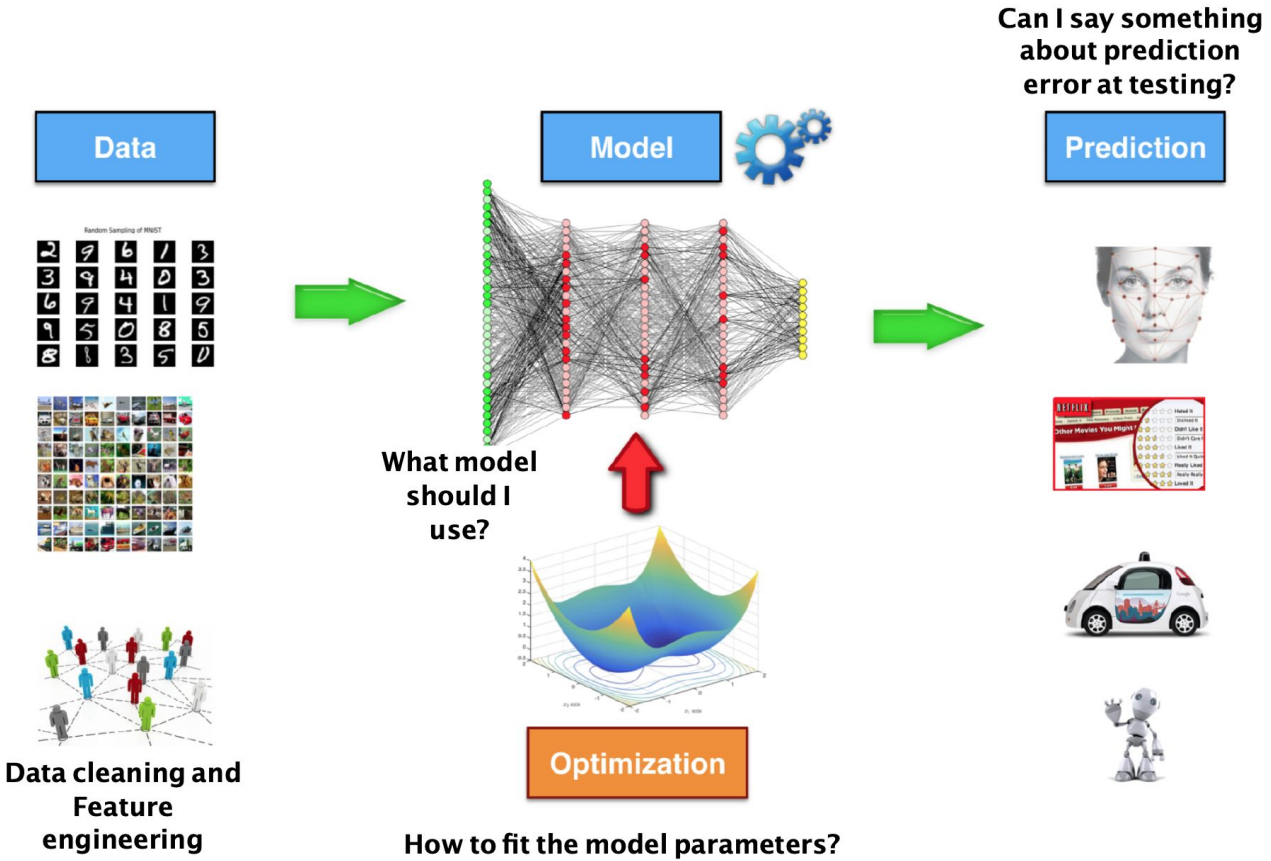
# CMPSC 448: Machine Learning

## Lecture 2. The Process of Learning

Rui Zhang
Fall 2022

PennState

# The big picture



# What does it mean to learn?

- Let's consider a simple food allergy prediction problem
  - binary classification problem as the label set is binary {0, 1}
- We are given **training** data where we know labels:

**training data**

| Egg | Milk | Fish | Wheat | Shellfish | Peanuts | ... |
|---|---|---|---|---|---|---|
| 0 | 0.7 | 0 | 0.3 | 0 | 0 | |
| 0.3 | 0.7 | 0 | 0.6 | 0 | 0.01 | |
| 0 | 0 | 0 | 0.8 | 0 | 0 | |
| 0.3 | 0.7 | 1.2 | 0 | 0.10 | 0.01 | |
| 0.3 | 0 | 1.2 | 0.3 | 0.10 | 0.01 | |

| Sick? |
|---|
| 1 |
| 1 |
| 0 |
| 1 |
| 1 |

**training labels**

- But the goal is to do well on any possible **testing** data:

**testing data**

| Egg | Milk | Fish | Wheat | Shellfish | Peanuts | ... |
|---|---|---|---|---|---|---|
| 0.5 | 0 | 1 | 0.6 | 2 | 1 | |
| 0 | 0.7 | 0 | 1 | 0 | 0 | |
| 3 | 1 | 0 | 0.5 | 0 | 0 | |

| Sick? |
|---|
| ? |
| ? |
| ? |

**testing labels**

**In reality and during learning, we have no idea what would be the test data**

# Goal of machine learning

- What we care about is the **TEST** error. Algorithms that can **generalize from training data to unseen test data**!

> Midterm analogy:
> - The training error is the practice midterm.
> - The test error is the actual midterm.
>
> Goal: do well on actual midterm, not the practice one.

- Memorization vs Learning:
  - We can do well on training data by memorizing it.
  - You've only learned if you can do well on predicting new situations (test error).

**Machine learning all is about generalizing (performance on unseen test data)**

# Is learning possible?

- Does training error say anything about test error?
  - In general, **NO**: Test data might have nothing to do with training data.
  - E.g., **adversary** takes training data and **flips all labels**.

| Egg | Milk | Fish |
|---|---|---|
| 0 | 0.7 | 0 |
| 0.3 | 0.7 | 1 |
| 0.3 | 0 | 0 |

| Sick? |
|---|
| 1 |
| 1 |
| 0 |

| Egg | Milk | Fish |
|---|---|---|
| 0 | 0.7 | 0 |
| 0.3 | 0.7 | 1 |
| 0.3 | 0 | 0 |

| Sick? |
|---|
| 0 |
| 0 |
| 1 |

**Training** **Test**

- In order to learn, we need some **assumptions**:
  - The training and test data need to be related in some way.
  - Most common assumption: **independent and identically distributed** (**i.i.d**).

# I.I.D assumption

- Training/test data is **independent and identically distributed (i.i.d)** if:
  - All objects come from the same distribution (identically distributed).
  - The object are sampled independently (order doesn't matter).
  - We do NOT need to know the underlying distribution as long as the samples are sampled i.i.d.

- Examples in terms of cards:
  - Pick a card, put it back in the deck, re-shuffle, repeat.
  - Pick a card, put it back in the deck, repeat.
  - Pick a card, don't put it back, re-shuffle, repeat

# I.I.D assumption in food allergy

| Egg | Milk | Fish | Wheat | Shellfish | Peanuts | ... |
|---|---|---|---|---|---|---|
| 0 | 0.7 | 0 | 0.3 | 0 | 0 | |
| 0.3 | 0.7 | 0 | 0.6 | 0 | 0.01 | |
| 0 | 0 | 0 | 0.8 | 0 | 0 | |
| 0.3 | 0.7 | 1.2 | 0 | 0.10 | 0.01 | |
| 0.3 | 0 | 1.2 | 0.3 | 0.10 | 0.01 | |

| Sick? |
|---|
| 1 |
| 1 |
| 0 |
| 1 |
| 1 |

Is the food allergy data i.i.d.?

- Do all the objects come from the same distribution?
- Does the order of the objects matter?

No!

- Being sick might depend on what you ate yesterday (not independent).
- Your eating habits might changed over time (not identically distributed).

# Learning Theory

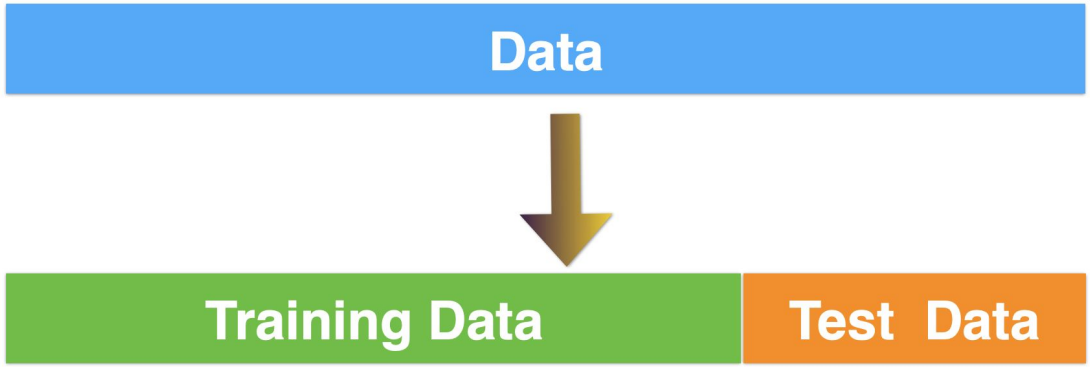
- The i.i.d assumption is rarely true:
  - But it is often a good approximation.
- Why does the i.i.d assumption make learning possible?
  - Patterns in training examples are likely to be the same in test examples.

**Learning theory explores how training error is related to test error, i.e., provides mathematical guarantees like this (under i.i.d. and other mild assumptions):**

**test error < training error + something small**

# What if I do not have test data

- At training time, I have no idea how test data going to look like?
- Let's split the training data into two parts (80%+20% or 70%+30%):
  - first part used only for training
  - second part used only for evaluation




- Basically trying to simulate the future test examples during training to evaluate model.
- The splitting should be random!

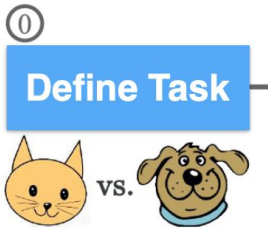
# Golden rules

- Even though what we care about is test error:

  THE TEST DATA CANNOT INFLUENCE THE TRAINING PHASE IN ANY WAY.

- We're measuring test error to see how well we do on new data:
  - If used during training, doesn't measure this.
  - You can start to overfit if you use it during training.
  - Midterm analogy: you are cheating on the test.

# Machine Learning Pipeline

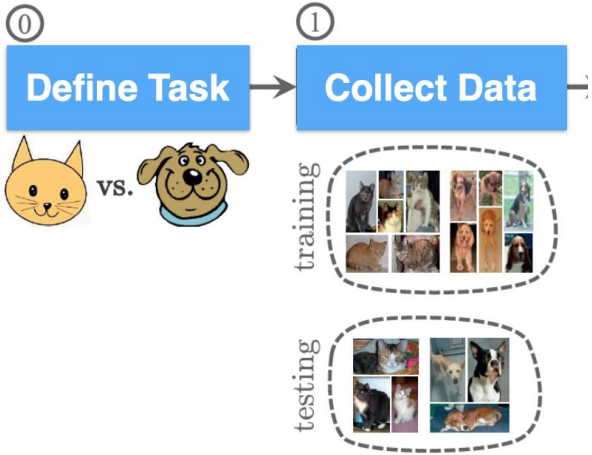



- Understanding the problem: What is the task we want to teach a computer to do?

# Understand the task

- What problem you intend to solve?
- What is the objective?
- How does data look like?
- What features are important/available?
- How much training data is available?
- How expensive would be to get labeled data?
- What is the nature of data?
    - Numerical
    - Images
    - Text

# Machine Learning Pipeline




- Understanding the problem: What is the task we want to teach a computer to do?
- Collect data: Gather data for training and testing sets. The larger and more diverse the data the better.

# Data Preparation

Machine learning typically assume ‘clean’ data.

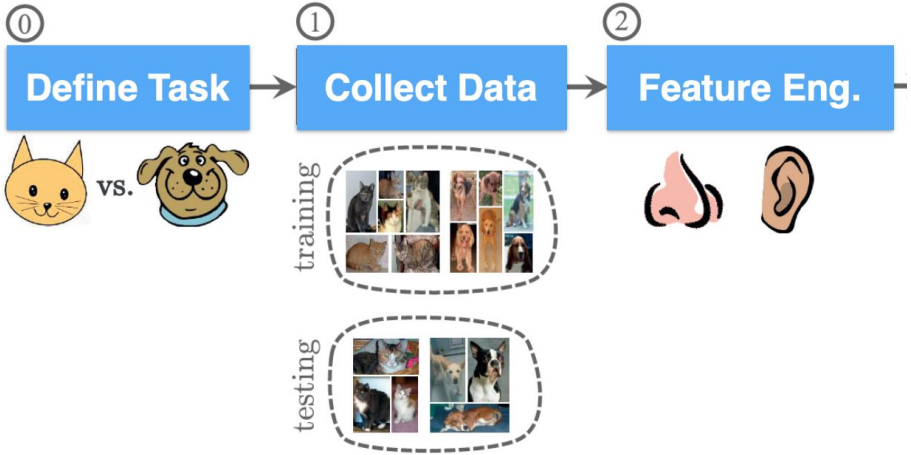
- Ways that data might NOT be ‘clean’:
  - Noise (e.g., distortion on phone).
  - Outliers (e.g., data entry or instrument error).
  - Missing values (no value available or not applicable)
  - Duplicated data (repetitions, or different storage formats).
  - Missing some important features!

- Any of these can lead to problems in analyses.
  - Want to fix these issues, if possible.
  - Some ML methods are robust to these.
  - Often, ML itself is the best way to detect/fix these.

**Garbage in/Garbage out**

If the data you give it is trash, the learning algorithm is unlikely to be able to overcome it.

# Machine Learning Pipeline

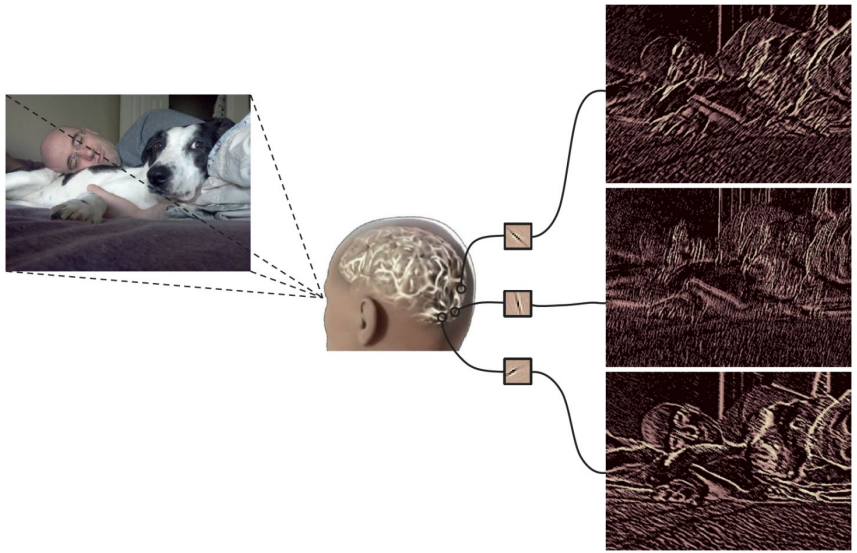



- Understanding the problem: What is the task we want to teach a computer to do?
- Collect data: Gather data for training and testing sets. The larger and more diverse the data the better.
- Feature engineering: What kind of features best describe the data?

# Features

A feature is representation of raw data. There are many ways to turn raw data into numeric measurements, which is why features can end up looking like a lot of things!

- Numerical features (ratings, likes)
- Textual (e.g., counts of words)
- Image (e.g., edge detector)



A good feature might beat a strong learning model!

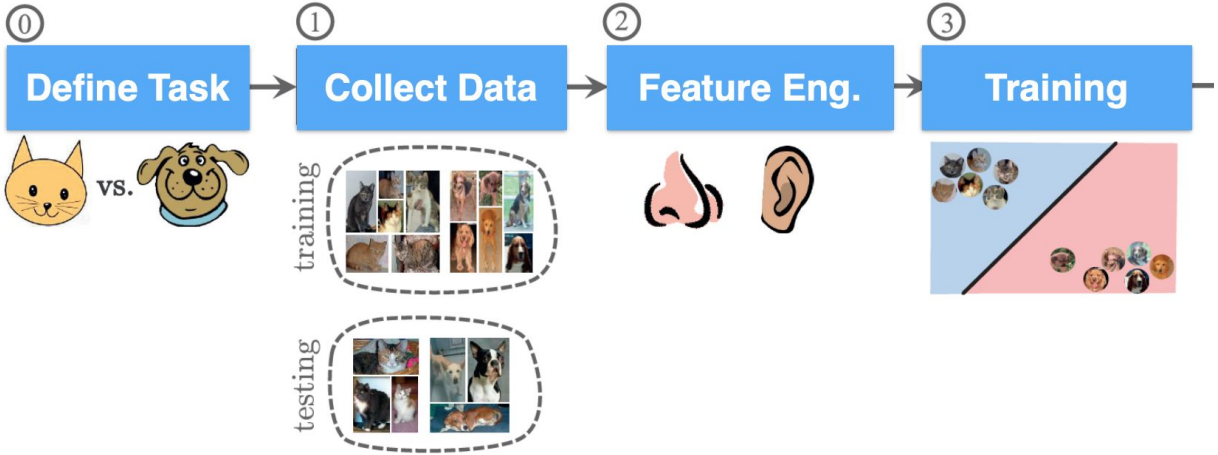
# Data Feature Matrix

- We'll define data as a collection of objects (training instances or samples), and their features.
- The output of data cleaning is the representation of training data!

| Instance | Job | City | Rating | Income | Age |
|---|---|---|---|---|---|
| 1 | | | | | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |

- Each row is an object, each column is a feature.

# Machine Learning Pipeline




- Understanding the problem: What is the task we want to teach a computer to do?
- Collect data: Gather data for training and testing sets. The larger and more diverse the data the better.
- Feature engineering: What kind of features best describe the data?
- Learning and optimization: Tune the parameters of an appropriate model on the training data using numerical optimization

# Model

- Humans try to understand the world around them by representing it in different ways.
- The data world is complex, random, and uncertain. At the same time, it’s one big data-generating machine.
- Statisticians and data scientists capture the uncertainty and randomness of data-generating processes with mathematical functions that express the shape and structure of the data itself.

# "All models are wrong, but some are useful"

- Humans try to understand the world around them by representing it in different ways.
- The data world is complex, random, and uncertain. At the same time, it’s one big data-generating machine.
- Statisticians and data scientists capture the uncertainty and randomness of data-generating processes with mathematical functions that express the shape and structure of the data itself.

**Question**: How do you have any clue whatsoever what functional form the data should take?

**Answer**: We do not know! But we can create model spaces and hopefully they will approximate well the actual functional form of the real data

# Fitting a model

- I cleaned my data, decided on a model, what is next?
  - We need to find the best model (parameters) among all models!

- Learning is almost like optimization (Optimization algorithms are our best friends here!)
  - Choose a model
  - Choose an objective function (loss function, error function, etc.)
  - Find parameters that maximize/minimize objective function over training data

# Examples of learning models

Different learning algorithms can be trained using different model spaces:

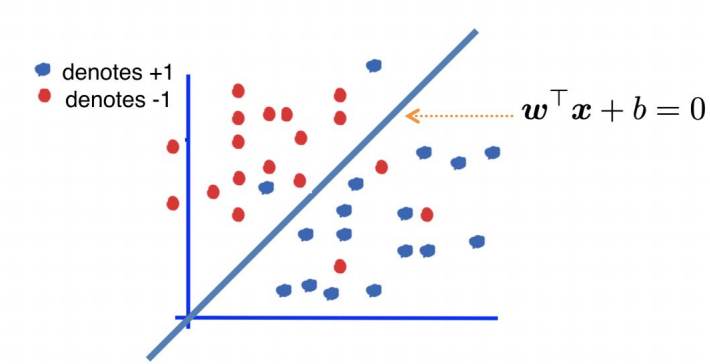
- Decision trees: A hierarchy of if-then-else rules

# Examples of learning models

Different learning algorithms can be trained using different model spaces:

- Decision trees: A hierarchy of if-then-else rules


- Linear models

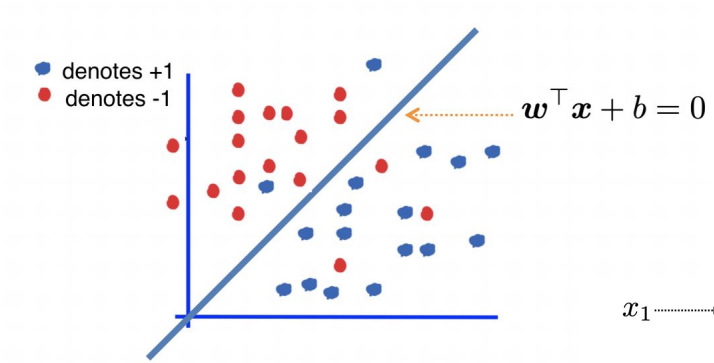


# Examples of learning models

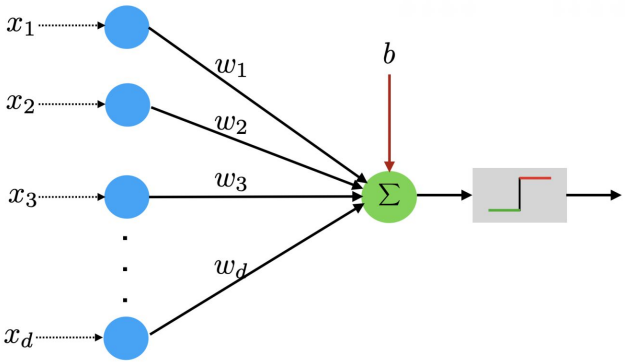
Different learning algorithms can be trained using different model spaces:

- Decision trees: A hierarchy of if-then-else rules


- Linear models

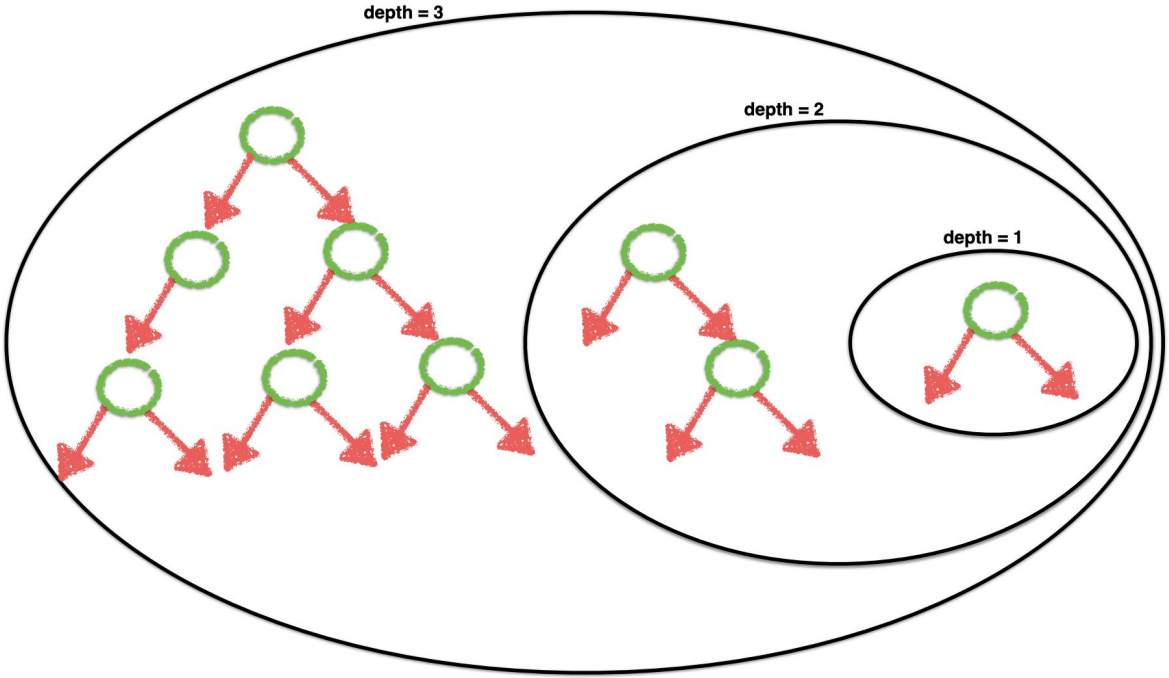



- Neural networks



# Model Complexity: Decision Trees

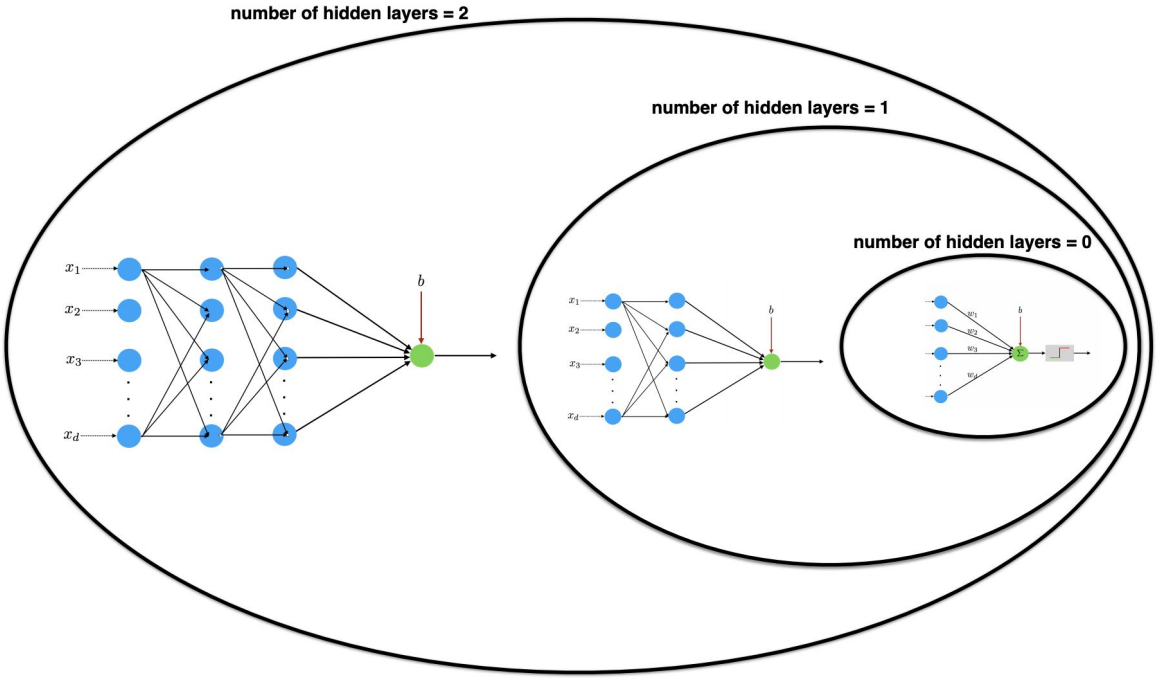
**The richness (complexity) of a model is the function space it can represent!**




By adding to the depth of decision trees, we can learn more complex decision rules!

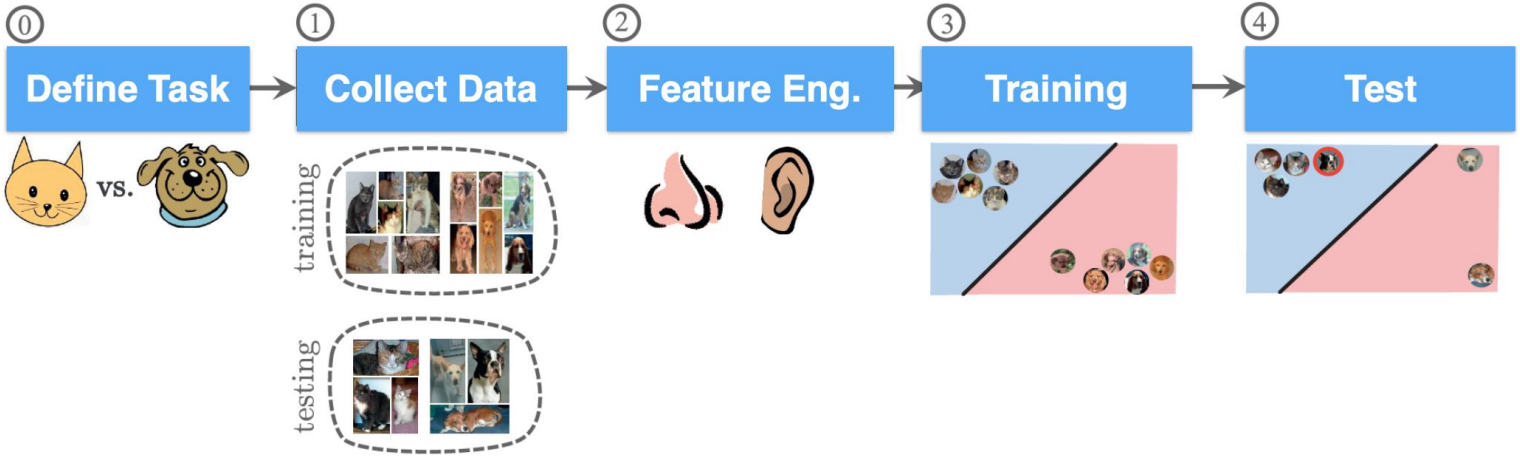
# Model Complexity: Neural Networks

**The richness (complexity) of a model is the function space it can represent!**




By increasing the number of hidden layers, we can learn more complex decision rules!

# Machine Learning Pipeline




- Understanding the problem: What is the task we want to teach a computer to do?
- Collect data: Gather data for training and testing sets. The larger and more diverse the data the better.
- Feature engineering: What kind of features best describe the data?
- Learning and optimization: Tune the parameters of an appropriate model on the training data using numerical optimization
- Test and evaluation: Evaluate the performance of the trained model on the testing data.

Miscellaneous analysis of models and repetition

# Best and good models

Question 1: what is the “BEST” machine learning model?

- The model that gets lower generalization error than all other models.

Question 2: which models always do better than random guessing?

- Models with lower generalization error than random for all problems.

No free lunch theorem:

- There is NO “best” model achieving the best generalization error for every problem.
- If model A generalizes better to new data than model B on one dataset, there is another dataset where model B works better.

In most competitions (e.g. Kaggle), mostly **an ensemble of different methods** is the winner!

# End of story

Suppose your learning algorithm's accuracy for food allergy problem is not yet good enough, what would you do?

- Get more data
- Collect a more diverse training set. For example, people of more states, race, etc
- Train the algorithm longer, by running more gradient descent iterations.
- Try a more complex model, e.g., neural network with more layers/hidden units/ parameters, or decision tree with larger depth.
- Try a smaller model
- Try adding regularization
- Change the problem definition
- Let’s get more insight about data
- Maybe ML is not suitable for this problem :-)

If you chose poorly, you might waste months. How do you proceed? We need to have a good understanding of key concepts to improve the accuracy.

# Let's do some real stuff: polynomial interpolation

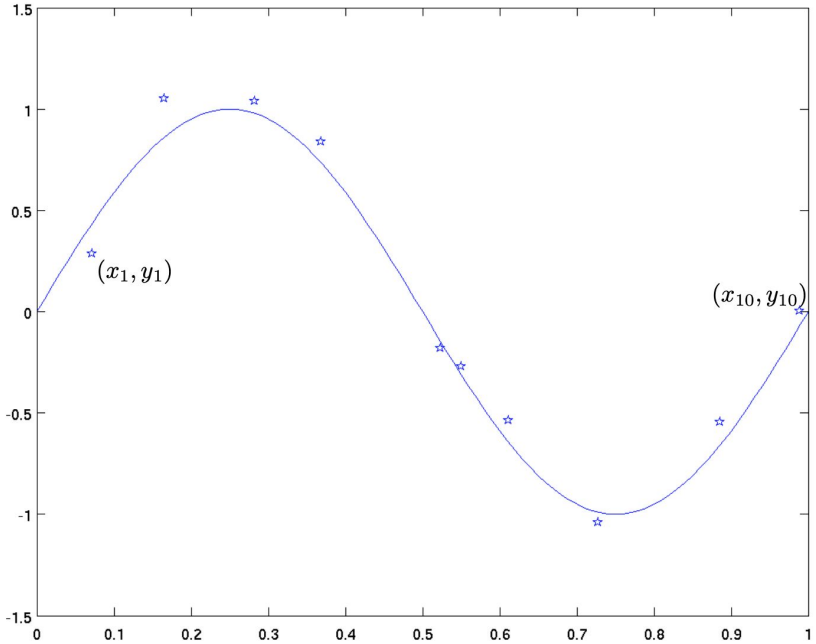
- Let’s conduct some simple regression experiments to investigate these issues!
- The plan:
  - We are going to generate noisy training data from a simple function
  - Then, we will try to fit different models with increasing complexity
  - Each model will be evaluated on some test data generated in the same way the training data is generated

# Toy example training data

- 10 data points were generated as follows:
  - Each $x_i$ is sampled uniformly from the interval [0,1]
  - A noise variable $\epsilon_i$ is sampled uniformly from the interval [-0.2, 0.2]
  - We compute the noisy labels as $y_i = \sin(2\pi x_i) + \epsilon_i$
  - The training data is denoted as

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_{10}, y_{10})\}$$

- Another 10 data points are generated for testing



# Toy example model space

- We will fit polynomials of degree 0, 1 , …. ,9 to the data.
- A polynomials of degree p in one dimension can be represented by:

$$f(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_p x^p$$

$f(x) = w_0$ Fixed horizontal line

$f(x) = w_0 + w_1 x$ A line with slope $w_1$

$f(x) = w_0 + w_1 x + w_2 x^2$ A quadratic function

- By increasing p, the model becomes more complex (richer)
- The parameters we have to fit based on training data are:

$$w_0, w_1, w_2, \ldots, w_p$$

- For now, let's not talk about how we fit the parameters

# Toy example evaluation (loss function)

- We need to define a loss function to evaluate a model (polynomial)
- Let focus on the single example $(x_i, y_i)$
- The prediction of a polynomial function, say p = 2 for this point is

$$f(x_i) = w_0 + w_1 x_i + w_2 x_i^2$$

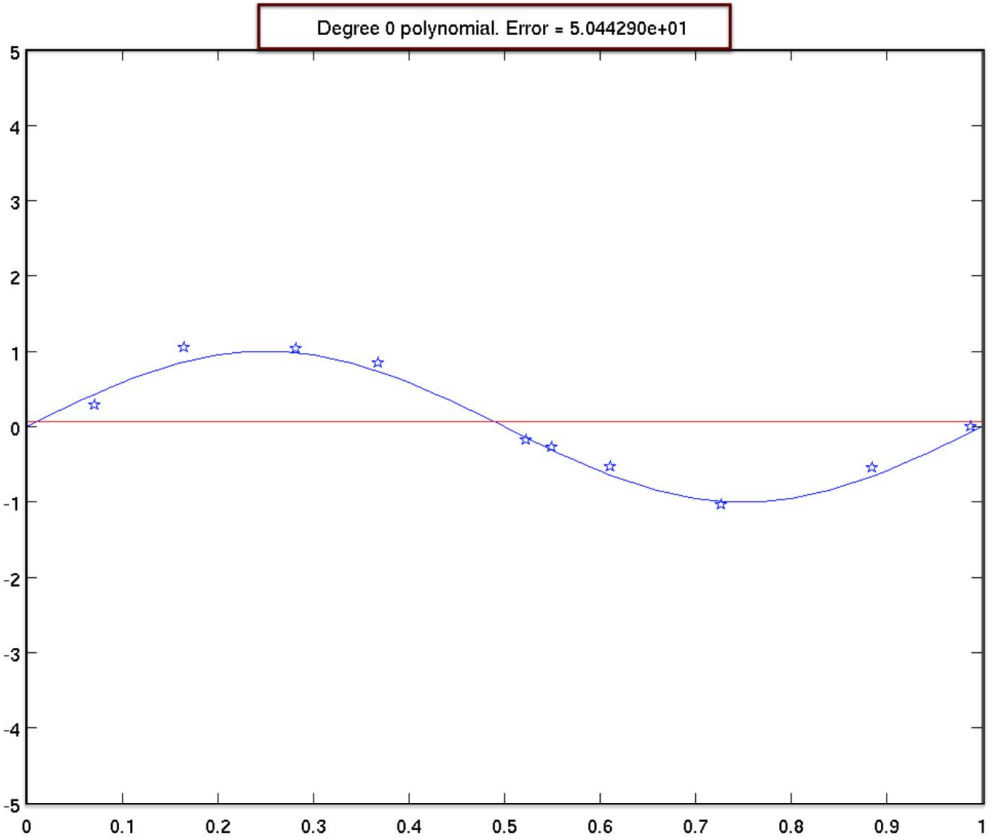
- We use the squared loss to measure the performance of a model:

(actual label - label predicted by model)$^2$

which is:

$$(y_i - f(x_i))^2 = (y_i - w_0 - w_1 x_i - w_2 x_i^2)^2$$

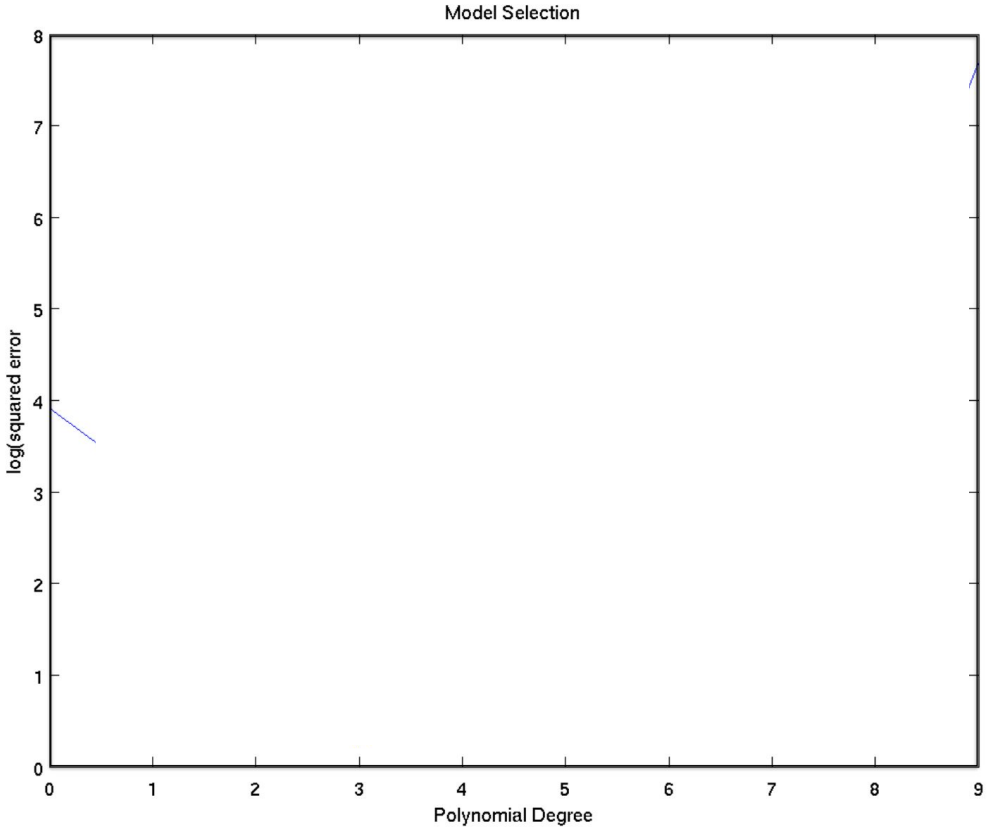
# Degree 0 polynomial



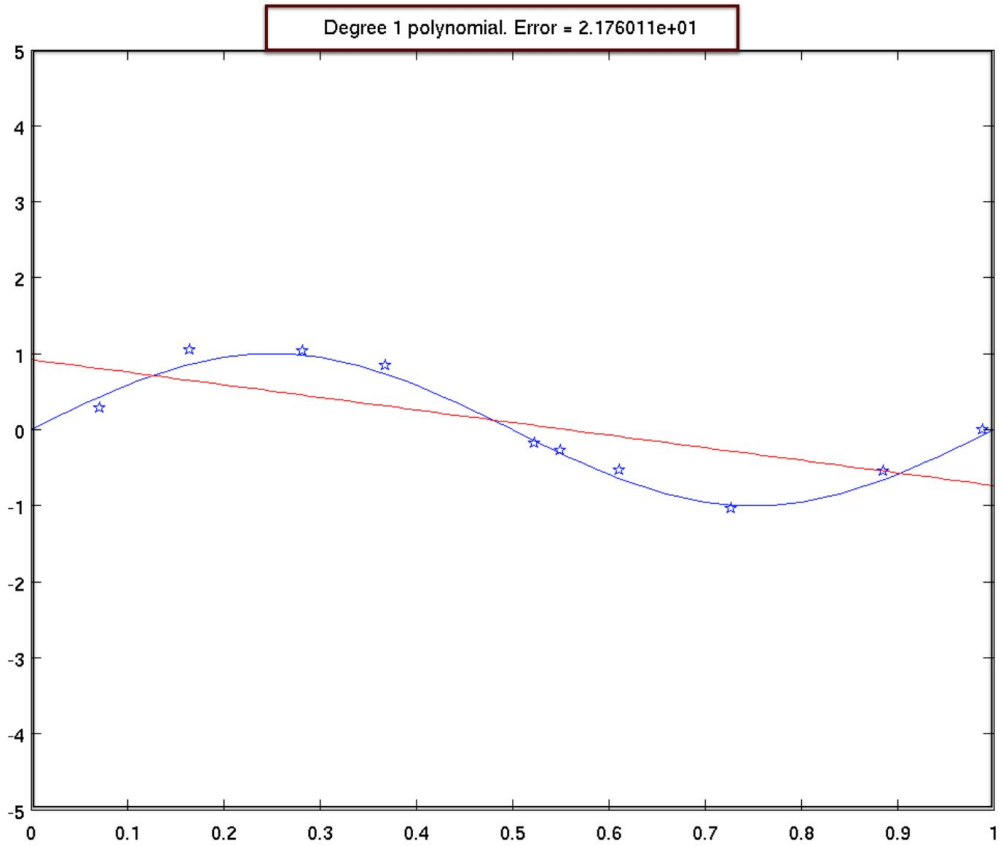
# Learned parameters of Deg 0

| | Deg 0 |
|---|---|
| $w_9$ | 0 |
| $w_8$ | 0 |
| $w_7$ | 0 |
| $w_6$ | 0 |
| $w_5$ | 0 |
| $w_4$ | 0 |
| $w_3$ | 0 |
| $w_2$ | 0 |
| $w_1$ | 0 |
| $w_0$ | 6.6e-2 |

# Generalization Error on Testing Set



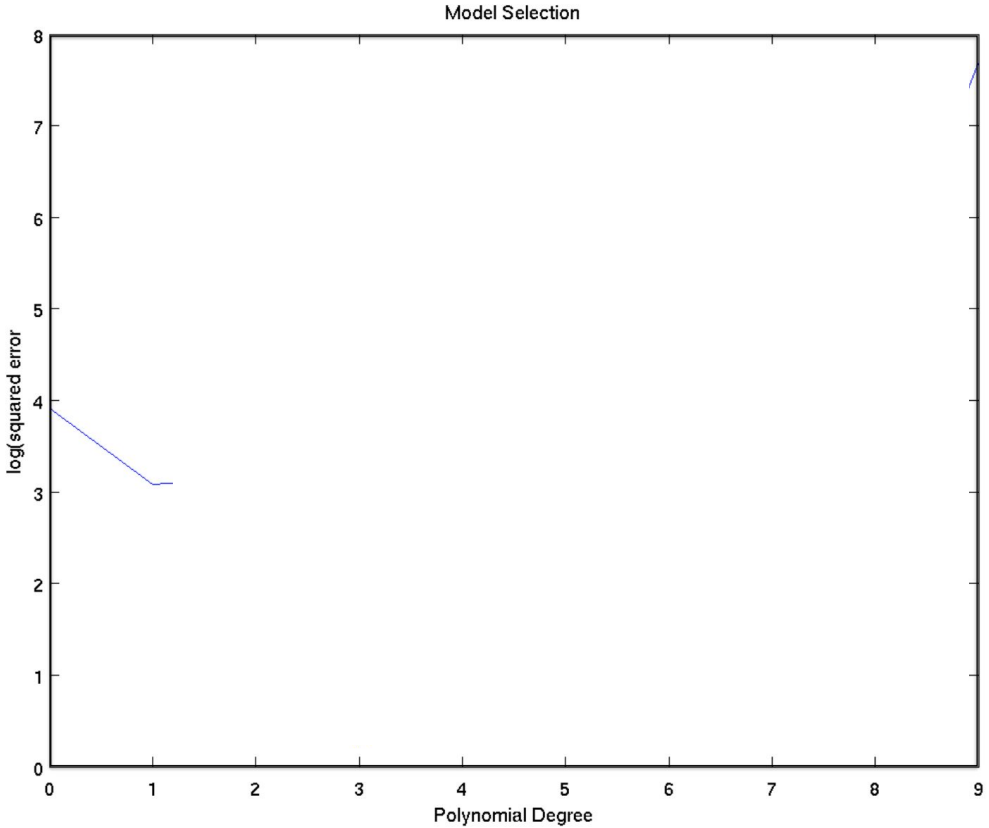
# Degree 1 polynomial



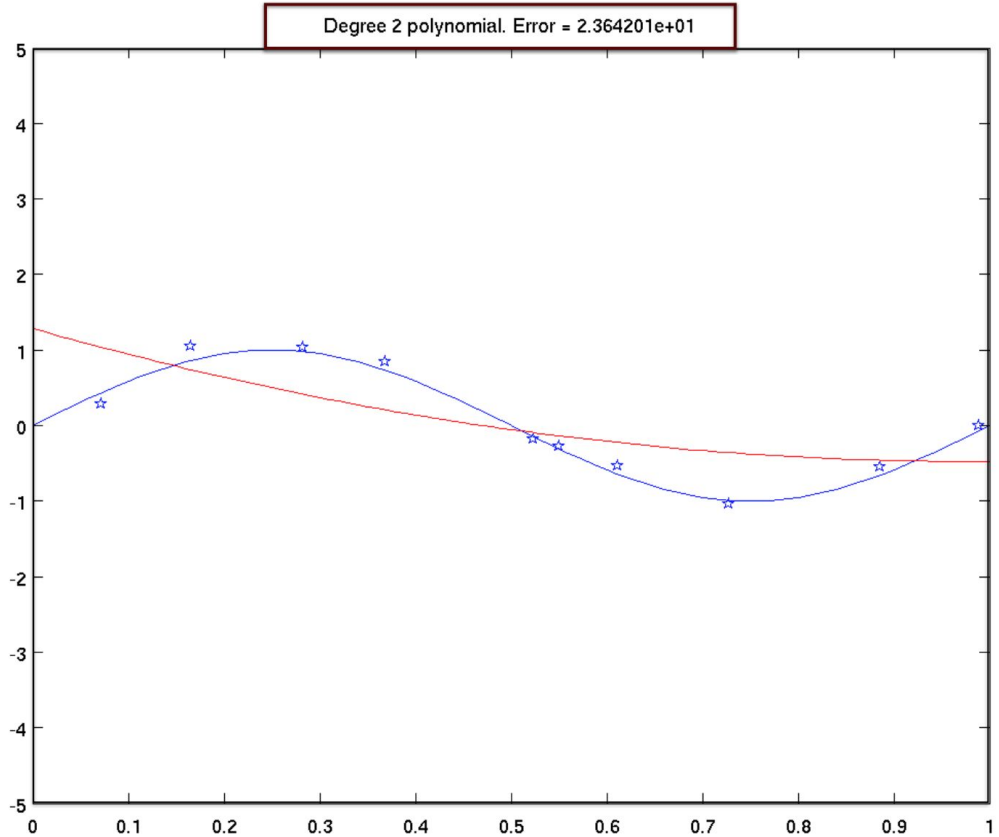
# Learned parameters of Deg 1

| | Deg 0 | Deg 1 |
|---|---|---|
| $w_9$ | 0 | 0 |
| $w_8$ | 0 | 0 |
| $w_7$ | 0 | 0 |
| $w_6$ | 0 | 0 |
| $w_5$ | 0 | 0 |
| $w_4$ | 0 | 0 |
| $w_3$ | 0 | 0 |
| $w_2$ | 0 | 0 |
| $w_1$ | 0 | -1.7e0 |
| $w_0$ | 6.6e-2 | 9.2e-1 |

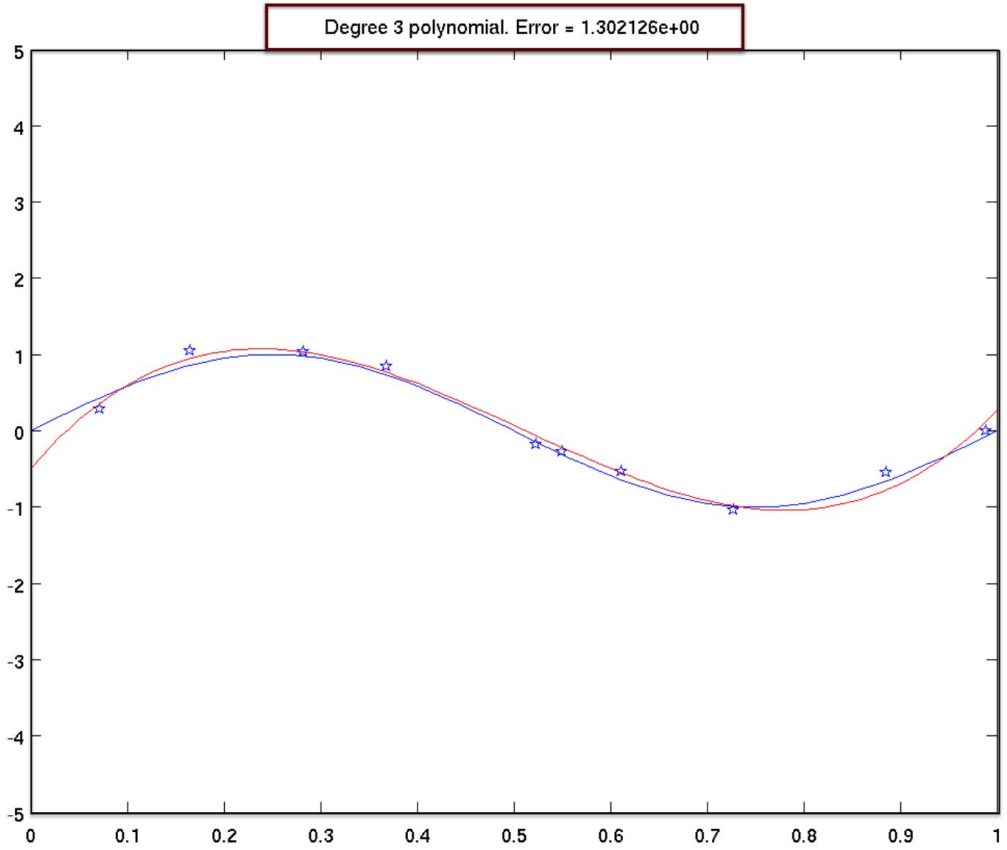
# Generalization Error on Testing Set



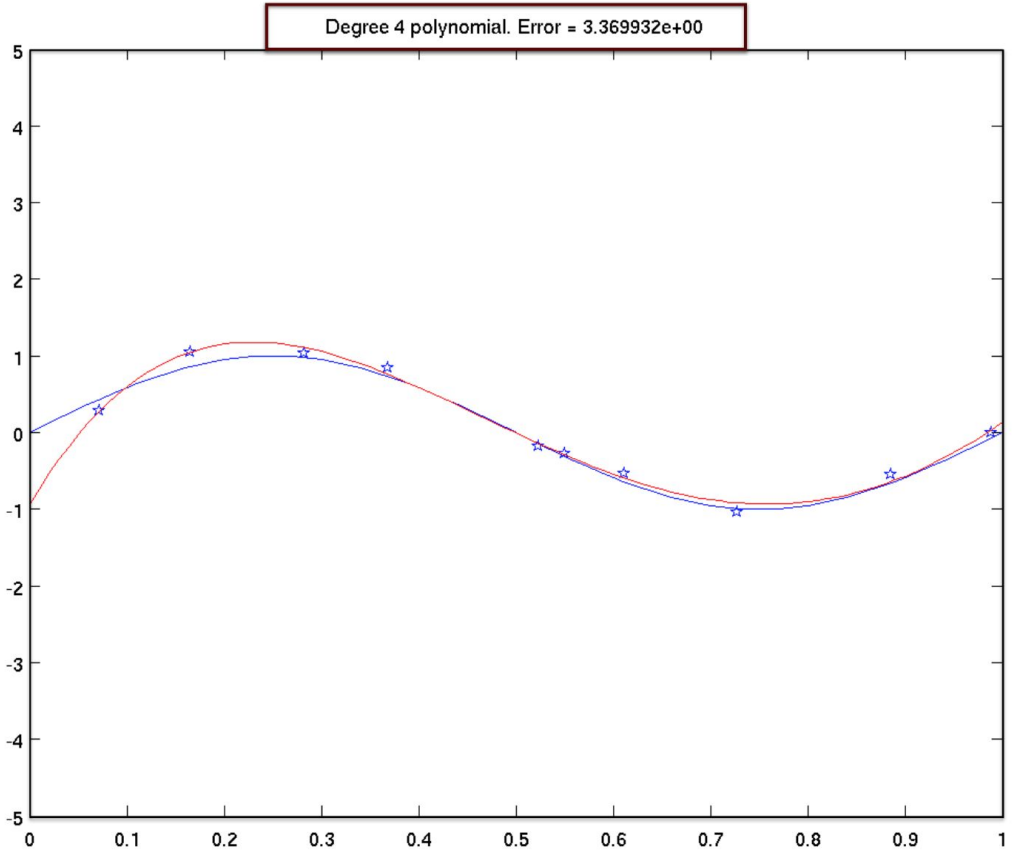
# Degree 2 polynomial



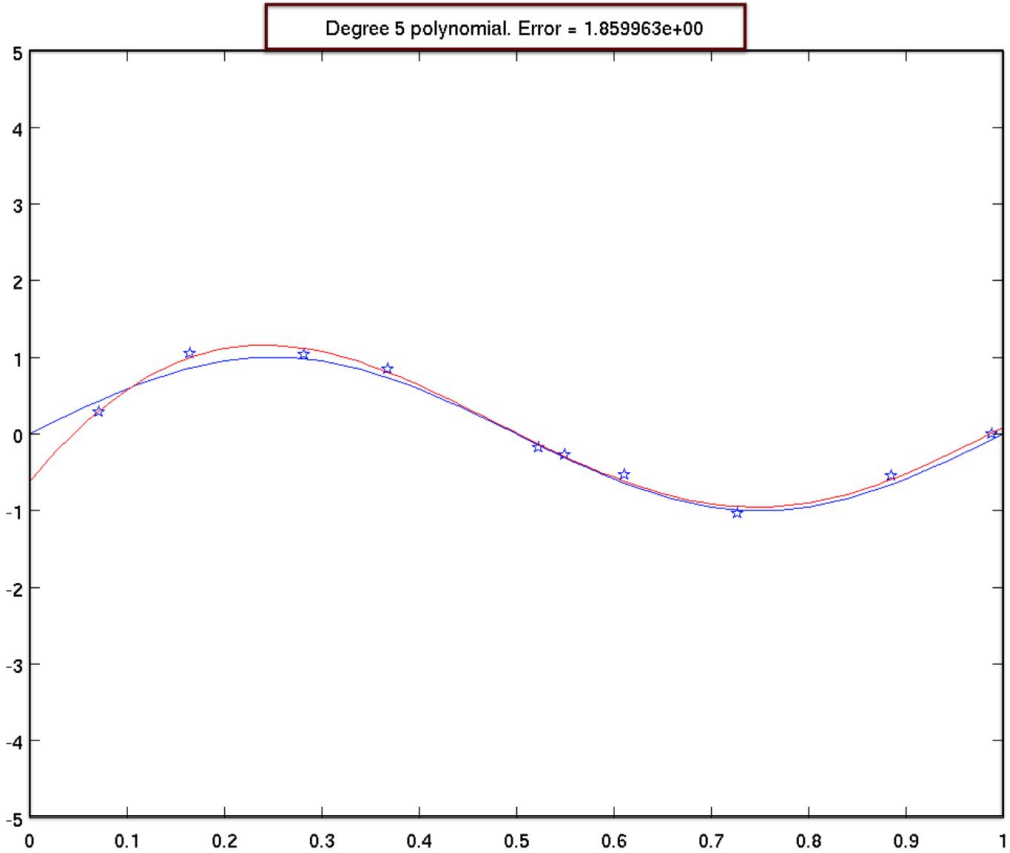
# Degree 3 polynomial



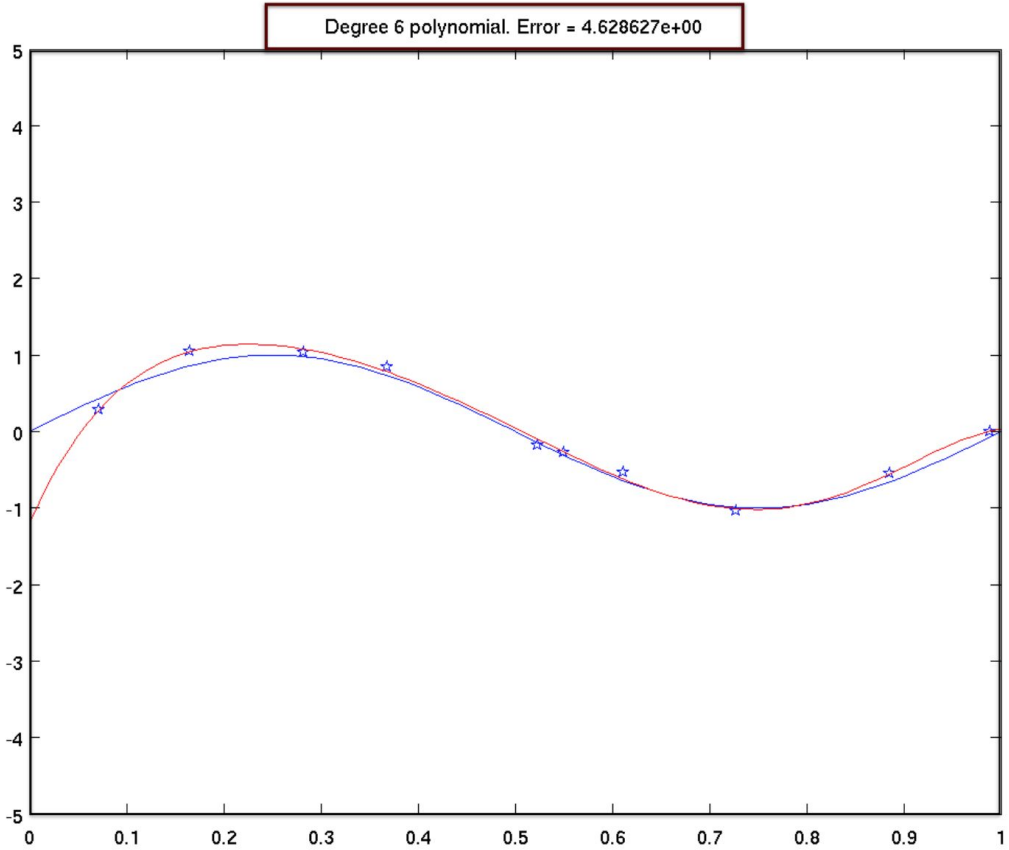
# Degree 4 polynomial



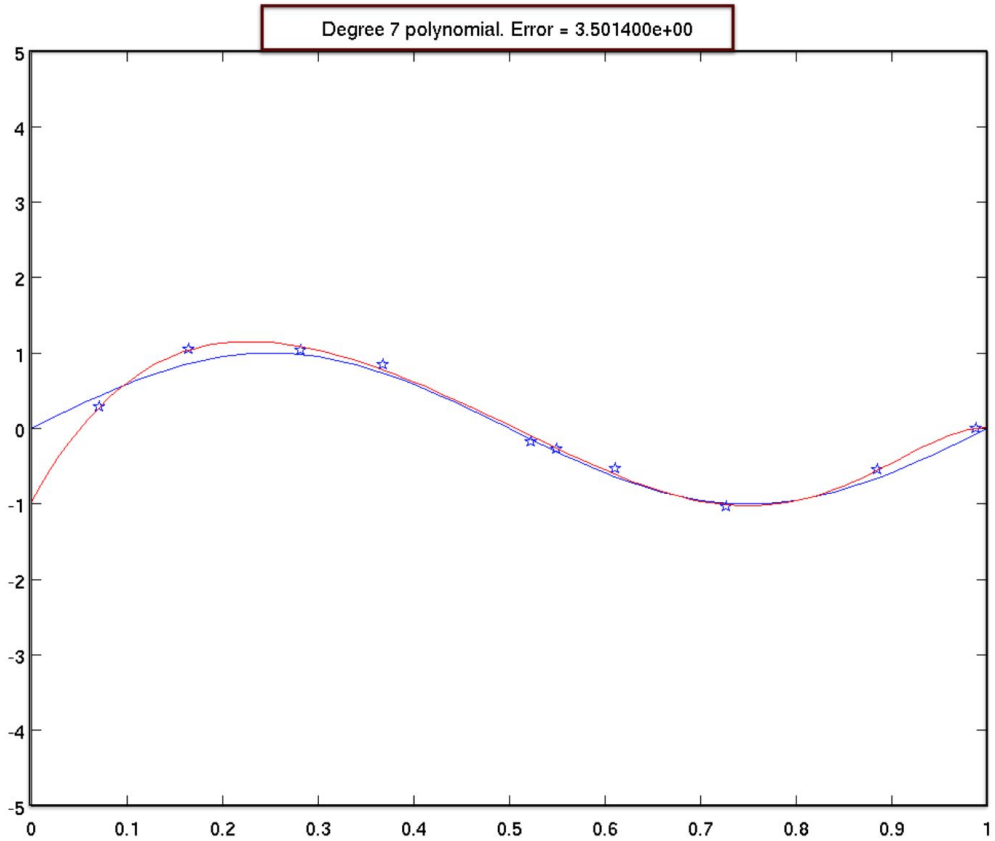
# Degree 5 polynomial



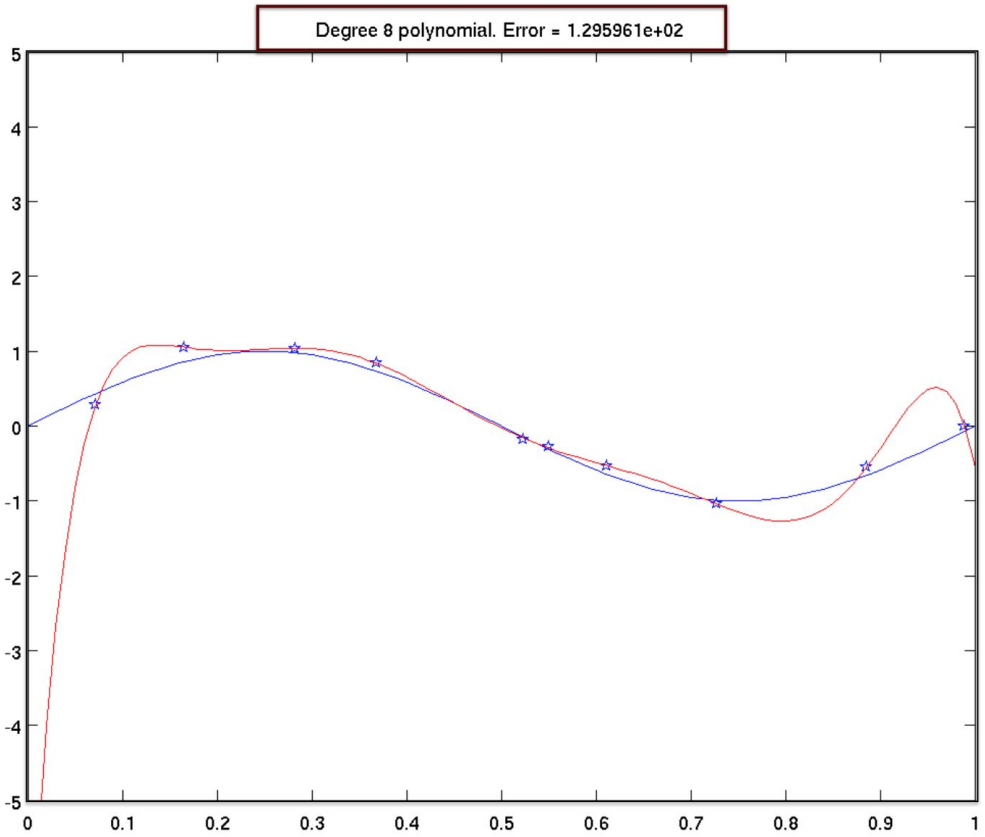
# Degree 6 polynomial



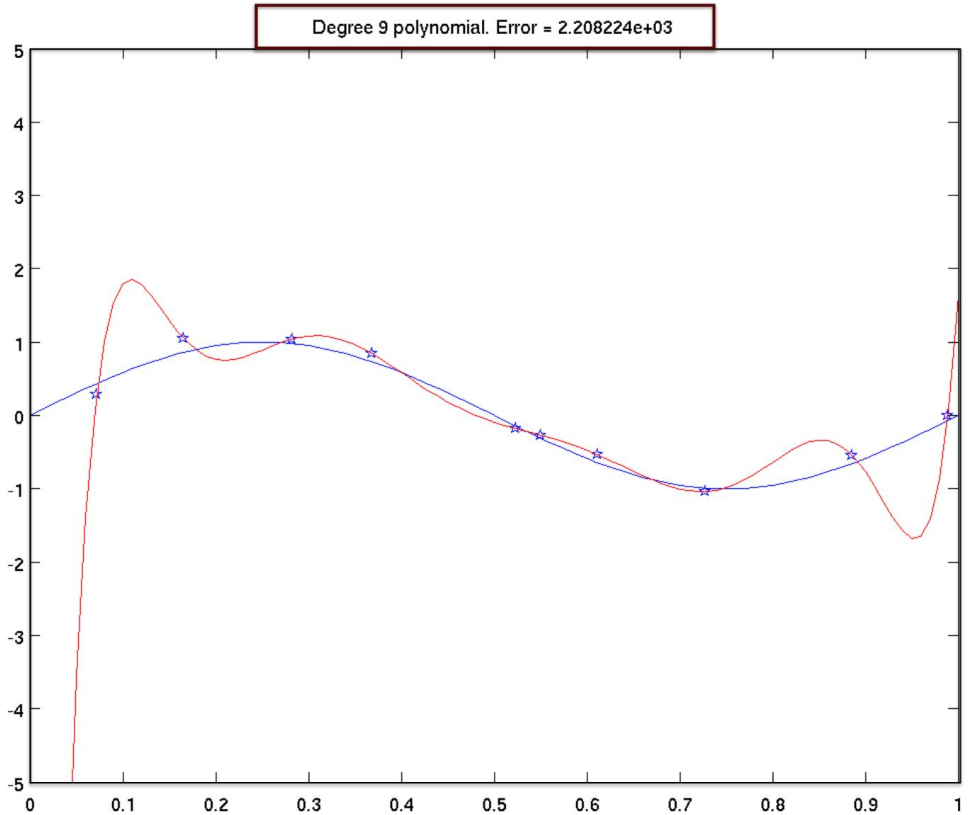
# Degree 7 polynomial



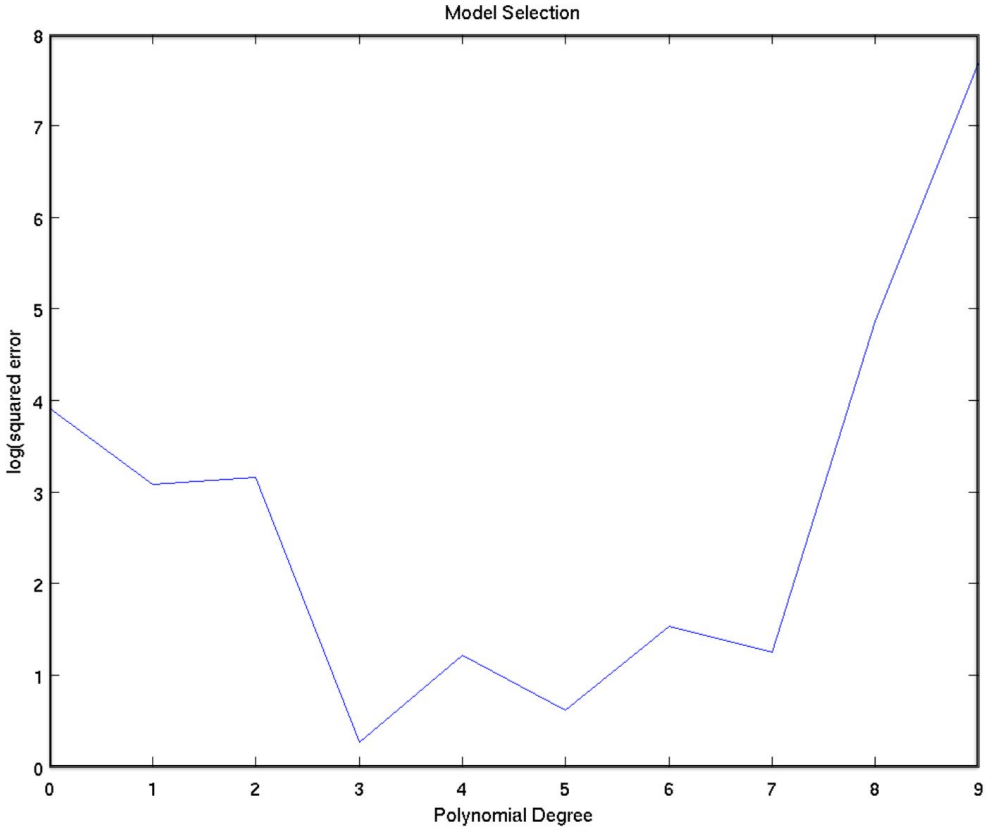
# Degree 8 polynomial



# Degree 9 polynomial



# Generalization Error of all models



# Learned parameters of all models

| | Deg 0 | Deg 1 | Deg 2 | Deg 3 | Deg 4 | Deg 5 | Deg 6 | Deg 7 | Deg 8 | Deg 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| $w_9$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.0e5 |
| $w_8$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1.2e4 | -4.8e5 |
| $w_7$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1.6e2 | 5.0e4 | 9.8e5 |
| $w_6$ | 0 | 0 | 0 | 0 | 0 | 0 | -1.6e2 | 4.3e2 | -8.6e4 | -1.1e6 |
| $w_5$ | 0 | 0 | 0 | 0 | 0 | -3.3e1 | 4.8e2 | -4.0e2 | 7.9e4 | 7.6e5 |
| $w_4$ | 0 | 0 | 0 | 0 | -1.9e1 | 6.9e1 | -5.6e2 | 1.2e2 | -4.2e4 | -3.2e5 |
| $w_3$ | 0 | 0 | 0 | 2.6e1 | 6.6e1 | -1.8e1 | 3.5e2 | 7.0e1 | 1.3e4 | 8.6e4 |
| $w_2$ | 0 | 0 | 1.8e0 | -4.0e1 | -6.8e1 | -3.2e1 | -1.4e2 | -7.7e1 | -2.4e3 | -1.3e4 |
| $w_1$ | 0 | -1.7e0 | -3.6e0 | 1.5e1 | 2.1e1 | 1.5e1 | 2.9e1 | 2.3e1 | 2.3e2 | 1.1e3 |
| $w_0$ | 6.6e-2 | 9.2e-1 | 1.3e0 | -4.9e-1 | -9.4e-1 | -6.3e-1 | -1.2e0 | -9.8e-1 | -7.8e0 | -3.4e1 |

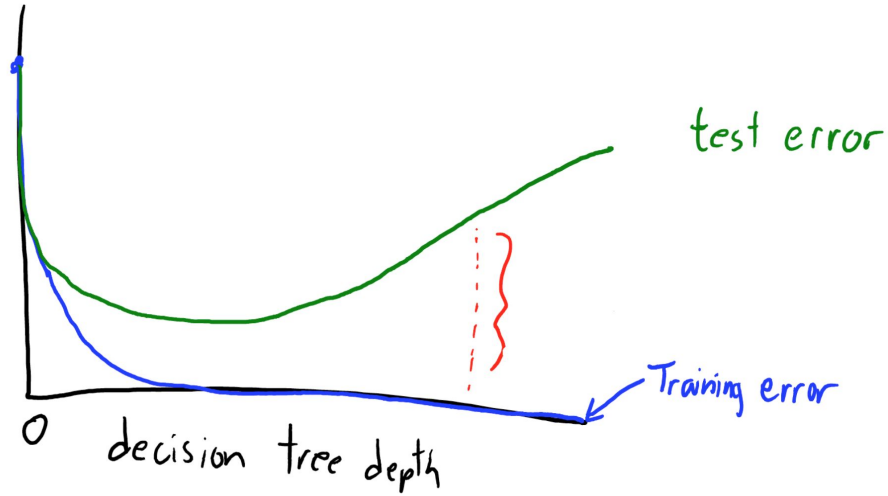
# Analysis

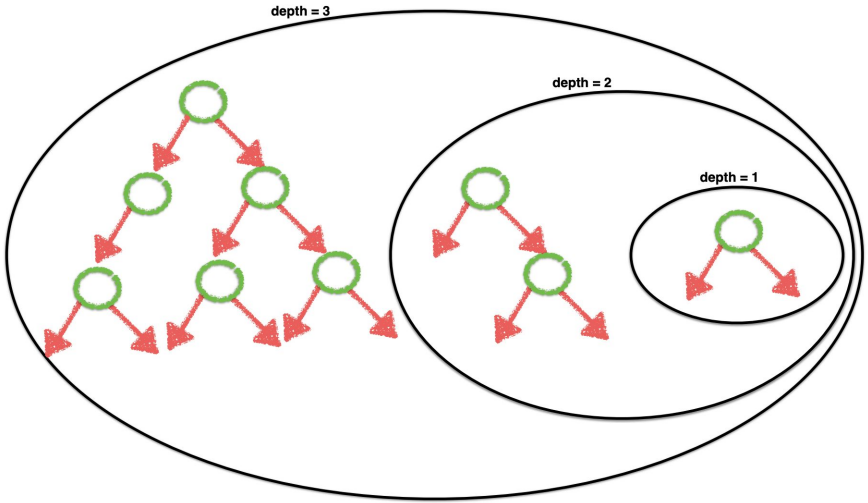
- Degree 9 polynomial can fit all of the training points exactly with zero training error.
- Degree 0 polynomial does not do a good job of fitting training data.
  - It computes the average of the values
- Degree 0 polynomial had **lower generalization error** than degree 9 polynomial.
- Degree 0 polynomials are contained in the class of degree 9 polynomials.
  - Degree 9 polynomials are more complex
  - Degree 9 polynomials have more representation power
- What explains this phenomenon?
  - Degree 9 polynomial fit all of the noisy training points exactly. It is trying to detect patterns in the noise— it is overfitting.

# The Fundamental Trade-Off

Training error vs. test error for choosing depth in decision trees:

- Training error gets better (decreases) with depth. Why? Model gets richer and richer and can approximate more complex functions!
- Test error initially goes down, but eventually increases.

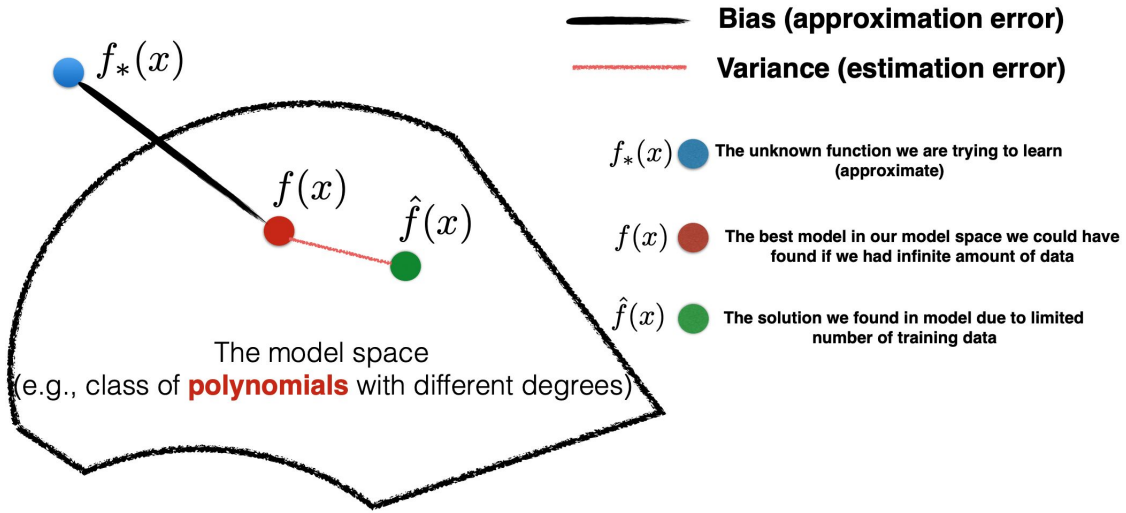


# Overfitting vs Underfitting

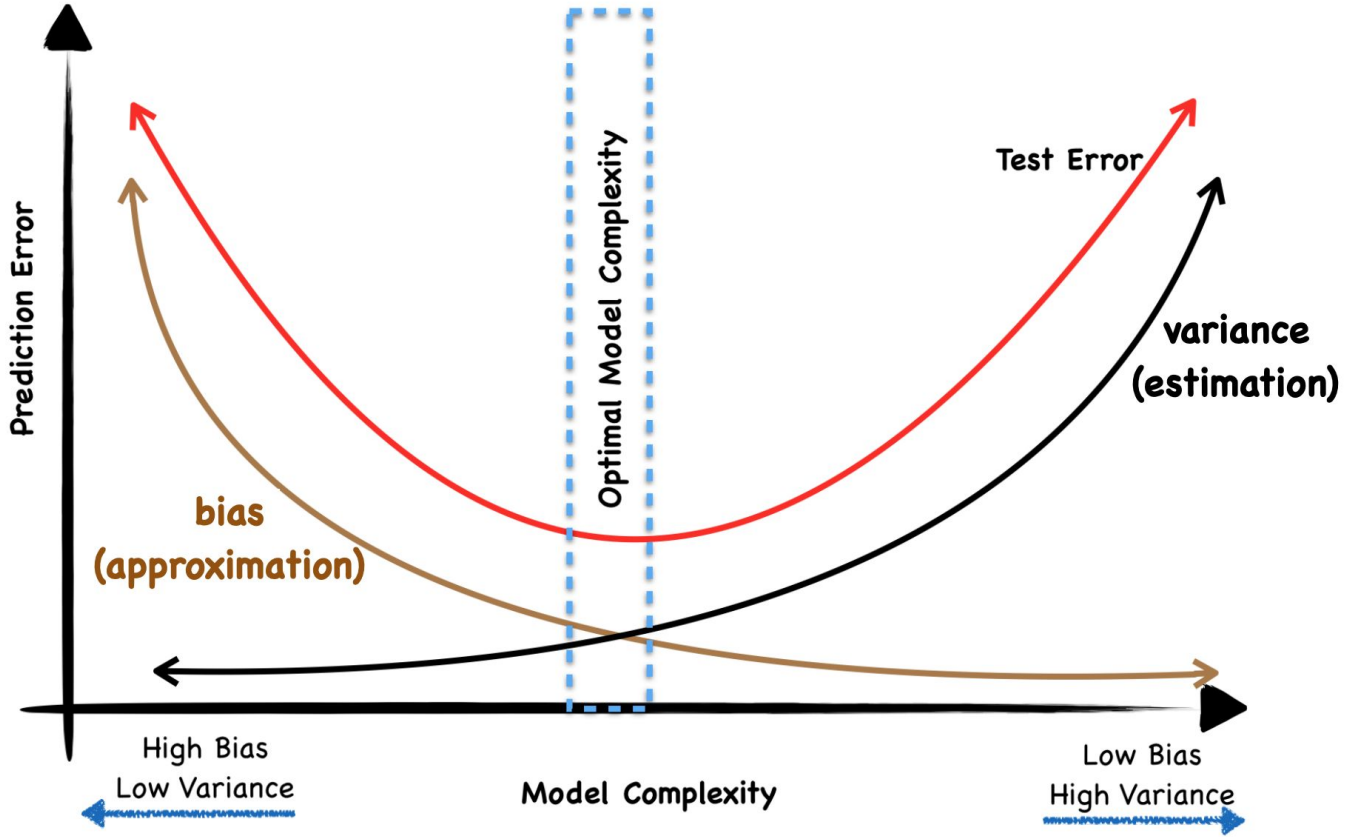
- Zero training error is not necessarily a good thing. There is the danger of **Overfitting**
  - When the parameters of a model are exactly tuned to a particular set of training data, it fails to predict future unseen observations reliably.
  - Happens for example when we learn with a very very complex model so the model fits (learns) the noise.
- **Underfitting** is the opposite: learning with a very very naive and simple model that does not fit data at all.

# Bias-Variance Tradeoff

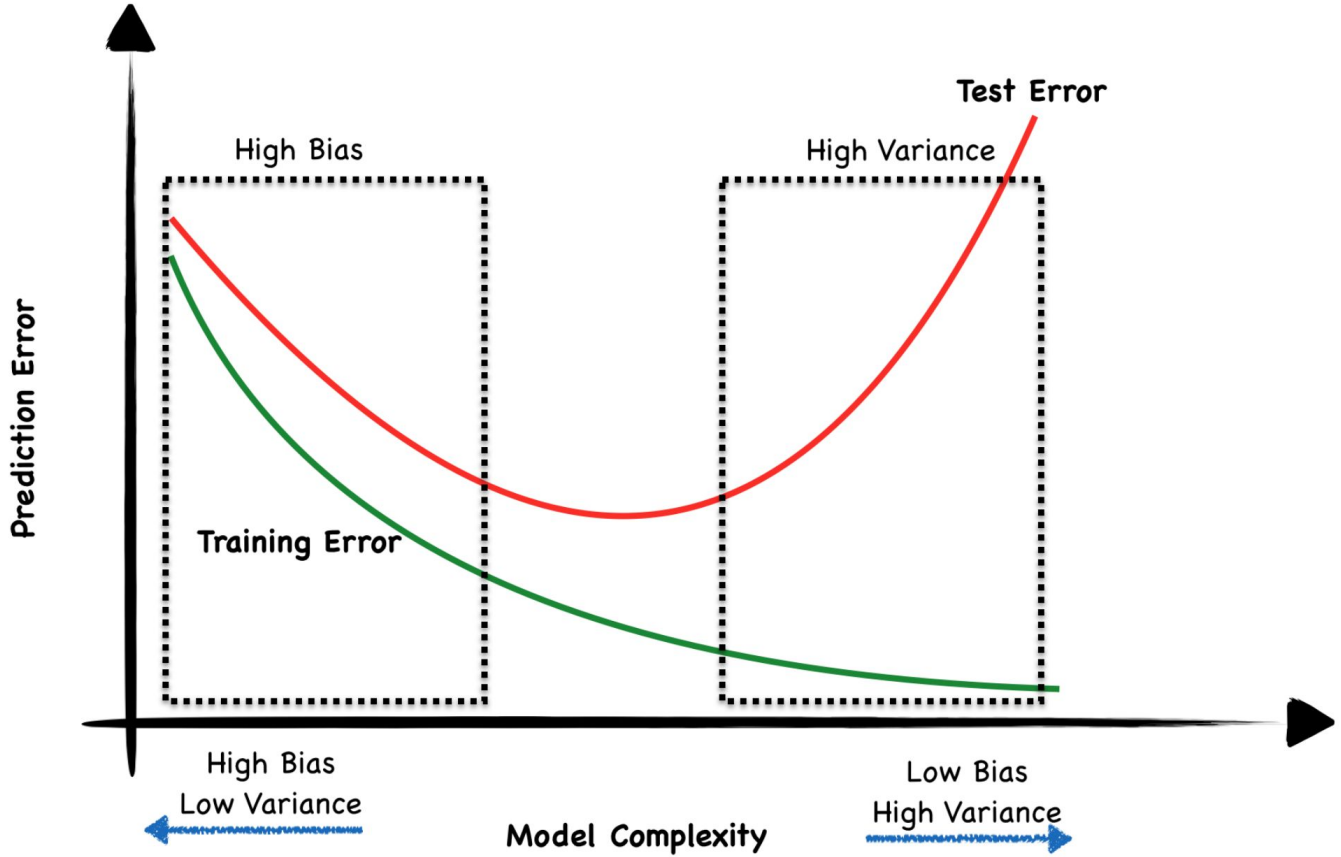



- **Bias**: roughly equal to expressive power: higher bias, lower expressive power
- **Variance**: equal to how well parameters can be estimated: higher variance, higher expressive power
- Bias is due to our assumption about model space.
- Variance is due to finite number of training data!
- Complex models have low bias and high variance
- Simple models have high bias and low variance

# Model complexity and bias-variance



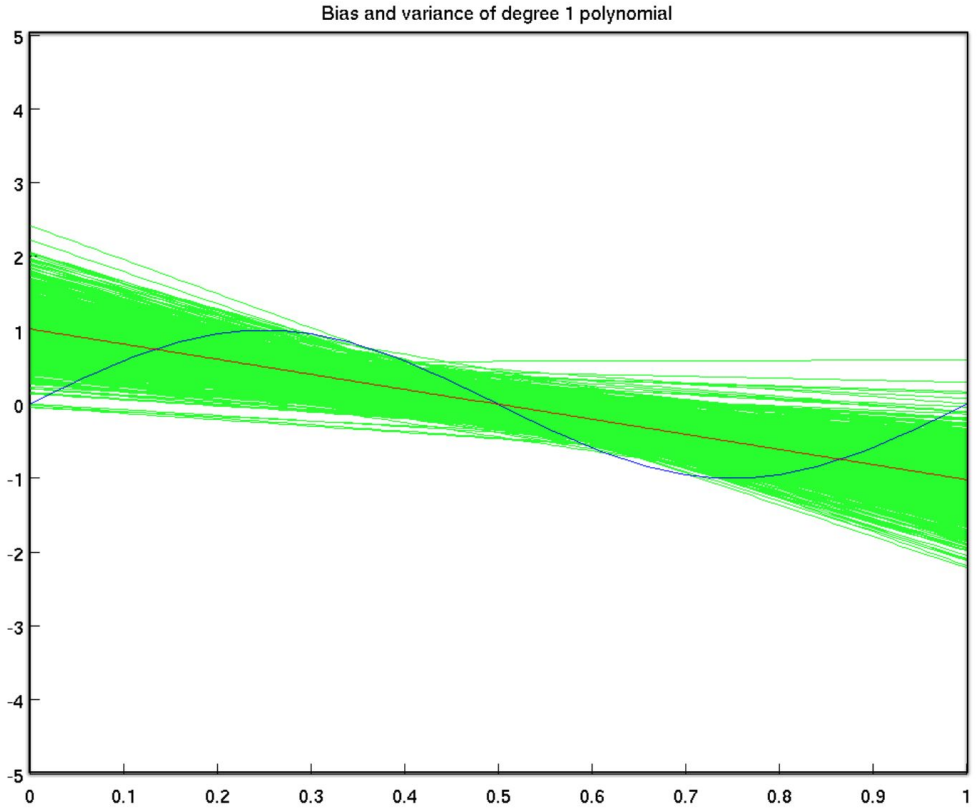
# Model complexity and generalization



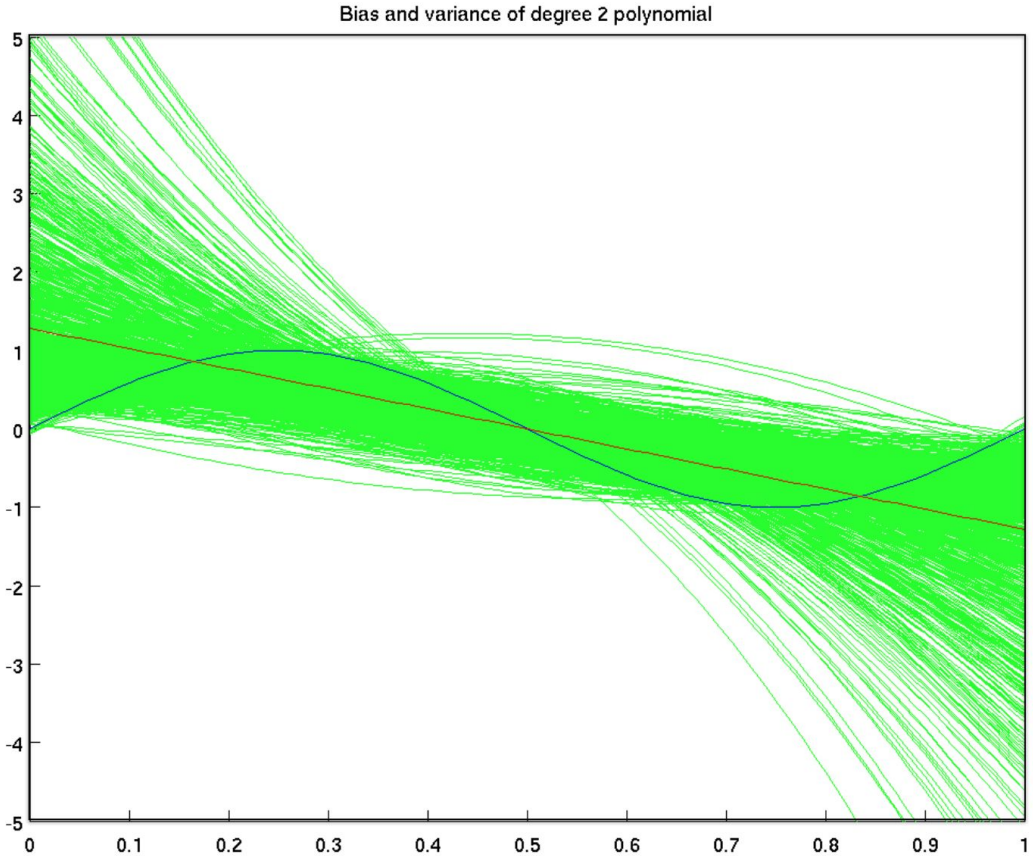
# Bias-variance in polynomial regression

how to get these green lines? Make multiple sets of sample points to train.

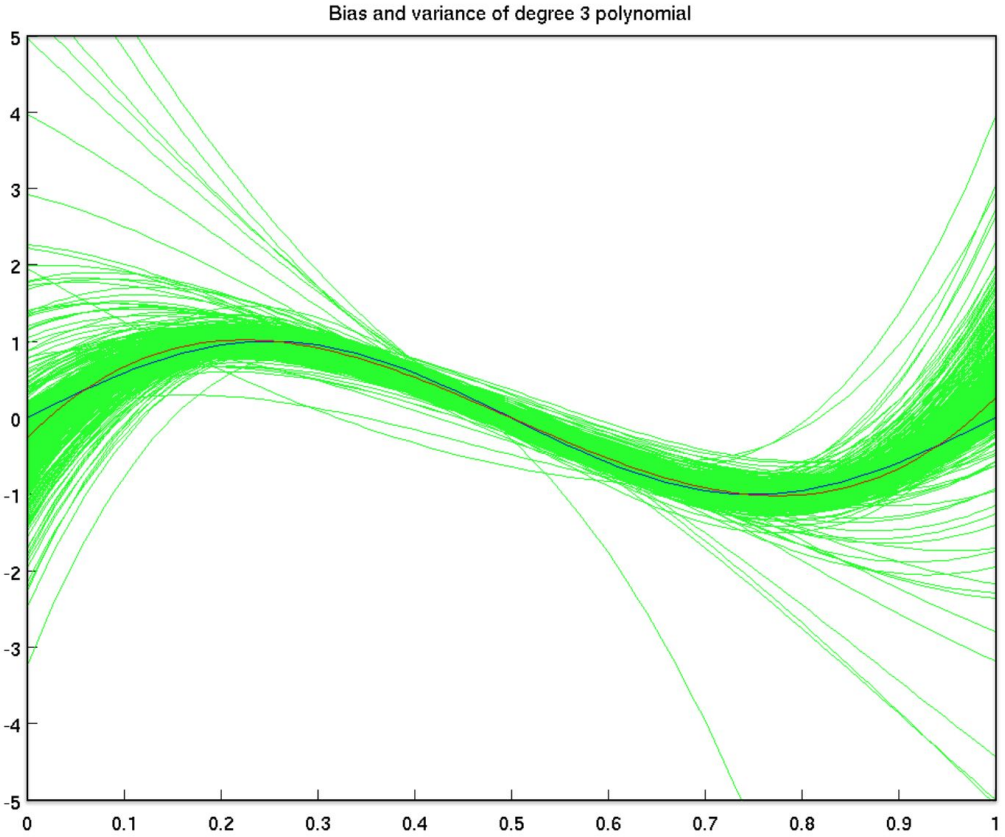
Here we can visualize the variance!



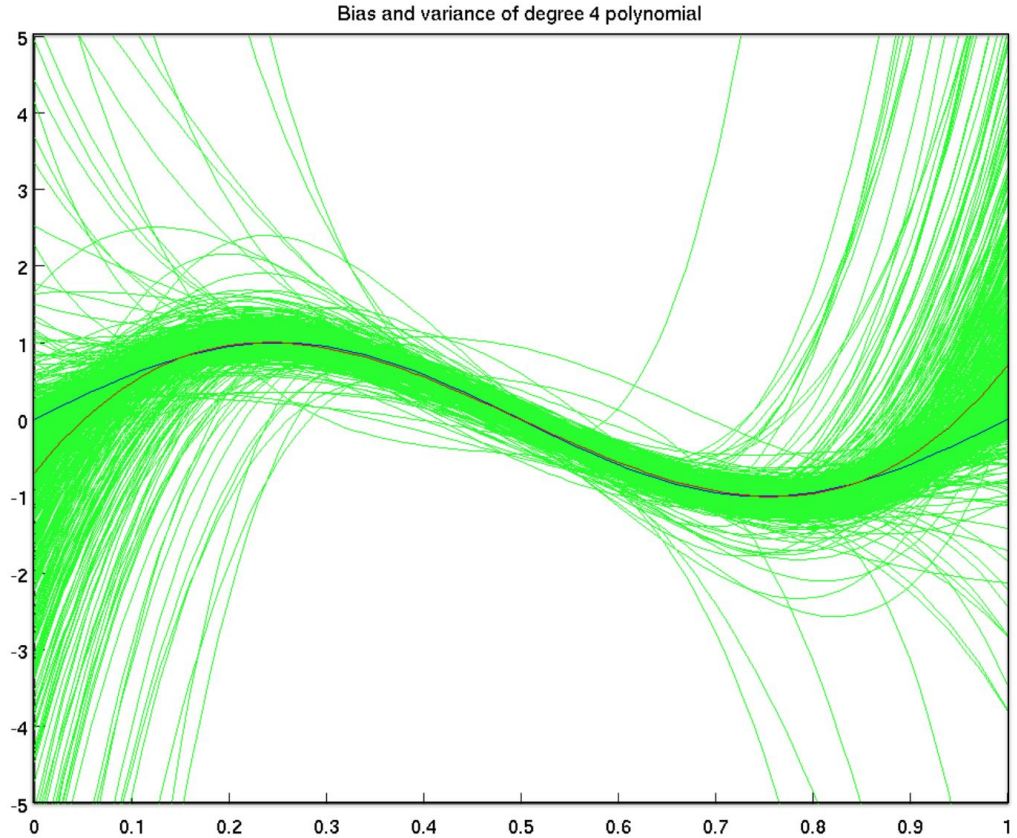
# Bias-variance in polynomial regression



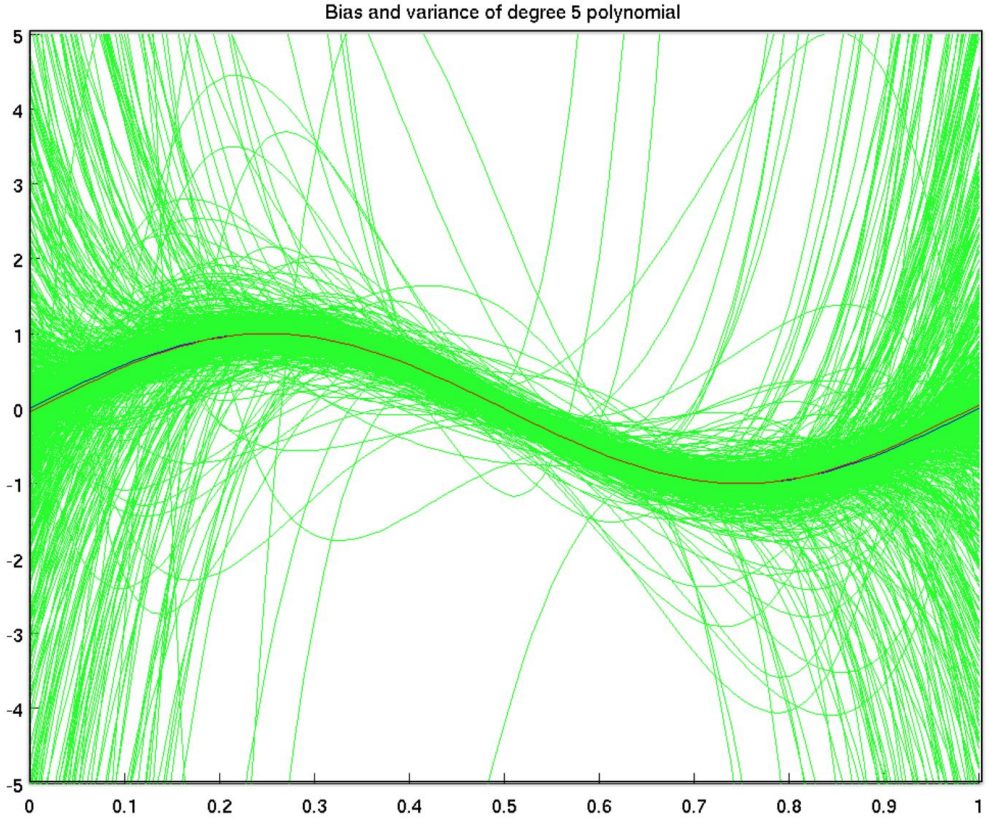
# Bias-variance in polynomial regression



# Bias-variance in polynomial regression

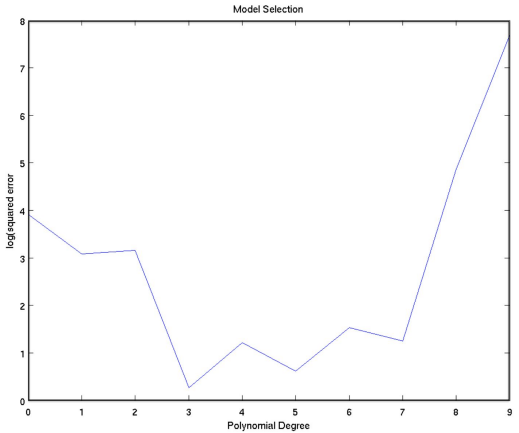


# Bias-variance in polynomial regression



# Takeaways and improving the performance

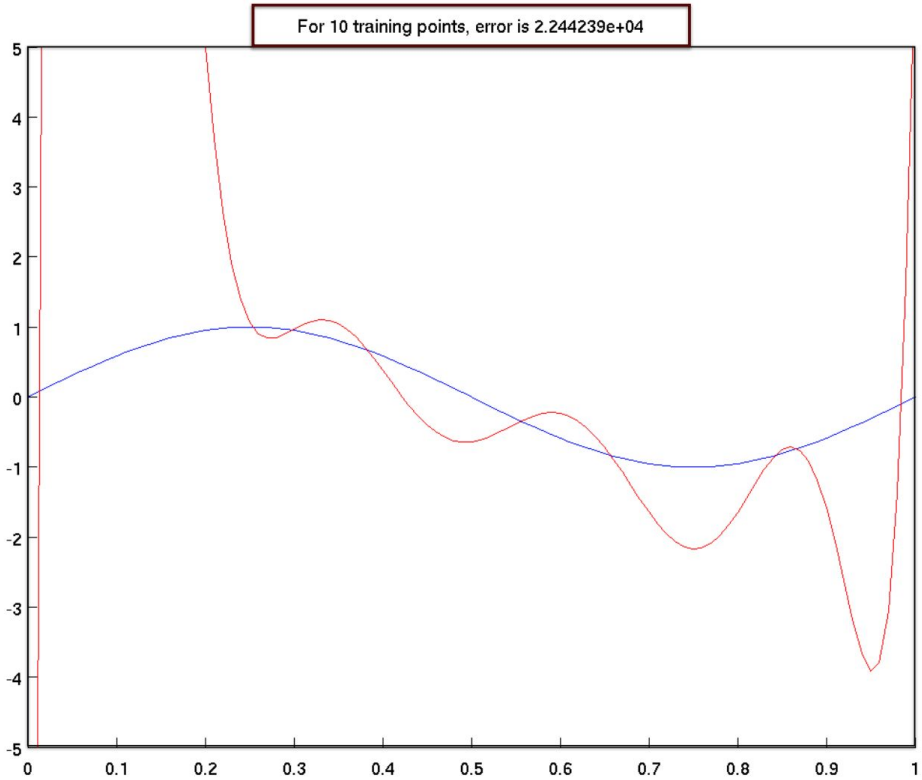
- We can improve performance by restricting degrees of freedom such as number of parameters (Occam's razor).
- Generally, as number of parameters increases
  - Model complexity increases.
  - Bias decreases.
  - Variance increases.




- Generalization error
  - Initially decreases: decrease in bias dominates increase in variance
  - Then increases again: decrease in bias becomes small while increase in variance becomes large.

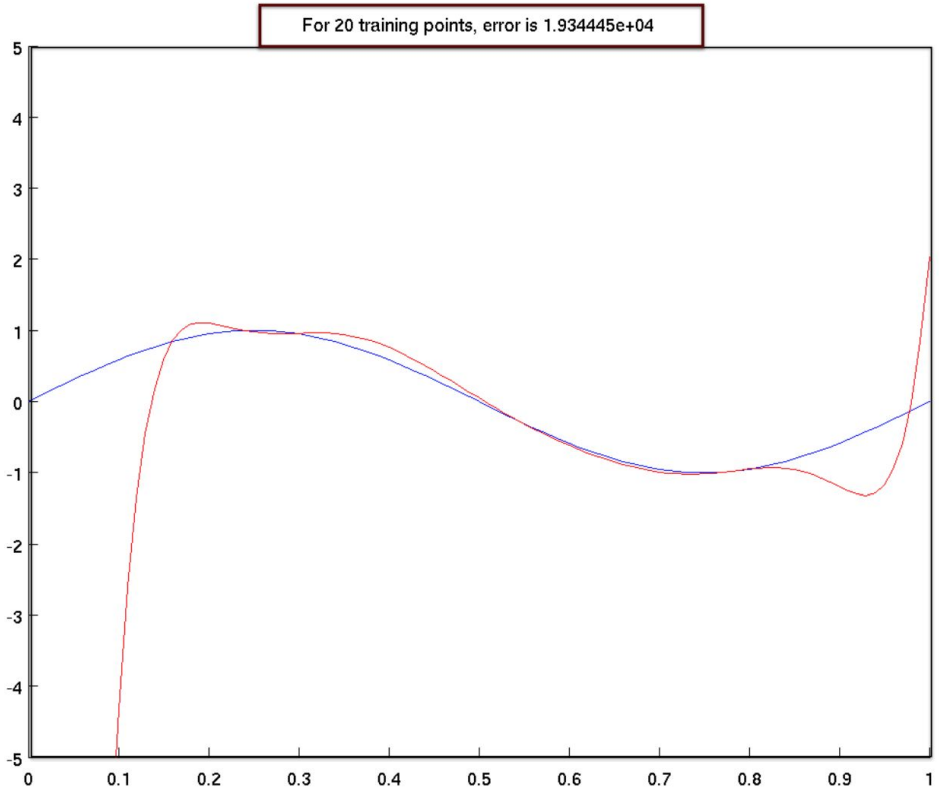
- How else can we improve performance? Let’s increase training data size!

# Polynomial regression and data size



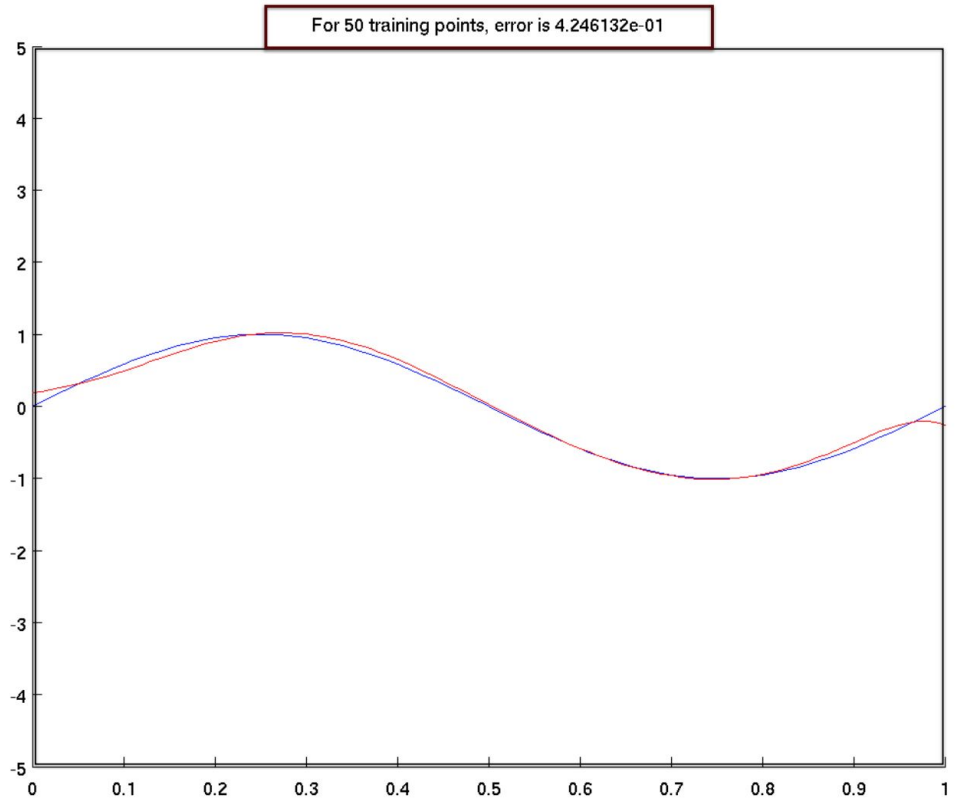

**Degree 9 polynomial with 10 samples**

# Polynomial regression and data size



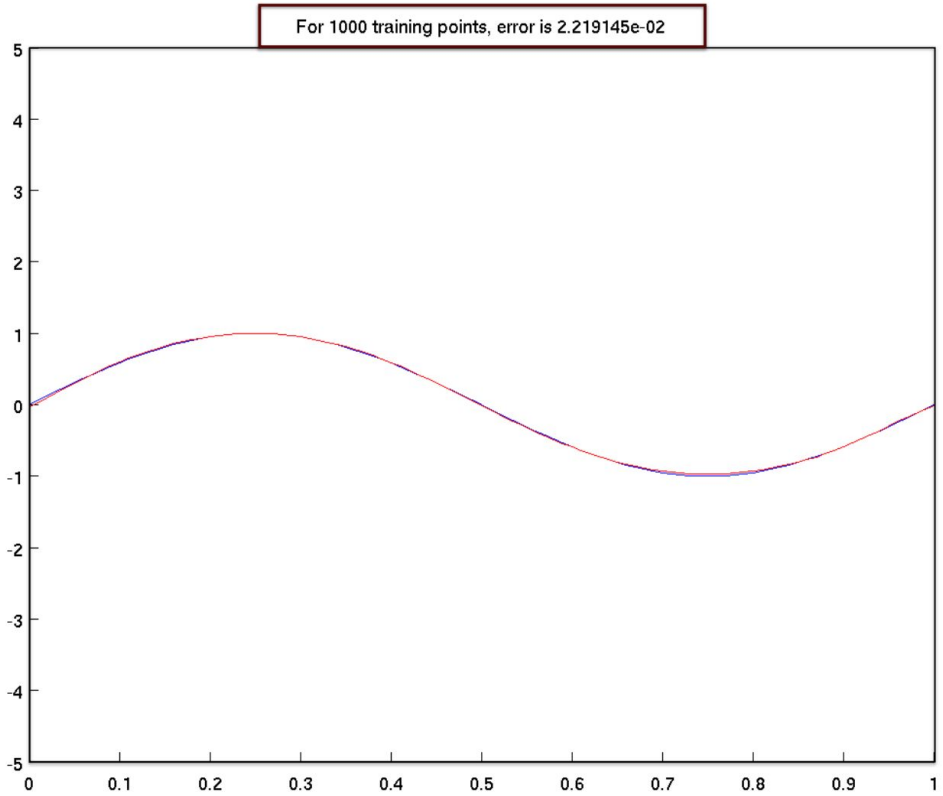

**Degree 9 polynomial with 20 samples**

# Polynomial regression and data size



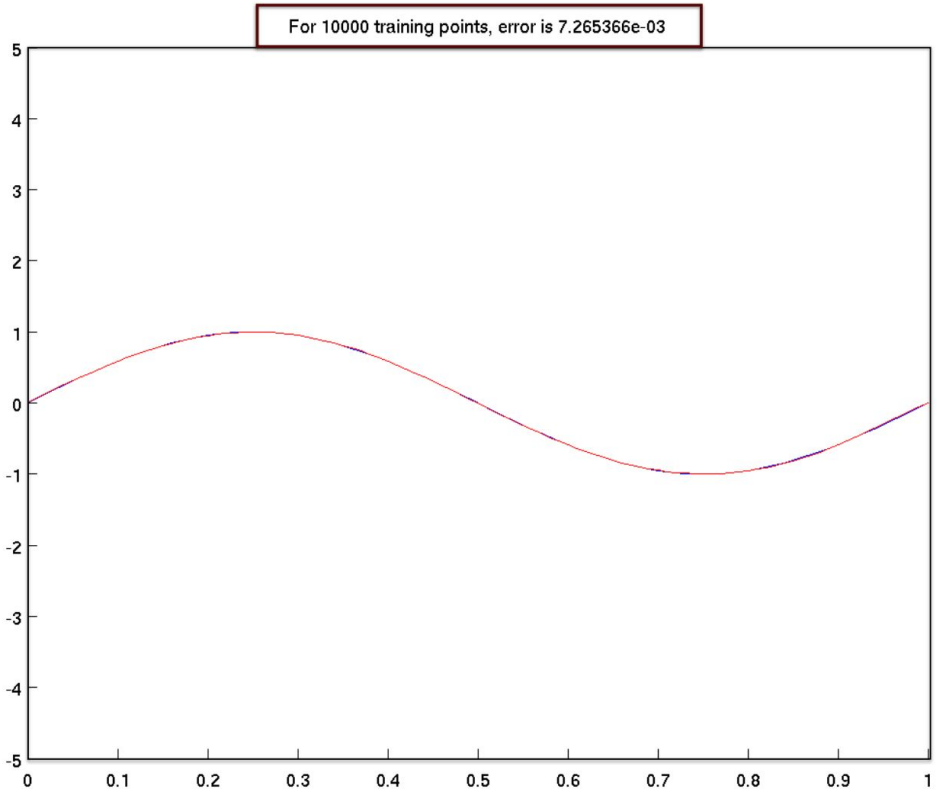

**Degree 9 polynomial with 50 samples**

# Polynomial regression and data size




**Degree 9 polynomial with 1000 samples**

# Polynomial regression and data size



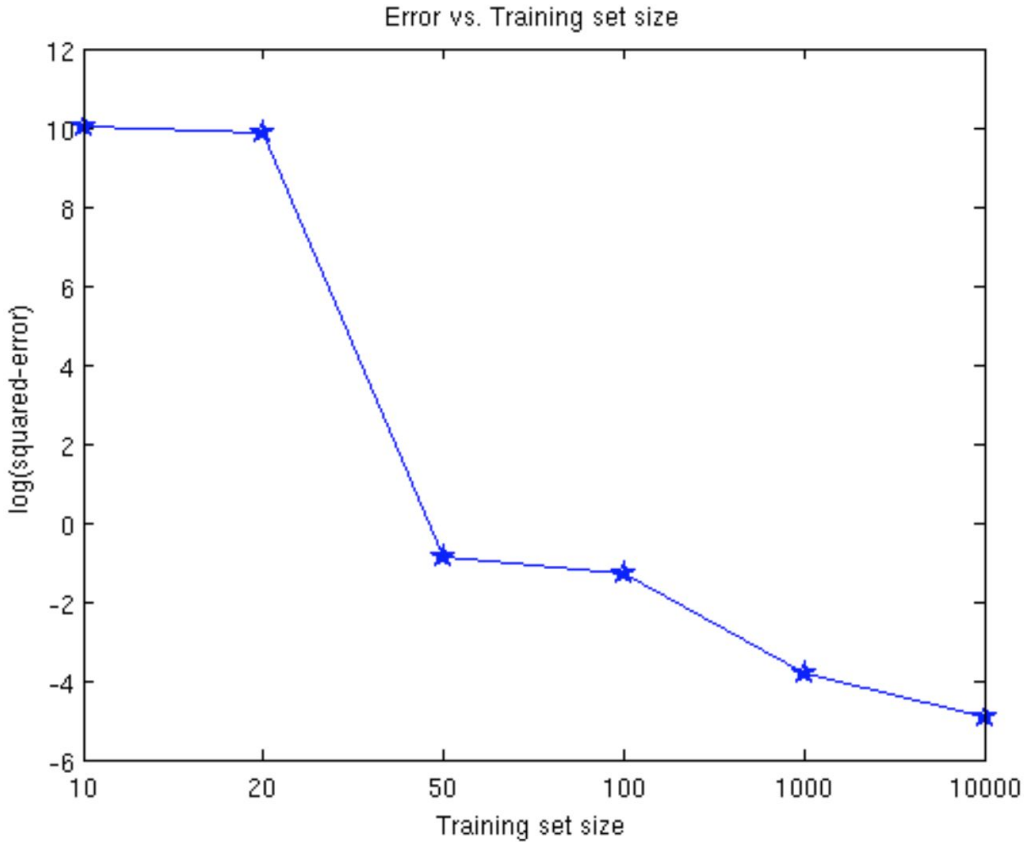

**Degree 9 polynomial with 10000 samples**

# Parameters with different data sizes

| Training size | 10 | 20 | 50 | 100 | 1K | 10K |
|---|---|---|---|---|---|---|
| $w_9$ | 4.18e5 | 6.02e4 | 2.44e2 | -3.18e3 | -1.32e3 | -3.87e2 |
| $w_8$ | -2.03e6 | -3.02e5 | -1.74e3 | 1.46e4 | 5.79e3 | 1.60e3 |
| $w_7$ | 4.21e6 | 6.54e5 | 4.55e3 | -2.81e4 | -1.05e4 | -2.64e3 |
| $w_6$ | -4.84e6 | -8.04e5 | -5.98e3 | 2.95e4 | 1.02e4 | 2.19e3 |
| $w_5$ | 3.38e6 | 6.14e5 | 4.32e3 | -1.86e4 | -5.69e3 | -9.55e2 |
| $w_4$ | -1.47e6 | -3.01e5 | -1.64e3 | 7.24e3 | 1.91e3 | 2.61e2 |
| $w_3$ | 3.88e5 | 9.45e4 | 2.50e2 | -1.76e3 | -3.89e2 | -7.01e1 |
| $w_2$ | -5.77e4 | -1.82e4 | -4.04e0 | 2.33e2 | 2.92e1 | 4.55e1 |
| $w_1$ | 3.88e3 | 1.94e3 | 2.21e0 | -9.14e0 | 5.81e0 | 6.31e0 |
| $w_0$ | -4.14e1 | -8.63e1 | 1.89e1 | 2.82e1 | -2.92e2 | 4.01e3 |

Smaller parameters as we increase the training size

# Error versus training size



# Error versus training size

- We can improve performance by restricting degrees of freedom such as number of parameters.
- We can improve performance by getting more data:
  - Variance decreases.
  - What happens to bias? Not changed because the model is not changed
  - Generalization error decreases.
- How else can we improve performance?
  - There are other ways of restricting parameters.
  - We saw large parameters can be associated with overfitting.
  - We can add penalties for large parameters

# Regularization

- What is regularization?
  A cure for our tendency to fit (get distracted by) the noise

- How does it work?
  By constraining the model so that we cannot fit the noise.
  (e.g., add penalties for large parameters)

- Side effects?
  The medication will have side effects – if we cannot fit the noise, maybe we cannot fit the actual data (the signal)?

Idea: instead of trying different models with different complexities, pick a complex model and penalizes the parameters.

# Regularization

- How to penalize large parameters in training a model?
- Recall that our model is expressed as the following function

$$f(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_p x^p$$

with parameters (p = 9):

$$w_0, w_1, w_2, \ldots, w_p$$

- Let's just limit the extent the parameters can increase, e.g., keep following quantities as small as possible

$$w_0^2 + w_1^2 + \ldots + w_9^2$$

or:

$$|w_0| + |w_1| + \ldots + |w_9|$$

# How to fit regularized model?

- Originally, we find parameters that minimizes training error (the discrepancy between the predictions of our model and actual labels)

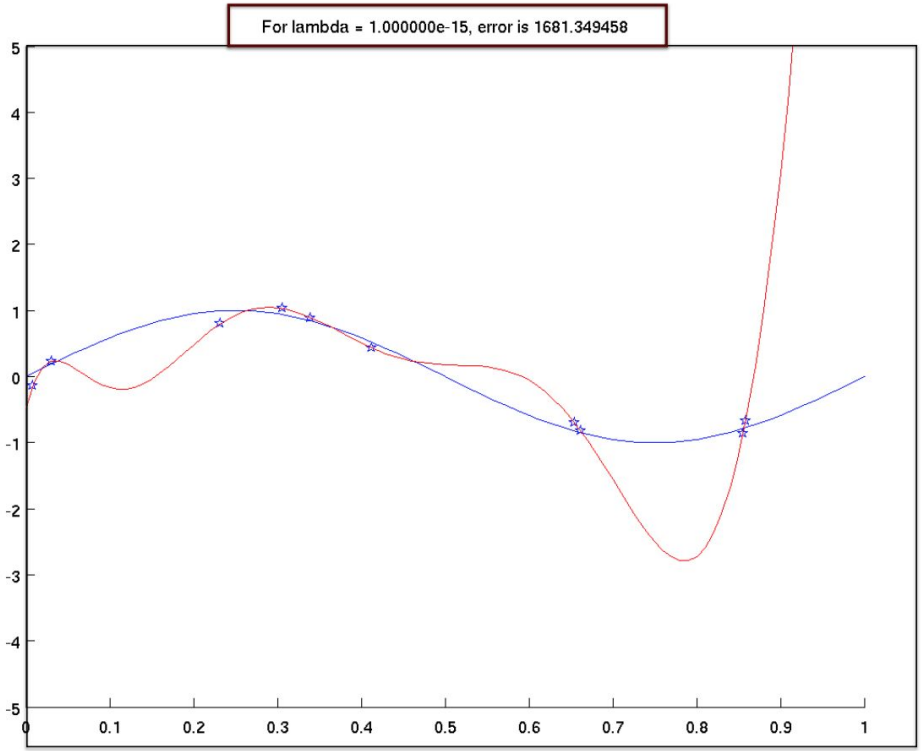
$$\underset{w_0, w_1, \ldots, w_9}{\textbf{minimum}} \quad \textbf{training error}$$

- Now, we find parameters that minimizes training error and penalizes parameters:

$$\underset{w_0, w_1, \ldots, w_9}{\textbf{minimum}} \quad \textbf{training error} + \lambda \left( w_0^2 + w_1^2 + \ldots + w_9^2 \right)$$

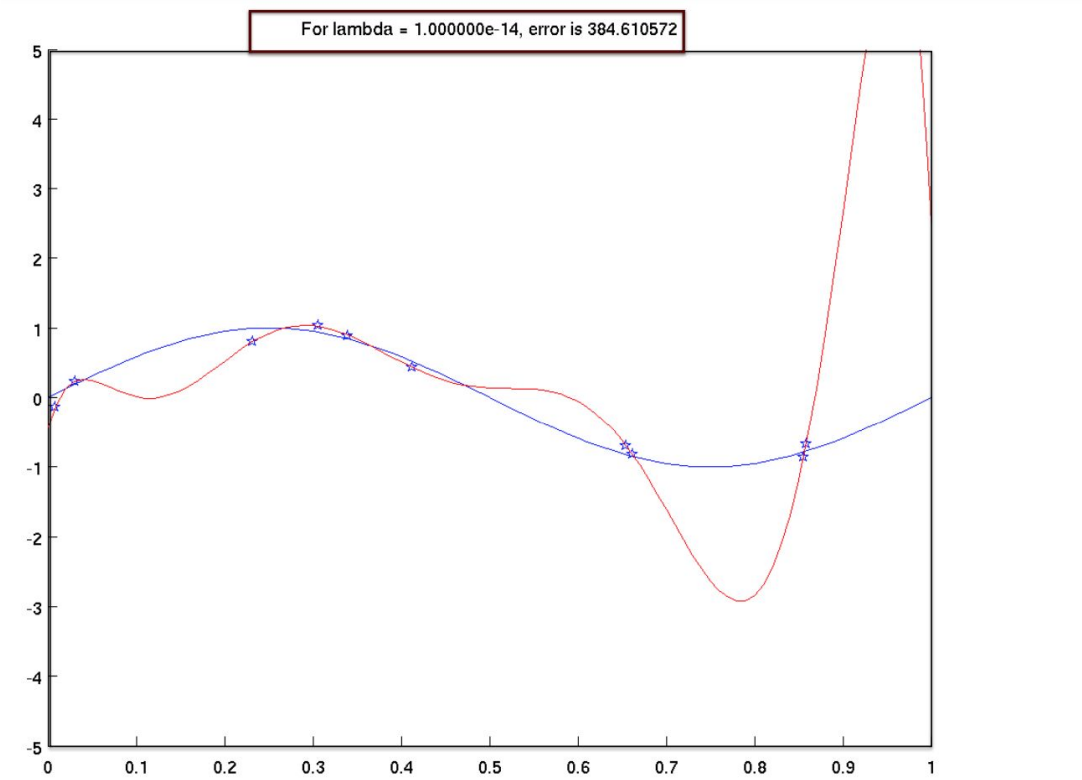
- $\lambda \geq 0$ is the regularization parameter, and controls how aggressive we are in penalizing the model parameters (e.g., $\lambda = 0$ means no penalization)!

# Regularized training



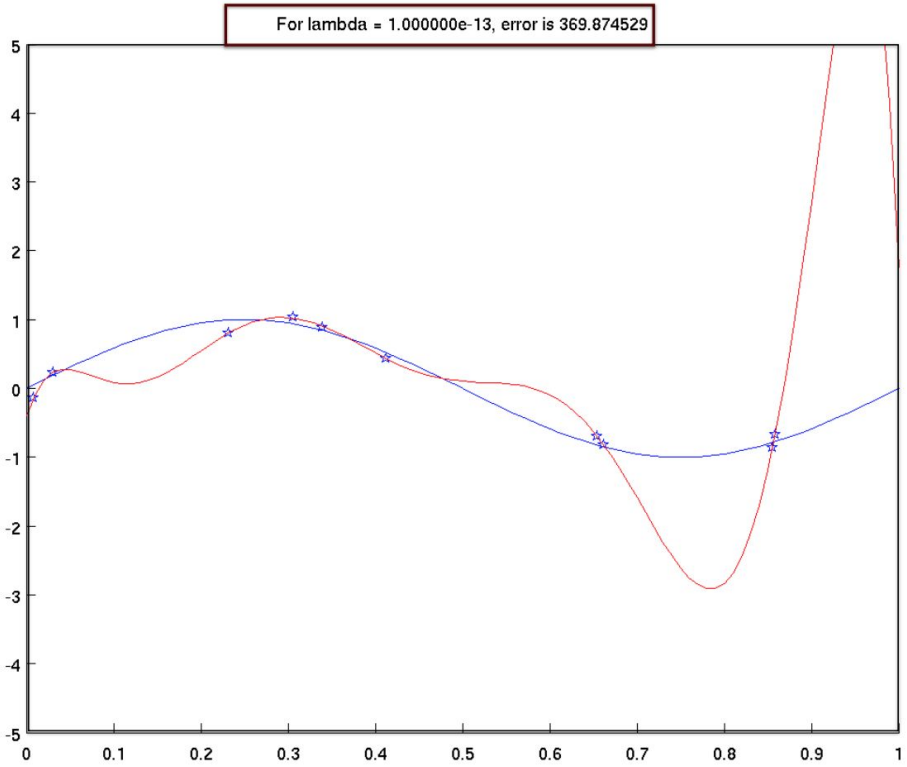

**Degree 9 polynomial with regularization parameter** $\lambda = 10^{-15}$

# Regularized training



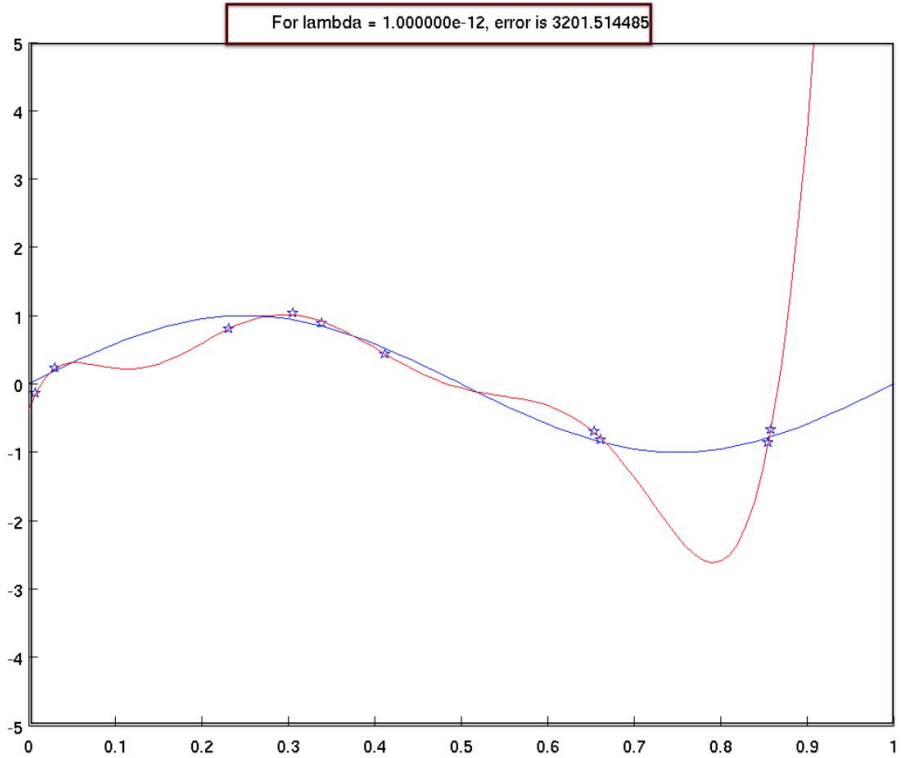

**Degree 9 polynomial with regularization parameter** $\lambda = 10^{-14}$

# Regularized training



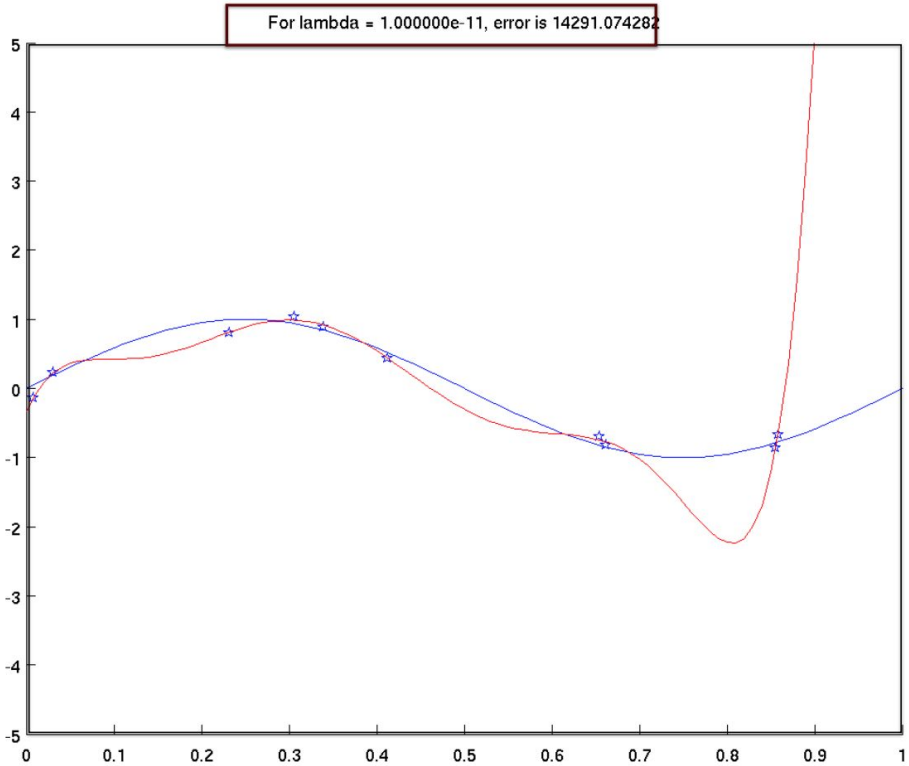

**Degree 9 polynomial with regularization parameter** $\lambda = 10^{-13}$

# Regularized training



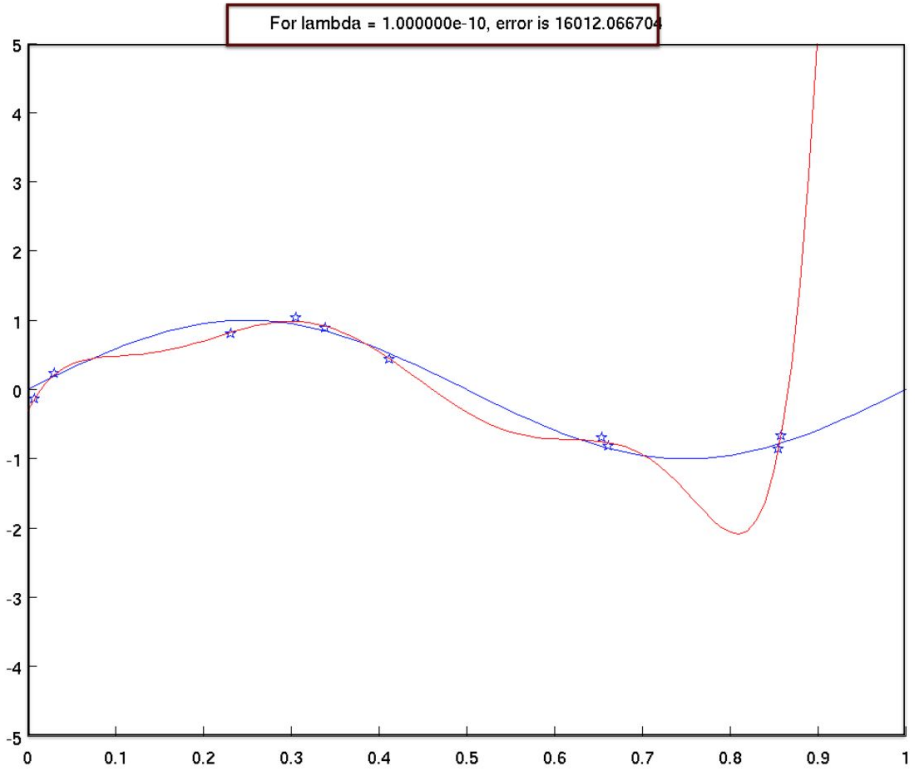

**Degree 9 polynomial with regularization parameter $\lambda = 10^{-12}$**

# Regularized training



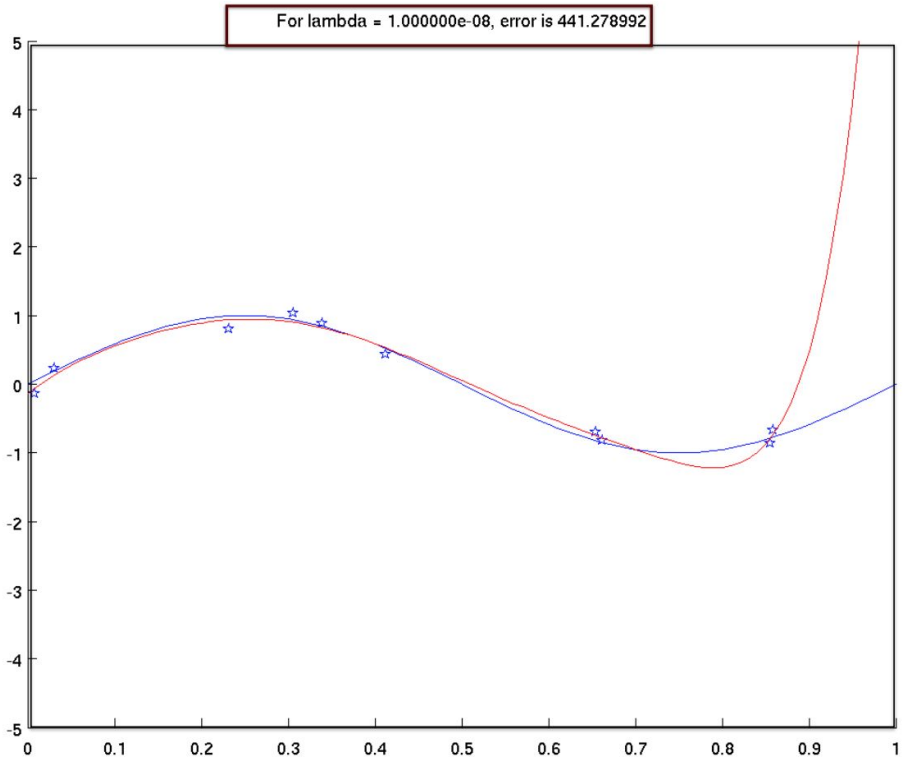

**Degree 9 polynomial with regularization parameter** $\lambda = 10^{-11}$

# Regularized training



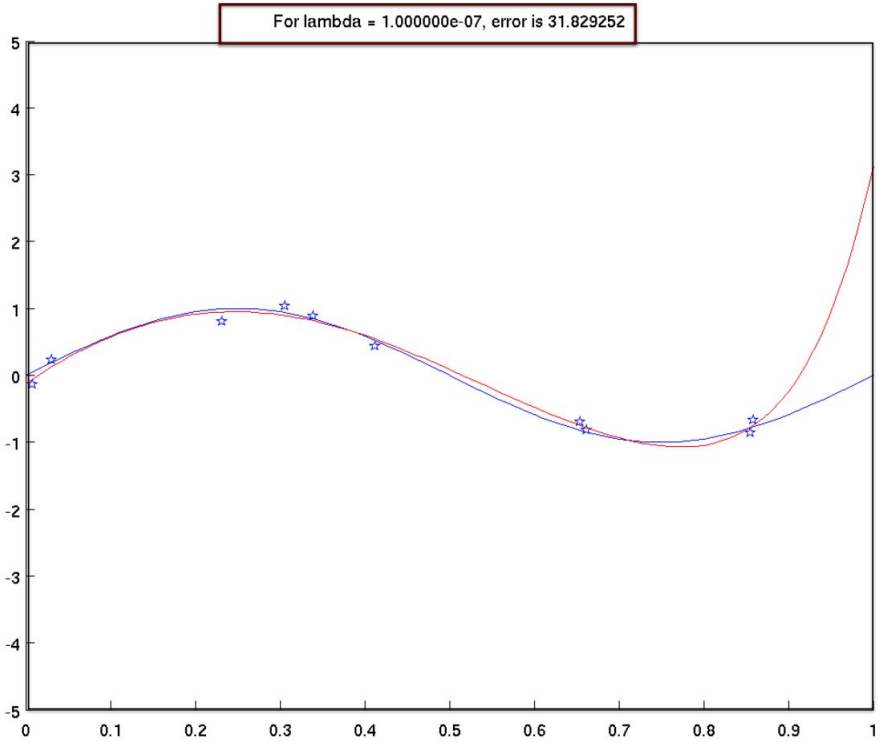

**Degree 9 polynomial with regularization parameter** $\lambda = 10^{-10}$

# Regularized training



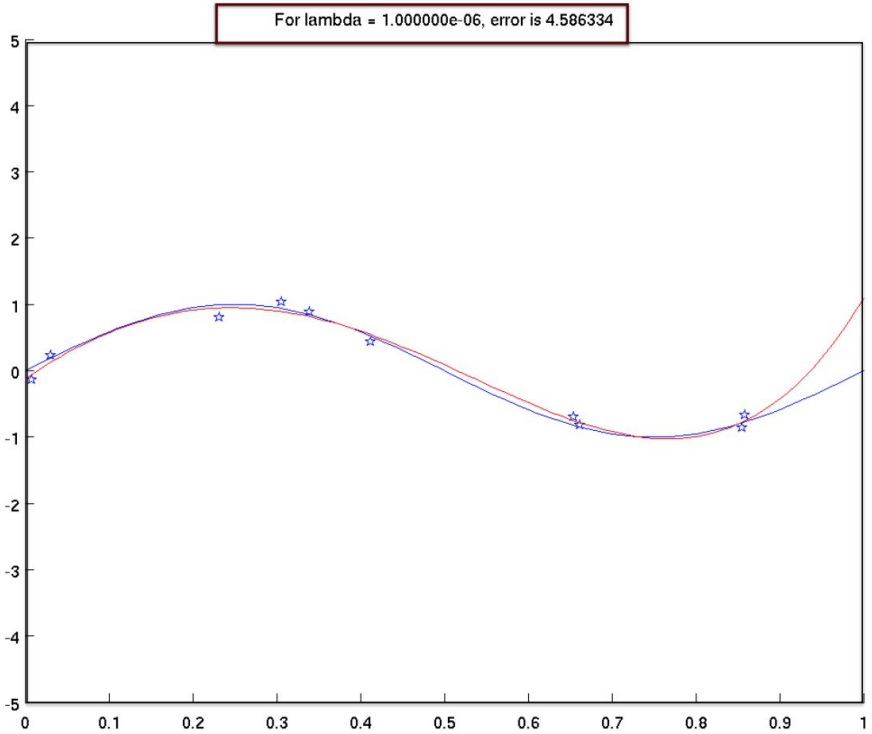

**Degree 9 polynomial with regularization parameter** $\lambda = 10^{-8}$

# Regularized training



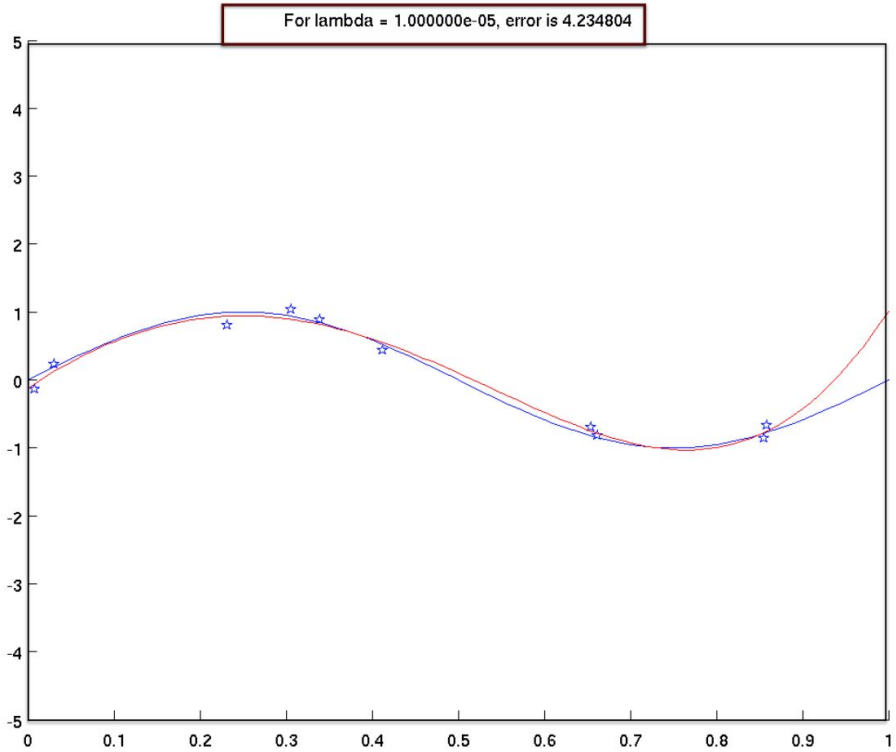

**Degree 9 polynomial with regularization parameter** $\lambda = 10^{-7}$

# Regularized training



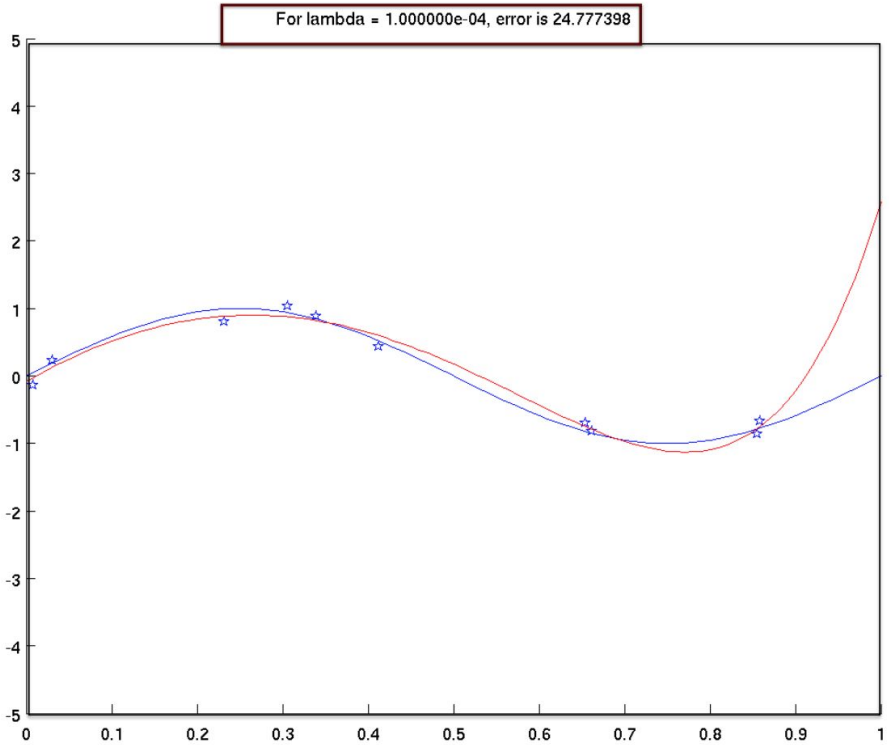

**Degree 9 polynomial with regularization parameter $\lambda = 10^{-6}$**

# Regularized training



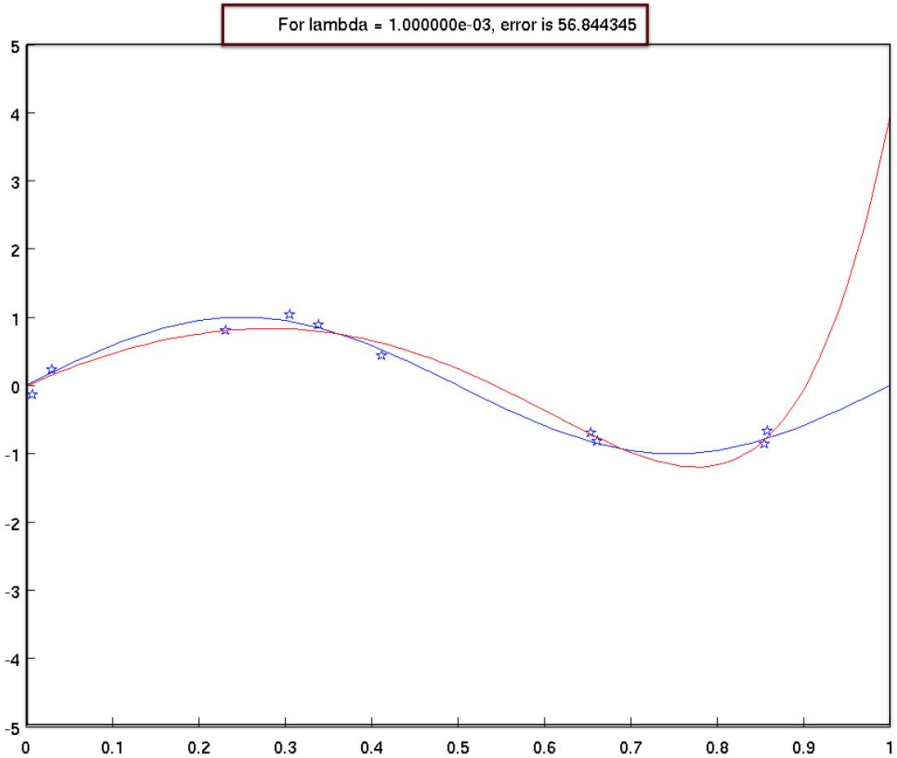

**Degree 9 polynomial with regularization parameter** $\lambda = 10^{-5}$

# Regularized training



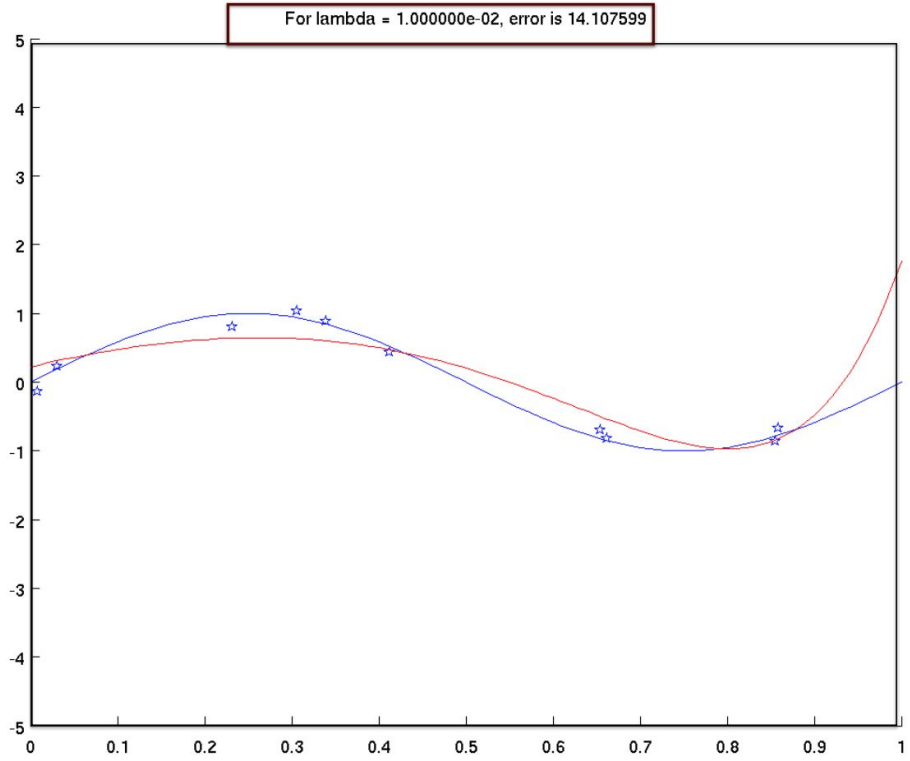

**Degree 9 polynomial with regularization parameter** $\lambda = 10^{-4}$

# Regularized training



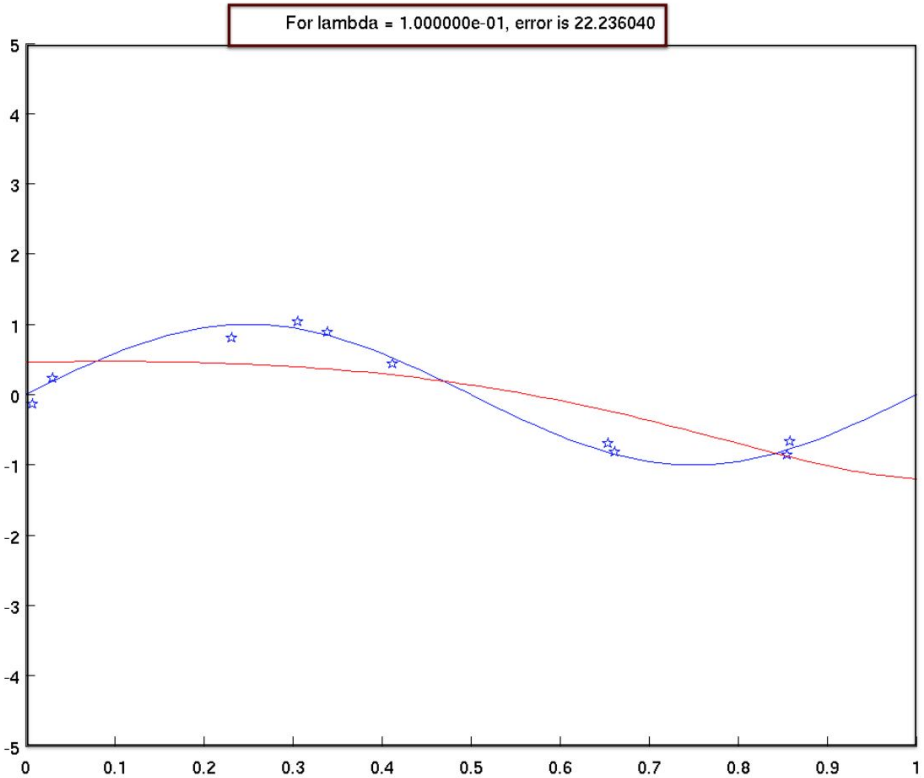

**Degree 9 polynomial with regularization parameter** $\lambda = 10^{-3}$

# Regularized training



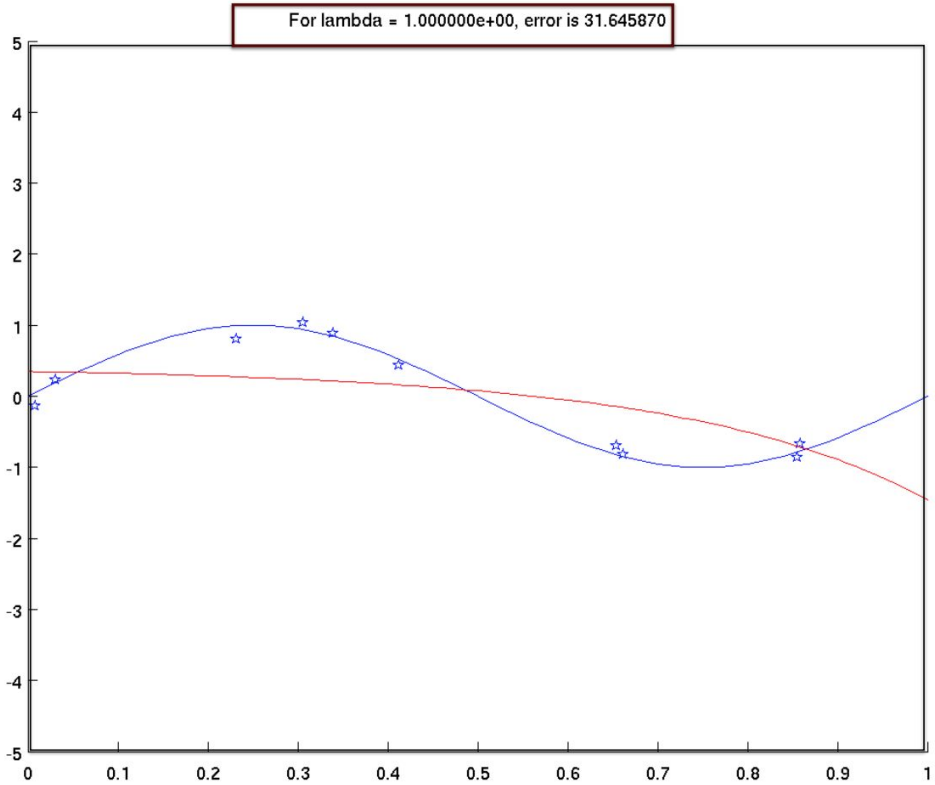

**Degree 9 polynomial with regularization parameter** $\lambda = 10^{-2}$

# Regularized training



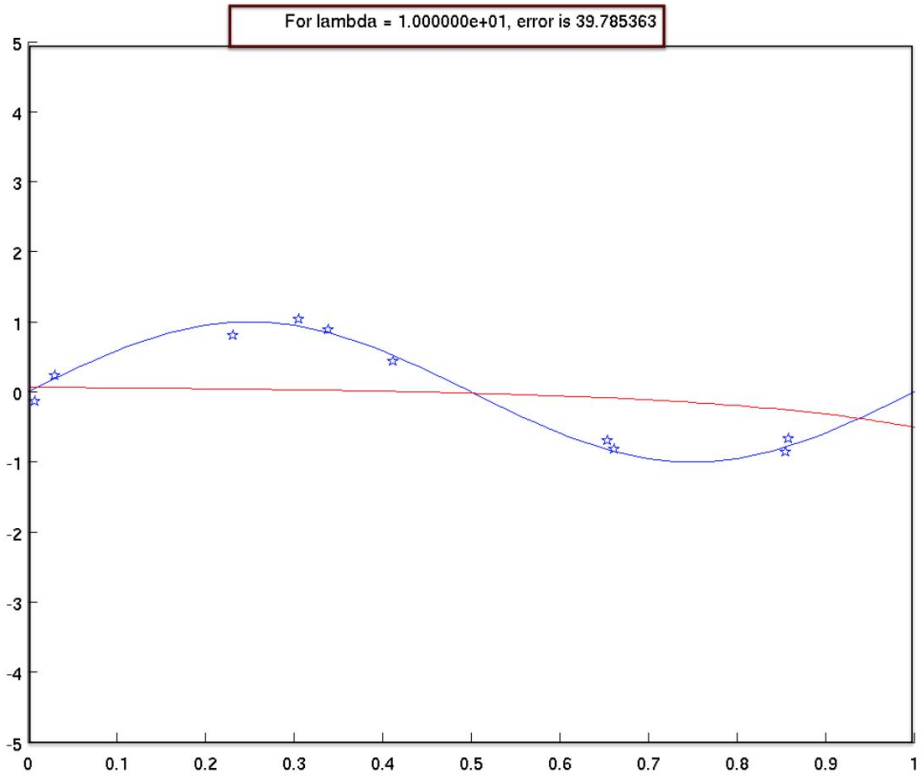

**Degree 9 polynomial with regularization parameter** $\lambda = 10^{-1}$

# Regularized training




**Degree 9 polynomial with regularization parameter $\lambda = 10^0$**

# Regularized training

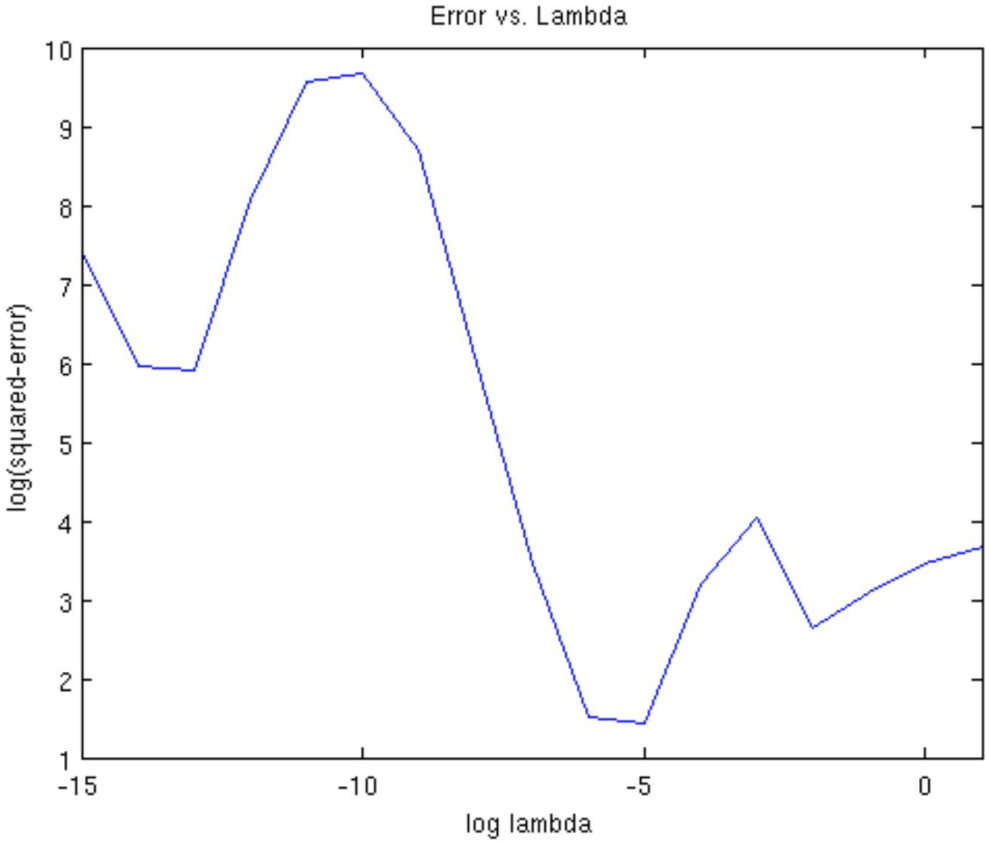



**Degree 9 polynomial with regularization parameter $\lambda = 10^1$**

# Regularization effect on coefficients

| Regularization | $\lambda = 10^{-15}$ | $\lambda = 10^{-12}$ | $\lambda = 10^{-9}$ | $\lambda = 10^{-6}$ | $\lambda = 10^{-3}$ | $\lambda = 10^{0}$ |
|---|---|---|---|---|---|---|
| $w_9$ | 1.14e4 | -8.37e3 | 1.72e3 | 3.44e0 | 3.14e0 | -5.93e2 |
| $w_8$ | -6.65e4 | 2.34e4 | -1.84e3 | -5.19e0 | 3.19e0 | -7.90e-2 |
| $w_7$ | 1.55e5 | -1.46e4 | -1.50e3 | -3.75e0 | 3.01e0 | -1.07e-1 |
| $w_6$ | -1.87e5 | -1.66e4 | 1.33e3 | 4.95e0 | 2.38e0 | -1.46e-1 |
| $w_5$ | 1.28e5 | 2.92e4 | 1.97e3 | 1.02e1 | 1.00e0 | -2.00e-1 |
| $w_4$ | -5.06e4 | -1.71e4 | -2.45e3 | 2.05e0 | -1.52e0 | -2.71e-1 |
| $w_3$ | 1.10e4 | 4.76e3 | 9.75e2 | -1.16e0 | -5.23e0 | -3.51e-1 |
| $w_2$ | -1.18e3 | -6.22e2 | -1.77e2 | -1.82e1 | -7.60e0 | -3.91e-1 |
| $w_1$ | 5.06e1 | 3.51e1 | 1.77e1 | 8.86e0 | 5.59e0 | -2.05e-1 |
| $w_0$ | -4.84e-1 | -3.87e-1 | -2.29e-1 | -1.29e-1 | -1.30e-2 | 3.49e-1 |

# Regularization parameters and generalization



# Takeaways

1. We can improve performance by restricting number of parameters (simpler models).

2. We can improve performance by getting more data.

3. We can improve performance by **regularization**:

- Aggressive regularization results in simpler models, thus increasing bias and decreasing variance
- Passive regularization results in more complex models thus decreasing bias while increasing variance.

# Hyperparameters

Note that regularization parameter $\lambda$ is not a part of model parameters, i.e,

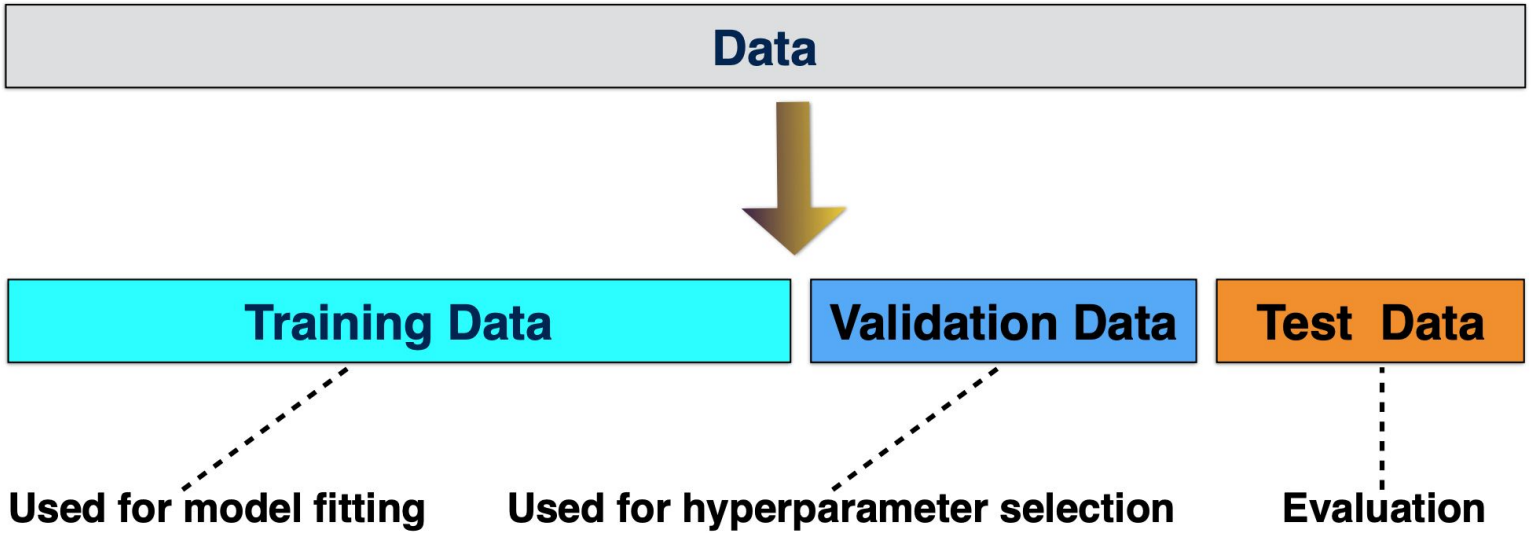
$$w_0, \ldots, w_9$$

To distinguish it from **model parameters** we call it a **hyper-parameter**

How to find the best value for the regularization parameter that results in minimum test error (better generalization)?

The answer to this question is part of our model selection task!

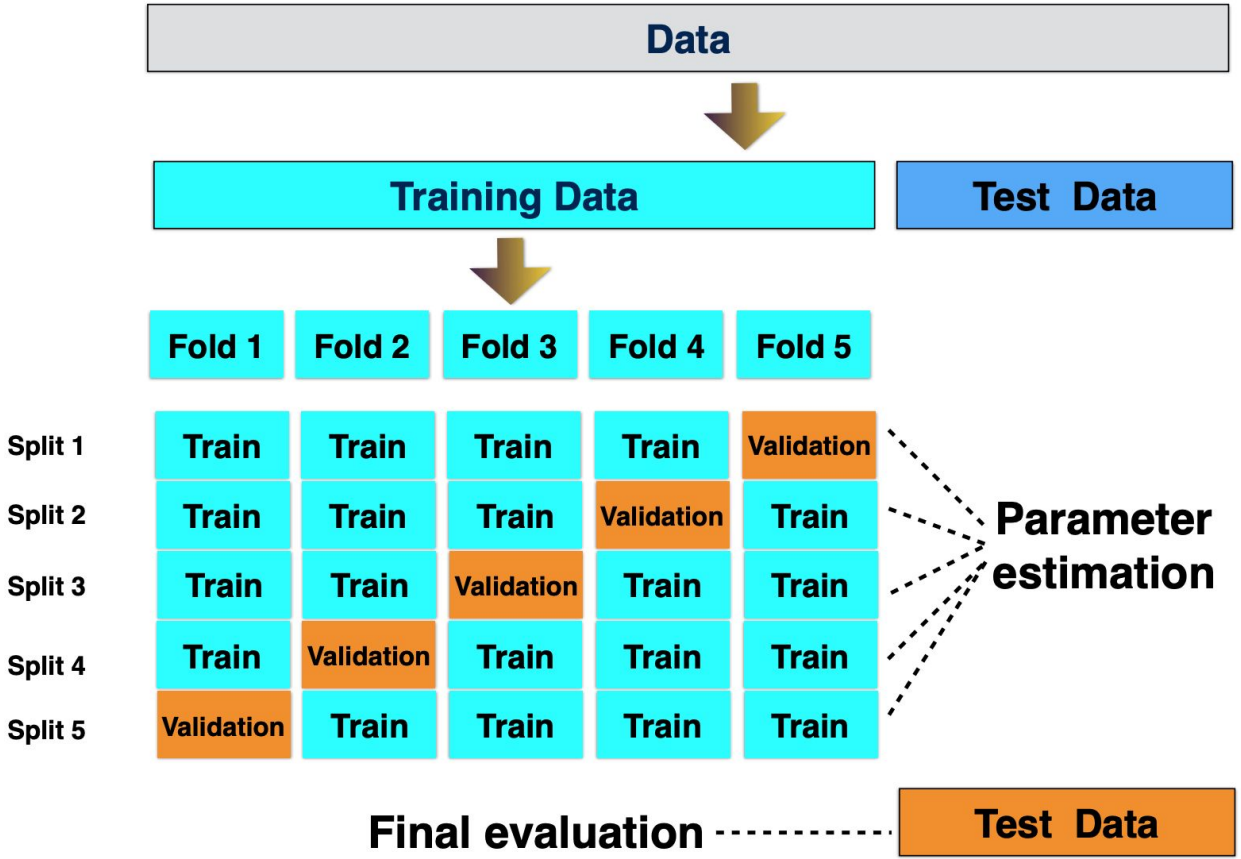
# Threefold split




**Pro: fast, simple**

**Con: high variance, bad use of data**

# K-fold cross validation



# All together