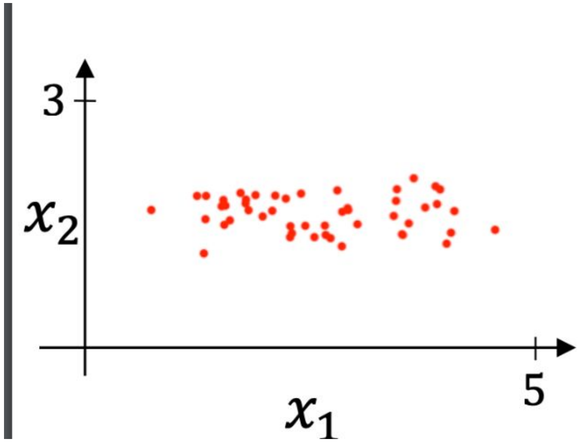
- Some contributing factors:
  - **Features have different scales.**
  - Too many features

Scale of the feature values



- **Why could this be a problem?**

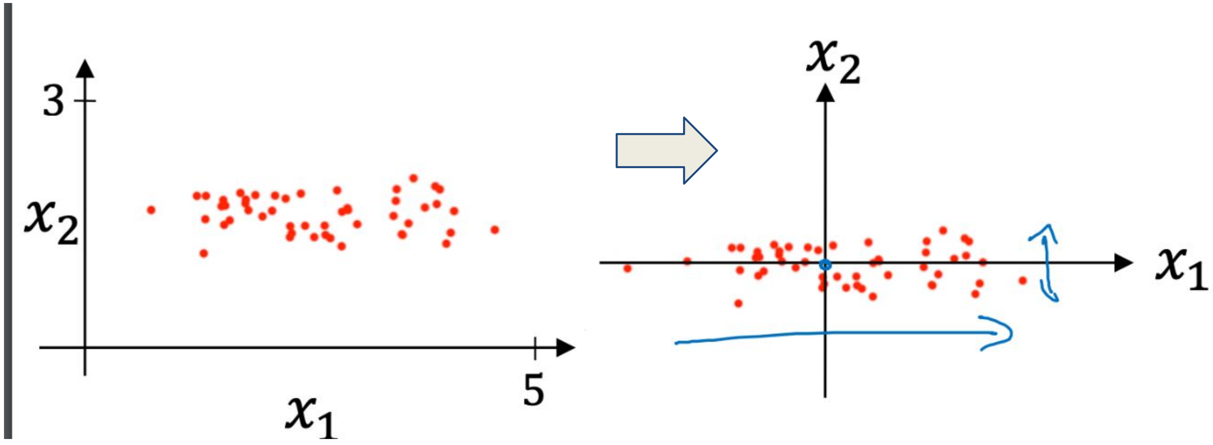
Scale of the feature values



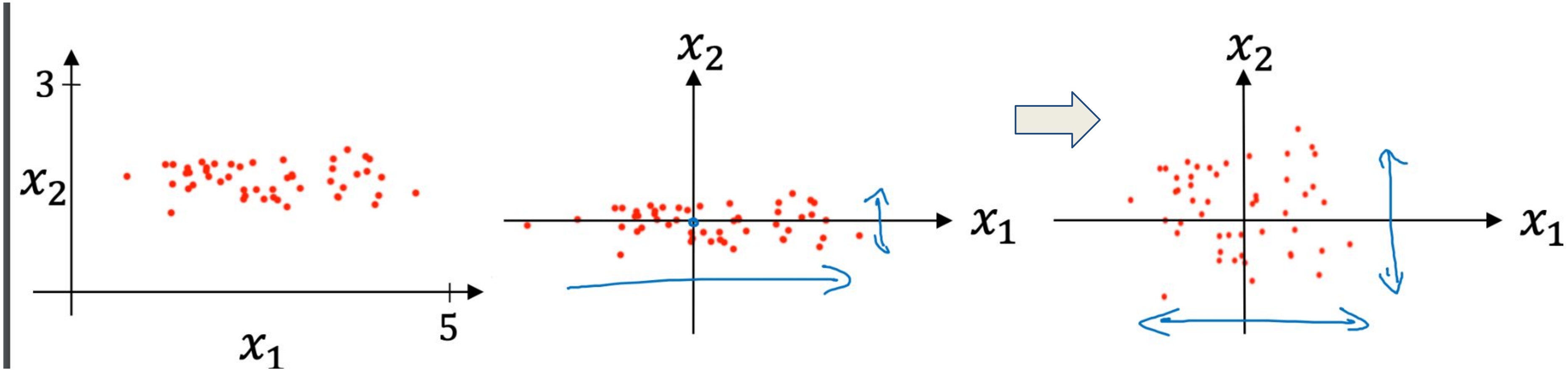
Issue: different feature scales impact the loss differently.

- Think MSE Loss.

Solution: Feature Normalization

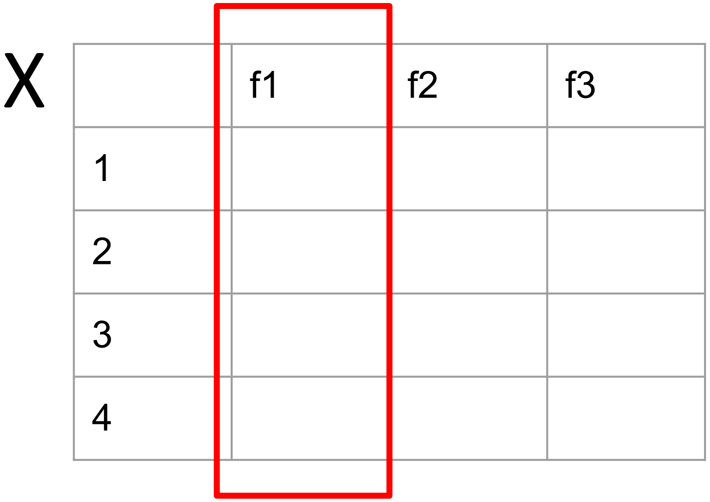


Solution: Feature Normalization



- Center the values around zero.
- Scale the values to fall between a fixed range, eg: -1 to 1.

Solution: Feature Normalization

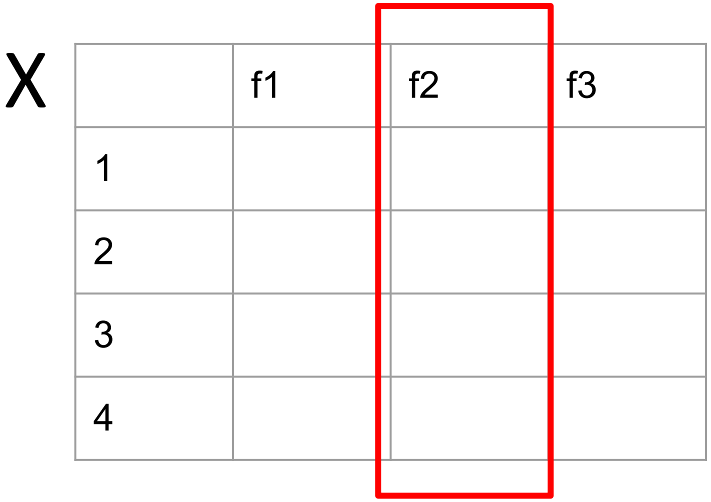


X

	f1	f2	f3
1			
2			
3			
4			

- Center the values around zero.
- Scale the values to fall between a fixed range.

Solution: Feature Normalization



X

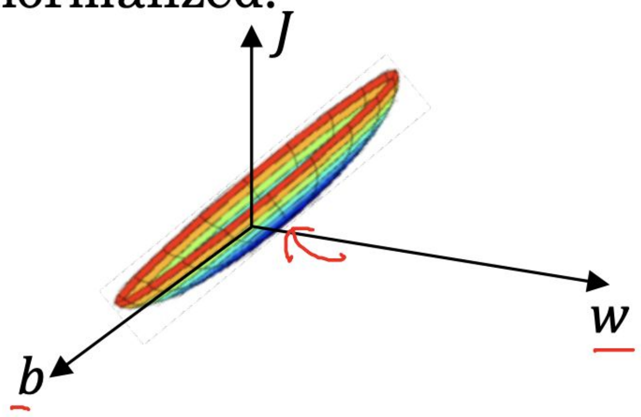
	f1	f2	f3
1			
2			
3			
4			

- Center the values around zero.
- Scale the values to fall between a fixed range.

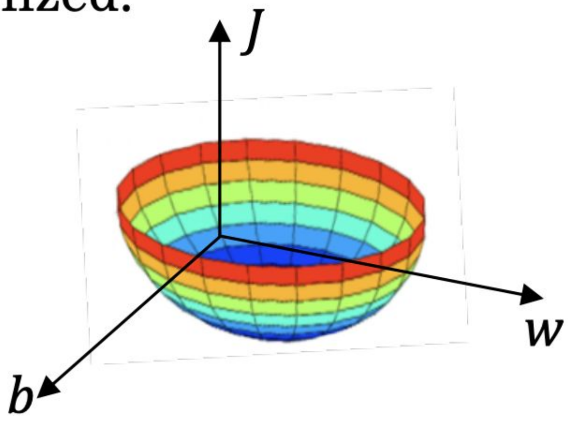
- **Operating across all rows, one column at a time**

Solution: Feature Normalization

Unnormalized:

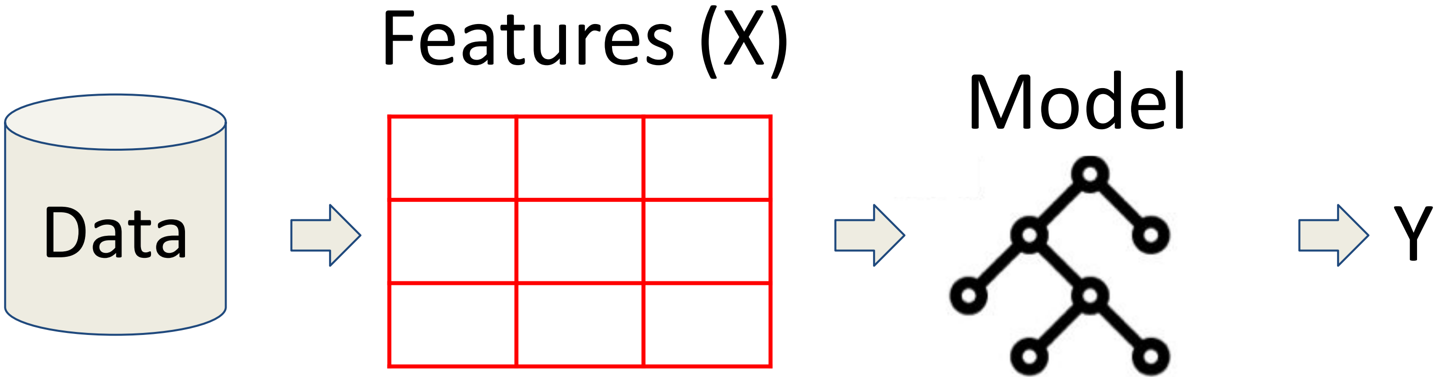


Normalized:



- Center the values around zero.
- Scale the values to fall between a fixed range.

Enemies of Machine Learning



Features (X)

Data

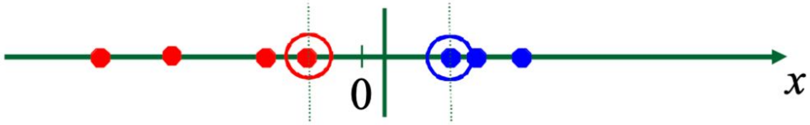
Model

Y

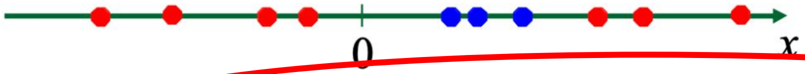
- Some contributing factors:
  - Features have different scales.
  - **Too many features**

Recall: **Kernel functions** in non-linear SVMs.

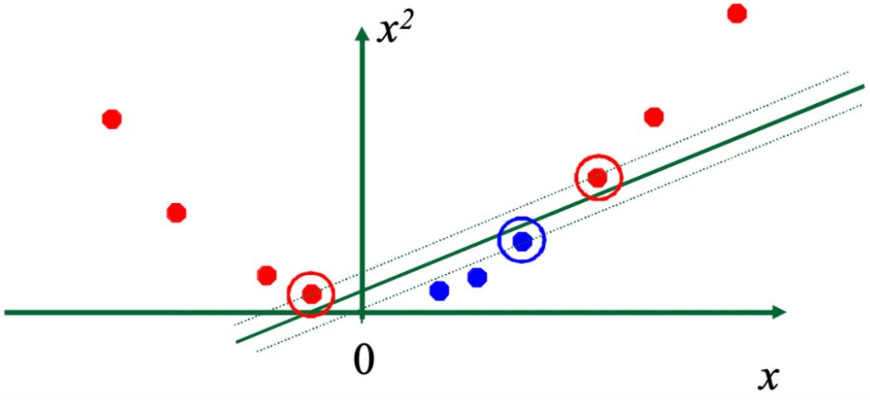
- Datasets that are linearly separable with some noise work out great:



- But what are we going to do if the dataset is just too hard?



- How about… mapping data to a higher-dimensional space:



Increasing feature dimensionality

Features (X)

x1	x2	x3

x1	x2	x3	x4	x5	x6	x7	x8

- Number of rows are fixed.
- Number of column of X increase.

Downsides?

- Hard to visualize.
- Increased computational time.

Increasing feature dimensionality

Features (X)

x1	x2	x3

→

x1	x2	x3	x4	x5	x6	x7	x8

- Number of rows are fixed.
- Number of column of X increase.

Downsides?

- **Hard to visualize.**
- Increased computational time.

Plotting a dataset for easy visualization

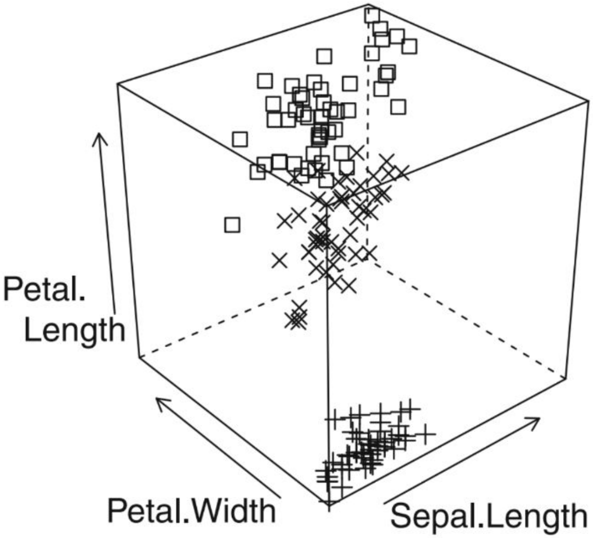
Iris dataset-

- 4 features
  - Sepal length/width
  - Petal length/width
- 3 classes

Plotting a dataset for easy visualization

Iris dataset-

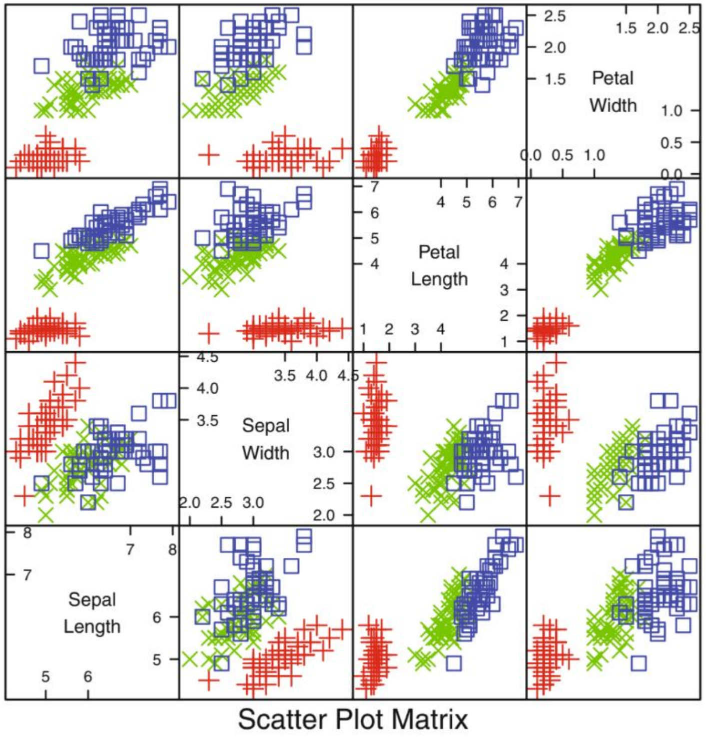
- 4 features
  - Sepal length/width
  - Petal length/width
- 3 classes



Plotting a dataset for easy visualization

Iris dataset-

- 4 features
  - Sepal length/width
  - Petal length/width
- 3 classes

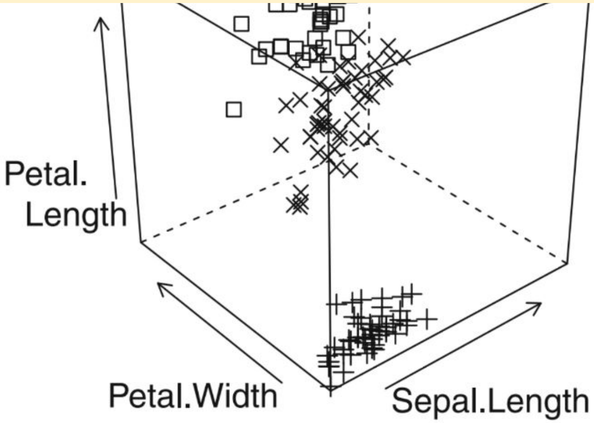


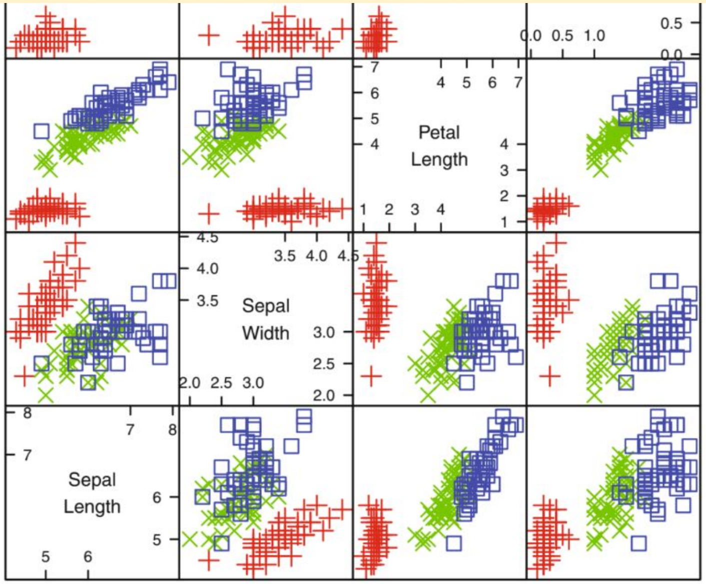
Plotting a dataset for easy visualization

Iris dataset-

- 4 features
  - Sepal length/width
  - Petal length/width
- 3 classes

Hard to get a more holistic view of the dataset





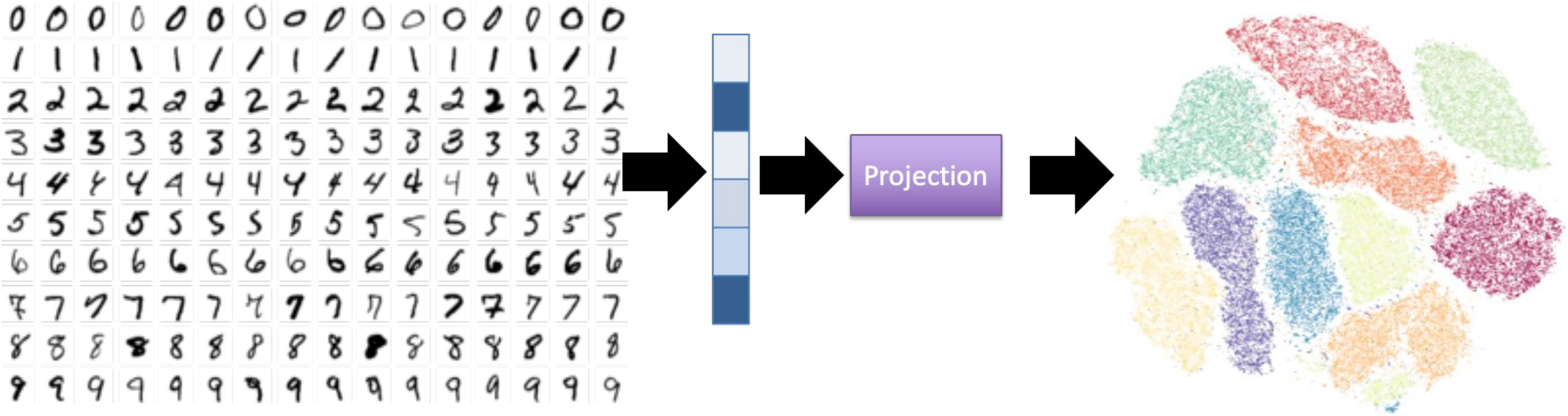
Scatter Plot Matrix

Plotting a dataset for easy visualization

MNIST Dataset- Digit Classification

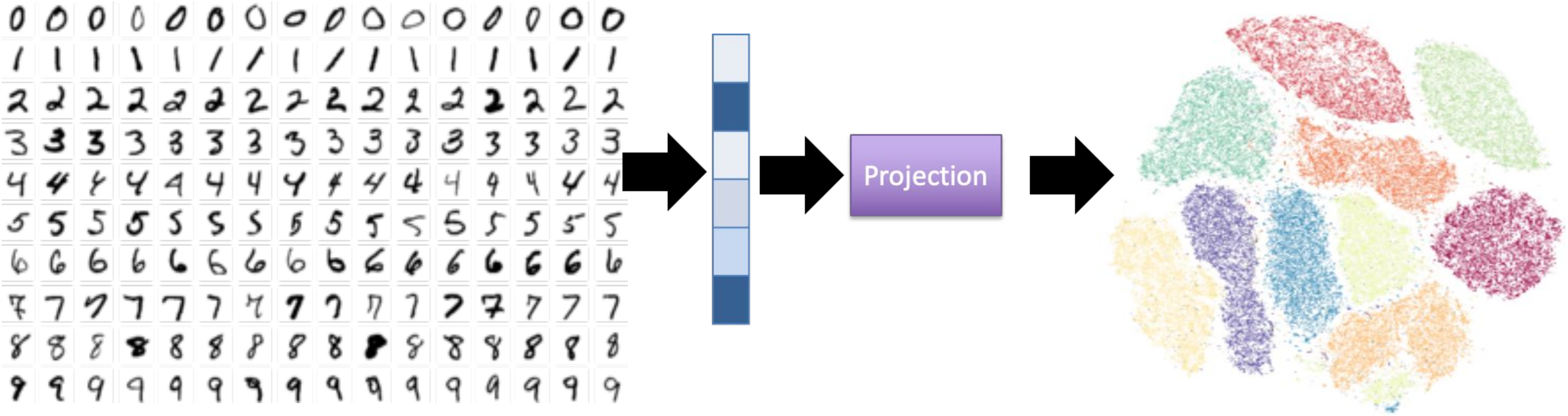
Plotting a dataset for easy visualization

MNIST Dataset- Digit Classification



Plotting a dataset for easy visualization

MNIST Dataset- Digit Classification



- **Project data to fewer dimensions.**
- Retain relative information about the entire feature space.

Why project to fewer dimensions?

Features (X)

x1	x2	x3

→

x1	x2	x3	x4	x5	x6	x7	x8

- Number of rows are fixed.
- Number of column of X increase.

Downsides?

✅ Hard to visualize: project to fewer dimensions.

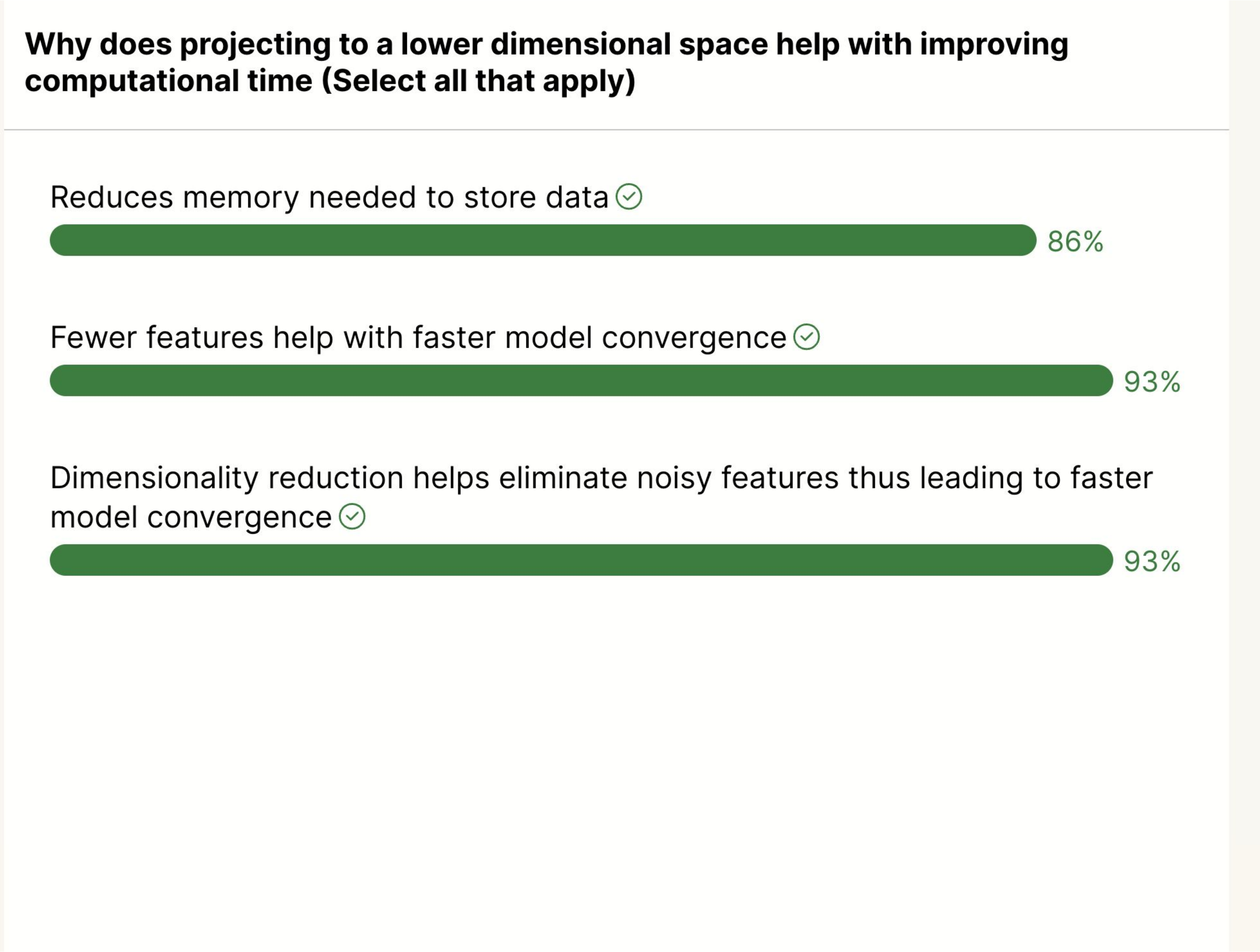
- Increased computational time.

Do not edit
How to change the design

Why does projecting to a lower dimensional space help with improving computational time (Select all that apply)

Presenting with animations, GIFs or speaker notes? Enable our Chrome extension

slido



Why does projecting to a lower dimensional space help with improving computational time (Select all that apply)

Reduces memory needed to store data ✓

86%

Fewer features help with faster model convergence ✓

93%

Dimensionality reduction helps eliminate noisy features thus leading to faster model convergence ✓

93%

Why project to fewer dimensions?

Features (X)

x1	x2	x3

→

x1	x2	x3	x4	x5	x6	x7	x8

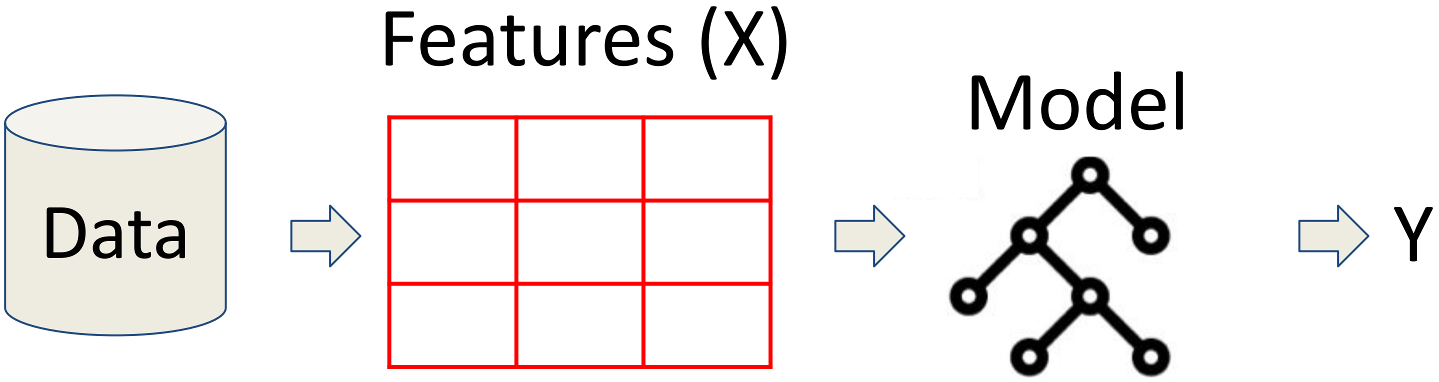
- Number of rows are fixed.
- Number of column of X increase.

Downsides?

✅ Hard to visualize : project to fewer dimensions.

✅ Increased computational time: project to fewer dimensions.

How to find a good (low-dimensional) feature space?



Features (X)

Model

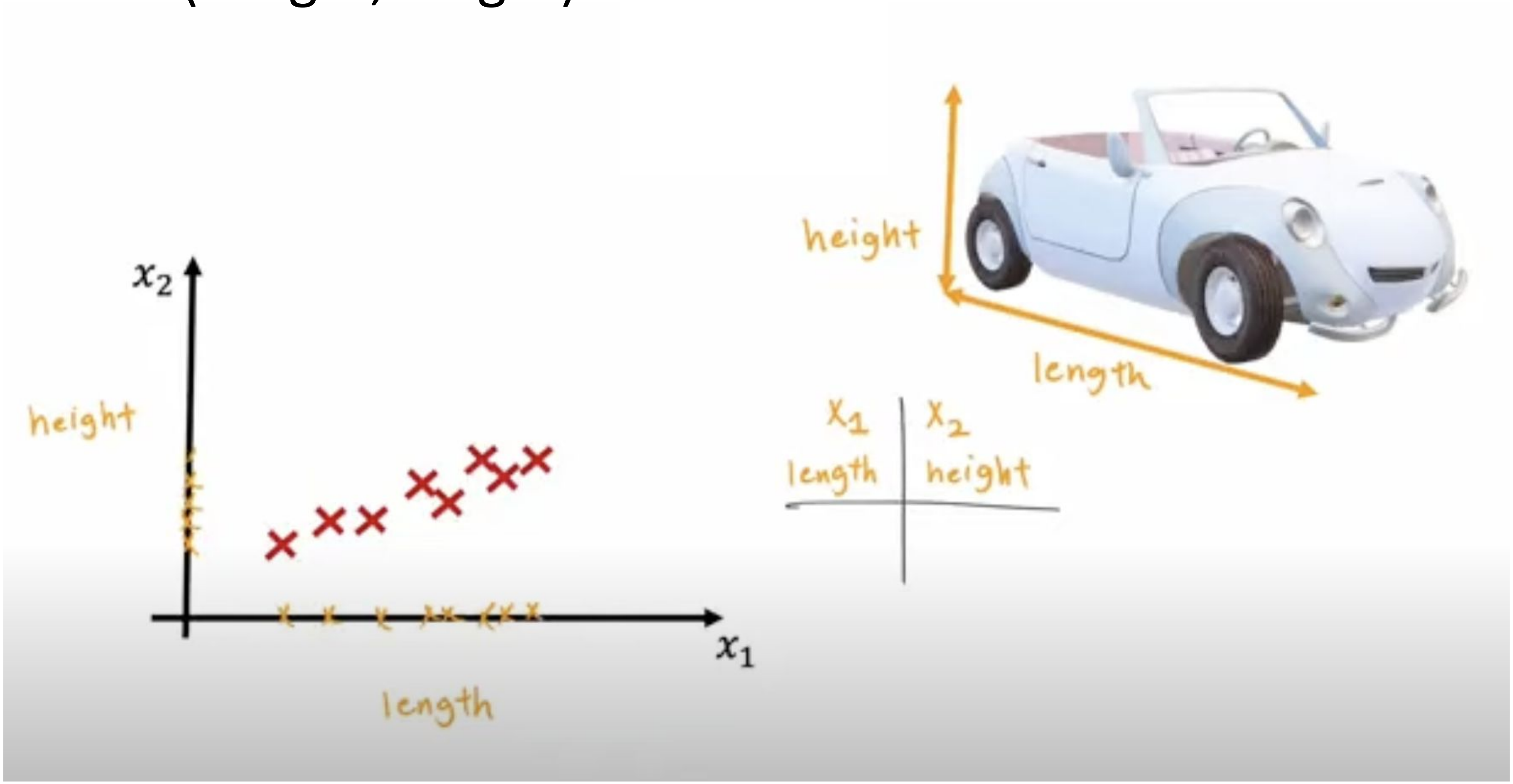
Data

Y

- Packs as much useful information about the target function as possible.
- As less correlated with the other features as possible.

Toy example: Car dataset

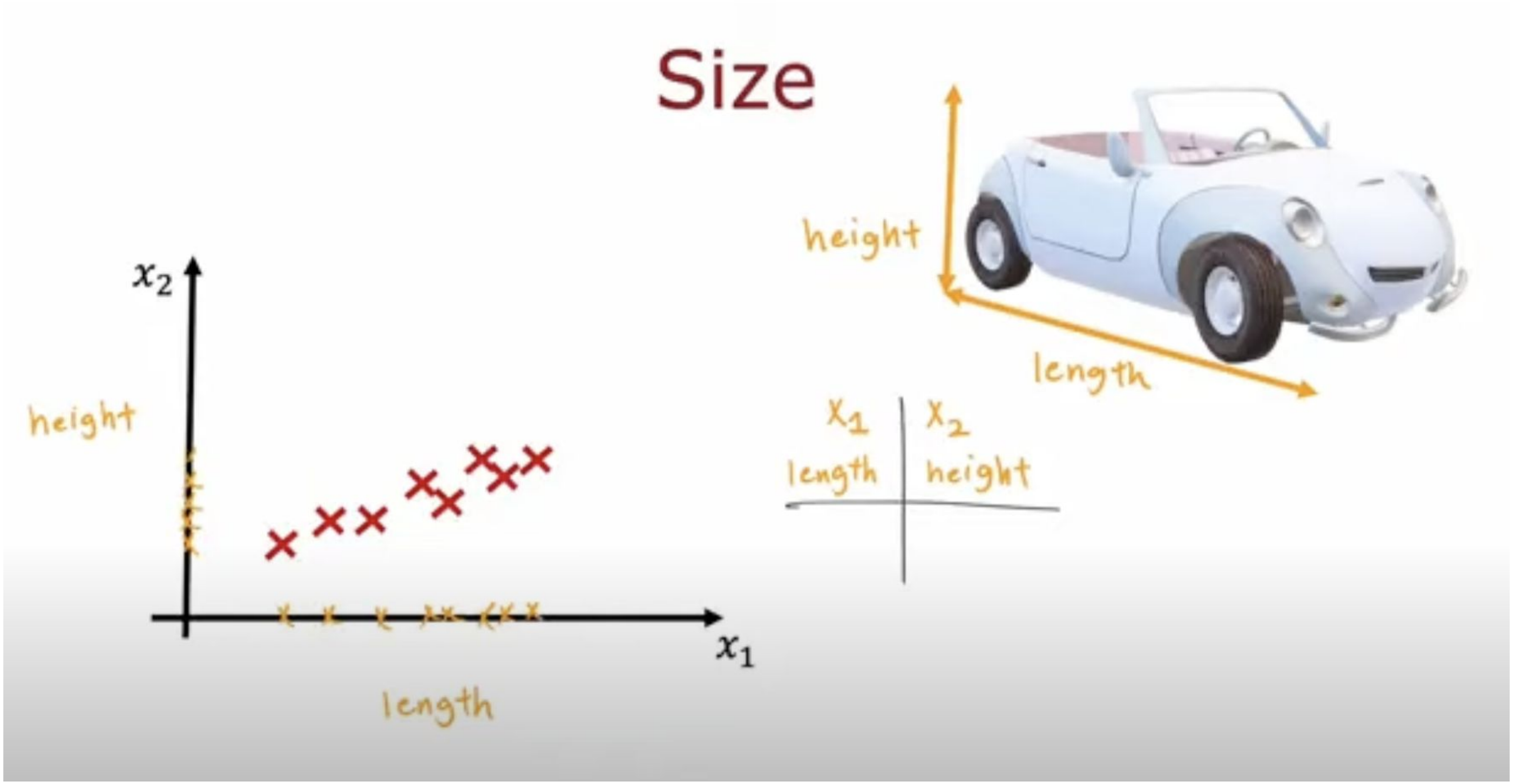
- (Length, height)



X_1	X_2
length	height

Slide credit: Andrew Ng

What if we define a new feature: car size

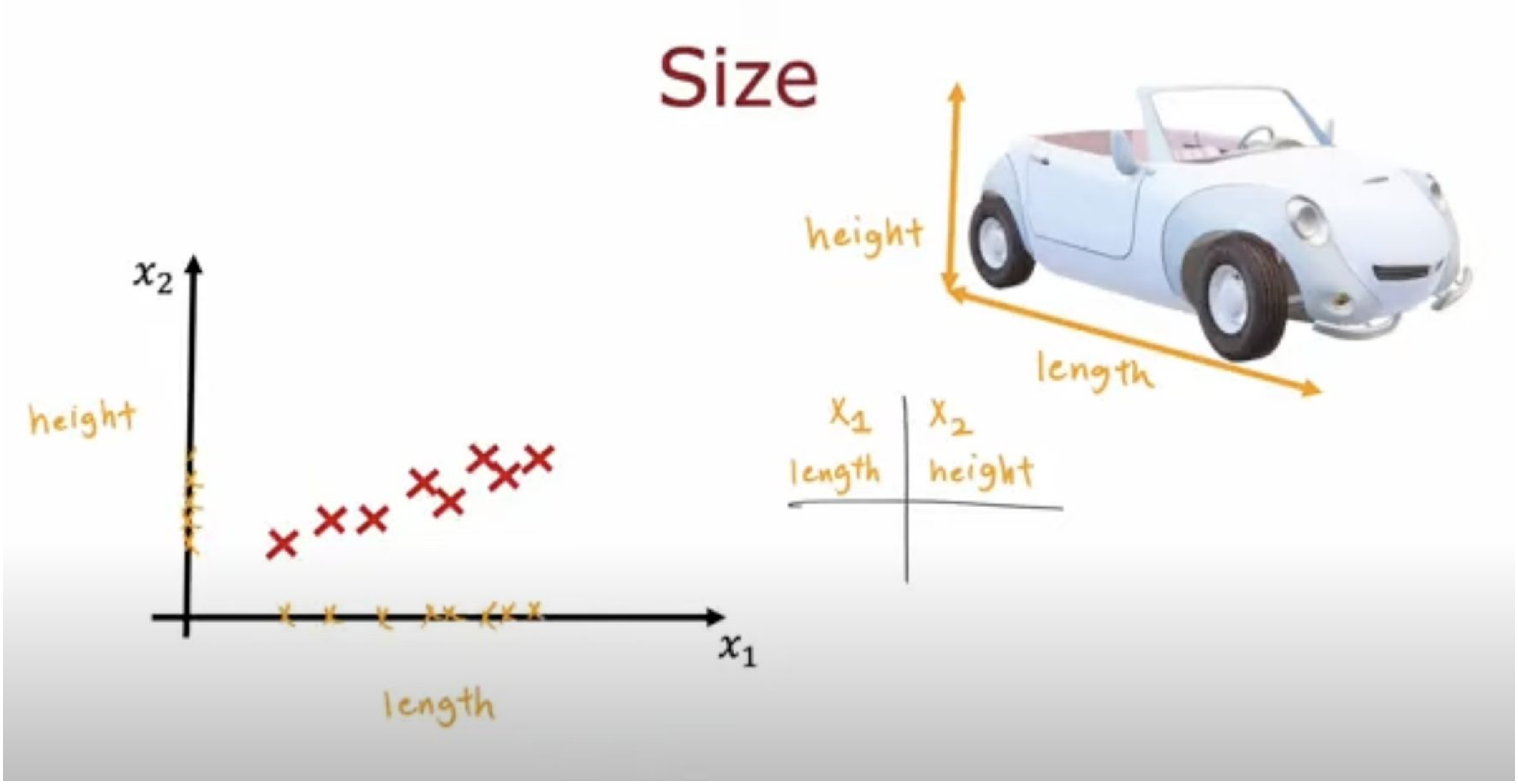


x_1	x_2
length	height

A third feature: that captures the car size

Slide credit: Andrew Ng

What if we define a new feature: car size

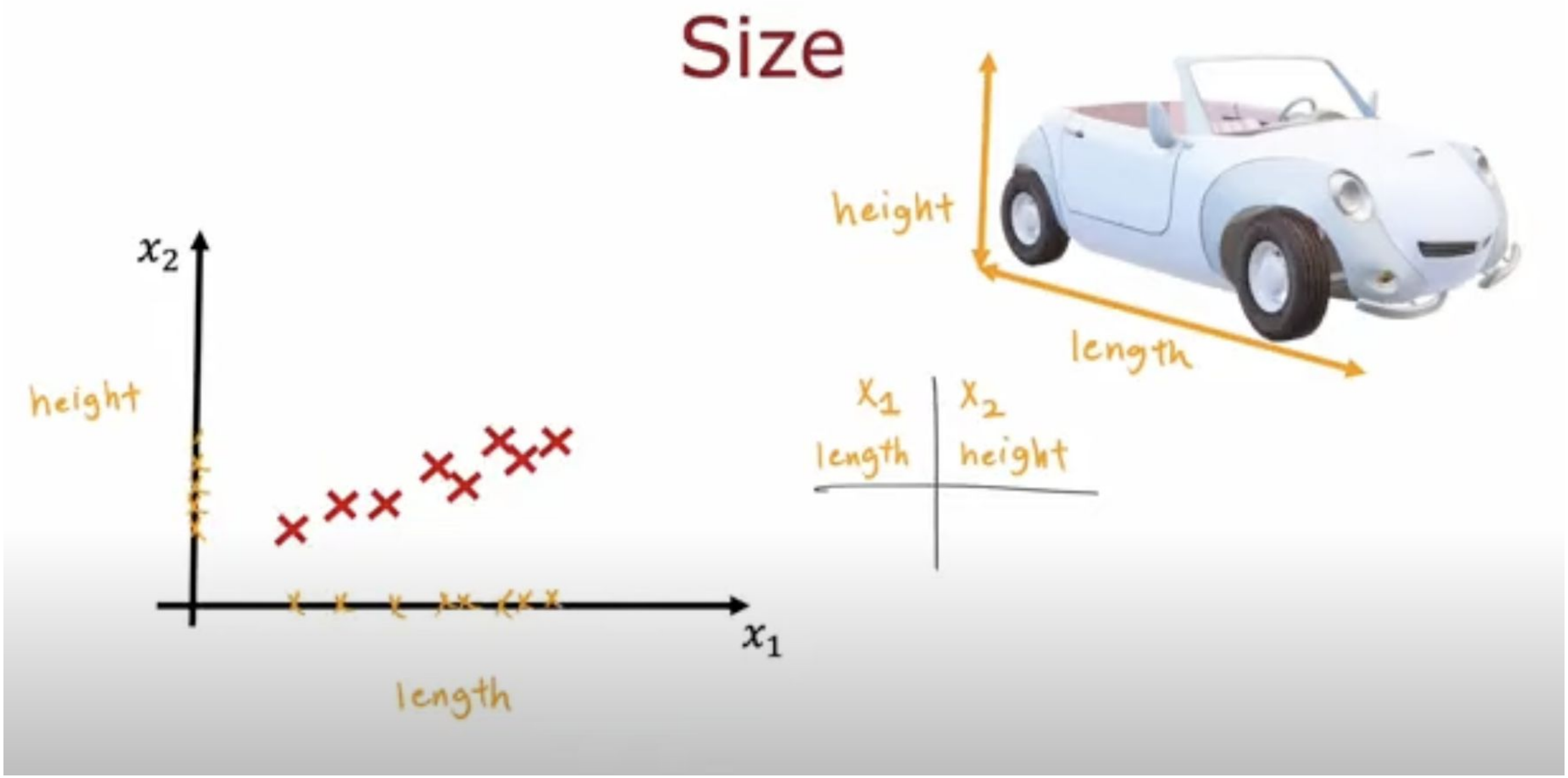


x_1	x_2
length	height

Goal: instead of (**length**, **height**) ; use only **size**

Slide credit: Andrew Ng

What if we define a new feature: car size



X_1	X_2
length	height

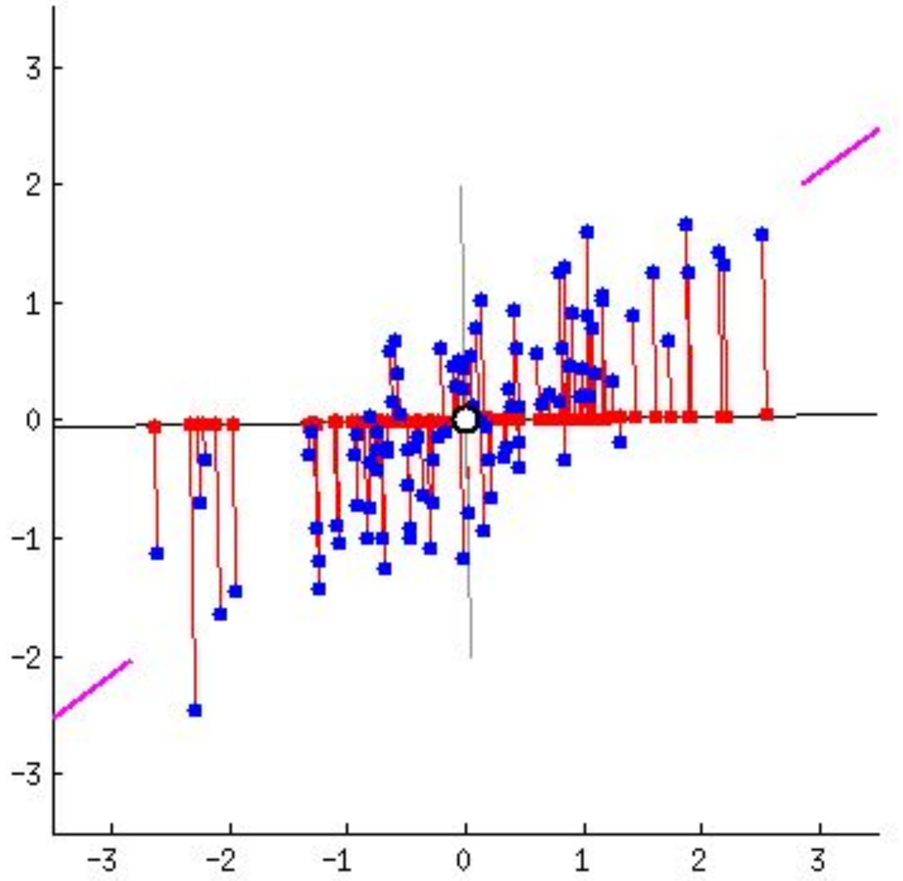
- **Goal:** instead of (**length**, **height**) ; use only **size**
- Using “size” feature does not lead to information loss compared to using (length, height)

Slide credit: Andrew Ng

Goal: Pick a direction that leads to minimal information loss

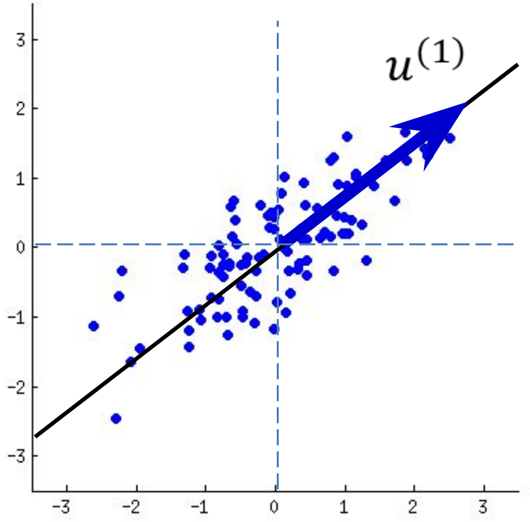
Normalize your data

Choose subspace with minimal "information loss"



Choose subspace with minimal "information loss"

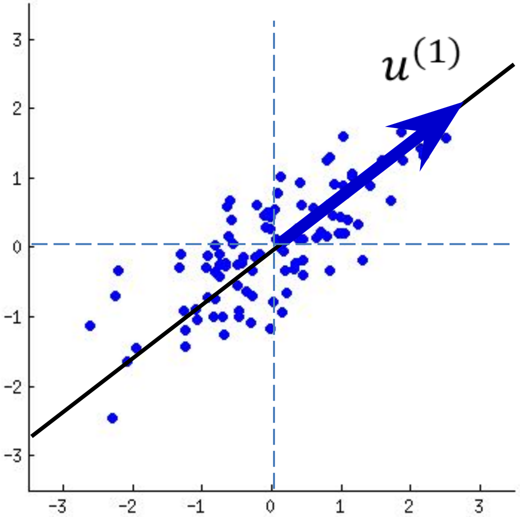
From 2D -> 1D



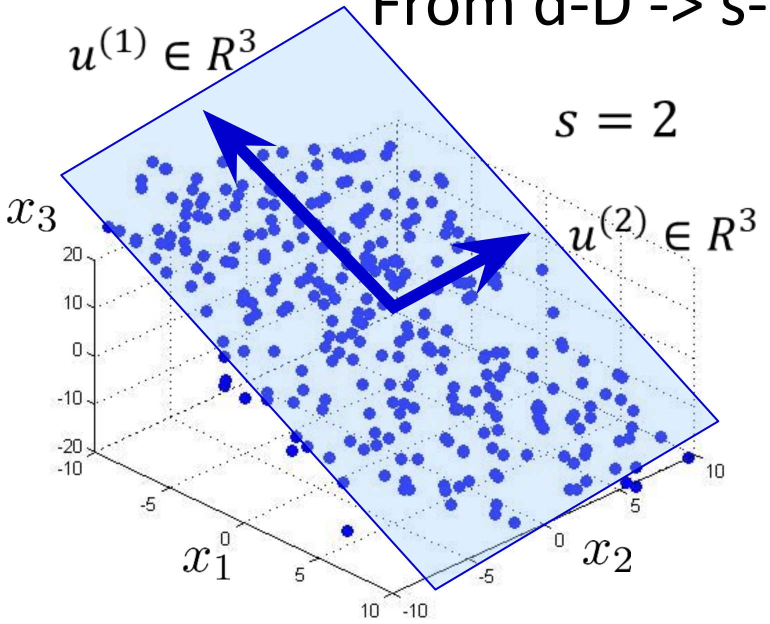
Find a direction (a vector $u^{(1)}$ onto which to project to minimize the projection error.

Choose subspace with minimal “information loss”

From 2D -> 1D



From d-D -> s-D



Find a direction (a vector $u^{(1)}$ onto which to project to minimize the projection error.

Find **s** vectors ($u^{(1)}$, $u^{(2)}$, $u^{(3)}$,..., $u^{(s)}$) onto which to project the data so as to minimize the projection error

Goal: Pick a direction that leads to minimal information loss

Normalize your data

Covariance matrix

Covariance

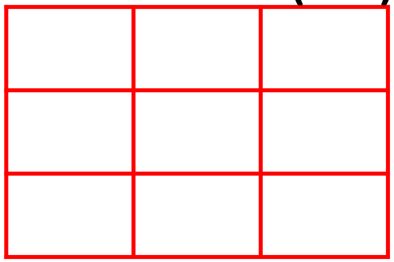
General idea: How similar two features are:

Formally: For two sets of N data items $\{x\}, \{y\}$**,**

$$cov(\{x\},\{y\}) = \frac{\sum_i \big(x_i - \mathrm{mean}(\{x\})\big)\big(y_i - \mathrm{mean}(\{y\})\big)}{N}$$

Covariance Matrix

Features (X^D)



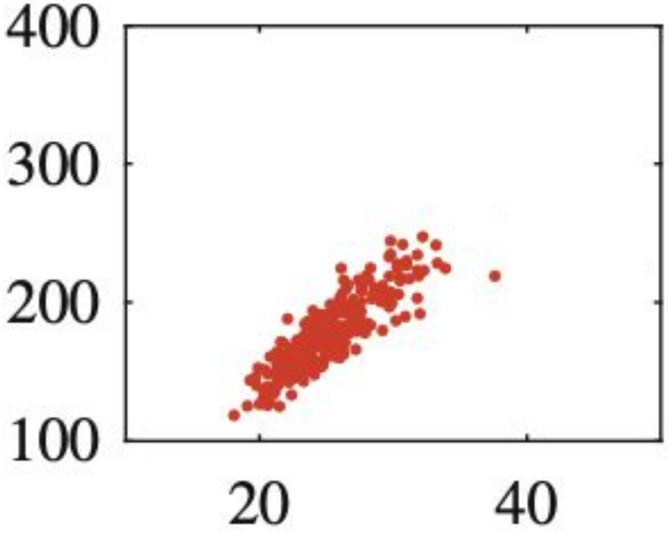
Square matrix of size **D** X **D**

$$Covmat(\{x\}) = \frac{\sum_i \left(x_i - \text{mean}(\{x\})\right)\left(x_i - \text{mean}(\{x\})\right)}{N}$$

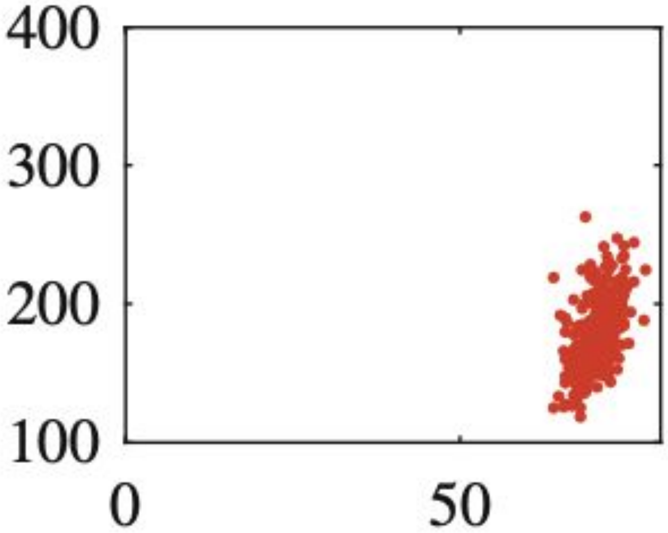
Some properties (more in book):

- Symmetric: $Covmat(\{x\}) = Covmat(\{x\})^T$
- Diagonal captures the variance of a given vector

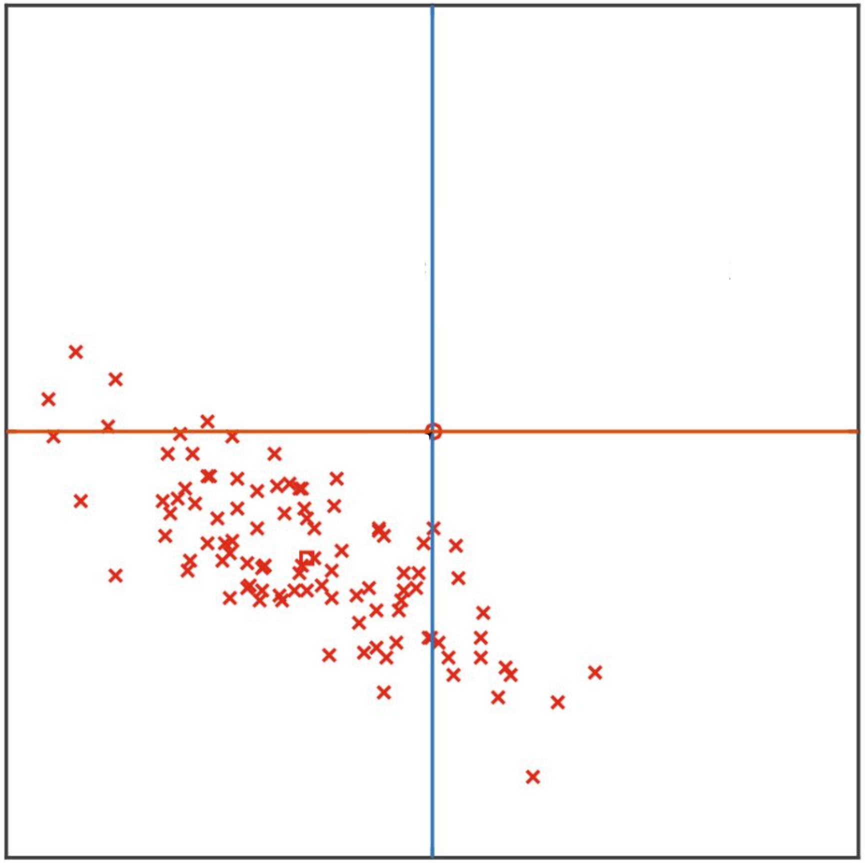
What is captured in these covariance matrices?



What is captured in these covariance matrices?

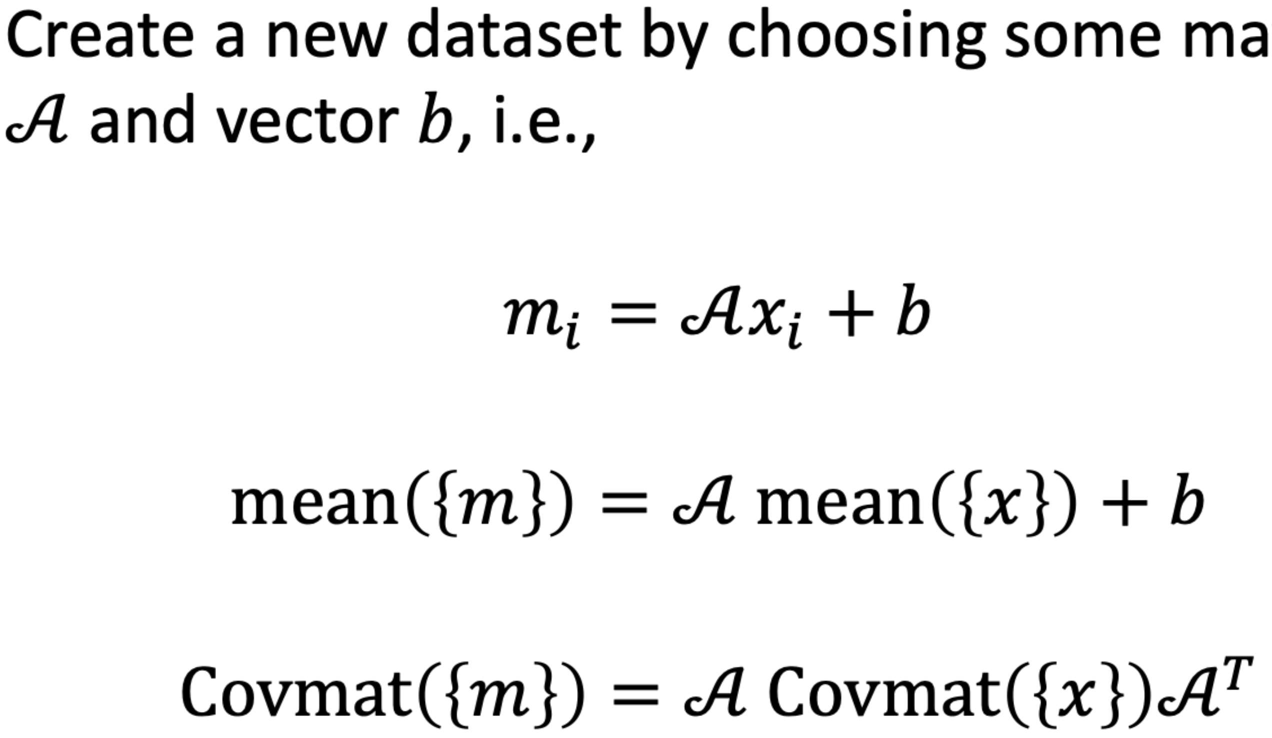


Center the data using mean and covariance



Affine Transformation

- Centering the data → better mean and covariances.



Create a new dataset by choosing some matrix $\mathcal{A}$ and vector b, i.e.,

$$m_i = \mathcal{A}x_i + b$$

$$\text{mean}(\{m\}) = \mathcal{A}\ \text{mean}(\{x\}) + b$$

$$\text{Covmat}(\{m\}) = \mathcal{A}\ \text{Covmat}(\{x\})\mathcal{A}^T$$

Proof in the textbook

Goal: Pick a set of directions that leads to minimal information loss

Normalize your data

Covariance matrix

Eigen vectors

Assume $\mathcal{S}$ is a $d \times d$ symmetric matrix $\mathbf{u}$ is a $d \times 1$ vector and λ is a scalar, then if

$$\mathcal{S}\mathbf{u} = \lambda\mathbf{u}$$

$\mathbf{u}$ is an eigenvector of $\mathcal{S}$ and λ is the **eigenvalue**

- **Eigenvectors capture the directions of greatest variance within a dataset.**

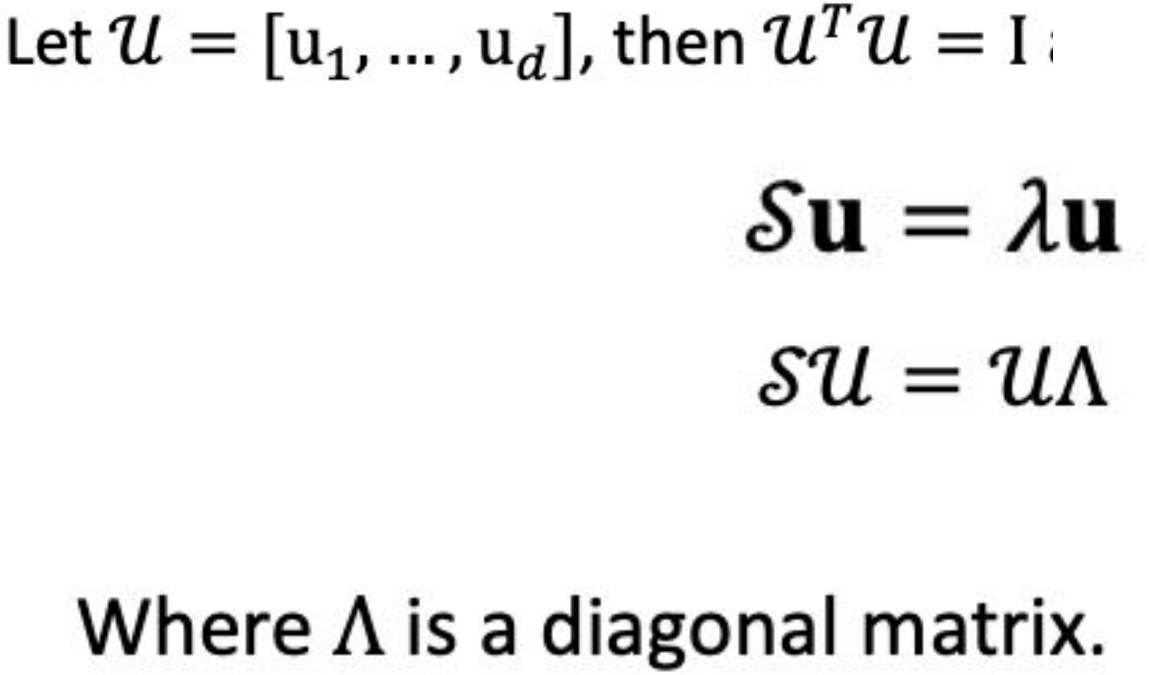
Constraints on Eigenvectors and values

$$\mathcal{S}\mathbf{u} = \lambda\mathbf{u}$$

- Eigenvalues (**λ**) are real numbers.
- Eigen vectors are **d distinct vectors.**
  - They are normal to one another.
  - Can be scaled to have unit length.

Let $\mathcal{U} = [\mathrm{u}_1, \ldots, \mathrm{u}_d]$, then $\mathcal{U}^T\mathcal{U} = \mathrm{I}$

Eigenvalues and vectors



Let $\mathcal{U} = [\mathrm{u}_1, \ldots, \mathrm{u}_d]$, then $\mathcal{U}^T\mathcal{U} = \mathrm{I}$

$$\mathcal{S}\mathbf{u} = \lambda\mathbf{u}$$

$$\mathcal{S}\mathcal{U} = \mathcal{U}\Lambda$$

Where Λ is a diagonal matrix.

- How to determine eigenvalues and vectors?
  - Use **singular value decomposition (SVD)**.

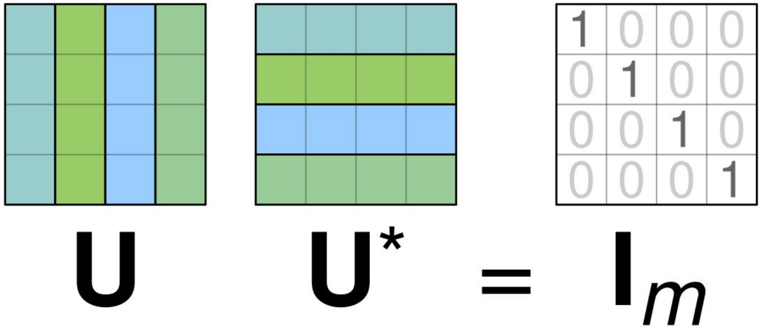
SVD Definition

For any $m{\times}p$ matrix $\mathcal{X}$, you can decompose it into:

$$\mathcal{X} = \mathcal{U}\Sigma\mathcal{V}^T$$

where $\mathcal{U}$ is $m{\times}m$, $\mathcal{V}$ is $p{\times}p$, and Σ is $m{\times}p$.

- Σ is diagonal with non-negative entries
- $\mathcal{U}$ and $\mathcal{V}$ are orthonormal



SVD Definition

For any $m \times p$ matrix $\mathcal{X}$, you can decompose it into:

$$\mathcal{X} = \mathcal{U}\Sigma\mathcal{V}^T$$

where $\mathcal{U}$ is $m \times m$, $\mathcal{V}$ is $p \times p$, and Σ is $m \times p$.

- Σ is diagonal with non-negative entries
- $\mathcal{U}$ and $\mathcal{V}$ are orthonormal

Eigen values

SVD Definition

For any $m\times p$ matrix $\mathcal{X}$, you can decompose it into:

$$\mathcal{X} = \mathcal{U}\Sigma\mathcal{V}^T$$

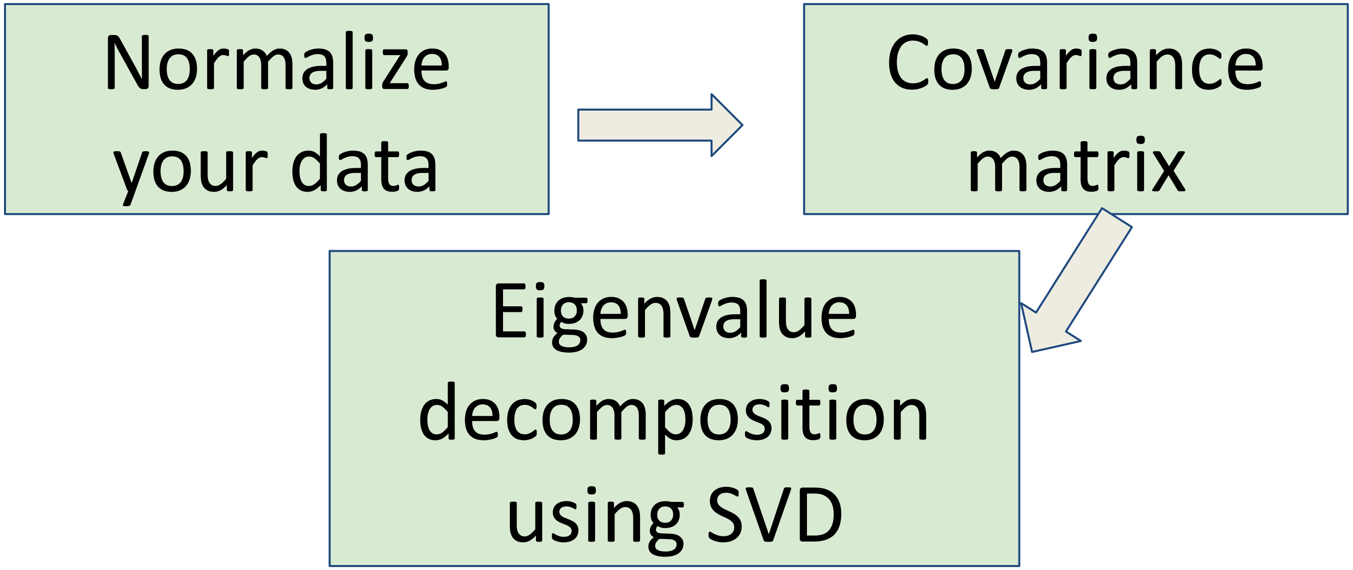
where $\mathcal{U}$ is $m\times m$, $\mathcal{V}$ is $p\times p$, and Σ is $m\times p$.

- Σ is diagonal with non-negative entries
- $\mathcal{U}$ and $\mathcal{V}$ are orthonormal

Eigen vectors

Building blocks of Principal component analysis (PCA)

Goal: Pick a set of directions that leads to minimal information loss



Normalize your data

Covariance matrix

Eigenvalue decomposition using SVD

Principal component analysis (PCA)

Normalize features (ensure every feature has zero mean) and optionally scale feature

Compute “covariance matrix” Σ:

$$\Sigma = \frac{1}{N}\sum_{i=1}^{N}(x^{(i)})(x^{i})^{T}$$

Compute its “eigenvectors”:

$$[\boldsymbol{U},\boldsymbol{S},\boldsymbol{V}] = \mathbf{svd}(\boldsymbol{\Sigma});$$

$$\mathcal{U} = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(n)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{d\times d}$$

Principal component analysis (PCA)

Normalize features (ensure every feature has zero mean) and optionally scale feature

Compute “covariance matrix” Σ:

$$\Sigma = \frac{1}{N}\sum_{i=1}^{N}(x^{(i)})(x^{i})^{T}$$

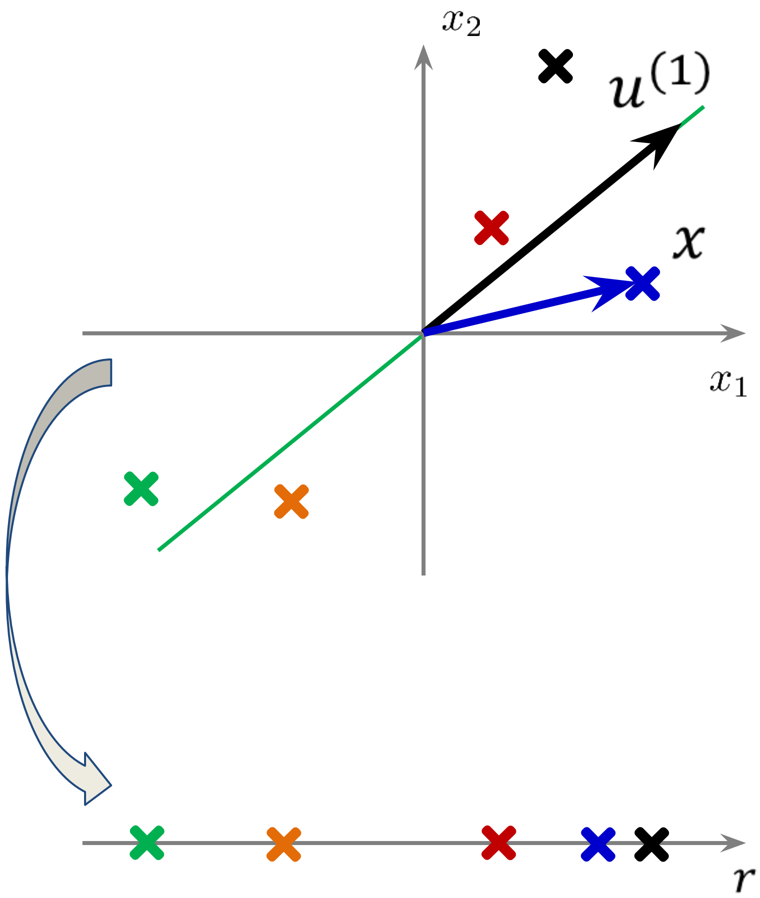
Compute its “eigenvectors”:

$[\boldsymbol{U},\boldsymbol{S},\boldsymbol{V}] = \mathbf{svd}(\boldsymbol{\Sigma});$

$$\mathcal{U} = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(n)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{d\times d}$$

Keep first s eigenvectors and project to get new features r

Principal Components Analysis

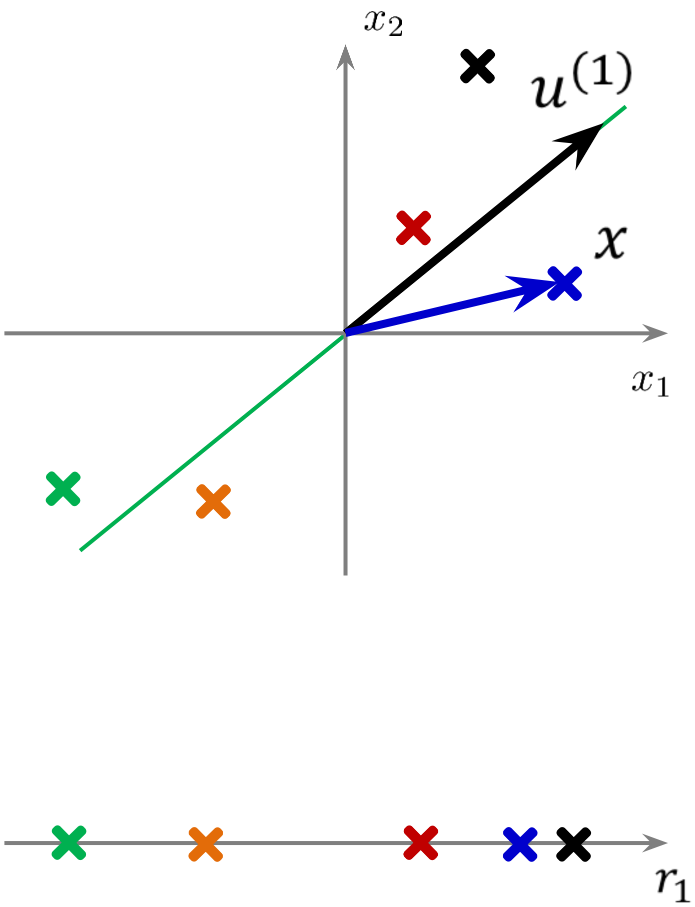


Find orthonormal basis vectors

$\mathcal{U}^* = [u^{(1)} \quad \ldots \quad u^{(s)}]$, where s ≪ d

$r_i = \mathcal{U}^{*T}(x_i - \text{mean}(\{x\}))$

Principal Components Analysis



Find orthonormal basis vectors

$$\mathcal{U}^* = [u^{(1)} \quad \ldots \quad u^{(s)}], \text{ where s} \ll \text{d}$$

$$r_i = \mathcal{U}^{*T}(x_i - \mathrm{mean}(\{x\}))$$

Reconstructed data point

$$\tilde{x} = \sum_{i=1}^{s} r_i u^{(s)}$$

Cost function:

$$J = \frac{1}{N}\sum_{i=1}^{N} \left\| \tilde{x}^i - x^i \right\|^2$$

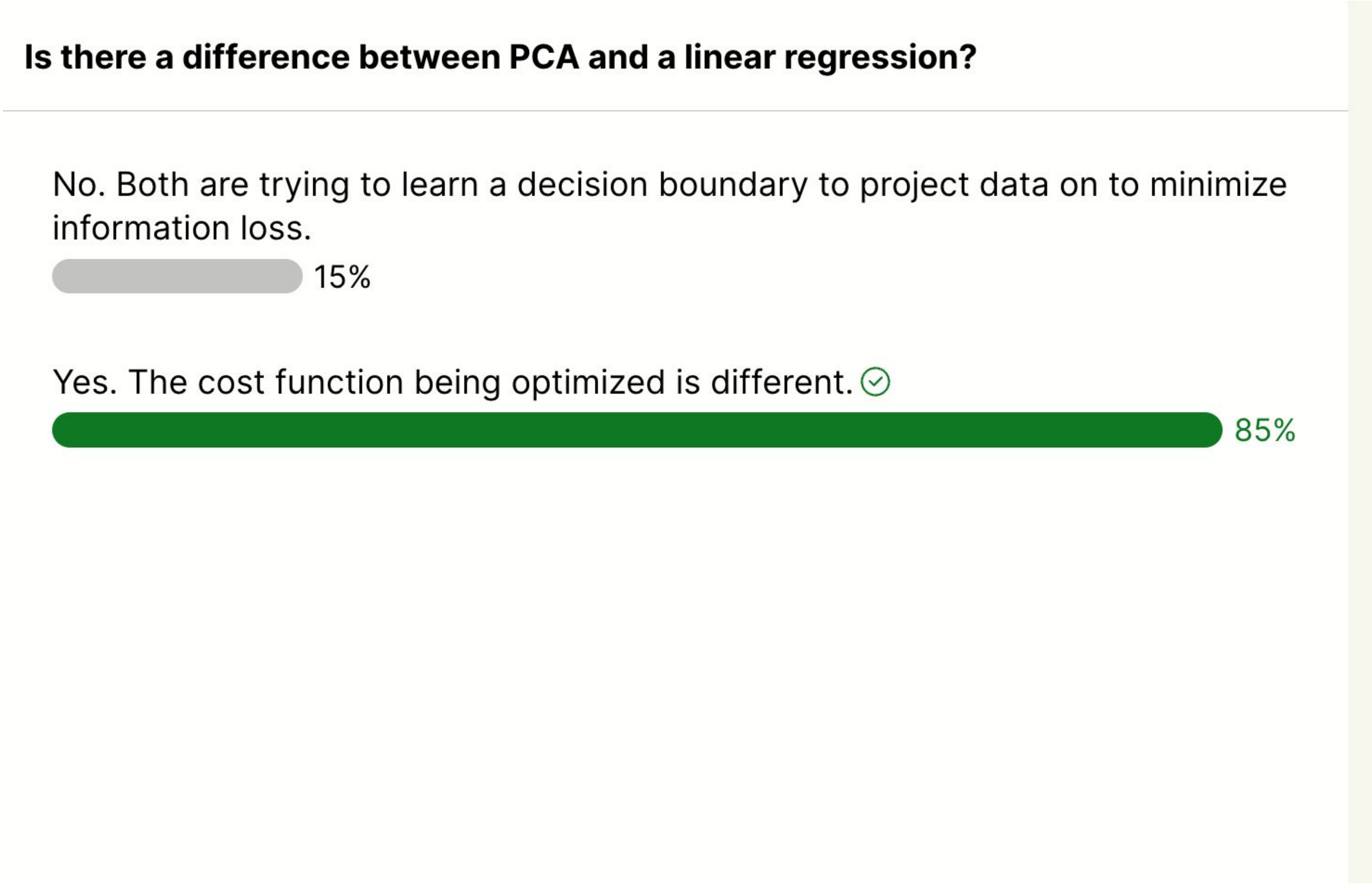
Want: $\min_{U} J$

Do not edit
How to change the design

Is there a difference between PCA and a linear regression?

ⓘ Presenting with animations, GIFs or speaker notes? Enable our Chrome extension

slido



Is there a difference between PCA and a linear regression?

No. Both are trying to learn a decision boundary to project data on to minimize information loss.

15%

Yes. The cost function being optimized is different. ✓

85%

Choosing *s* (number of principal components)

Average squared projection error:

Total variation in the data:

Typically, choose *s* to be smallest value so that

$$\boxed{\frac{\frac{1}{N}\sum_{i=1}^{N}\left\|\tilde{x}^i - x^i\right\|^2}{\frac{1}{N}\sum_{i=1}^{N}\|x_i\|^2} \le 0.01 \text{ (1\%)}}$$

“99% of variance is retained”

Choosing *s* (number of principal components)

$$[\boldsymbol{U}, \boldsymbol{S}, \boldsymbol{V}] = \mathbf{svd}(\boldsymbol{\Sigma})$$

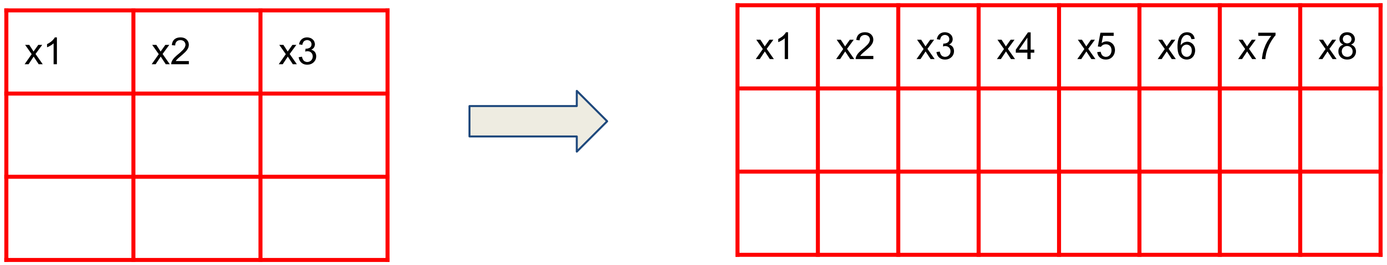
Pick smallest value of *s* for which

$$\boxed{\frac{\sum_{i=1}^{s} S_{ii}}{\sum_{i=1}^{N} S_{ii}} \geq 0.99}$$

(99% of variance retained in the eigen values)

Why project to fewer dimensions?

Features (X)



x1	x2	x3

x1	x2	x3	x4	x5	x6	x7	x8

- Number of rows are fixed.
- Number of column of X increase.

Downsides?

✅ Hard to visualize : project to fewer dimensions.

✅ Increased computational time: project to fewer dimensions.

Do not edit

How to change the design

What is the guaranteed solution to reduce overfitting? (select all that apply)

Next Class

CCA: Multidimensional scaling, Word Embeddings, TF-IDF, Canonical Correlation Analysis

Reading: Forsyth Ch 6.2-6.3, 7.1