EE2211 Introduction to Machine Learning

Lecture 7

Juan Helen Zhou
helen.zhou@nus.edu.sg

Electrical and Computer Engineering Department
National University of Singapore

Acknowledgement: EE2211 development team
Thomas, Helen, Xinchao, Kar-Ann, Chen Khong, Robby and Haizhou

Course Contents

- Introduction and Preliminaries (Xinchao)
  - Introduction
  - Data Engineering
  - Introduction to Probability and Statistics
- Fundamental Machine Learning Algorithms I (Helen)
  - Systems of linear equations
  - Least squares, Linear regression
  - Ridge regression, Polynomial regression
- Fundamental Machine Learning Algorithms II (Helen)
  - Over-fitting, bias/variance trade-off
  - Optimization, Gradient descent
  - Decision Trees, Random Forest
- Performance and More Algorithms (Xinchao)
  - Performance Issues
  - K-means Clustering
  - Neural Networks

Fundamental ML Algorithms: Linear Regression

References for Lectures 7-9:

Main

- [Book1] Andriy Burkov, "**The Hundred-Page Machine Learning Book**", 2019. (**read first, buy later**: http://themlbook.com/wiki/doku.php)
  - Chapters 3, 4, 5, & 7.
- [Book2] Andreas C. Muller and Sarah Guido, "**Introduction to Machine Learning with Python**: A Guide for Data Scientists", O'Reilly Media, Inc., 2017

Supplementary

- [Book3] Jeff Leek, "**The Elements of Data Analytic Style**: A guide for people who want to analyze data", Lean Publishing, 2015.
- [Book4] Stephen Boyd and Lieven Vandenberghe, "**Introduction to Applied Linear Algebra**", Cambridge University Press, 2018 (available online) http://vmls-book.stanford.edu/
- [Ref 5] **Professor Vincent Tan's notes (chapters 7-9): (useful) https://vyftan.github.io/papers/ee2211book.pdf**

Fundamental ML Algorithms: Overfitting, Bias-Variance Tradeoff

Module III Contents

- Overfitting, underfitting & model complexity
- Feature selection & Regularization
- Bias-variance trade-off
- Loss function
- Optimization
- Gradient descent
- Decision trees
- Random forest

Regression Review

repeat

- Goal: Given feature(s) x, we want to predict target y
  - x can be 1-D or more than 1-D
  - y is 1-D
- Two types of input data
  - Training set $\{x_i, y_i\}$, from $i = 1, \cdots, N$
  - Test set $\{x_j, y_j\}$, from $j = 1, \cdots, M$
- Learning/Training
  - Training set used to estimate regression coefficients $\hat{w}$
- Prediction/Testing/Evaluation
  - Prediction performed on test set to evaluate performance

Regression Review: Linear Case

repeat

- x is 1D & y is 1-D
- Linear relationship between x & y
- Illustration (4 training samples):

$$\mathbf{X}_{train} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \end{pmatrix} \quad \mathbf{y}_{train} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$$

- Training/Learning (primal) on training set

$$\hat{\mathbf{w}} = (\mathbf{X}_{train}^T \mathbf{X}_{train})^{-1} \mathbf{X}_{train}^T \mathbf{y}_{train}$$

- Prediction/Testing/Evaluation on test set

$$\hat{\mathbf{y}}_{test} = \mathbf{X}_{test} \hat{\mathbf{w}}$$

Regression Review: Polynomial

- x is 1-D (or more than 1-D) & y is 1-D repeated
- Polynomial relationship between x & y
- Quadratic illustration (4 training samples, x is 1-D):

$$\mathbf{X}_{train} = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \end{pmatrix} \quad \mathbf{y}_{train} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$$

- Training/Learning (primal) on training set

$$\hat{\mathbf{w}} = (\mathbf{X}_{train}^T \mathbf{X}_{train})^{-1} \mathbf{X}_{train}^T \mathbf{y}_{train}$$

- Prediction/Testing/Evaluation on test set

$$\hat{\mathbf{y}}_{test} = \mathbf{X}_{test} \hat{\mathbf{w}}$$

Regression Review: Polynomial

- x is 1-D (or more than 1-D) & y is 1-D repeated
- Polynomial relationship between x & y
- Quadratic illustration (4 training samples, x is 1-D):

$$\mathbf{P}_{train} = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \end{pmatrix} \quad \mathbf{y}_{train} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$$

- Training/Learning (primal) on training set

$$\hat{\mathbf{w}} = (\mathbf{P}_{train}^T \mathbf{P}_{train})^{-1} \mathbf{P}_{train}^T \mathbf{y}_{train}$$

- Prediction/Testing/Evaluation on test set

$$\hat{\mathbf{y}}_{test} = \mathbf{P}_{test} \hat{\mathbf{w}}$$

Note on Training & Test Sets

- Linear is special case of polynomial => use "**P**" instead of "**X**" from now on
- Training/Learning (primal) on training set

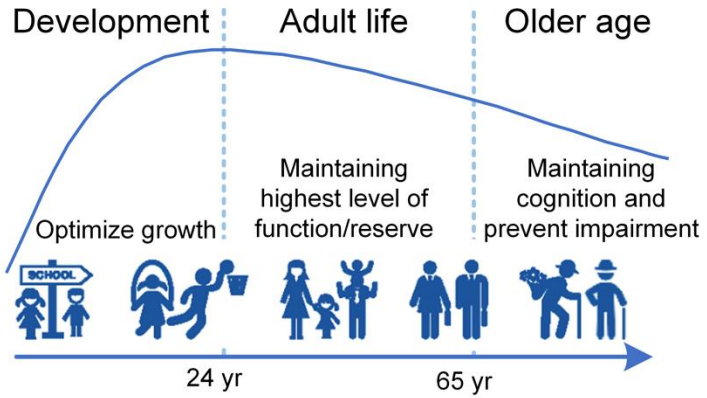
$$\hat{\mathbf{w}} = (\mathbf{P}_{train}^T \mathbf{P}_{train})^{-1} \mathbf{P}_{train}^T \mathbf{y}_{train}$$

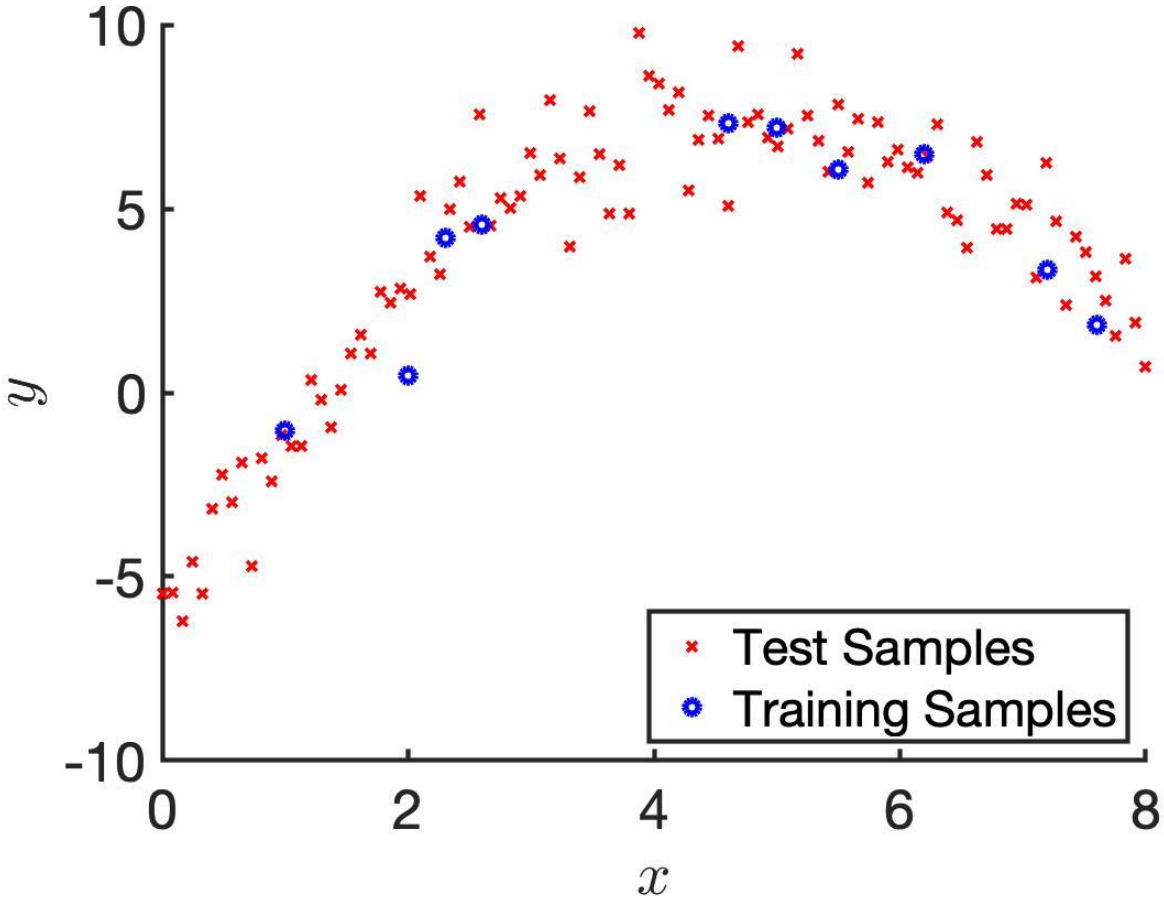
- Prediction/Testing/Evaluation on test set

$$\hat{\mathbf{y}}_{test} = \mathbf{P}_{test} \hat{\mathbf{w}}$$

- There should be **zero** overlap between training & test sets
- Important goal of regression: prediction on **new unseen data**, i.e., test set
- Why is test set important for evaluation?

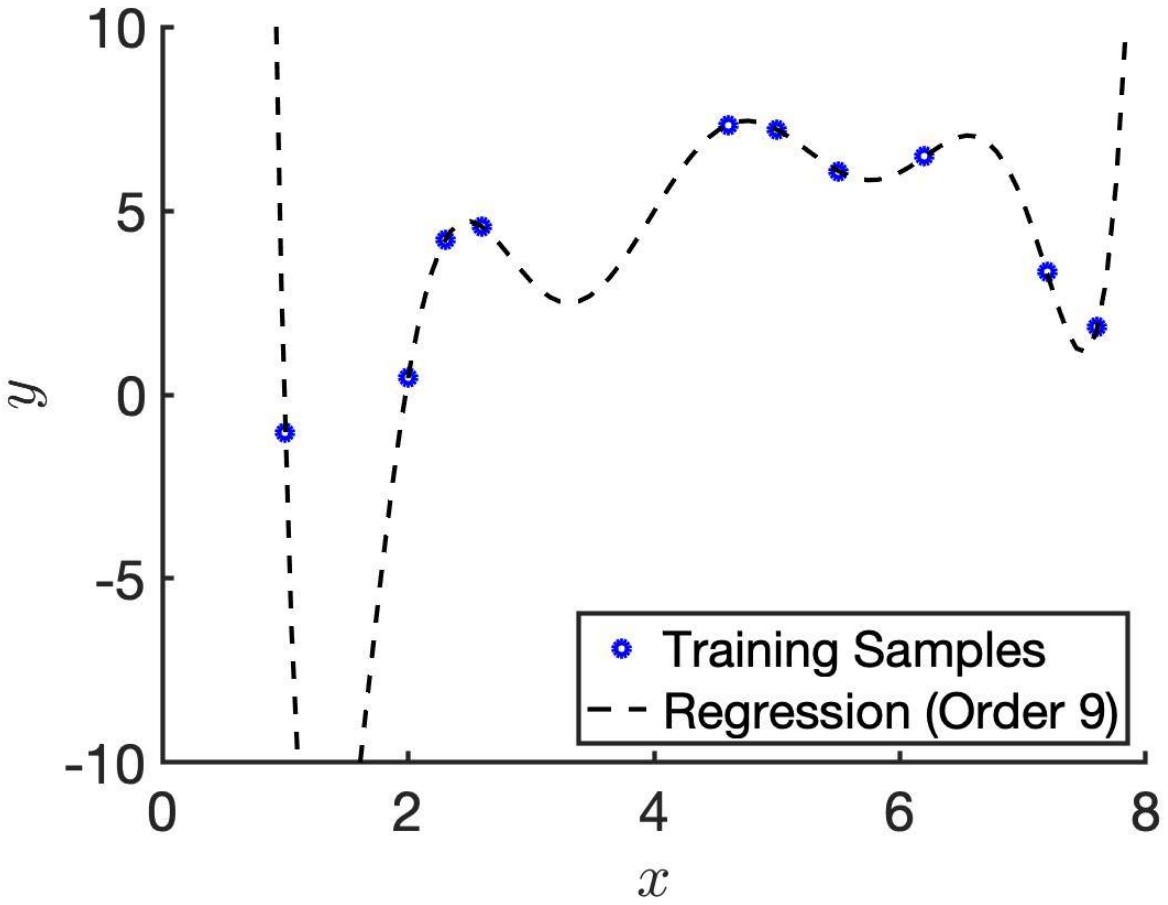
Overfitting Example





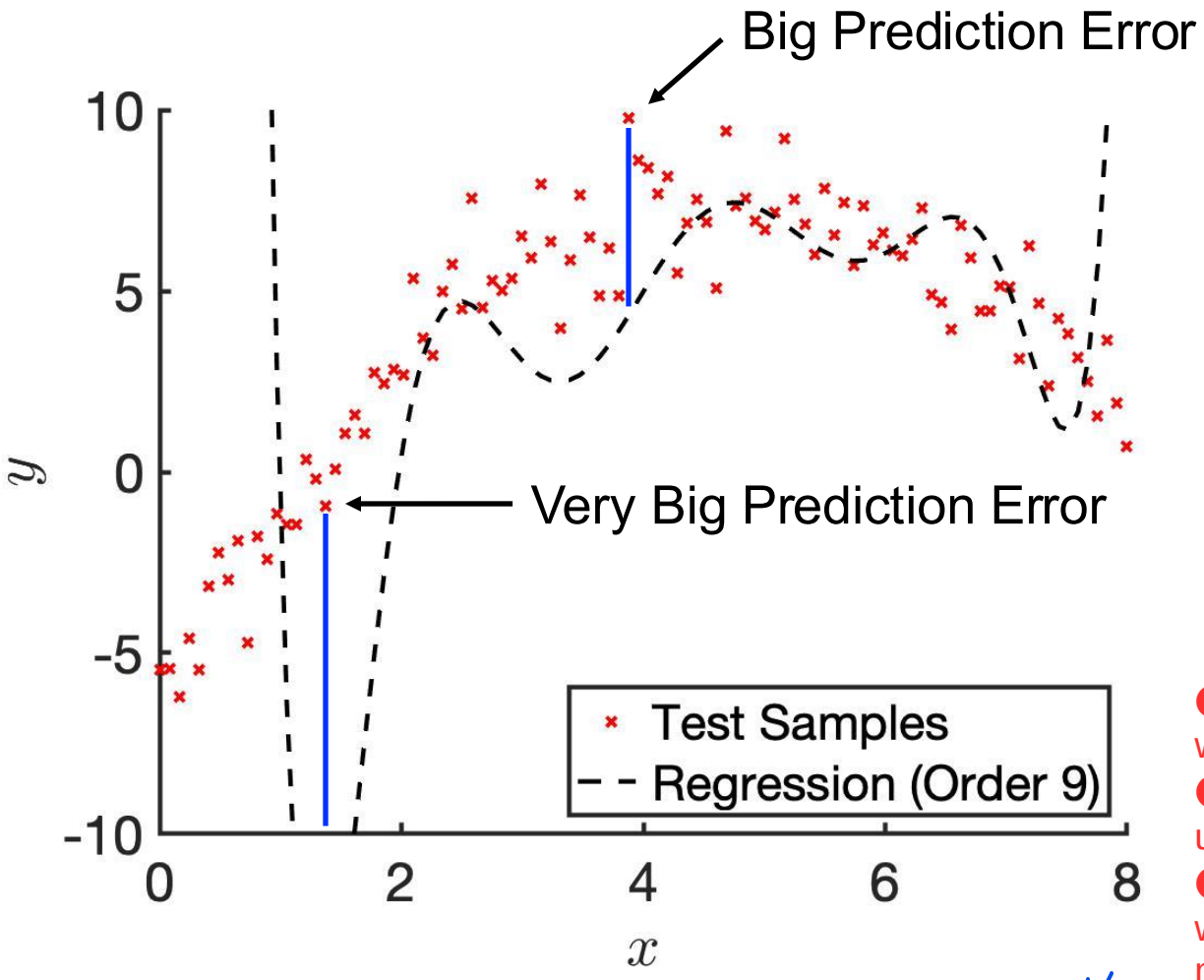
Overfitting Example

using order 9 regression has overfitted (as shown with previous and next slide



	Training Set Fit	
Order 9	Good	

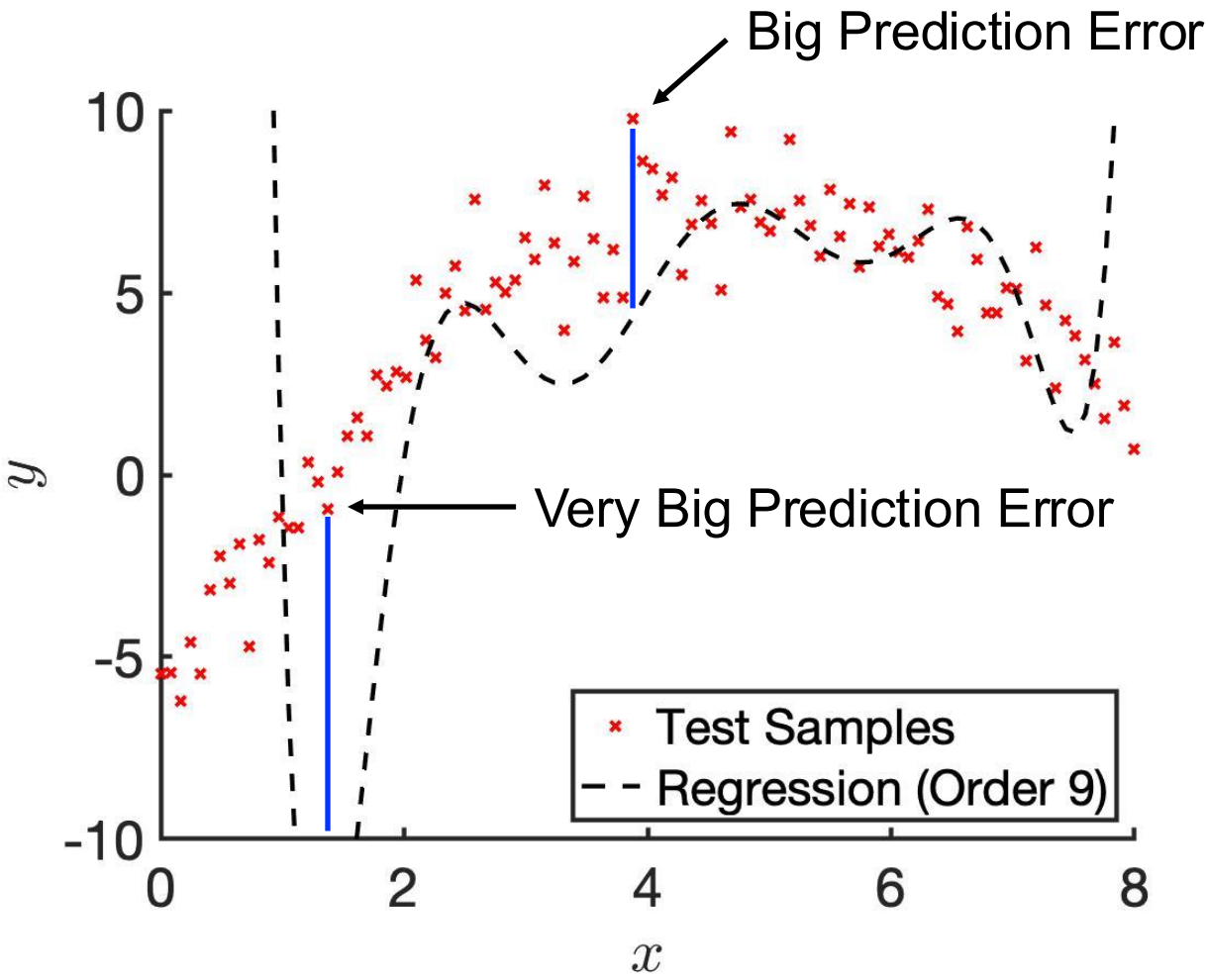
Overfitting Example



	Training Set Fit	Test Set Fit
Order 9	Good	Bad

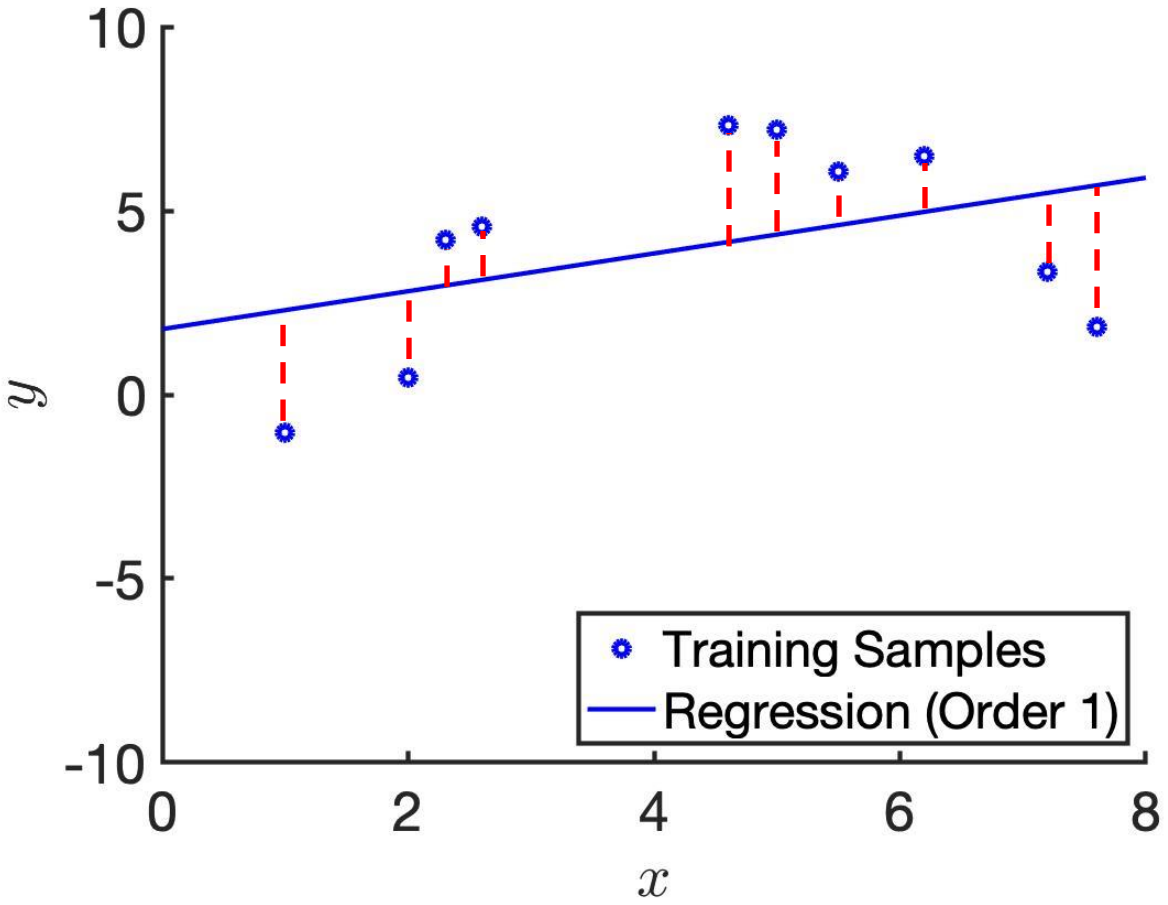
- Regression fits the training data well but not the test data
- has very big prediction error when using test data (MSE is big)
- Because training data set is small while trying to fit a very complicated model of order 9, alot parameter
- Training good, test bad ⇒ over fitting

Overfitting Example



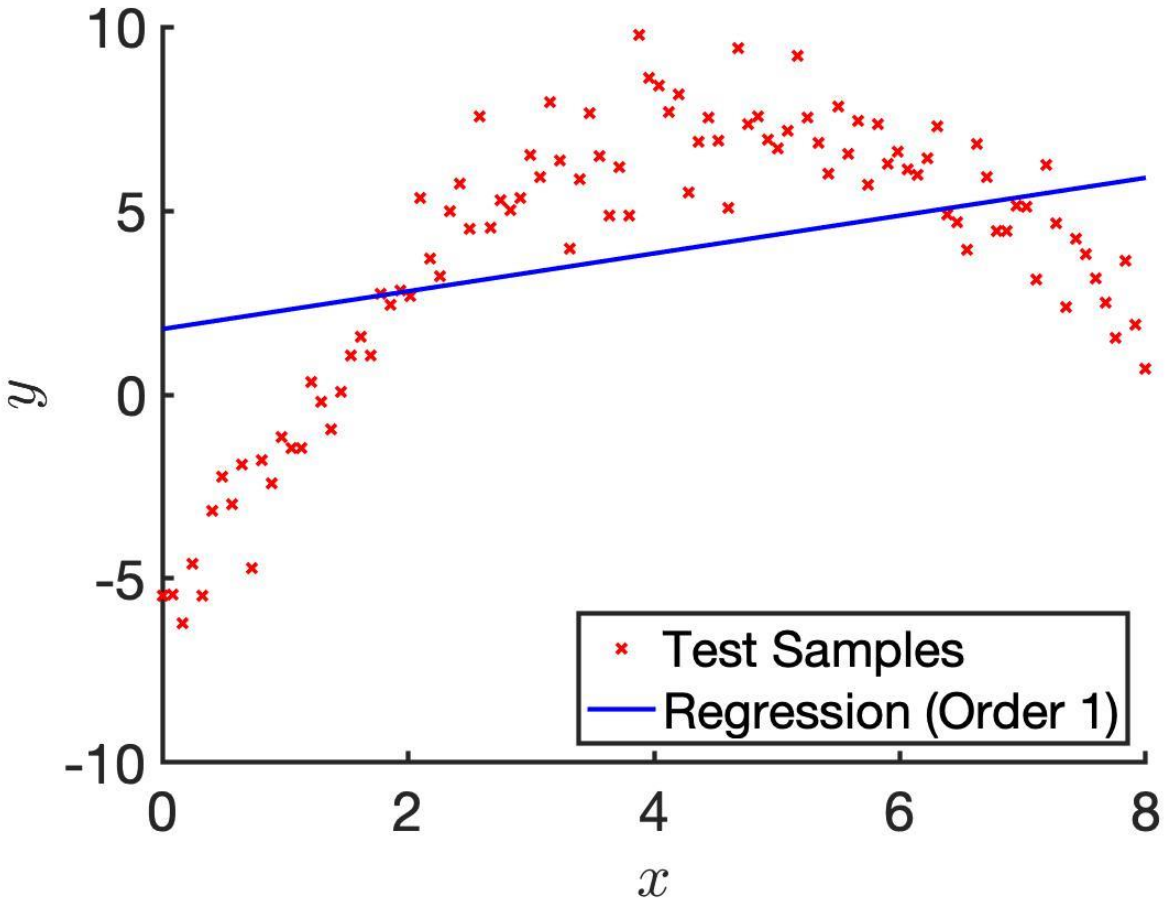
- If we take one of the blue lines and compute the square of its length, this is called **"squared error" for that particular data point**
- If we average squared errors across all the red crosses, it's called **mean squared error (MSE)** in the test set

Underfitting Example



	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 1	Bad	

Underfitting Example

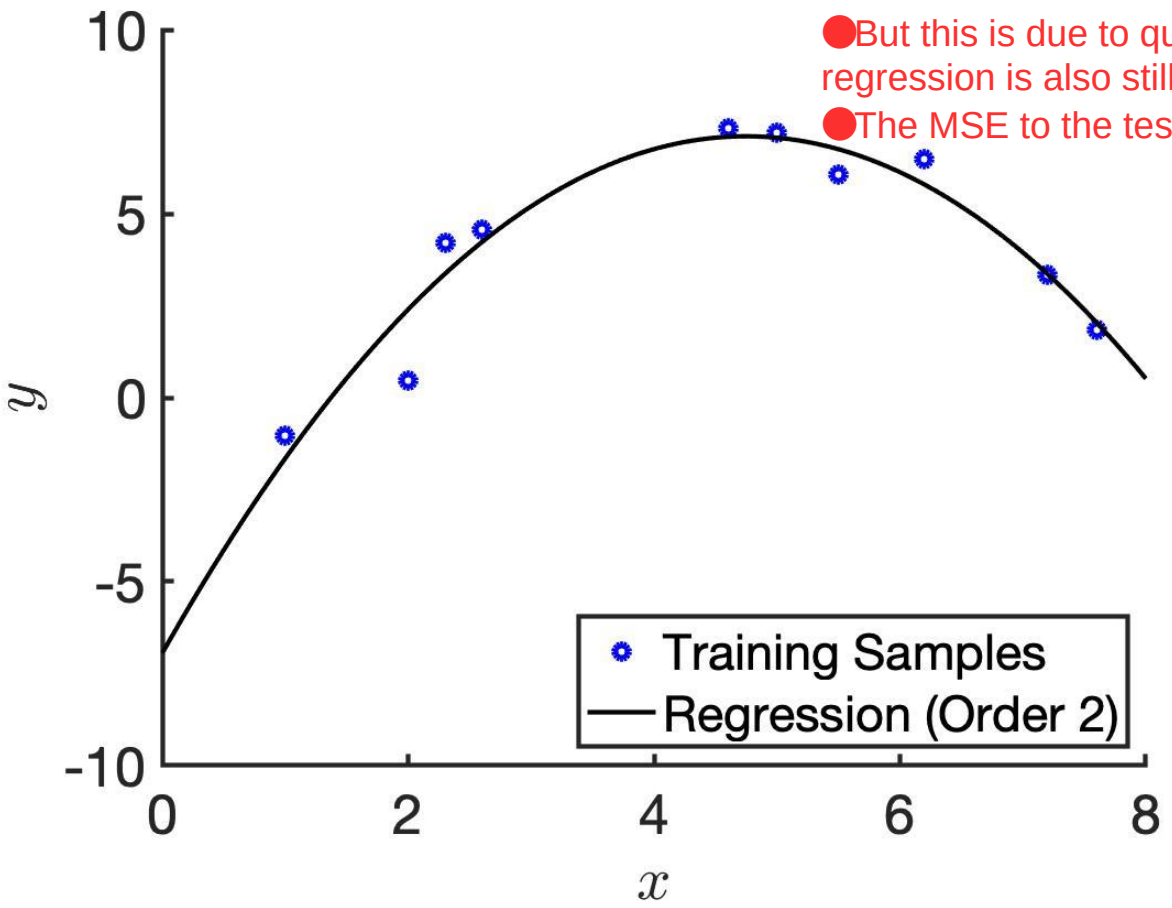


	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 1	Bad	Bad

- Regression is bad for the training data and the test data ⇒ under fitting
- Means model is too simple

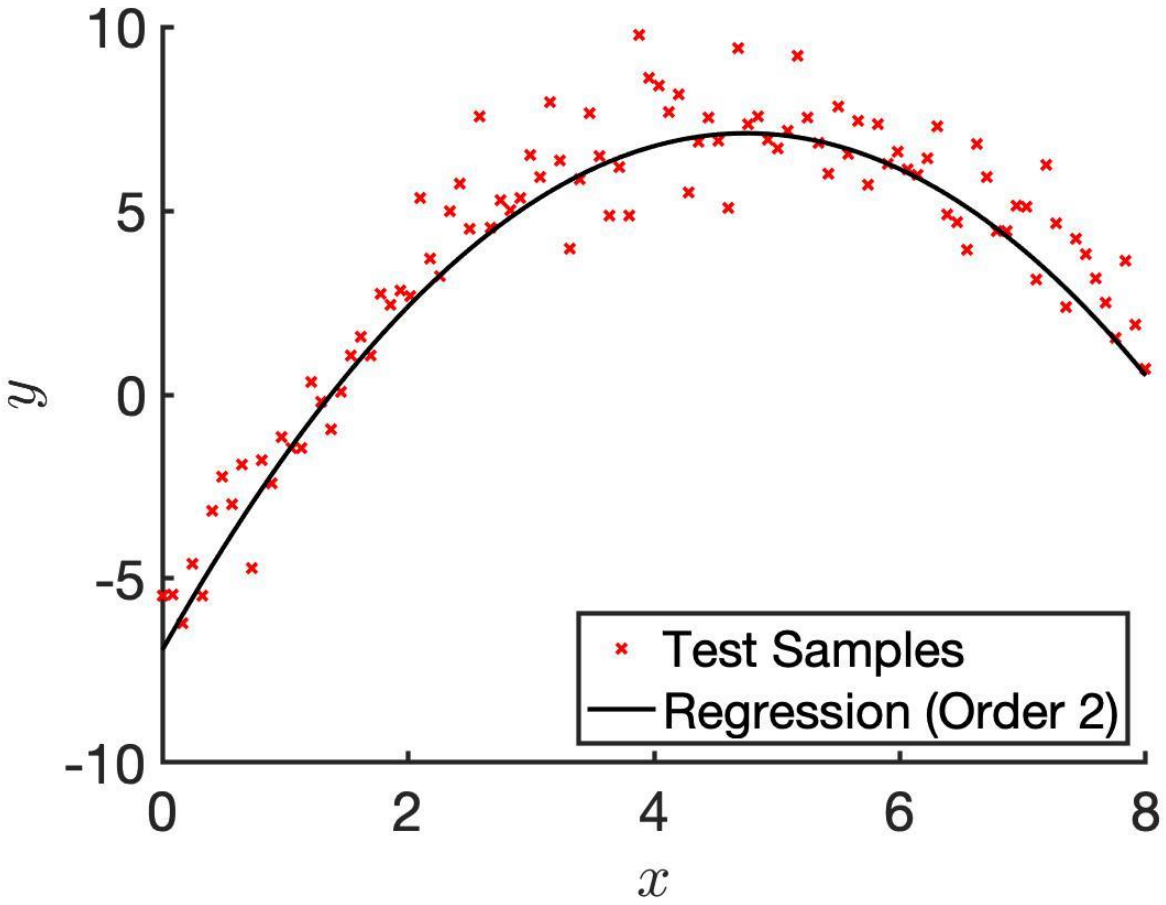
"Just Nice"

- Regression does not pass through all the training data, the MSE is actually larger than the 9 order overfitting case
- But this is due to quadratic only having 3 parameters, so the regression is also still good
- The MSE to the test data is also very good



	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 1	Bad	Bad
Order 2	Good	

“Just Nice”



	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 1	Bad	Bad
Order 2	Good	Good

Overfitting & Underfitting

Overfitting →
Underfitting →

	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 1	Bad	Bad
Order 2	Good	Good

Overfitting & Underfitting

- **Overfitting** occurs when model predicts the training data well, but predicts new data (e.g., from test set) poorly
- **Reason 1**
  - Model is too complex for the data
  - Previous example: Fit order 9 polynomial to 10 data points
- **Reason 2**
  - Too many features but number of training samples too small
  - Even linear model can overfit, e.g., linear model with 9 input features (i.e., w is 10-D) and 10 data points in training set => data might not be enough to estimate 10 unknowns well
- **Solutions**
  - Use **simpler models** (e.g., lower order polynomial)
  - Use **regularization** (see next part of lecture)

usually when high accuracy on training set but low on accuracy on test set

can reduce features too

Overfitting & Underfitting

- **Underfitting** is the inability of trained model to predict the targets in the training set

usually when there is already low accuracy on training set
52% is considered low
increase max tree deoth (more complex model)
decrease min sample for splitting node (more complex)
use random forest instead of decision tree will help to increase test accuracy too because it reduces variance

- **Reason 1**
  - Model is too simple for the data
  - Previous example: Fit order 1 polynomial to 10 data points that came from an order 2 polynomial
  - **Solution**: Try more complex model
- **Reason 2**
  - Features are not informative enough
  - **Solution**: Try to develop more informative features

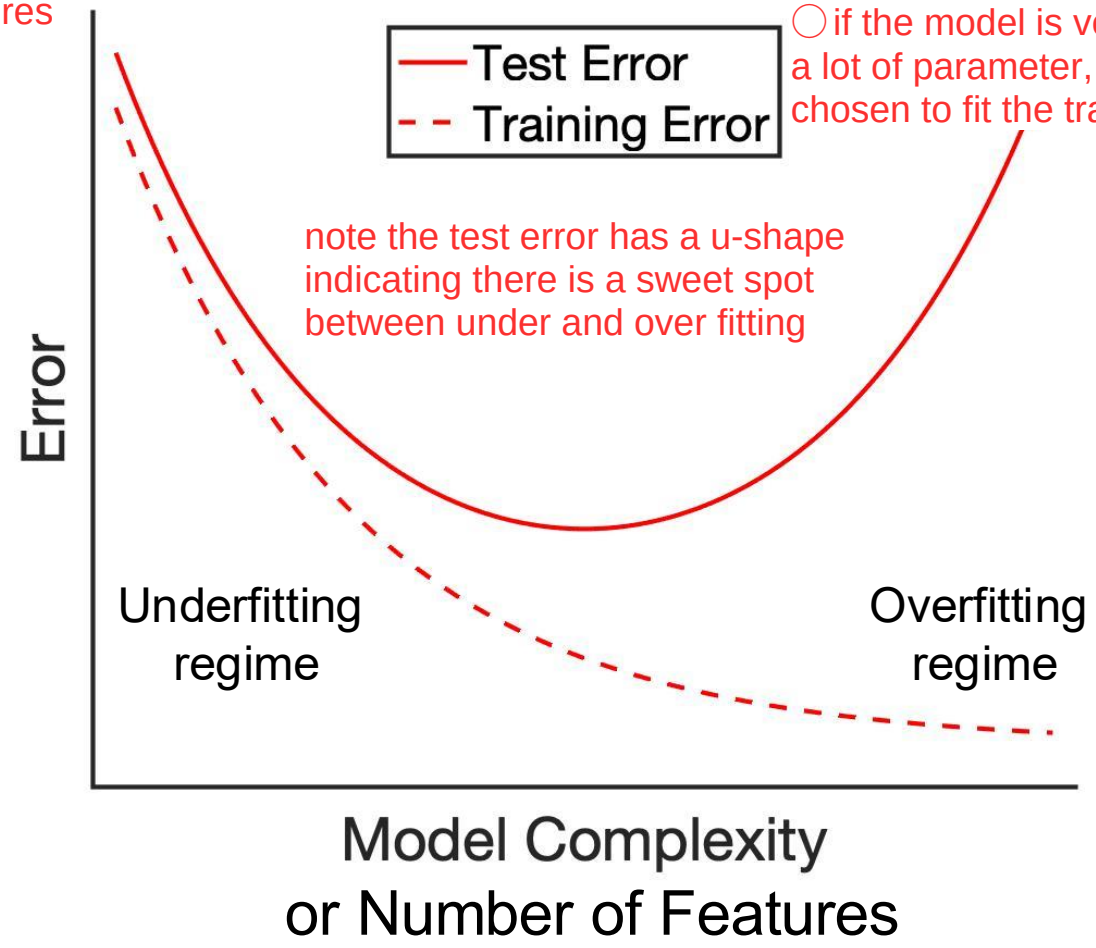
perform feature extraction
perform better feature selection

Overfitting / Underfitting Schematic

When both training and test error are high, is when model complexity is too low or number of features is not enough ⇒ underfitting

When training error is low but test error is high, is when model complexity is too high or number of features is too much ⇒ over fitting

◯ if the model is very complicated and has a lot of parameter, the parameter can be chosen to fit the training data very well



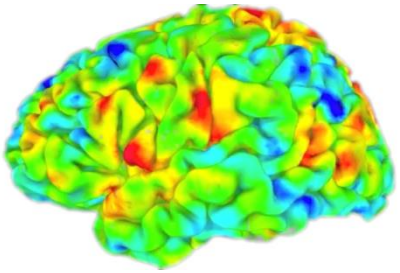
huh

Non-invasive Mind Reading Using AI Models

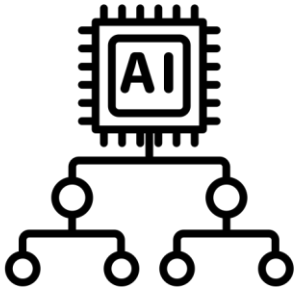
What subjects are viewing?

AI reconstruct from subjects' brain activity

Image or Video



Brain recording



Brain Foundation Model + Generative AI

Reconstructed Video

Poll questions

Feature Selection

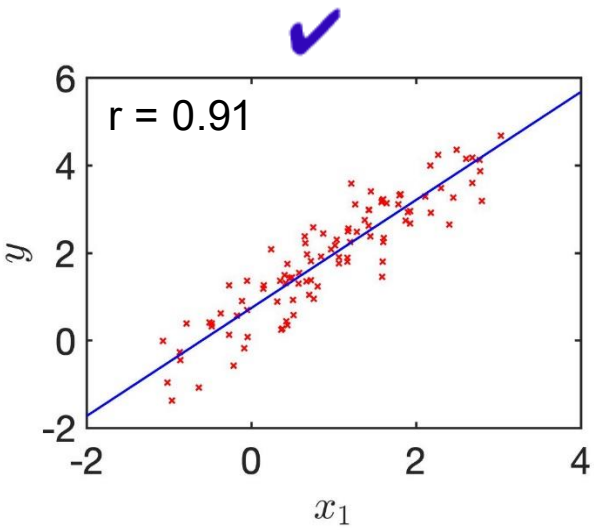
- Less features might reduce overfitting
  - Want to discard useless features & keep good features, so perform feature selection
- Feature selection procedure
  - Step 1: feature selection in training set
  - Step 2: fit model using selected features in training set
  - Step 3: evaluate trained model using test set
- Very common mistake
  - Feature selection with test set (or full dataset) leads to inflated performance
  - Do not perform feature selection with test data

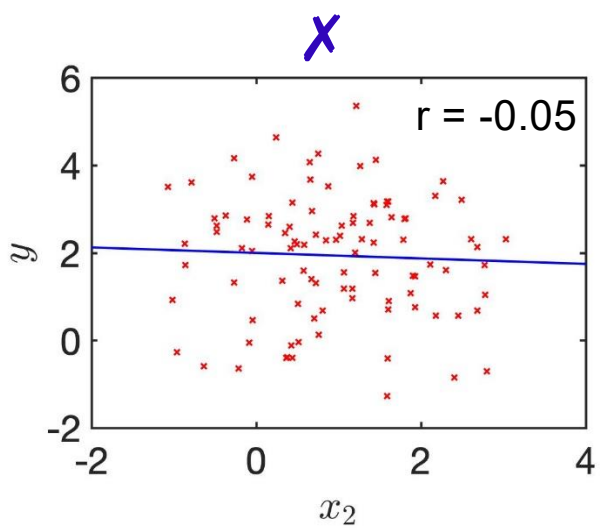
as the names implied, training set is used to train and test set is used to test how good the model is

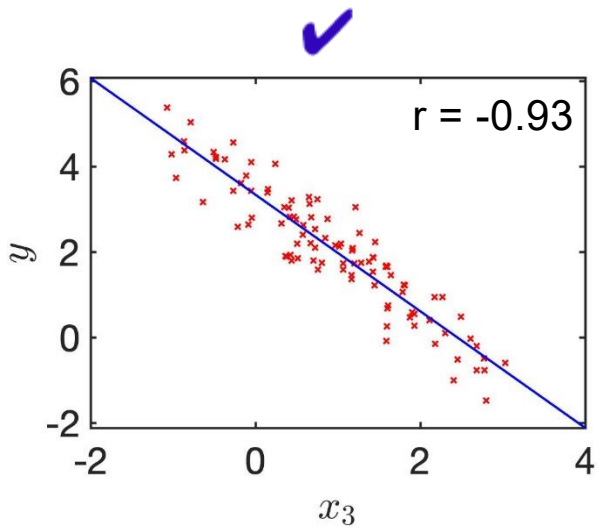
- Both will lead to information leakage
- If need to compare two models, give both the same training set and test with the same test set

Selecting Features With Pearson's r

- Given features x, we want to predict target y
- Assume x & y both continuous
- Compute Pearson's correlation coefficient between each feature & target y in the training set
  - Pearson's correlation r measures linear relationship between two variables







Selecting Features With Pearson's r

- Given features x, we want to predict target y
- Assume x & y both continuous
- Compute Pearson's correlation coefficient between each feature & target y in the training set
  - Pearson's correlation r measures linear relationship between two variables
- Two options
  - Option 1: Pick K features with largest absolute correlations
  - Option 2: Pick all features with absolute correlations > C
  - K & C are "magic" numbers set by the ML practitioner
- Other metrics besides Pearson's correlation are possible

This version is correlation with target (feature-target correlation)
- this removes features that have little to no linear relationship with the target.
- reduces dimensionality, allowing models to be simpler so less over fitting

another way is to compute correlation between features to identify redundant features. Features that have high correlation are most likely contributing similar information. This also avoids multi-collinearity

Regularization

go back to ridge, already explained quite a bit there

- Regularization is an umbrella term that includes methods forcing learning algorithm to build less complex models.
- Motivation 1: Solve an ill-posed problem
  - For example, estimate 10th order polynomial with just 5 datapoints
- Motivation 2: Reduce overfitting
- For example, in previous lecture, we added $\lambda \mathbf{w}^T \mathbf{w}$:

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

- Minimizing with respect to $\mathbf{w}$, primal solution is

$$\hat{\mathbf{w}} = (\mathbf{P}^T \mathbf{P} + \lambda \mathbf{I})^{-1} \mathbf{P}^T \mathbf{y}$$

regularisation changes the shape of the cost function, hence global minimum will change (dont know increase or decrease) and (may or may not speed up optimisation)

- For $\lambda > 0$, matrix becomes invertible (Motivation 1)

Regularization

- Regularization is an umbrella term that includes methods forcing learning algorithm to build less complex models.
- Motivation 1: Solve an ill-posed problem
  - For example, estimate 10th order polynomial with just 5 datapoints
- Motivation 2: Reduce overfitting
- For example, in previous lecture, we added $\lambda \mathbf{w}^T \mathbf{w}$:

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

- Minimizing with respect to $\mathbf{w}$, primal solution is

$$\hat{\mathbf{w}} = (\mathbf{P}^T \mathbf{P} + \lambda \mathbf{I})^{-1} \mathbf{P}^T \mathbf{y}$$

see ridge. tldr large w will cause model to fluctuate even with little variance in input data (expound MSE). Impose high penalty on large w

- $\hat{\mathbf{w}}$ might also perform better in test set, i.e., reduces overfitting (Motivation 2) – will show example later

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{Pw} - \mathbf{y})^T (\mathbf{Pw} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

Cost function quantifying data fitting error in training set

Regularization

cost function C(w) = DataLoss(w) + λRegularization(w).

By increasing regularization hyperparameter, we are putting more weight on the regularization term, so (1) Regularization (w*) > Regularization (w**) and less weight on the data-loss term, so (4) DataLoss(w*) < DataLoss(w**).

Regularisation term has more weight and minimised more, meaning the dataloss is minimised less. Hence regularisatio results in greater loss.

Another way to think is regularisation causes weights to be not as accurate anymore, so there will be more loss

Regularization

Because its squared, its called L2 - Regularization

Bigger λ means regularisation dominates, forces model to be less complex at the cost of increasing MSE to training data

- Consider minimization from previous slide

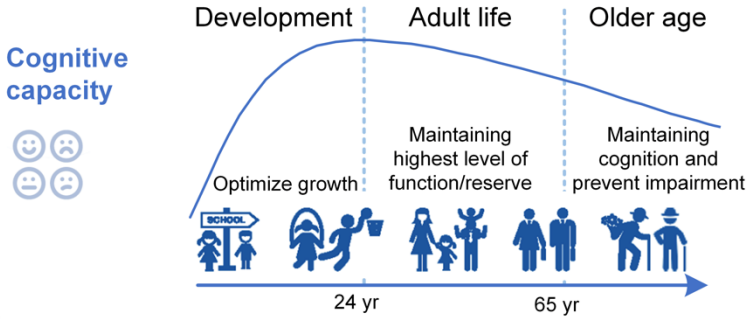
$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

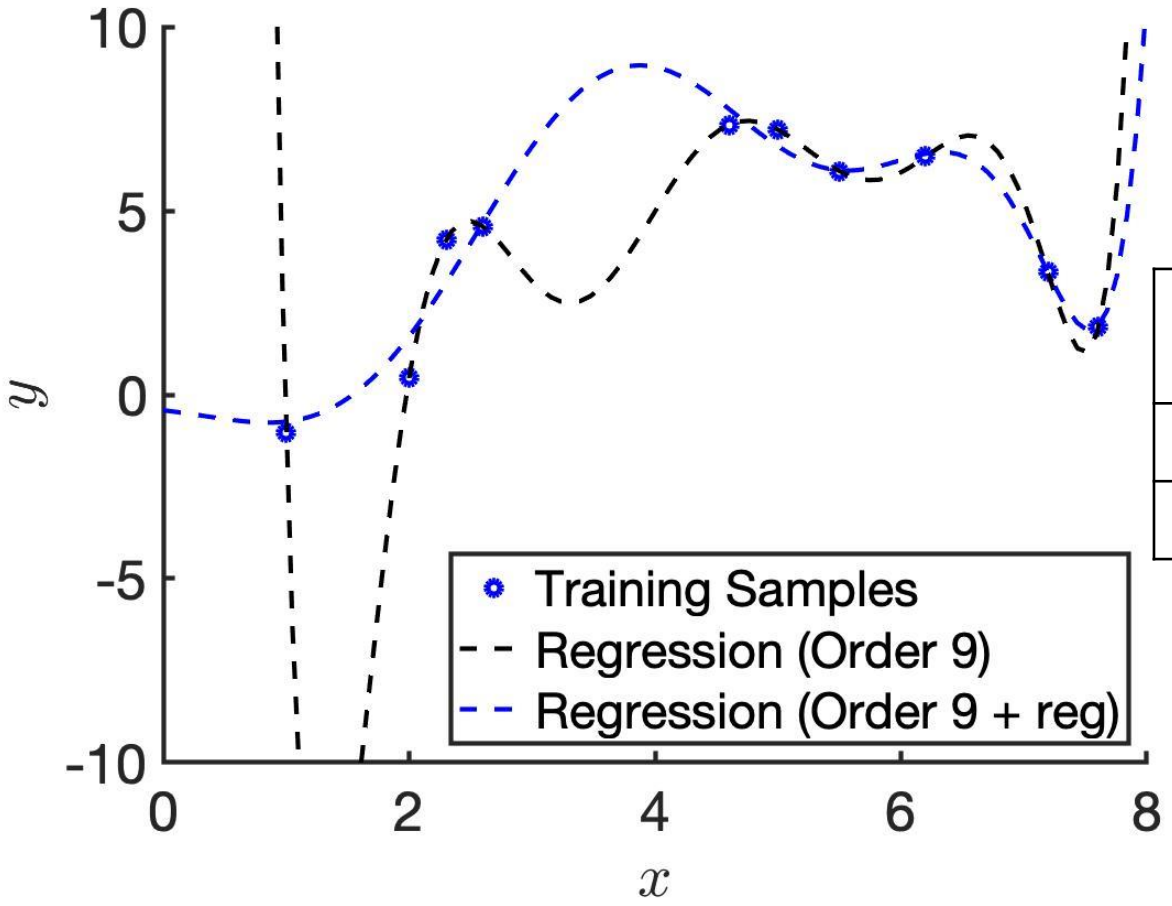
- $\mathbf{w}^T \mathbf{w} = w_0^2 + w_1^2 + \cdots + w_d^2$ — L2 - Regularization
- Encourage $w_0, \cdots, w_d$ to be small (also called shrinkage or weight-decay) => constrain model complexity
- More generally, most machine learning algorithms can be formulated as the following optimization problem

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, \mathbf{Data\text{-}Loss}(\mathbf{w}) + \lambda \mathbf{Regularization}(\mathbf{w})$$

- **Data-Loss(w)** quantifies fitting error to training set given parameters **w:** smaller error => better fit to training data
- **Regularization(w)** penalizes more complex models

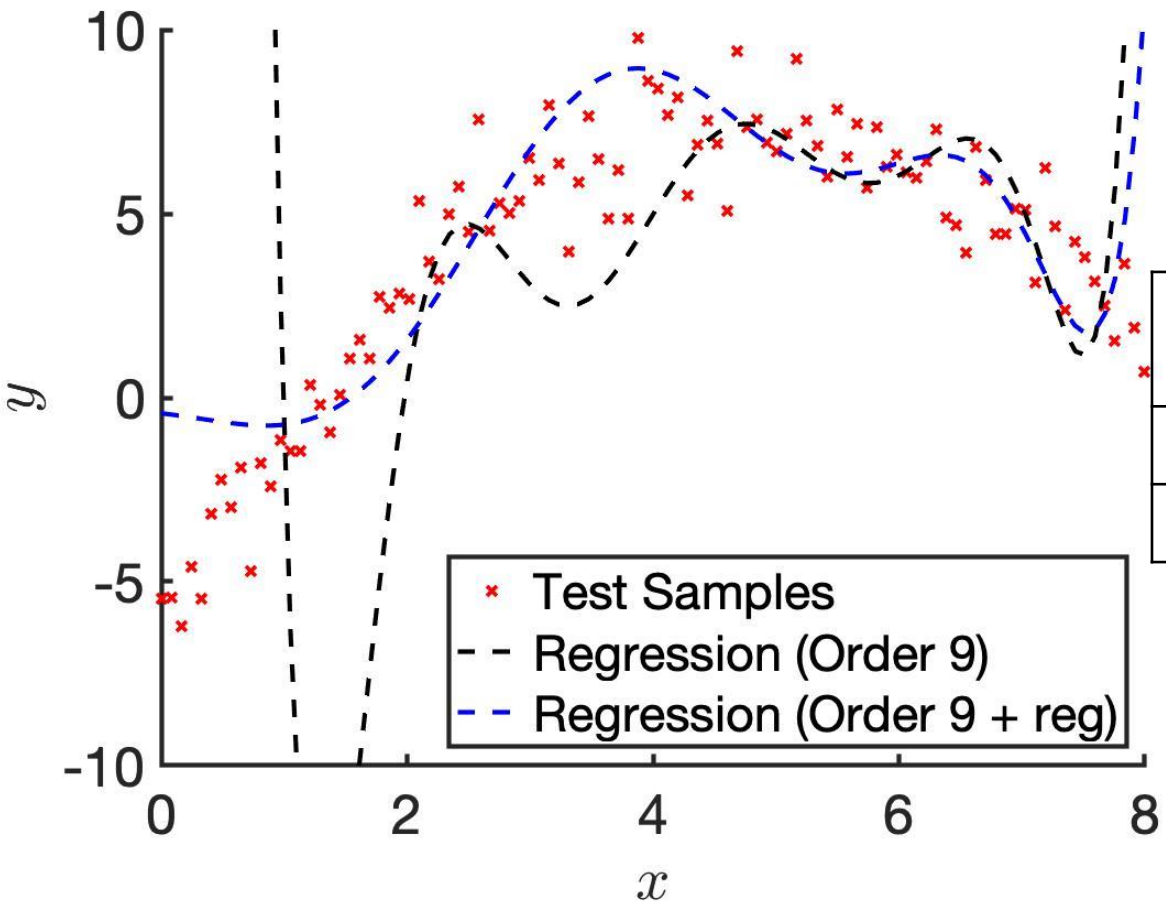
Regularization Example





	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 9, $\lambda = 1$	Good	

Regularization Example



Can see that with Ridge regression, the blue curve does not fit the training data as well anymore. But it fit the overall data much better

	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 9, $\lambda = 1$	Good	Good

Python demo

Bias-Variance Decomposition Theorem

Both bias and variance are fundamental sources of error. Both contribute to prediction error

- **Test error** = Bias Squared + Variance + Irreducible Noise
  - Mathematical details in optional uploaded material (won't be tested)
- "**Variance**" refers to variability of prediction models across different training sets

  Variance is error for being too sensitive to random noise, from overfitting (e.g. high w) - low training error, high test error.
  - In previous example, every time the training set of 10 samples changes, the trained model changes
  - "Variance" quantifies variability across trained models
- "**Bias**" refers to how well an average prediction model will perform
  - In previous example, every time the training set of 10 samples changes, the trained model changes

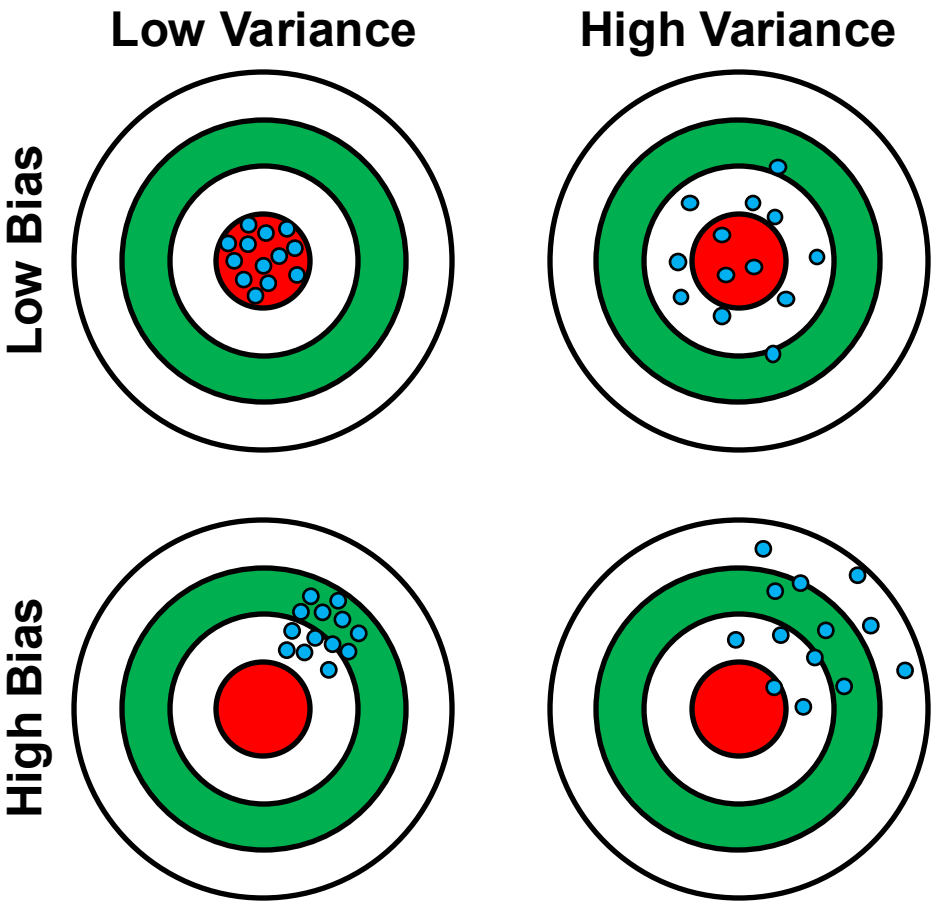
    Bias is error caused by wrong asssumptions from oversimplified models (tends of underfit - high training error and test error).
  - If we average the trained models, how well will this average trained model perform?
- "**Irreducible Noise**" reflects the fact that even if we are perfect modelers, it might not be possible to predict target y with 100% accuracy from feature(s) x

Bias versus Variance

- Suppose we are trying to predict red target below:

Low Bias: blue predictions on average close to red target
Low Variance: low variability among blue predictions

High Bias: blue predictions on average not close to red target
Low Variance: Low variability among blue predictions

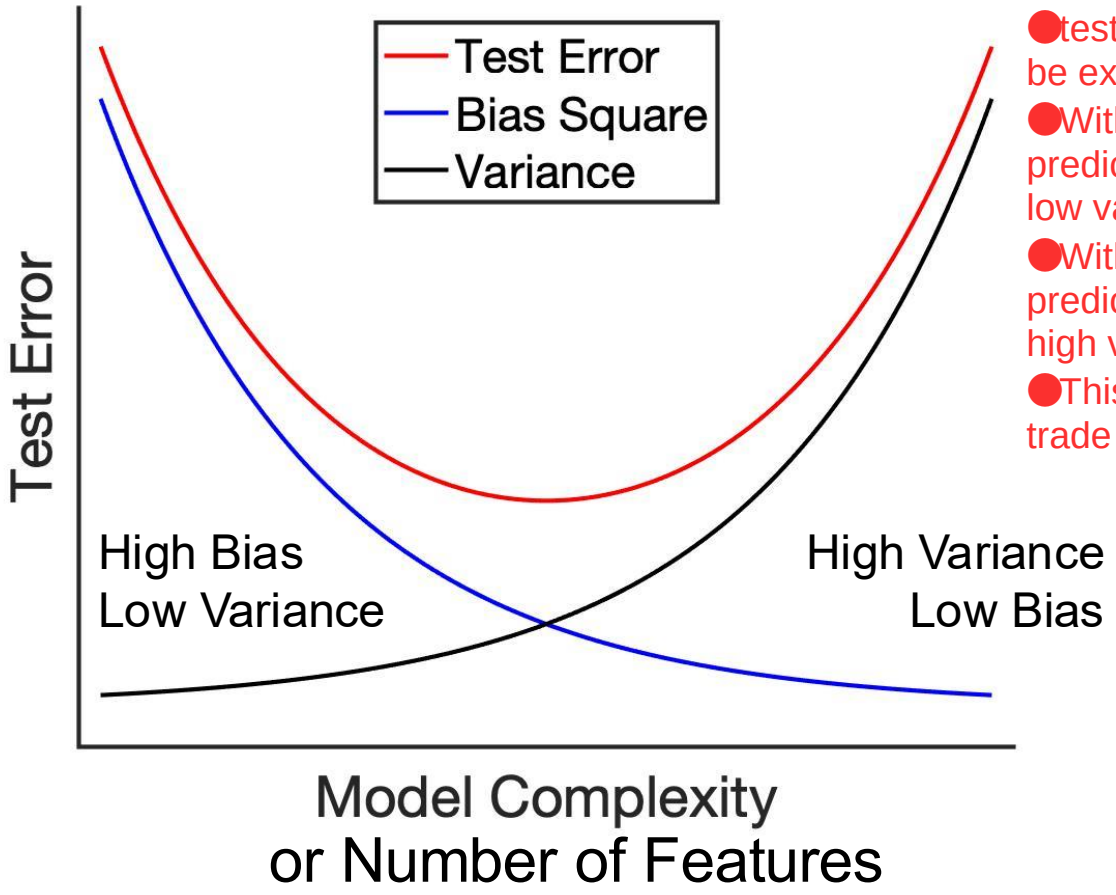


Low Bias: blue predictions on average close to red target
High Variance: large variability among blue predictions

High Bias: blue predictions on average not close to red target
High Variance: high variability among blue predictions

Bias + Variance Trade off

- Test error = Bias Squared + Variance + Irreducible Noise

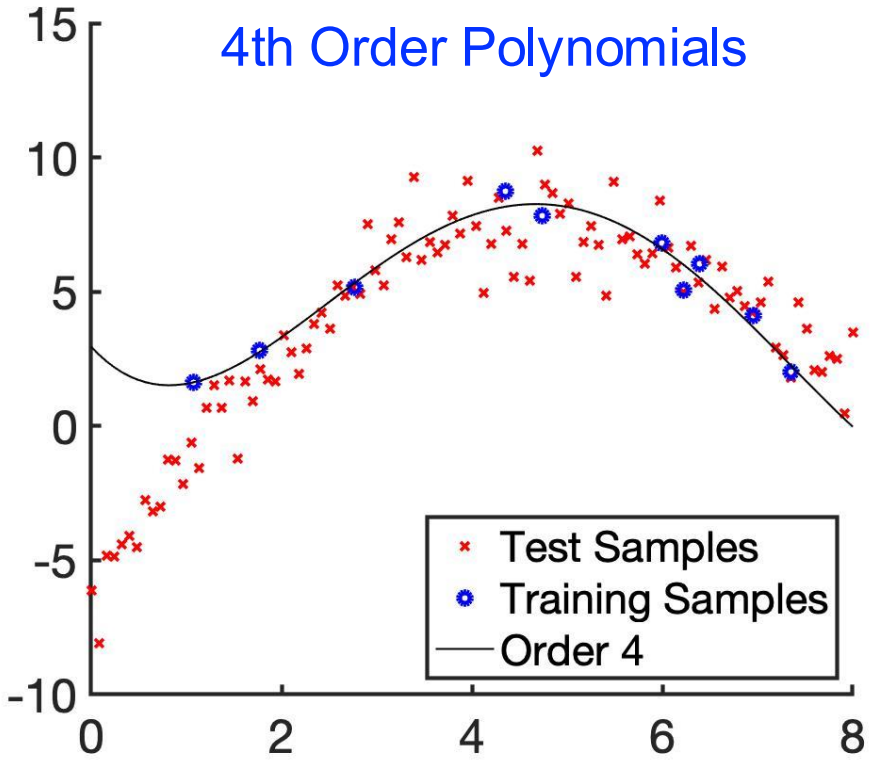


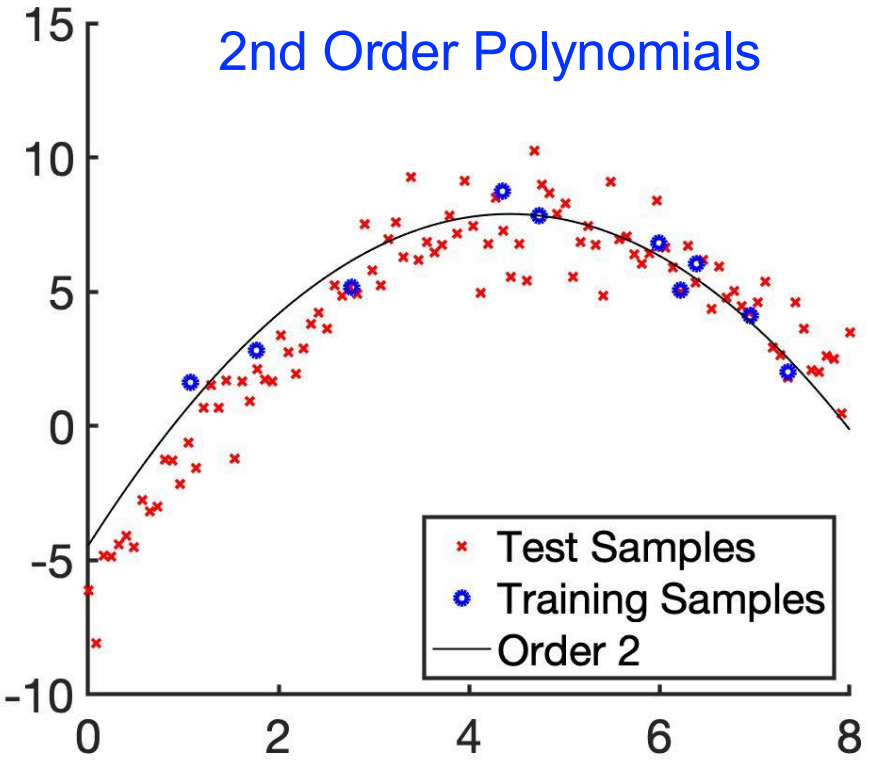
- test error with u shape can be explained here
- With simple models, predictions has high bias but low variance
- With complex models, predictions has low bias but high variance
- This is the bias variance trade off

Bias + Variance Example

- Simulate data from order 2 polynomial (+ noise)
- Randomly sample 10 training samples each time
- Fit with order 2 polynomial
- Fit with order 4 polynomial

- 4th order has big error on LHS
- 2nd order fit training and test well
- Both fit training quite well

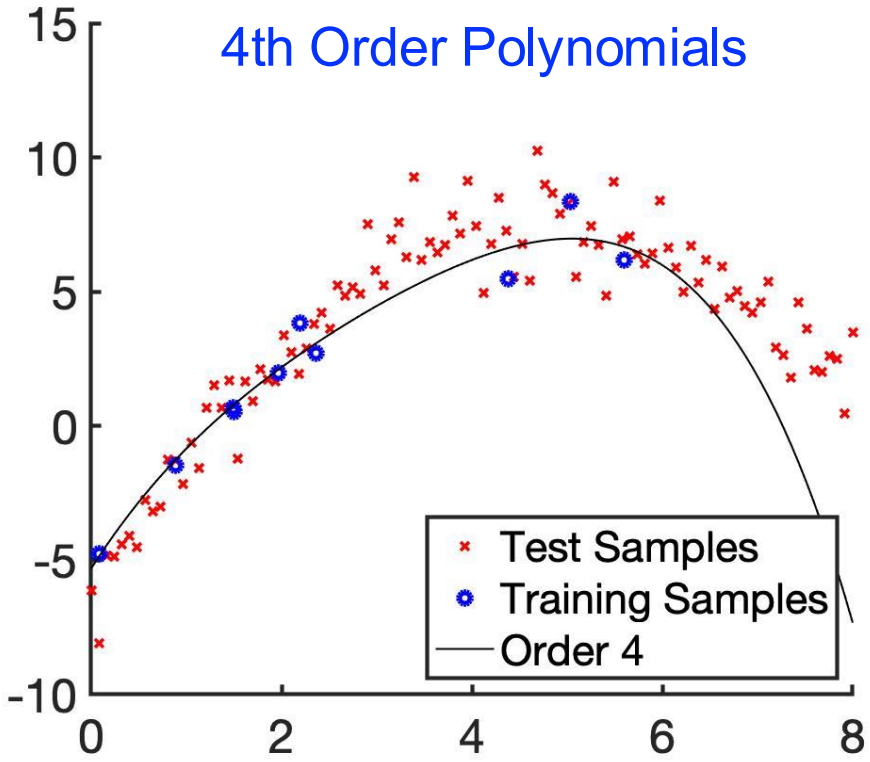


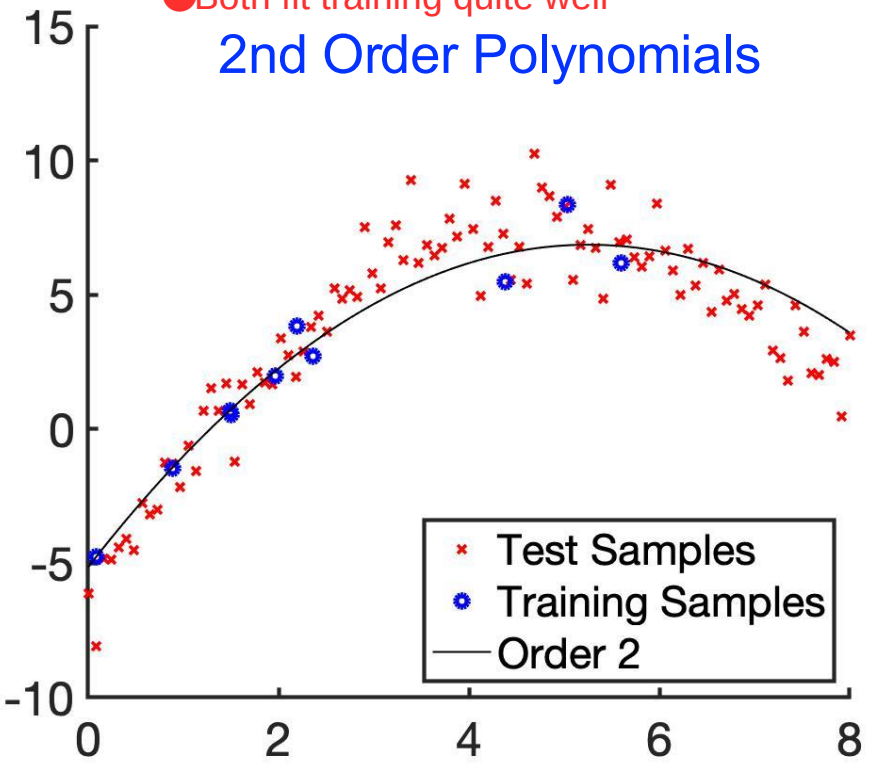


Bias + Variance Example

- Simulate data from order 2 polynomial (+ noise)
- Randomly sample 10 training samples each time
- Fit with order 2 polynomial
- Fit with order 4 polynomial

- 4th order has big error on RHS
- 2nd order fit training and test well
- Both fit training quite well

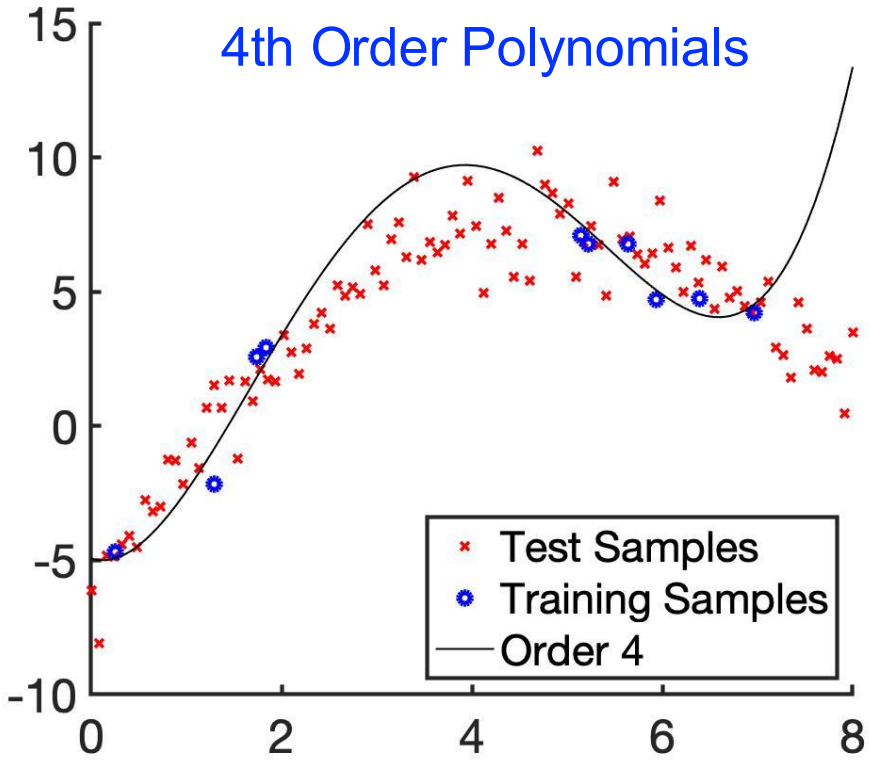


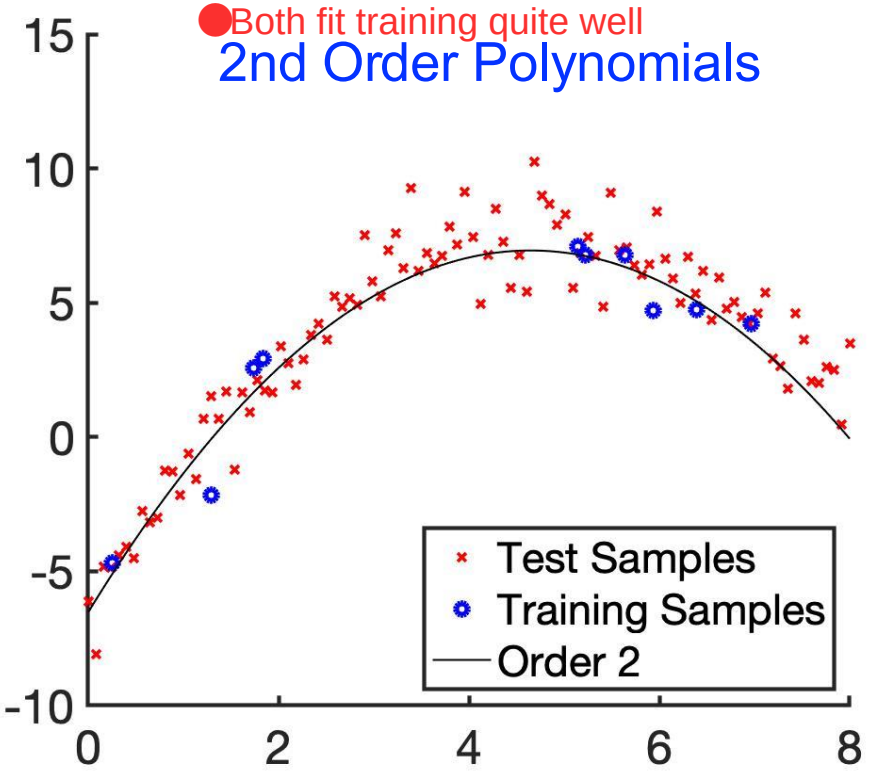


Bias + Variance Example

- Simulate data from order 2 polynomial (+ noise)
- Randomly sample 10 training samples each time
- Fit with order 2 polynomial
- Fit with order 4 polynomial

- 4th order has big error on RHS
- 2nd order fit training and test well
- Both fit training quite well

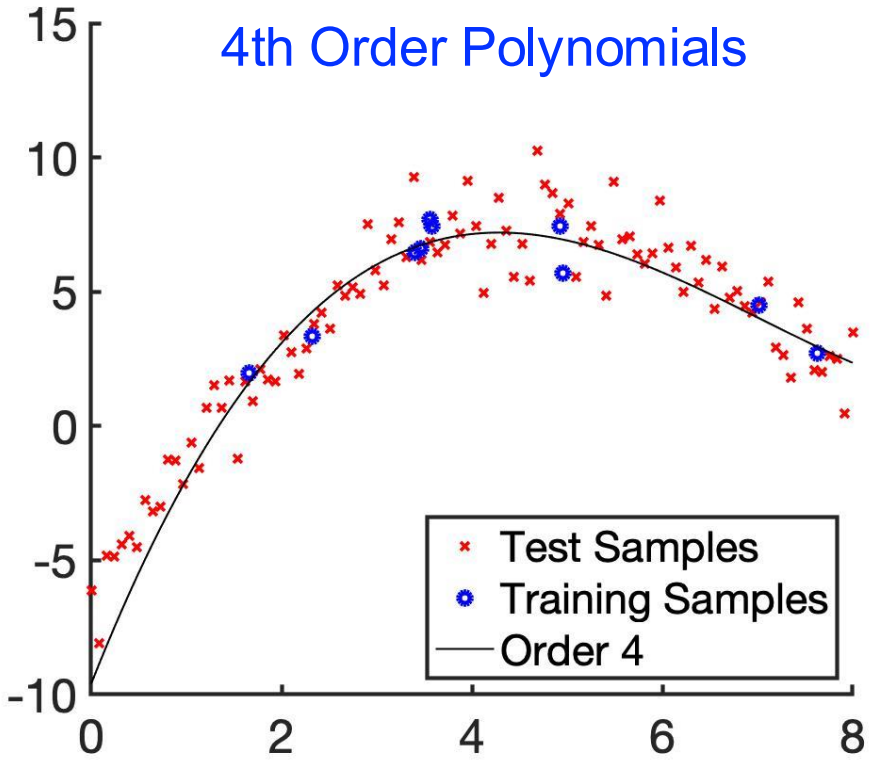


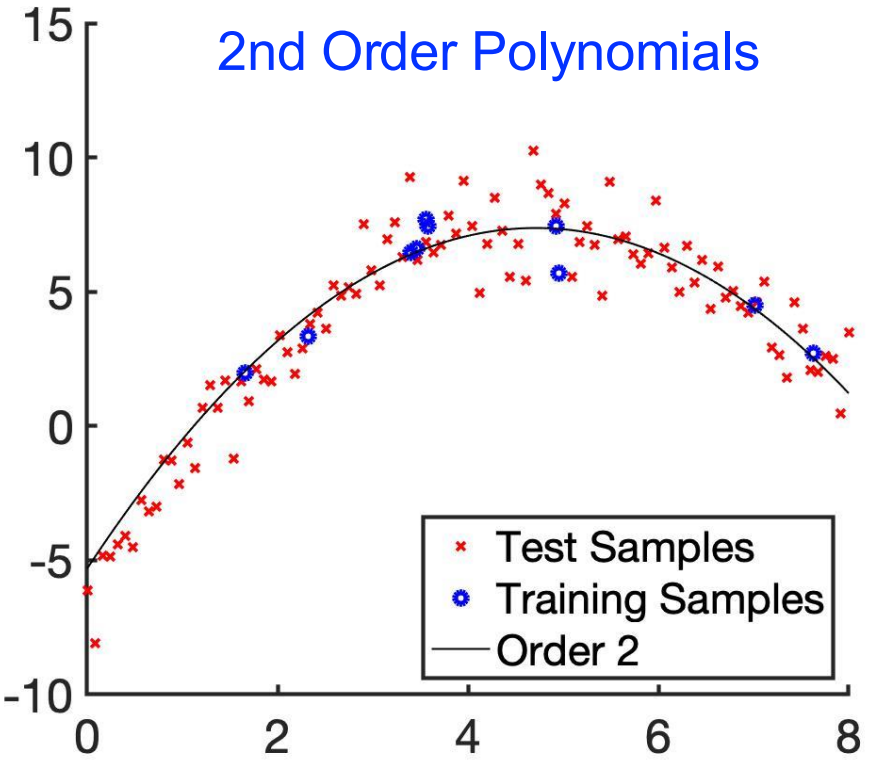


Bias + Variance Example

- Simulate data from order 2 polynomial (+ noise)
- Randomly sample 10 training samples each time
- Fit with order 2 polynomial
- Fit with order 4 polynomial

Both fit training and test quite well

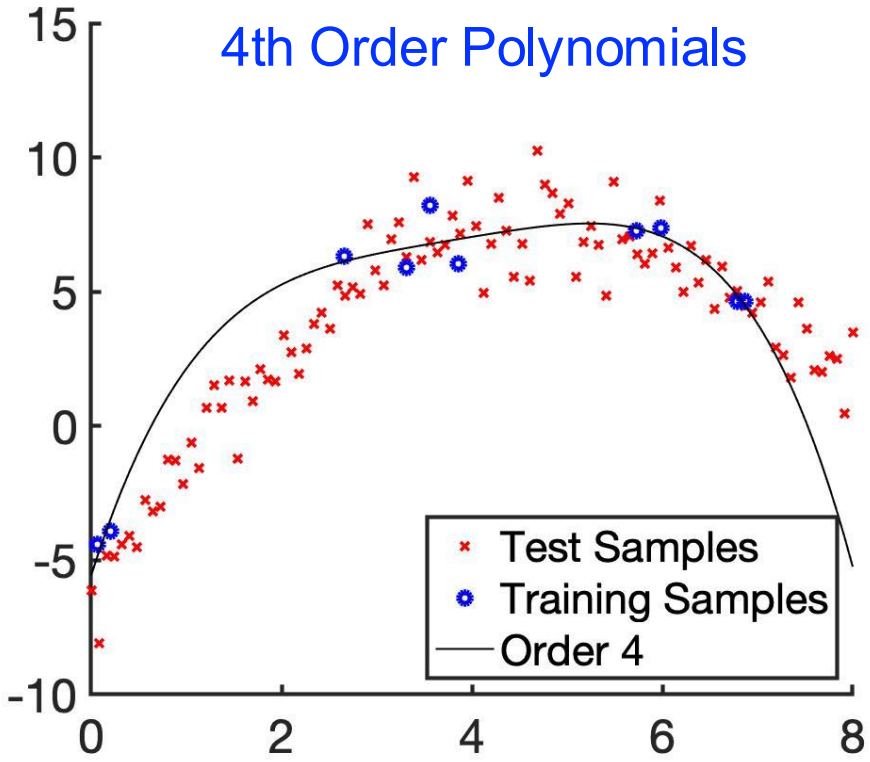


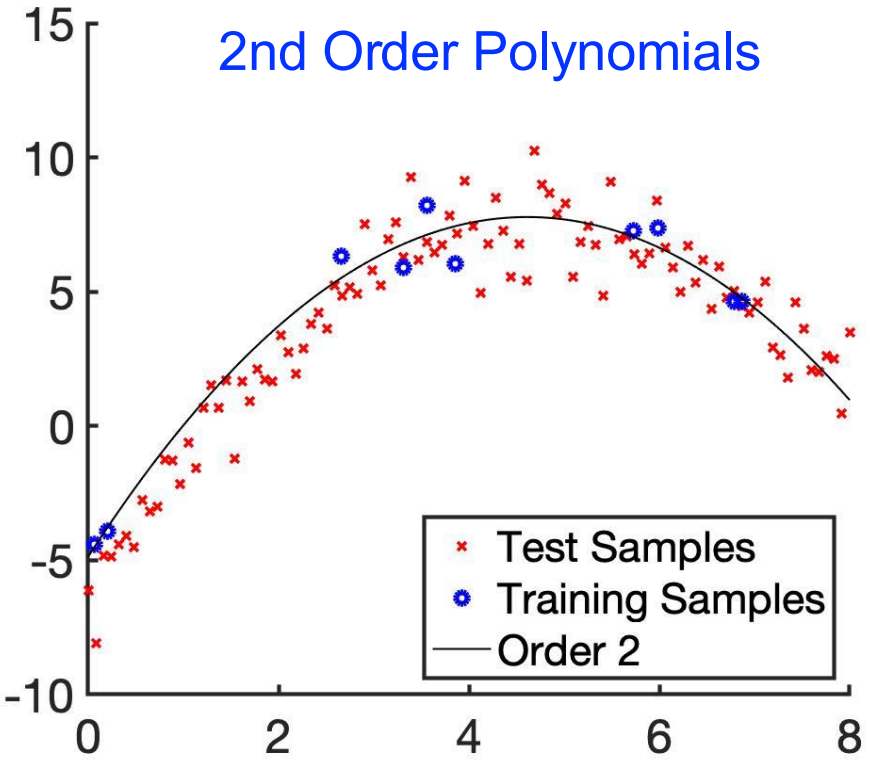


Bias + Variance Example

- Simulate data from order 2 polynomial (+ noise)
- Randomly sample 10 training samples each time
- Fit with order 2 polynomial
- Fit with order 4 polynomial

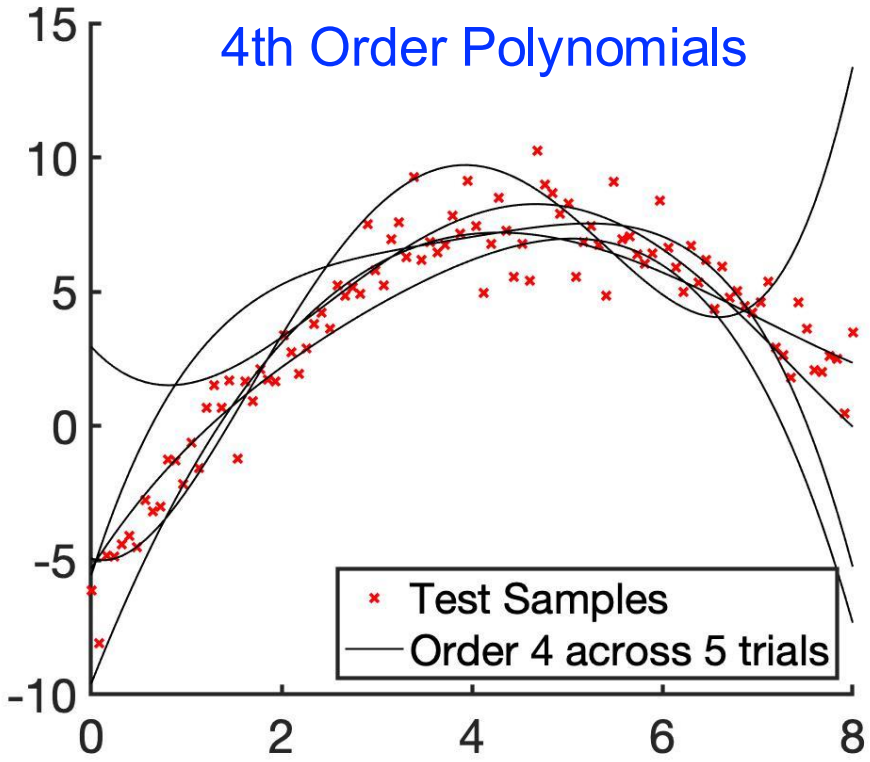
- 4th order has big error on LHS
- 2nd order fit training and test well
- Both fit training quite well

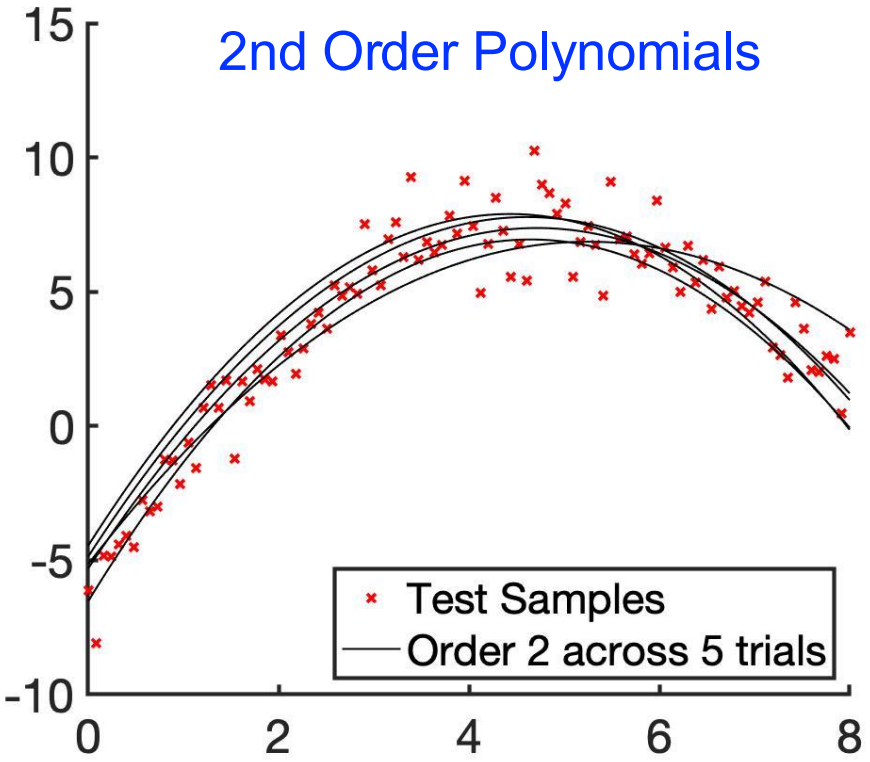




Bias + Variance Example

- Simulate data from order 2 polynomial (+ noise)
- Randomly sample 10 training samples each time
- Fit with order 2 polynomial: low variance
- Fit with order 4 polynomial: high variance

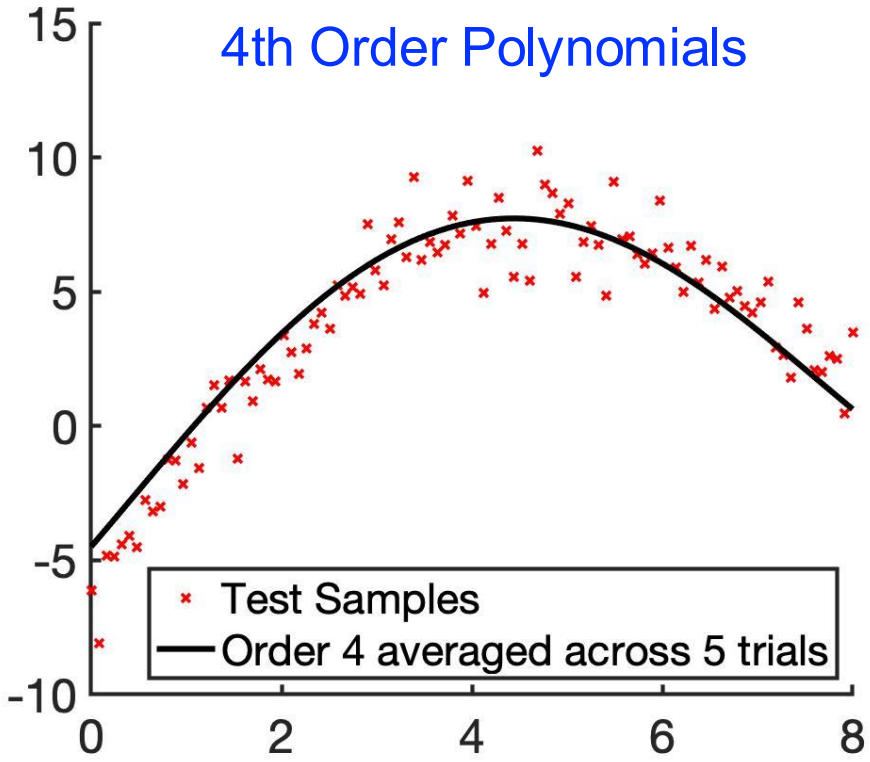


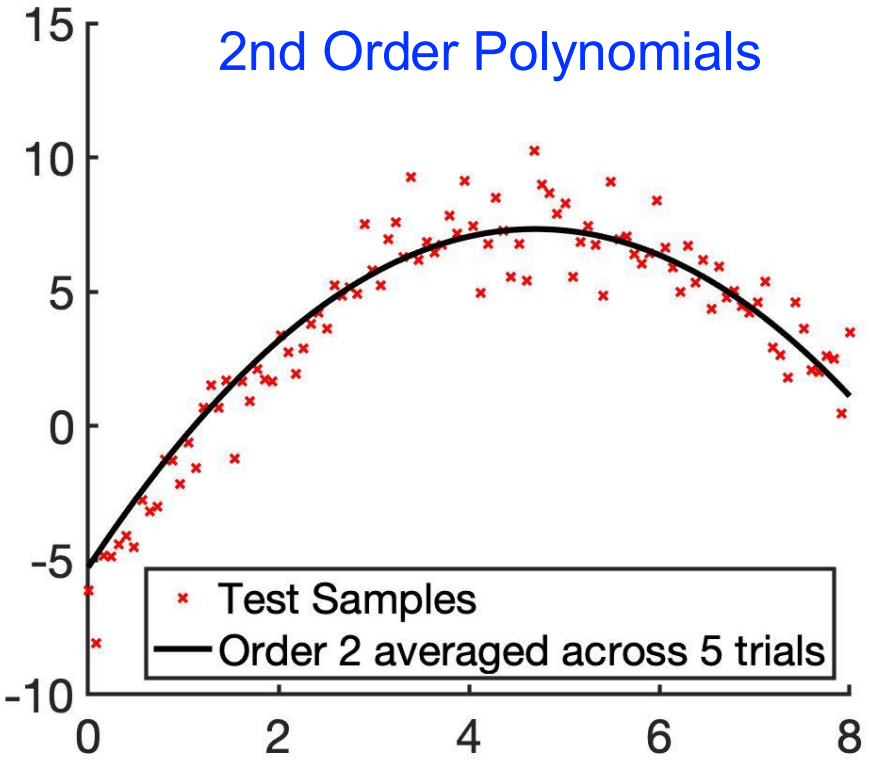


Bias + Variance Example

- Simulate data from order 2 polynomial (+ noise)
- Randomly sample 10 training samples each time
- Fit with order 2 polynomial: low variance, low bias
- Fit with order 4 polynomial: high variance, low bias

- Both averages follows the test data well
- Both has low bias

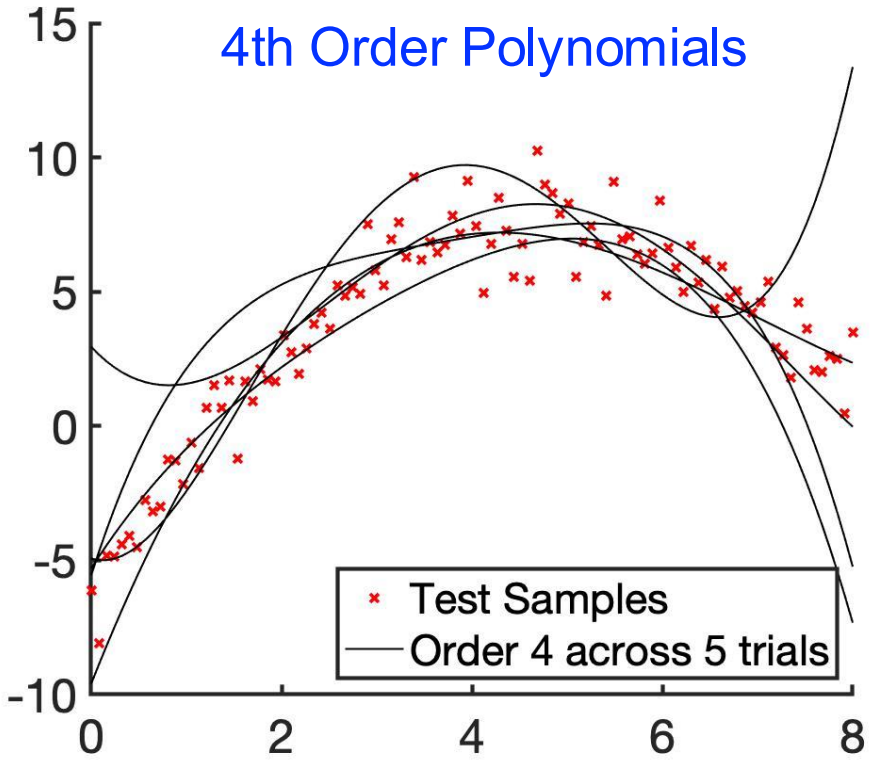


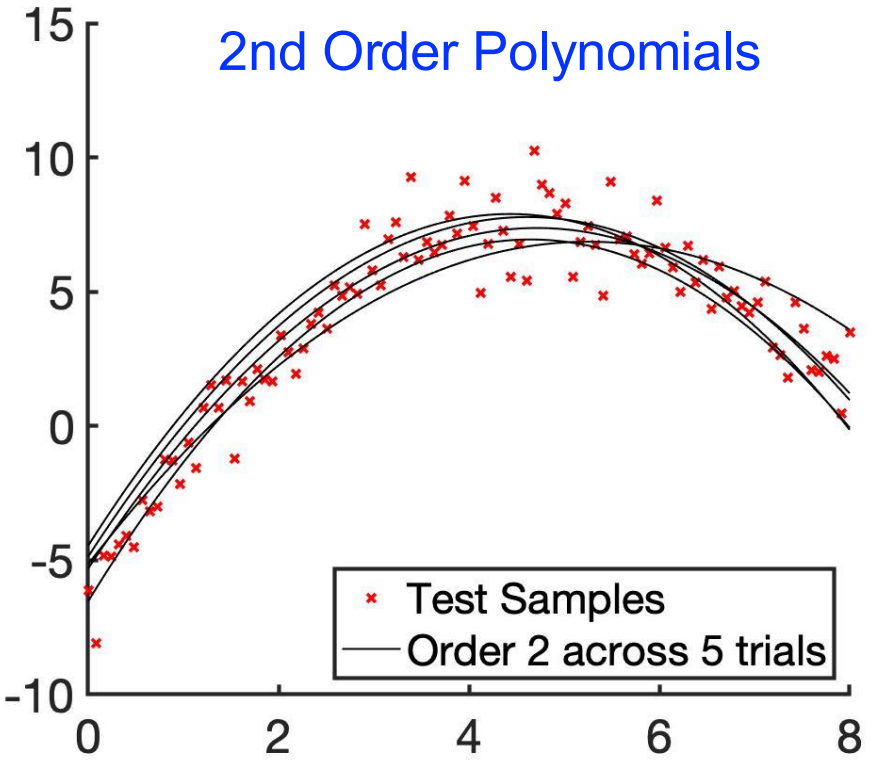


Bias + Variance Example

- Simulate data from order 2 polynomial (+ noise)
- Randomly sample 10 training samples each time
- Fit with order 2 polynomial: low variance, low bias
- Fit with order 4 polynomial: high variance, low bias

} Order 2 Achieves Lower Test Error





Summary

- Overfitting, underfitting & model complexity
  - Overfitting: low error in training set, high error in test set
  - Underfitting: high error in both training & test sets
  - Overly complex models can overfit; Overly simple models can underfit
- Feature selection
  - Extract useful features from training set
- Regularization (e.g., L2 regularization)
  - Solve “ill-posed” problem (e.g., more unknowns than data points)
  - Reduce overfitting
- Bias-Variance Decomposition Theorem
  - Test error = Bias Squared + Variance + Irreducible Noise
  - Can be interpreted as trading off bias & variance:
    - Overly complex models can have high variance, low bias
    - Overly simple models can have low variance, high bias

EE2211 Introduction to Machine Learning

Lecture 8

Juan Helen Zhou
helen.zhou@nus.edu.sg

Electrical and Computer Engineering Department
National University of Singapore

Acknowledgement: EE2211 development team
Thomas, Helen, Xinchao, Kar-Ann, Chen Khong, Robby and Haizhou

Course Contents

- Introduction and Preliminaries (Xinchao)
  - Introduction
  - Data Engineering
  - Introduction to Probability and Statistics
- Fundamental Machine Learning Algorithms I (Helen)
  - Systems of linear equations
  - Least squares, Linear regression
  - Ridge regression, Polynomial regression
- Fundamental Machine Learning Algorithms II (Helen)
  - Over-fitting, bias/variance trade-off
  - Optimization, Gradient descent
  - Decision Trees, Random Forest
- Performance and More Algorithms (Xinchao)
  - Performance Issues
  - K-means Clustering
  - Neural Networks

Fundamental ML Algorithms: Optimization, Gradient Descent

Module III Contents

- Overfitting, underfitting and model complexity
- Regularization
- Bias-variance trade-off
- Loss function
- Optimization
- Gradient descent
- Decision trees
- Random forest

Review

repeated

- Supervised learning: given feature(s) x, we want to predict target y
- Most supervised learning algorithms can be formulated as the following optimization problem

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, \mathbf{Data\text{-}Loss}(\mathbf{w}) + \lambda \mathbf{Regularization}(\mathbf{w})$$

- **Data-Loss(w)** quantifies fitting error to training set given parameters **w:** smaller error => better fit to training data
- **Regularization(w)** penalizes more complex models
- For example, in the case of polynomial regression (previous lectures):

$$\underset{\mathbf{w}}{\operatorname{argmin}} \underbrace{(\mathbf{P}\mathbf{w} - \mathbf{y})^T(\mathbf{P}\mathbf{w} - \mathbf{y})}_{\textbf{Data-Loss(w)}} + \underbrace{\lambda \mathbf{w}^T\mathbf{w}}_{\textbf{Reg(w)}}$$

Review

- For polynomial regression (previous lectures)

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}}\, (\mathbf{P}\mathbf{w} - \mathbf{y})^T(\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T\mathbf{w}$$

$$= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} (\mathbf{p}_i^T\mathbf{w} - y_i)^2 + \lambda \mathbf{w}^T\mathbf{w}$$

$\mathbf{p}_i^T\mathbf{w}$ is prediction of i-th training sample

y_i is target of i-th training sample

Review

- For polynomial regression (previous lectures)

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}}\, (\mathbf{P}\mathbf{w} - \mathbf{y})^T(\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T\mathbf{w}$$

$$= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} (\mathbf{p}_i^T\mathbf{w} - y_i)^2 + \lambda \mathbf{w}^T\mathbf{w}$$

- Linear regression with 2 features, $\mathbf{p}_i = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}_i$ ← Bias/Offset; ← Feature 1 of i-th sample; ← Feature 2 of i-th sample

- Quadratic regression with 1 feature, $\mathbf{p}_i = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}_i$ ← Bias/Offset; x is feature of i-th sample

Loss Function & Learning Model

- For polynomial regression (previous lectures)

$$\begin{aligned}\underset{\mathbf{w}}{\operatorname{argmin}} C(\mathbf{w}) &= \underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w}-\mathbf{y})^T(\mathbf{P}\mathbf{w}-\mathbf{y}) + \lambda \mathbf{w}^T\mathbf{w} \\ &= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} (\mathbf{p}_i^T\mathbf{w} - y_i)^2 + \lambda \mathbf{w}^T\mathbf{w}\end{aligned}$$

- Let $f(\mathbf{x}_i, \mathbf{w})$ be the prediction of target y_i from features $\mathbf{x}_i$ for i-th training sample. For example, suppose $f(\mathbf{x}_i, \mathbf{w}) = \mathbf{p}_i^T\mathbf{w}$, then above becomes

$$\underset{\mathbf{w}}{\operatorname{argmin}} C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2 + \lambda \mathbf{w}^T\mathbf{w}$$

- Let $L(f(\mathbf{x}_i, \mathbf{w}), y_i)$ be the penalty for predicting $f(\mathbf{x}_i, \mathbf{w})$ when true value is y_i. For example, suppose $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$, then above becomes

$$\underset{\mathbf{w}}{\operatorname{argmin}} C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda \mathbf{w}^T\mathbf{w}$$

Loss Function & Learning Model

- From previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda \mathbf{w}^T \mathbf{w}$$

- To make it even more general, we can write

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda R(\mathbf{w})$$

Cost Function Loss Function Learning Model Regularization

how to write cost function, loss function, learning model, and regularisation into one statement

Building Blocks of ML algorithms

- From previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda \mathbf{w}^T \mathbf{w}$$

- To make it even more general, we can write

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda R(\mathbf{w})$$

- Learning model f reflects our belief about the relationship between the features $\mathbf{x}_i$ & target y_i
- Loss function L is the penalty for predicting $f(\mathbf{x}_i, \mathbf{w})$ when the true value is y_i
- Regularization R encourages less complex models
- Cost function C is the final optimization criterion we want to minimize
- Optimization routine to find solution to cost function

Motivation for Gradient Descent

- Different learning function f, loss function L & regularization R give rise to different learning algorithms
- In polynomial regression (previous lectures), optimal $\mathbf{w}$ can be written with the following "closed-form" formula (primal solution):

$$\hat{\mathbf{w}} = (\mathbf{P}_{train}^T \mathbf{P}_{train} + \lambda \mathbf{I})^{-1} \mathbf{P}_{train}^T \mathbf{y}_{train}$$

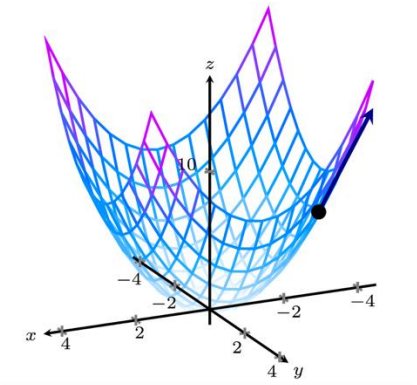
- For other learning function f, loss function L & regularization R, optimizing $C(\mathbf{w})$ might not be so easy
- Usually have to estimate $\mathbf{w}$ iteratively with some algorithm
- Optimization workhorse for modern machine learning is gradient descent

minimisation by finding global minimum of a convex problem

Gradient Descent Algorithm

- Suppose we want to minimize $C(\mathbf{w})$ with respect to $\mathbf{w} = [w_1, \cdots, w_d]^T$
- Gradient $\nabla_{\mathbf{w}} C(\mathbf{w}) = \begin{pmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_d} \end{pmatrix}$

The product of the learning rate and the gradient, which determines the direction and step size to move towards minimizing the cost.



  - $\nabla_{\mathbf{w}} C(\mathbf{w})$ is vector & function of $\mathbf{w}$
  - $\nabla_{\mathbf{w}} C(\mathbf{w})$ is direction at $\mathbf{w}$ where C is increasing most rapidly, so $-\nabla_{\mathbf{w}} C(\mathbf{w})$ is direction at $\mathbf{w}$ where C is decreasing most rapidly

- Gradient Descent:

Initialize $\mathbf{w}_0$ and learning rate η;
while *true* **do**
  Compute $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \eta \nabla_{\mathbf{w}} C(\mathbf{w}_k)$
  if *converge* **then**
    return $\mathbf{w}_{k+1}$
  end
end

According to multi-variable calculus, if eta is not too big, then $C(\mathbf{w}_{k+1}) < C(\mathbf{w}_k)$ => we get better $\mathbf{w}$ after each iteration

Gradient Descent Algorithm

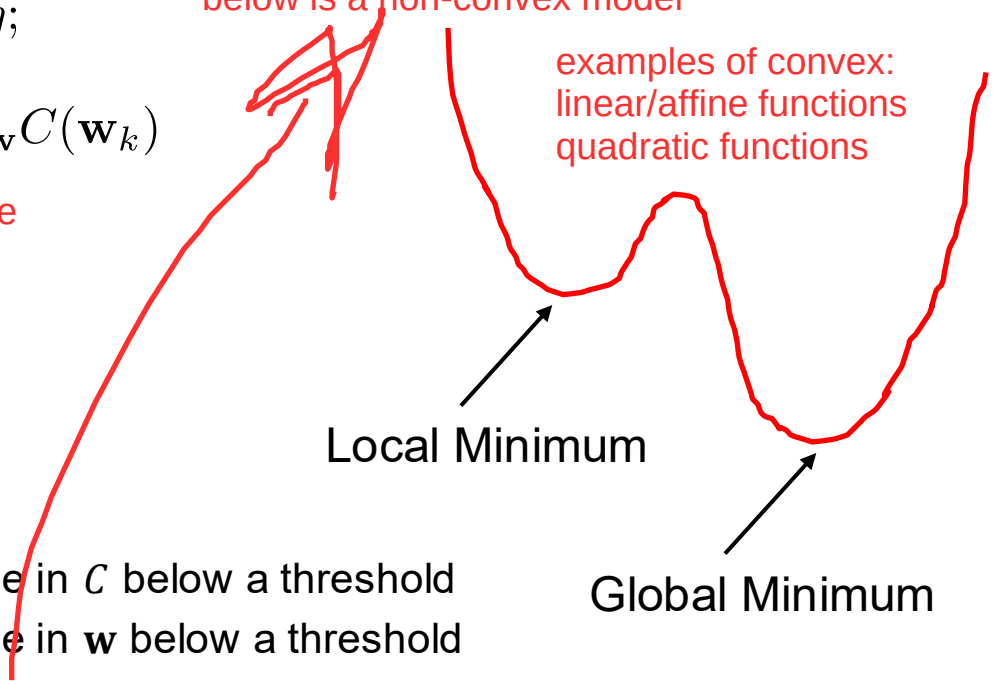
- Gradient Descent:

Initialize $\mathbf{w}_0$ and learning rate η;
while *true* **do**
  Compute $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \eta \nabla_{\mathbf{w}} C(\mathbf{w}_k)$
  if *converge* **then**
    return $\mathbf{w}_{k+1}$
  end
end

this is the descent

unless our minimisation problem is a convex function. For a convex function, any local minimum is the global minimum. The graph below is a non-convex model

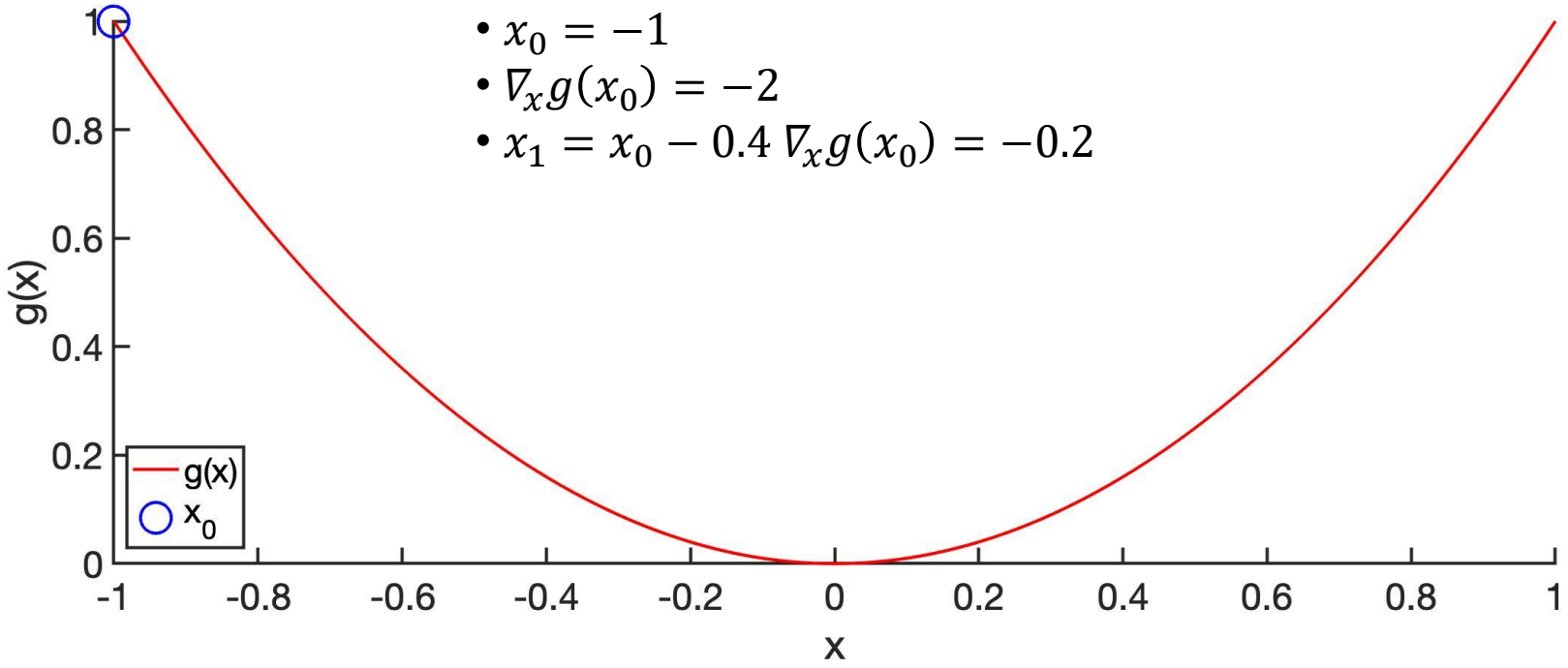
examples of convex:
linear/affine functions
quadratic functions



- Possible convergence criteria
  - Set maximum iteration k
  - Check percentage or absolute change in C below a threshold
  - Check percentage or absolute change in $\mathbf{w}$ below a threshold
- Gradient descent can only find local minimum
  - Because gradient = 0 at local minimum, so $\mathbf{w}$ won't change after that
- Many variations of gradient descent, e.g., change how gradient is computed or learning rate η decreases with increasing k

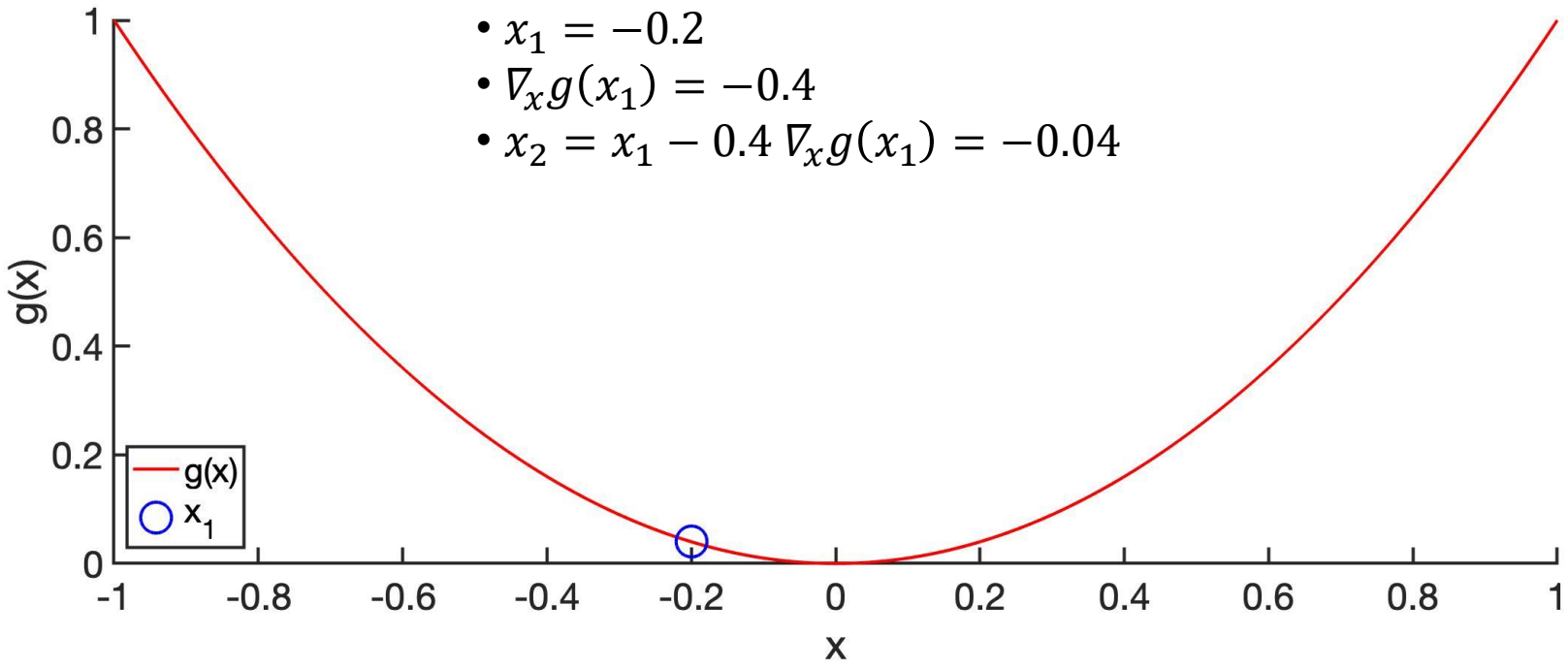
Find x to minimize $g(x) = x^2$

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



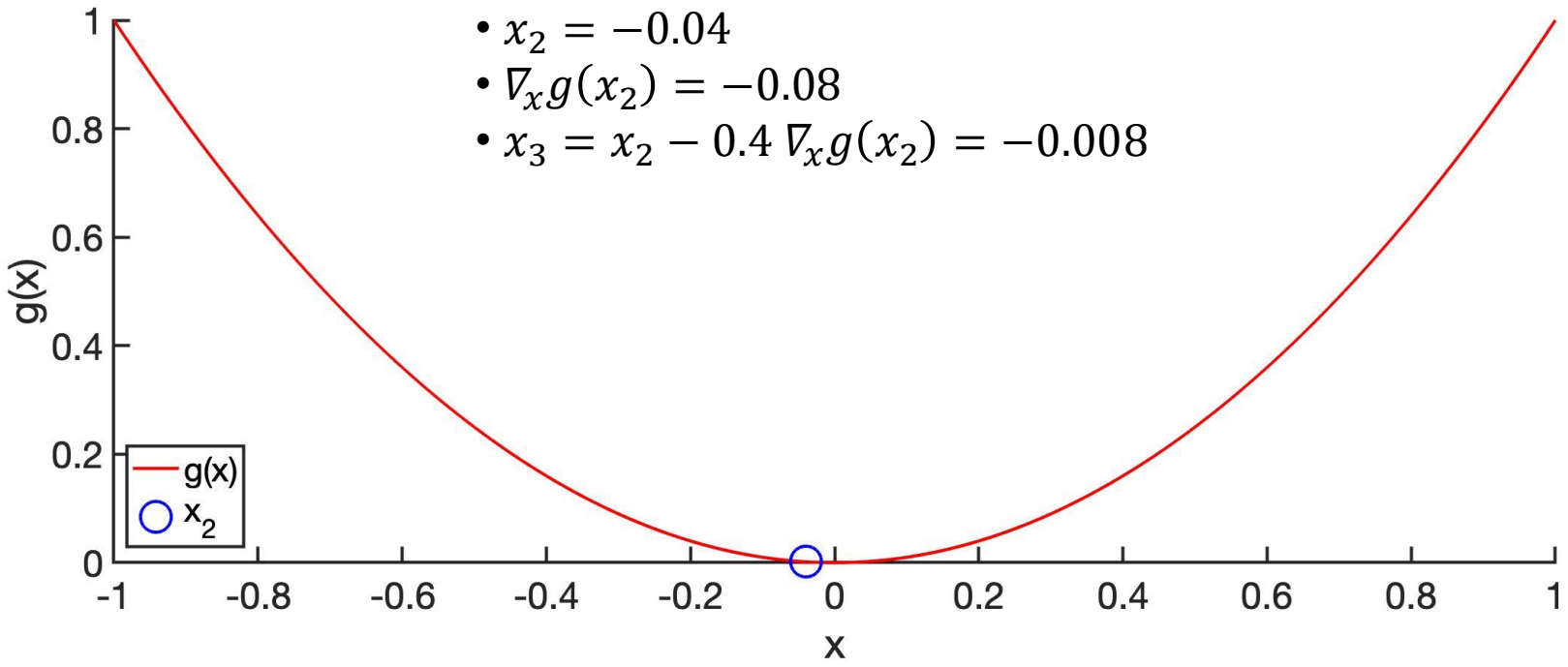
Find x to minimize $g(x) = x^2$

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



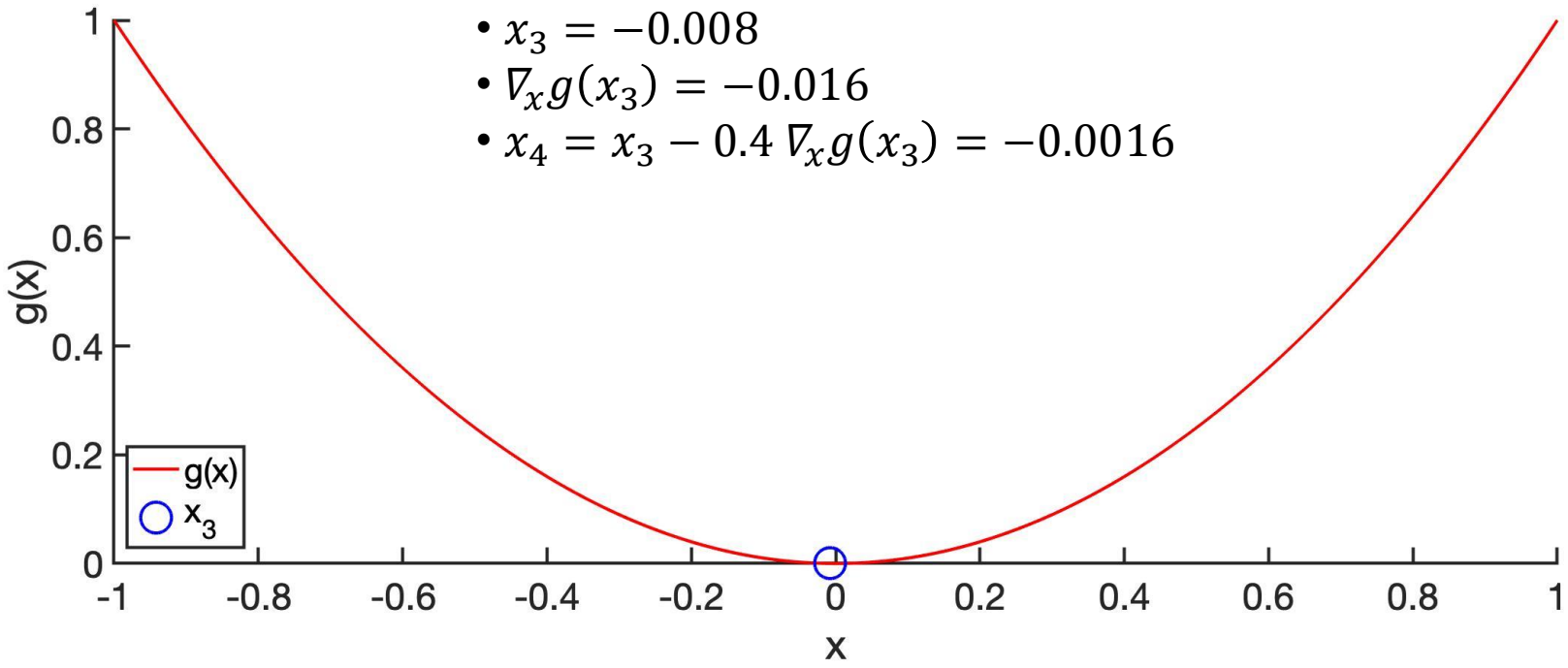
Find x to minimize $g(x) = x^2$

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



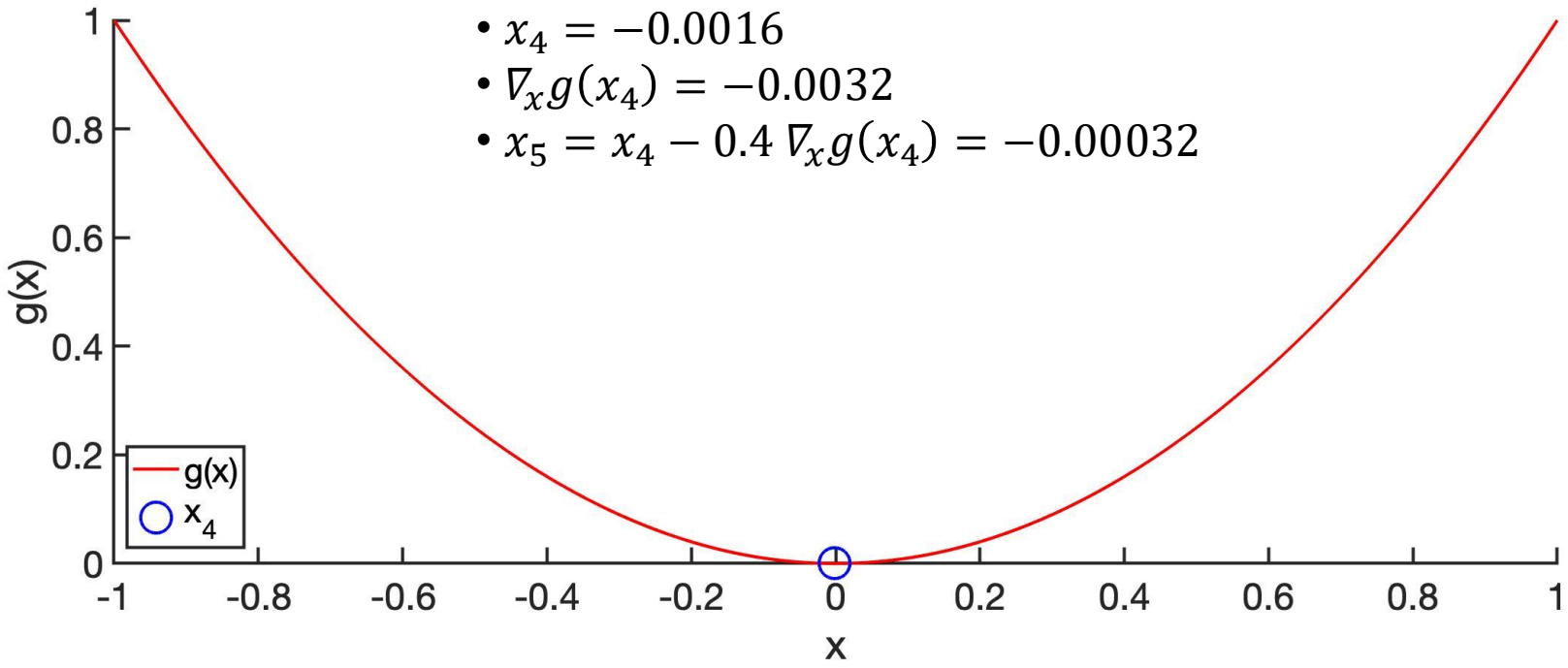
Find x to minimize $g(x) = x^2$

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



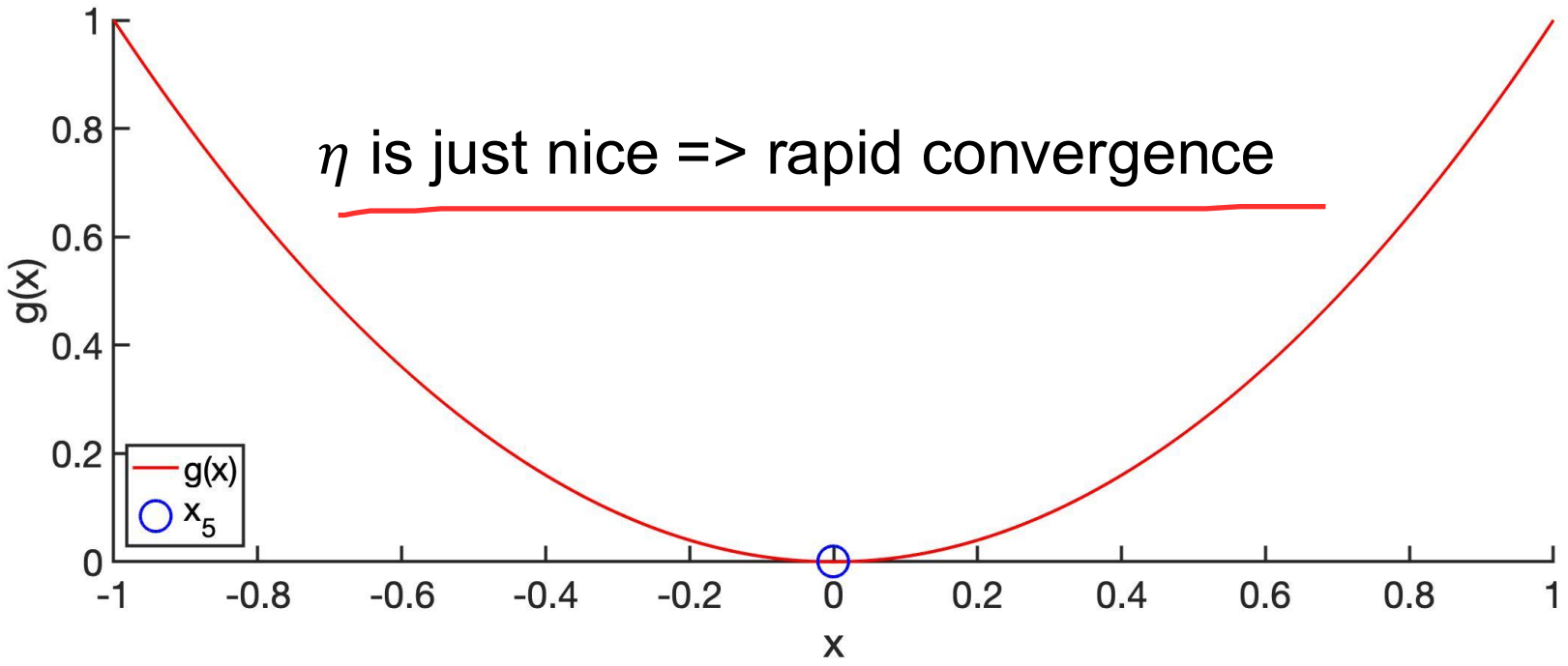
Find x to minimize $g(x) = x^2$

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



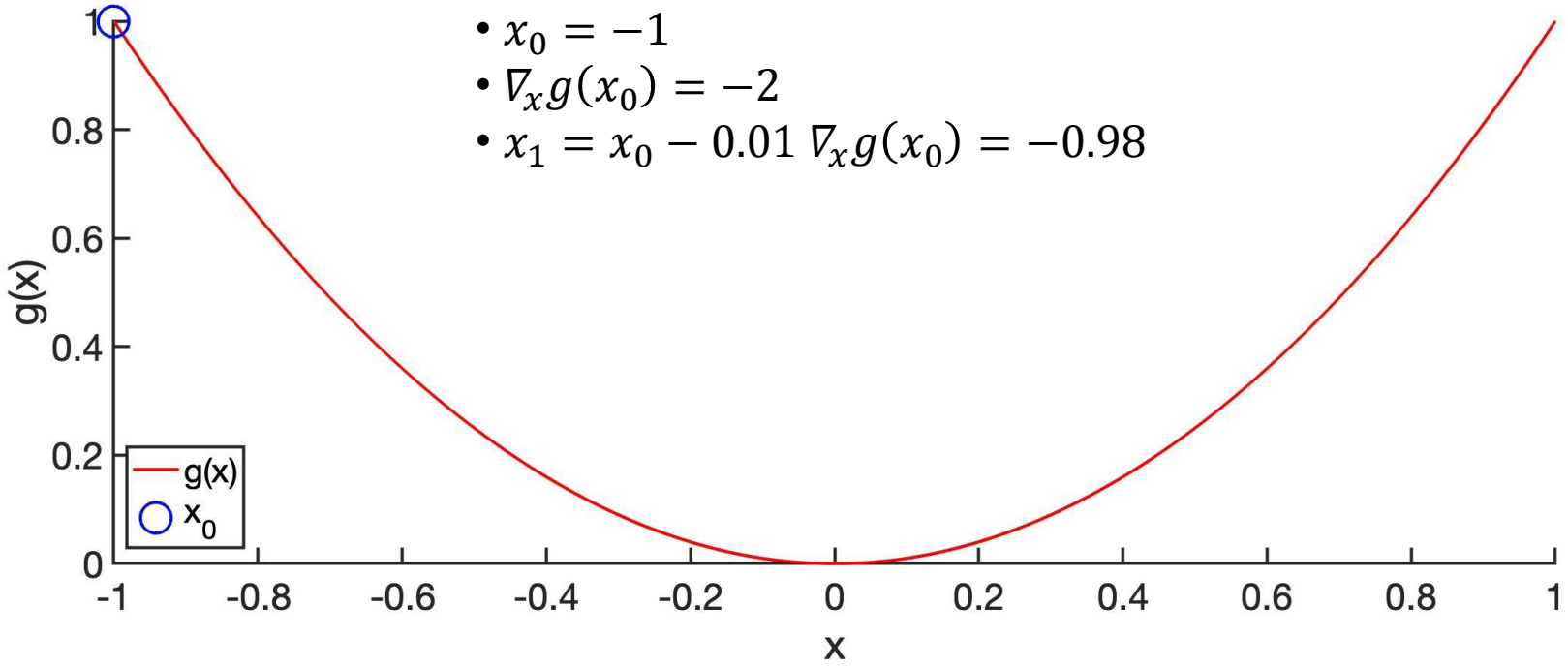
Find x to minimize $g(x) = x^2$

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.4$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



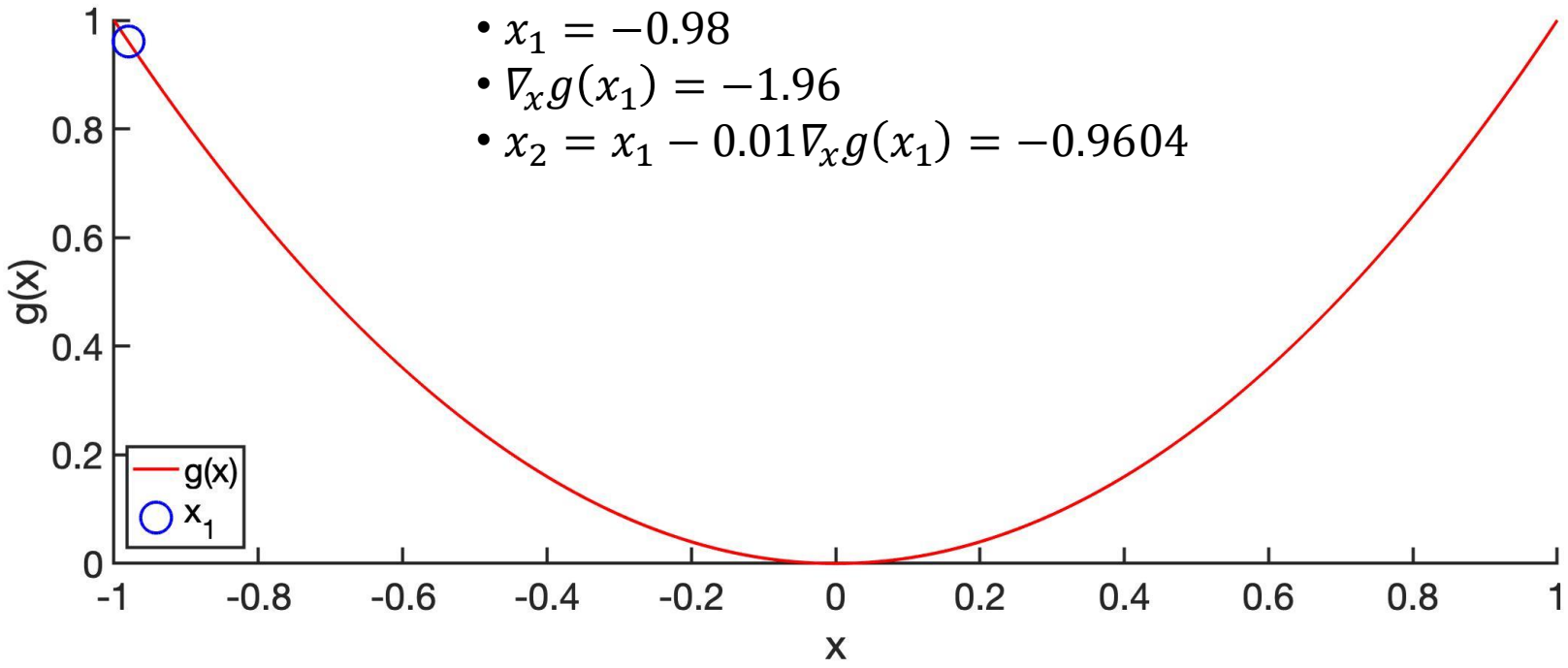
What if learning rate η is too small?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.01$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



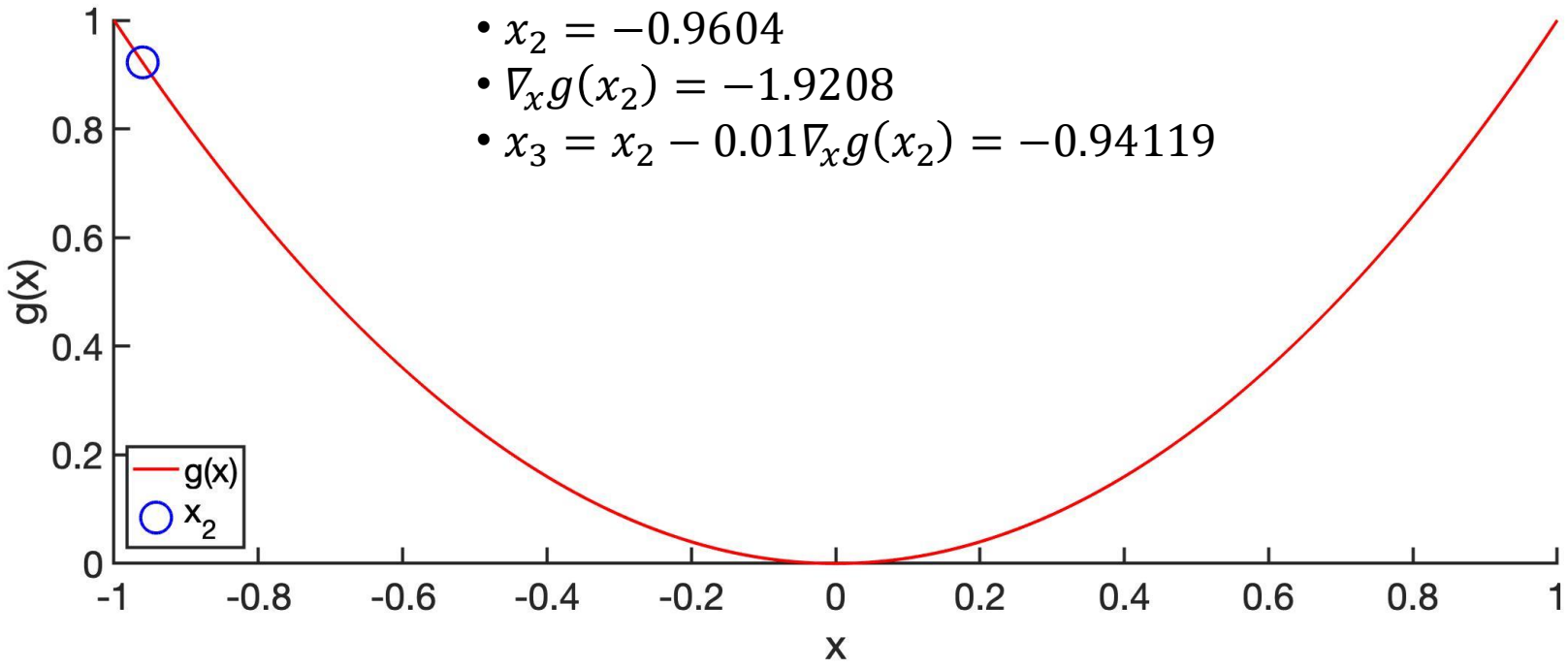
What if learning rate η is too small?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.01$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



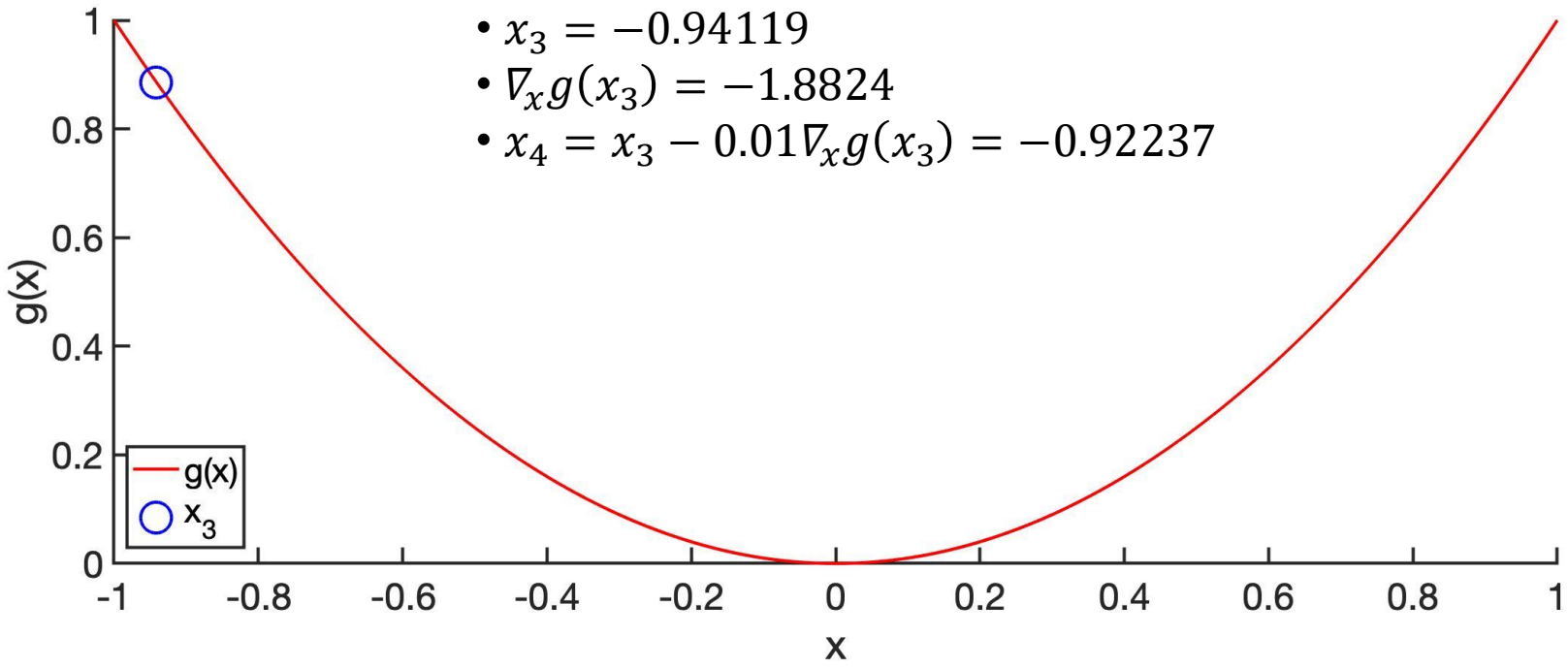
What if learning rate η is too small?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.01$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



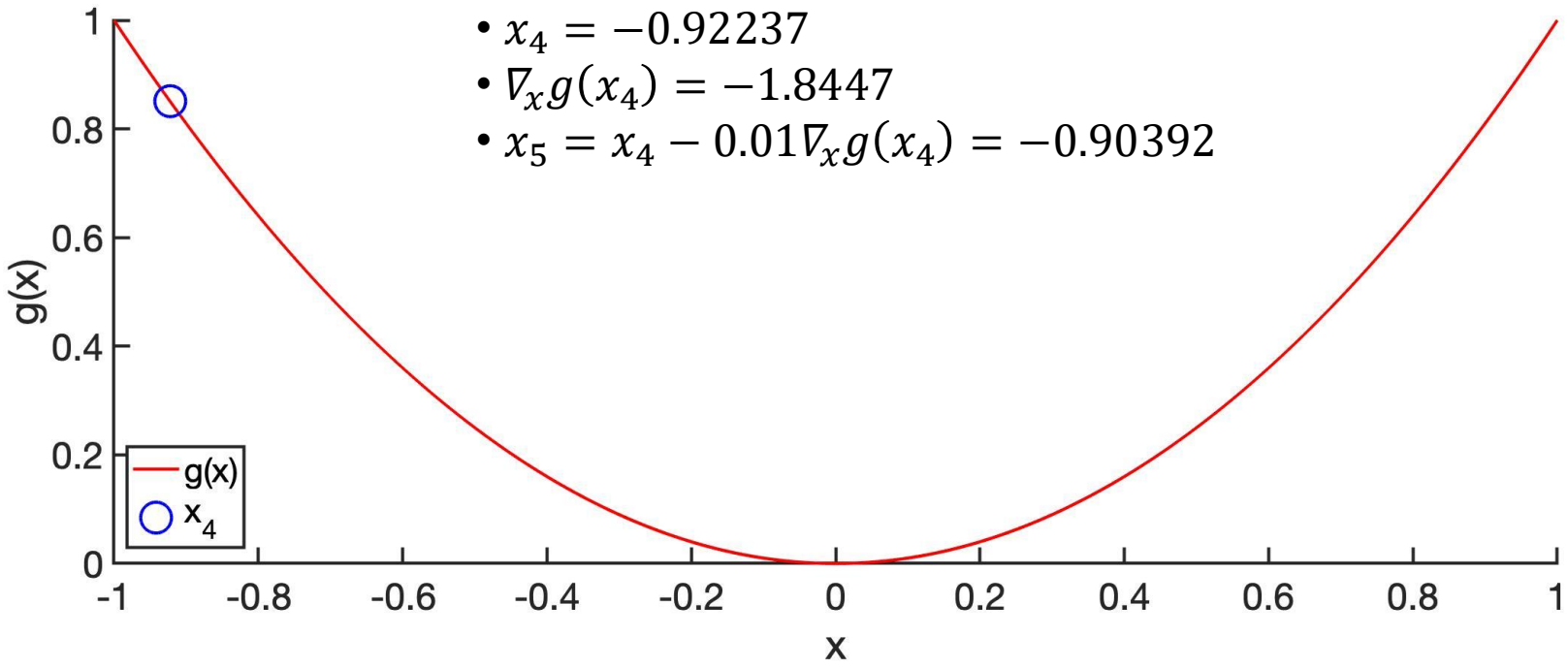
What if learning rate η is too small?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.01$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



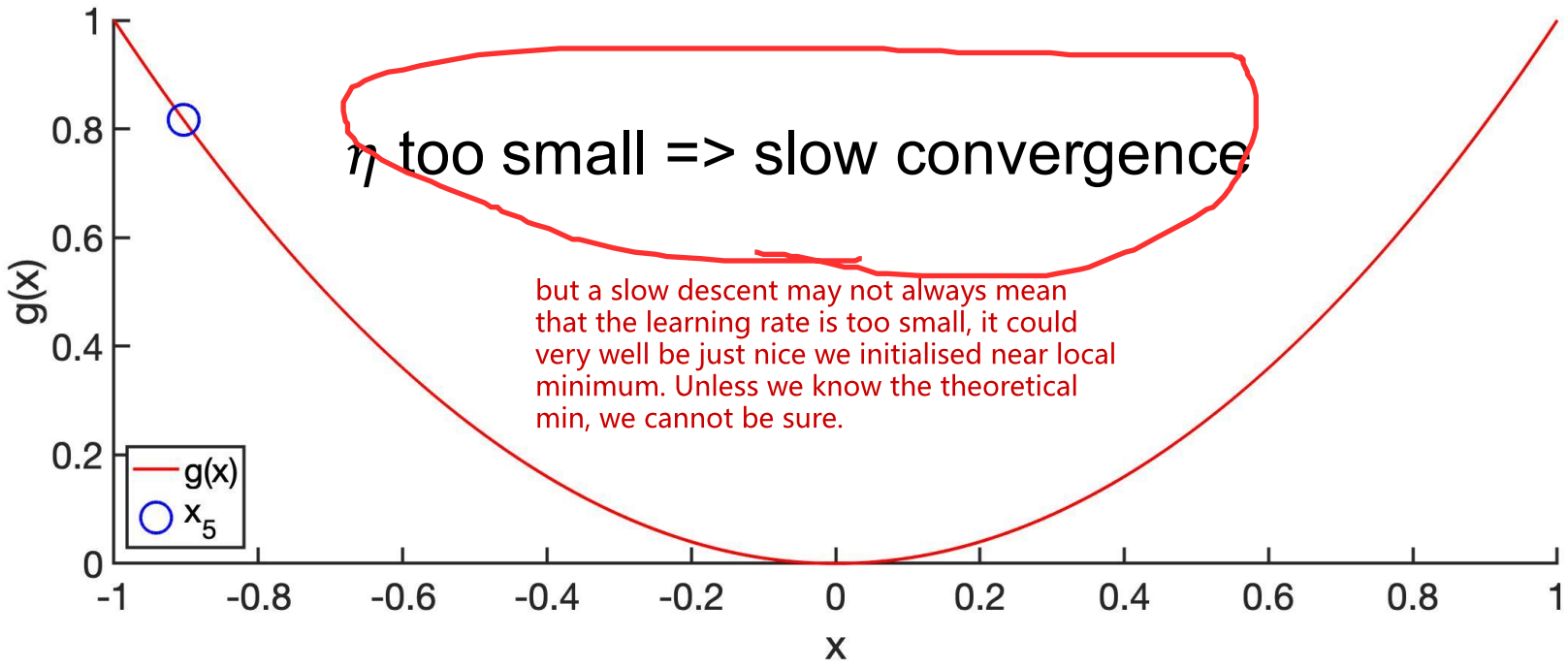
What if learning rate η is too small?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.01$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



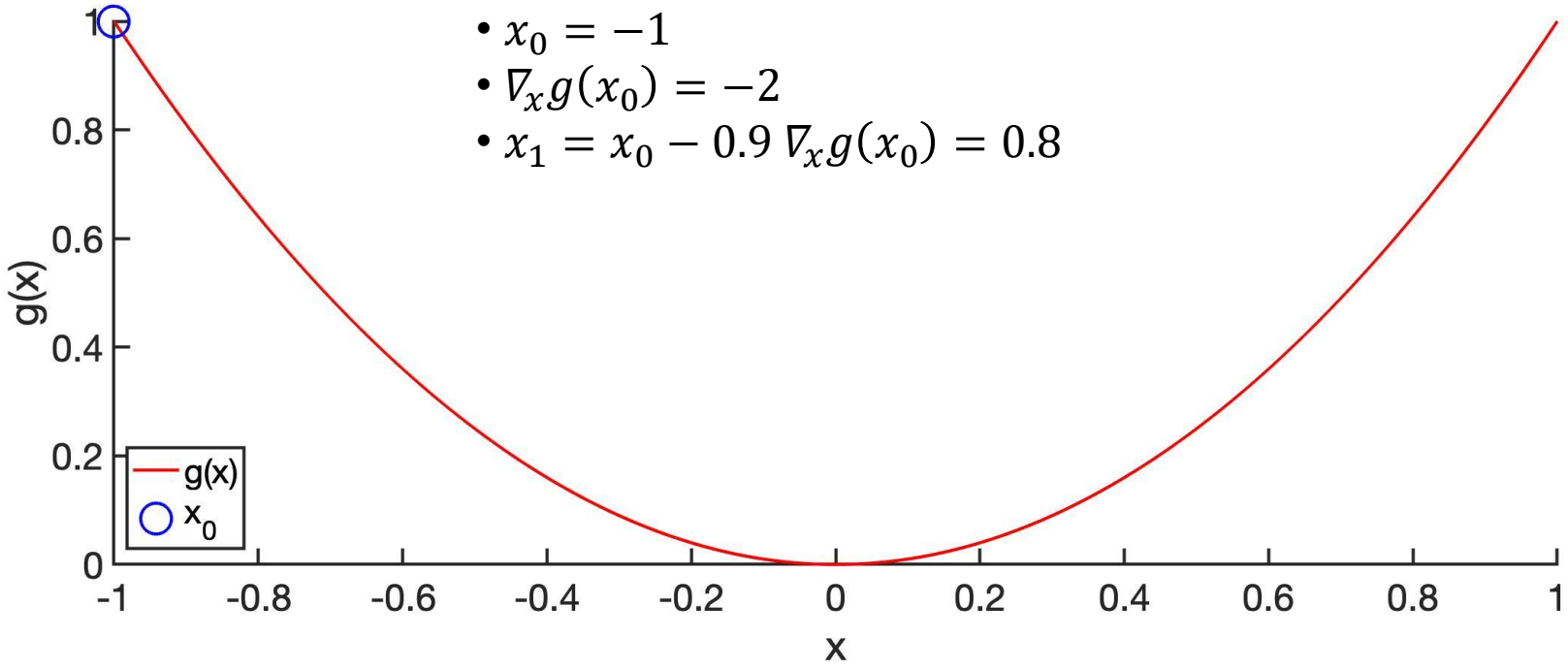
What if learning rate η is too small?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.01$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



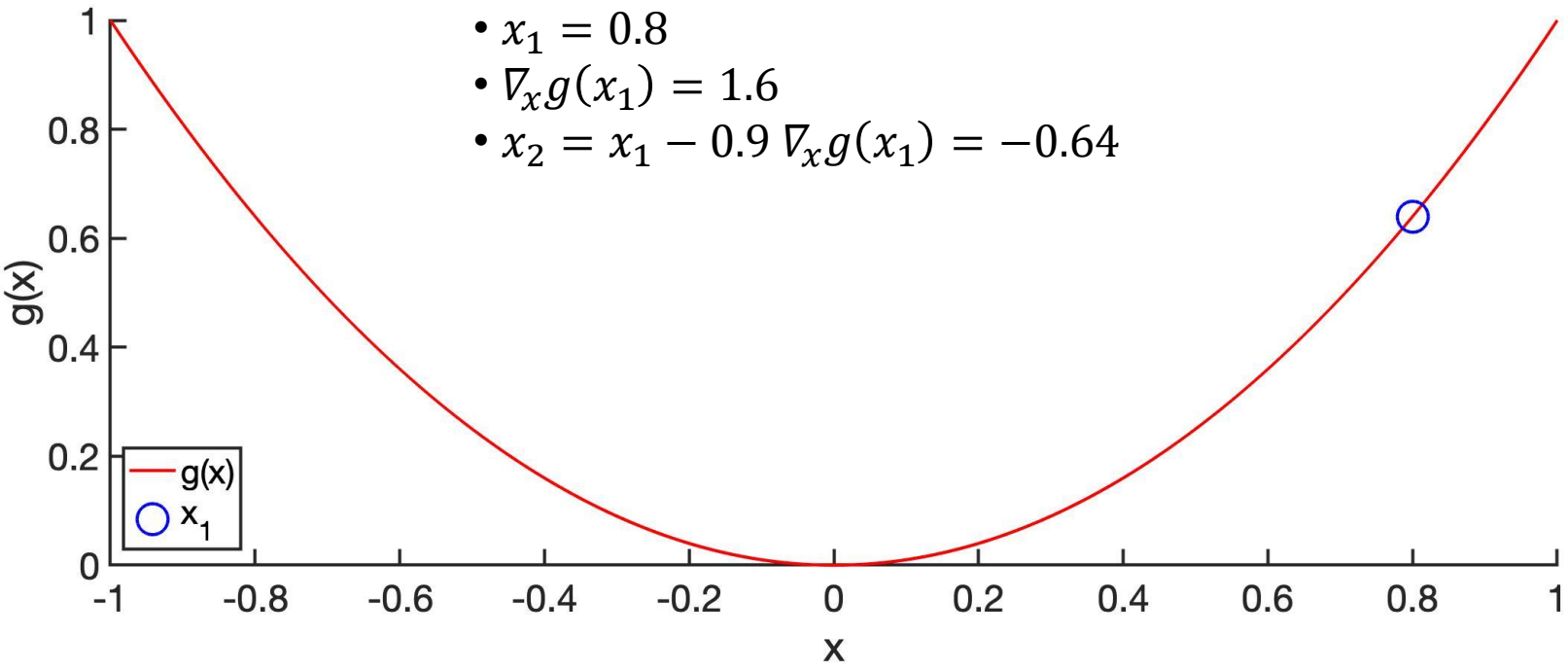
What if learning rate η is too big?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.9$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



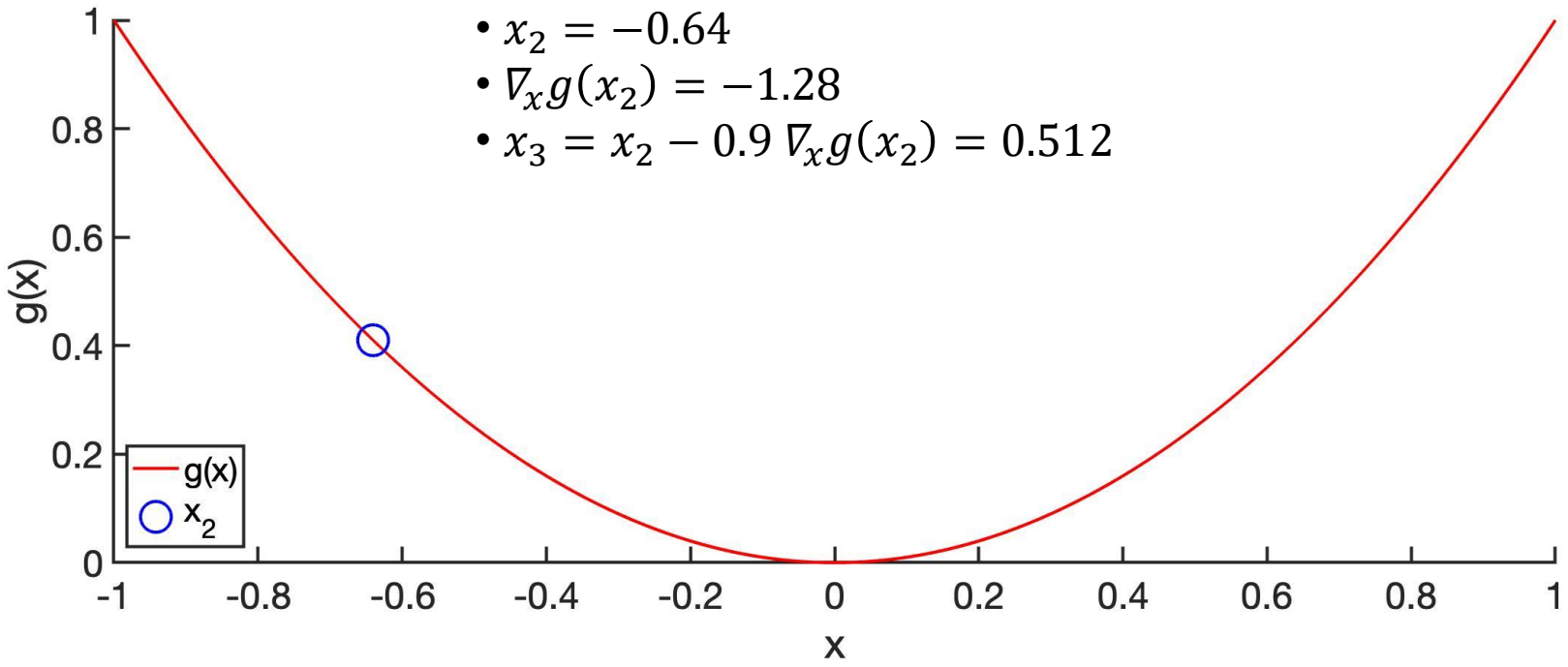
What if learning rate η is too big?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.9$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



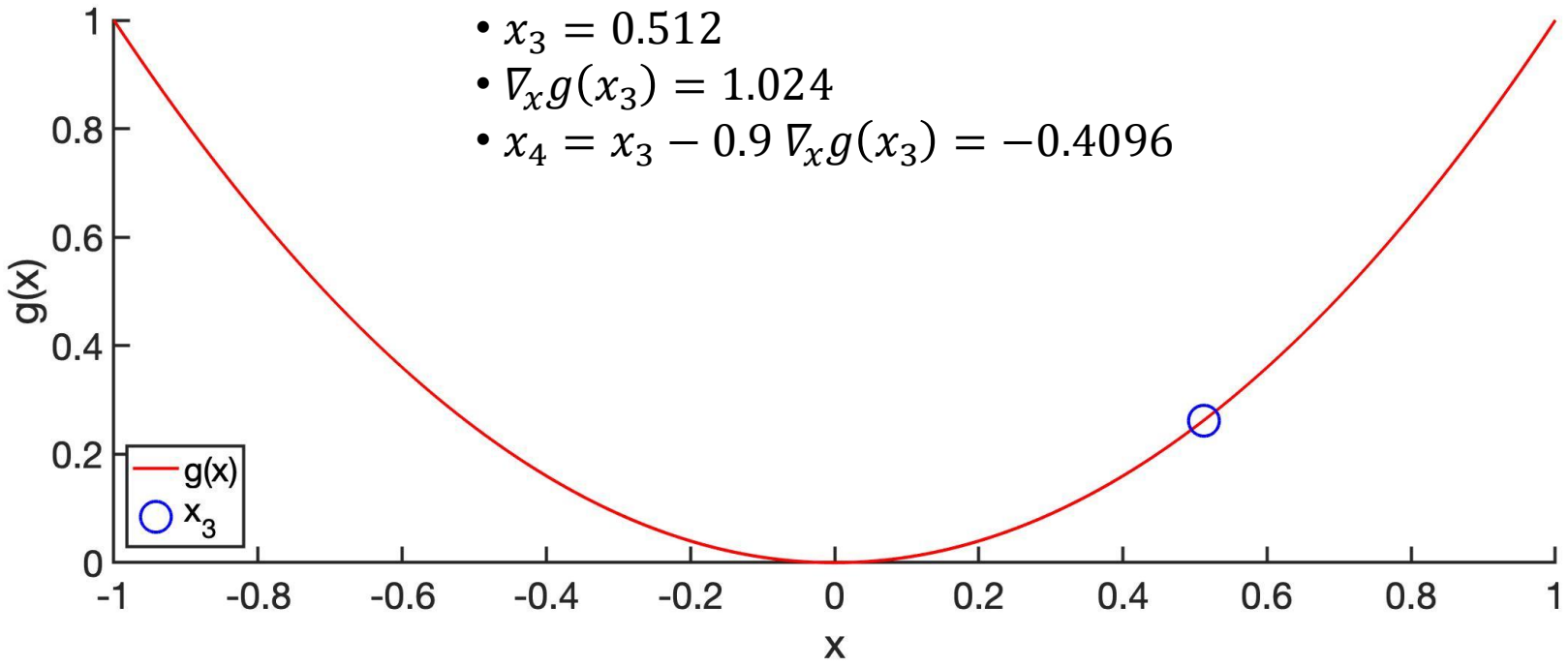
What if learning rate η is too big?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.9$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



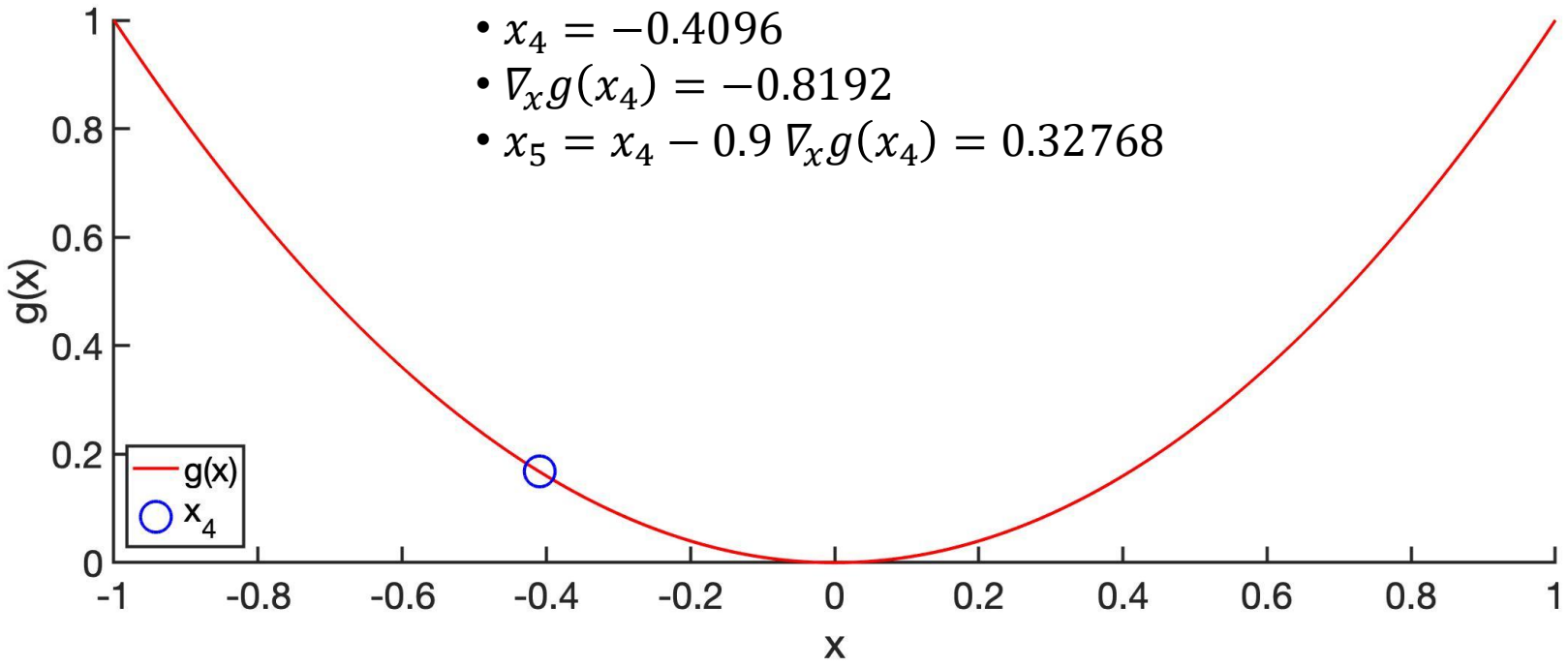
What if learning rate η is too big?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.9$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



What if learning rate η is too big?

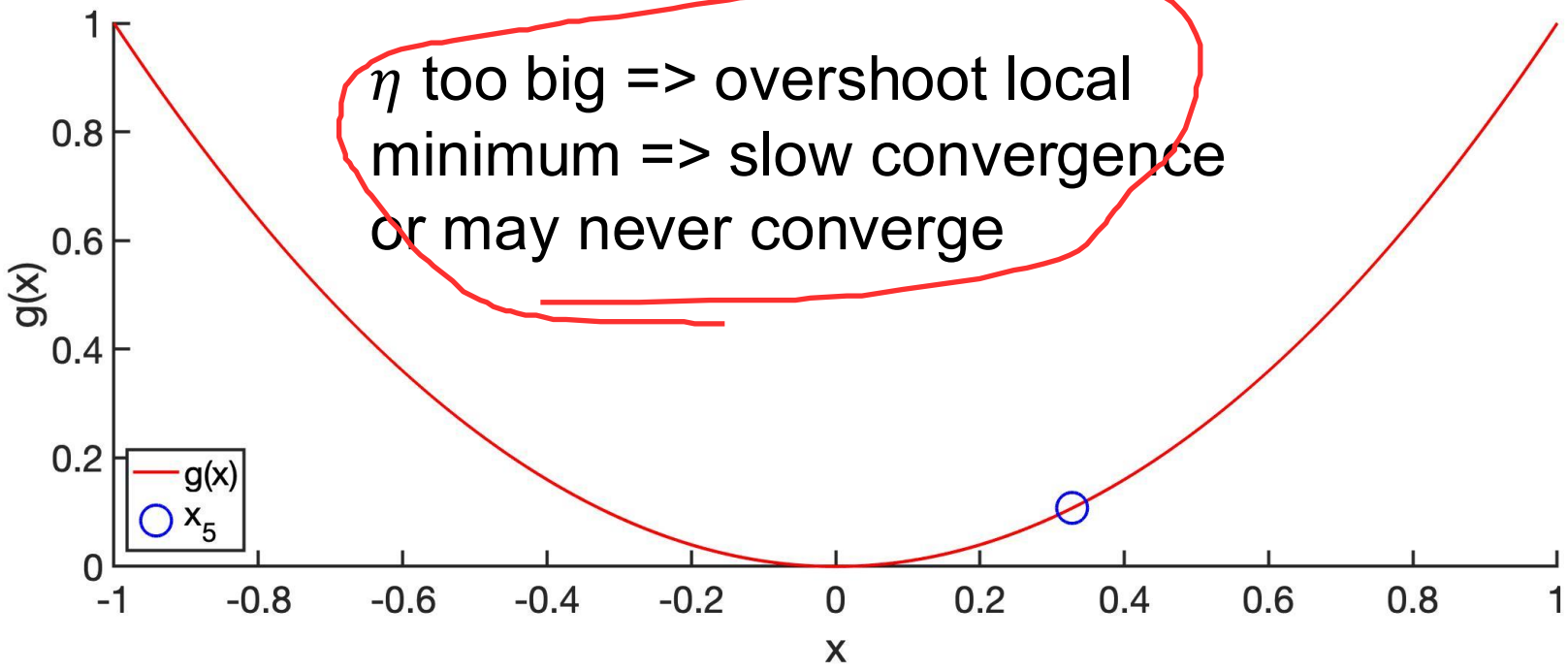
- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.9$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



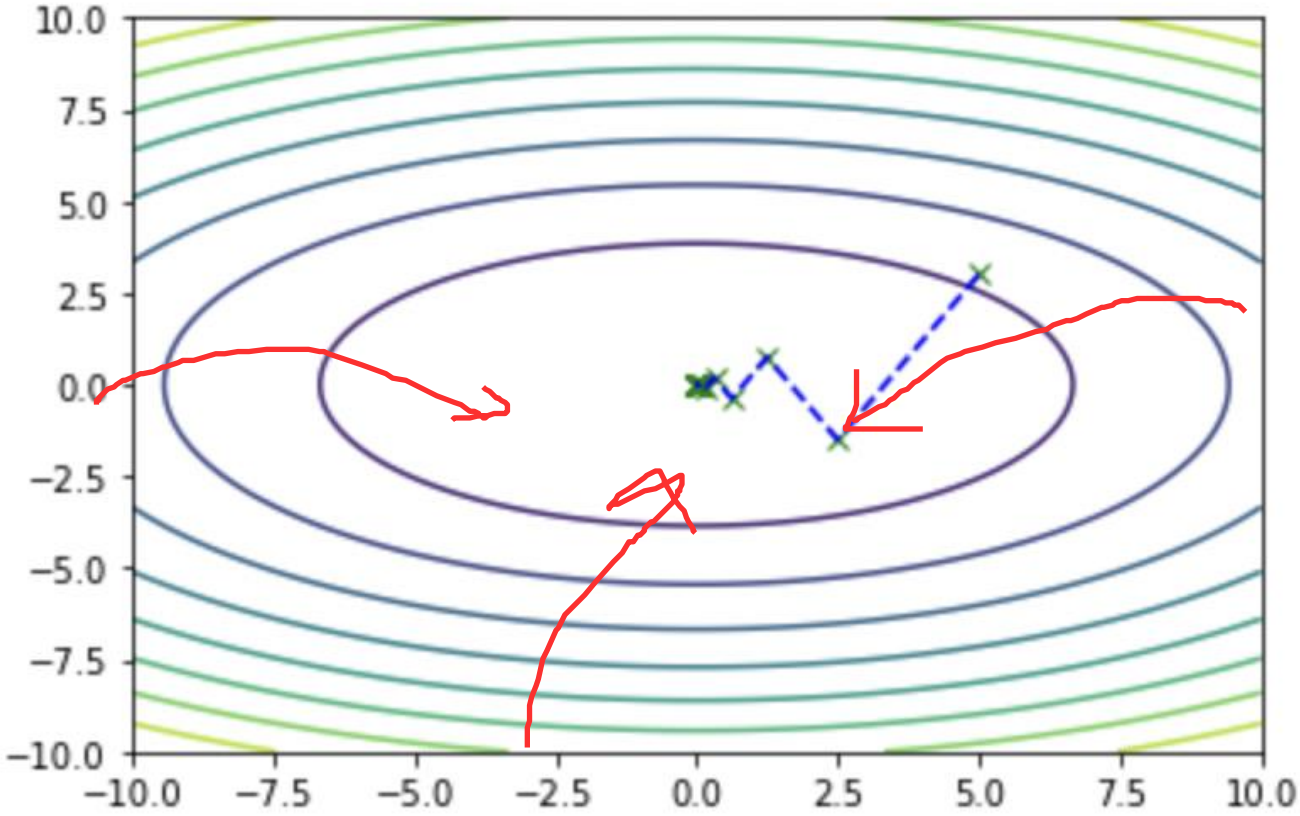
What if learning rate η is too big?

- Obviously minimum corresponds to $x = 0$, but let's do gradient descent
  - Gradient $\nabla_x g(x) = 2x$
  - Initialize $x_0 = -1$, learning rate $\eta = 0.9$
  - At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

Python demo



Gradient descent: quadratic function



Convergence to the foot of the valley

Different Learning Models

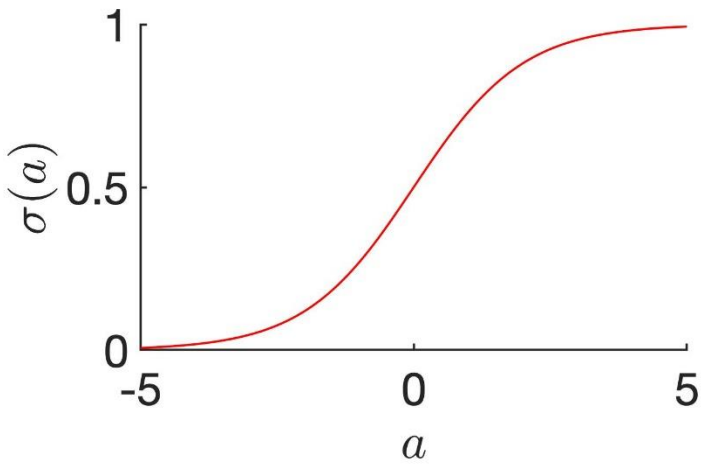
- Different learning models $f(\mathbf{x}_i, \mathbf{w})$ reflect our beliefs about the relationship between the features $\mathbf{x}_i$ and target y_i
  - For example, $f(\mathbf{x}_i, \mathbf{w}) = \mathbf{p}_i^T \mathbf{w}$ assumes polynomial relationship between features and target
- Suppose we are performing classification (rather than regression), so y_i is class -1 or class 1
  - $\mathbf{p}_i^T \mathbf{w}$ is number between $-\infty$ to ∞.
  - Can use sigmoid function to map $\mathbf{p}_i^T \mathbf{w}$ to between 0 and 1:

$$f(\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{p}_i^T \mathbf{w})$$

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

$a = \mathbf{p}^T \mathbf{w}$

now value range is (0,1) where we evaluate based on probability.
this is also known as (binomial) logistic regression



Different Learning Models

- Different learning models $f(\mathbf{x}_i, \mathbf{w})$ reflect our beliefs about the relationship between the features $\mathbf{x}_i$ and target y_i
  - For example, $f(\mathbf{x}_i, \mathbf{w}) = \mathbf{p}_i^T \mathbf{w}$ assumes polynomial relationship between features and target
- Suppose we are performing classification (rather than regression), so y_i is class -1 or class 1
  - $\mathbf{p}_i^T \mathbf{w}$ is number between $-\infty$ to ∞.
  - Can use sigmoid function to map $\mathbf{p}_i^T \mathbf{w}$ to between 0 and 1:

$$f(\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{p}_i^T \mathbf{w})$$

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

this is binary logistic regression

  - If $f(\mathbf{x}_i, \mathbf{w})$ is closer to 0 (or 1), we predict class -1 (or class 1)
- More generally, in one layer neural network: $f(\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{p}_i^T \mathbf{w})$, where activation function σ can be sigmoid or some other functions & $\mathbf{p}$ is linear

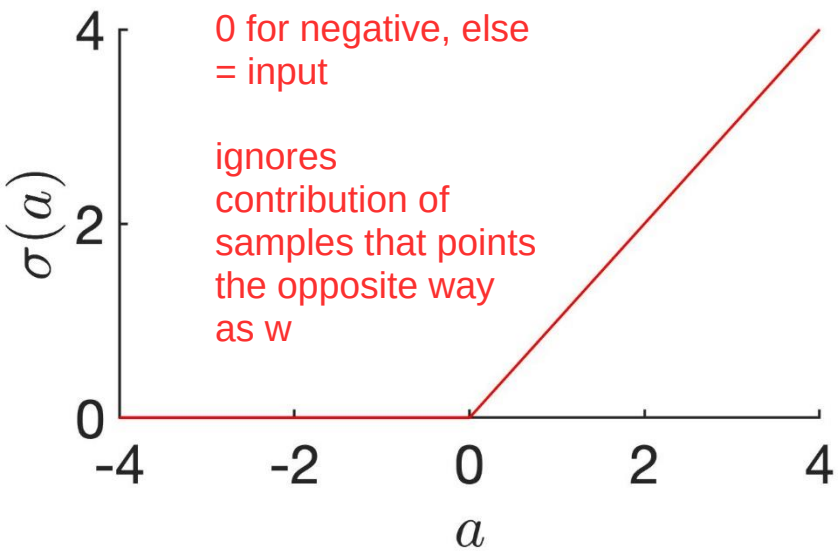
one of the common activation function used in MLP

Different Learning Models

- $f(\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{p}_i^T \mathbf{w})$, where σ can be different functions:
- Rectified linear unit (ReLU): $\sigma(a) = \max(0, a)$
- Exponential: $\sigma(a) = \exp(-a)$

✓ used in MLP

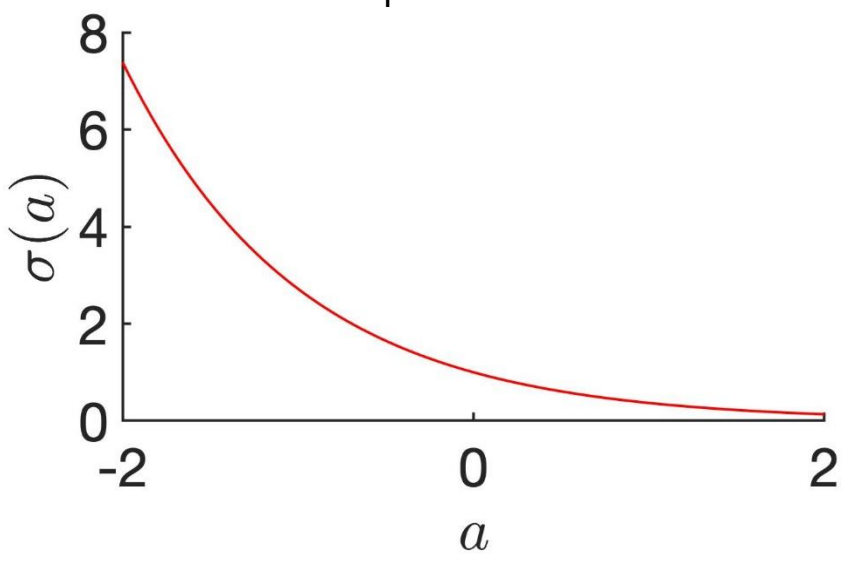
ReLU



0 for negative, else = input

ignores contribution of samples that points the opposite way as w

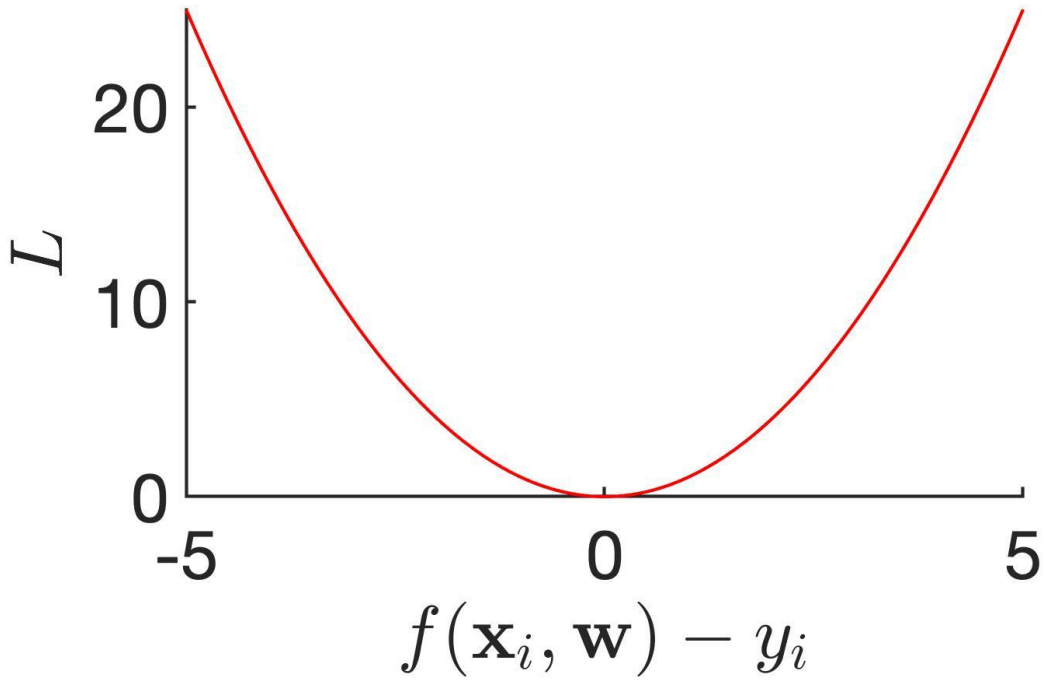
Exponential



Different Loss Functions

- Different loss functions $L(f(\mathbf{x}_i, \mathbf{w}), y_i)$ encodes the penalty when we predict $f(\mathbf{x}_i, \mathbf{w})$ but the true value is y_i
  - $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$ is called the square error loss

example of a loss function



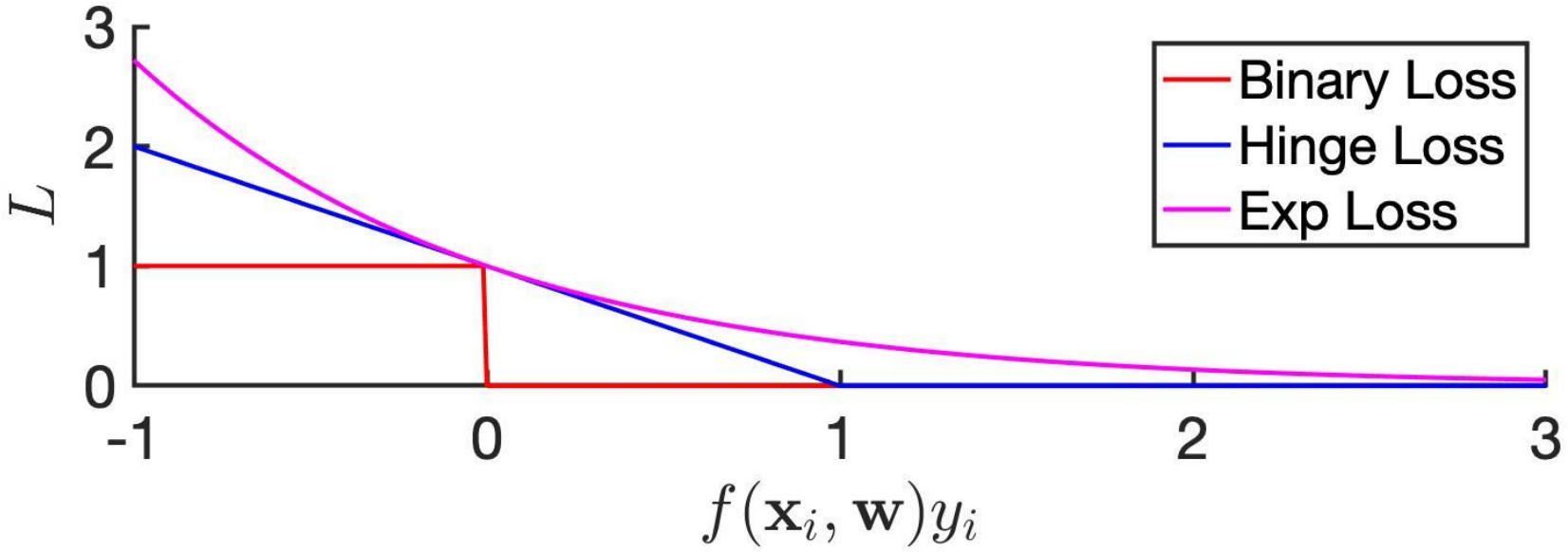
Different Loss Functions

- Different loss functions $L(f(\mathbf{x}_i, \mathbf{w}), y_i)$ encodes the penalty when we predict $f(\mathbf{x}_i, \mathbf{w})$ but the true value is y_i
  - $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$ is called the square error loss
- Suppose we are performing classification (rather than regression), so y_i is class -1 or class 1, then square error loss makes less sense. Instead, we can use
  - Binary loss (or 0–1 loss): $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \begin{cases} 0 & \text{if } f(\mathbf{x}_i, \mathbf{w}) = y_i \\ 1 & \text{if } f(\mathbf{x}_i, \mathbf{w}) \neq y_i \end{cases}$
  - In practice, hard to constrain $f(\mathbf{x}_i, \mathbf{w})$ to be exactly -1 or 1, so we can declare "victory" if $f(\mathbf{x}_i, \mathbf{w})$ & y have the same sign:

$$L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \begin{cases} 0 & \text{if } f(\mathbf{x}_i, \mathbf{w}) y_i > 0 \\ 1 & \text{if } f(\mathbf{x}_i, \mathbf{w}) y_i < 0 \end{cases}$$

Different Loss Functions

- Binary loss, where y_i is class -1 or class 1 & $f(\mathbf{x}_i, \mathbf{w})$ is a number between $-\infty$ and ∞: $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \begin{cases} 0 & \text{if } f(\mathbf{x}_i, \mathbf{w}) y_i > 0 \\ 1 & \text{if } f(\mathbf{x}_i, \mathbf{w}) y_i < 0 \end{cases}$
- Binary loss not differentiable, so two other possibilities
  - Hinge loss: $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \max(0, 1 - f(\mathbf{x}_i, \mathbf{w}) y_i)$
  - Exponential loss: $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \exp(-f(\mathbf{x}_i, \mathbf{w}) y_i)$



Summary

- Building blocks of machine learning algorithms
  - Learning model: reflects our belief about relationship between features & target we want to predict
  - Loss function: penalty for wrong prediction
  - Regularization: penalizes complex models
  - Optimization routine: find minimum of overall cost function
- Gradient descent algorithm
  - At each iteration, compute gradient & update model parameters in direction opposite to gradient
  - If learning rate η is too big => may not converge
  - If learning rate η is too small => converge very slowly
- Different learning models, e.g., linear, polynomial, sigmoid, ReLU, exponential, etc
- Different loss functions, e.g., square error, binary, logistic, etc

problem that trees can fix

EE2211 Introduction to Machine Learning

Lecture 9

Juan Helen Zhou
helen.zhou@nus.edu.sg

Electrical and Computer Engineering Department
National University of Singapore

Acknowledgement: EE2211 development team
Thomas, Helen, Xinchao, Kar-Ann, Chen Khong, Robby and Haizhou

Course Contents

- Introduction and Preliminaries (Xinchao)
  - Introduction
  - Data Engineering
  - Introduction to Probability and Statistics
- Fundamental Machine Learning Algorithms I (Helen)
  - Systems of linear equations
  - Least squares, Linear regression
  - Ridge regression, Polynomial regression
- Fundamental Machine Learning Algorithms II (Helen)
  - Over-fitting, bias/variance trade-off
  - Optimization, Gradient descent
  - Decision Trees, Random Forest
- Performance and More Algorithms (Xinchao)
  - Performance Issues
  - K-means Clustering
  - Neural Networks

Fundamental ML Algorithms: Decision Trees, Random Forest

Module III Contents

- Overfitting, underfitting and model complexity
- Bias-variance trade-off
- Regularization
- Loss function
- Optimization
- Gradient descent
- Decision trees
- Random forest

Review

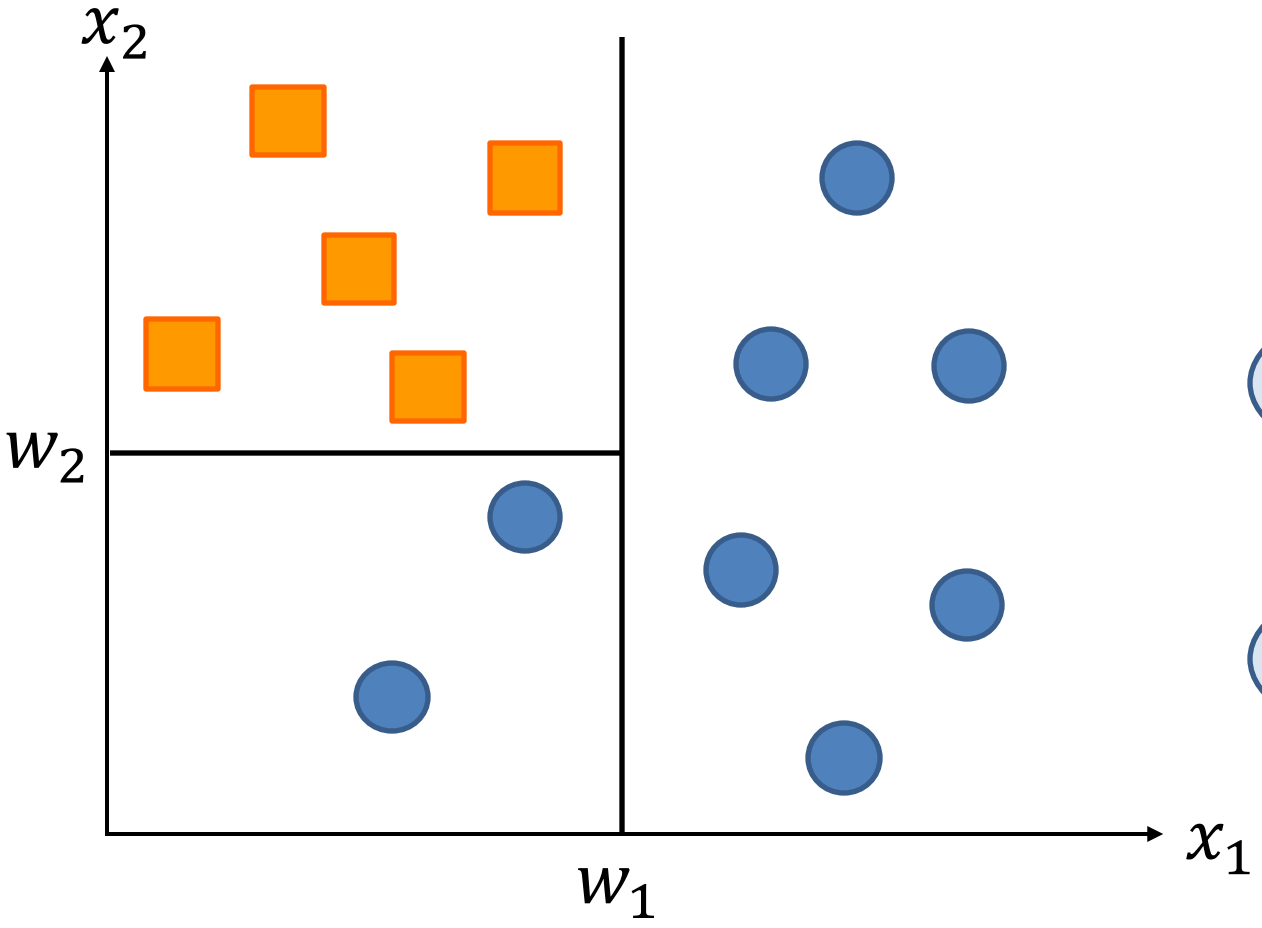
- Supervised learning: given feature(s) x, we want to predict target y to be some $f(x)$
  - If y is continuous, problem is called "regression"
  - If y is discrete, problem is called "classification"
- Previous lectures used linear models for regression (and classification)
  - Nonlinearity added by using polynomial regression or other learning models
- New approach today: trees

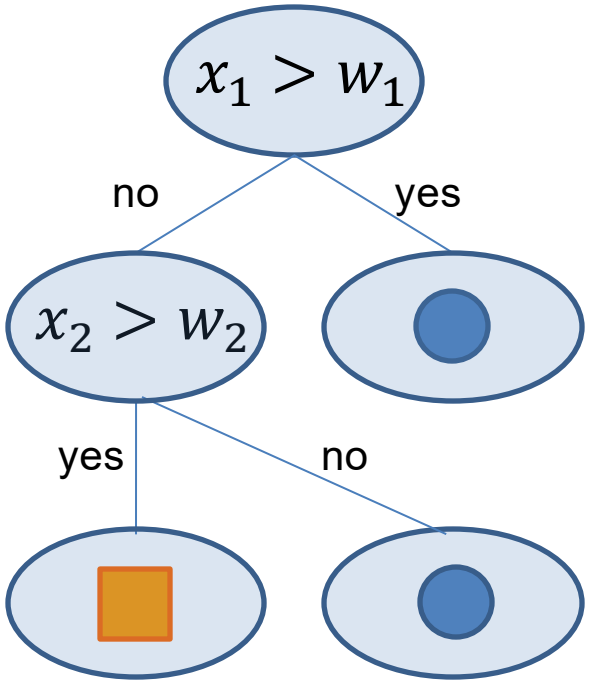
main problem with past learning models is we have an initial assumption about the relationship between the features and the target e.g. linear/ quadratic

Regression Tree seeks to predict continuous variables - regression task)
classification trees seeks to predict discrete variables (or classify things into classes - classification task)

Decision Tree Classification Example

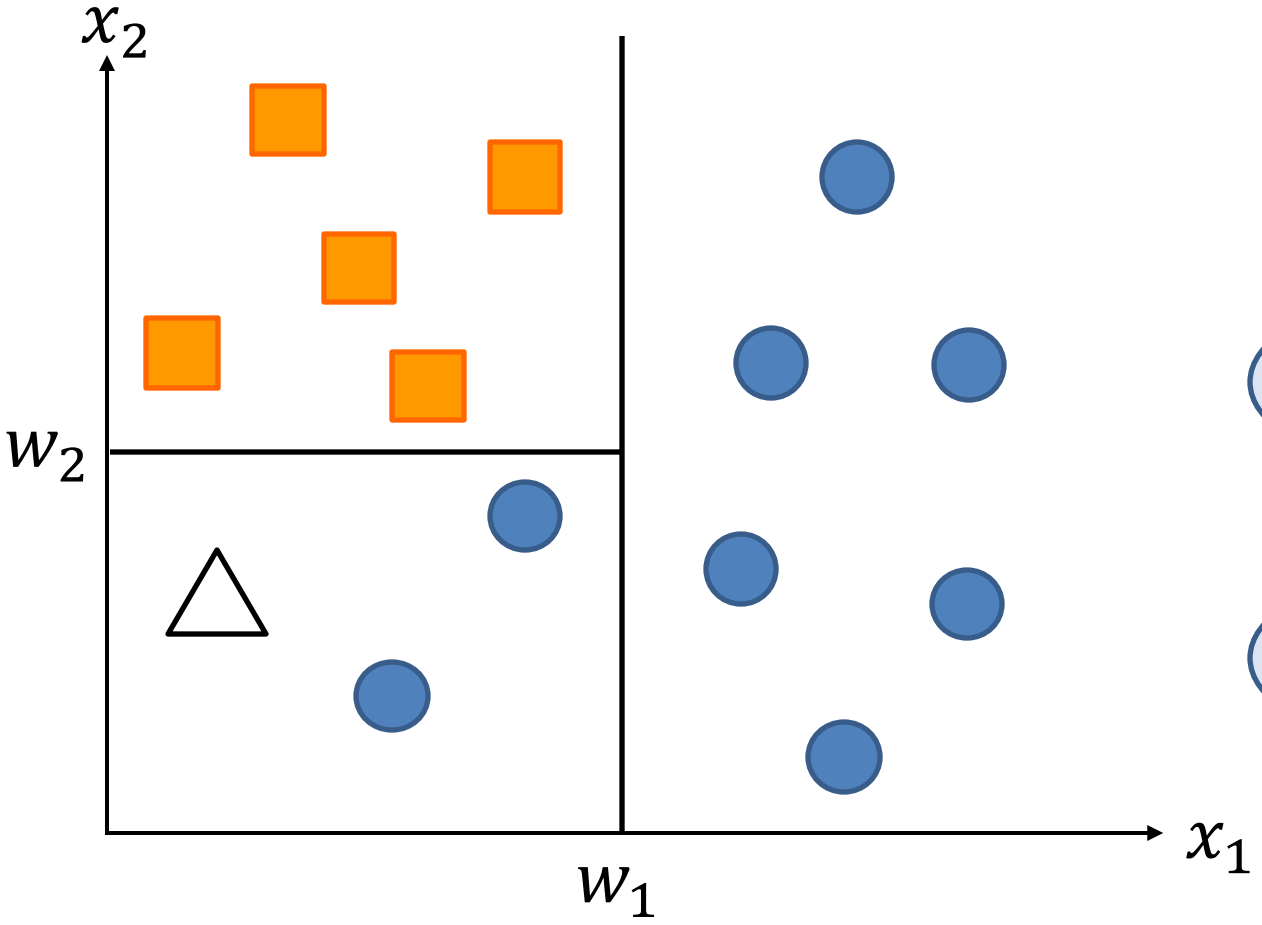
- Goal: predict class labels using two features x_1 & x_2

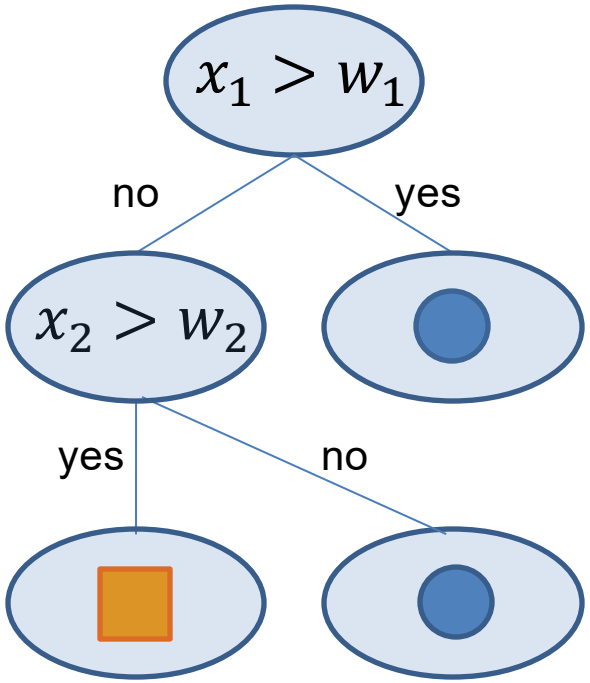




Decision Tree Classification Example

- In our test set, we observe datapoint △ shown below. How would the decision tree classify this point?

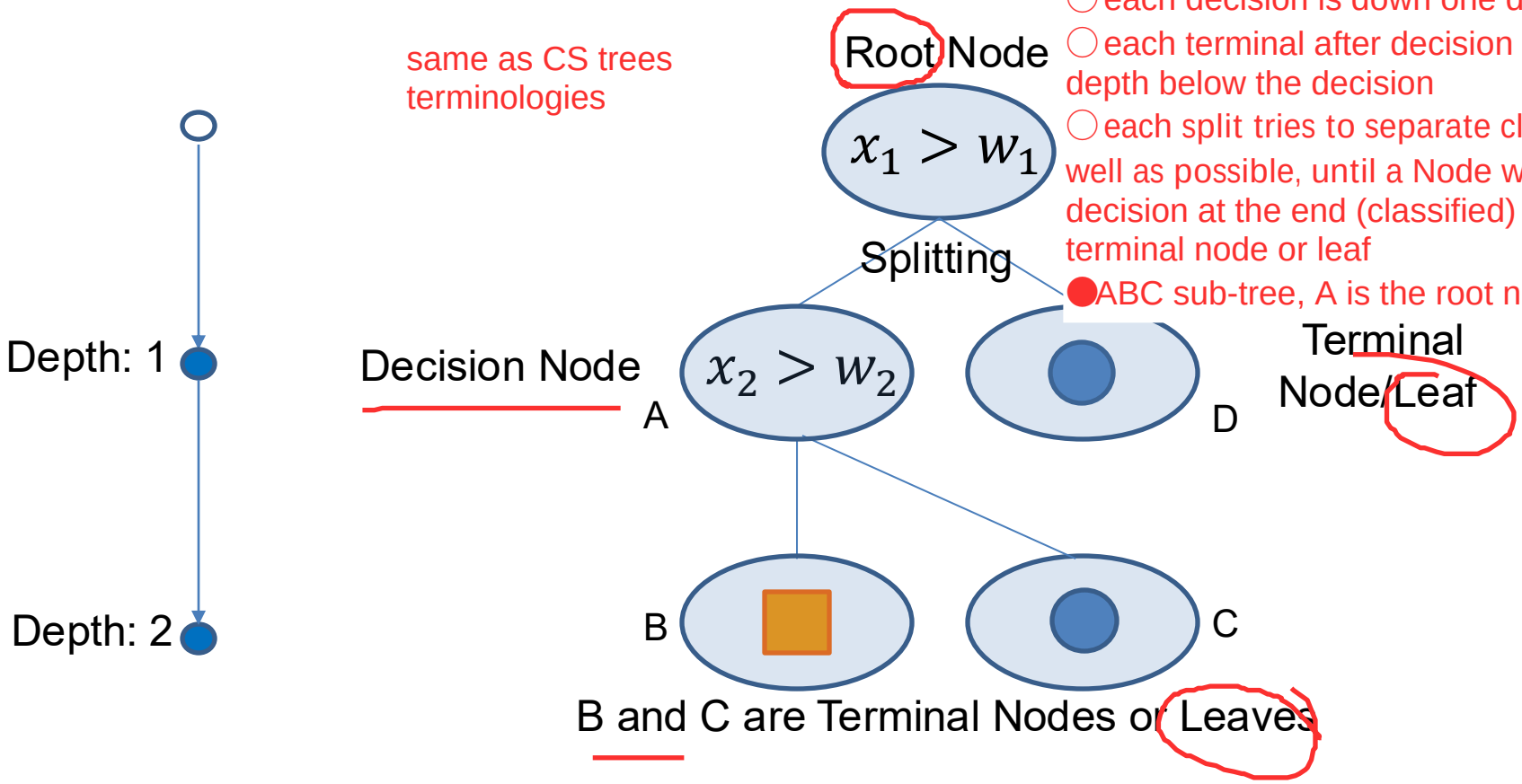




Basic Terminologies

- At each decision node, a decision is made after asking qns about a feature
- Decision is causes splitting of branch
  - root node is and must be a decision node, which is first node at depth 0
  - each decision is down one depth
  - each terminal after decision is one depth below the decision
  - each split tries to separate classes as well as possible, until a Node with no decision at the end (classified) is the terminal node or leaf
- ABC sub-tree, A is the root node

same as CS trees terminologies



A-B-C forms a **sub-tree** or **branch**.
A is **parent node** of B and C; B and C are **children nodes** of A.

Building a Classification Decision Tree

- Classification tree learning (or tree induction or tree growing) is the construction of a classification tree given training data
- For a given training set, there can be many trees with 0 training error, so we prefer less "complex" trees
- Complexity can be defined as number of nodes in the tree
- Finding smallest tree is computationally hard, so we typically use some greedy algorithm not guaranteed to find the best tree
- We need to first define the concept of node impurity

tree can keep splitting until each leaf has just one sample or all leaves are pure! which is 0 training error (or impurity).

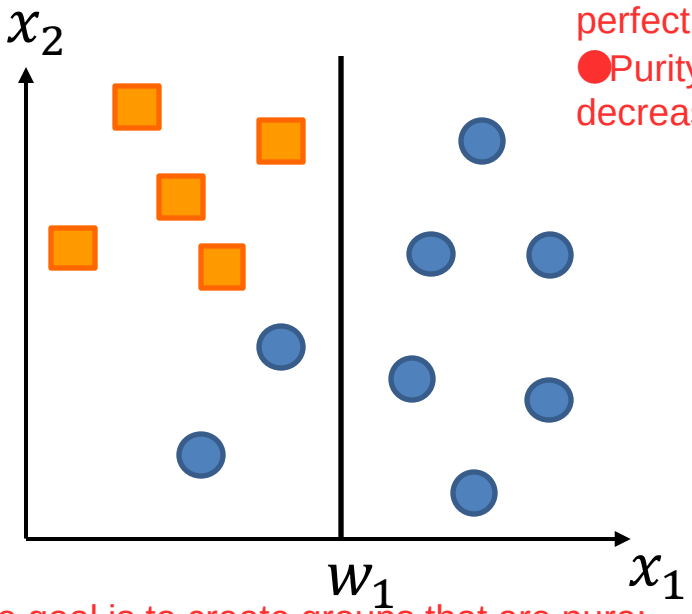
NP-hard to find a very small tree to classify with high purity, too many permutations and computation. At each step, choose the best local splitEven though it might not be the global best

Key: the decision tree MODEL is modeled by all the decision rules at each decision node, since that is how the model classify a given test data starting from root, goes through all the decision nodes, then land on a leaf node, which is considered classified.

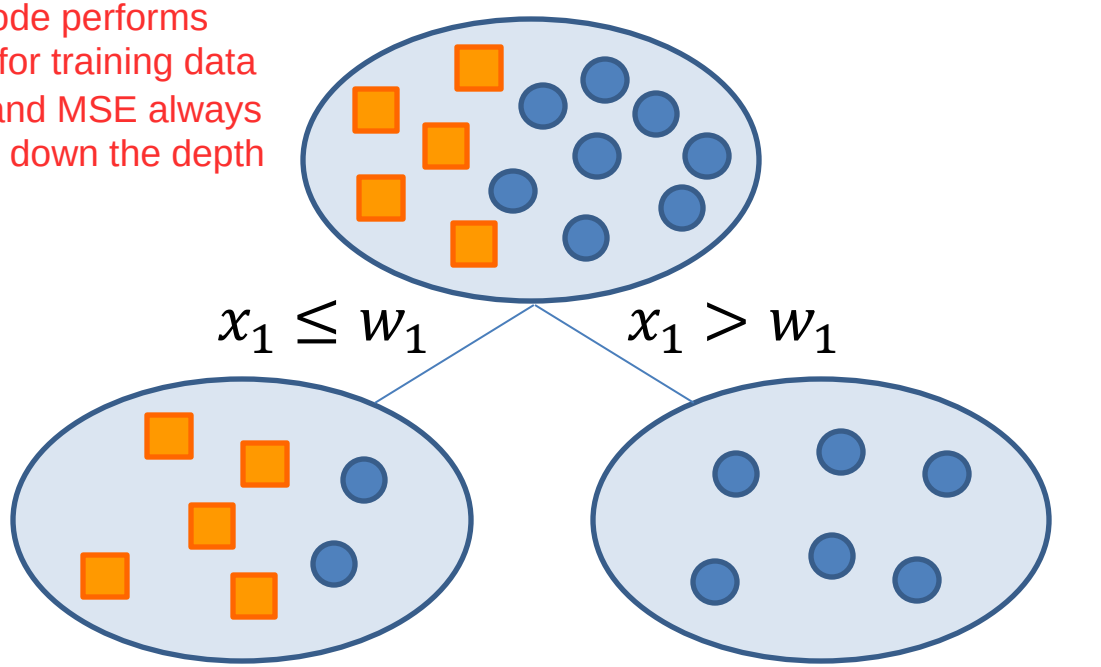
every split is a component of the weight vector. Each split is by a decision rule which was trained by comparing all features and threshold and computing each impurity before determining the exact decision rule that minimises the impurity. All these decision rules at each split makes up the typical weight vector

More complicated (mode nodes and more decisions = large weight) tree means it has overfitted itself to the training data by tuning all its decisions so finely based on training data, hence we prefer less complex trees

Node Impurity



- Pure node performs perfectly for training data
- Purity and MSE always decrease down the depth



"Impure" because it contains ■ & ●

"Pure" because it only contains ●

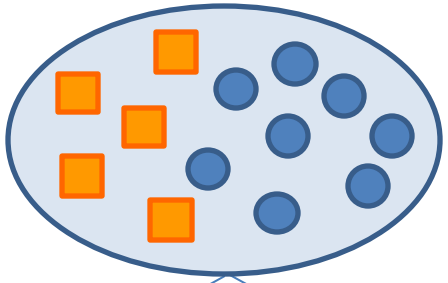
The goal is to create groups that are pure:
A pure node contains samples of only one class.
An impure node contains a mix of classes.
A perfect classifier would split the data until all leaf nodes are pure.

- "Purity" is desirable because if a node contains only training data from one class, then prediction for training data in the node is perfect

should minimise, decrease over decisions (closer to classification

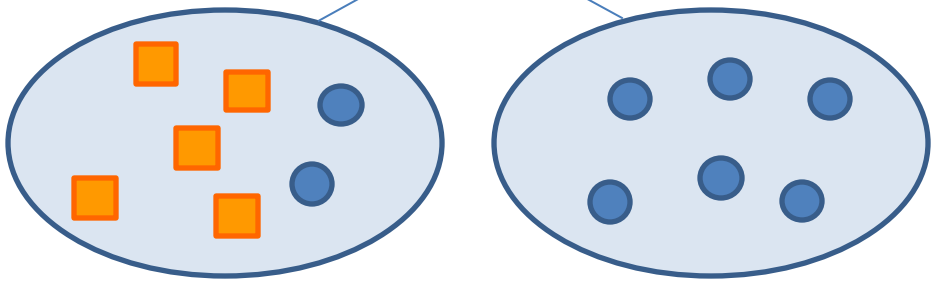
Node Impurity Measures

- Let ● be class 1 & ■ be class 2
- For particular node m, let p_i be the fraction (or probability) of data samples in node m belonging to class i
- Let Q_m be impurity of node m
- 3 impurity measures: Gini, entropy, misclassification rate



$$p_1 = \frac{8}{13}$$
$$p_2 = \frac{5}{13}$$

All of them:
Are 0 when a node is pure (all one class)
Are high when the classes are mixed
Diff lies in how the impurity is measured.



$$p_1 = \frac{2}{7}$$
$$p_2 = \frac{5}{7}$$

$$p_1 = 1$$
$$p_2 = 0$$

The tree tries to pick the split that reduces impurity the most.

Gini Impurity

To make sure the decision is making progress, check the Overall Gini at that depth. Gini should decrease compared to parent node (lower Gini, means gets purer and means making progress

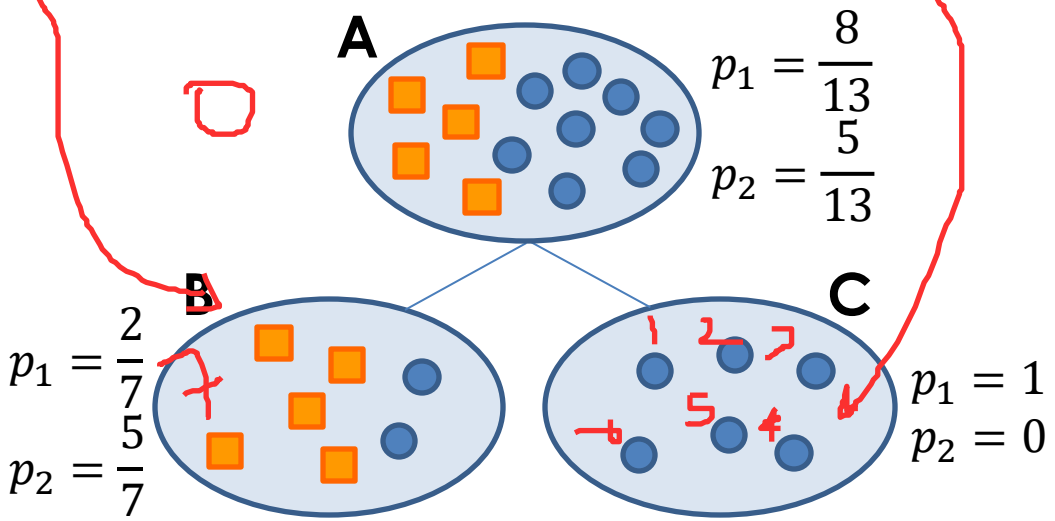
- Let K = # classes, define $Q_m = 1 - \Sigma_{i=1}^{K} p_i^2 = 1 - p_1^2 - p_2^2$
- Node A: $Q_{\mathrm{A}} = 1 - (8/13)^2 - (5/13)^2 = 0.4734$
- Node B: $Q_{\mathrm{B}} = 1 - (2/7)^2 - (5/7)^2 = 0.4082$
- Node C: $Q_{\mathrm{C}} = 1 - 1^2 - 0^2 = 0$
- Overall Gini (depth 1) = fraction of data samples in node B x Q_{B} + fraction of data samples in node C x Q_{C}
  - $\left(\frac{7}{13}\right) \times 0.4082 + \left(\frac{6}{13}\right) \times 0 = 0.2198$
- Observe lower impurity at depth 1 compared with root
- Same Gini formula for more than 2 classes:
  $Q_m = 1 - \Sigma_{i=1}^{K} p_i^2$

50/50 cannot tell apart

HIgher Gini, more impure. Gini = 0, purest pure. will not > 1

max impurity is when the class is evenly distributed in a node, cannot tell apart the class apart (i.e. 0.5)

Cost at depth is just the weighted average of the cost of nodes

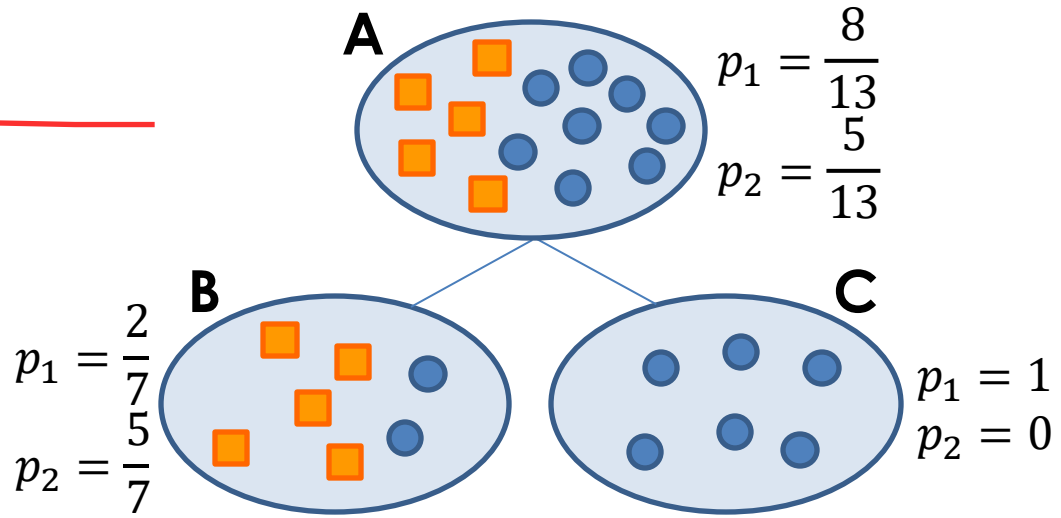


Entropy

Similar to Gini, High entropy, more impure. Grows more sharply near perfectly pure modes than Gini

- Entropy = 0, purest pure; max is 1 (main diff)
- log0 is undefined, but multiplied by 0

- Let K = # classes, define $Q_m = -\Sigma_{i=1}^{K} p_i \log_2 p_i = -p_1 \log_2 p_1 - p_2 \log_2 p_2$
- Node A: $Q_{\mathrm{A}} = -(8/13)\log_2(8/13) - (5/13)\log_2(5/13) = 0.9612$
- Node B: $Q_{\mathrm{B}} = -(2/7)\log_2(2/7) - (5/7)\log_2(5/7) = 0.8631$
- Node C: $Q_{\mathrm{C}} = -1\log_2 1 - 0\log_2 0 = 0$
- Overall entropy (depth 1) = fraction of data samples in node B x Q_{B} + fraction of data samples in node C x Q_{C}
  - $\left(\frac{7}{13}\right) \times 0.8631 + \left(\frac{6}{13}\right) \times 0 = 0.4648$
- Observe lower impurity at depth 1 compared with root
- Same entropy formula for more than 2 classes:
  $Q_m = -\Sigma_{i=1}^{K} p_i \log_2 p_i$



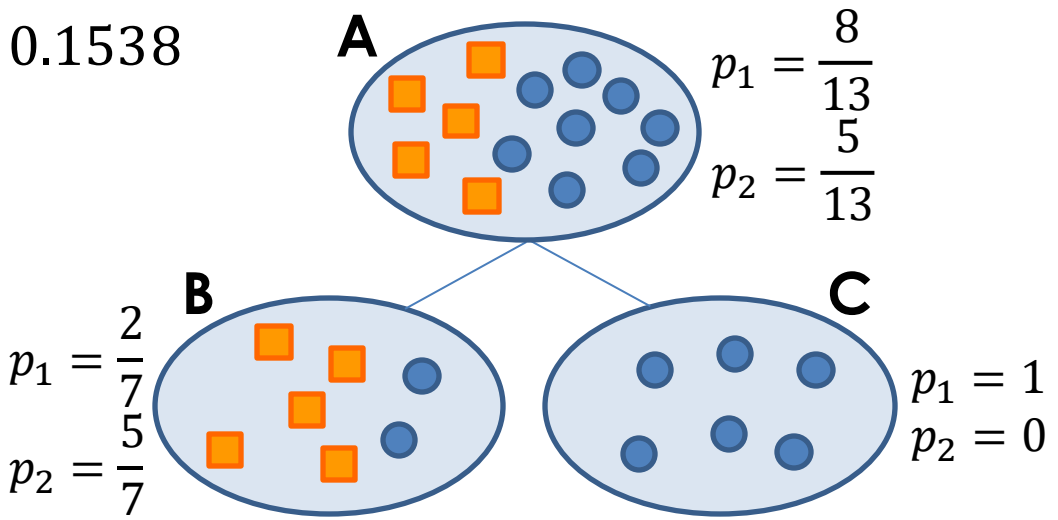
looks only at the most common class. SImple but not very senstive. Gini or entropy is better. max is 0.5

higher means more impure

Overall misclass is also simply total misclassified samples at the depth/total data point ◯ total misclassification at the depth = 2 ◯ total = 13 ◯ 2/13

Misclassification rate

- Let K = # classes, define $Q_m = 1 - \max_i p_i = 1 - \max(p_1, p_2)$
- Node A: $p_1 > p_2$, so best classification = class 1 => $Q_{\mathrm{A}} = 1 - 8/13 = 5/13$
- Node B: $p_2 > p_1$, so best classification = class 2 => $Q_{\mathrm{B}} = 1 - 5/7 = 2/7$
- Node C: $p_1 > p_2$, so best classification = class 1 => $Q_{\mathrm{C}} = 1 - 1 = 0$
- Overall misclassification rate (depth 1) = fraction of data samples in node B x Q_{B} + fraction of data samples in node C x Q_{C}
  - $\left(\frac{7}{13}\right) \times \left(\frac{2}{7}\right) + \left(\frac{6}{13}\right) \times 0 = 0.1538$
- Observe lower impurity at depth 1 compared with root
- Same misclassification rate formula for more than 2 classes: $Q_m = 1 - \max_i p_i$



Classification Tree Learning

Algorithm: Classification Tree Learning

Input: Impurity measure Q, parameter max_depth & training set

Output: Tree

start with all training data in root (very impure)

this is leaf node SO FAR, because we can always convert a laef node into a decision node to split further

this means at a decision node, try every feature and and split threshold such that this decision rule maximises impurity reduction when split into two nodes

```
1 root ← all training samples
2 for d ← 1 to max_depth do
3     for each leaf node m at depth d − 1 do
4         Find best feature & best threshold, so splitting
          node m into two reduces the most impurity
5         Use decision rule to distribute training samples
          from node m across two new leaf nodes
6 return tree
```

forms the rule

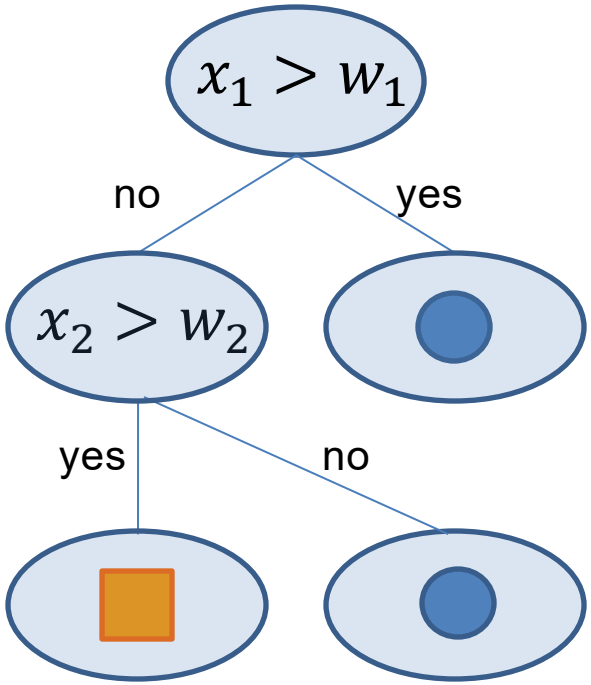
return tree here because it is a recursion so need to return pointerbut at the end, each leaf node should be as pure (one class) to predict the (majority) class of the samples inside it

When comparing cost to choose the best threshold, the impurity is the node and the two new nodes, not of other nodes

Advantages / Disadvantages

Advantages

- Easy to visualize & understand tree



Advantages / Disadvantages

Advantages

decisions can be done using either and depending on data it will just be either regression tree or classification tree

- Easy to visualize & understand tree
- Can work with a mix of continuous and discrete data
- Less data cleaning required — less affected by outliers
- Makes less assumptions about the relationship between features & target — unlike all past models, there is no fixed model form like linear or quadratic

Disadvantages

too refined, too many splits, over learned

- Trees can become overly complex resulting in overfitting
- Trees can be unstable, e.g., small changes in training data can result very different trees

Unstable as any change to a node will affect all the nodes after it, variation to training data affects training outcomes a lot (feature or threshold, improve by random forest) ⇒ high variance

To reduce overfitting…

- One or more of the following can help reduce overfitting
  - Set maximum depth for the tree
  - Set minimum number of samples for splitting a leaf node, e.g., if leaf node has less than 10 samples, then do not split node
  - Set minimum decrease in impurity, e.g., if selecting the best feature & threshold does not improve impurity by at least 1%, then do not split the leaf node
  - Instead of looking at all features when considering how to split a leaf node, we can randomly look at a subset (e.g., square root of the total number of features)

Regression Trees

●No longer majority to predict, prediction is now average of traget values of the sample in that leaf (Prediction = average of the Ytrain) because the value that minimises squared error is mean
●MSE = sum of error squared of Ytrain and Yave
○applicable at any depth of node ○root will be Yave of all the value
●At a whole depth is also the weighted average of MSE

- Classification trees seek to predict discrete variables (i.e., classification)
- Regression trees seek to predict continuous variables (i.e., regression)

predicts class^
predicts number

- Can use same approach as before, but instead of minimizing impurity, we can try to minimize mean square error (MSE)
- Suppose there are J_m training samples in a leaf node m of the regression tree with target values $y_1, y_2, \cdots, y_{J_m}$
  - We can predict $\hat{y}_m = \frac{1}{J_m} \sum_{j=1}^{J_m} y_j$

at each leaf

  - Then MSE of node m is given by $S_m = \frac{1}{J_m} \sum_{j=1}^{J_m} (y_j - \hat{y}_m)^2$
- Across all leaf nodes, total MSE $S = \sum_m \frac{J_m}{N} S_m$, where N is the total number of data samples

Regression Tree Learning

- Algorithm is basically the same as classification tree learning
- Various approaches to reduce overfitting also apply here

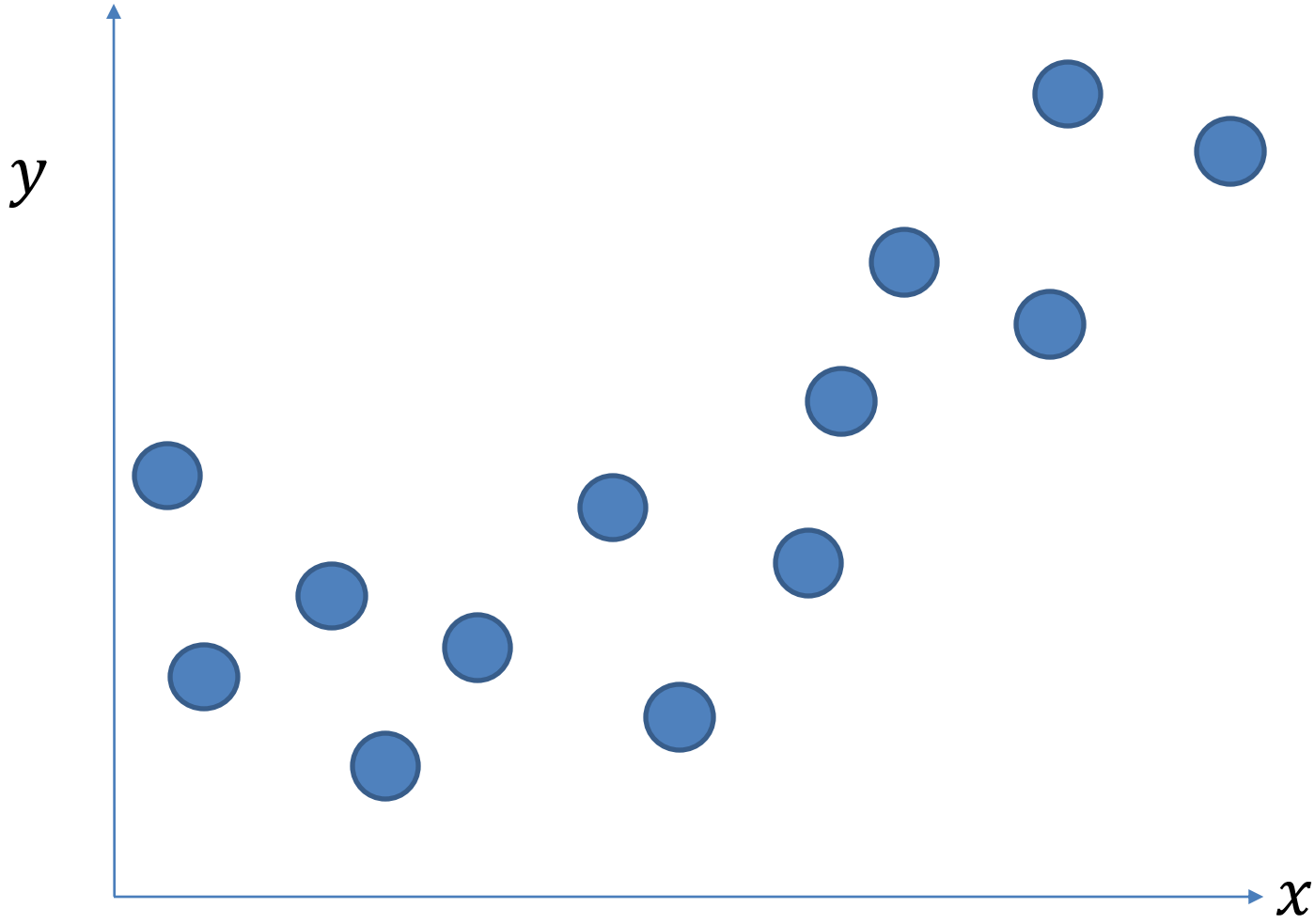
Algorithm: Regression Tree Learning

Input: parameter *max_depth* & training set
Output: Tree

```
1 root ← all training samples
2 for d ← 1 to max_depth do
3     for each leaf node m at depth d − 1 do
4         Find best feature & best threshold, so splitting
           node m into two reduces MSE the most
5         Use decision rule to distribute training samples
           from node m across two new leaf nodes
6 return tree
```

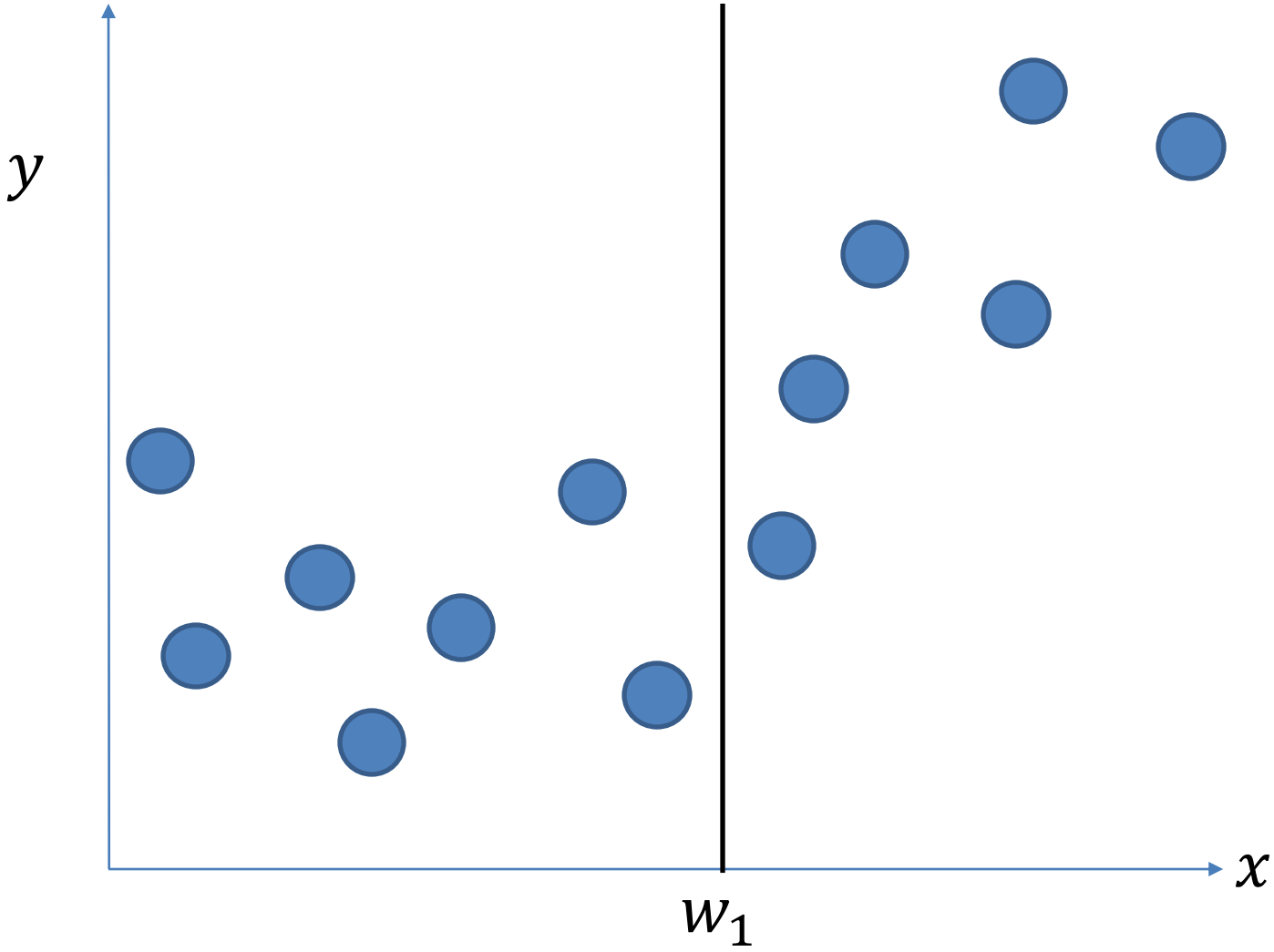
only diff

Regression Tree Example

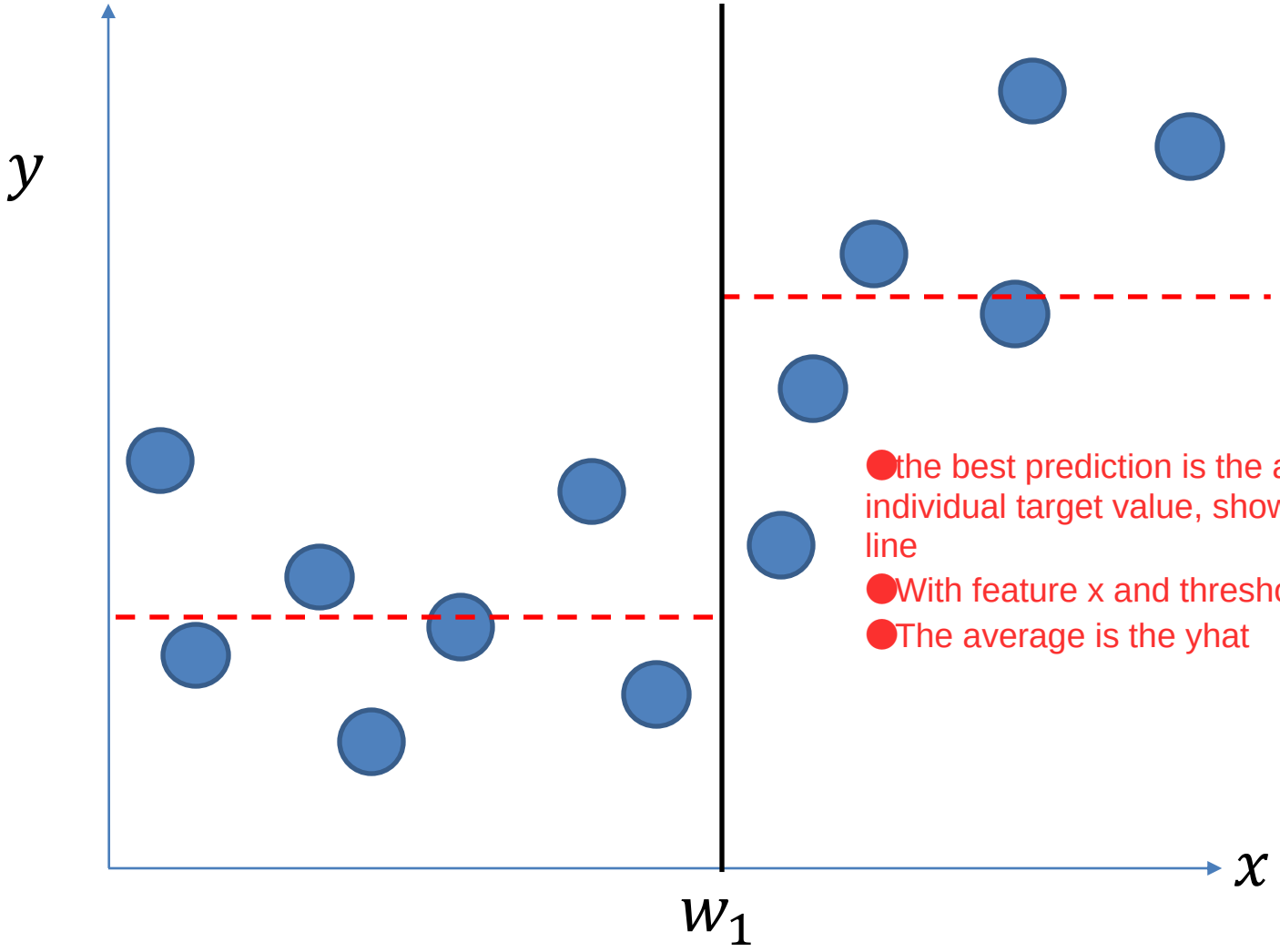


Regression Tree Example

once again as said previously in slide 89 a split (decision) is a component in weight vector

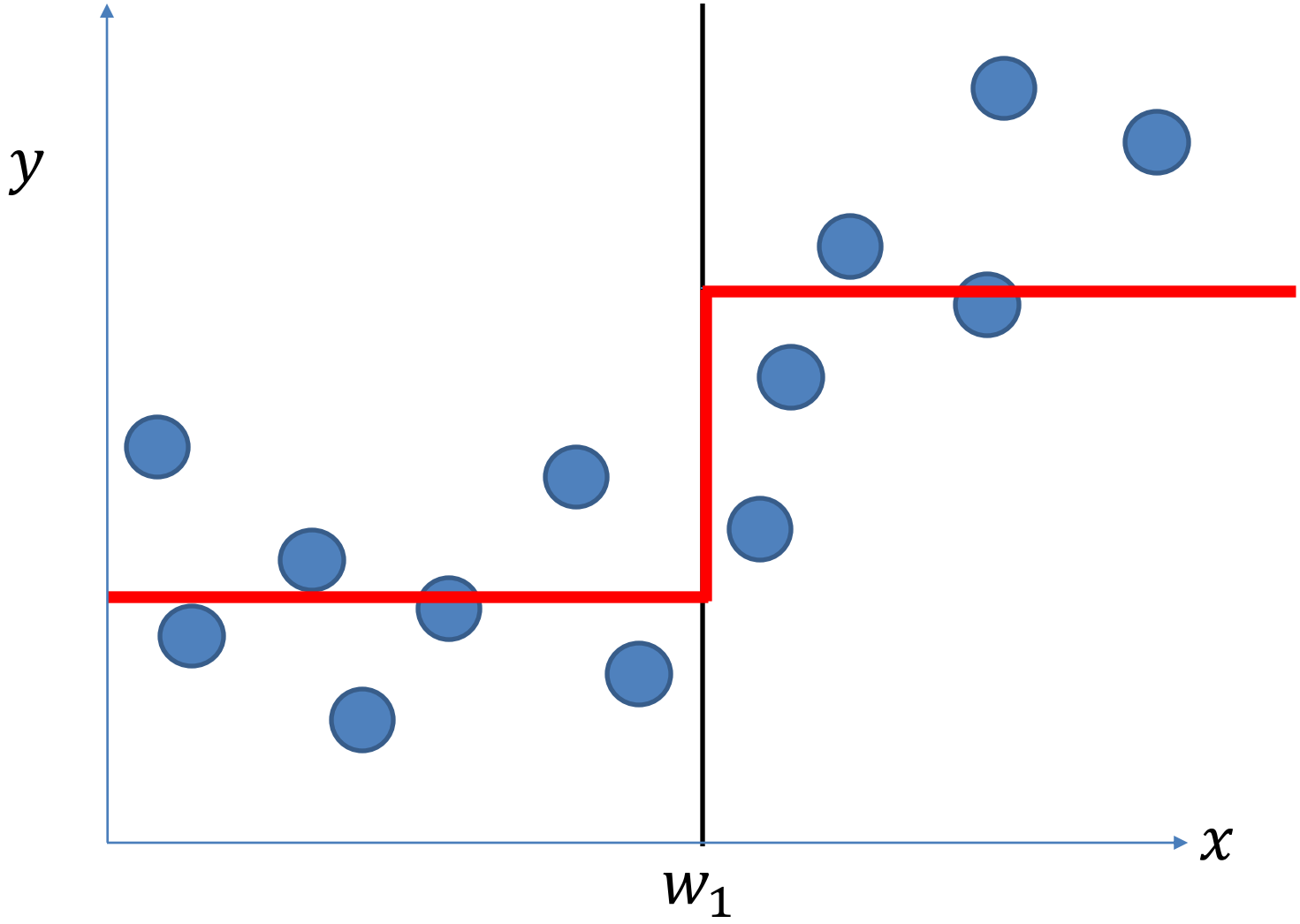


Regression Tree Example

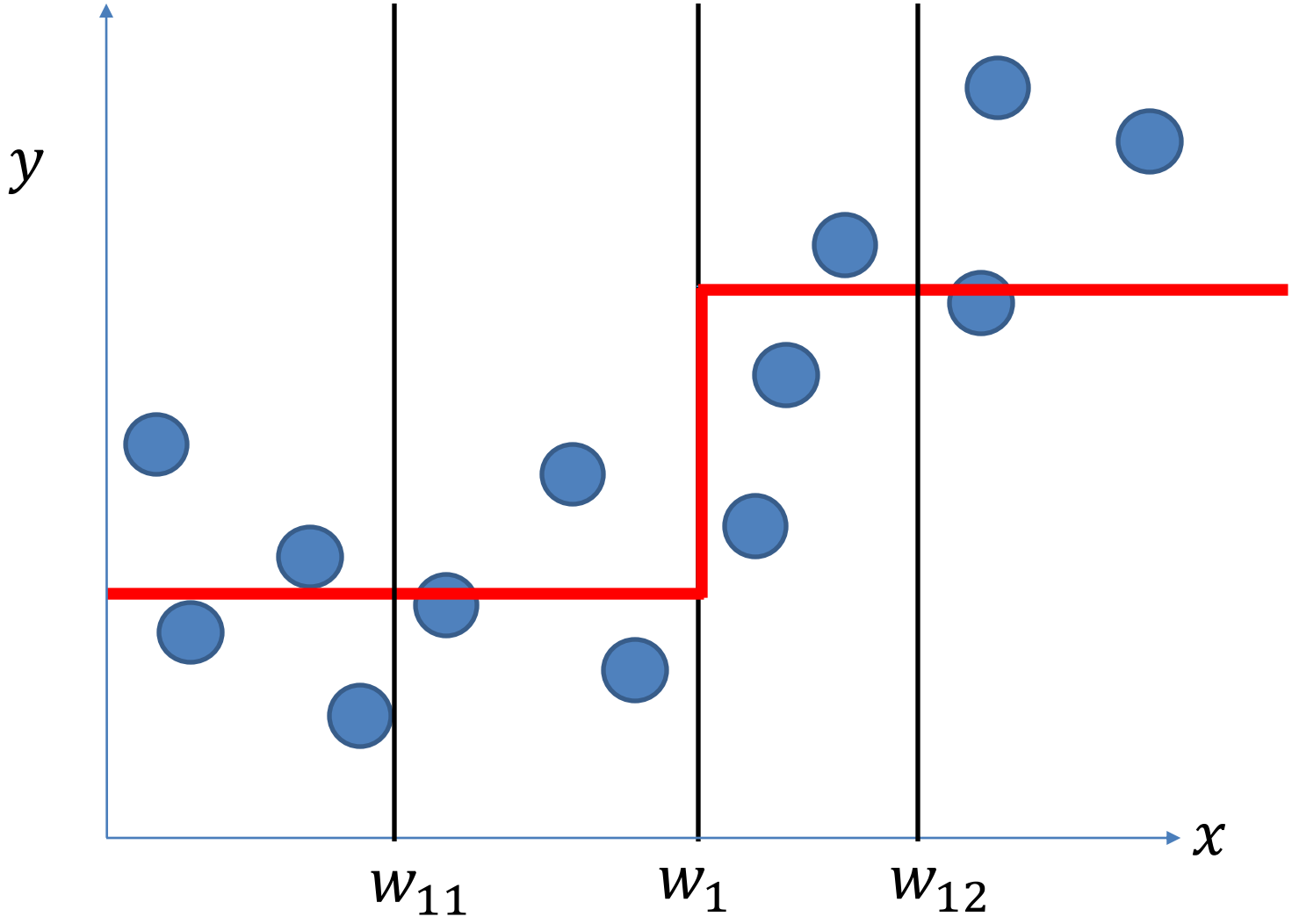


- the best prediction is the average of the individual target value, shown by the red line
- With feature x and threshold w1
- The average is the yhat

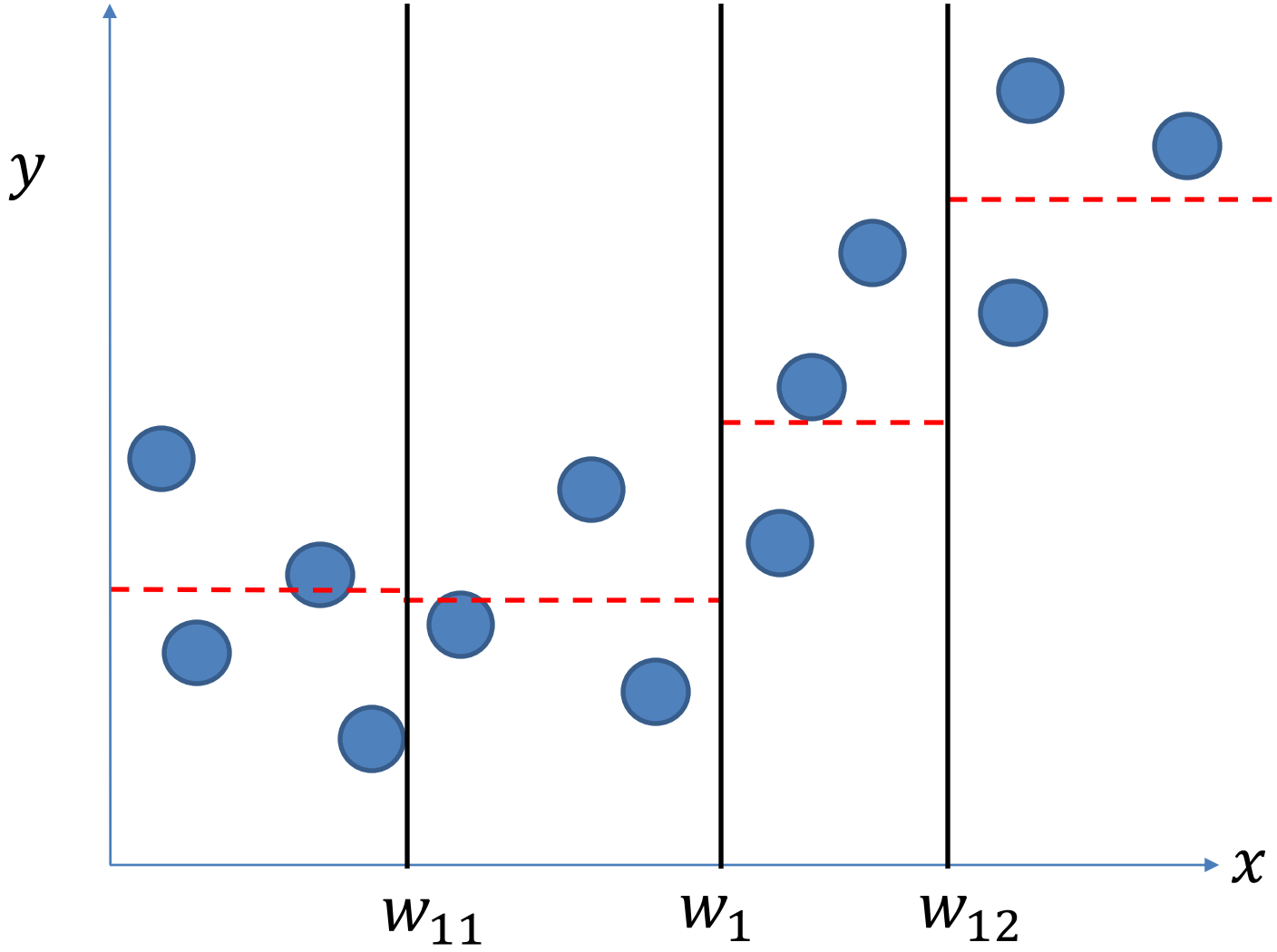
Regression Tree Example



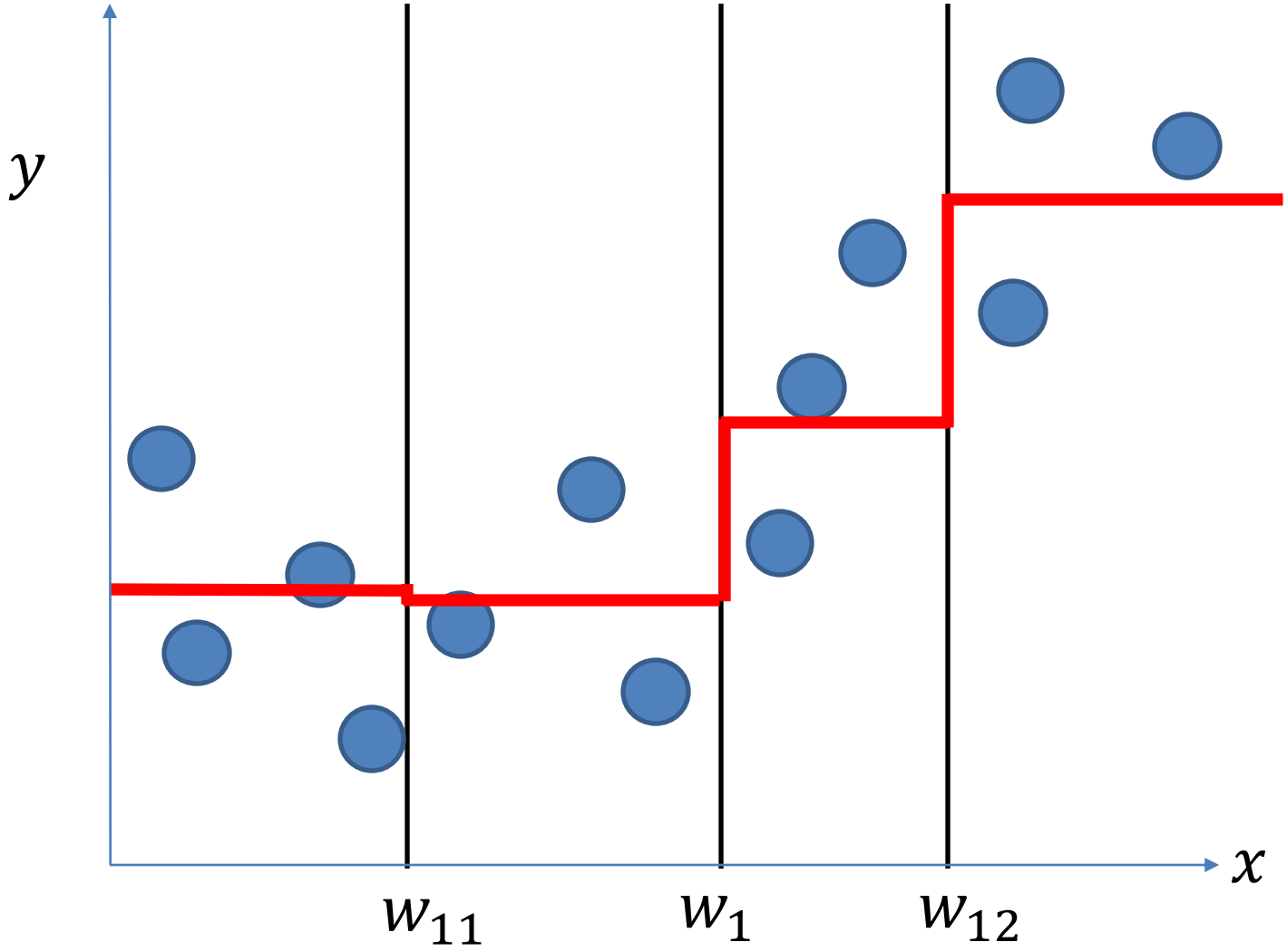
Regression Tree Example



Regression Tree Example



Regression Tree Example



Example of Regression Tree

Example of Regression Tree

- Consider house prices in Singapore.
- Target variable is Price P.
- Attributes are House Size S and Number of Rooms R.

	House Size ('000 sq ft)	Num of Rooms	Price ('000,000 SGD)
1	0.5	2	0.19
2	0.6	1	0.23
3	1.0	3	0.28
4	2.0	5	0.42
5	3.0	4	0.53
6	3.2	6	0.75
7	3.8	7	0.80

- Note that I have arranged the data points in increasing order of P, which so happens to be increasing order of S as well. However, this is not the same order as that of R.

Example of Regression Tree

Calculation of MSE for House Size Split

- Focus first on the House Size attribute S. If we set the threshold at $\tau = 0.75$, then the targets of the two classes are $\{0.19, 0.23\}$ and $\{0.28, 0.42, 0.53, 0.75, 0.80\}$. The individual conditional MSEs are

$$\text{MSE}_{P|S<0.75} = 4 \times 10^{-4} \quad \text{and} \quad \text{MSE}_{P|S\geq 0.75} = 0.0385.$$

- Thus, the averaged conditional MSE with a split of S at 0.75 is

$$\text{MSE}_{P|S(0.75)} = \frac{2}{7}\text{MSE}_{P|S<0.75} + \frac{5}{7}\text{MSE}_{P|S\geq 0.75} = 0.0276.$$

- Sweep through all possible thresholds τ to determine the best threshold for attribute S.

$\text{MSE}_{P\|S(0.55)}$	$\text{MSE}_{P\|S(0.75)}$	$\text{MSE}_{P\|S(1.5)}$	$\text{MSE}_{P\|S(2.5)}$	$\text{MSE}_{P\|S(3.1)}$	$\text{MSE}_{P\|S(3.5)}$
0.0402	0.0276	0.0145	0.0102	0.0116	0.0325

$$S_m = \frac{1}{J_m}\sum_{j=1}^{J_m}(y_j - \hat{y}_m)^2 \qquad \text{MSE } S = \sum_m \frac{J_m}{N} S_m$$

Example of Regression Tree

Calculation of MSE for # Rooms Split

- Rearrange the target variables in order of the house sizes. Doing so we get $(0.23, 0.19, 0.28, 0.53, 0.42, 0.75, 0.80)$. Now we sweep through all possible thresholds τ for R to get the following averaged conditional MSEs.
- We get the following table.

$\text{MSE}_{P\|R(1.5)}$	$\text{MSE}_{P\|R(2.5)}$	$\text{MSE}_{P\|R(3.5)}$	$\text{MSE}_{P\|R(4.5)}$	$\text{MSE}_{P\|R(5.5)}$	$\text{MSE}_{P\|R(6.5)}$
0.0435	0.0276	0.0145	0.0222	0.0116	0.0325

Choose the threshold with minimum MSE

Example of Regression Tree

Where is the First Split?

- We should first split the dataset into two branches, the left branch indicating $S < 2.5$ and the right with $S \geq 2.5$.
- Split the dataset into two sub-datasets and we may decide to stop or split the R feature.
- If we decide to stop, then for any new/test house with a house size of < 2.5, we will predict that its price is the average of the houses in our training set whose size is < 2.5, i.e.,

$$(0.19 + 0.23 + 0.28 + 0.42)/4 = 0.28.$$

- For a new/test house with a house size of ≥ 2.5, we will predict that its price is the average of the houses in our training set whose size is ≥ 2.5, i.e.,

$$(0.53 + 0.75 + 0.80)/3 = 0.6933.$$

Python demo

To reduce instability…

- For both classification & regression trees, small perturbations to data can result in very different trees => low bias, high variance
- To reduce variance, we can perturb training set to generate M perturbed training sets
  - Train one tree for each perturbed training set
  - Average predictions across the M trees?
- For example, if we have 100 regression trees (trained from 100 perturbed training sets)
  - Given features x from new test sample, the i-th tree predicts $f_i(x)$
  - Then final prediction is $\frac{1}{100}\sum_{i=1}^{100} f_i(x)$
- For example, if we have 100 classification trees (trained from 100 perturbed training sets)
  - Given features x from new test sample, the i-th tree predicts $g_i(x)$
  - Then final prediction is the most frequent class among 100 predictions $g_1(x), g_2(x), \cdots, g_{100}(x)$

no assumption about the data

due to compounded decisions

change inputs but dont change outputs

for regression task

for classification task

Random Forest

instead of using a single decision tree to predict

random tree has much lower variance since it is trained to deal with noise. It has ~~similar~~ same low bias as single decision trees squared

- To perturb data, we can apply "bootstrapping" to the training set to create a new "bootstrapped" dataset
- Bootstrapping is procedure in which we sample data with replacement
  - For example, given training set with 3 data samples $\{x_1, y_1\}$, $\{x_2, y_2\}$, $\{x_3, y_3\}$, a bootstrapped training set might comprise sample 1 $\{x_2, y_2\}$, sample 2 $\{x_2, y_2\}$ and sample 3 $\{x_1, y_1\}$
  - Bootstrapped dataset is the same size as original dataset
  - Bootstrapped dataset might contain repeated samples
  - Bootstrapped dataset might not contain some samples from original dataset

Since each individual tree in a Random Forest is typically allowed to grow deep (unless explicitly pruned), they are, individually, as likely to capture the complex patterns in the data as a single decision tree, thereby having a similar squared bias.

The forest does not reduce bias because it's the average of the biases of its trees, and each tree has a similar bias as a well-trained single decision tree.
No new assumption nor "less" assumption.

Random Forest

Algorithm: Random Forest Learning

Input: parameter max_trees & N training samples
Output: Forest

```
for t ← 1 to max_trees do
    dataset ← bootstrap(N training samples)
    trees[t] ← TreeLearning(dataset)
forest ← average(trees)
return forest
```

quite straightforward

- To increase randomness, when training the trees, instead of looking at all features when considering how to split a node, we can randomly look at a subset (e.g., square root of the total number of features)

looking at fewer features reduces overfitting also

Summary

- Decision tree: series of binary decisions to arrive at prediction of target variable
- Classification tree predicts discrete target class
- Classification tree learning
  - Impurity measures: Gini, Entropy, Misclassification rate
  - For each leaf node, find best feature and threshold to split node to minimize impurity
- Regression tree predicts continuous target
  - Minimize MSE instead of impurity
- Random forest
  - Generate multiple bootstrapped training sets
  - Train on each bootstrapped training set & average