Intro

...based on Toronto University ML class

What is learning?

“The activity or process of gaining knowledge or skill by studying, practicing, being taught, or experiencing something."

Merriam Webster dictionary

L'attività di imparare, nel caso delle macchine queste dovrebbero migliorare con esperienze, valutando ciò in base ad indici di prestazioni

What is learning?

“The activity or process of gaining knowledge or skill by studying, practicing, being taught, or experiencing something."

Merriam Webster dictionary

“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.”

Tom Mitchell

What is machine learning?

ne sono esempi il riconoscimento dello spam, ma ciò sarebbe meglio se adattivo, ovvero non devo inserire io programmatore tutte le parole potenzialmente usate in un messaggio di spam.

- For many problems, it's difficult to program the correct behavior by hand
  - recognizing people and objects
  - understanding human speech
- Machine learning approach: program an algorithm to automatically learn from data, or from experience
- Why might you want to use a learning algorithm?
  - hard to code up a solution by hand (e.g. vision, speech)
  - system needs to adapt to a changing environment (e.g. spam detection)
  - want the system to perform better than the human programmers

What is machine learning?

- It's similar to statistics...
  - Both fields try to uncover patterns in data
  - Both fields draw heavily on calculus, probability, and linear algebra, and share many of the same core algorithms
- But it's not statistics!
  - Stats is more concerned with helping scientists and policymakers draw good conclusions; ML is more concerned with building autonomous agents
  - Stats puts more emphasis on interpretability and mathematical rigor; ML puts more emphasis on predictive performance, scalability, and autonomy

Relations to AI

ML è un "pezzo" di IA

- Nowadays, machine learning is often brought up with artificial intelligence (AI)
- AI does not always imply a learning based system
  - Symbolic reasoning
  - Rule based system
  - Tree search
  - etc.
- Learning based system $\rightarrow$ learned based on the data $\rightarrow$ more flexibility, good at solving pattern recognition problems.

Relations to human learning

- Human learning is:
  - Very data efficient
  - An entire multitasking system (vision, language, motor control, etc.)
  - Takes at least a few years :)
- For serving specific purposes, machine learning doesn't have to look like human learning in the end.
- It may borrow ideas from biological systems, e.g., neural networks.
- It may perform better or worse than humans.

What is machine learnng

- Types of machine learning
  - **Supervised learning:** have labeled examples of the correct behavior
  - **Reinforcement learning:** learning system (agent) interacts with the world and learns to maximize a scalar reward signal
  - **Unsupervised learning:** no labeled examples instead, looking for interesting patterns in the data

Nel supervised ho esempi di comportamento corretto (etichette), nel reinforcement learning ho interazione con il mondo esterno ai fini di apprendere. Con unsupervised non ci sono etichette.

Posso migliorare perchè le componenti neurali non sono "lineari", in tal caso potrei fare ben poco.

History of Machine Learning

- 1957 - Perceptron algorithm (implemented as a circuit!)
- 1959 - Arthur Samuel wrote a learning-based checkers program that could defeat him
- 1969 - Minsky and Papert's book *Perceptrons* (limitations of linear models)
- 1980s - Some foundational ideas
  - Connectionist psychologists explored neural models of cognition
  - 1984 - Leslie Valiant formalized the problem of learning as PAC (Probably Approximately Correct) learning
  - 1988 - Backpropagation (re-)discovered by Geoffrey Hinton and colleagues
  - 1988 - Judea Pearl's book *Probabilistic Reasoning in Intelligent Systems* introduced Bayesian networks

History of Machine Learning

- 1990s - the “AI Winter”, a time of pessimism and low funding
  But looking back, the ’90s were also sort of a golden age for ML research
  - Markov chain Monte Carlo
  - variational inference
  - kernels and support vector machines
  - boosting
  - convolutional networks
  - reinforcement learning
- 2000s - applied AI fields (vision, NLP, etc.) adopted ML
- 2010s - deep learning
  - 2010-2012 - neural nets smashed previous records in speech-to-text and object recognition
  - increasing adoption by the tech industry
  - 2016 - AlphaGo defeated the human Go champion
  - 2018-now - generating photorealistic images and videos
  - 2020 - GPT3 language model
- now - increasing attention to ethical and societal implications

Applications

Computer vision: Object detection, semantic segmentation, pose estimation, and almost every other task is done with ML.

Figure 4. More results of **Mask R-CNN** on COCO test images, using ResNet-101-FPN and running at 5 fps, with 35.7 mask AP (Table 1).

Instance segmentation - Link

DAQUAR 1553
What is there in front of the sofa?
Ground truth: table
IMG+BOW: table (0.74)
2-VIS+BLSTM: table (0.88)
LSTM: chair (0.47)

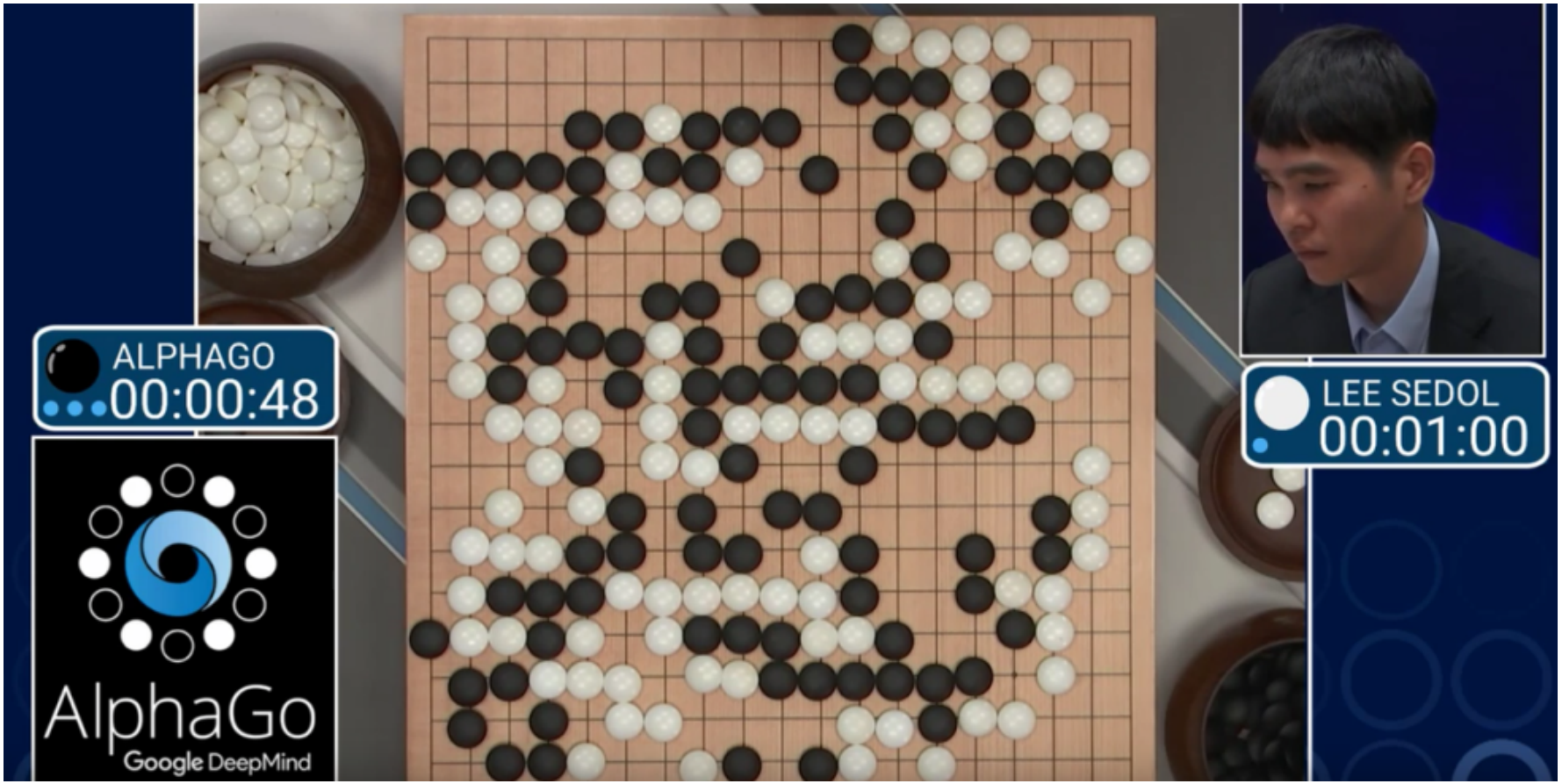
COCOQA 5078
How many leftover donuts is the red bicycle holding?
Ground truth: three
IMG+BOW: two (0.51)
2-VIS+BLSTM: three (0.27)
BOW: one (0.29)

Applications

Speech: Speech to text, personal assistants, speaker identification...

Applications

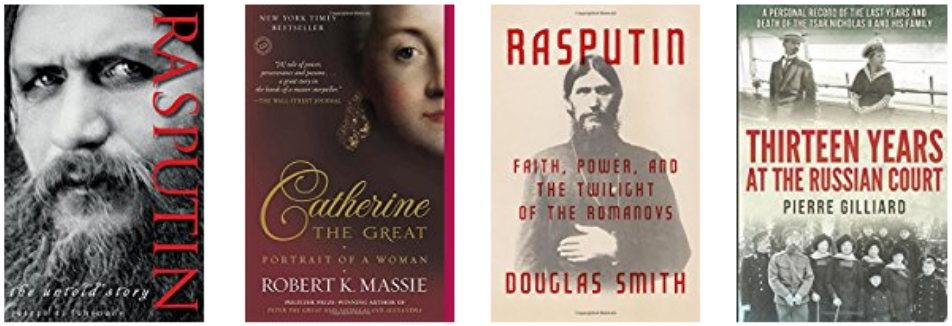
Playing Games



Applications

E-commerce & Recommender Systems : Amazon, netflix, ...

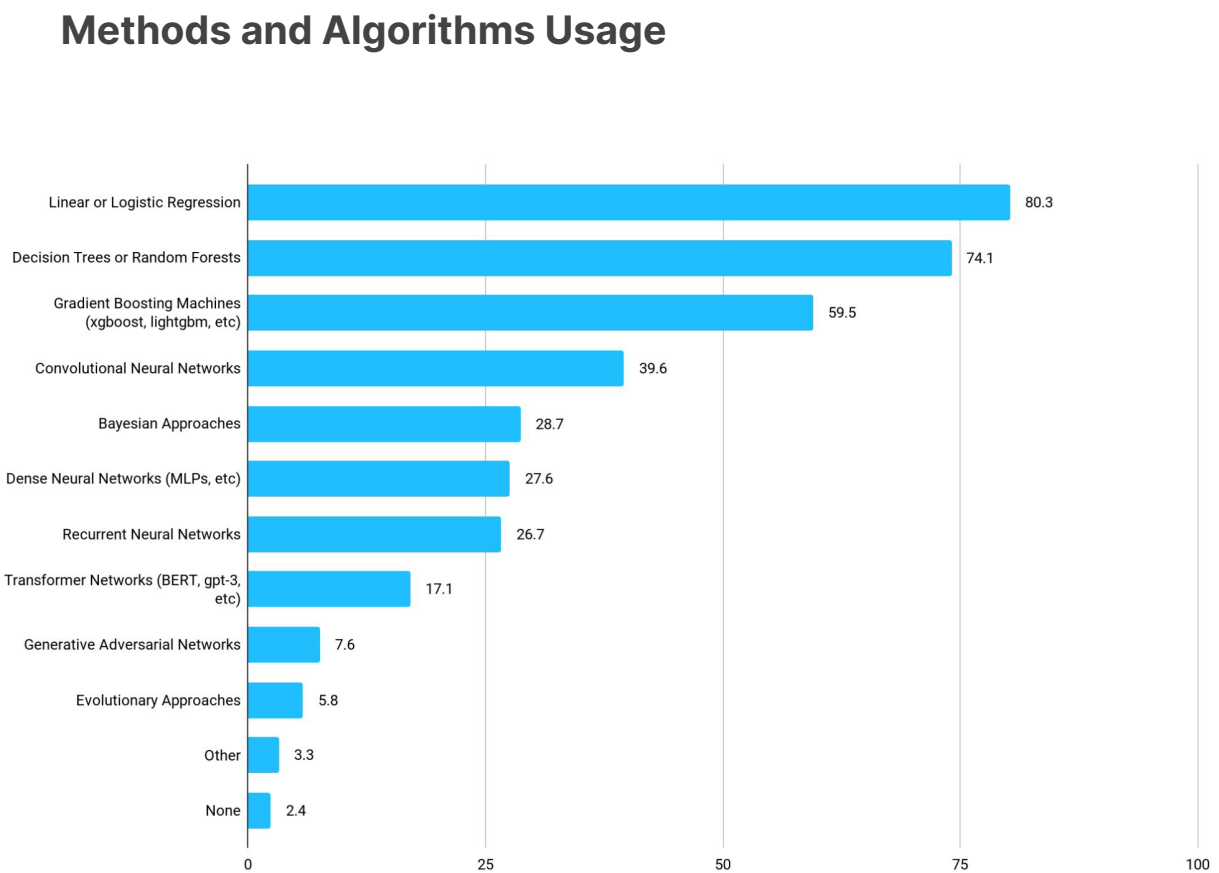
Inspired by your shopping trends



Related to items you've viewed See more

Why this class

2022 Kaggle survey of data science and ML practitioners: what data science methods do you use at work?



ML Workflow

- ML workflow sketch:
  1. Should I use ML on this problem?
     - Is there a pattern to detect?
     - Can I solve it analytically?
     - Do I have data?
  2. Gather and organize data.
     - Preprocessing, cleaning, visualizing.
  3. Establishing a baseline.
  4. Choosing a model, loss, regularization, ...
  5. Optimization
  6. Hyperparameter search.
  7. Analyze performance & mistakes, and iterate back to step 4 (or 2).

Preliminaries and Nearest Neighbor Methods

...based on Toronto University ML class

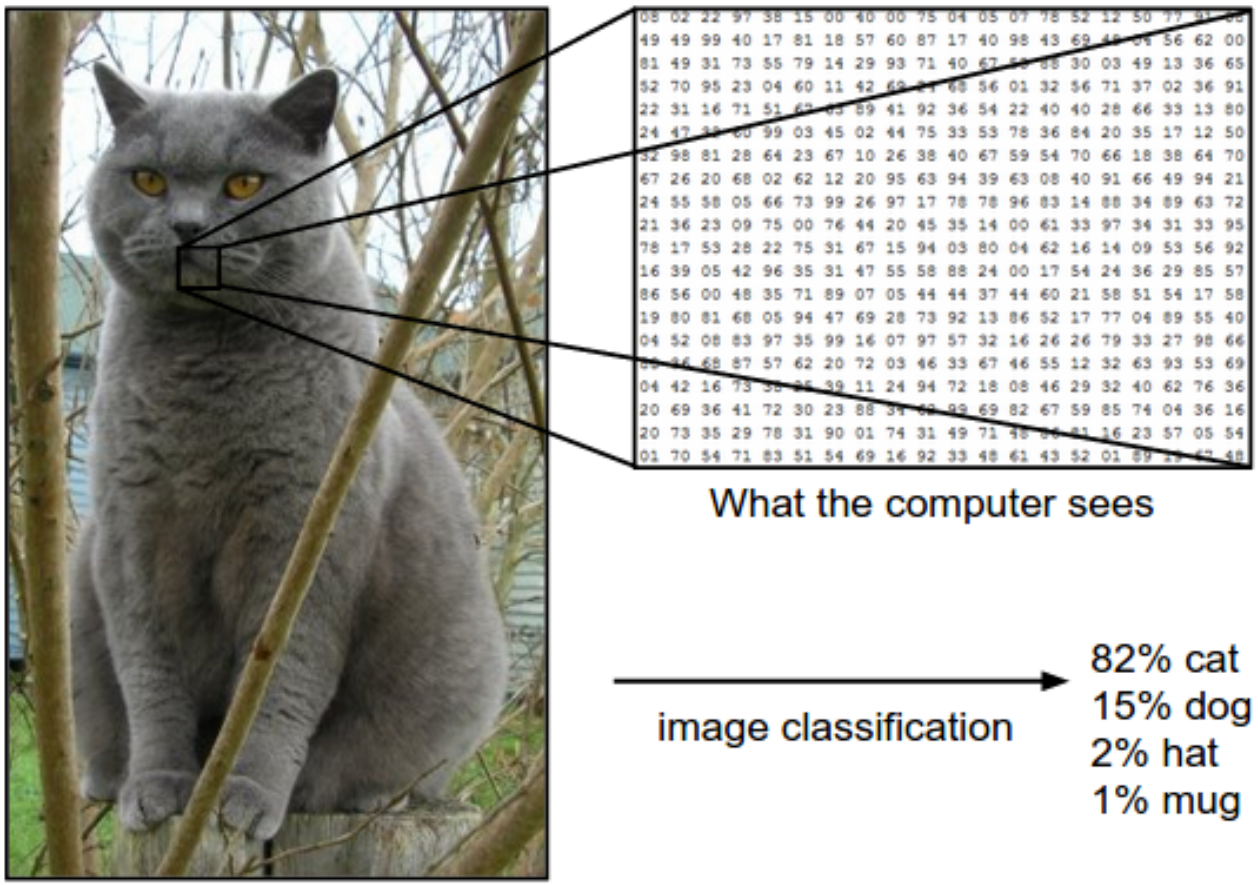
Introduction

- We start with supervised learning.
- We are given a training set consisting of inputs and corresponding labels, e.g.

Task	Inputs	Labels
object recognition	image	object category
image captioning	image	caption
document classification	text	document category
speech-to-text	audio waveform	text
⋮	⋮	⋮

Input Vectors

What an image looks like to a computer

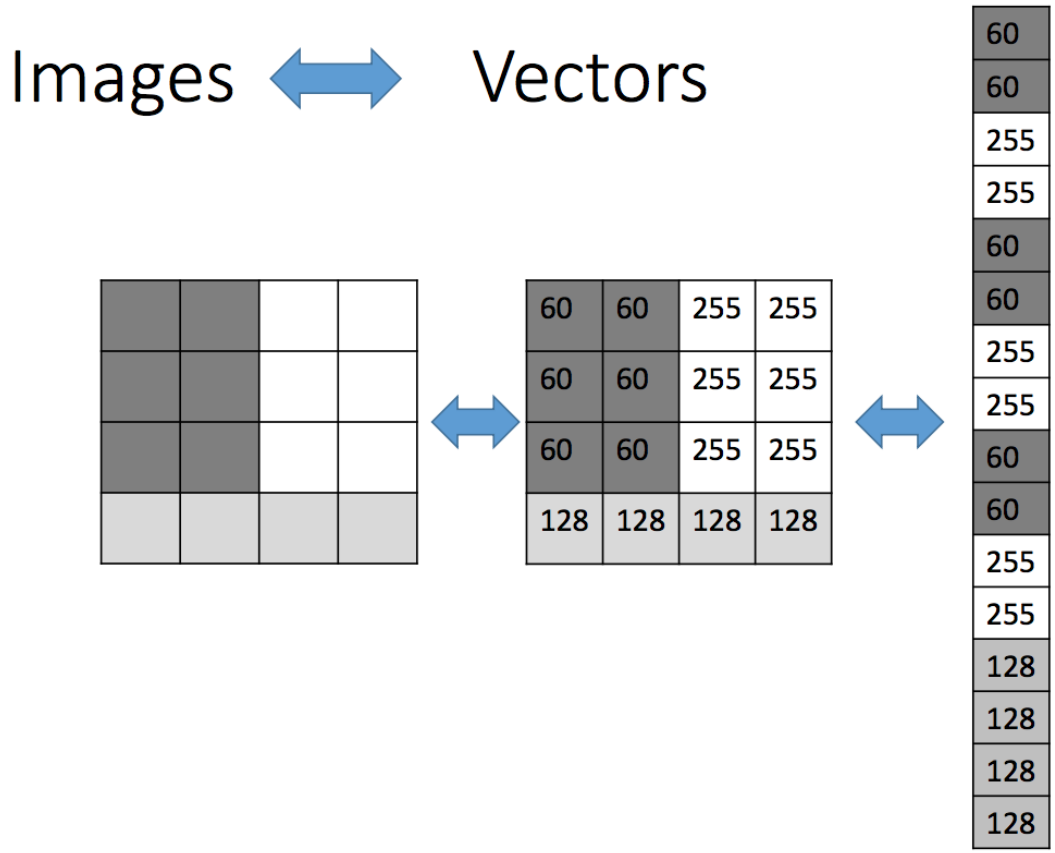


Input Vectors

- Machine learning algorithms need to handle lots of types of data: images, text, audio waveforms, credit card transactions, etc.
- Common strategy: represent the input as an input vector in $\mathbb{R}^d$
  - Representation = mapping to another space that's easy to manipulate
  - Vectors are a great representation since we can do linear algebra!

Input Vectors

Can use raw pixels



Can do much better if you compute a vector of meaningful features.

Input Vectors

- Mathematically, our training set consists of a collection of pairs of an input vector $\mathbf{x} \in \mathbb{R}^d$ and its corresponding target, or label, t
  - Regression: t is a real number (e.g. stock price)
  - Classification: t is an element of a discrete set $\{1, \ldots, C\}$
- Denote the training set $\{(\mathbf{x}^{(1)}; t^{(1)}), \ldots, (\mathbf{x}^{(N)}; t^{(N)})\}$

Nearest Neighbors

di tipo supervised, parto da set di dati (input,img,text che vedo come vettore) ed etichette.

- Suppose we're given a novel input vector $\mathbf{x}$ we'd like to classify.
- The idea: find the nearest input vector to $\mathbf{x}$ in the training set and copy its label.
- Can formalize nearest in terms of Euclidean distance

$$\| \mathbf{x}^{(a)} - \mathbf{x}^{(b)} \|_2 = \sqrt{\sum_{j=1}^{d} (x_j^{(a)} - x_j^{(b)})^2}$$

un esempio è la categorizzazione di animali partendo da una immagine

Algorithm

1. Find example (x^*, t^*) (from the stored training set) closest to $\mathbf{x}$. That is:

$$\mathbf{x}^* = \arg \min_{\mathbf{x}^{(i)} \in \text{training set}} distance(\mathbf{x}^{(i)}, \mathbf{x})$$

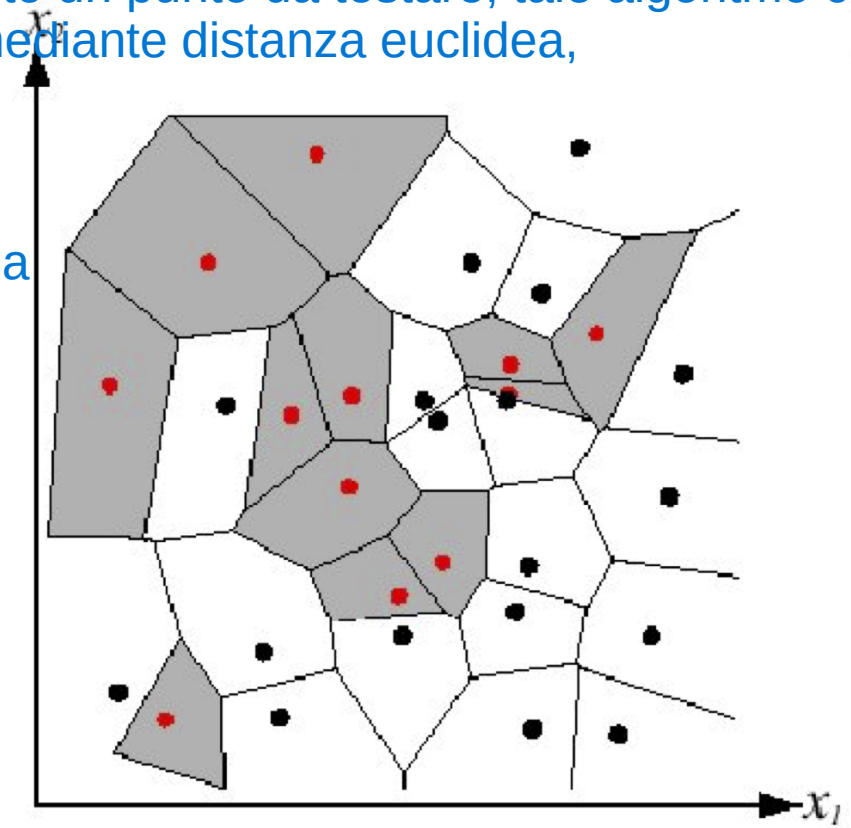
2. Output $y = t^*$

il set di dati training è come se fossero degli "esempi".

Nearest Neighbors: Decision Boundaries

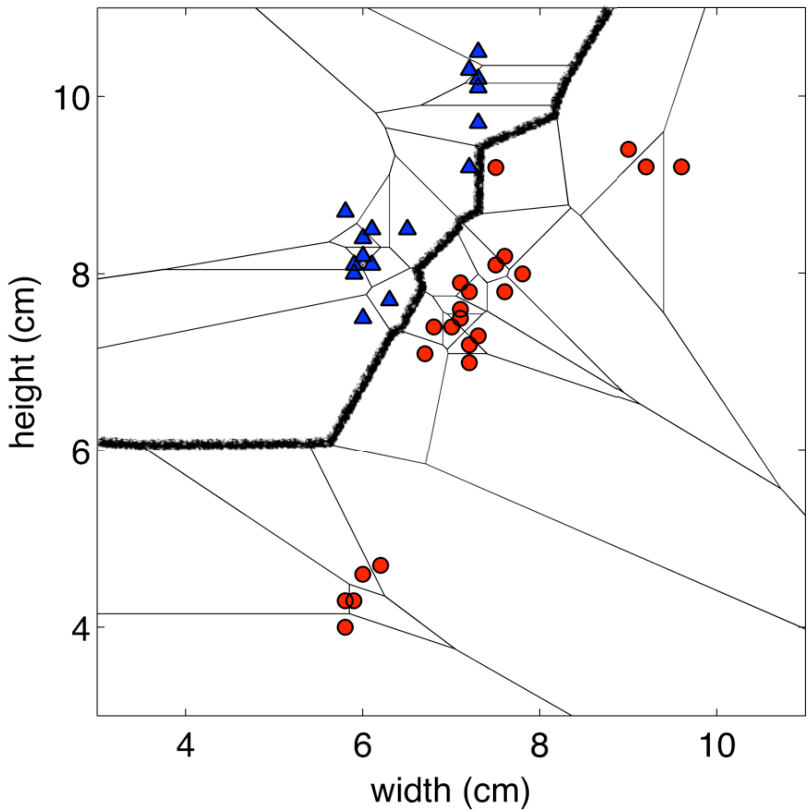
We can visualize the behavior in the classification setting using a Voronoi diagram.

nel training abbiamo delle coppie di punti a cui associo una etichetta (ad esempio true/false). Se viene fornito un punto da testare, tale algoritmo cerca il punto più vicino al nuovo punto inserito mediante distanza euclidea,
vede la sua etichetta,
ed userà questa per
il nuovo punto.
Devo calcolare la distanza da ogni punto, non so a priori quale sia il più vicino!



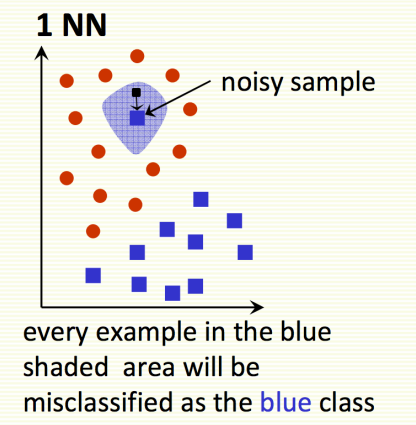
Nearest Neighbors: Decision Boundaries

Decision boundary: the boundary between regions of input space assigned to different categories.



In questa variante non lavoro in aree, ma con una linea di delimitazione.

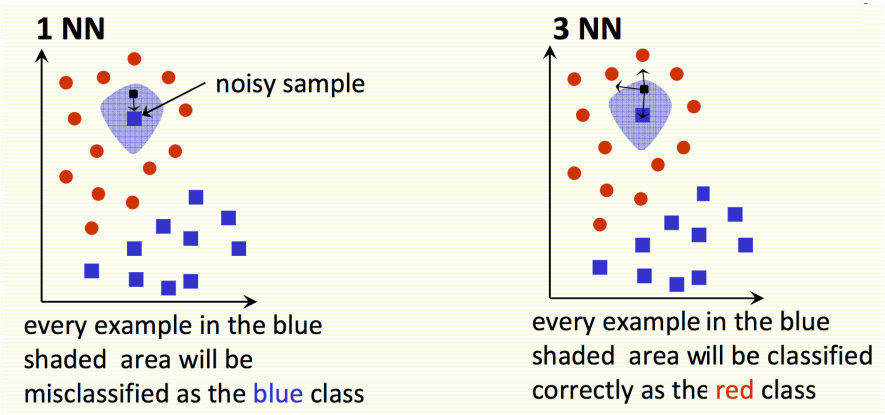
Nearest Neighbors



- Nearest neighbors sensitive to noise or mis-labeled data ("class noise").

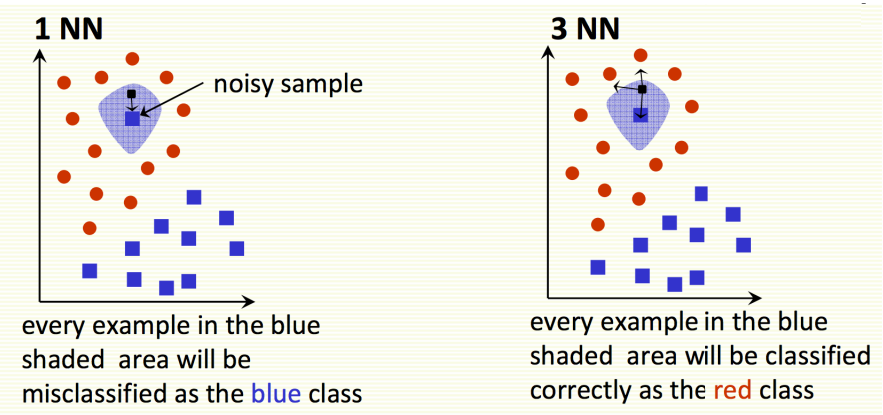
Nella pratica questo algoritmo va bene se non ci sono errori nel training (ad esempio ho sbagliato una label o una posizione). Se così non fosse, tale rumore si propaga molto velocemente.

Nearest Neighbors



- Nearest neighbors sensitive to noise or mis-labeled data ("class noise"). Solution?

Nearest Neighbors



- Nearest neighbors sensitive to noise or mis-labeled data ("class noise"). Solution?
- Smooth by having k-nearest neighbors vote

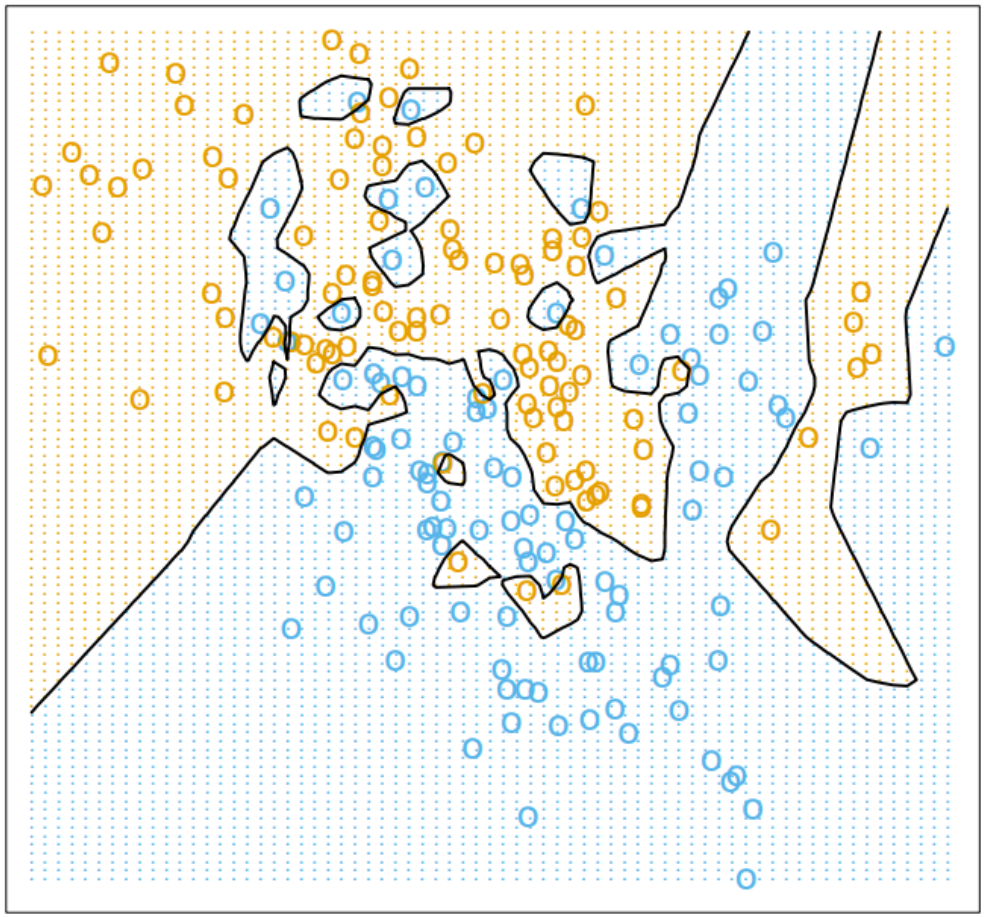
Algorithm

1. Find k examples $(x^{(i)}, t^{(i)})$ closest to the test instance $\mathbf{x}$.
2. Classification output is majority class

$$y = \arg\max_c \sum_{i=1}^{k} \mathbb{I}_{\{t^{(i)}=c\}}$$

K-Nearest Neighbors

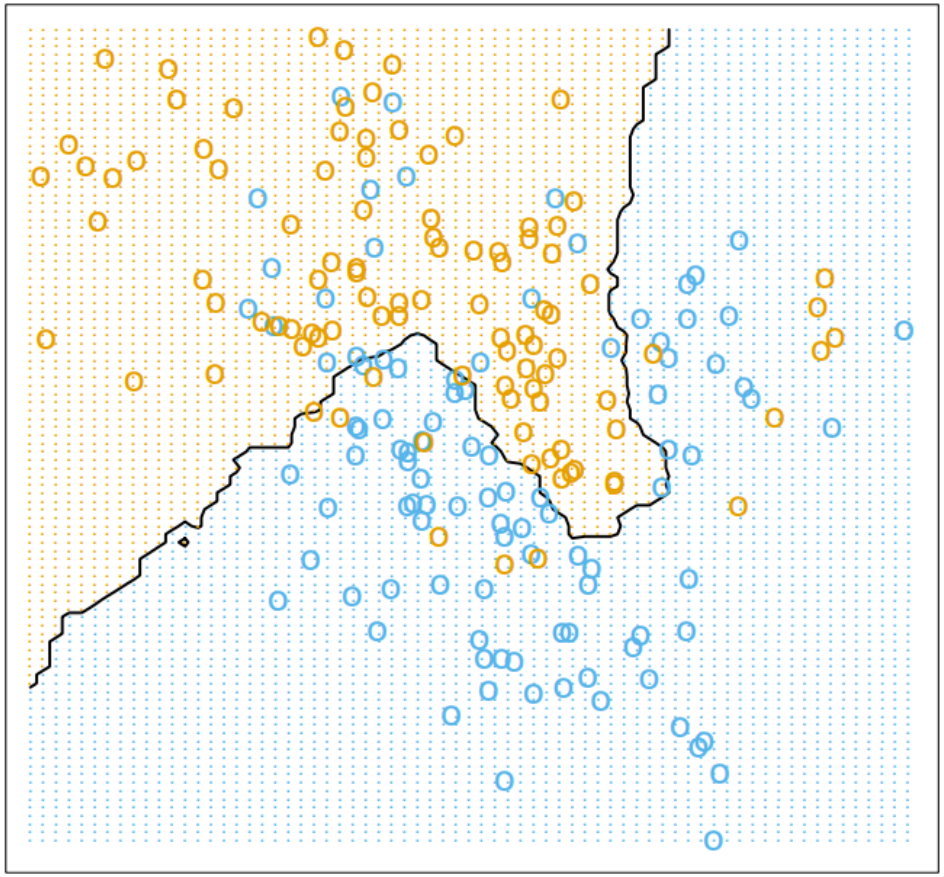
k=1



K-Nearest Neighbors

in questa variante non vedo l'etichetta del nodo più vicino, ma l'etichetta maggiormente presente nei k nodi vicini al nodo. Più k cresce, più la demarcazione è rigida.

k=15



K-Nearest Neighbors

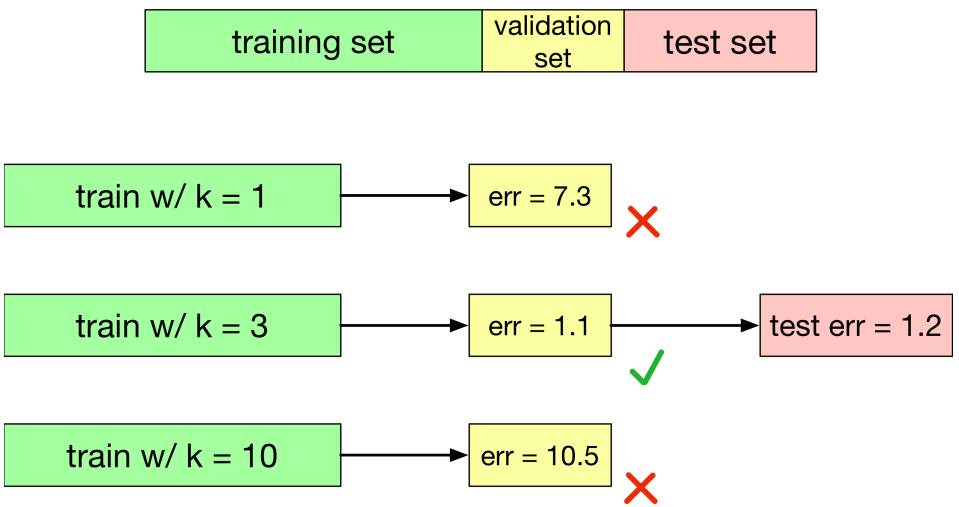
Come scelgo il k migliore? Parliamo di "k migliore" perchè non possiamo parlare di k corretto.

Tradeoffs in choosing k?

- Small k
  - Good at capturing fine-grained patterns
  - May overfit, i.e. be sensitive to random idiosyncrasies in the training data
- Large k
  - Makes stable predictions by averaging over lots of examples
  - May underfit, i.e. fail to capture important regularities
- Balancing k
  - Optimal choice of k depends on number of data points n.
  - Nice theoretical properties if $k \to \infty$ and $\frac{k}{n} \to 0$.
  - Rule of thumb: choose $k < \sqrt{n}$.
  - We can choose k using validation set (next slides).

K-Nearest Neighbors

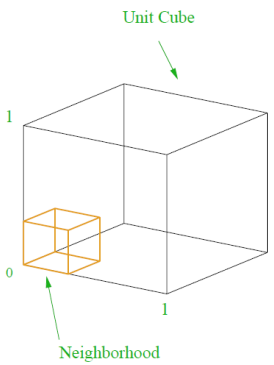
- We would like our algorithm to generalize to data it hasn't seen before.
- We can measure the generalization error (error rate on new examples) using a test set.
- k is an example of a hyperparameter, something we can't fit as part of the learning algorithm itself
- We can tune hyperparameters using a validation set:

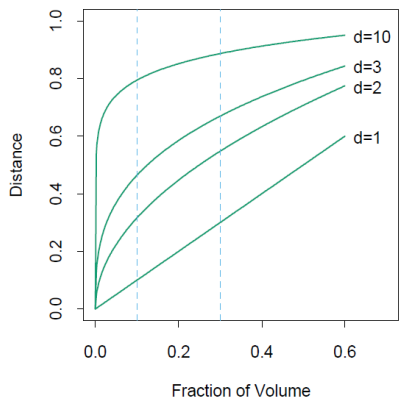


- The test set is used only at the very end, to measure the generalization performance of the final configuration.

Pitfalls: The Curse of Dimensionality

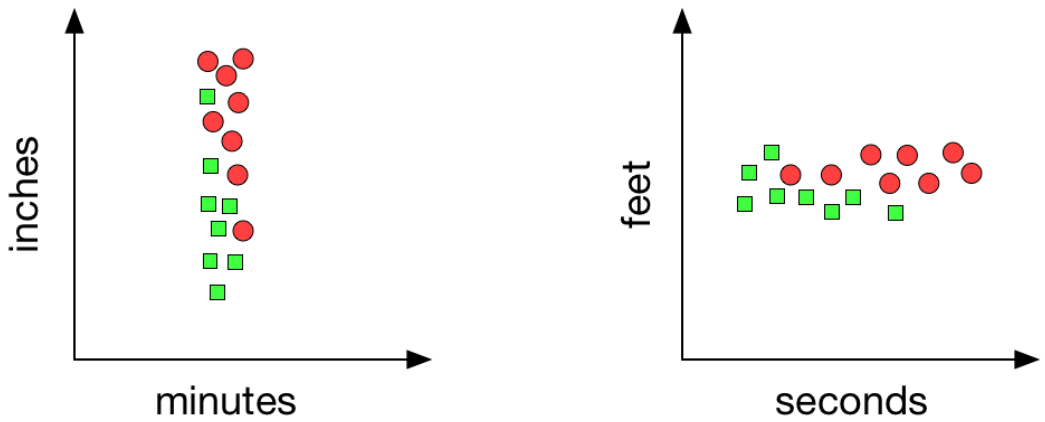
- Low-dimensional visualizations are misleading! In high dimensions, “most" points are far apart.
- If we want the nearest neighbor to be closer than ϵ, how many points do we need to guarantee it?
  - The volume of a single ball of radius ϵ is $\mathcal{O}(\epsilon^d)$
  - The total volume of $[0, 1]^d$ is 1.
  - Therefore $\mathcal{O}((\frac{1}{\epsilon})^d)$ balls are needed to cover the volume.





Pitfalls: Normalization

- Nearest neighbors can be sensitive to the ranges of different features.
- Often, the units are arbitrary:



- Simple fix: normalize each dimension to be zero mean and unit variance, i.e., compute the mean μ_j and standard deviation σ_j, and consider $\tilde{x}_j$ in place of the original x_j, where

$$\tilde{x}_j = \frac{x_j - \mu_j}{\sigma_j}$$

Pitfalls: Computational Cost

- Number of computations at training time: 0
- Number of computations at test time, per query (naive algorithm)
  - Calculuate D-dimensional Euclidean distances with N data points: $\mathcal{O}(ND)$
  - Sort the distances: $\mathcal{O}(N \log N)$
- This must be done for *each* query, which is very expensive by the standards of a learning algorithm!
- Need to store the entire dataset in memory! Tons of work has gone into algorithms and data structures for efficient nearest neighbors with high dimensions and/or large datasets.

Conclusions

- Simple algorithm that does all its work at test time - in a sense, no learning!
- Can control the complexity by varying k
- Suffers from the Curse of Dimensionality
- Next: parametric models, which learn a compact summary of the data rather than referring back to it at test time.

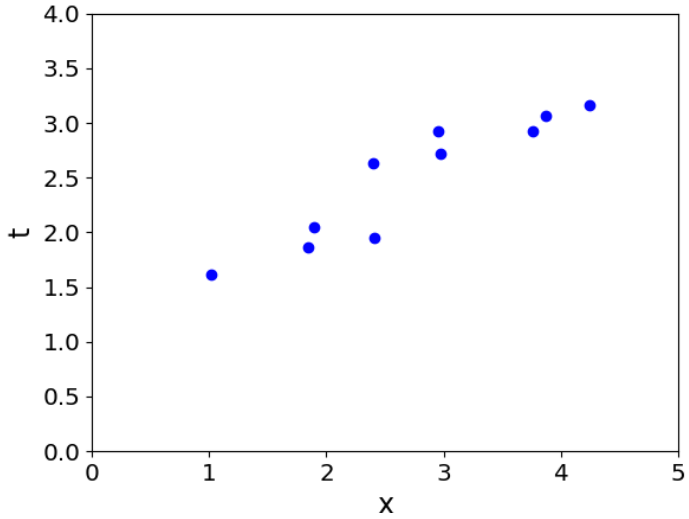
Linear Regression

...based on Toronto University ML class

Overview

- Second learning algorithm of the course: linear regression.
  - Task: predict scalar-valued targets (e.g. stock prices)
  - ...as linear function of the inputs
- While KNN was a complete algorithm, linear regression exemplifies a modular approach that will be used throughout this course:
  - choose a model describing the relationships between variables of interest
  - define a loss function quantifying how bad the fit to the data is
  - choose a regularizer saying how much we prefer different candidate models (or explanations of data)
  - fit a model that minimizes the loss function and satisfies the constraint/penalty imposed by the regularizer, possibly using an optimization algorithm
- Mixing and matching these modular components give us a lot of new ML methods.

Supervised Learning Setup



In supervised learning:

- There is input $\mathbf{x} \in \mathcal{X}$, typically a vector of features (or covariates)
- There is target $t \in \mathcal{T}$ (also called response, outcome, output, class)
- Objective is to learn a function $f : \mathcal{X} \to \mathcal{T}$ such that $t \approx y = f(x)$ based on some data $\mathcal{D} = \{(x^{(i)}, t^{(i)}) \text{ for } i = 1, 2, \ldots, N\}$

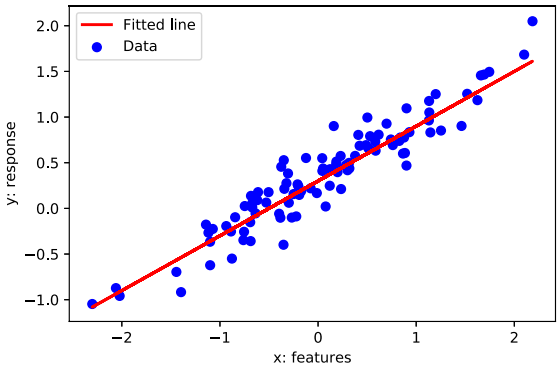
Linear Regression - Model

- Model: In linear regression, we use a linear function of the features $\mathbf{x} = (x_1, \ldots, x_D) \in \mathbb{R}^D$ to make predictions y of the target value $t \in R$:

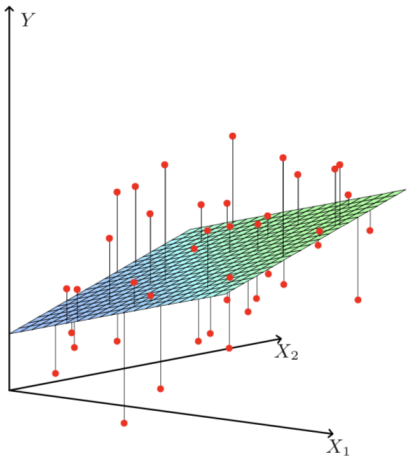
$$y = f(\mathbf{x}) = \sum_j w_j x_j + b$$

  - y is the prediction
  - $\mathbf{w}$ is a vector of weights
  - b is the bias (or intercept)
- $\mathbf{w}$ and b together are the parameters
- We hope that our prediction is close to the target: $y \approx t$.

What is linear? 1 feature vs D features



- if we have only 1 feature $y = wx + b$ where w, x and $b \in \mathbb{R}$
- y is linear in x



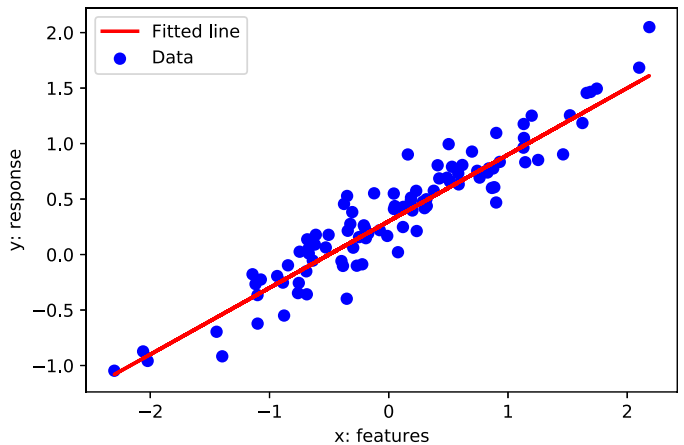
- if we have D features $y = \mathbf{w}^\top \mathbf{x} + b$ where $\mathbf{w}, \mathbf{x} \in \mathbb{R}^D$ $b \in \mathbb{R}$
- y is linear in $\mathbf{x}$

Relation between the prediction y and inputs $\mathbf{x}$ is linear in both cases.

Linear Regression

We have a dataset $\mathcal{D} = \{(x^{(i)}, t^{(i)}) \text{ for } i = 1, 2, \ldots, N\}$ where

- $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_D^{(i)})^\top \in \mathbb{R}^D$ are the inputs (e.g. age, height)
- $t^{(i)} \in \mathbb{R}$ is the target or response (e.g. income)
- predict $t^{(i)}$ with a linear function of $\mathbf{x}^{(i)}$:



- $t^{(i)} \approx y^{(i)} = \mathbf{w}^\top \mathbf{x}^{(i)} + b$
- Different $(\mathbf{w}, b)$ define different lines.
- We want the "best" line $(\mathbf{w}, b)$.
- How to quantify "best"?

Linear Regression - Loss Function

- A loss function $\mathcal{L}(y, t)$ defines how bad it is if, for some example $\mathbf{x}$, the algorithm predicts y, but the target is actually t.
- Squared error loss function:

$$\mathcal{L}(y, t) = \frac{1}{2}(y - t)^2$$

- $y - t$ is the residual, and we want to make this small in magnitude
- The $\frac{1}{2}$ factor is just to make the calculations convenient.
- Cost function: loss function averaged over all training examples

$$\begin{aligned}\mathcal{J}(\mathbf{w}, b) &= \frac{1}{2N}\sum_{i=1}^{N}\left(y^{(i} - t^{(i)}\right)^2 \\ &= \frac{1}{2N}\sum_{i=1}^{N}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - t^{(i)}\right)^2\end{aligned}$$

- Terminology varies. Some call "cost" empirical or average loss.

Vectorization

- Notation-wise, $\frac{1}{2N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)})^2$ gets messy if we expand $y^{(i)}$:

$$\frac{1}{2N}\sum_{i=1}^{N}\left(\sum_{j=1}^{D}\left(w_j x_j^{(i)} + b\right) - t^{(i)}\right)^2$$

- The code equivalent is to compute the prediction using a for loop:

```
y = b
for j in range(M):
    y += w[j] * x[j]
```

- Excessive super/sub scripts are hard to work with, and Python loops are slow, so we vectorize algorithms by expressing them in terms of vectors and matrices.

$$\mathbf{w} = (w_1, \ldots, w_D)^\top, \mathbf{x} = (x_1, \ldots, x_D)^\top \quad y = \mathbf{w}^\top \mathbf{x} + b$$

- This is simpler and executes much faster:

```
y = np.dot(w, x) + b
```

Vectorization

Why vectorize?

- The equations, and the code, will be simpler and more readable. Gets rid of dummy variables/indices!
- Vectorized code is much faster
  - Cut down on Python interpreter overhead
  - Use highly optimized linear algebra libraries (hardware support)
  - Matrix multiplication very fast on GPU (Graphics Processing Unit)

Vectorization

- We can organize all the training examples into a design matrix $\mathbf{X}$ with one row per training example, and all the targets into the target vector $\mathbf{t}$.

one feature across all training examples

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}^{(1)\top} \\ \mathbf{x}^{(2)\top} \\ \mathbf{x}^{(3)\top} \end{pmatrix} = \begin{pmatrix} 8 & 0 & 3 & 0 \\ 6 & -1 & 5 & 3 \\ 2 & 5 & -2 & 8 \end{pmatrix}$$

one training example (vector)

- Computing the predictions for the whole dataset:

$$\mathbf{X}\mathbf{w} + b\mathbf{1} = \begin{pmatrix} \mathbf{w}^\top \mathbf{x}^{(1)} + b \\ \vdots \\ \mathbf{w}^\top \mathbf{x}^{(N)} + b \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{pmatrix} = \mathbf{y}$$

Vectorization

- Computing the squared error cost across the whole dataset:

$$\mathbf{y} = \mathbf{X}\mathbf{w} + b\mathbf{1}$$

$$\mathcal{J} = \frac{1}{2N} \parallel \mathbf{y} - \mathbf{t} \parallel^2$$

- We can also add a column of 1's to design matrix, combine the bias and the weights, and conveniently write

$$\mathbf{X} = \begin{bmatrix} 1 & [\mathbf{x}^{(1)}]^\top \\ \vdots & \vdots \\ 1 & [\mathbf{x}^{(N)}]^\top \end{bmatrix} \in \mathbb{R}^{N \times (D+1)} \text{ and } \mathbf{w} = \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_N \end{bmatrix} \in \mathbb{R}^{D+1}$$

Then, our prediction reduces to $\mathbf{y} = \mathbf{X}\mathbf{w}$.

Solving the Minimization Problem

We defined a cost function. This is what we'd like to minimize.

Two commonly applied mathematical approaches:

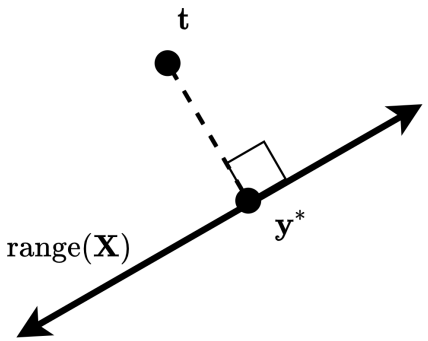
- Algebraic, e.g., using inequalities:
  - to show z^* minimizes $f(z)$, show that $\forall z, f(z) \geq f(z^*)$
  - to show that $a = b$, show that $a \geq b$ and $b \geq a$
- Calculus: minimum of a smooth function (if it exists) occurs at a critical point, i.e. point where the derivative is zero (or equivalently the gradient).
  - multivariate generalization: set the partial derivatives to zero (or equivalently the gradient).

Solutions may be direct or iterative

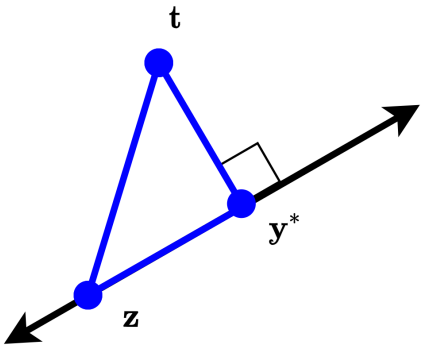
- Sometimes we can directly find provably optimal parameters (e.g. set the gradient to zero and solve in closed form). We call this a direct solution.
- We may also use optimization techniques that iteratively get us closer to the solution. We will get back to this soon.

Direct Solution I: Linear Algebra

- We seek w to minimize $\| \mathbf{Xw} - \mathbf{t} \|^2$
- $range(\mathbf{X}) = \{\mathbf{Xw} | \mathbf{w} \in \mathbb{R}^D\}$ is a D-dimensional subspace of $\mathbb{R}^N$.
- Recall that the closest point $\mathbf{y}^* = \mathbf{Xw}^*$ in subspace $range(\mathbf{X})$) of $\mathbb{R}^N$ to arbitrary point $t \in \mathbb{R}^N$ is found by orthogonal projection.



- We have $(\mathbf{y}^* - \mathbf{t}) \perp \mathbf{Xw}, \forall \mathbf{w} \in \mathbb{R}^D$



- Why is $\mathbf{y}^*$ the closest point to $\mathbf{t}$?
  - Consider any $\mathbf{z} = \mathbf{Xw}$
  - By Pythagorean theorem and the trivial inequality ($x^2 \geq 0$):

$$\begin{aligned} \| \mathbf{z} - \mathbf{t} \|^2 &= \| \mathbf{y}^* - \mathbf{t} \|^2 + \| \mathbf{y}^* - \mathbf{z} \|^2 \\ &\geq \| \mathbf{y}^* - \mathbf{t} \|^2 \end{aligned}$$

Direct Solution I: Linear Algebra

- From the previous slide, we have$(\mathbf{y}^* - \mathbf{t}) \perp \mathbf{X}\mathbf{w}, \forall \mathbf{w} \in \mathbb{R}^D$
- Equivalently, the columns of the design matrix $\mathbf{X}$ are all orthogonal to $(\mathbf{y}^* - \mathbf{t})$, and we have that:

$$\begin{aligned}
\mathbf{X}^\top(\mathbf{y}^* - \mathbf{t}) &= 0 \\
\mathbf{X}^\top\mathbf{X}\mathbf{w}^* - \mathbf{X}^\top\mathbf{t} &= 0 \\
\mathbf{X}^\top\mathbf{X}\mathbf{w}^* &= \mathbf{X}^\top\mathbf{t} \\
\mathbf{w}^* &= (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{t}
\end{aligned}$$

- While this solution is clean and the derivation easy to remember, like many algebraic solutions, it is somewhat ad hoc.
- On the hand, the tools of calculus are broadly applicable to differentiable loss functions...

Direct Solution II: Calculus

- Partial derivative: derivative of a multivariate function with respect to one of its arguments.

$$\frac{\partial}{\partial x_1} f(x_1, x_2) = \lim_{h \to 0} \frac{f(x_1 + h, x_2) - f(x_1, x_2)}{h}$$

- To compute, take the single variable derivative, pretending the other arguments are constant.
- Example: partial derivatives of the prediction y

$$\frac{\partial y}{\partial w_j} = \frac{\partial}{\partial w_j} \left[\sum_{j'} w_{j'} x_{j'} + b \right] = x_j \qquad \frac{\partial y}{\partial b} = \frac{\partial}{\partial b} \left[\sum_{j'} w_{j'} x_{j'} + b \right] = 1$$

Direct Solution II: Calculus

- For loss derivatives, apply the chain rule:

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{d\mathcal{L}}{dy}\frac{\partial y}{\partial w_j} \qquad \frac{\partial \mathcal{L}}{\partial b} = \frac{d\mathcal{L}}{dy}\frac{\partial y}{\partial b}$$

$$= \frac{d}{dy}\left[\frac{1}{2}(y-t)^2\right] \cdot x_j \qquad = y - t$$

$$= (y-t)x_j$$

- For cost derivatives, use linearity and average over data points:

$$\frac{\partial \mathcal{J}}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)})x_j^{(i)} \qquad \frac{\partial \mathcal{J}}{\partial b} = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)})$$

- Minimum must occur at a point where partial derivatives are zero.

$$\frac{\partial \mathcal{J}}{\partial w_j} = 0 \quad (\forall j) \qquad \frac{\partial \mathcal{J}}{\partial b} = 0$$

Direct Solution II: Calculus

- The derivation on the previous slide gives a system of linear equations, which we can solve efficiently.
- As is often the case for models and code, however, the solution is easier to characterize if we vectorize our calculus.
- We call the vector of partial derivatives the gradient
- Thus, the gradient of $f : \mathbb{R}^D \to \mathbb{R}$, denoted $\nabla f(\mathbf{w})$, is:

$$\left(\frac{\partial}{\partial w_1} f(\mathbf{w}), \ldots, \frac{\partial}{\partial w_D} f(\mathbf{w}) \right)$$

  The gradient points in the direction of the greatest rate of increase.

Direct Solution II: Calculus

- We seek $\mathbf{w}$ to minimize $\mathcal{J}(\mathbf{w}) = \frac{1}{2} \parallel \mathbf{X}\mathbf{w} - \mathbf{t} \parallel^2$
- Taking the gradient with respect to $\mathbf{w}$ we get:

$$\nabla_{\mathbf{w}} \mathcal{J}(\mathbf{w}) = \mathbf{X}^\top \mathbf{X}\mathbf{w} - \mathbf{X}^\top \mathbf{t} = \mathbf{0}$$

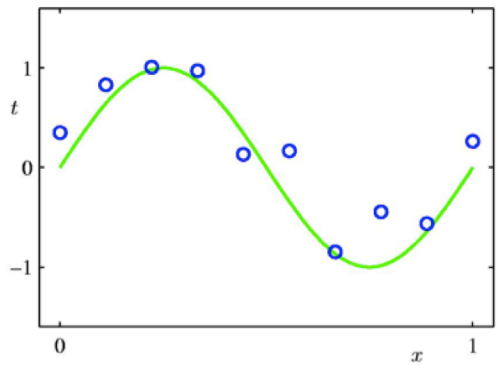
- We get the same optimal weights as before:

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t}$$

- Linear regression is one of only a handful of models in this course that permit direct solution.

Feature Mapping (Basis Expansion)

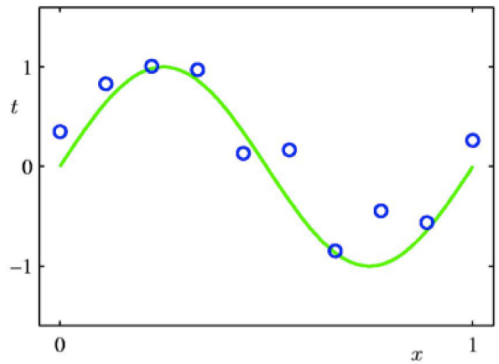
The relation between the input and output may not be linear.



- We can still use linear regression by mapping the input features to another space using feature mapping (or basis expansion). $\psi(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}^d$ and treat the mapped feature (in $\mathbb{R}^d$) as the input of a linear regression procedure.
- Let us see how it works when $x \in \mathbb{R}$ and we use a polynomial feature mapping.

Polynomial Feature Mapping

If the relationship doesn't look linear, we can fit a polynomial.



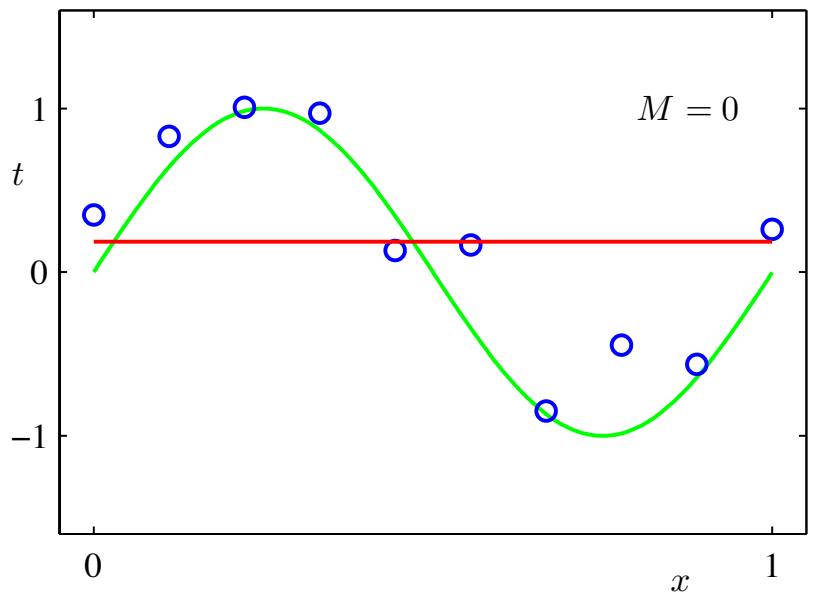
Fit the data using a degree-M polynomial function of the form:

$$y = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{i=0}^{M} w_i x^i$$

- Here the feature mapping is $\boldsymbol{\psi}(x) = [1, x, x^2, \ldots, x^M]^\top$.
- We can still use linear regression to find $\mathbf{w}$ since $y = \boldsymbol{\psi}(x)^\top \mathbf{w}$ is linear in $w_0, w_1, \ldots$
- In general ψ can be any function, i.e., $\boldsymbol{\psi}(x) = [1, sin(2\pi x), cos(2\pi x), sin(4\pi x), \ldots]^\top$.

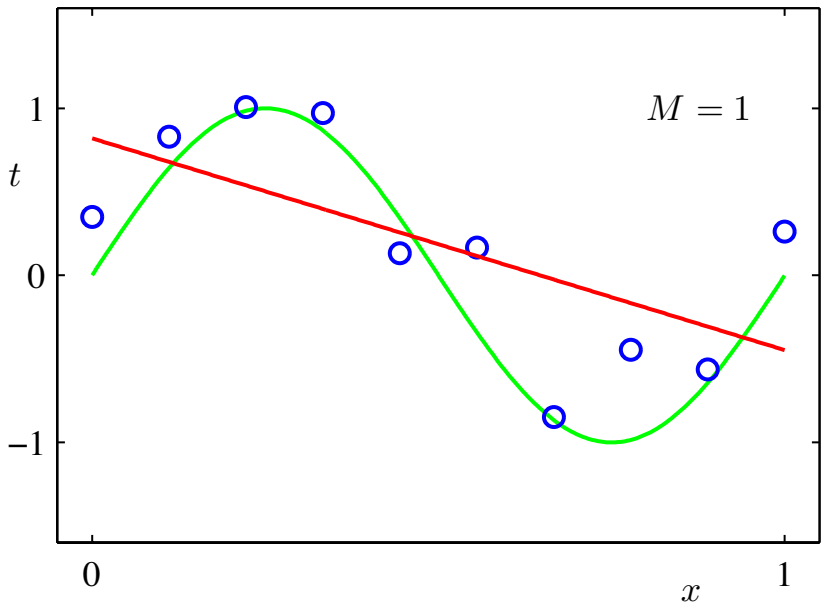
Polynomial Feature Mapping with $M = 0$

$$y = w_0$$



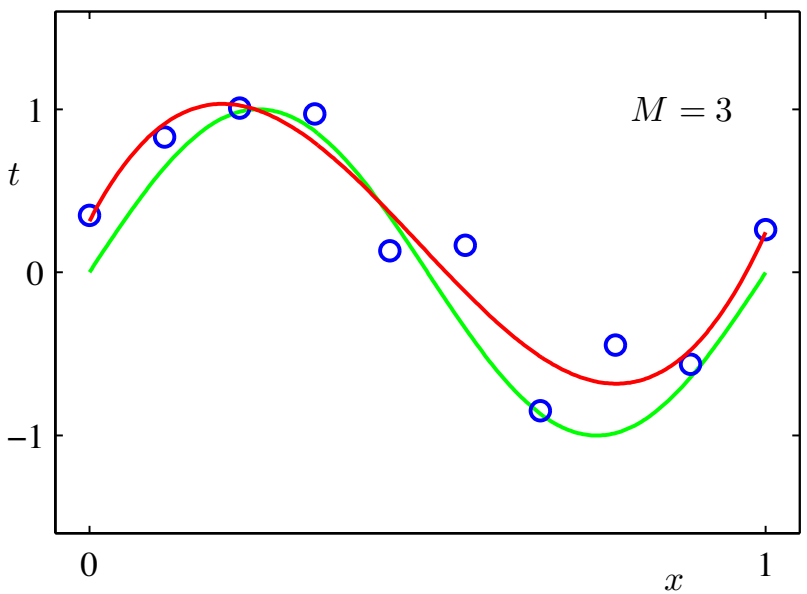
Polynomial Feature Mapping with $M = 1$

$$y = w_0 + w_1 x$$



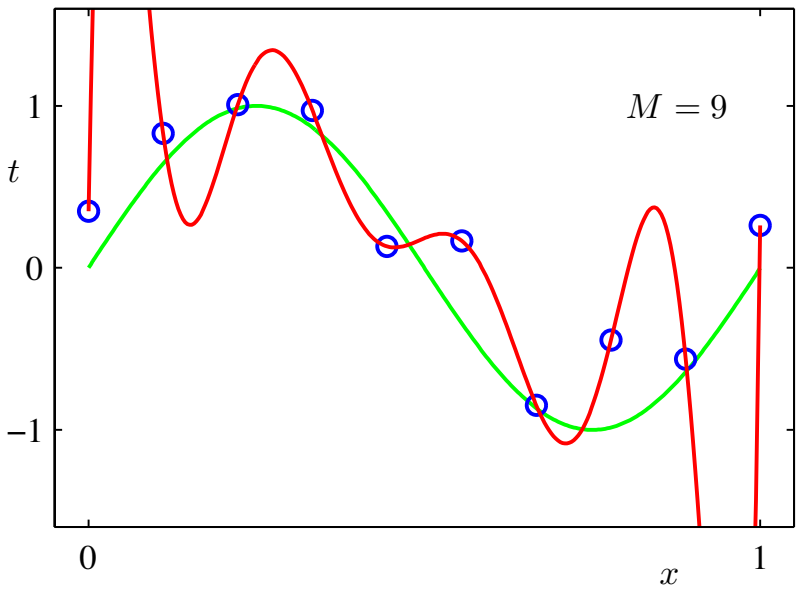
Polynomial Feature Mapping with $M = 3$

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$



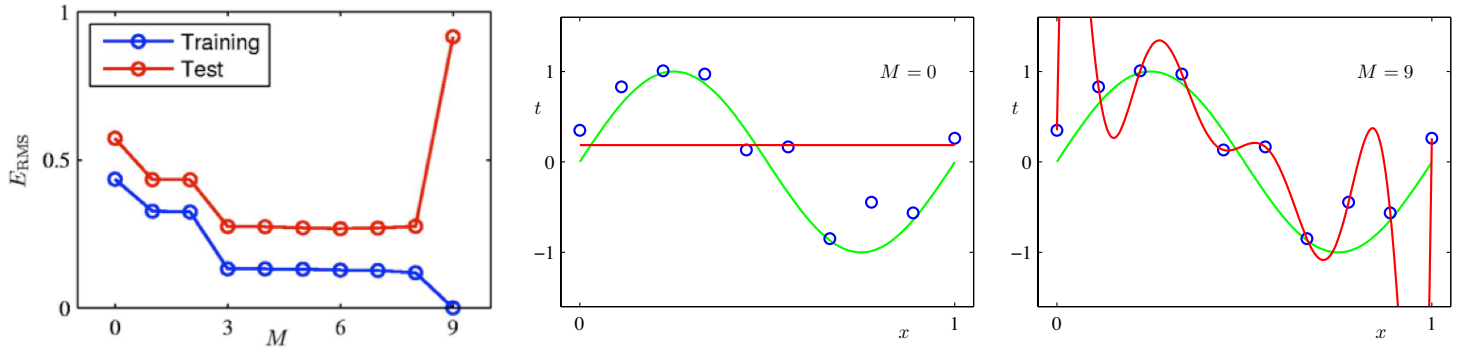
Polynomial Feature Mapping with $M = 9$

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots + w_9 x^9$$



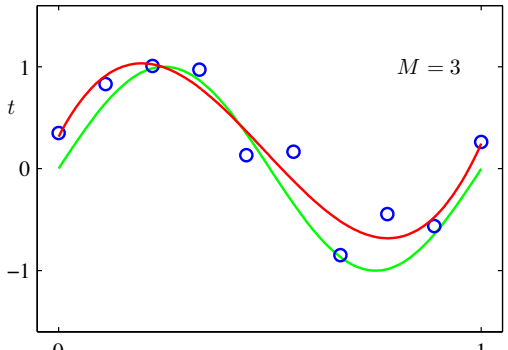
Model Complexity and Generalization

- Underfitting ($M = 0$): model is too simple - does not fit the data.
- Overfitting ($M = 9$): model is too complex - fits perfectly.



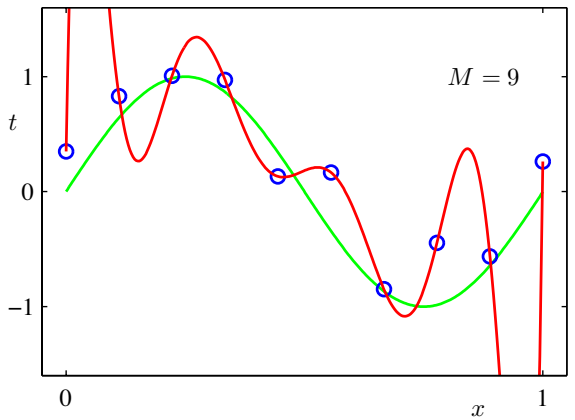
- Good model ($M = 3$): Achieves small test error (generalizes well).

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$



Model Complexity and Generalization

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



- As M increases, the magnitude of coefficients gets larger.
- For $M = 9$, the coefficients have become finely tuned to the data.
- Between data points, the function exhibits large oscillations.

Regularization

- The degree M of the polynomial controls the model's complexity.
- The value of M is a hyperparameter for polynomial expansion, just like k in KNN. We can tune it using a validation set.
- Restricting the number of parameters/basis functions (M) is a crude approach to controlling the model complexity.
- Another approach: keep the model large, but regularize it
  - Regularizer: a function that quantifies how much we prefer one hypothesis vs. another

L^2 Regularization

- We can encourage the weights to be small by choosing as our regularizer the L^2 penalty.

$$\mathcal{R}(\mathbf{w}) = \frac{1}{2} \parallel \mathbf{w} \parallel^2 = \frac{1}{2} \sum_j w_j^2$$

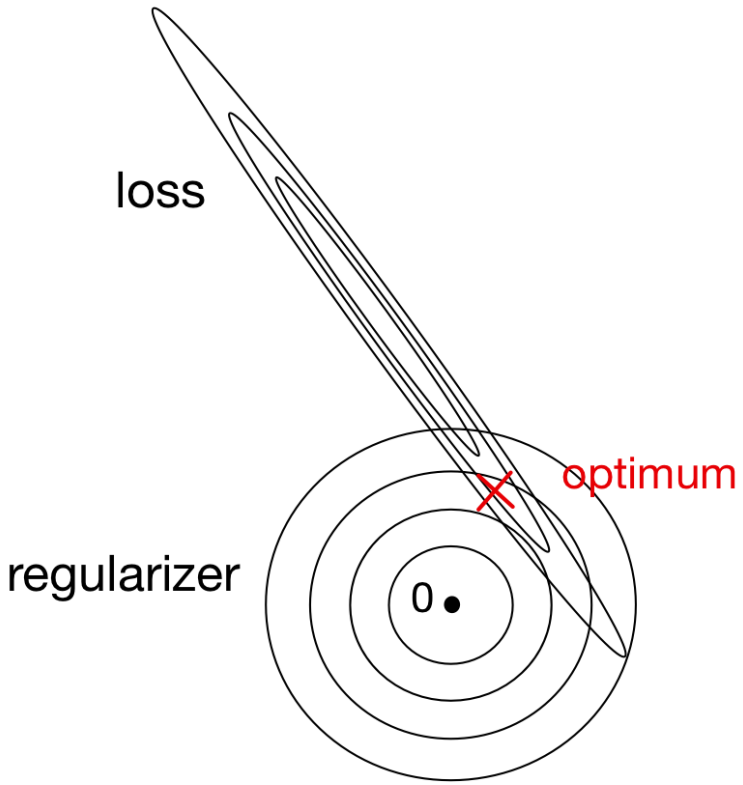
- The regularized cost function makes a tradeoff between fit to the data and the norm of the weights.

$$\mathcal{J}_{\text{reg}}(\mathbf{w}) = \mathcal{J}(\mathbf{w}) + \lambda \mathcal{R}(\mathbf{w}) = \mathcal{J}(\mathbf{w}) + \frac{\lambda}{2} \sum_j w_j^2$$

- If you fit training data poorly, $\mathcal{J}$ is large. If your optimal weights have high values, $\mathcal{R}$ is large.
- Large λ penalizes weight values more.
- Like M, λ is a hyperparameter we can tune with a validation set.

L^2 Regularization

- The geometric picture



L^2 Regularizated Least Squares

For the least square problem, we have $\mathcal{J}(\mathbf{w}) = \frac{1}{2N} \parallel \mathbf{X}\mathbf{w} - \mathbf{t} \parallel^2$.

- When $\lambda > 0$ ((with regularization), regularized cost gives)

$$\begin{aligned}\mathbf{w}_\lambda = \arg\min_{\mathbf{w}} \mathcal{J}_{\text{reg}}(\mathbf{w}) &= \arg\min_{\mathbf{w}} \frac{1}{2N} \parallel \mathbf{X}\mathbf{w} - \mathbf{t} \parallel^2 + \frac{\lambda}{2} \parallel \mathbf{w} \parallel^2 \\ &= (\mathbf{X}^\top \mathbf{X} + \lambda N \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{t}\end{aligned}$$

- The case $\lambda = 0$ (no regularization) reduces to least squares solution!
- Note that it is also common to formulate this problem as $\arg\min_{\mathbf{w}} \frac{1}{2} \parallel \mathbf{X}\mathbf{w} - \mathbf{t} \parallel^2 + \frac{\lambda}{2} \parallel \mathbf{w} \parallel^2$ in which case the solution is $\mathbf{w}_\lambda = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{t}$

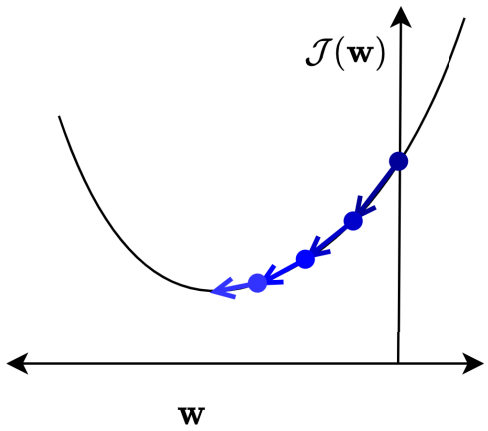
Conclusions...so far

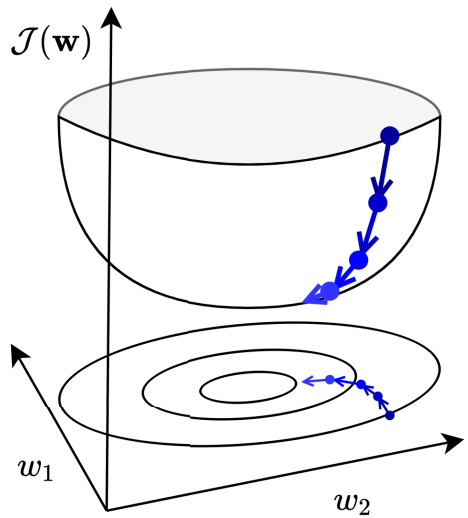
Linear regression exemplifies recurring themes of this course:

- choose a model and a loss function
- formulate an optimization problem
- solve the minimization problem using one of two strategies
  - direct solution (set derivatives to zero)
  - gradient descent (next topic)
- vectorize the algorithm, i.e. represent in terms of linear algebra
- make a linear model more powerful using features
- improve the generalization by adding a regularizer

Gradient Descent

- Now let's see a second way to minimize the cost function which is more broadly applicable: gradient descent.
- Many times, we do not have a direct solution: Taking derivatives of $\mathcal{J}$ w.r.t $\mathbf{w}$ and setting them to 0 doesn't have an explicit solution.
- Gradient descent is an iterative algorithm, which means we apply an update repeatedly until some criterion is met.
- We initialize the weights to something reasonable (e.g. all zeros) and repeatedly adjust them in the direction of steepest descent.





Gradient Descent

- Observe
  - if $\partial \mathcal{J} / \partial w_j > 0$, then increasing w_j increases $\mathcal{J}$.
  - if $\partial \mathcal{J} / \partial w_j < 0$, then increasing w_j decreases $\mathcal{J}$.
- The following update always decreases the cost function for small enough α (unless $\partial \mathcal{J} / \partial w_j = 0$):

$$w_j \leftarrow w_j - \alpha \frac{\partial \mathcal{J}}{\partial w_j}$$

- $\alpha > 0$ is a learning rate (or step size). The larger it is, the faster $\mathbf{w}$ changes.
  - We'll see later how to tune the learning rate, but values are typically small, e.g. 0.01 or 0.0001.
  - If cost is the sum of N individual losses rather than their average, smaller learning rate will be needed ($\alpha' = \alpha / N$).

Gradient Descent

- This gets its name from the gradient:

$$\nabla_{\mathbf{w}} \mathcal{J} = \frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial \mathcal{J}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{J}}{\partial w_D} \end{pmatrix}$$

  - This is the direction of fastest increase in $\mathcal{J}$
- Update rule in vector form:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}}$$

  And for linear regression we have:

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) \mathbf{x}^{(i)}$$

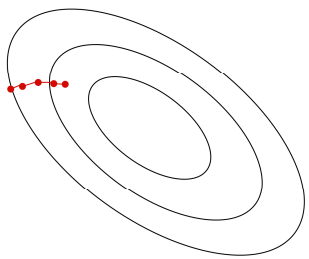
- So gradient descent updates $\mathbf{w}$ in the direction of fastest decrease.
- Observe that once it converges, we get a critical point, i.e. $\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \mathbf{0}$.

Gradient Descent for Linear Regression

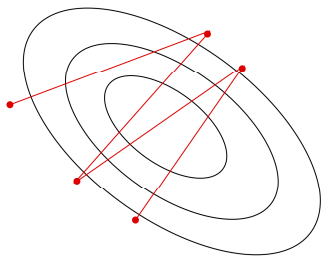
- The squared error loss of linear regression is a convex function.
- Even for linear regression, where there is a direct solution, we sometimes need to use GD.
- Why gradient descent, if we can find the optimum directly?
  - GD can be applied to a much broader set of models
  - GD can be easier to implement than direct solutions
  - For regression in high-dimensional space, GD is more efficient than direct solution
    - Linear regression solution: $(\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{t}$
    - Matrix inversion is an $\mathcal{O}(D^3)$ algorithm
    - Each GD update costs $\mathcal{O}(ND)$
    - Even less with stochastic GD (SGD, in a few slides)
    - Huge difference if $D \gg 1$

Learning Rate (Step Size)

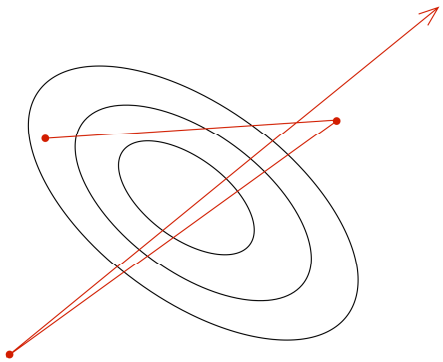
- In gradient descent, the learning rate α is a hyperparameter we need to tune. Here are some things that can go wrong:



α too small:
slow progress



α too large:
oscillations

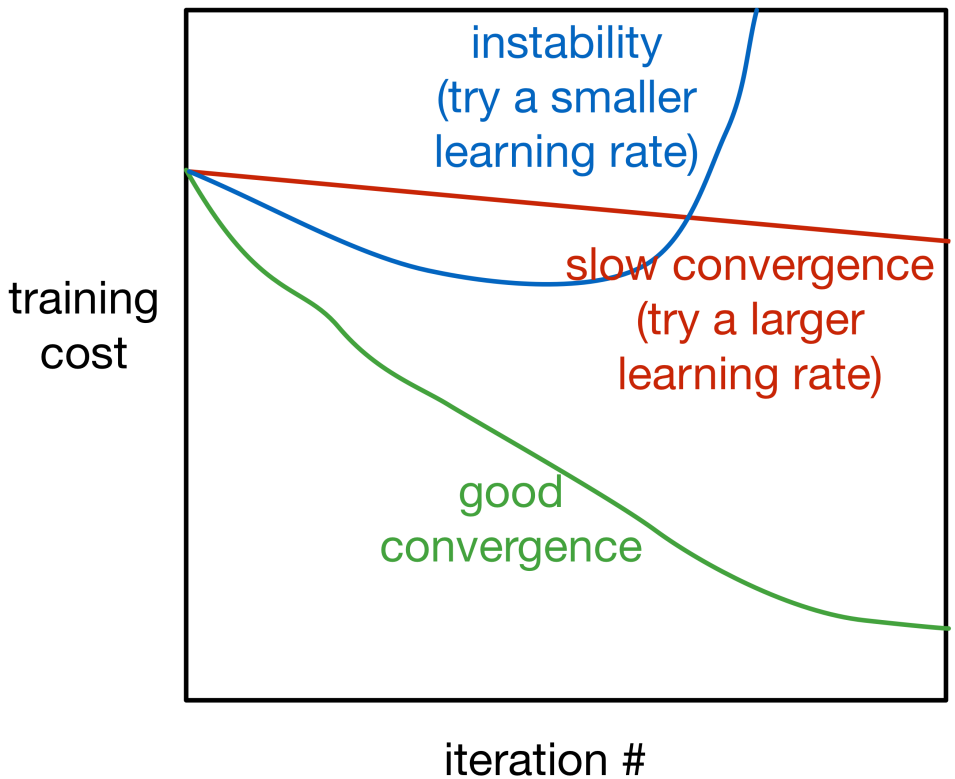


α much too large:
instability

- Good values are typically between 0.001 and 0.1. You should do a grid search if you want good performance (i.e. try 0.1, 0.03, 0.01,. . .).

Training Curves

- To diagnose optimization problems, it's useful to look at training curves: plot the training cost as a function of iteration.



- Warning: in general, it's very hard to tell from the training curves whether an optimizer has converged. They can reveal major problems, but they can't guarantee convergence.

Stochastic Gradient Descent

- So far, the cost function $\mathcal{J}$ has been the average loss over the training examples:

$$\mathcal{J}(\boldsymbol{\theta})) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}^{(i)} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(y(\mathbf{x}^{(i)}, \boldsymbol{\theta}), t^{(i)})$$

  ($\boldsymbol{\theta}$ denotes the parameters; e.g., in linear regression, $\boldsymbol{\theta} = (\mathbf{w}, b)$)

- By linearity,

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{\theta}} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}}$$

- Computing the gradient requires summing over all of the training examples. This is known as batch training.
- Batch training is impractical if you have a large dataset $N \gg 1$ (e.g. millions of training examples)!

Stochastic Gradient Descent

- Stochastic gradient descent (SGD): update the parameters based on the gradient for a single training example,
  1. Choose i uniformly at random,
  2. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}}$
- Cost of each SGD update is independent of N!
- SGD can make significant progress before even seeing all the data!
- Mathematical justification: if you sample a training example uniformly at random, the stochastic gradient is an unbiased estimate of the batch gradient:

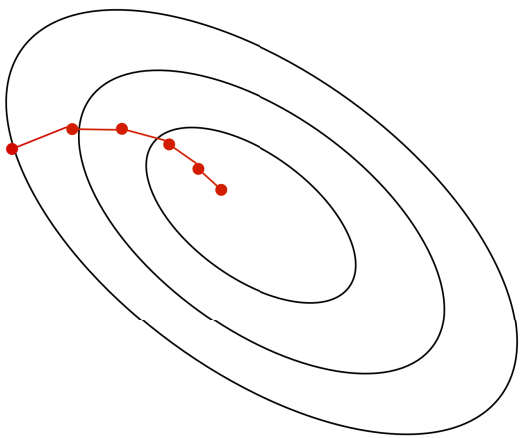
$$E\left[\frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}}\right] = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial \mathcal{L}^{(i)}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{\theta}}$$

Stochastic Gradient Descent

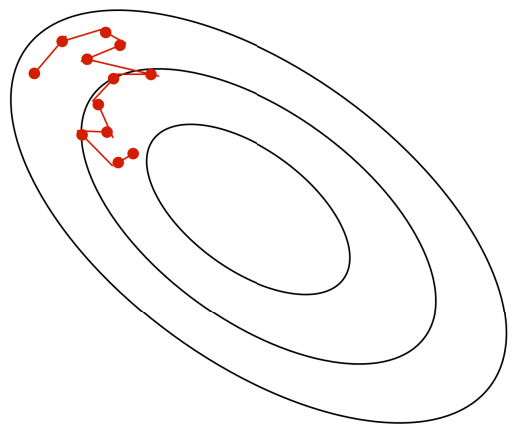
- Problems with using single training example to estimate gradient:
  - Variance in the estimate may be high
  - We can't exploit efficient vectorized operations
- Compromise approach:
  - compute the gradients on a randomly chosen medium-sized set of training examples $\mathcal{M} \subset \{1, \ldots, N\}$ called a mini-batch.
- Stochastic gradients computed on larger mini-batches have smaller variance.
- The mini-batch size $|\mathcal{M}|$ is a hyperparameter that needs to be set.
  - Too large: requires more compute; e.g., it takes more memory to store the activations, and longer to compute each gradient update
  - Too small: can't exploit vectorization, has high variance
  - A reasonable value might be $|\mathcal{M}| = 100$.

Stochastic Gradient Descent

- Batch gradient descent moves directly downhill (locally speaking).
- SGD takes steps in a noisy direction, but moves downhill on average.



batch gradient descent

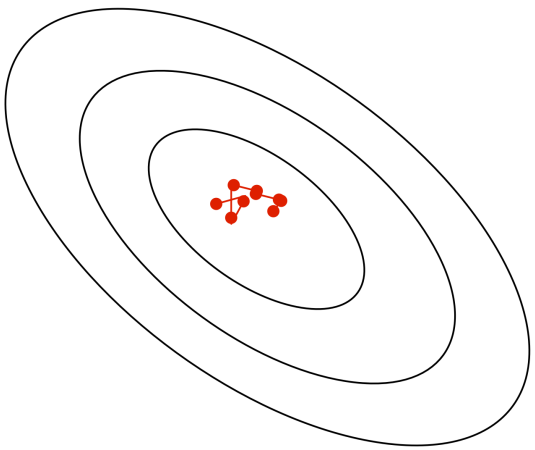


stochastic gradient descent

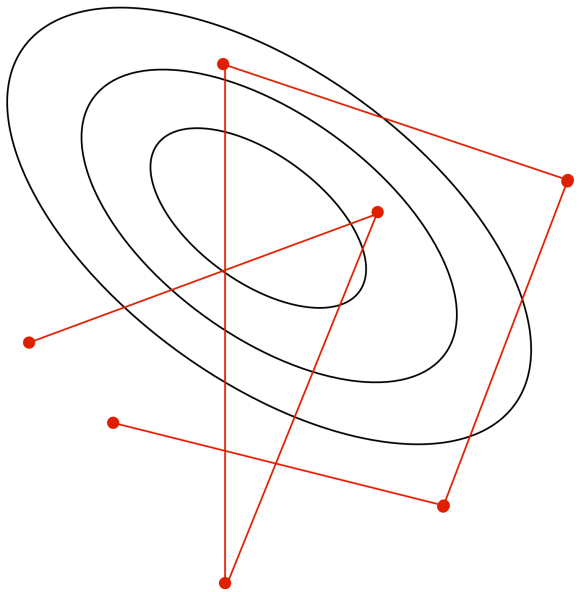
Stochastic Gradient Descent Learning Rate

- In stochastic training, the learning rate also influences the fluctuations due to the stochasticity of the gradients.

small learning rate

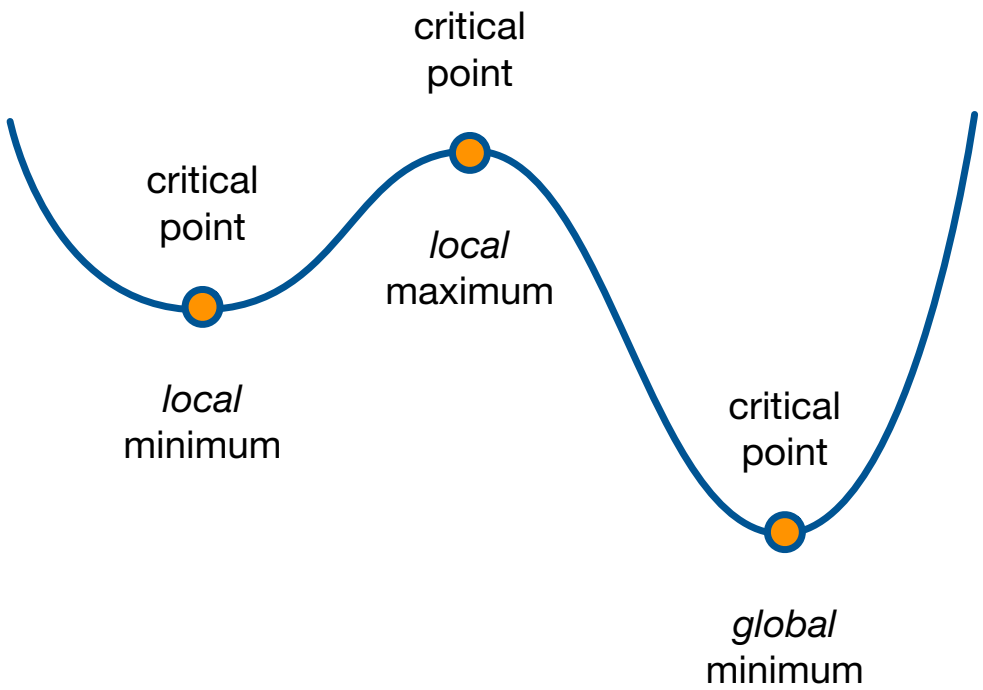


large learning rate



- Typical strategy:
  - Use a large learning rate early in training so you can get close to the optimum
  - Gradually decay the learning rate to reduce the fluctuations

When are critical points optimal?



- Gradient descent finds a critical point, but it may be a local optima.
- Convexity is a property that guarantees that all critical points are global minima.

Conclusions

- Linear regression exemplifies a modular approach that will be used throughout this course:
  - choose a model describing the relationships between variables of interest (**linear**)
  - define a loss function quantifying how bad the fit to the data is (**s**quared error)
  - fit/optimize the model (**gradient descent, stochastic gradient descent, convexity**)
- By mixing and matching these modular components, we can obtain new ML methods.
- Next apply this framework to classification

Logistic Regression

...based on Toronto University ML class

Overview

- Classification: predicting a discrete-valued target
  - Binary classification: predicting a binary-valued target
  - Multiclass classification: predicting a discrete(> 2)-valued target
- Examples of binary classification
  - predict whether a patient has a disease, given the presence or absence of various symptoms
  - classify e-mails as spam or non-spam
  - predict whether a financial transaction is fraudulent

Binary linear classification

- **classification**: given a D-dimensional input $\mathbf{x} \in \mathbb{R}^D$ predict a discrete-valued target
- **binary**: predict a binary target $t \in \{0, 1\}$
  - Training examples with $t = 1$ are called positive examples, and training examples with $t = 0$ are called negative examples.
  - $t \in \{0, 1\}$ or $t \in \{-1, +1\}$ is for computational convenience.
- **linear**: model prediction y is a linear function of $\mathbf{x}$, followed by a threshold r:

$$z = \mathbf{w}\top\mathbf{x} + b$$

$$y = \begin{cases} 1 & \text{if } z \geq r \\ 0 & \text{if } z < r \end{cases}$$

Some Simplifications

Eliminating the threshold

- We can assume without loss of generality that the threshold $r = 0$:

$$\mathbf{w}^\top \mathbf{x} + b \geq r \rightarrow \mathbf{w}^\top \mathbf{x} + \underbrace{b - r}_{w_0} \geq 0$$

Eliminating the bias

- Add a dummy feature x_0 which always takes the value 1. The weight $w_0 = b$ is equivalent to a bias (same as linear regression)

Simplified model

- Receive input $\mathbf{x} \in \mathbb{R}^{D+1}$ with $x_0 = 1$:

$$z = \mathbf{w}^\top \mathbf{x}$$

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Logistic Regression

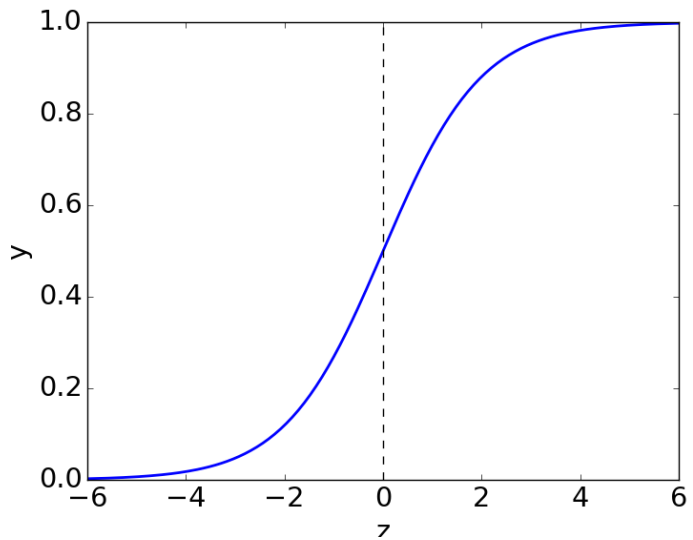
Take a probabilistic approach to learn a classifier

- $f(\mathbf{x}) = f_{\mathbf{w}}(\mathbf{x})$ should provide $P(y = 1 | \mathbf{x}; \mathbf{w})$
  - Want $0 \leq f_{\mathbf{w}}(\mathbf{x}) \leq 1$

Logistic regression model

$$f_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$$

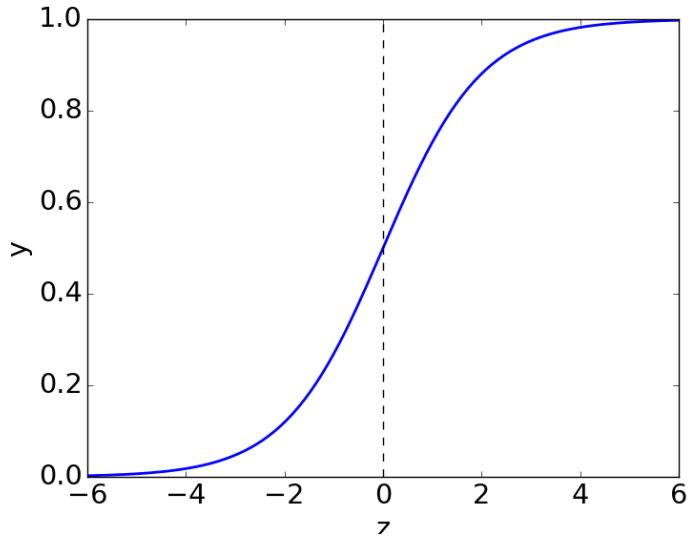
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Logistic Regression

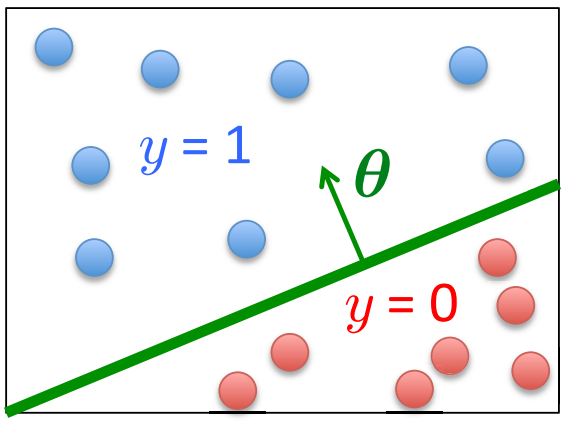
$$f_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$$

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



Assume a threshold, e.g., 0,5, and...

- Predict $y = 1$ if $f_{\mathbf{w}}(\mathbf{x}) > 0,5$
- Predict $y = 0$ if $f_{\mathbf{w}}(\mathbf{x}) \leq 0,5$



Logistic Regression Objective Function

- Given $\{(x^{(1)}, t^{(1)}), (x^{(2)}, t^{(2)}), \ldots, (x^{(N)}, t^{(N)})\}$ with $x^{(i)} \in \mathbb{R}^d$, $t^{(i)} \in \{0, 1\}$
- and the Logistic Regression Model: $f_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$, $\sigma(z) = \frac{1}{1+e^{-z}}$

Logistic Regression Objective Function

- Given $\{(x^{(1)}, t^{(1)}), (x^{(2)}, t^{(2)}), \ldots, (x^{(N)}, t^{(N)})\}$ with $x^{(i)} \in \mathbb{R}^d$, $t^{(i)} \in \{0, 1\}$
- and the Logistic Regression Model: $f_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$, $\sigma(z) = \frac{1}{1+e^{-z}}$
- We can't just use the squared loss as in linear regression:

$$\mathcal{J}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} (f_{\mathbf{w}}(\mathbf{x}^{(i)}) - t^{(i)})^2$$

Logistic Regression Objective Function

- Given $\{(x^{(1)}, t^{(1)}), (x^{(2)}, t^{(2)}), \ldots, (x^{(N)}, t^{(N)})\}$ with $x^{(i)} \in \mathbb{R}^d$, $t^{(i)} \in \{0, 1\}$
- and the Logistic Regression Model: $f_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$, $\sigma(z) = \frac{1}{1+e^{-z}}$
- We can't just use the squared loss as in linear regression:

$$\mathcal{J}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} (f_{\mathbf{w}}(\mathbf{x}^{(i)}) - t^{(i)})^2$$

  - using the Logistic Regression Model

$$f_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}}$$

    results in a non-convex optimization problem

Deriving the Cost Function via Maximum Likelihood Estimation

- Define Likelihood of observed data $\mathbf{t} = (t^{(1)}, \ldots, t^{(N)})$, given $\mathbf{x} = (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)})$ and parameter $\mathbf{w}$ as

$$l(\mathbf{w}) = P(\mathbf{t}|\mathbf{x}\,;\mathbf{w})$$

  that is the probability of observing $\mathbf{t}$ given $\mathbf{x}$
  - which is function of the parameters $\mathbf{w}$
- Given the independence between $t^{(i)}$ and $\mathbf{x}^{(j)}$, $i \neq j$, we can rewrite as:

$$l(\mathbf{w}) = P(\mathbf{t}|\mathbf{x}\,;\mathbf{w}) = \prod_{i=1}^{N} P(t^{(i)}|\mathbf{x}^{(i)}\,;\mathbf{w})$$

Deriving the Cost Function via Maximum Likelihood Estimation

Maximum Likelihood Estimation

- We look for the $\mathbf{w}$ which maximizes the likelihood $l(\mathbf{w})$

$$\mathbf{w}_{MLE} = \arg\max_{\mathbf{w}} l(\mathbf{w}) = \arg\max_{\mathbf{w}} \prod_{i=1}^{N} P(t^{(i)}|\mathbf{x}^{(i)}; \mathbf{w})$$

- We can take the log without changing the solution
  - log function is monothonically increasing

$$\mathbf{w}_{MLE} = \arg\max_{\mathbf{w}} \log \prod_{i=1}^{N} P(t^{(i)}|\mathbf{x}^{(i)}; \mathbf{w})$$

$$= \arg\max_{\mathbf{w}} \sum_{i=1}^{N} \log P(t^{(i)}|\mathbf{x}^{(i)}; \mathbf{w})$$

Deriving the Cost Function via Maximum Likelihood Estimation

- ...which we can rewrite as follows:

$$\mathbf{w}_{MLE} = \arg\max_{\mathbf{w}} \sum_{i=1}^{N} \log P(t^{(i)}|\mathbf{x}^{(i)}; \mathbf{w})$$

$$= arg\max_{\mathbf{w}} \sum_{i=1}^{N} \left[t^{(i)} \log P(t^{(i)} = 1|\mathbf{x}^{(i)}; \mathbf{w}) + \left(1 - t^{(i)}\right) \log \left(1 - P(t^{(i)} = 1|\mathbf{x}^{(i)}; \mathbf{w})\right) \right]$$

Deriving the Cost Function via Maximum Likelihood Estimation

- ...which we can rewrite as follows:

$$\mathbf{w}_{MLE} = \arg\max_{\mathbf{w}} \sum_{i=1}^{N} \log P(t^{(i)}|\mathbf{x}^{(i)}\,;\mathbf{w})$$

$$= arg\max_{\mathbf{w}} \sum_{i=1}^{N} \left[t^{(i)} \log P(t^{(i)}=1|\mathbf{x}^{(i)}\,;\mathbf{w}) + \left(1-t^{(i)}\right) \log\left(1-P(t^{(i)}=1|\mathbf{x}^{(i)}\,;\mathbf{w})\right)\right]$$

- **Logistic Regression**
  - take the (negative of the) log-likelihood as cost function

$$\mathcal{J}(\mathbf{w}) = -\sum_{i=1}^{N} \left[t^{(i)} \log P(t^{(i)}=1|\mathbf{x}^{(i)}\,;\mathbf{w}) + \left(1-t^{(i)}\right) \log\left(1-P(t^{(i)}=1|\mathbf{x}^{(i)}\,;\mathbf{w})\right)\right]$$

$$= -\sum_{i=1}^{N} \left[t^{(i)} \log f_{\mathbf{w}}(\mathbf{x}) + \left(1-t^{(i)}\right) \log\left(1-f_{\mathbf{w}}(\mathbf{x})\right)\right]$$

Deriving the Cost Function via Maximum Likelihood Estimation

- **Logistic Regression**

$$
\begin{aligned}
\mathcal{J}(\mathbf{w}) &= -\sum_{i=1}^{N} \left[t^{(i)} \log P(t^{(i)} = 1 | \mathbf{x}^{(i)}; \mathbf{w}) + \left(1 - t^{(i)}\right) \log \left(1 - P(t^{(i)} = 1 | \mathbf{x}^{(i)}; \mathbf{w})\right) \right] \\
&= -\sum_{i=1}^{N} \left[t^{(i)} \log f_{\mathbf{w}}(\mathbf{x}) + \left(1 - t^{(i)}\right) \log \left(1 - f_{\mathbf{w}}(\mathbf{x})\right) \right] \\
&= -\sum_{i=1}^{N} \left[t^{(i)} \log \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}} + \left(1 - t^{(i)}\right) \log \left(1 - \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}}\right) \right]
\end{aligned}
$$

  - ...which is convex :-)

Gradient Descent for Logistic Regression

- How do we minimize the cost $\mathcal{J}$ for logistic regression? No direct solution.
  - Taking derivatives of $\mathcal{J}$ w.r.t. $\mathbf{w}$ and setting them to 0 doesn't have an explicit solution.
- However, the logistic loss is a convex function in $\mathbf{w}$, so let's consider the gradient descent.
  - Recall: we initialize the weights to something reasonable and repeatedly adjust them in the direction of steepest descent.
  - A standard initialization is $\mathbf{w} = 0$.

Gradient of Logistic Loss

Back to logistic regression:

$$\mathcal{L}_{CE}(y, t) = -t \log(y) - (1 - t) \log(1 - y)$$

$$y = \frac{1}{1 + e^{-z}} \text{ and } z = \mathbf{w}^\top \mathbf{x}$$

Therefore

$$\frac{\partial \mathcal{L}_{CE}}{\partial w_j} = \frac{\partial \mathcal{L}_{CE}}{\partial y} \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_j} = \left(-\frac{t}{y} + \frac{1 - t}{1 - y} \right) \cdot y(1 - y) \cdot x_j$$

$$= (y - t) x_j$$

Gradient descent (coordinatewise) update to find the weights of logistic regression:

$$w_j \leftarrow w_j - \alpha \frac{\partial \mathcal{J}}{\partial w_j}$$

$$= w_j - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) x_j^{(i)}$$

Gradient Descent for Logistic Regression

Comparison of gradient descent updates:

- Linear regression

$$\mathbf{w} = \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) \mathbf{x}^{(i)}$$

- Logistic regression

$$\mathbf{w} = \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) \mathbf{x}^{(i)}$$

- Not a coincidence! These are both examples of generalized linear models. But we won't go in further detail.
- Notice $\frac{1}{N}$ in front of sums due to averaged losses. This is why you need smaller learning rate when cost is summed losses ($\alpha' = \alpha/N$).

Multiclass Classification and Softmax Regression

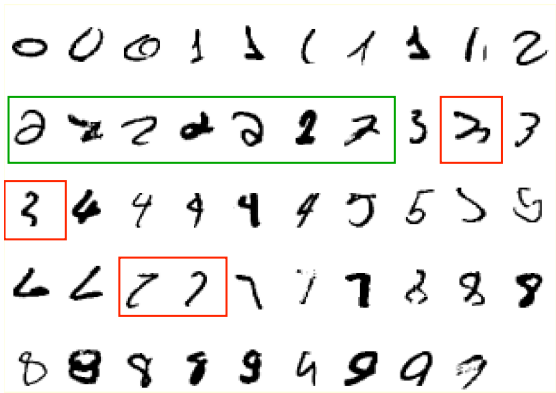
...based on Toronto University ML class

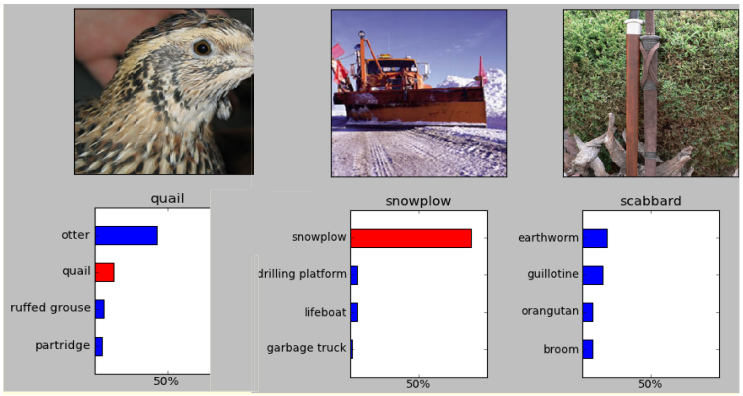
Overview

- Classification: predicting a discrete-valued target
  - Binary classification: predicting a binary-valued target
  - Multiclass classification: predicting a discrete(> 2)-valued target
- Examples of multi-class classification
  - predict the value of a handwritten digit
  - classify e-mails as spam, travel, work, personal

Multiclass Classification

- Classification tasks with more than two categories:





Multiclass Classification

- Targets form a discret set $\{1, \ldots, K\}$.
- It's often more convenient to represent them as one-hot vectors, or a one-of-K encoding:

$$\mathbf{t} = \underbrace{(0, \ldots, 0, 1, 0, \ldots, 0)}_{\text{entry } k \text{ is } 1} \in \mathbb{R}^K$$

Multiclass Linear Classification

- We can start with a linear function of the inputs.
- Now there are D input dimensions and K output dimensions, so we need $K \times D$ weights, which we arrange as a weight matrix $\mathbf{W}$.
- Also, we have a K-dimensional vector $\mathbf{b}$ of biases.
- A linear function of the inputs:

$$z_k = \sum_{j=1}^{D} w_{kj} x_j + b_k \text{ for } k = 1, 2, \ldots, K$$

- or in vectorized form

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{d}$$

- We can eliminate the bias $\mathbf{b}$ by taking $\mathbf{W} \in \mathbb{R}^{K \times (D+1)}$ and adding a dummy variable $x_0 = 1$, that is

$$\mathbf{z} = \mathbf{W}\mathbf{x}$$

Multiclass Linear Classification

- How can we turn this linear prediction into a one-hot prediction?
- We can interpret the magnitude of z_k as an measure of how much the model prefers k as its prediction.
- ...that is

$$y_i = \begin{cases} 1 & i = \arg\max z_k \\ 0 & \text{otherwise} \end{cases}$$

Softmax Regression

- We need to soften our predictions for the sake of optimization.
- We want soft predictions that are like probabilities, i.e., $0 \leq y_k \leq 1$ and $\sum_k y_k = 1$.
- A natural activation function to use is the softmax function, a multivariable generalization of the logistic function:

$$y_k = \text{softmax}(z_1, \ldots, z_K)_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}$$

  - Outputs can be interpreted as probabilities (positive and sum to 1)
  - If z_k is much larger than the others, then $\text{softmax}(\mathbf{z})_k \approx 1$ and it behaves like argmax.
- The inputs z_k are called the logits.

Softmax Regression

- If a model outputs a vector of class probabilities, we can use cross-entropy as the loss function:

$$\begin{aligned}\mathcal{L}_{CE}(\mathbf{y}, \mathbf{t}) &= -\sum_{k=1}^{k} t_k log y_k \\ &= -\mathbf{t}^\top (\log \mathbf{y})\end{aligned}$$

  where the log is applied elementwise.
- Just like with logistic regression, we typically combine the softmax and cross-entropy into a softmax-cross-entropy function.

Softmax Regression

- Softmax regression (with dummy $x_0 = 1$):

$$\mathbf{z} = \mathbf{W}\mathbf{x}$$
$$\mathbf{y} = \text{softmax}(\mathbf{z})$$
$$\mathcal{L}_{CE} = -\mathbf{t}^\top (\log \mathbf{y})$$

- Gradient descent updates can be derived for each row of $\mathbf{W}$:

$$\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{w}_k} = \frac{\partial \mathcal{L}_{CE}}{\partial z_k} \cdot \frac{\partial z_k}{\partial \mathbf{w}_k} = (y_k - t_k) \cdot \mathbf{x}$$
$$\mathbf{w}_k \leftarrow \mathbf{w}_k - \alpha \frac{1}{N} \sum_{i=1}^{N} (y_k^{(i)} - t_k^{(i)}) \mathbf{x}^{(i)}$$

- Similar to linear/logistic regression (no coincidence)

A Few Basic Concepts

- A hypothesis is a function $f : \mathcal{X} \to \mathcal{T}$ that we might use to make predictions (recall $\mathcal{X}$ is the input space and $\mathcal{T}$ is the target space).
- The hypothesis space $\mathcal{H}$ for a particular machine learning model or algorithm is set of hypotheses that it can represent.
  - E.g., in linear regression, $\mathcal{H}$ is the set of functions that are linear in the data features
  - The job of a machine learning algorithm is to find a good hypothesis $f \in \mathcal{H}$

A Few Basic Concepts

- If an algorithm's hypothesis space $\mathcal{H}$ can be defined using a finite set of parameters, denoted $\boldsymbol{\theta}$, we say the algorithm is parametric.
  - In linear regression, $\boldsymbol{\theta} = (\mathbf{w}, b)$
  - Other examples: logistic regression, neural networks, k-means and Gaussian mixture models
- If the members of $\mathcal{H}$ are defined in terms of the data, we say that the algorithm is non-parametric.
  - In k-nearest neighbors, the learned hypothesis is defined in terms of the training data
  - Other examples: decision trees, support vector machines, kernel density estimation
  - These models can sometimes be understood as having an infinite number of parameters