CSCI-567: Machine Learning (Fall 2019)

Prof. Victor Adamchik

U of Southern California

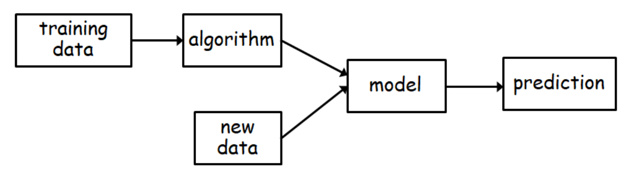
Sep. 10, 2019

Outline

Outline

Predictive modeling

Predictive modeling (i.e. supervised learning) is a process of creating a model using data to make a prediction on new data.



Predictive modeling is a problem of finding a mapping function f from training data ($\boldsymbol{x} \in \mathbb{R}^{\mathrm{D}}$) to output variables.

There are important differences between classification and regression problems.

- continuous vs discrete
- measure *prediction errors* differently.
- lead to quite different learning algorithms.

Classification

Classification is a problem of finding a mapping function f from training data ($\boldsymbol{x} \in \mathbb{R}^{\mathrm{D}}$) to *discrete* output variables ($y \in \mathrm{C}$).

- The output variables are called labels or classes or categories.
- The mapping function predicts the class for a given observation.
- The classification *accuracy* is computed as the percentage of correctly classified examples out of all examples.

Regression

Regression is a problem of finding a mapping function f from training data ($\boldsymbol{x} \in \mathbb{R}^{\mathrm{D}}$) to a *continuous* output variable ($y \in \mathbb{R}$).

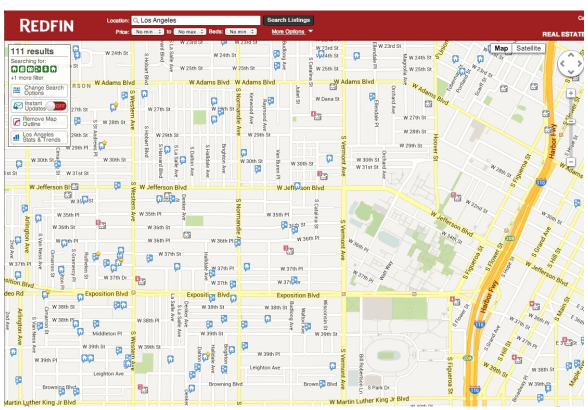
- The output variable is a continuous quantity; pricing optimization, sales forecasting, rating forecasting are some examples.
- Regression predictions can be evaluated using the *mean squared error*.

In some cases, a classification problem can be converted to a regression problem. Some algorithms do this by predicting a probability for each class.

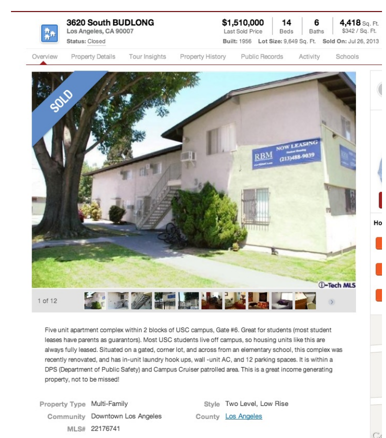
Linear Regression: regression with linear models.

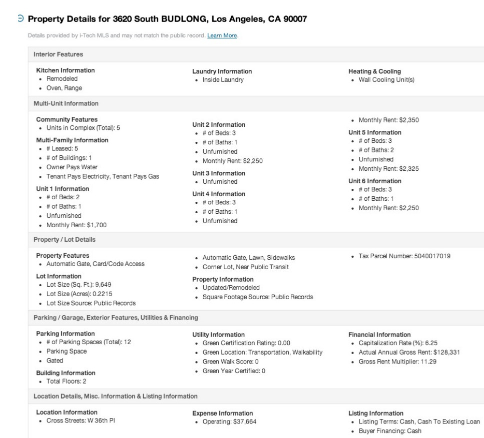
Ex: Predicting the sale price of a house

Retrieve historical sales records (training data)

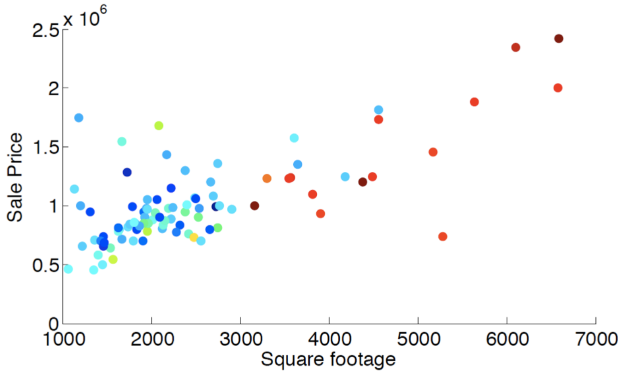


Features used to predict





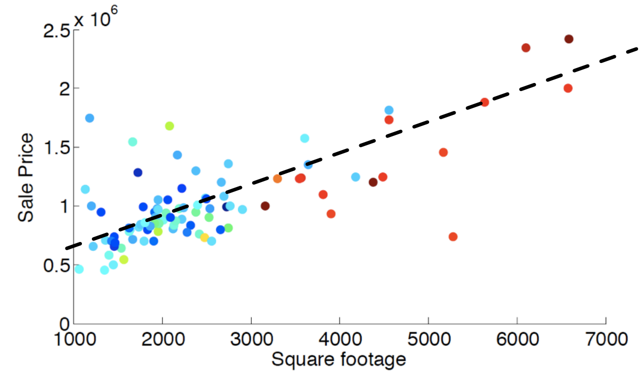
Correlation between square footage and sale price



In linear regression, the goal is to predict y from x using a linear function.

Possibly linear relationship

Sale price $\approx$ **price_per_sqft** $\times$ square_footage $+$ **fixed_expense**
(*slope*) (*intercept*)



How to learn the unknown parameters?

How to measure error for one prediction?

- The classification error (0-1 loss, i.e. *right* or *wrong*) is *inappropriate* for continuous outcomes.
- We can look at
  - *absolute* error: | prediction - sale price |
  - or *squared* error: (prediction - sale price)^2 (**most common**)

Goal: pick the model (unknown parameters) that minimizes the average/total prediction error, but *on what set*?

- test set, ideal but we *cannot use test set while training*
- training set? (minimize the training error)

Example

Predicted price $=$ **price_per_sqft** $\times$ square_footage $+$ **fixed_expense**

one model: price_per_sqft $=$ 0.3K, fixed_expense $=$ 210K

sqft	sale price (K)	prediction (K)	squared error
2000	810	810	0
2100	907	840	67^2
1100	312	540	228^2
5500	2,600	1,860	740^2
···	···	···	···
Total			$0 + 67^2 + 228^2 + 740^2 + \cdots$

Adjust **price_per_sqft** and **fixed_expense** such that the total squared error is minimized.

Formal setup for linear regression

Input: $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$ (features, covariates, context, predictors, etc)

Output: $y \in \mathbb{R}$ (responses, targets, outcomes, etc)

Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, \mathsf{N}\}$

Here x_{nd} represents the dth dimension of the nth sample $\boldsymbol{x}_n$

Linear model: $f : \mathbb{R}^{\mathsf{D}} \to \mathbb{R}$, with $f(\boldsymbol{x}) = w_0 + \sum_{d=1}^{D} w_d x_d$

Linear regression has been around for more than 200 years...

Notation Convenience

NOTE: for notation convenience, we will

- append 1 to each $\boldsymbol{x}_n$ as the first feature: $\boldsymbol{x} = [1,\ x_1,\ x_2,\ \ldots,\ x_{\mathsf{D}}]^{\mathbf{T}}$
- append w_0 to weights: $\boldsymbol{w} = [w_0,\ w_1,\ w_2,\ \ldots,\ w_{\mathsf{D}}]^{\mathbf{T}}$

The model becomes

$$f : \mathbb{R}^{\mathsf{D}+1} \to \mathbb{R}$$

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\mathbf{T}}\boldsymbol{x} = \boldsymbol{x}^{\mathbf{T}}\boldsymbol{w}$$

So please pay attention to notations!

Goal

Minimize total squared error

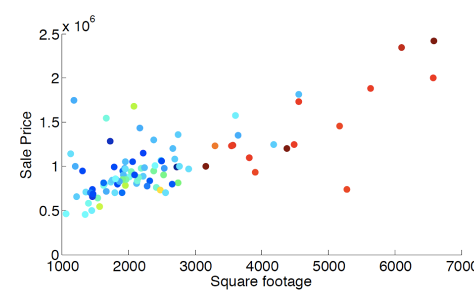
- **Residual Sum of Squares** (RSS), a function of $\boldsymbol{w}$

$$\mathrm{RSS}(\boldsymbol{w}) = \sum_{n=1}^{N} (f(\boldsymbol{x}_n) - y_n)^2 = \sum_{n=1}^{N} (\boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{w} - y_n)^2$$

- find $\boldsymbol{w}^* = \underset{\boldsymbol{w} \in \mathbb{R}^{\mathsf{D}+1}}{\operatorname{argmin}} \mathrm{RSS}(\boldsymbol{w})$
- minimize the Euclidean distance, or find the **least squares solution**
- *reduce machine learning to optimization*

Warm-up: $\mathsf{D} = 0$

Only one parameter w_0: constant prediction $f(x) = w_0$



f is a horizontal line, where should it be?

Use calculus to find the value of w_0 that minimizes the RSS

Warm-up: D $= 0$

Optimization objective becomes

$$\mathrm{RSS}(w_0) = \sum_{n=1}^{N} (w_0 - y_n)^2$$

$$\frac{\partial \mathrm{RSS}(w_0)}{\partial w_0} = 2\sum_{n=1}^{N} (w_0 - y_n) = 0$$

$$N\, w_0 - \sum_{n=1}^{N} y_n = 0$$

It follows that $w_0 = \frac{1}{N}\sum_n y_n$, i.e. the **average**

Exercise: what if we use absolute error instead of squared error?

Warm-up: D $= 1$

Optimization objective becomes

$$\mathrm{RSS}(\boldsymbol{w}) = \sum_n (w_0 + w_1 x_n - y_n)^2$$

General approach: find *stationary points*, i.e., points with *zero gradient*

$$\begin{cases} \frac{\partial \mathrm{RSS}(\boldsymbol{w})}{\partial w_0} = 0 \\ \frac{\partial \mathrm{RSS}(\boldsymbol{w})}{\partial w_1} = 0 \end{cases} \Rightarrow \begin{array}{ll} \sum_n (w_0 + w_1 x_n - y_n) & = 0 \\ \sum_n (w_0 + w_1 x_n - y_n) x_n & = 0 \end{array}$$

$$\Rightarrow \begin{array}{ll} N w_0 + w_1 \sum_n x_n & = \sum_n y_n \\ w_0 \sum_n x_n + w_1 \sum_n x_n^2 & = \sum_n y_n x_n \end{array} \qquad (\textbf{a linear system})$$

$$\Rightarrow \begin{pmatrix} N & \sum_n x_n \\ \sum_n x_n & \sum_n x_n^2 \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \end{pmatrix} = \begin{pmatrix} \sum_n y_n \\ \sum_n x_n y_n \end{pmatrix}$$

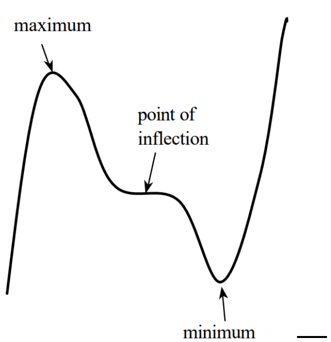
Least square solution for D $= 1$

Assuming the matrix is invertible:

$$\Rightarrow \begin{pmatrix} w_0^* \\ w_1^* \end{pmatrix} = \begin{pmatrix} N & \sum_n x_n \\ \sum_n x_n & \sum_n x_n^2 \end{pmatrix}^{-1} \begin{pmatrix} \sum_n y_n \\ \sum_n x_n y_n \end{pmatrix}$$

Are stationary points minimizers?

- not true in general
- yes for **convex** objectives



General least square solution

Objective

$$\mathrm{RSS}(\boldsymbol{w}) = \sum_n (\boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{w} - y_n)^2$$

Again, find stationary points (**multivariate calculus**)

$$\begin{aligned} \frac{1}{2}\nabla \mathrm{RSS}(\boldsymbol{w}) &= \sum_n \boldsymbol{x}_n (\boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{w} - y_n) = \left(\sum_n \boldsymbol{x}_n \boldsymbol{x}_n^{\mathrm{T}}\right) \boldsymbol{w} - \sum_n \boldsymbol{x}_n y_n \\ &= (\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}) \boldsymbol{w} - \boldsymbol{X}^{\mathrm{T}} \boldsymbol{y} = \boldsymbol{0} \end{aligned}$$

where

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1^{\mathrm{T}} \\ \boldsymbol{x}_2^{\mathrm{T}} \\ \vdots \\ \boldsymbol{x}_{\mathrm{N}}^{\mathrm{T}} \end{pmatrix} \in \mathbb{R}^{\mathrm{N} \times (D+1)}, \quad \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{\mathrm{N}} \end{pmatrix} \in \mathbb{R}^{\mathrm{N}}$$

General least square solution

$$(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})\boldsymbol{w} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{0} \quad \Rightarrow \quad \boldsymbol{w}^* = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$

assuming $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ (**called a covariance matrix**) is invertible for now.

Again by convexity $\boldsymbol{w}^*$ is the minimizer of RSS.

Verify the solution when $\mathsf{D} = 1$**:**

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_{\mathsf{N}} \end{pmatrix} \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \cdots & \cdots \\ 1 & x_{\mathsf{N}} \end{pmatrix} = \begin{pmatrix} N & \sum_n x_n \\ \sum_n x_n & \sum_n x_n^2 \end{pmatrix}$$

when $\mathsf{D} = 0$**:** $(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1} = \frac{1}{N}$, $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} = \sum_n y_n$

Another approach

RSS is the Euclidean norm squared:

$$\begin{aligned} \mathrm{RSS}(\boldsymbol{w}) &= \sum_n (\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n - y_n)^2 = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 \\ &= (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^{\mathrm{T}} (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) \\ &= \left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}} - \boldsymbol{y}^{\mathrm{T}}\right)(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) \\ &= \boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - \boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w} - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w} \\ &= \boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - 2\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w} + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w} \end{aligned}$$

Note: $\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w} = \left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}\right)^{\mathrm{T}}$

Multivariate Calculus

RSS is given by

$$\mathrm{RSS}(\boldsymbol{w}) = \boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - 2\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w} + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w}$$

$$\begin{aligned} \nabla\mathrm{RSS}(\boldsymbol{w}) &= \nabla\left(\boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - 2\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w} + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w}\right) \\ &= 0 - 2\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} + \nabla\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w}\right) \\ &= -2\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} + 2\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w} \qquad (\textit{prove it !}) \end{aligned}$$

It follows

$$(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})\boldsymbol{w} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{0} \quad \Rightarrow \quad \boldsymbol{w}^* = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$

Multivariate Calculus

Is it a minimizer?

$$\boldsymbol{w}^* = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$

We will use a second derivative (the **Hessian** matrix)

$$\nabla^2\mathrm{RSS}(\boldsymbol{w}) = \nabla\left(-2\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} + 2\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w}\right) = 2\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$$

A symmetric matrix M is said to be a **positive semi-definite** (PSD) if $u^{\mathrm{T}}Mu \geq 0$ for any vector u.

Note: $u^{\mathrm{T}}\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)u = (\boldsymbol{X}u)^{\mathrm{T}}\boldsymbol{X}u = \|\boldsymbol{X}u\|_2^2 \geq 0$ and is 0 if $u = 0$.

The Hessian matrix of a convex function is positive semi-definite.

Computational complexity

Bottleneck of computing

$$\boldsymbol{w}^* = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$

is to invert the matrix $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} \in \mathbb{R}^{(\mathsf{D}+1)\times(\mathsf{D}+1)}$

- naively need $O(\mathsf{D}^3)$ time
- there are many faster approaches (such as conjugate gradient)

What if $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ is not invertible

Why would that happen?

One situation: $\mathsf{N} < \mathsf{D}+1$, i.e. not enough data to estimate all parameters.

Example: $\mathsf{D} = \mathsf{N} = 1$

sqft	sale price
1000	500K

Any line passing through this single point is a minimizer of RSS.

How about the following?

$\mathsf{D} = 1, \mathsf{N} = 2$

sqft	sale price
1000	500K
1000	600K

Any line passing **the average** is a minimizer of RSS.

$\mathsf{D} = 2, \mathsf{N} = 3$**?**

sqft	#bedroom	sale price
1000	2	500K
1500	3	700K
2000	4	800K

Again *infinitely many minimizers*. How to resolve this issue?

Eigendecomposition

The decomposition of a square matrix A into matrices composed of its *eigenvectors* and *eigenvalues* is called eigendecomposition.

$$A = U\Lambda U^{-1}$$

where Λ is a diagonal matrix of eigenvalues of A, and each column of U is an eigenvector of A.

If A is symmetric $U^{\mathrm{T}}U = \boldsymbol{I}$, then

$$A = U\Lambda U^{-1} = U\Lambda U^{\mathrm{T}} = U^{\mathrm{T}}\Lambda U$$

and its inverse

$$A^{-1} = U^{\mathrm{T}}\Lambda^{-1}U$$

How to resolve this issue?

Eigendecomposition:

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} = \boldsymbol{U}^{\mathrm{T}} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \lambda_{\mathrm{D}} & 0 \\ 0 & \cdots & 0 & \lambda_{\mathrm{D}+1} \end{bmatrix} \boldsymbol{U}$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \lambda_{\mathrm{D}+1} \geq 0$ are **eigenvalues**.

Inverse:

$$(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1} = \boldsymbol{U}^{\mathrm{T}} \begin{bmatrix} \frac{1}{\lambda_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{\lambda_2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{1}{\lambda_{\mathrm{D}}} & 0 \\ 0 & \cdots & 0 & \frac{1}{\lambda_{\mathrm{D}+1}} \end{bmatrix} \boldsymbol{U}$$

How to solve this problem?

Non-invertible $\Rightarrow$ some eigenvalues are 0.

One natural fix: add something positive

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda\boldsymbol{I} = \boldsymbol{U}^{\mathrm{T}} \begin{bmatrix} \lambda_1+\lambda & 0 & \cdots & 0 \\ 0 & \lambda_2+\lambda & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \lambda_{\mathrm{D}}+\lambda & 0 \\ 0 & \cdots & 0 & \lambda_{\mathrm{D}+1}+\lambda \end{bmatrix} \boldsymbol{U}$$

where $\lambda > 0$ and $\boldsymbol{I}$ is the identity matrix. Now it is invertible:

$$(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda\boldsymbol{I})^{-1} = \boldsymbol{U}^{\mathrm{T}} \begin{bmatrix} \frac{1}{\lambda_1+\lambda} & 0 & \cdots & 0 \\ 0 & \frac{1}{\lambda_2+\lambda} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{1}{\lambda_{\mathrm{D}}+\lambda} & 0 \\ 0 & \cdots & 0 & \frac{1}{\lambda_{\mathrm{D}+1}+\lambda} \end{bmatrix} \boldsymbol{U}$$

Solution

The solution becomes

$$\boldsymbol{w}^* = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$

- not a minimizer of the original RSS

λ is a *hyper-parameter*, can be tuned by cross-validation.

How do we predict?

$$f(\boldsymbol{x}) = \boldsymbol{w}^{*\mathrm{T}}\boldsymbol{x}$$

Comparison to NNC

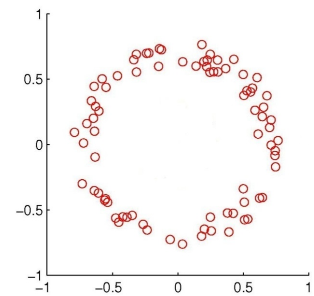
Parametric versus non-parametric

- **Parametric methods**: the size of the model does *not grow* with the size of the training set N.
  - e.g. linear regression, Naive Bayes
- **Non-parametric methods**: the size of the model *grows* with the size of the training set.
  - NNC, Decision Trees

Outline

What if linear model is not a good fit?

Example: a straight line is a bad fit for the following data



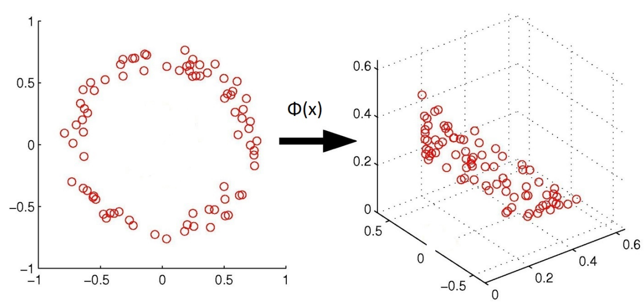
Solution: nonlinearly transformed features

1. Use a nonlinear mapping

$$\boldsymbol{\phi}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^D \rightarrow \boldsymbol{z} \in \mathbb{R}^M$$

to transform the data to a more complicated feature space

2. Then apply linear regression (hope: linear model is a better fit for the new feature space).



Regression with nonlinear basis

Model: $f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$ where $\boldsymbol{w} \in \mathbb{R}^M$

Objective:

$$\mathrm{RSS}(\boldsymbol{w}) = \sum_n \left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) - y_n\right)^2$$

Similar least square solution:

$$\boldsymbol{w}^* = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y} \quad \text{where} \quad \boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathrm{T}} \\ \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}_N)^{\mathrm{T}} \end{pmatrix} \in \mathbb{R}^{N \times M}$$

Example

Polynomial basis functions for $\mathsf{D} = 1$

$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix} \quad \Rightarrow \quad f(x) = w_0 + \sum_{m=1}^{M} w_m x^m$$

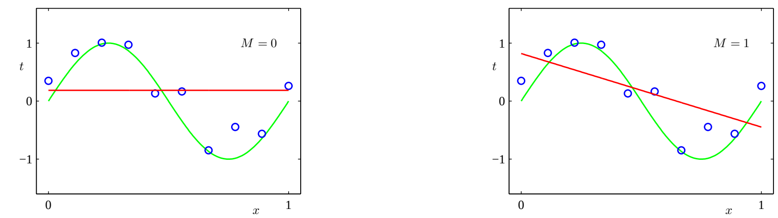
General case:

$$\boldsymbol{\phi}(x) = \left[\prod_{i=1}^{D} x_i^{a_i} \quad \text{, s.t.} \quad \sum_{i=1}^{D} a_i \le M \right]$$

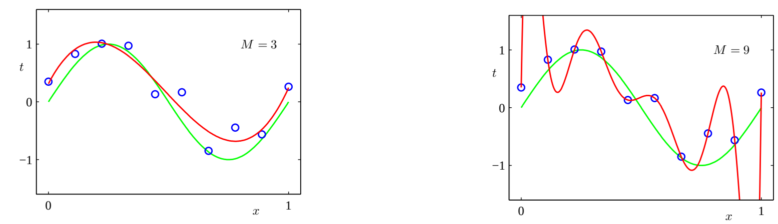
Learning a linear model in the new space
$=$ learning an *M-degree polynomial model* in the original space

Example

Fitting a sine function with a polynomial ($M = 0, 1,$ or 3):



M=9: *overfitting*



Why nonlinear?

Can I use a fancy **linear** feature map? For example,

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{bmatrix} x_1 - x_2 \\ 3x_4 - x_3 \\ 2x_1 + x_4 + x_5 \\ \vdots \end{bmatrix} = \boldsymbol{A}\boldsymbol{x} \quad \text{for some } \boldsymbol{A} \in \mathbb{R}^{\mathsf{M}\times\mathsf{D}}$$

No, it basically *does nothing* since

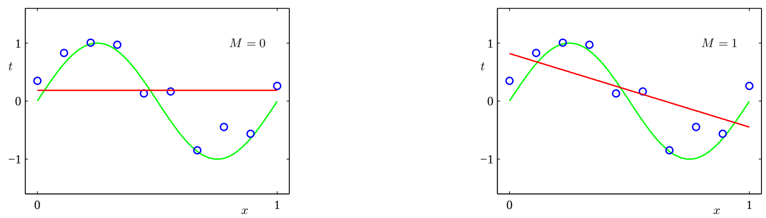
$$\min_{\boldsymbol{w}\in\mathbb{R}^{\mathsf{M}}} \sum_n \left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}_n - y_n\right)^2 = \min_{\boldsymbol{w}'\in\mathsf{Im}(\boldsymbol{A}^{\mathrm{T}})\subset\mathbb{R}^{\mathsf{D}}} \sum_n \left(\boldsymbol{w}'^{\mathrm{T}}\boldsymbol{x}_n - y_n\right)^2$$

We will see more nonlinear mappings soon.

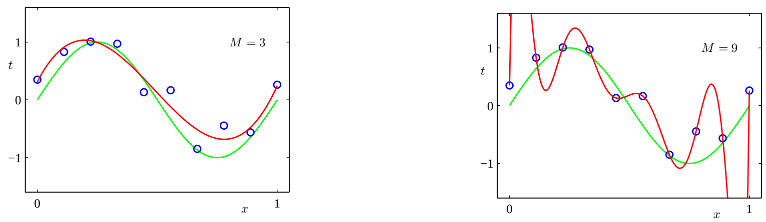
Outline

Should we use a very complicated mapping?

Ex: fitting a sine function with a polynomial:



Training error is zero:



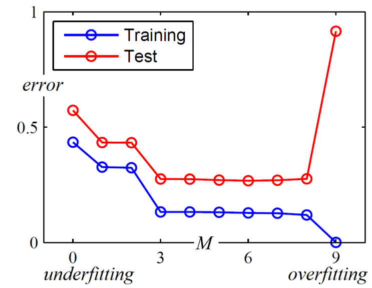
Underfitting and Overfitting

$M \leq 2$ is *underfitting* the data

- large training error
- large test error

$M \geq 9$ is *overfitting* the data

- small training error
- large test error

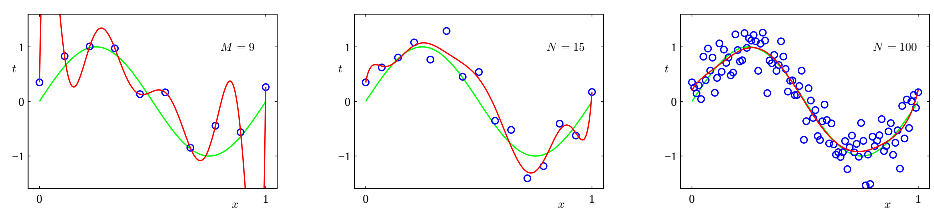


More complicated models $\Rightarrow$ larger gap between training and test error

How to prevent overfitting?

Method 1: use more training data

The more, the merrier. We increase N - the number of training points.



More data $\Rightarrow$ smaller gap between training and test error

Method 2: control the model complexity

For polynomial basis, the **degree** M controls the complexity

	$M=0$	$M=1$	$M=3$	$M=9$
w_0	0.19	0.82	0.31	0.35
w_1		-1.27	7.99	232.37
w_2			-25.43	-5321.83
w_3			17.37	48568.31
w_4				-231639.30
w_5				640042.26
w_6				-1061800.52
w_7				1042400.18
w_8				-557682.99
w_9				125201.43

Intuitively, *large weights $\Rightarrow$ more complex model*

Use cross-validation to pick hyperparameter M

Are there still other ways to control complexity?

How to make $\boldsymbol{w}$ small?

Regularized linear regression: new objective

$$\mathcal{E}(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) + \lambda R(\boldsymbol{w})$$

Goal: find $\boldsymbol{w}^* = \operatorname{argmin}_w \mathcal{E}(\boldsymbol{w})$

- $R : \mathbb{R}^{\mathsf{D}} \to \mathbb{R}^+$ is the *regularizer*
  - measure how complex the model $\boldsymbol{w}$ is
  - common choices: $\|\boldsymbol{w}\|_2^2$, $\|\boldsymbol{w}\|_1$, etc.
- $\lambda > 0$ is the *regularization coefficient*
  - $\lambda = 0$, no regularization
  - $\lambda \to +\infty$, $\boldsymbol{w} \to \operatorname{argmin}_w R(\boldsymbol{w})$
  - i.e. control **trade-off** between training error and complexity

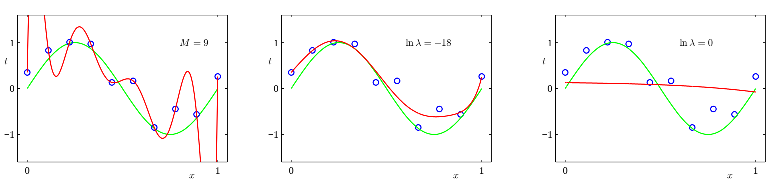
The effect of λ

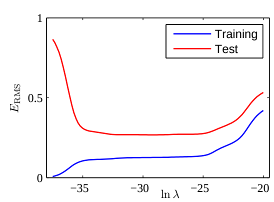
when we increase regularization coefficient λ

	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
w_0	0.35	0.35	0.13
w_1	232.37	4.74	-0.05
w_2	-5321.83	-0.77	-0.06
w_3	48568.31	-31.97	-0.06
w_4	-231639.30	-3.89	-0.03
w_5	640042.26	55.28	-0.02
w_6	-1061800.52	41.32	-0.01
w_7	1042400.18	-45.95	-0.00
w_8	-557682.99	-91.53	0.00
w_9	125201.43	72.68	0.01

The trade-off

When we increase regularization coefficient λ, overfitting decreases:





How to choose the right amount of regularization?

Can we tune λ on the training dataset?

No: as this will set λ to zero, i.e., without regularization, defeating our intention to use it to control model complexity and to gain better generalization.

λ is a hyperparameter. To tune it,

- We can use a development/holdout dataset independent of training and testing dataset.
- We can use cross-validation.

The procedure is similar to choose K in the nearest neighbor classifiers.

The root of overfitting

Dealing with over and underfitting is really about dealing with bias and variance.

Mathematically, the expected prediction error can be decomposed into bias and variance components.

Simpler models have a smaller variance but a larger bias.

Complex models have a larger variance but a smaller bias.

Thus, we balance bias and variance by choosing λ.

Regularization reduces variance (because they lead to simpler models) but then increase the bias.

How to solve the new objective?

Simple for $R(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2$:

$$\mathcal{E}(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|_2^2 = \|\boldsymbol{\Phi w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$$

$$\begin{aligned}
&\nabla\mathcal{E}(\boldsymbol{w}) = 2(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi w} - \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}) + 2\lambda\boldsymbol{w} = 0 \\
&\Rightarrow \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda\boldsymbol{I}\right)\boldsymbol{w} = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y} \\
&\Rightarrow \boldsymbol{w}^* = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}
\end{aligned}$$

Note the same form as in the fix when $\boldsymbol{X}^T\boldsymbol{X}$ is not invertible!

For other regularizers, as long as it's **convex**, standard optimization algorithms can be applied.

Equivalent form

Regularization is also sometimes formulated as

$$\operatorname*{argmin}_{\boldsymbol{w}} \mathrm{RSS}(w) \qquad \textbf{subject to } R(\boldsymbol{w}) \leq \beta$$

where β is some hyperparameter.

Finding the solution becomes a *constrained optimization problem*.

Choosing either λ or β can be done by cross-validation.

Summary

Linear regression summarized:

$$\begin{aligned}
\boldsymbol{w}^* &= \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} \\
\boldsymbol{w}^* &= \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} \\
\boldsymbol{w}^* &= \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}
\end{aligned}$$

It is important to understand the derivation.

Overfitting: small training error but large test error.

Preventing Overfitting: more data and/or regularization.

Typical steps

Typical steps of developing a machine learning system:

- Collect data, split into training, development, and test sets.
- *Train an ML model* with training data to learn from.
- Evaluate it using the test data and report performance.
- Use the model to predict future/make decisions.

How to do the *red part* exactly?

General idea to provide ML algorithms

1. Pick a set of **models** $\mathcal{F}$
   - e.g. $\mathcal{F} = \{f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \mid \boldsymbol{w} \in \mathbb{R}^{\mathsf{D}}\}$
   - e.g. $\mathcal{F} = \{f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Phi}(\boldsymbol{x}) \mid \boldsymbol{w} \in \mathbb{R}^{\mathsf{M}}\}$
2. Define **error/loss** $L(y', y)$
3. Find **empirical risk minimizer (ERM)**:

$$\boldsymbol{f}^* = \operatorname*{argmin}_{f \in \mathcal{F}} \sum_{n=1}^{N} L(f(x_n), y_n)$$

or **regularized empirical risk minimizer**:

$$\boldsymbol{f}^* = \operatorname*{argmin}_{f \in \mathcal{F}} \sum_{n=1}^{N} L(f(x_n), y_n) + \lambda R(f)$$

ML becomes optimization