Fundamentals of Machine Learning

Mohammad Emtiyaz Khan
AIP (RIKEN), Tokyo

http://emtiyaz.github.io

emtiyaz.khan@riken.jp

Jan 30, 2019

Goals

Understand (some) fundamentals of Machine learning[1].

Part I : Understand the basic set-up to analyze data under a machine-learning framework.

1. Before Machine Learning.
2. ML Problem: Regression.
3. Model: Linear Regression.
4. Cost Function: MSE.
5. Algorithm 1: Gradient Descent.

Part II : Understand what can go wrong when learning from data and how to correct it.

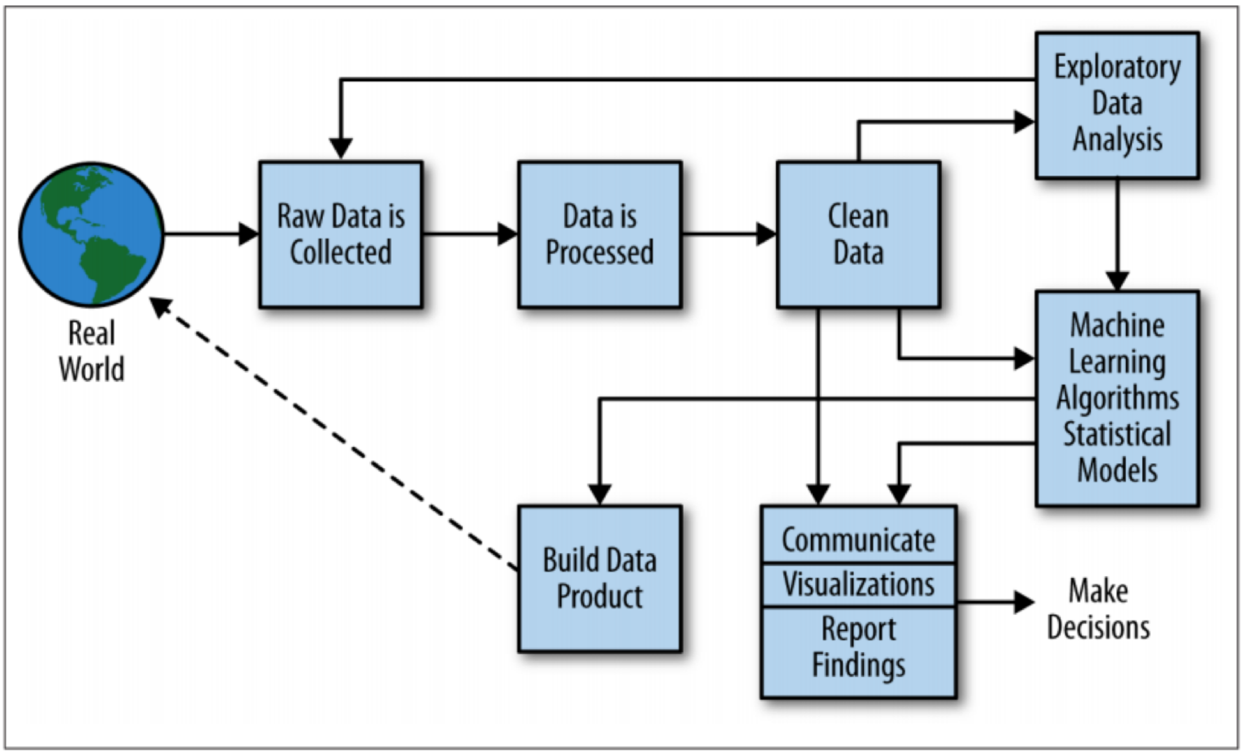
6. Challenge: Overfitting.
7. Solutions: Regularization.
8. Bias-Variance Decomposition.
9. Recent Advances.

[1]Some figures are taken from Hastie, Tibshirani, and Friedman's book on statistical learning and also from Chris Bishop's Machine learning book

1 Before Machine Learning

Acquiring Data

Data is the most important component of modern Machine Learning. There are many important steps that can have a huge impact on the performance of a machine-learning system. To name a few: data collection, cleaning, validation, pre-processing, and storage.


Real World
Raw Data is Collected
Data is Processed
Clean Data
Exploratory Data Analysis
Machine Learning Algorithms Statistical Models
Build Data Product
Communicate Visualizations Report Findings
Make Decisions

Picture taken from "Doing data science".

Defining an ML problem

Once we have some data, the next step is to re-define the real-world problem in the context of data, and then to convert it to a machine-learning problem.

ML problems can be categorized into 3 main types: supervised, unsupervised, and reinforcement learning. In practice, a successful end-to-end system might require a combination of these problems.

2 ML Problem: Regression

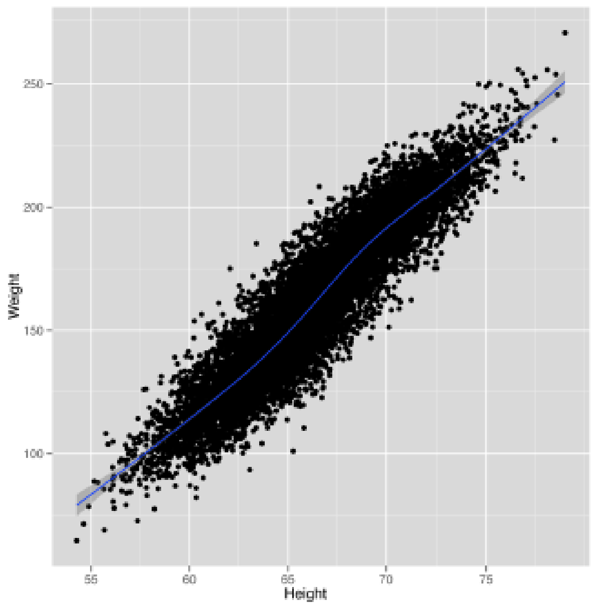
What is regression?

Regression is to relate input variables to the output variable, to either predict outputs for new inputs and/or to understand the effect of the input on the output.

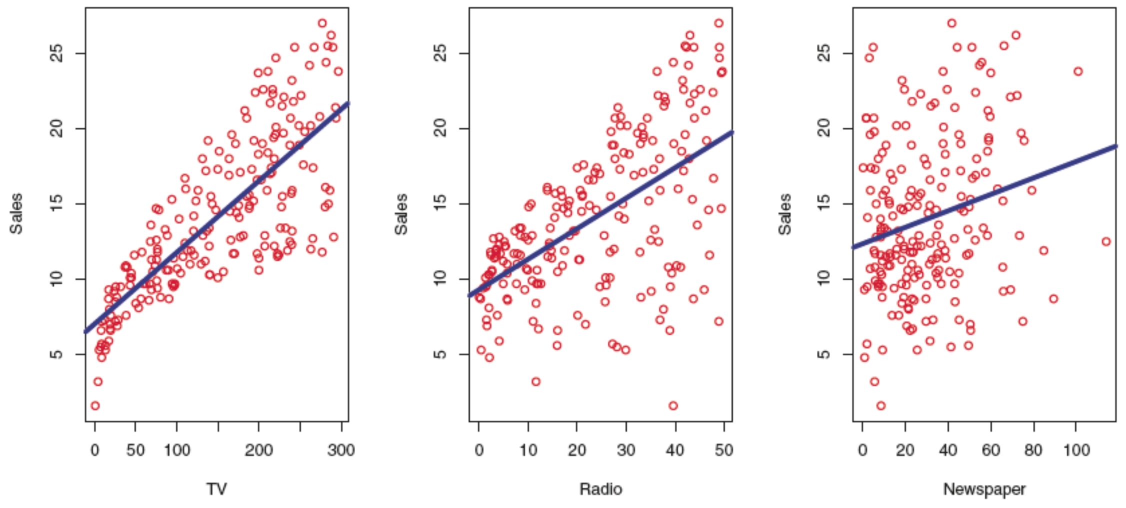
Dataset for regression

In regression, data consists of pairs $(y_n, \mathbf{x}_n)$, where y_n is the n'th output and $\mathbf{x}_n$ is a vector of D inputs. Number of pairs N is the data-size and D is the dimensionality.

Examples of regression



(a) Height is correlated with weight. Taken from "Machine Learning for Hackers"



(b) How does advertisement in TV, radio, and newspaper affect sales? Taken from the book "An Introduction to statistical learning"

Two goals of regression

In prediction, we wish to predict the output for a new input vector, e.g. what is the weight of a person who is 170 cm tall?

In interpretation, we wish to understand the effect of inputs on output, e.g. are taller people heavier too?

The regression function

For both the goals, we need to find a function that approximates the output "well enough" given inputs.

$$y_n \approx f(\mathbf{x}_n), \text{ for all } n$$

Additional Notes

Prediction vs Interpretation

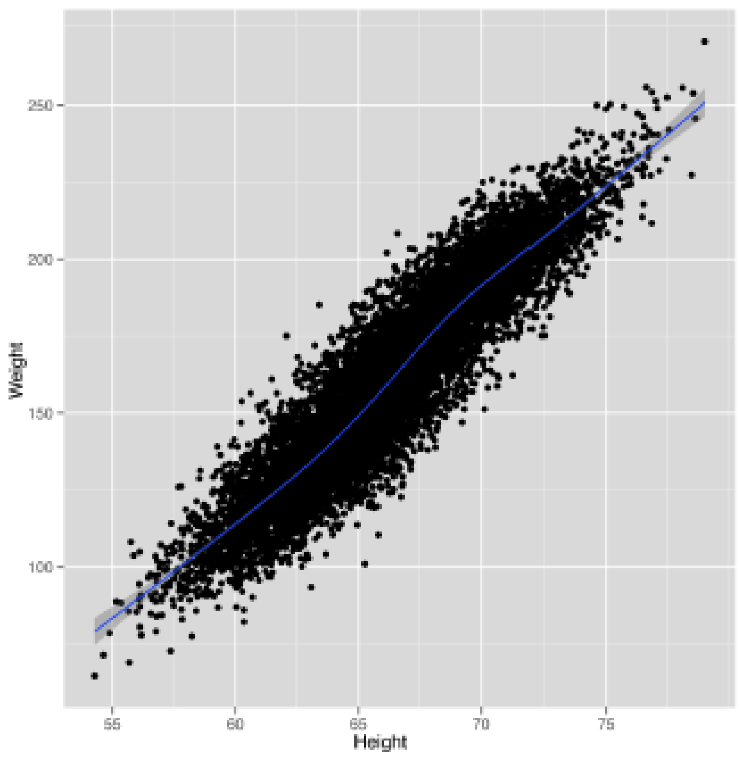
Some questions to think about: are these prediction tasks or interpretation task?

1. What is the life-expectancy of a person who has been smoking for 10 years?
2. Does smoking cause cancer?
3. When the number of packs a smoker smokes per day doubles, their predicted life span gets cut in half?
4. A massive scale earthquake will occur in California within next 30 years.
5. More than 300 bird species in north America could reduce their habitat by half or more by 2080.

3 Model: Linear Regression

What is it?

Linear regression is a model that assumes a linear relationship between inputs and the output.



Why learn about *linear* regression?

Plenty of reasons: simple, easy to understand, most widely used, easily generalized to non-linear models. Most importantly, you can learn almost all fundamental concepts of ML with regression alone.

Simple linear regression

With only one input dimension, it is simple linear regression.

$$y_n \approx f(\mathbf{x}_n) := \beta_0 + \beta_1 x_{n1}$$

Here, β_0 and β_1 are parameters of the model.

Multiple linear regression

With multiple input dimension, it is multiple linear regression.

$$\begin{aligned} y_n &\approx f(\mathbf{x}_n) \\ &:= \beta_0 + \beta_1 x_{n1} + \ldots + \beta_D x_{nD} \\ &= \widetilde{\mathbf{x}}_n^T \boldsymbol{\beta} \qquad (1) \end{aligned}$$

Learning/estimation/fitting

Given data, we would like to find $\boldsymbol{\beta} = [\beta_0, \beta_1, \ldots, \beta_D]$. This is called learning or estimating the parameters or fitting the model.

Additional Notes

$p > n$ Problem

Consider the following simple situation: You have $N = 1$ and you want to fit $y_1 \approx \beta_0 + \beta_1 x_{11}$, i.e. you want to find β_0 and β_1 given one pair (y_1, x_{11}). Is it possible to find such a line?

This problem is related to something called $p > n$ problem. In our notation, this will be called $D > N$ problem, i.e. the number of parameters exceeds number of data examples.

Similar issues will arise when we use gradient descent or least-squares to fit a linear model. These problems are all solved by using regularization, which we will learn later.

4 Cost Function: MSE

Motivation

Consider the following models.

1-parameter model: $y_n \approx \beta_0$
2-parameter model: $y_n \approx \beta_0 + \beta_1 x_{n1}$

How can we estimate (or guess) values of $\boldsymbol{\beta}$ given the data $\mathcal{D}$?

What is a cost function?

Cost functions (or utilities or energy) are used to learn parameters that explain the data well. They define how costly our mistakes are.

Two desirable properties of cost functions

When y is real-valued, it is desirable that the cost is symmetric around 0, since both +ve and -ve errors should be penalized equally.

Also, our cost function should penalize "large" mistakes and "very-large" mistakes almost equally.

Mean Square Error (MSE)

MSE is one of the most popular cost function.

$$MSE(\boldsymbol{\beta}) := \frac{1}{2N}\sum_{n=1}^{N}[y_n - f(\mathbf{x}_n)]^2$$

Does it have both the properties?

An exercise for MSE

Compute MSE for 1-param model:

$$\mathcal{L}(\beta_0) := \frac{1}{2N}\sum_{n=1}^{N}[y_n - \beta_0]^2 \qquad (2)$$

Each row contains a y_n and column is β_0. First, compute MSE for for $y_n = \{1, 2, 3, 4\}$ and draw MSE as a function of β_0 (by adding the first four rows). Then add $y_n = 20$ to it, and redraw MSE. What do you observe and why?

$y_n \backslash \beta_0$	1	2	3	4	5	6	7
1							
2							
3							
4							
MSE							
20							
MSE							

Additional Notes

A question for cost functions

Is there an automatic way to define loss functions?

Nasty cost functions: Visualization

See Andrej Karpathy Tumblr post for many cost functions gone “wrong” for neural network. http://lossfunctions.tumblr.com/.

5 Algorithm 1: Gradient Descent

Learning/estimation/fitting

Given a cost function $\mathcal{L}(\boldsymbol{\beta})$, we wish to find $\boldsymbol{\beta}^*$ that minimizes the cost:

$$\min_{\boldsymbol{\beta}} \mathcal{L}(\boldsymbol{\beta}), \quad \text{subject to } \boldsymbol{\beta} \in \mathbb{R}^{D+1}$$

This is learning posed as an optimization problem. We will use an algorithm to solve the problem.

Grid search

Grid search is one of the simplest algorithms where we compute cost over a grid (of say M points) to find the minimum. This is extremely simple and works for any kind of loss when we have very few parameters and the loss is easy to compute.

For a large number of parameters, however, grid search has too many "for-loops", resulting in exponential computational complexity. Choosing a good range of values is another problem.

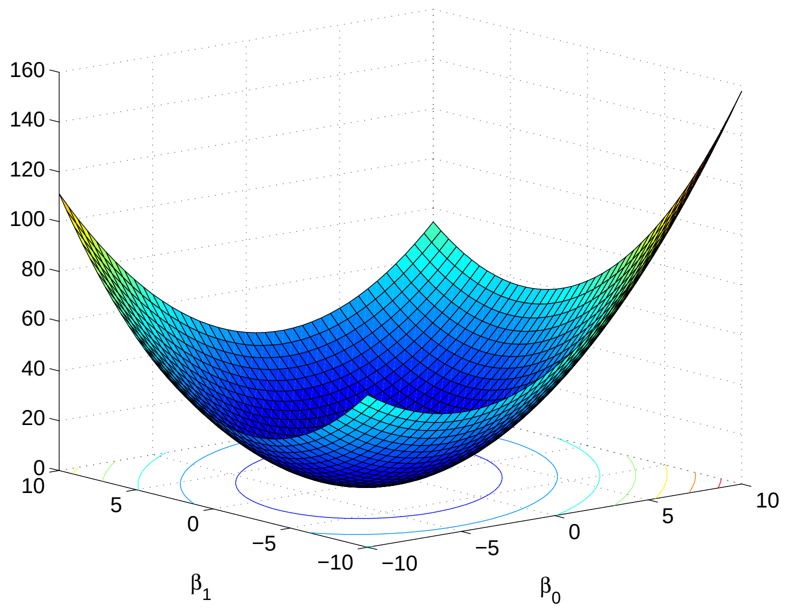
Are there any other issues?

Follow the gradient

A gradient (at a point) is the slope of the tangent (at that point). It points to the direction of largest increase of the function.

For 2-parameter model, MSE is shown below.

(I used $\mathbf{y}^T = [2, -1, 1.5]$ and $\mathbf{x}^T = [-1, 1, -1]$).



Batch gradient descent

To minimize the function, take a step in the (opposite) direction of the gradient

$$\boldsymbol{\beta}^{(k+1)} \leftarrow \boldsymbol{\beta}^{(k)} - \alpha \frac{\partial \mathcal{L}(\boldsymbol{\beta}^{(k)})}{\partial \boldsymbol{\beta}}$$

where $\alpha > 0$ is the step-size (or learning rate).

Gradient descent for 1-parameter model to minimize MSE:

$$\beta_0^{(k+1)} = (1-\alpha)\beta_0^{(k)} + \alpha \bar{y}$$

Where $\bar{y} = \sum_n y_n / N$. When is this sequence guaranteed to converge?

Gradients for MSE

$$\mathcal{L}(\boldsymbol{\beta}) = \frac{1}{2N} \sum_{n=1}^{N} (y_n - \widetilde{\mathbf{x}}_n^T \boldsymbol{\beta})^2 \qquad (3)$$

then the gradient is given by,

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = -\frac{1}{N} \sum_{n=1}^{N} (y_n - \widetilde{\mathbf{x}}_n^T \boldsymbol{\beta}) \widetilde{\mathbf{x}}_n \qquad (4)$$

What is the computational complexity of batch gradient descent?

Stochastic gradient descent

When N is large, choose a random pair $(\mathbf{x}_i, y_i)$ in the training set and approximate the gradient:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} \approx -\frac{1}{N}\left[N(y_i - \widetilde{\mathbf{x}}_i^T \boldsymbol{\beta})\widetilde{\mathbf{x}}_i\right] \tag{5}$$

Using the above "stochastic" gradient, take a step:

$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} + \alpha^{(k)}(y_i - \widetilde{\mathbf{x}}_i^T \boldsymbol{\beta}^{(k)})\widetilde{\mathbf{x}}_i$$

What is the computational complexity?

For convergence, $\alpha^k \to 0$ "appropriately". One such condition called Robbins-Monroe condition suggests to take α^k such that:

$$\sum_{k=1}^{\infty} \alpha^{(k)} = \infty, \qquad \sum_{k=1}^{\infty} (\alpha^{(k)})^2 < \infty \tag{6}$$

One way to obtain such sequence is $\alpha^{(k)} = 1/(1+k)^r$ where $r \in (0.5, 1)$.

6 Challenge: Overfitting

Motivation

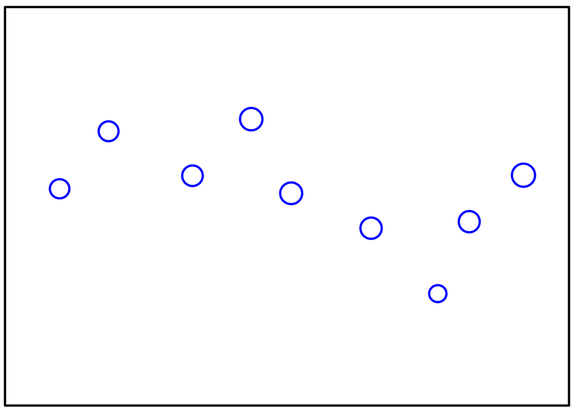
Linear model can be easily modified to obtain more powerful non-linear model. We can use basis function expansion to get a non-linear regression model, and then use a sequence of these models to construct a deep model.

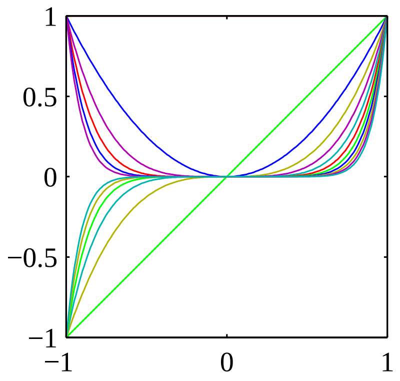
Consider simple linear regression. Given one-dimensional input x_n, we can generate a polynomial basis.

$$\boldsymbol{\phi}(x_n) = [1,\, x_n,\, x_n^2,\, x_n^3,\, \dots,\, x_n^M]$$

Then we fit a linear model using the original *and* the generated features:

$$y_n \approx \beta_0 + \beta_1 x_n + \beta_2 x_n^2 + \dots + \beta_M x_n^M$$



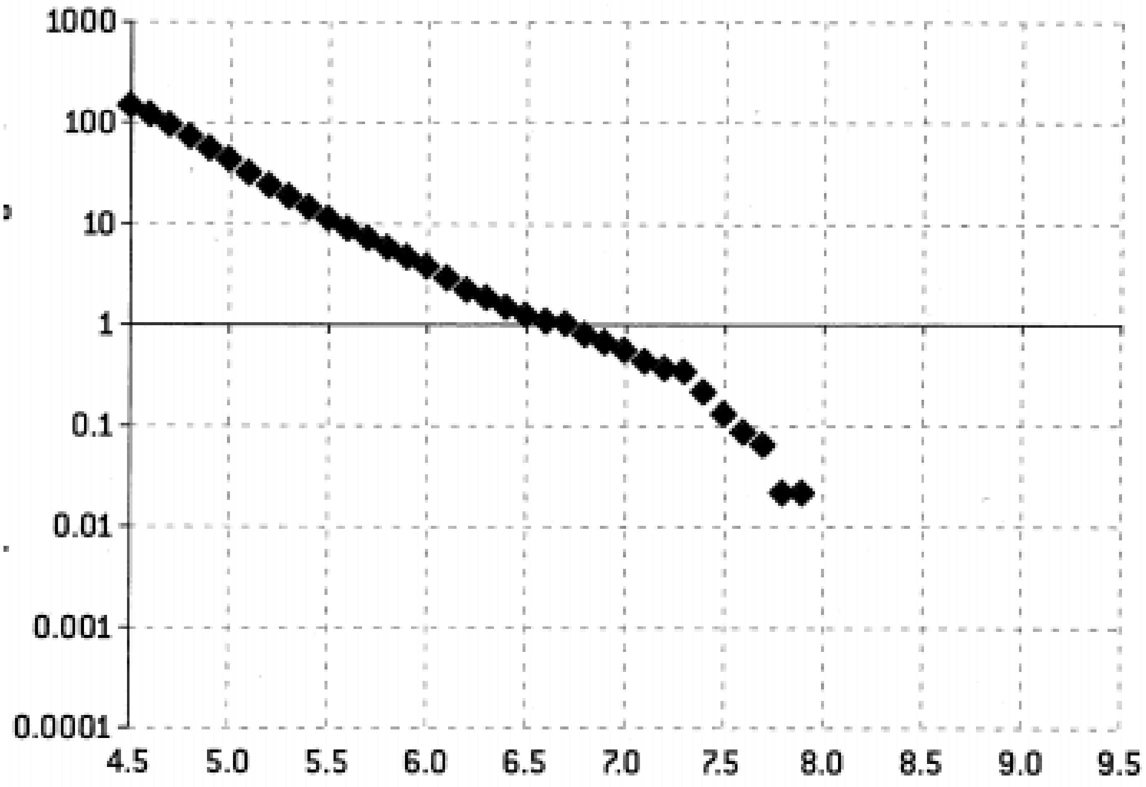


Overfitting and Underfitting

Overfitting is fitting the noise in addition to the signal. Underfitting is not fitting the signal well. In reality, it is very difficult to be able to tell the signal from the noise.

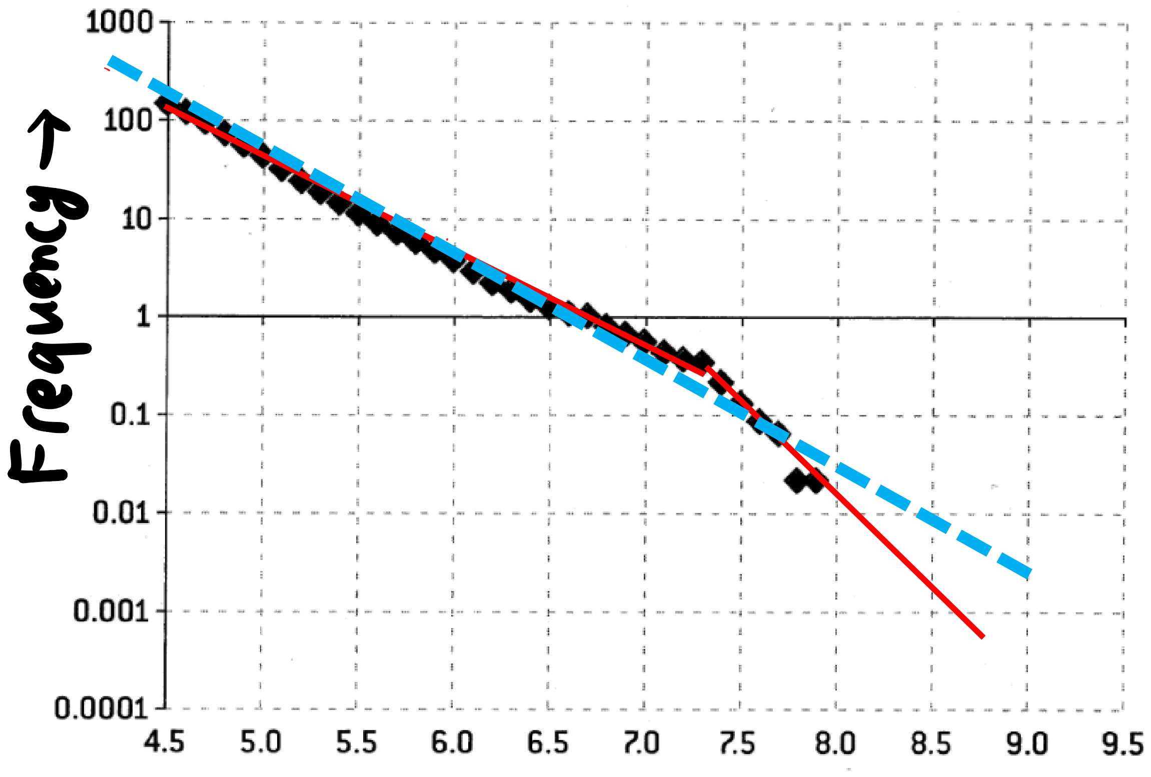
Which is a better fit?

Try a real situation. Below, y-axis is the frequency of an event and x-axis is the magnitude. It is clear that as magnitude increases, frequency decreases.

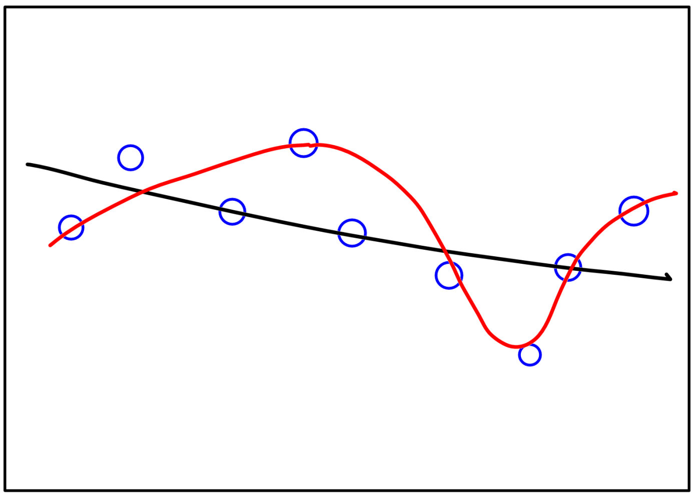


This example is taken from Nate Silver's book.

Which model is a better fit? blue or red?

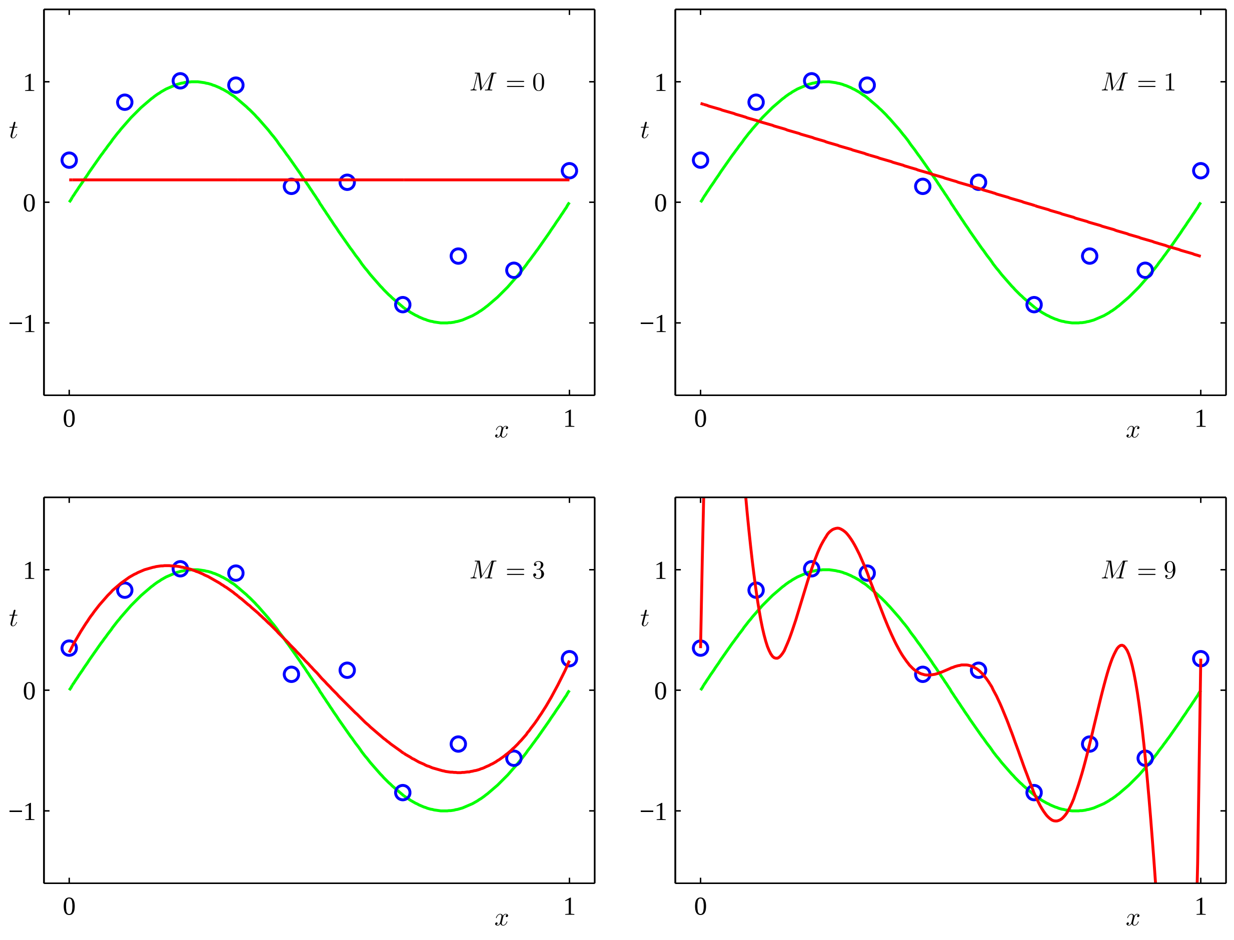


Another example: Which model is a better fit? black or red? Data is denoted by circle.

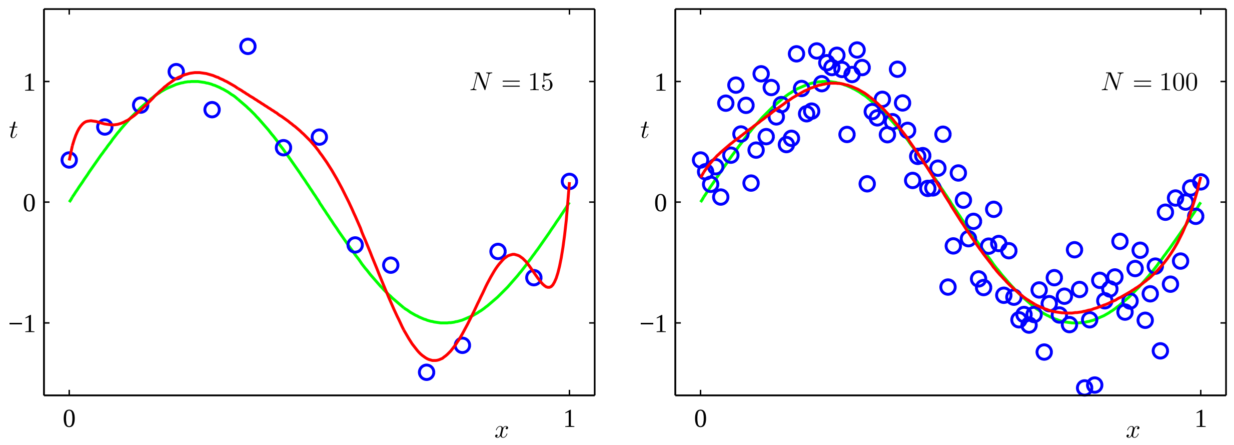


Complex models overfit easily

Circles are data points, green line is the truth & red line is the model fit. M is the maximum degree in the generated polynomial basis.



If you increase the amount of data, overfitting *might* reduce.



Occam’s razor

One solution is dictated by Occam’s razor which states that “Simpler models are better – in absence of certainty.”

Sometimes, if you increase the amount of data, you might reduce overfitting. But, when unsure, choose a simple model over a complicated one.

Additional Notes

Read about overfitting in the paper by Pedro Domingos (section 3 and 5 of “A few useful things to know about machine learning”). You can also read Nate Silver’s book on “The signal and the noise” (the earthquake example is taken from this book).

7 Solutions: Regularization

What is regularization?

Through regularization, we can penalize complex models and favor simpler ones:

$$\min_{\beta} \quad \mathcal{L}(\boldsymbol{\beta}) + \frac{\lambda}{2N} \sum_{j=1}^{M} \beta_j^2$$

The second term is a regularizer (with $\lambda > 0$). The main point here is that an input variable weighted by a small β_j will have less influence on the output.

Regularization Parameter

The parameter λ can be tuned to reduce overfitting. But, how do you choose λ?

The generalization error

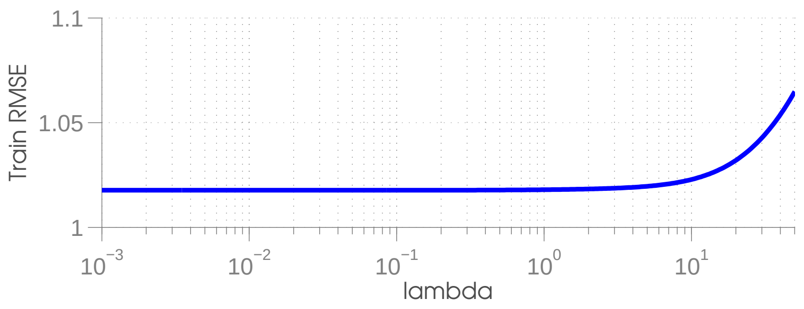
The generalization error of a learning method is the expected prediction error for *unseen* data, i.e. mistakes made on the data that we are going to see in the future. This quantifies how well the method *generalizes*.

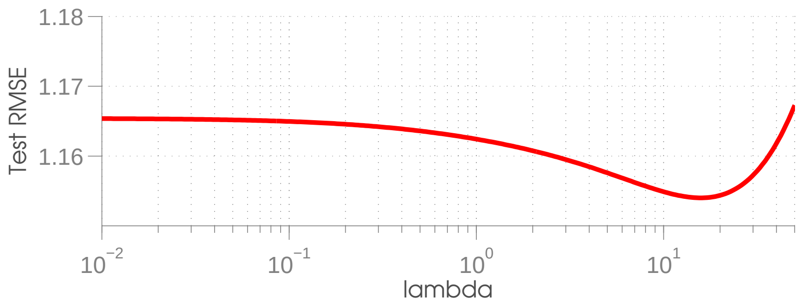
Simulating the future

Ideally, we should choose λ to minimize the mistakes that will be made in the future. Obviously, we do not have the future data, but we can always *simulate the future* using the data in hand.

Splitting the data

For this purpose, we split the data into train and validation sets, e.g. 80% as training data and 20% as validation data. We pretend that the validation set is the future data. We fit our model on the training set and compute a prediction-error on the validation set. This gives us an *estimate* of the generalization error (one instant of the future).

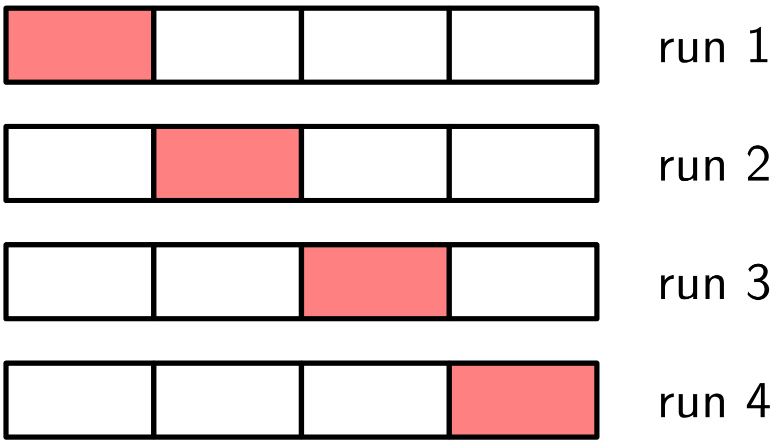




Cross-validation

Random splitting (aka bootstrap) is not an efficient method.

K-fold cross-validation allows us to do this efficiently. We randomly partition the data into K groups. We train on $K-1$ groups and test on the remaining group. We repeat this until we have tested on all K sets. We then average the results.



Cross-validation returns an estimate of the *generalization error*.

Additional Notes

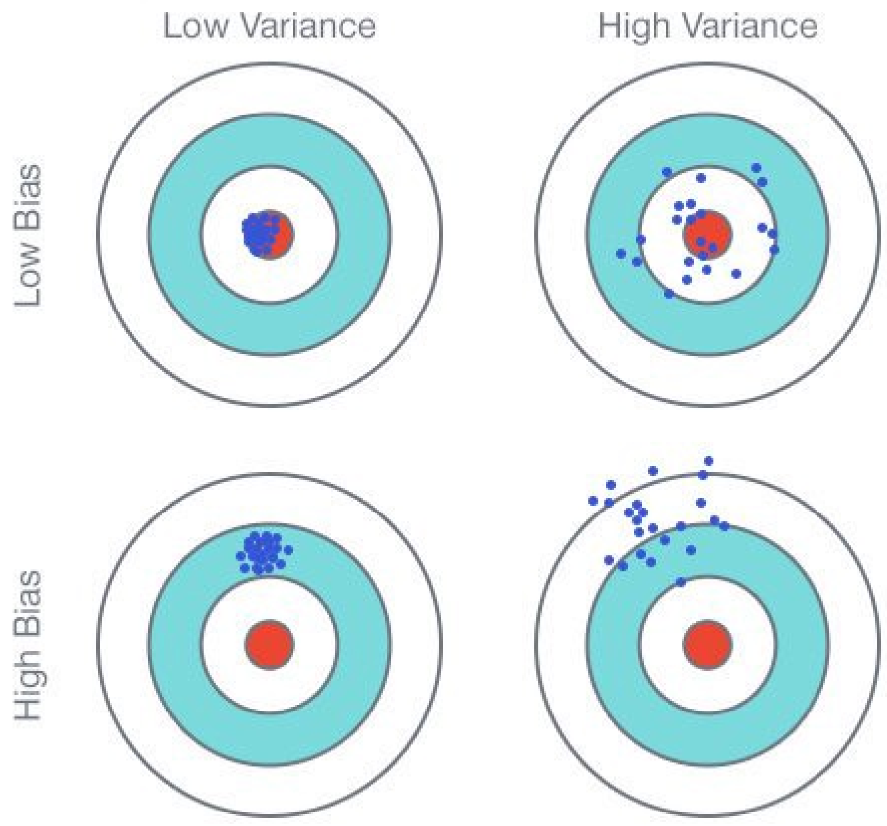
Details on cross-validation are in Chapter 7 in the book by Hastie, Tibshirani, and Friedman (HTF). You can also read about bootstrap in Section 7.11 in HTF book. This method is related to random splitting and is a very popular method.

8 Bias-Variance Decomposition

What is bias-variance?

One natural question is how does the test error vary wrt λ? When λ is high, the model underfits, while when λ is small, the model overfits. Therefore, a good value is somewhere in between.

Bias-variance decomposition explains the shape of this curve.



Generalization error

Given training data $\mathcal{D}_{tr}$ of size N, we would like to estimate the expected error made in future prediction. This error is the generalization error. Below is a definition suppose that we have infinite test data $\mathcal{D}_{te}$,

$$teErr(\mathcal{D}_{tr}) := \mathbb{E}_{\mathcal{D}_{te}}[\{y - f(\mathbf{x})\}^2]$$

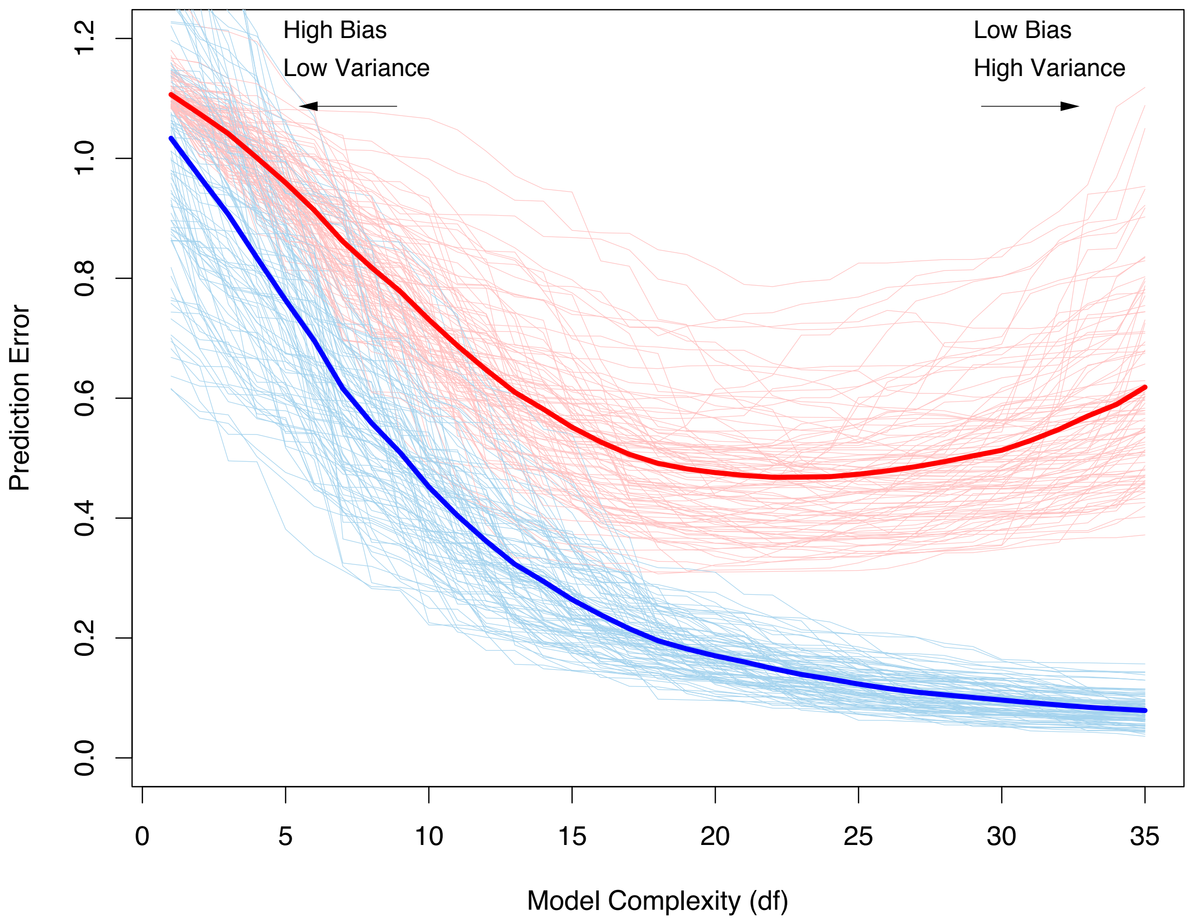
Generalization error is different from the training error which measures how well you fit the data.

$$trErr(\mathcal{D}_{tr}) := \sum_{n=1}^{N}[\{y_n - f(\mathbf{x}_n)\}^2]$$

Errors vs model complexity

As we increase the model complexity, how do these errors vary? The blue line shows training error for a dataset with $N = 50$, while the red line shows the generalization error for that dataset.

Simple model have high train and generalization error since they have a high bias, while complex model have low train but high generalization error because they have high variance.



Bias-variance decomposition

The shape of these curves can be explained using bias-variance decomposition. The following four points can be explained by using the decomposition:

1. both bias and variance contribute to generalization error.
2. For bias, both model-bias and estimation-bias are important. When we increase model complexity, we increase generalization error due to increased variance.
3. Regularization increases estimation bias while reducing variance.

9 Recent Advances

Deep Learning & Overfitting

Deep learning has shown a new (but old) way to combat overfitting. For many applications, more data and deep architecture combined with stochastic gradient-descent is able to get us to a good minimum which generalizes well.

Challenges

There are many challenges ahead. Learning from nasty, unreliable data still remains a challenge (e.g. small sample size, redundant data, non-stationary data, sequential learning).

On the other hand, living beings - even young ones - are very good in dealing with such data. How do they do it, and how can we design ML methods that can learn like them?