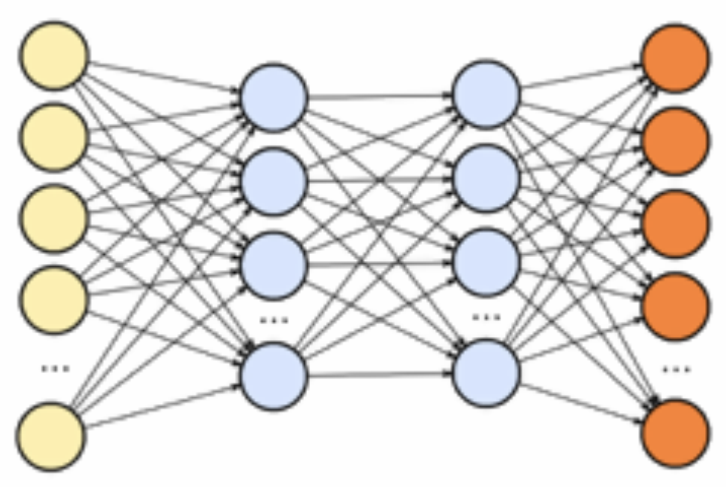
Deep Learning

Simon Du



CSEP590: Deep Learning

Instructor: Simon Du

Teaching Assistant: Siting, Ruizhe Shi

Course Website (contains all logistic information): https://courses.cs.washington.edu/courses/csep590a/26wi/

Questions: Ed Discussion

Announcements: Canvas

Homework: Canvas

CSEP590: Deep Learning

What this class is:

- **Fundamentals of DL:** Neural network architecture, approximation properties, optimization, generalization, generative models, representation learning
- **Preparation for further learning:** the field is fast-moving, you will be able to apply the fundamentals and teach yourself the latest

What this class is not:

- **An easy course:** mathematically easy
- **A survey course:** laundry list of algorithms

Prerequisites

- Working knowledge of:
  - Linear algebra
  - Vector calculus
  - Probability and statistics
  - Algorithms
  - Machine leanring (CSEP546)
- Mathematical maturity
- “Can I learn these topics concurrently?”

Lecture

- Time: Thursday 6:30 - 9:20PM
- CSE2 010 or Zoom (see website for the schedule)
- Slides + handwritten notes (e.g., derivations, proofs)
- Zoom link on Canvas
- Tentative schedule on course website

Homework (40%)

- 2 homework (20%+20%)
  - Each contains both theoretical questions and programming questions
  - Related to course materials
  - Collaboration okay but must write who you collaborated with. You must write, submit, and understand your answers and code.
  - Submit on Canvas
  - Must be **typed**
  - **Two** late days
  - Tentative timeline:
    - HW 1 due: 2/5
    - HW 2 due: 2/19

Course Project (60%)

- Group of 3 - 5.
- Topic: literature review (state-of-the-art) or an application or original research.
- Post on Ed Discussion to form teams.
- Some potential topics are in listed on Canvas. OK to do a project not listed.
- You can work on a project related to your research.
- Proposal (due: 1/33): **5%**
  - Format: NeurIPS Latex format, ~1 - 1.5 pages
- Presentations on (3/12 on Zoom): **20%**
- Final report (due: 3/19): **35%**
  - Format: NeurIPS Latex format, ~8 pages
- Submit on Canvas

Possible Topics

- Approximation properties
- Advanced optimization methods
- Optimization theory for deep learning
- Generalization theory for deep learning
- Deep reinforcement learning
- Implicit regularization
- Meta-learning
- Robustness
- Neural network compression
- Pre-training, fine-tuning, RLHF, RLVR
- Deep learning application
- …

Communication Chanels

- **Announcements**
  - Canvas
- **questions about class, homework help**
  - Ed Discussion
  - Office hours (Zoom):
    - Simon Du: Friday 10:00 - 11:00 AM
    - Siting Li: Thursday 11:00 - 12:00 PM
    - Ruizhe Shi: Friday 19:00 - 20:00 PM
  - **Regrade requests**
    - Canvas
  - **Personal concerns:**
    - Email to instructor or TAs

Topic: Machine Learning Review

- General setup
- Regression
- Train/Test Split
- Regularization
- Classification
- Basic optimization methods
- Fully-connected neural network

Topic: Optimization

- Review: Back-propagation
- Auto-differentiation
- Advanced optimizers: momentum (Nesterov acceleration), adaptive method (AdaGrad, Adam)
- Techniques for improving optimization: batch-norm, layer-norm, ..

Topic: Architecture

- Convolutional neural network
- Recurrent neural network
  - LSTM
- Attention-based neural network
  - Transformer
- General framework

Topic: Theoretical Foundation

- Why neural networks can express the (regression, classification, …) function you want?
- Construction of such desired neural networks
- Universal approximation theorem
- global convergence of gradient of over-parameterized neural networks
- Neural Tangent Kernel

Topic: Generalization

- Measures of generalization
- Double descent
- Techniques for improving generalization
- Generalization theory beyond VC-dimension
- Implicit regularization
- Why NN outperforms kernel

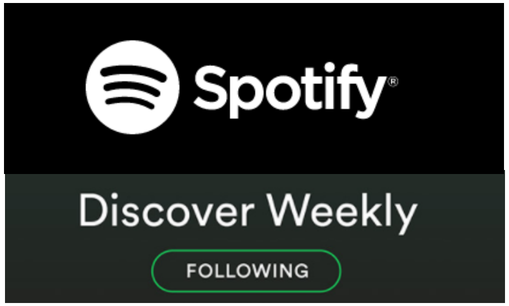
Topic 6: Representation Learning / Pre-Training

- Multi-task representation learning
- Auto-regressive pre-training
- Multi-modal learning
- Contrastive learning
- Meta-learning
- Data
- Theory

Topic 7: Generative Models

- Generative adversarial network
- Variational Auto-Encoder
- Energy-based models
- Normalizing flows
- Diffusion models

Machine Learning Review

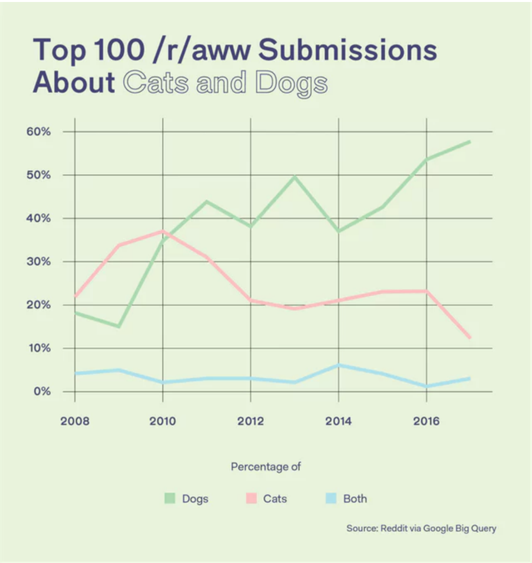


ML uses past data to make predictions

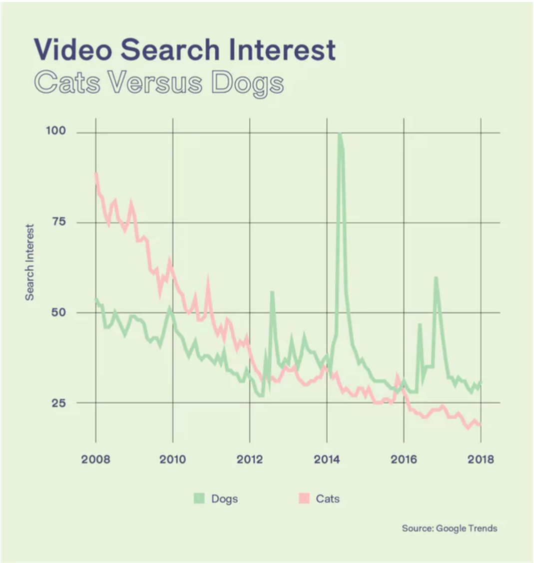
Traditional algorithms

Social media mentions of Cats vs. Dogs

Reddit



Google



Twitter?

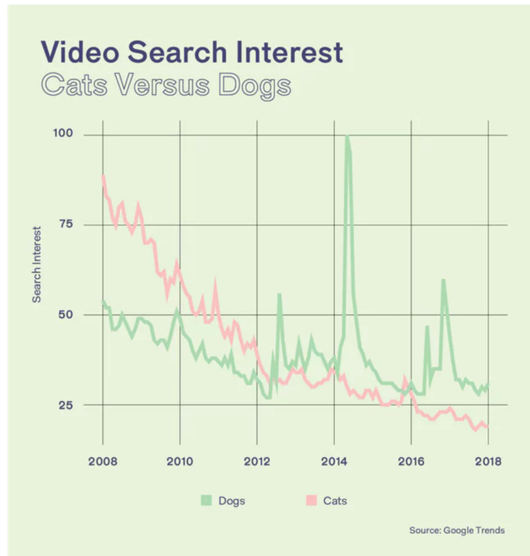
Graphics courtesy of https://theoutline.com/post/3128/dogs-cats-internet-popularity?zd=1

Traditional algorithms

Social media mentions of Cats vs. Dogs

Reddit

Google



Twitter?

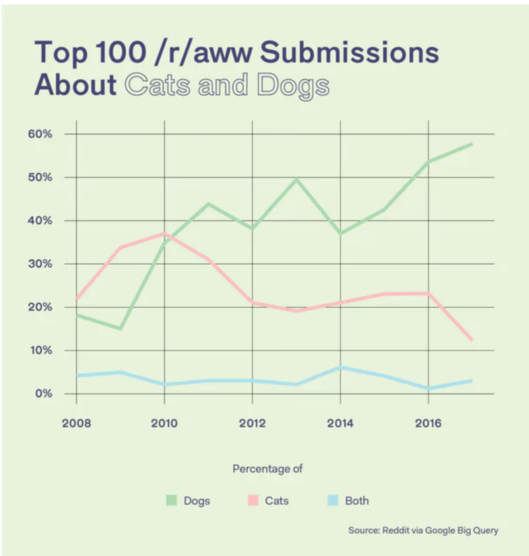
Write a program that sorts tweets into those containing "**cat**", "**dog**", or ***other***

Graphics courtesy of https://theoutline.com/post/3128/dogs-cats-internet-popularity?zd=1

Traditional algorithms

Social media mentions of Cats vs. Dogs

Reddit



Google

Twitter?

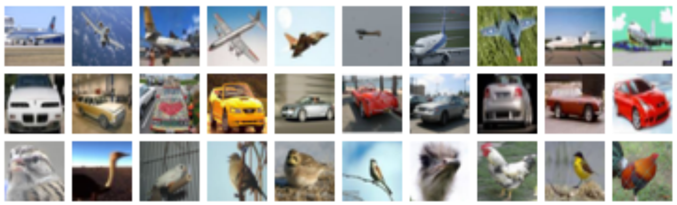
```
cats = []
dogs = []
other = []
for tweet in tweets:
    if "cat" in tweet:
        cats.append(tweet)
    elseif "dog" in tweet:
        dogs.append(tweet)
    else:
        other.append(tweet)
return cats, dogs, other
```

Write a program that sorts tweets into those containing "**cat**", "**dog**", or ***other***

Graphics courtesy of https://theoutline.com/post/3128/dogs-cats-internet-popularity?zd=1

Machine learning algorithms

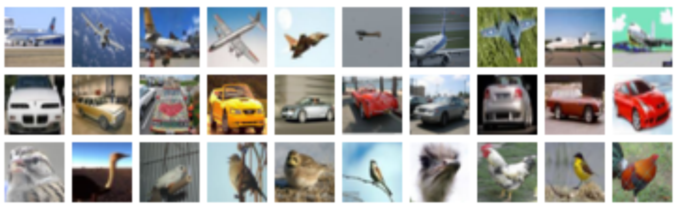
Write a program that sorts images
into those containing “**birds**”,
“**airplanes**”, or ***other.***



airplane
other
bird

Machine learning algorithms

Write a program that sorts image
into those containing "**birds**", "**airplanes**", or ***other.***

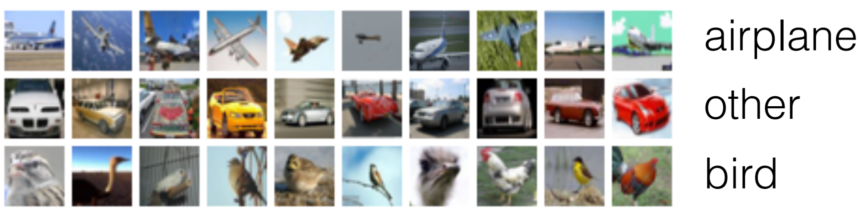


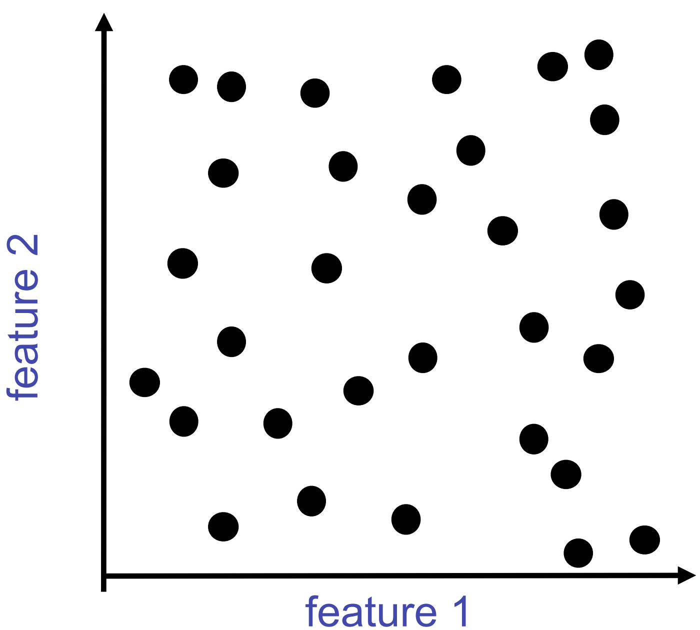
airplane
other
bird

```
birds = []
planes = []
other = []
for image in images:
    if bird in image:
        birds.append(image)
    elseif plane in image:
        planes.append(image)
    else:
        other.append(tweet)
return birds, planes, other
```

Machine learning algorithms

Write a program that sorts image
into those containing “**birds**”,
“**airplanes**”, or ***other.***



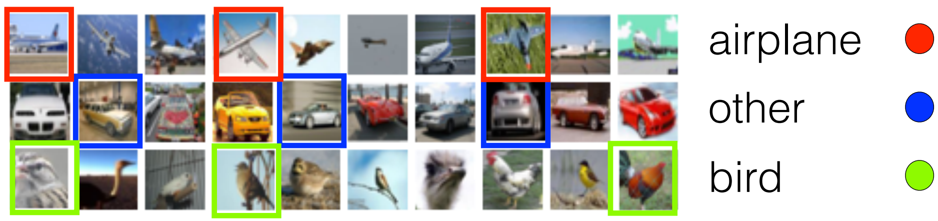


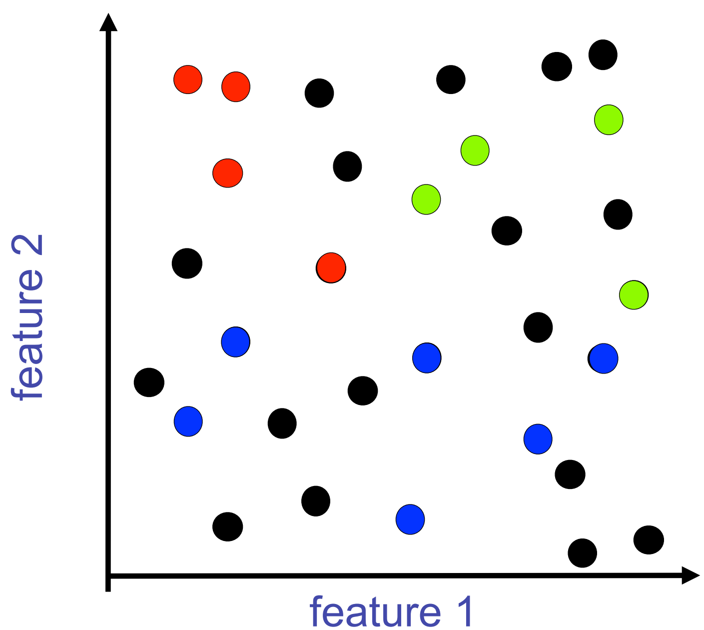

```
birds = []
planes = []
other = []
for image in images:
    if bird in image:
        birds.append(image)
    elseif plane in image:
        planes.append(image)
    else:
        other.append(tweet)
return birds, planes, other
```

Machine learning algorithms

Write a program that sorts image
into those containing "**birds**",
"**airplanes**", or ***other.***

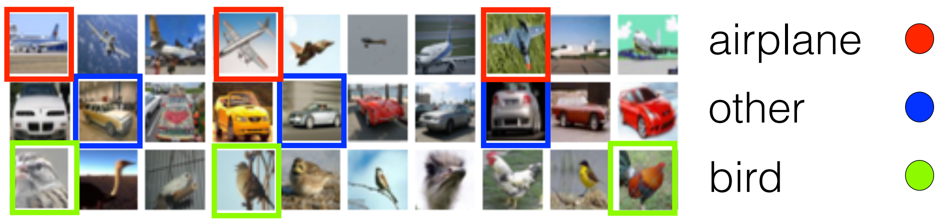
```
birds = []
planes = []
other = []
for image in images:
    if bird in image:
        birds.append(image)
    elseif plane in image:
        planes.append(image)
    else:
        other.append(tweet)
return birds, planes, other
```

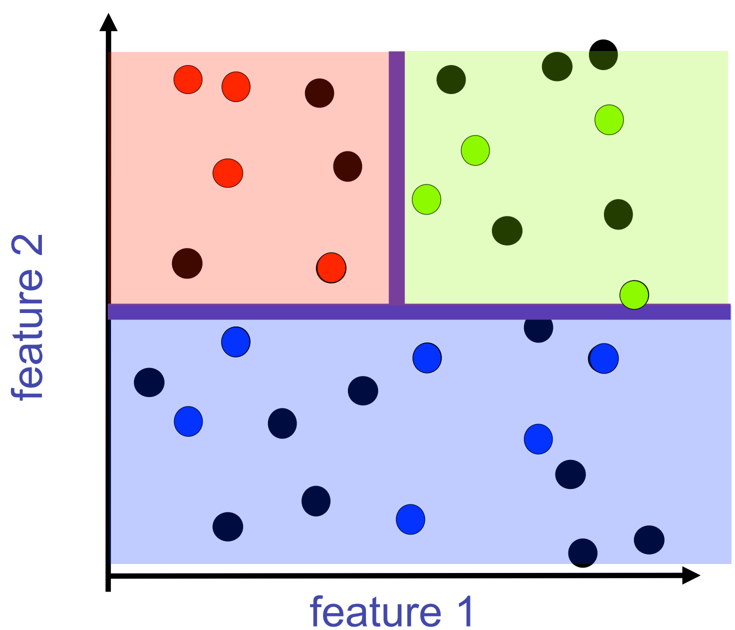




Machine learning algorithms

Write a program that sorts image
into those containing "**birds**", "**airplanes**", or ***other.***

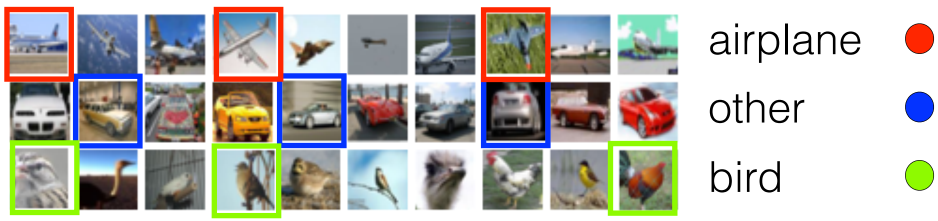




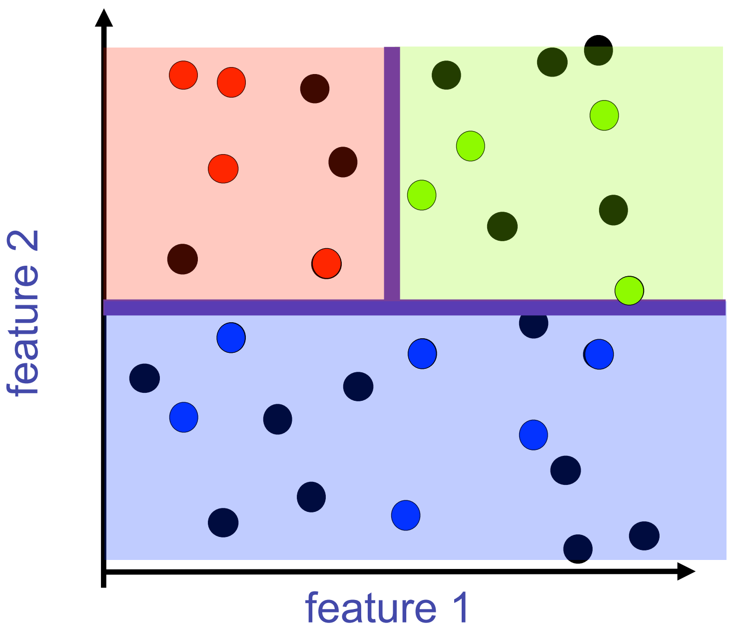

```
birds = []
planes = []
other = []
for image in images:
    if bird in image:
        birds.append(image)
    elseif plane in image:
        planes.append(image)
    else:
        other.append(tweet)
return birds, planes, other
```

Machine learning algorithms

Write a program that sorts images into those containing "**birds**", "**airplanes**", or ***other.***




```
birds = []
planes = []
other = []
for image in images:
    if bird in image:
        birds.append(image)
    elseif plane in image:
        planes.append(image)
    else:
        other.append(tweet)
return birds, planes, other
```



The decision rule of

`if "cat" in tweet:`

is **hard coded by expert.**

The decision rule of

`if bird in image:`

is **LEARNED using DATA**

Machine Learning Ingredients

- **Data**: past observations
- **Hypotheses/Models**: devised to capture the patterns in data
- **Prediction**: apply model to forecast future observations

Your first consulting job

- *Billionaire*: I have special coin, if I flip it, what's the probability it will be heads?

- *You*: Please flip it a few times: HHTHT

- *You*: The probability is:

- *Billionaire:* Why?

Coin – Binomial Distribution

- **Data**: sequence *D= (HHTHT…),* **k heads** out of **n flips**
- **Hypothesis:** P(Heads) = θ, P(Tails) = 1-θ
  - Flips are i.i.d.:
    - Independent events
    - Identically distributed according to Binomial distribution

- $P(\mathcal{D}|\theta) =$

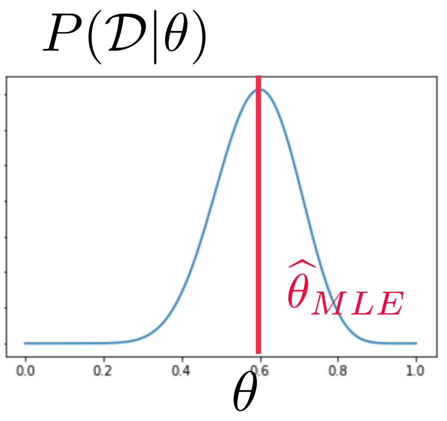
Maximum Likelihood Estimation

- **Data**: sequence *D= (HHTHT…),* **k heads** out of **n flips**
- **Hypothesis:** P(Heads) = θ, P(Tails) = 1-θ

$$P(\mathcal{D}|\theta) = \theta^k (1-\theta)^{n-k}$$

- Maximum likelihood estimation (MLE): Choose θ that maximizes the probability of observed data:

$$\begin{aligned}\widehat{\theta}_{MLE} &= \arg\max_\theta \; P(\mathcal{D}|\theta) \\ &= \arg\max_\theta \; \log P(\mathcal{D}|\theta)\end{aligned}$$



Your first learning algorithm

$$\widehat{\theta}_{MLE} = \arg\max_{\theta} \log P(\mathcal{D}|\theta)$$

$$= \arg\max_{\theta} \log \theta^k (1-\theta)^{n-k}$$

- Set derivative to zero: $$\frac{d}{d\theta} \log P(\mathcal{D}|\theta) = 0$$

Maximum Likelihood Estimation

Observe $X_1, X_2, \ldots, X_n$ drawn IID from $f(x;\theta)$ for some "true" $\theta = \theta_*$

Likelihood function $L_n(\theta) = \prod_{i=1}^{n} f(X_i;\theta)$

Log-Likelihood function $l_n(\theta) = \log(L_n(\theta)) = \sum_{i=1}^{n} \log(f(X_i;\theta))$

Maximum Likelihood Estimator (MLE) $\widehat{\theta}_{MLE} = \arg\max_{\theta} L_n(\theta)$

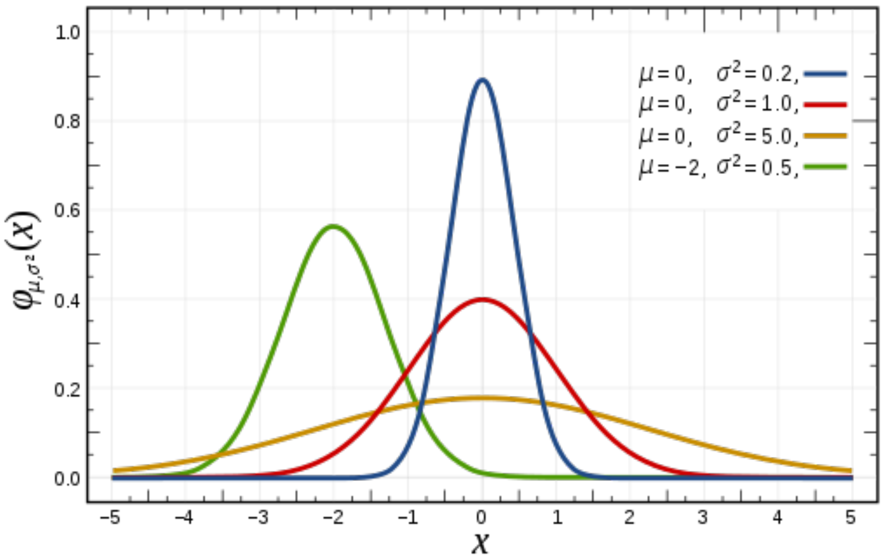
Recap

- Learning is…
  - Collect some data
    - E.g., coin flips
  - Choose a hypothesis class or model
    - E.g., binomial
  - Choose a loss function
    - E.g., data likelihood
  - Choose an optimization procedure
    - E.g., set derivative to zero to obtain MLE

What about continuous variables?

- *Billionaire*: What if I am measuring a **continuous variable**?
- ***You*: Let me tell you about Gaussians…**

$$P(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$



Some properties of Gaussians

- affine transformation (multiplying by scalar and adding a constant)
  - $X \sim N(\mu,\sigma^2)$
  - $Y = aX + b \quad \rightarrow \quad Y \sim N(a\mu+b, a^2\sigma^2)$

- Sum of Gaussians
  - $X \sim N(\mu_X,\sigma^2_X)$
  - $Y \sim N(\mu_Y,\sigma^2_Y)$
  - $Z = X+Y \quad \rightarrow \quad Z \sim N(\mu_X+\mu_Y, \sigma^2_X+\sigma^2_Y)$

MLE for Gaussian

- Prob. of i.i.d. samples $D=\{x_1,\ldots,x_n\}$ (e.g., temperature):

$$P(\mathcal{D}|\mu,\sigma) = P(x_1,\ldots,x_n|\mu,\sigma)$$
$$= \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^n \prod_{i=1}^{n} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}$$

- Log-likelihood of data:

$$\log P(\mathcal{D}|\mu,\sigma) = -n\log(\sigma\sqrt{2\pi}) - \sum_{i=1}^{n}\frac{(x_i-\mu)^2}{2\sigma^2}$$

- What is $\widehat{\theta}_{MLE}$ for $\theta = (\mu, \sigma^2)$?

Your second learning algorithm: MLE for mean of a Gaussian

- What's MLE for mean?

$$\frac{d}{d\mu}\log P(\mathcal{D}|\mu,\sigma) = \frac{d}{d\mu}\left[-n\log(\sigma\sqrt{2\pi}) - \sum_{i=1}^{n}\frac{(x_i-\mu)^2}{2\sigma^2}\right]$$

MLE for variance

- Again, set derivative to zero:

$$\frac{d}{d\sigma} \log P(\mathcal{D}|\mu, \sigma) = \frac{d}{d\sigma} \left[-n \log(\sigma\sqrt{2\pi}) - \sum_{i=1}^{n} \frac{(x_i - \mu)^2}{2\sigma^2} \right]$$

Learning Gaussian parameters

- MLE:

$$\widehat{\mu}_{MLE} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\widehat{\sigma^2}_{MLE} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \widehat{\mu}_{MLE})^2$$

- MLE for the variance of a Gaussian is **biased**

$$\mathbb{E}[\widehat{\sigma^2}_{MLE}] \neq \sigma^2$$

- Unbiased variance estimator:

$$\widehat{\sigma^2}_{unbiased} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \widehat{\mu}_{MLE})^2$$

Maximum Likelihood Estimation

Observe $X_1, X_2, \ldots, X_n$ drawn IID from $f(x;\theta)$ for some "true" $\theta = \theta_*$

Likelihood function $L_n(\theta) = \prod_{i=1}^{n} f(X_i;\theta)$

Log-Likelihood function $l_n(\theta) = \log(L_n(\theta)) = \sum_{i=1}^{n} \log(f(X_i;\theta))$

Maximum Likelihood Estimator (MLE) $\widehat{\theta}_{MLE} = \arg\max_{\theta} L_n(\theta)$

Under benign assumptions, as the number of observations $n \to \infty$ we have $\widehat{\theta}_{MLE} \to \theta_*$

The MLE is a "recipe" that begins with a *model* for data $f(x;\theta)$

Maximum Likelihood Estimation

Observe $X_1, X_2, \ldots, X_n$ drawn IID from $f(x;\theta)$ for some "true" $\theta = \theta_*$

Likelihood function $L_n(\theta) = \prod_{i=1}^{n} f(X_i;\theta)$

Log-Likelihood function $l_n(\theta) = \log(L_n(\theta)) = \sum_{i=1}^{n} \log(f(X_i;\theta))$

Maximum Likelihood Estimator (MLE) $\widehat{\theta}_{MLE} = \arg\max_{\theta} L_n(\theta)$

Under benign assumptions, as the number of observations $n \to \infty$ we have $\widehat{\theta}_{MLE} \to \theta_*$

Why is it useful to recover the "true" parameters θ_* of a probabilistic model?

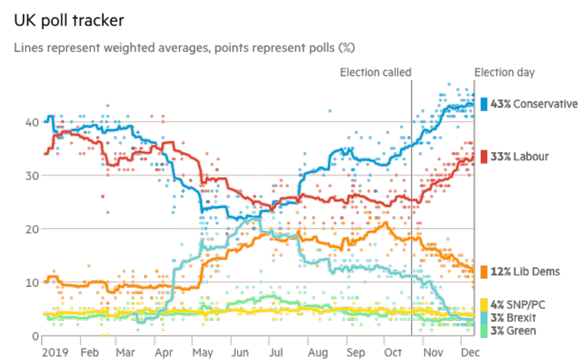
- **Estimation** of the parameters θ_* is the goal
- Help **interpret** or summarize large datasets
- Make **predictions** about future data
- **Generate** new data $X \sim f(\,\cdot\,;\widehat{\theta}_{MLE})$

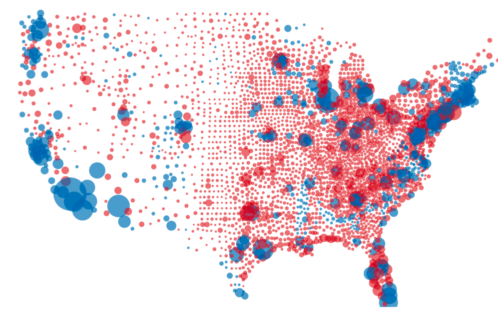
Estimation

Observe $X_1, X_2, \ldots, X_n$ drawn IID from $f(x;\theta)$ for some "true" $\theta = \theta_*$

Opinion polls
How does the greater population feel about an issue? Correct for over-sampling?

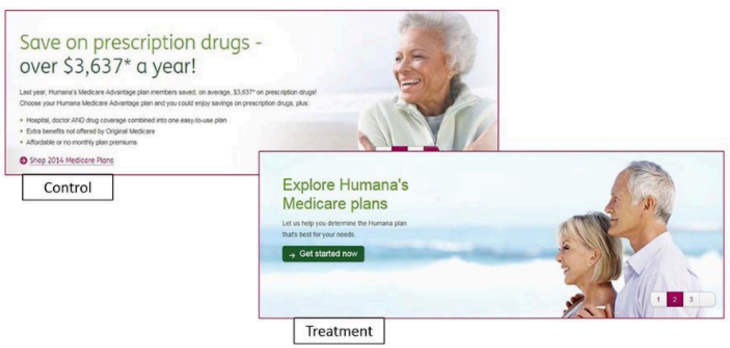
- θ_* is "true" average opinion
- $X_1, X_2, \ldots$ are sample calls





A/B testing
How do we figure out which ad results in more click-through?

- θ_* are the "true" average rates
- $X_1, X_2, \ldots$ are binary "clicks"

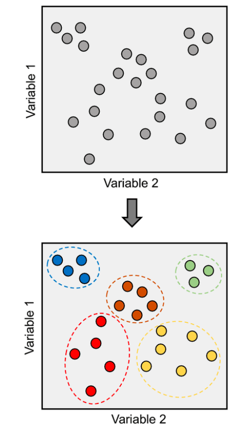


Interpret

Observe $X_1, X_2, \ldots, X_n$ drawn IID from $f(x;\theta)$ for some "true" $\theta = \theta_*$

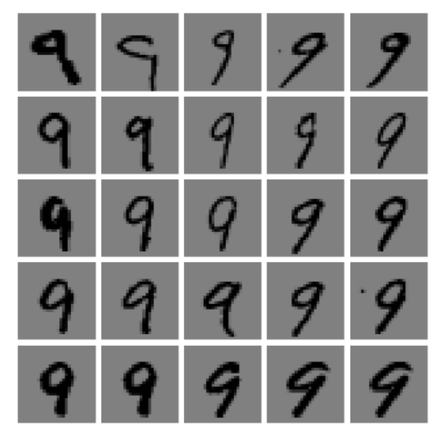
Customer segmentation / clustering
Can we identify distinct groups of customers by their behavior?

- θ_* describes "center" of distinct groups
- $X_1, X_2, \ldots$ are individual customers



Data exploration
What are the degrees of freedom of the dataset?

- θ_* describes the principle directions of variation
- $X_1, X_2, \ldots$ are the individual images

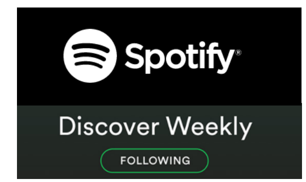


Predict

Observe $X_1, X_2, \ldots, X_n$ drawn IID from $f(x;\theta)$ for some "true" $\theta = \theta_*$

Content recommendation
Can we predict how much someone will like a movie based on past ratings?

- θ_* describes user's preferences
- $X_1, X_2, \ldots$ are (movie, rating) pairs



Object recognition / classification
Identify a flower given just its picture?

- θ_* describes the characteristics of each kind of flower
- $X_1, X_2, \ldots$ are the (image, label) pairs

(a) (b) (c)

Figure 1.1: Three types of Iris flowers: Setosa, Versicolor and Virginica. Used with kind permission of Dennis Kramb and SIGNA.

index	sl	sw	pl	pw	label
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
	...				
50	7.0	3.2	4.7	1.4	Versicolor
	...				
149	5.9	3.0	5.1	1.8	Virginica

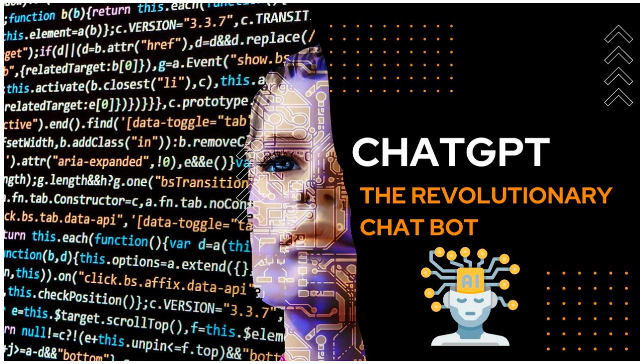
Generate

$$\textbf{Observe } X_1, X_2, \ldots, X_n \text{ drawn IID from } f(x;\theta) \text{ for some "true" } \theta = \theta_*$$

Text generation
Can AI generate text that could have been written like a human?

- θ_* describes language structure
- $X_1, X_2, \ldots$ are text snippets found online

"Kaia the dog wasn't a natural pick to go to mars. No one could have predicted she would..."



https://chat.openai.com/chat

Image to text generation
Can AI generate an image from a prompt?

- θ_* describes the coupled structure of images and text
- $X_1, X_2, \ldots$ are the (image, caption) pairs found online

"dog talking on cell phone under water, oil painting"

https://labs.openai.com/

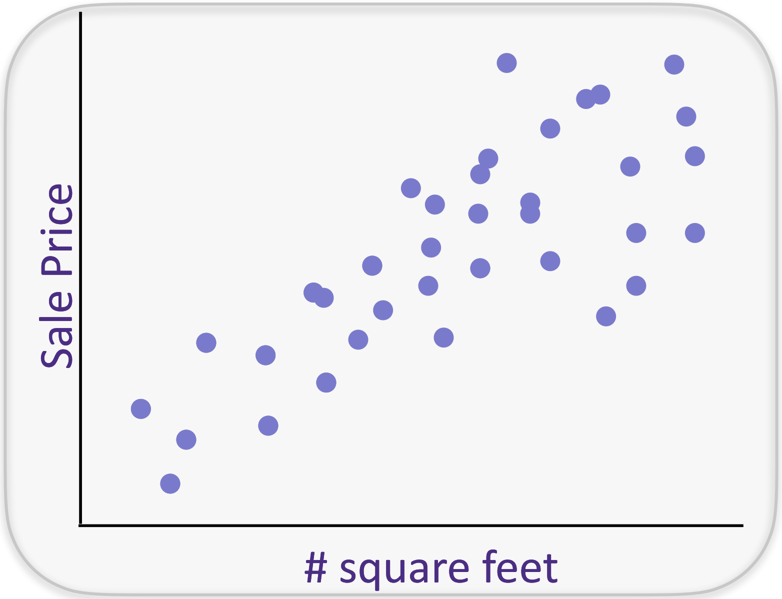
Linear Regression

The regression problem, 1-dimensional

Given past sales data on zillow.com, predict:

y = **House sale price** *from*

x = **{# sq. ft.}**



Training Data:

$$\{(x_i, y_i)\}_{i=1}^n$$

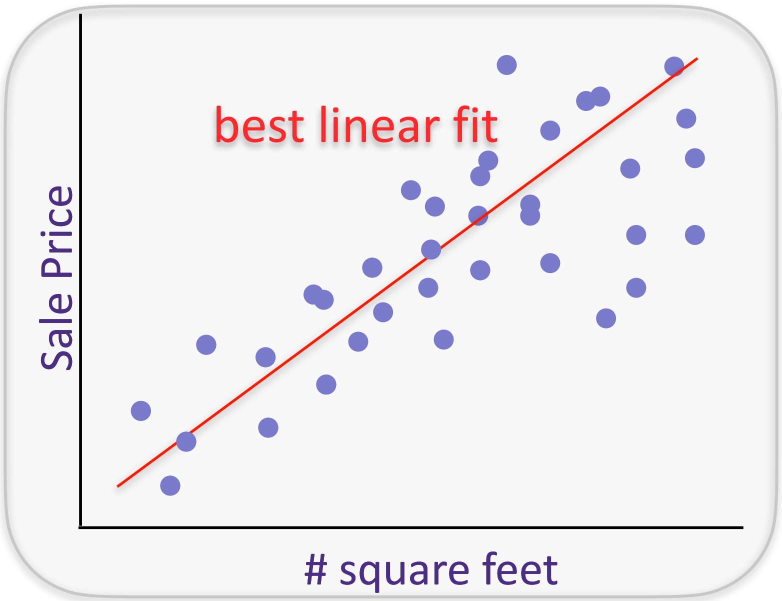
$$x_i \in \mathbb{R}$$
$$y_i \in \mathbb{R}$$

Fit a function to our data, 1-d

Given past sales data on zillow.com, predict:

y = **House sale price** *from*

x = **{# sq. ft.}**



Training Data: $\{(x_i, y_i)\}_{i=1}^n$ $\quad x_i \in \mathbb{R}$, $y_i \in \mathbb{R}$

Hypothesis/Model: linear

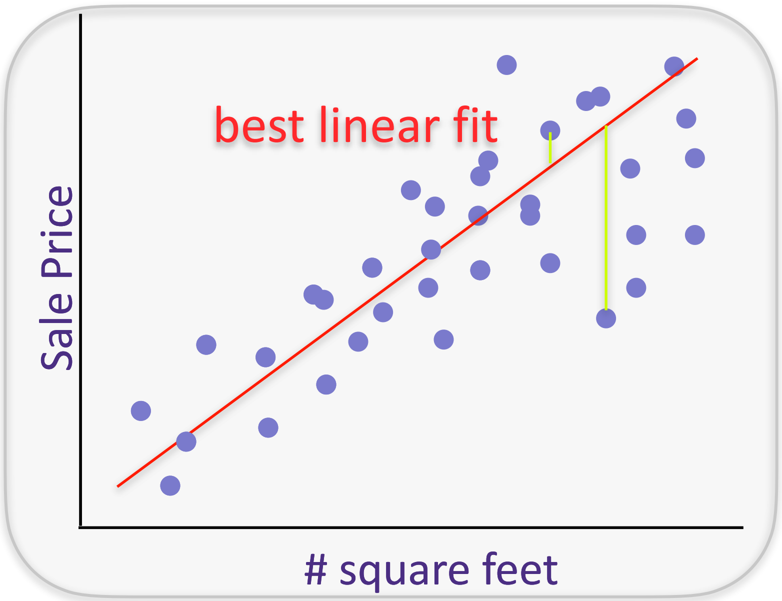
$$y_i = x_i w + \epsilon_i \qquad \epsilon_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2)$$

Fit a function to our data, 1-d

Given past sales data on zillow.com, predict:

y = **House sale price** *from*

x = **{# sq. ft.}**



Training Data: $\{(x_i, y_i)\}_{i=1}^n$ $\quad x_i \in \mathbb{R}$, $y_i \in \mathbb{R}$

Hypothesis/Model: linear

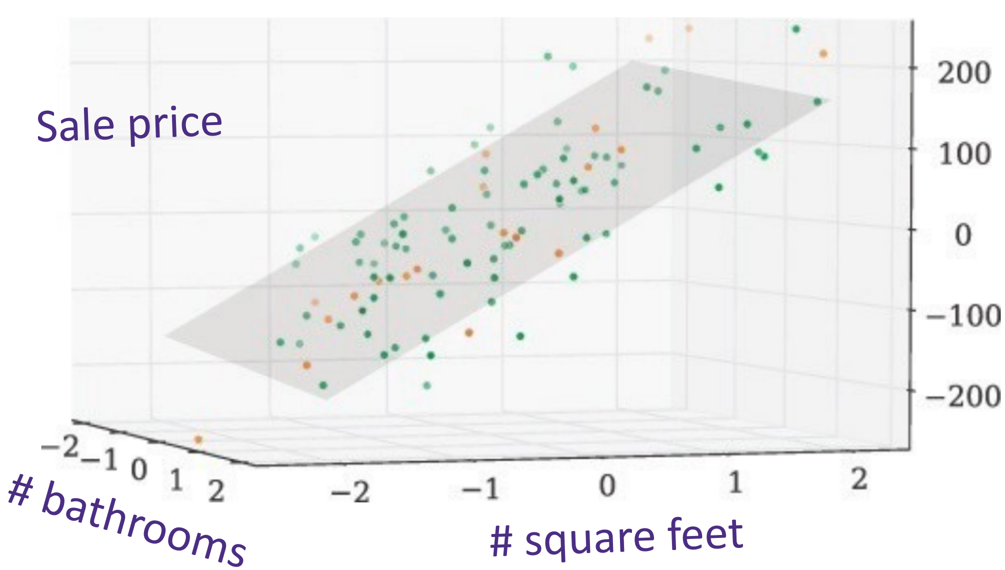
$$y_i = x_i w + \epsilon_i \qquad \epsilon_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2)$$

The regression problem, d-dim

Given past sales data on zillow.com, predict:

y = **House sale price** *from*

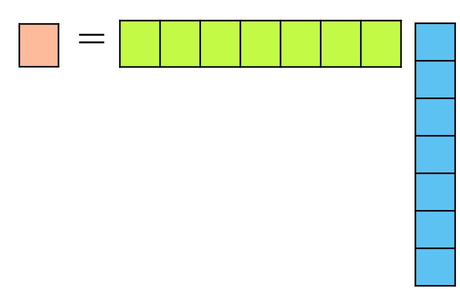
x = **{# sq. ft., zip code, date of sale, etc.}**



Training Data: $\{(x_i, y_i)\}_{i=1}^n$ $\quad x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$

Hypothesis/Model: linear

$$y_i = x_i^T w + \epsilon_i \qquad \epsilon_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2)$$

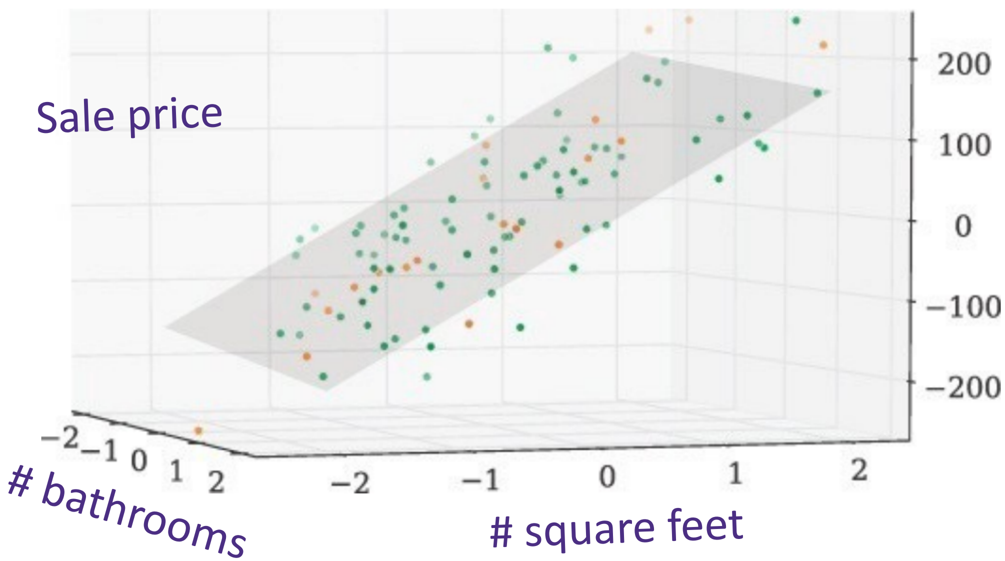


The regression problem, d-dim

Given past sales data on zillow.com, predict:

y = **House sale price** *from*

x = **{# sq. ft., zip code, date of sale, etc.}**



Training Data: $\{(x_i, y_i)\}_{i=1}^n$ $\quad x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$

Hypothesis/Model: linear

$$y_i = x_i^T w + \epsilon_i \qquad \epsilon_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2)$$

$$p(y|x, w, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y - x^\top w)^2 / 2\sigma^2}$$

Maximizing log-likelihood

Training Data:
$\{(x_i, y_i)\}_{i=1}^n$

$x_i \in \mathbb{R}^d$
$y_i \in \mathbb{R}$

$$p(y|x, w, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y - x^\top w)^2/2\sigma^2}$$

Likelihood: $$P(\mathcal{D}|w, \sigma) = \prod_{i=1}^n p(y_i|x_i, w, \sigma) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y_i - x_i^\top w)^2/2\sigma^2}$$

Maximum Likelihood Estimation

Observe $X_1, X_2, \ldots, X_n$ drawn IID from $f(x;\theta)$ for some "true" $\theta = \theta_*$

Likelihood function $L_n(\theta) = \prod_{i=1}^{n} f(X_i;\theta)$

Log-Likelihood function $l_n(\theta) = \log(L_n(\theta)) = \sum_{i=1}^{n} \log(f(X_i;\theta))$

Maximum Likelihood Estimator (MLE) $\widehat{\theta}_{MLE} = \arg\max_{\theta} L_n(\theta)$

Under benign assumptions, as the number of observations $n \to \infty$ we have $\widehat{\theta}_{MLE} \to \theta_*$

Why is it useful to recover the "true" parameters θ_* of a probabilistic model?

- **Estimation** of the parameters θ_* is the goal
- Help **interpret** or summarize large datasets
- Make **predictions** about future data
- **Generate** new data $X \sim f(\,\cdot\,;\,\widehat{\theta}_{MLE})$

Maximizing log-likelihood

Training Data: $\{(x_i, y_i)\}_{i=1}^n$ $\quad x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$

$$p(y|x,w,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y-x^\top w)^2/2\sigma^2}$$

Likelihood: $$P(\mathcal{D}|w,\sigma) = \prod_{i=1}^n p(y_i|x_i,w,\sigma) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y_i-x_i^\top w)^2/2\sigma^2}$$

Maximize (wrt w): $$\log P(\mathcal{D}|w,\sigma) = \log\left(\prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y_i-x_i^\top w)^2/2\sigma^2}\right)$$

Maximizing log-likelihood

Training Data: $\{(x_i, y_i)\}_{i=1}^n$ $\quad x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$

$$p(y|x,w,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y-x^\top w)^2/2\sigma^2}$$

Likelihood: $P(\mathcal{D}|w,\sigma) = \prod_{i=1}^n p(y_i|x_i,w,\sigma) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y_i-x_i^\top w)^2/2\sigma^2}$

Maximize (wrt w): $\log P(\mathcal{D}|w,\sigma) = \log\left(\prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y_i-x_i^\top w)^2/2\sigma^2}\right)$

$$\widehat{w}_{MLE} = \arg\min_w \sum_{i=1}^n (y_i - x_i^\top w)^2$$

Maximizing log-likelihood

$$\widehat{w}_{MLE} = \arg\min_{w} \sum_{i=1}^{n} (y_i - x_i^\top w)^2$$

Set derivate=0, solve for w

$$\widehat{w}_{MLE} = \left(\sum_{i=1}^{n} x_i x_i^\top \right)^{-1} \sum_{i=1}^{n} x_i y_i$$

The regression problem in matrix notation

$$\widehat{w}_{MLE} = \arg\min_w \sum_{i=1}^{n} (y_i - x_i^\top w)^2$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix}$$

d : # of features

n : # of examples/datapoints

The regression problem in matrix notation

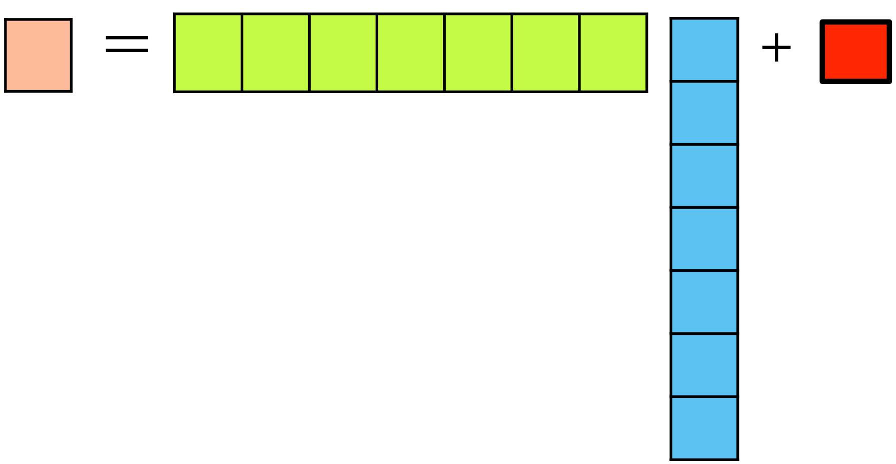
$$\widehat{w}_{MLE} = \arg\min_{w} \sum_{i=1}^{n} (y_i - x_i^\top w)^2$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix}$$

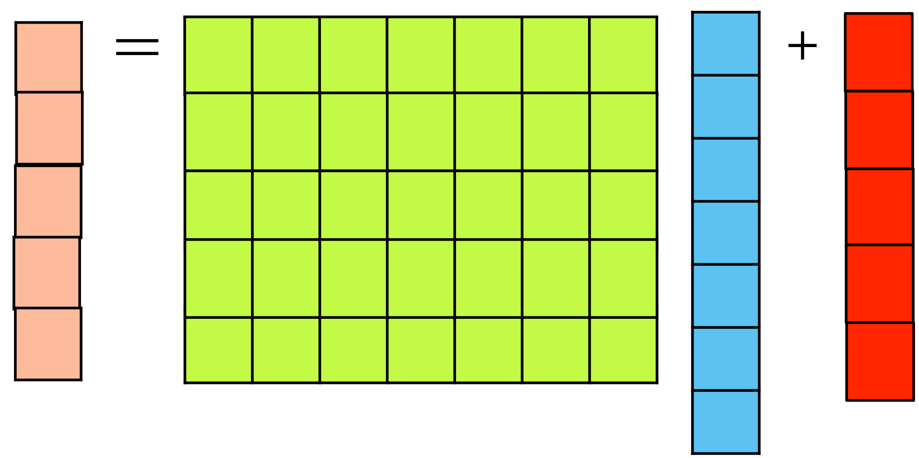
d : # of features

n : # of examples/datapoints

$$y_i = x_i^T w + \epsilon_i$$



$$\mathbf{y} = \mathbf{X}w + \epsilon$$



The regression problem in matrix notation

$$\widehat{w}_{MLE} = \arg\min_w \sum_{i=1}^n (y_i - x_i^\top w)^2$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix}$$

d : # of features
n : # of examples/datapoints

$$y_i = x_i^T w + \epsilon_i$$

$$\mathbf{y} = \mathbf{X}w + \epsilon$$

$$\begin{aligned} \widehat{w}_{LS} &= \arg\min_w ||\mathbf{y} - \mathbf{X}w||_2^2 \\ &= \arg\min_w (\mathbf{y} - \mathbf{X}w)^T(\mathbf{y} - \mathbf{X}w) \end{aligned}$$

ℓ_2 norm: $\|z\|_2 = \sqrt{\sum_{i=1}^n z_i^2} = \sqrt{z^\top z}$

The regression problem in matrix notation

$$\widehat{w}_{MLE} = \arg\min_w \sum_{i=1}^{n} (y_i - x_i^\top w)^2$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix}$$

d : # of features

n : # of examples/datapoints

$$y_i = x_i^T w + \epsilon_i$$

$$\mathbf{y} = \mathbf{X}w + \epsilon$$

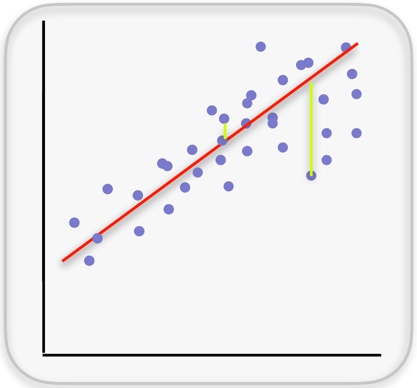
$$\widehat{w}_{LS} = \arg\min_w ||\mathbf{y} - \mathbf{X}w||_2^2$$

$$= \arg\min_w (\mathbf{y} - \mathbf{X}w)^T (\mathbf{y} - \mathbf{X}w)$$

$$\widehat{w}_{LS} = \widehat{w}_{MLE} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

The regression problem in matrix notation

$$\widehat{w}_{LS} = \arg\min_w ||\mathbf{y} - \mathbf{X}w||_2^2$$
$$= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$



What about an offset?

$$\widehat{w}_{LS}, \widehat{b}_{LS} = \arg\min_{w,b} \sum_{i=1}^{n} \left(y_i - (x_i^T w + b)\right)^2$$
$$= \arg\min_{w,b} ||\mathbf{y} - (\mathbf{X}w + \mathbf{1}b)||_2^2$$

Dealing with an offset

$$\widehat{w}_{LS}, \widehat{b}_{LS} = \arg\min_{w,b} ||\mathbf{y} - (\mathbf{X}w + \mathbf{1}b)||_2^2$$

Dealing with an offset

$$\widehat{w}_{LS}, \widehat{b}_{LS} = \arg\min_{w,b} ||\mathbf{y} - (\mathbf{X}w + \mathbf{1}b)||_2^2$$

$$\mathbf{X}^T\mathbf{X}\widehat{w}_{LS} + \widehat{b}_{LS}\mathbf{X}^T\mathbf{1} = \mathbf{X}^T\mathbf{y}$$
$$\mathbf{1}^T\mathbf{X}\widehat{w}_{LS} + \widehat{b}_{LS}\mathbf{1}^T\mathbf{1} = \mathbf{1}^T\mathbf{y}$$

If $\mathbf{X}^T\mathbf{1} = 0$ (i.e., if each feature is mean-zero) then

$$\widehat{w}_{LS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

$$\widehat{b}_{LS} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

Make Predictions

$$\widehat{w}_{LS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

$$\widehat{b}_{LS} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

A new house is about to be listed. What should it sell for?

$$\hat{y}_{\text{new}} = x_{\text{new}}^T \hat{w}_{LS} + \hat{b}_{LS}$$

Process

Decide on a **model** for the likelihood function $f(x;\theta)$

Find the function which fits the data best

Choose a loss function- least squares

Pick the function which minimizes loss on data

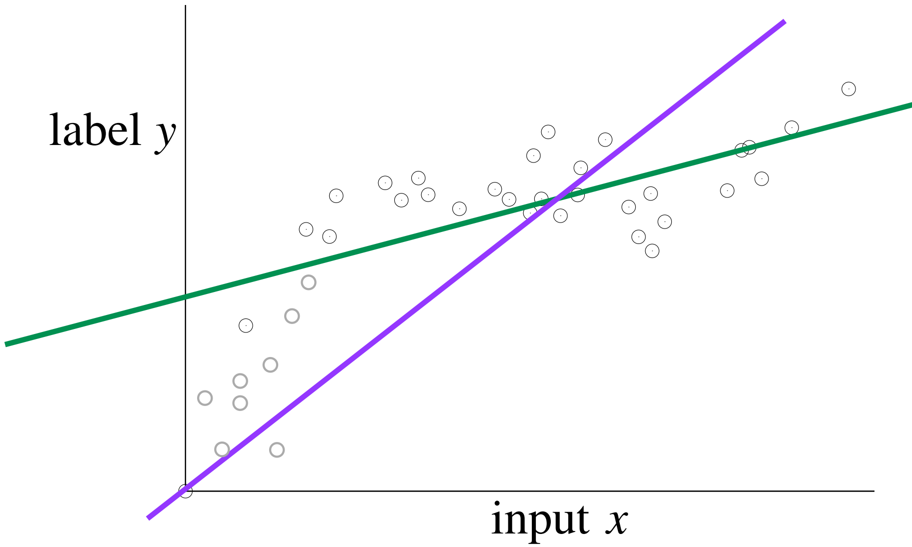
Use function to make prediction on new examples

Linear regression with non-linear basis functions

Quadratic regression in 1-dimension

- **Data:** $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \ \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

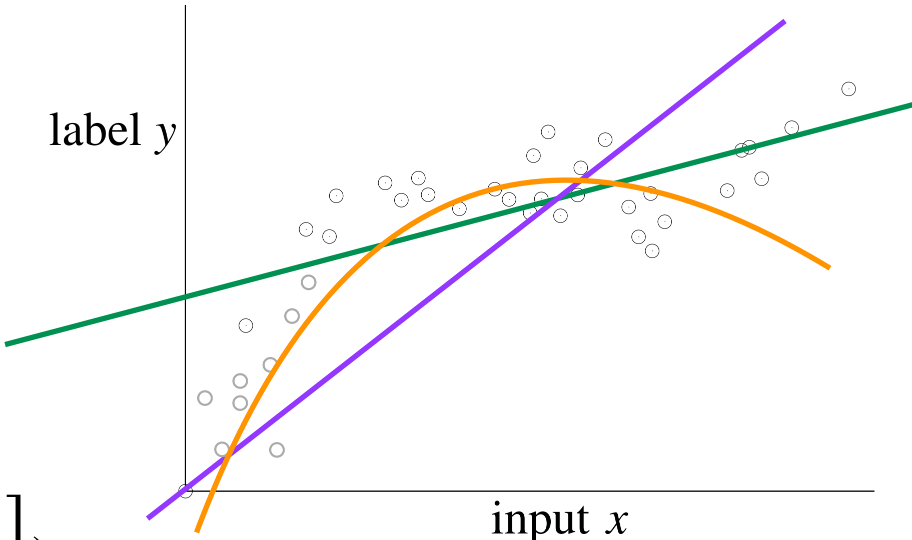
- **Linear model with parameter** (b, w_1)**:**
  - $\hat{y}_i = b + w_1 x_i$



Quadratic regression in 1-dimension

- **Data:** $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \ \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

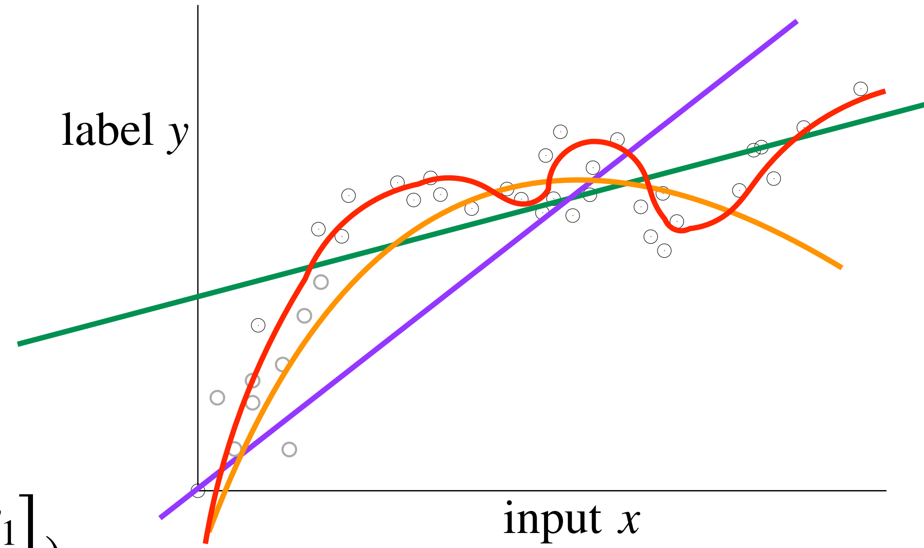
- **Linear model with parameter** (b, w_1)**:**
  - $\hat{y}_i = b + w_1 x_i$
- **Quadratic model with parameter** $(b, w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix})$**:**
  - $\hat{y}_i = b + w_1 x_i + w_2 x_i^2$



Quadratic regression in 1-dimension

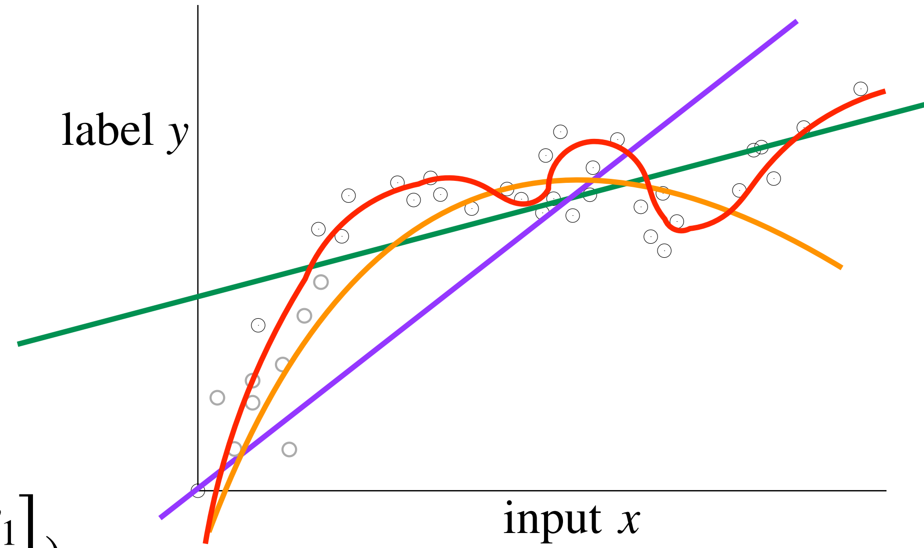
- **Data:** $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

- **Linear model with parameter** (b, w_1)**:**
  - $\hat{y}_i = b + w_1 x_i$
- **Quadratic model with parameter** $(b, w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix})$**:**
  - $\hat{y}_i = b + w_1 x_i + w_2 x_i^2$
- **Degree-p polynomial model with parameter** $(b, w = \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix})$**:**
  - $\hat{y}_i = b + w_1 x_i + w_2 x_i^2 + \ldots + w_p x_i^p$



Quadratic regression in 1-dimension

- **Data:** $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

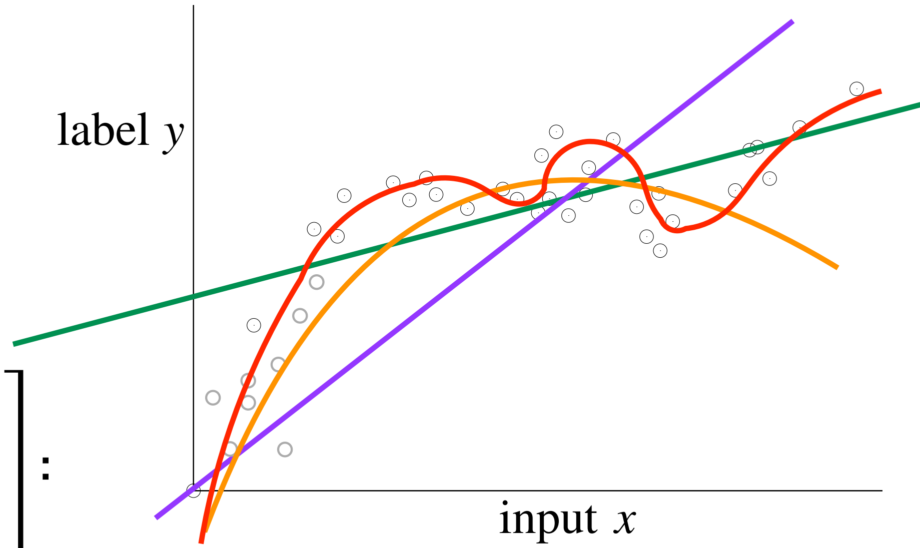


- **Linear model with parameter** (b, w_1):
  - $\hat{y}_i = b + w_1 x_i$
- **Quadratic model with parameter** $(b, w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix})$:
  - $\hat{y}_i = b + w_1 x_i + w_2 x_i^2$
- **Degree-p polynomial model with parameter** $(b, w = \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix})$:
  - $\hat{y}_i = b + w_1 x_i + w_2 x_i^2 + \ldots + w_p x_i^p$
- **General p-features with parameter** $w = \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix}$:
  - $\hat{y}_i = \langle w, h(x_i) \rangle$ where $h : \mathbb{R} \to \mathbb{R}^p$

Quadratic regression in 1-dimension

- **Data:** $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \ \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

- **General p-features with parameter** $w = \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix}$**:**
  - $\hat{y}_i = \langle w, h(x_i) \rangle$ where $h : \mathbb{R} \to \mathbb{R}^p$

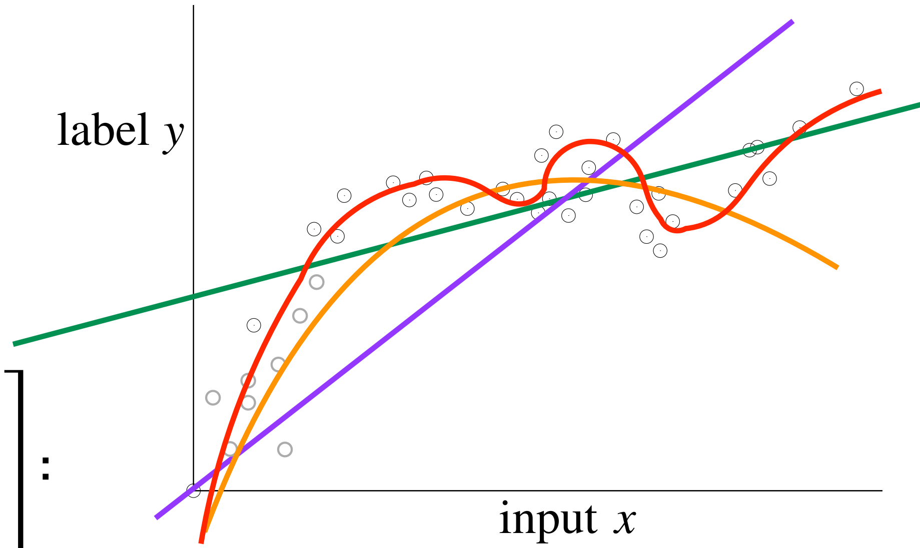


Note: h can be arbitrary non-linear functions!

$$h(x) = \left[\log(x), x^2, \sin(x), \sqrt{x}\right]^\top$$

Quadratic regression in 1-dimension

- **Data:** $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$



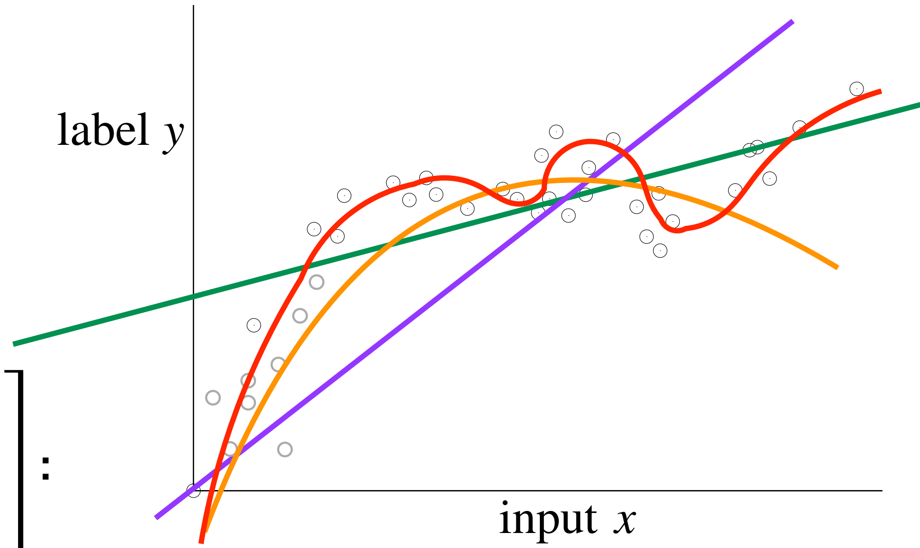
- **General p-features with parameter** $w = \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix}$:
  - $\widehat{y}_i = \langle w, h(x_i) \rangle$ where $h : \mathbb{R} \rightarrow \mathbb{R}^p$

How do we learn w?

Quadratic regression in 1-dimension

- **Data:** $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \ \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

- **General p-features with parameter** $w = \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix}$:
  - $\widehat{y}_i = \langle w, h(x_i) \rangle$ where $h : \mathbb{R} \to \mathbb{R}^p$



How do we learn w?

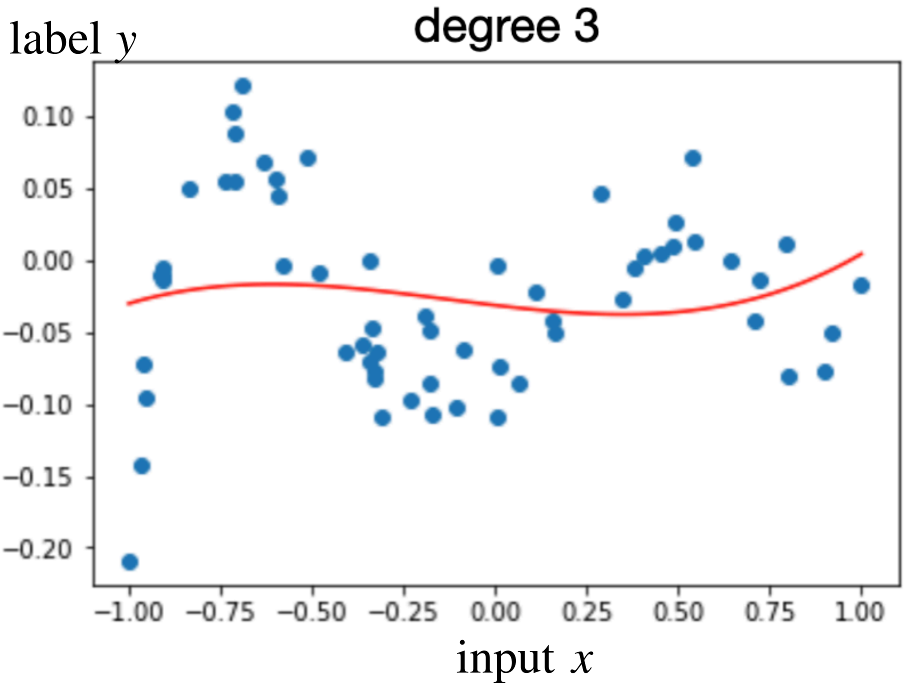
$$\mathbf{H} = \begin{bmatrix} - - h(x_1)^\top - - \\ \vdots \\ - - h(x_n)^\top - - \end{bmatrix} \in \mathbb{R}^{n \times p}$$

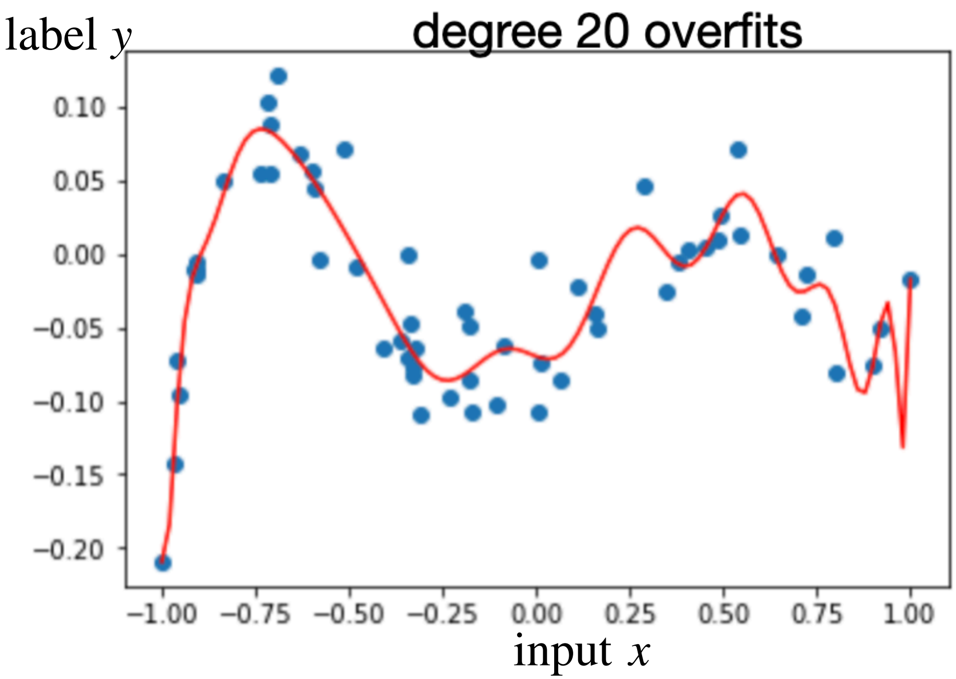
$$\widehat{w} = \arg\min_w \|\mathbf{H}w - \mathbf{y}\|_2^2$$

For a new test point x, predict
$\widehat{y} = \langle \widehat{w}, h(x) \rangle$

Which p should we choose?

- First instance of class of models with different representation power = model complexity





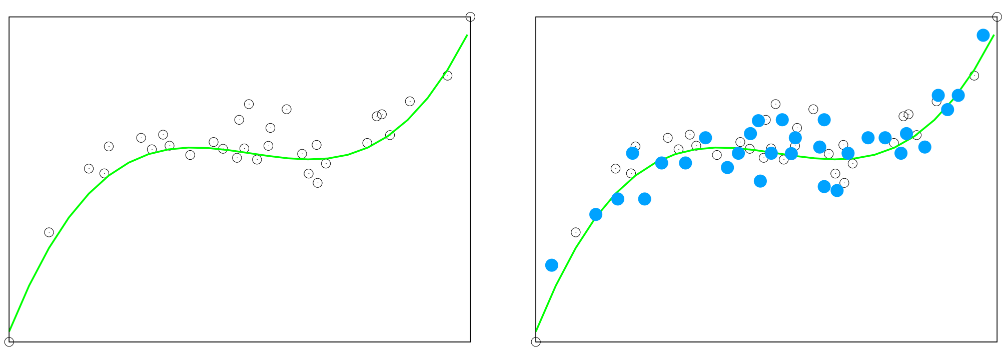
- How do we determine which is better model?

Generalization

- we say a predictor **generalizes** if it performs as well on unseen data as on training data (we will formalize the next lecture)
- the data used to train a predictor is **training data** or **in-sample data**
- we want the predictor to work on **out-of-sample data**
- we say a predictor **fails to generalize** if it performs well on in-sample data but does not perform well on out-of-sample data

Generalization

- we say a predictor **generalizes** if it performs as well on unseen data as on training data (we will formalize the next lecture)
- the data used to train a predictor is **training data** or **in-sample data**
- we want the predictor to work on **out-of-sample data**
- we say a predictor **fails to generalize** if it performs well on in-sample data but does not perform well on out-of-sample data

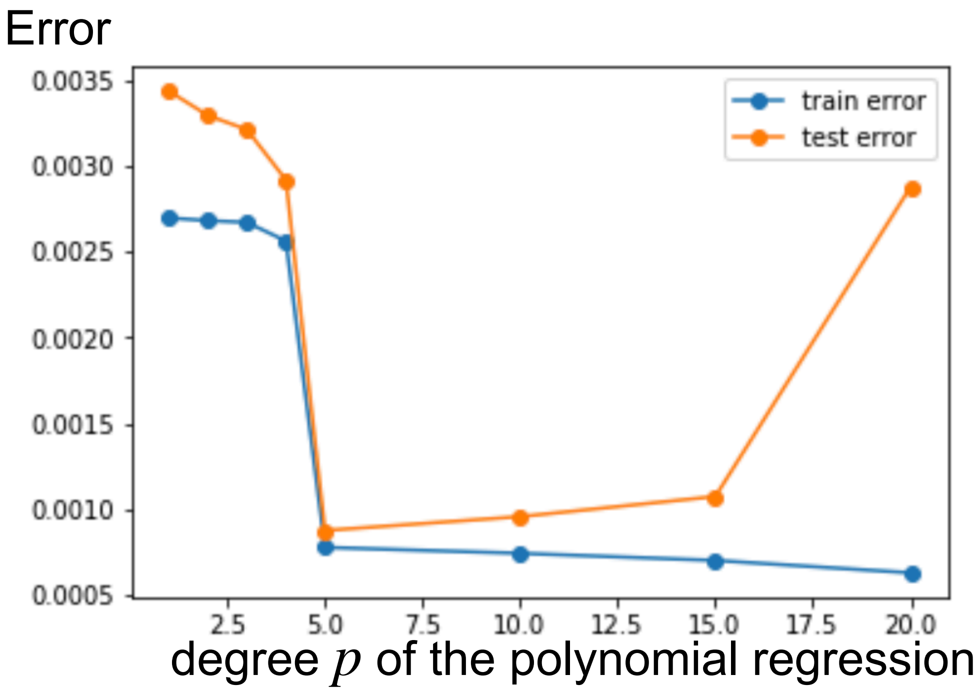


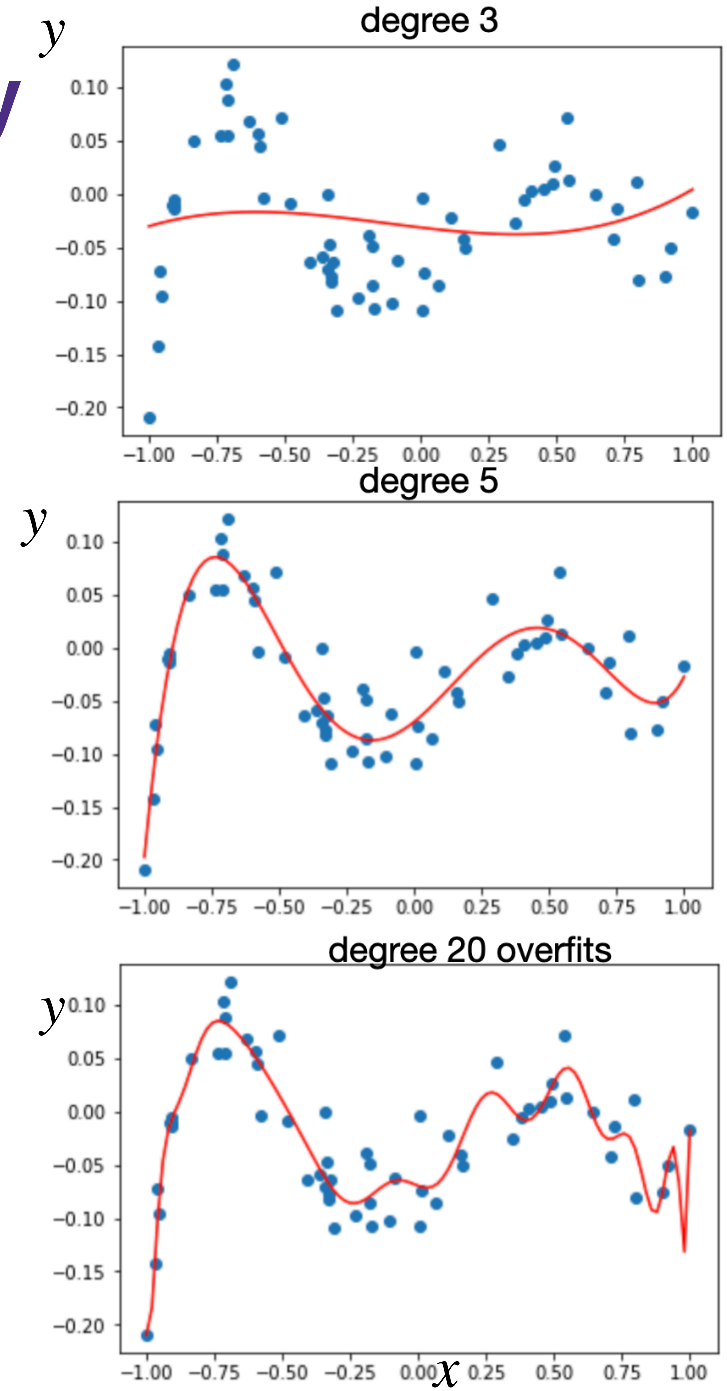
- **train** a cubic predictor on 32 (**in-sample**) white circles: Mean Squared Error (MSE) 174
- **predict** label y for 30 (**out-of-sample**) blue circles: MSE 192
- conclude this predictor/model generalizes, as in-sample MSE $\simeq$ out-of-sample MSE

Split the data into training and testing

- a way to mimic how the predictor performs on unseen data
- given a single dataset $S = \{(x_i, y_i)\}_{i=1}^{n}$
- we split the dataset into two: training set and test set (e.g., 90/10)
- **training set** used to train the model
  - $$\text{minimize} \quad \mathscr{L}_{\text{train}}(w) = \frac{1}{|S_{\text{train}}|} \sum_{i \in S_{\text{train}}} (y_i - x_i^T w)^2$$
- **test set** used to evaluate the model
  - $$\mathscr{L}_{\text{test}}(w) = \frac{1}{|S_{\text{test}}|} \sum_{i \in S_{\text{test}}} (y_i - x_i^T w)^2$$
- this assumes that test set is similar to unseen data
- **test set should never be used in training or picking unknowns**

Train/test error vs. complexity





- Degree $p = 5$, since it achieves **minimum test error**
- **Train error** monotonically decreases with model complexity
- **Test error** has a U shape

test set should never be used in training or picking degree

Cross-Validation

How… How… How????????

- How do we pick the number of basis functions…

- We could use the test data, but…

How… How… How????????

- How do we pick the number of basis functions…
- We could use the test data, but…
  - Never ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever train on the test data

(LOO) Leave-one-out cross validation

- **Consider a validation set with 1 example:**
  - **D – training data**
  - **D\j – training data with j th data point (x_j ,y_j) moved to validation set**
- **Learn classifier $f_{D\backslash j}$ with *D*\j dataset**
- **Estimate true error** as squared error on predicting $\mathbf{y_j}$:
  - **Unbiased estimate of $\text{error}_{true}(f_{D\backslash j})$!**

(LOO) Leave-one-out cross validation

- **Consider a validation set with 1 example:**
  - **D – training data**
  - **D\j – training data with j th data point (x_j ,y_j) moved to validation set**
- **Learn classifier $f_{D\backslash j}$ with *D*\j dataset**
- **Estimate true error** as squared error on predicting **y_j**:
  - **Unbiased estimate of $\text{error}_{\text{true}}(f_{D\backslash j})$!**

- **LOO cross validation**: Average over all data points *j*:
  - **For each data point you leave out, learn a new classifier $f_{D\backslash j}$**
  - **Estimate error** as:

$$\text{error}_{LOO} = \frac{1}{n}\sum_{j=1}^{n}(y_j - f_{\mathcal{D}\backslash j}(x_j))^2$$

LOO cross validation is (almost) unbiased estimate!

- When computing LOOCV error, we only use *N-1* data points
  - So it's not estimate of true error of learning with *N* data points
  - Usually pessimistic, though – learning with less data typically gives worse answer

- LOO is almost unbiased! Use LOO error for model selection!!!
  - E.g., picking degree

Computational cost of LOO

- Suppose you have 100,000 data points
- You implemented a great version of your learning algorithm
  - Learns in only 1 second
- Computing LOO will take about 1 day!!!
  -

Use *k*-fold cross validation

- Randomly divide training data into *k* equal parts
  - $D_1, \ldots, D_k$
- For each *i*
  - Learn classifier $f_{D \backslash D_i}$ using data point not in D_i
  - Estimate error of $f_{D \backslash D_i}$ on validation set D_i:

1	2	3	4	5
Train	Train	Validation	Train	Train

$$\text{error}_{\mathcal{D}_i} = \frac{1}{|\mathcal{D}_i|} \sum_{(x_j, y_j) \in \mathcal{D}_i} (y_j - f_{\mathcal{D} \backslash \mathcal{D}_i}(x_j))^2$$

Use *k*-fold cross validation

> **Randomly divide training data into *k* equal parts**
  - **$D_1,\ldots,D_k$**

1	2	3	4	5
Train	Train	Validation	Train	Train

> **For each *i***
  - **Learn classifier $f_{D\backslash Di}$ using data point not in D_i**
  - **Estimate error of $f_{D\backslash Di}$ on validation set D_i:**

$$\text{error}_{\mathcal{D}_i} = \frac{1}{|\mathcal{D}_i|} \sum_{(x_j, y_j) \in \mathcal{D}_i} (y_j - f_{\mathcal{D}\backslash\mathcal{D}_i}(x_j))^2$$

> **k-fold cross validation error is average** over data splits:

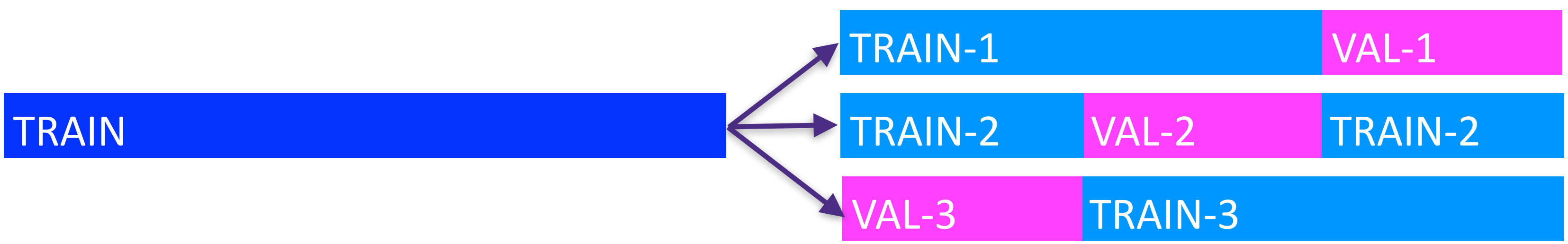
$$error_{k-fold} = \frac{1}{k} \sum_{i=1}^{k} error_{\mathcal{D}_i}$$

> **k-fold cross validation properties:**
  - **Much faster to compute** than LOO
  - **More (pessimistically) biased** – using much less data, only *n(k-1)/k*
  - **Usually, k = 10**

Recap

> Given a dataset, begin by splitting into

> Model selection: Use k-fold cross-validation on **TRAIN** to train predictor and choose magic parameters such as degree



> Model assessment: Use **TEST** to assess the accuracy of the model you output

- **Never ever ever ever ever train or choose parameters based on the test data**

Ridge Regression

UNIVERSITY *of* WASHINGTON

Regularization in Linear Regression

Recall Least Squares: $$\widehat{w}_{LS} = \arg\min_w \sum_{i=1}^n \left(y_i - x_i^T w\right)^2$$

$$= \arg\min_w (\mathbf{y} - \mathbf{X}w)^T(\mathbf{y} - \mathbf{X}w)$$

when $(\mathbf{X}^T\mathbf{X})^{-1}$ exists.... $= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$

Regularization in Linear Regression

Recall Least Squares:

$$\widehat{w}_{LS} = \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2$$

$$= \arg\min_w (\mathbf{y} - \mathbf{X}w)^T(\mathbf{y} - \mathbf{X}w)$$

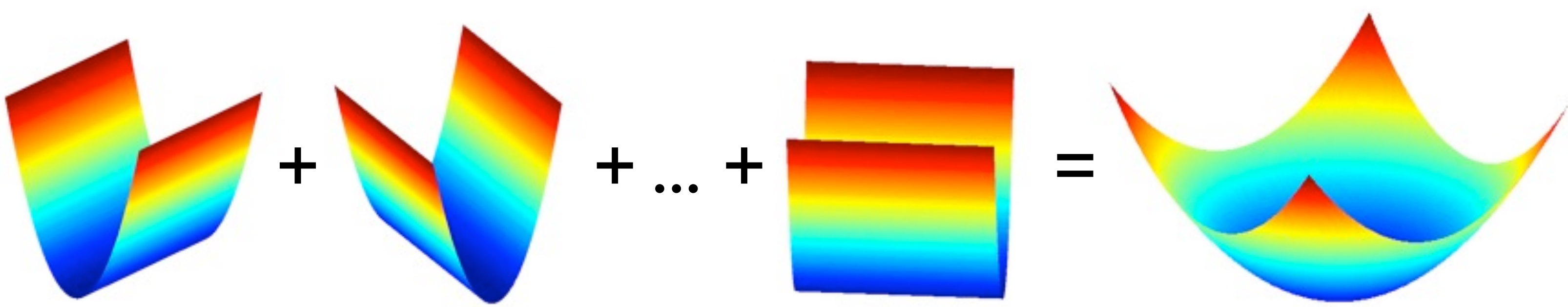
In general:

$$= \arg\min_w w^T(\mathbf{X}^T\mathbf{X})w - 2y^T\mathbf{X}w$$

Regularization in Linear Regression

Recall Least Squares: $$\widehat{w}_{LS} = \arg\min_w \sum_{i=1}^n \left(y_i - x_i^T w\right)^2$$
$$= \arg\min_w (\mathbf{y} - \mathbf{X}w)^T(\mathbf{y} - \mathbf{X}w)$$

In general: $$= \arg\min_w w^T(\mathbf{X}^T\mathbf{X})w - 2y^T\mathbf{X}w$$



$$(y_1 - x_1^T w)^2 + (y_2 - x_2^T w)^2 + \cdots + (y_n - x_n^T w)^2 = \sum_{i=1}^n (y_i - x_i^T w)^2$$

What if $x_i \in \mathbb{R}^d$ and $d > n$?

Regularization in Linear Regression

Recall Least Squares: $\widehat{w}_{LS} = \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2$

When $x_i \in \mathbb{R}^d$ and $d > n$ the objective function is flat in some directions:

Regularization in Linear Regression

Recall Least Squares: $\widehat{w}_{LS} = \arg\min_w \sum_{i=1}^n \left(y_i - x_i^T w\right)^2$

When $x_i \in \mathbb{R}^d$ and $d > n$ the objective function is flat in some directions:

Implies optimal solution is *not unique* and unstable due to lack of curvature:

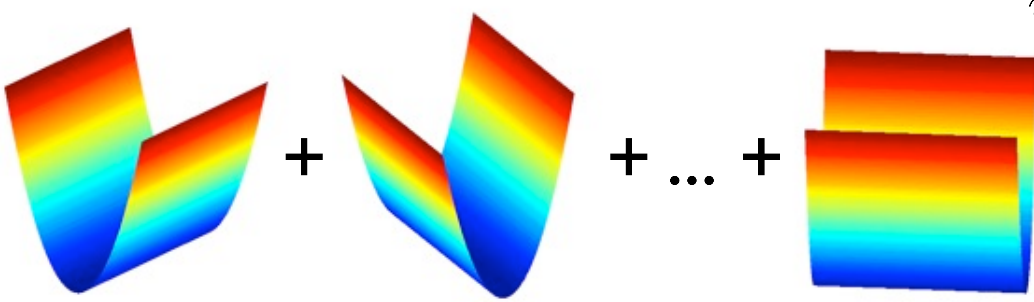
- small changes in training data result in large changes in solution
- often the *magnitudes* of *w* are "very large"

Regularization imposes "simpler" solutions by a "complexity" penalty

Ridge Regression

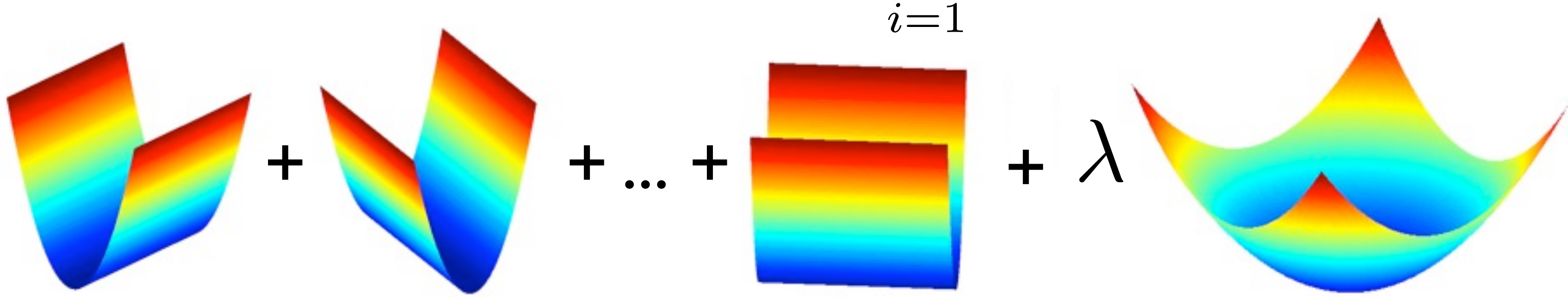
- Old Least squares objective:

$$\widehat{w}_{LS} = \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2$$



- Ridge Regression objective:

$$\widehat{w}_{ridge} = \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$



Minimizing the Ridge Regression Objective

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$

Shrinkage Properties

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$

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

Classification
Logistic Regression

Thus far, regression:

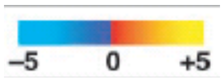
predict a continuous value given some inputs

Reading Your Brain, Simple Example

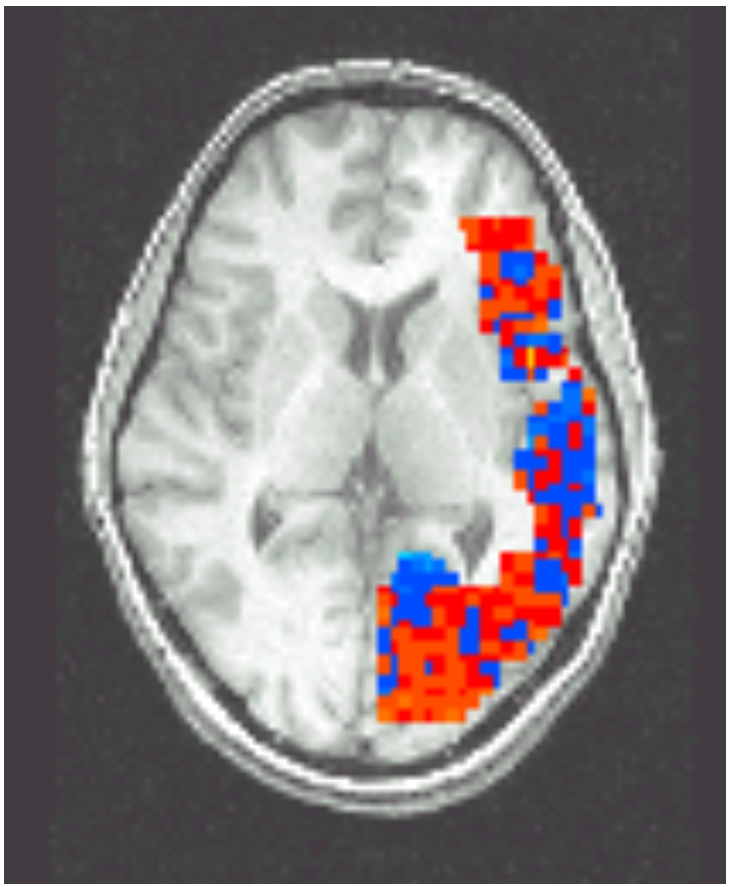
[Mitchell et al.]

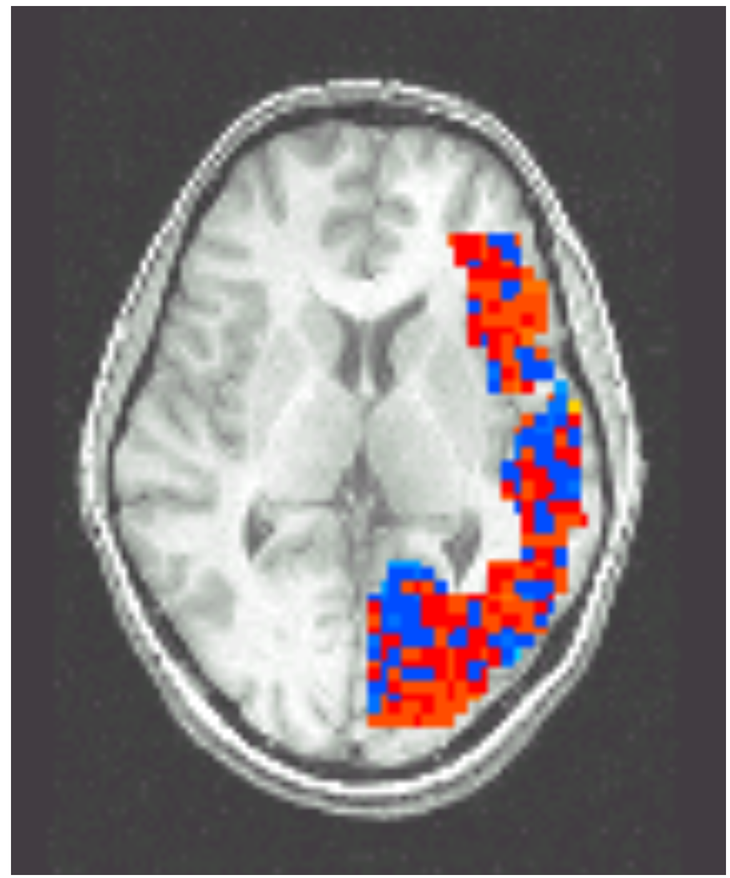
Pairwise classification accuracy: 85%

Person



Animal





Classification

- **Learn** $f : \mathcal{X} \rightarrow \mathcal{Y}$
  - $\mathcal{X} \subset \mathbb{R}^d$ **- features**
  - $\mathcal{Y} = \{1, \ldots, k\}$ **- target classes**

- **Loss Function** $\ell(f(x), y) = \mathbf{1}\{f(x) \neq y\}$
- **Expected loss of f:**

$$\mathbb{E}_{XY}[\mathbf{1}\{f(X) \neq Y\}] = \mathbb{E}_X[\mathbb{E}_{Y|X}[\mathbf{1}\{f(x) \neq Y\}|X = x]]$$

$$\mathbb{E}_{Y|X}[\mathbf{1}\{f(x) \neq Y\}|X = x] = \sum_i P(Y = i|X = x)\mathbf{1}\{f(x) \neq i\} = \sum_{i \neq f(x)} P(Y = i|X = x)$$

$$= 1 - P(Y = f(x)|X = x)$$

- **Suppose you knew P(Y|X) exactly, how should you classify?**

Classification

- **Learn** $f : \mathcal{X} \to \mathcal{Y}$
  - $\mathcal{X} \subset \mathbb{R}^d$ **- features**
  - $\mathcal{Y} = \{1, \ldots, k\}$ **- target classes**

- **Loss Function** $\ell(f(x), y) = \mathbf{1}\{f(x) \neq y\}$
- **Expected loss of f:**

$$\mathbb{E}_{XY}[\mathbf{1}\{f(X) \neq Y\}] = \mathbb{E}_X[\mathbb{E}_{Y|X}[\mathbf{1}\{f(x) \neq Y\}|X = x]]$$

$$\mathbb{E}_{Y|X}[\mathbf{1}\{f(x) \neq Y\}|X = x] = \sum_i P(Y = i|X = x)\mathbf{1}\{f(x) \neq i\} = \sum_{i \neq f(x)} P(Y = i|X = x)$$
$$= 1 - P(Y = f(x)|X = x)$$

- **Suppose you knew P(YlX) exactly, how should you classify?**
- **Bayes-Optimal classifier:**

$$f(x) = \arg\max_y \mathbb{P}(Y = y|X = x)$$

Bayes Optimal Binary Classifier

- **Bayes-Optimal classifier:** $f(x) = \arg\max_y \mathbb{P}(Y = y | X = x)$
- **Suppose we don't know $P(Y = y | X = x)$, but have n iid examples**

$$\{(x_i, y_i)\}_{i=1}^n \qquad Y \in \{0, 1\}$$

- **Suppose $\mathscr{X}$ is discrete so that $X \in \{1,2,\ldots,m\}$. What is a natural estimator for $P(Y = y | X = x)$?**

Bayes Optimal Binary Classifier

- **Bayes-Optimal classifier:** $f(x) = \arg\max_y \mathbb{P}(Y = y | X = x)$
- **Suppose we don't know** $P(Y = y \,|\, X = x)$**, but have n iid examples**

$$\{(x_i, y_i)\}_{i=1}^n \qquad\qquad Y \in \{0, 1\}$$

- **Suppose** $\mathcal{X}$ **is discrete so that** $X \in \{1,2,\ldots,m\}$**. What is a natural estimator for** $P(Y = y \,|\, X = x)$**?**

$$\hat{f}(x) = \arg\max_{y \in \{0,1\}} \frac{\sum_{i=1}^n \mathbf{1}[\mathbf{x_i = x, y_i = y}]}{\sum_{i=1}^n \mathbf{1}[\mathbf{x_i = x}]}$$

What if $\mathcal{X}$ **is continuous? That is, what if** $X \in \mathbb{R}^d$**?**

Bayes Optimal Binary Classifier

- **Bayes-Optimal classifier:** $f(x) = \arg\max_y \mathbb{P}(Y = y | X = x)$
- **Suppose we don't know $P(Y = y \,|\, X = x)$, but have n iid examples**

$$\{(x_i, y_i)\}_{i=1}^n \qquad\qquad Y \in \{0, 1\}$$

- **Suppose $\mathscr{X}$ is discrete so that $X \in \{1,2,\ldots,m\}$. What is a natural estimator for $P(Y = y \,|\, X = x)$?**

$$\hat{f}(x) = \arg\max_{y \in \{0,1\}} \frac{\sum_{i=1}^n \mathbf{1}[\mathbf{x_i = x, y_i = y}]}{\sum_{i=1}^n \mathbf{1}[\mathbf{x_i = x}]}$$

What if $\mathscr{X}$ is continuous? That is, what if $X \in \mathbb{R}^d$?

We need a <u>model</u> to explain observations

Logistic Regression

Recall linear regression:

- We assumed that for any x, we have $p(Y=y\,|\,X=x) = \frac{1}{\sqrt{2\pi}} e^{(y-w^T x)^2/2}$.
- Given data $\{(x_i, y_i)\}_{i=1}^n$ we then computed the MLE for w.

Logistic Regression

Recall linear regression:

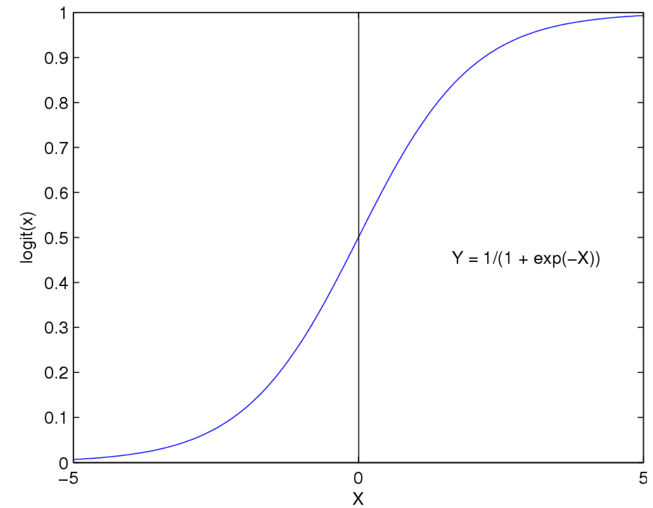
- We assumed that for any x, we have $p(Y = y \,|\, X = x) = \frac{1}{\sqrt{2\pi}} e^{(y - w^T x)^2/2}$.
- Given data $\{(x_i, y_i)\}_{i=1}^n$ we then computed the MLE for w.

Logistic regression uses a model specialized for classification:

$$\mathbb{P}[Y = 1 | X = x, w] = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$

$$\mathbb{P}[Y = 0 | X = x, w] = 1 - \sigma(w^T x) = \frac{\exp(-w^T x)}{1 + \exp(-w^T x)}$$

$$= \frac{1}{1 + \exp(w^T x)}$$



Logistic Regression

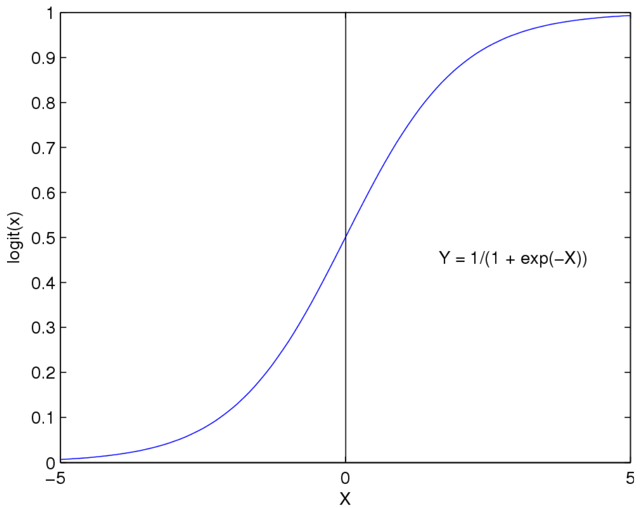
Recall linear regression:

- We assumed that for any x, we have $p(Y=y\,|\,X=x) = \frac{1}{\sqrt{2\pi}} e^{(y-w^T x)^2/2}$.
- Given data $\{(x_i, y_i)\}_{i=1}^n$ we then computed the MLE for w.

Logistic regression uses a model specialized for classification:

$$\mathbb{P}[Y=1|X=x,w] = \sigma(w^T x) = \frac{1}{1+\exp(-w^T x)}$$

$$\mathbb{P}[Y=0|X=x,w] = 1-\sigma(w^T x) = \frac{\exp(-w^T x)}{1+\exp(-w^T x)} = \frac{1}{1+\exp(w^T x)}$$

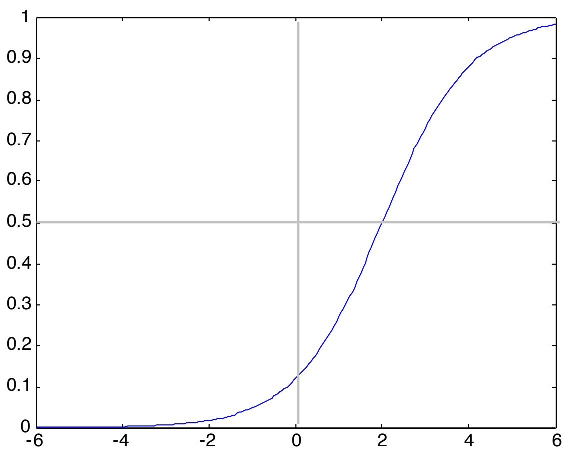


Features can be discrete or continuous!

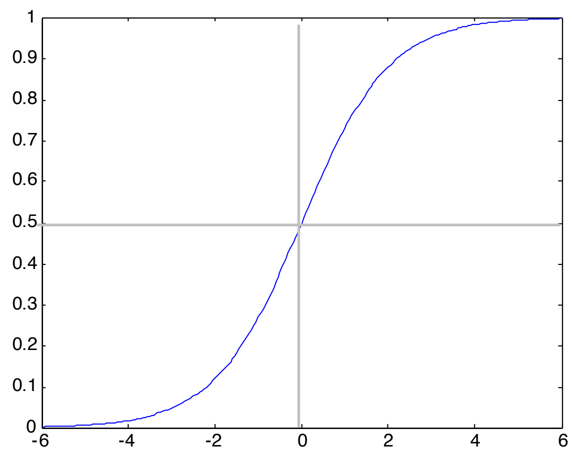
Understanding the sigmoid

$$\sigma(w_0 + \sum_k w_k x_k) = \frac{1}{1 + e^{w_0 + \sum_k w_k x_k}}$$

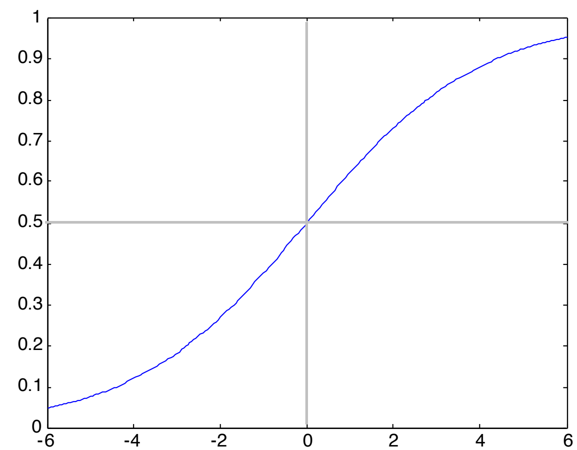
w_0=-2, w_1=-1



w_0=0, w_1=-1



w_0=0, w_1=-0.5



Sigmoid for binary classes

$$\mathbb{P}(Y = 0|w, X) = \frac{1}{1 + \exp(w_0 + \sum_k w_k X_k)}$$

$$\mathbb{P}(Y = 1|w, X) = 1 - \mathbb{P}(Y = 0|w, X) = \frac{\exp(w_0 + \sum_k w_k X_k)}{1 + \exp(w_0 + \sum_k w_k X_k)}$$

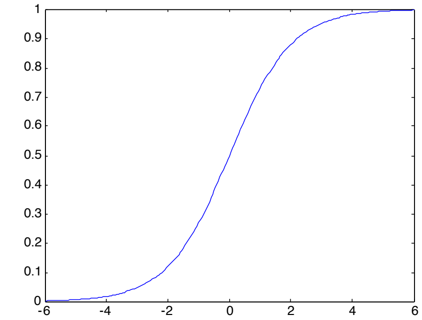
$$\frac{\mathbb{P}(Y = 1|w, X)}{\mathbb{P}(Y = 0|w, X)} = \exp(w_0 + \sum_k w_k X_k)$$

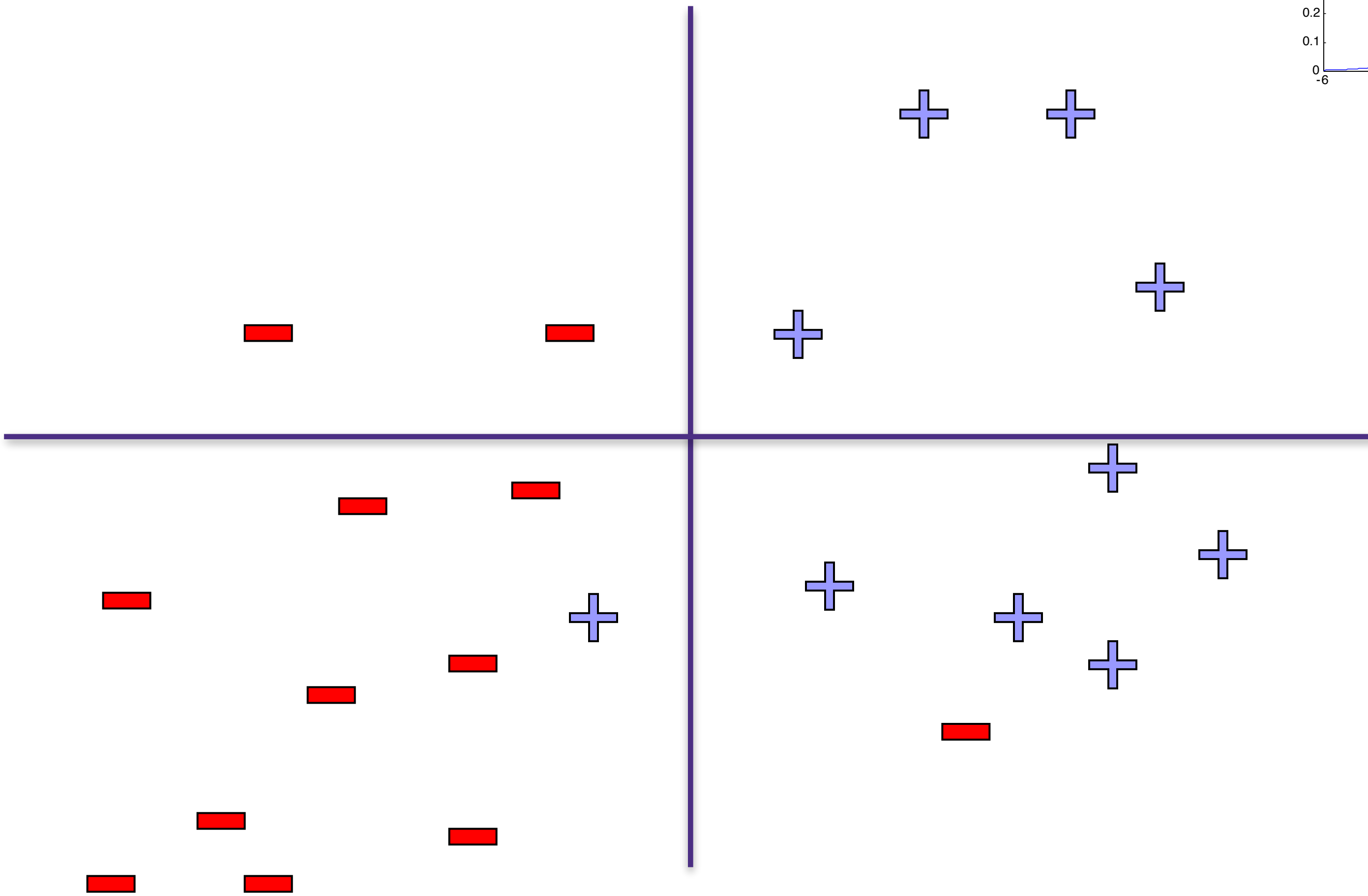
Linear Decision Rule!

$$\log \frac{\mathbb{P}(Y = 1|w, X)}{\mathbb{P}(Y = 0|w, X)} = w_0 + \sum_k w_k X_k$$

Logistic Regression – a Linear classifier

$$\frac{1}{1 + exp(-z)}$$





$$\ln \frac{P(Y=0|X)}{P(Y=1|X)} = w_0 + \sum_i w_i X_i$$

Loss function: Conditional Likelihood

- **Have a bunch of iid data:** $\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, \quad y_i \in \{-1, 1\}$

$$P(Y=-1|x,w) = \frac{1}{1+\exp(w^T x)}$$

$$P(Y=1|x,w) = \frac{\exp(w^T x)}{1+\exp(w^T x)}$$

- **This is equivalent to:**

$$P(Y=y|x,w) = \frac{1}{1+\exp(-y\, w^T x)}$$

- **So we can compute the maximum likelihood estimator:**

$$\widehat{w}_{MLE} = \arg\max_w \prod_{i=1}^n P(y_i|x_i, w)$$

Loss function: Conditional Likelihood

- **Have a bunch of iid data:** $\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, \quad y_i \in \{-1, 1\}$

$$P(Y = y|x, w) = \frac{1}{1 + \exp(-y\, w^T x)}$$

$$\widehat{w}_{MLE} = \arg\max_w \prod_{i=1}^n P(y_i|x_i, w)$$

$$= \arg\min_w \sum_{i=1}^n \log(1 + \exp(-y_i\, x_i^T w))$$

Logistic Loss: $\ell_i(w) = \log(1 + \exp(-y_i\, x_i^T w))$

Squared error Loss: $\ell_i(w) = (y_i - x_i^T w)^2$

(MLE for Gaussian noise)

Loss function: Conditional Likelihood

- **Have a bunch of iid data:** $\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, \quad y_i \in \{-1, 1\}$

$$P(Y = y|x, w) = \frac{1}{1 + \exp(-y\, w^T x)}$$

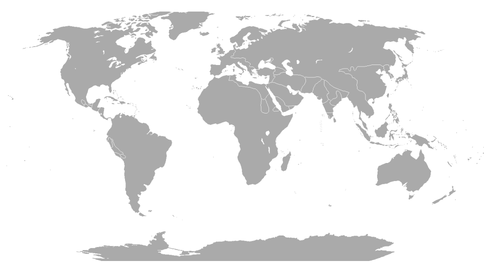
$$\begin{aligned}\widehat{w}_{MLE} &= \arg\max_w \prod_{i=1}^n P(y_i|x_i, w) \\ &= \arg\min_w \sum_{i=1}^n \log(1 + \exp(-y_i\, x_i^T w)) = J(w)\end{aligned}$$

Bad news: no closed-form solution to maximize $J(\mathbf{w})$

How do we encode categorical data *y*?

- so far, we considered Binary case where there are two categories
- encoding y is simple: {+1,-1}

- multi-class classification predicts categorial y
- taking values in $C = \{c_1, \ldots, c_k\}$
- c_j's are called **classes** or **labels**
- examples:



Country of birth
(Argentina, Brazil, USA,...)

Zipcode
(10005, 98195,...)

All English words

- a **k-class classifier** predicts y given x

Embedding c_j's in real values

- for optimization we need to **embed** raw categorical c_j's into real valued vectors
- there are many ways to embed categorial data
  - True->1, False->-1
  - Yes->1, Maybe->0, No->-1
  - Yes->(1,0), Maybe->(0,0), No->(0,1)
  - Apple->(1,0,0), Orange->(0,1,0), Banana->(0,0,1)
  - Ordered sequence:
    (Horse 3, Horse 1, Horse 2) -> (3,1,2)
- we use **one-hot embedding** (a.k.a. **one-hot encoding**)
  - each class is a standard basis vector in $k-$dimension

Country of birth
(Argentina, Brazil, USA,...)

196 categories

x	$h_1(x)$	$h_2(x)$	...	$h_{195}(x)$	$h_{196}(x)$
Brazil		1			
Zimbabwe					1

196 features

Multi-class logistic regression

- data: categorical y in $\{c_1, \ldots, c_k\}$ with k categories

  we use one-hot encoding, s.t. $y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ implies that $y = c_1$

- model: linear vector-function makes a linear prediction $\hat{y} \in \mathbb{R}^k$

$$\hat{y}_i \;=\; f(x_i) \;=\; w^T x_i \quad \in \mathbb{R}^k$$

with model parameter matrix $w \in \mathbb{R}^{d \times k}$ and sample $x_i \in \mathbb{R}^d$

$$f(x_i) \;=\; \begin{bmatrix} f_1(x_i) \\ f_2(x_i) \\ \vdots \\ f_k(x_i) \end{bmatrix} \;=\; \underbrace{\begin{bmatrix} w_{1,0} & w_{1,1} & w_{1,2} & \cdots \\ w_{2,0} & w_{2,1} & w_{2,2} & \cdots \\ \vdots & & & \\ w_{k,0} & w_{k,1} & w_{k,2} & \cdots \end{bmatrix}}_{w^T} \underbrace{\begin{bmatrix} 1 \\ x_i[1] \\ \vdots \\ x_i[d] \end{bmatrix}}_{x_i} \;=\; \begin{bmatrix} w_{1,0} + w_{1,1}x_i[1] + w_{1,2}x_i[2] + \cdots \\ w_{2,0} + w_{2,1}x_i[1] + w_{2,2}x_i[2] + \cdots \\ \vdots \\ w_{k,0} + w_{k,1}x_i[1] + w_{k,2}x_i[2] + \cdots \end{bmatrix}$$

$$w = \begin{bmatrix} w[:,1] & w[:,2] & \cdots & w[:,k] \end{bmatrix}$$

- Logistic regression

2 classes

$$\mathbb{P}(y_i = -1 \,|\, x_i) = \frac{1}{1 + e^{w^T x_i}}$$

$$\mathbb{P}(y_i = +1 \,|\, x_i) = \frac{1}{1 + e^{-w^T x_i}} = \frac{e^{w^T x_i}}{1 + e^{w^T x_i}}$$

k classes

$$\mathbb{P}(y_i = c_1 \,|\, x_i) = \frac{e^{w[:,1]^T x_i}}{e^{w[:,1]^T x_i} + \cdots + e^{w[:,k]^T x_i}}$$

$$\vdots$$

$$\mathbb{P}(y_i = c_k \,|\, x_i) = \frac{e^{w[:,k]^T x_i}}{e^{w[:,1]^T x_i} + \cdots + e^{w[:,k]^T x_i}}$$

Without loss of generality setting w[:,1]=0 when $k = 2$ recovers the original binary class case

Maximum Likelihood Estimator

$$\text{maximize}_w \; \frac{1}{n} \sum_{i=1}^{n} \log(\mathbb{P}(y_i \,|\, x_i))$$

$$\text{maximize}_{w \in \mathbb{R}^d} \; \frac{1}{n} \sum_{i=1}^{n} \log\left(\frac{1}{1 + e^{-y_i w^T x_i}} \right)$$

$$\text{maximize}_{w \in \mathbb{R}^{d \times k}} \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} \mathbf{I}\{y_i = c_j\} \log\left(\frac{e^{w[:,j]^T x_i}}{\sum_{j'=1}^{k} e^{w[:,j']^T x_i}} \right)$$

$\mathbf{I}\{y_i = j\}$ is an indicator that is one only if $y_i = j$

Kernels

Creating Features

- Feature mapping $\phi : \mathbb{R}^d \to \mathbb{R}^p$ maps original data into a rich and high-dimensional feature space (usually $d \ll p$)

For example, in d=1, one can use

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_k(x) \end{bmatrix} = \begin{bmatrix} x \\ x^2 \\ \vdots \\ x^k \end{bmatrix}$$

For example, for d>1, one can generate vectors $\{u_j\}_{j=1}^p \subset \mathbb{R}^d$

and define features:

$$\phi_j(x) = \cos(u_j^T x)$$

$$\phi_j(x) = (u_j^T x)^2$$

$$\phi_j(x) = \frac{1}{1 + \exp(u_j^T x)}$$

Creating Features

- Feature mapping $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^p$ maps original data into a rich and high-dimensional feature space (usually $d \ll p$)

For example, in d=1, one can use

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_k(x) \end{bmatrix} = \begin{bmatrix} x \\ x^2 \\ \vdots \\ x^k \end{bmatrix}$$

For example, for d>1,
one can generate vectors $\{u_j\}_{j=1}^p \subset \mathbb{R}^d$

and define features:

$$\phi_j(x) = \cos(u_j^T x)$$

$$\phi_j(x) = (u_j^T x)^2$$

$$\phi_j(x) = \frac{1}{1 + \exp(u_j^T x)}$$

- How many coefficients/parameters are there for degree-k polynomials for $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$?

How do we deal with high-dimensional lifts/data?

The kernel trick:

A function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *kernel* for a map $\phi : \mathbb{R}^d \to \mathbb{R}^p$ if $K(x, x') = \langle \phi(x), \phi(x') \rangle$ for all x, x'

Big idea: if we can represent our

- training algorithms and
- decision rules for prediction

as functions of dot products of feature maps (i.e. $\{\langle \phi(x), \phi(x') \rangle\}$) and we can find a kernel for our feature map such that

$K(x \,.\, x') = \langle \phi(x), \phi(x') \rangle$

then we can avoid explicitly computing and storing (high-dimensional) $\{\phi(x_i)\}_{i=1}^n$ and instead only work with the kernel matrix of the training data $\{K(x_i, x_j)\}_{i,j \in \{1,\ldots,n\}}$

Recap: Kernels are much more efficient to compute than features

- As illustrating examples, consider polynomial features of degree exactly k
  - $\phi(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ for $k = 1$ and $d = 2$, then $K(x, x') = x_1 x_1' + x_2 x_2'$
  - $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_2 x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$, then $K(x, x') = (x^T x')^2$

Recap: Kernels are much more efficient to compute than features

- As illustrating examples, consider polynomial features of degree exactly k
  - $\phi(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ for $k = 1$ and $d = 2$, then $K(x, x') = x_1 x_1' + x_2 x_2'$
  - $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_2 x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$, then $K(x, x') = (x^T x')^2$
- Note that for a data point x_i, **explicitly** computing the feature $\phi(x_i)$ takes memory/time $p = d^k$
- For a data point x_i, if we can make predictions by only computing the kernel, then computing $\{K(x_i, x_j)\}_{j=1}^n$ takes memory/time dn
  - The features are **implicit** and accessed only via kernels, making it efficient

Examples of popular Kernels

- Polynomials of degree exactly k

$$K(x, x') = (x^T x')^k$$

- Polynomials of degree up to k

$$K(x, x') = (1 + x^T x')^k$$

- Gaussian (squared exponential) kernel
(a.k.a RBF kernel for Radial Basis Function)

$$K(x, x') = \exp\Big(- \frac{\|x - x'\|_2^2}{2\sigma^2}\Big)$$

- Sigmoid

$$K(x, x') = \tanh(\gamma x^T x' + r)$$

- All these kernels are efficient to compute, but the corresponding features are in high-dimensions

Ridge Linear Regression as Kernels

- Recall Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- Consider the trivial kernel $K(x, x') = x^T x'$
- Training: $\widehat{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_{d\times d})^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{y}$
- Prediction: $x_{\text{new}} \in \mathbb{R}^d$

$$\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}} = \mathbf{y}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{X}x_{\text{new}}$$

- Hence, to make prediction on any future data points, all we need to know is
  - $\mathbf{X}x_{\text{new}} = \begin{bmatrix} x_1^T x_{\text{new}} \\ \vdots \\ x_n^T x_{\text{new}} \end{bmatrix} = \begin{bmatrix} K(x_1, x_{\text{new}}) \\ \vdots \\ K(x_n, x_{\text{new}}) \end{bmatrix} \in \mathbb{R}^n$, and $\mathbf{X}\mathbf{X}^T = \begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \cdots \\ \vdots & \vdots & \\ K(x_n, x_1) & K(x_n, x_2) & \cdots \end{bmatrix} \in \mathbb{R}^{n\times n}$
- **Key idea**: Now consider $\widehat{w} = \arg\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} \left(y_i - w^T\phi(x_i)\right)^2 + \lambda\|w\|_2^2$ and use an *any* kernel $K(x, x') = \phi(x)^T\phi(x')$!

The Kernel Trick

- Given data $\{(x_i, y_i)\}_{i=1}^n$, pick a kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

1. For a choice of a loss, use a linear predictor of the form

$$\widehat{w} = \sum_{i=1}^n \alpha_i x_i \quad \text{for some } \alpha = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} \in \mathbb{R}^n \text{ to be learned}$$

Prediction is $\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}} = \sum_{i=1}^n \alpha_i \, x_i^T x_{\text{new}}$

2. Design an algorithm that finds α while accessing the data only via $\{x_i^T x_j\}$
3. Substitute $x_i^T x_j$ with $K(x_i, x_j)$, and find α using the above algorithm from step 2.
4. Make prediction with $\widehat{y}_{\text{new}} = \sum_{i=1}^n \alpha_i K(x_i, x_{\text{new}})$

   (replacing $x_i^T x_{\text{new}}$ with $K(x_i, x_{\text{new}})$)

The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$ (Step 1. We will prove it later)

$$\widehat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

$$\widehat{\alpha}_{\text{kernel}} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

(Step 3. Switch inner product with kernel)

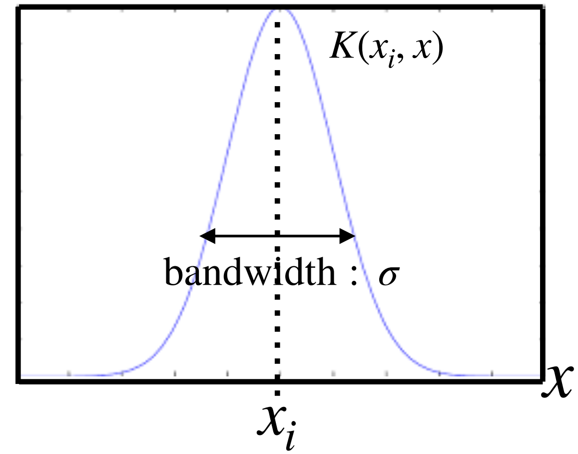
$$= \arg\min_{\alpha} ||\mathbf{y} - \mathbf{K}\alpha||_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

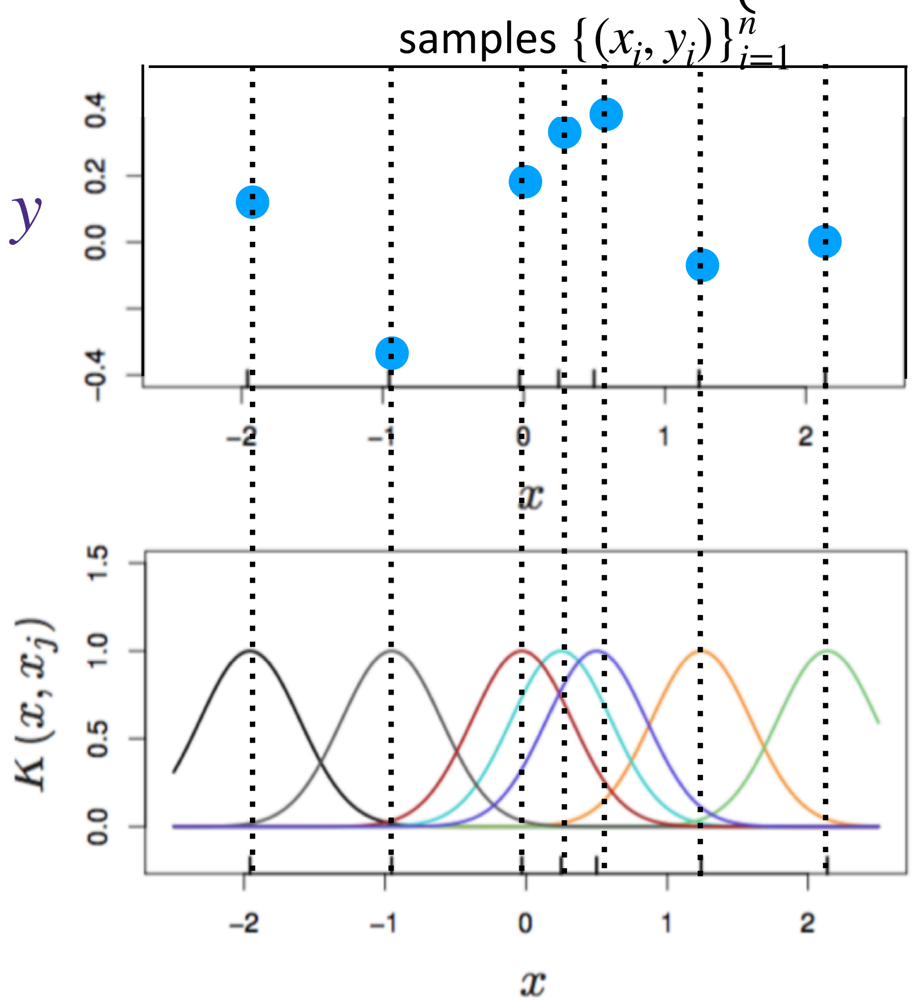
Where $\mathbf{K}_{ij} = K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$

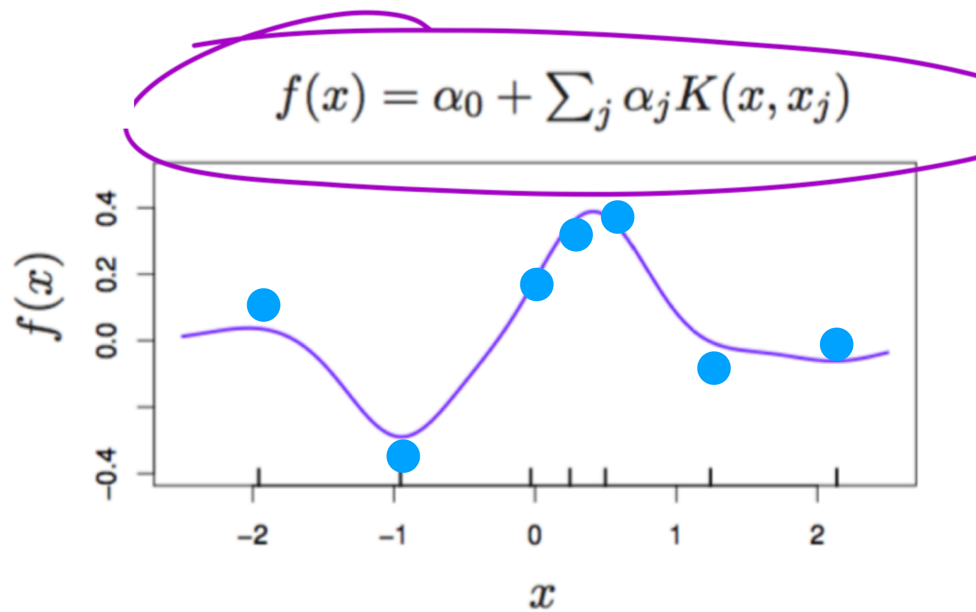
(Solve for $\widehat{\alpha}_{\text{kernel}}$)

Thus, $\widehat{\alpha}_{\text{kernel}} = (\mathbf{K} + \lambda \mathbf{I}_{n \times n})^{-1} \mathbf{y}$

RBF kernel $k(x_i, x) = \exp\left\{ -\frac{\|x_i - x\|_2^2}{2\sigma^2} \right\}$







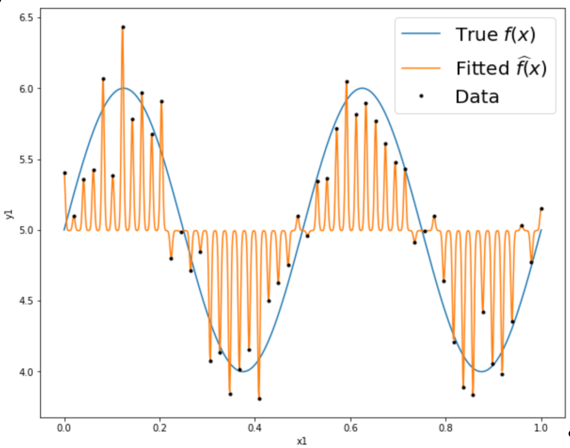
- predictor $f(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x)$ is taking weighted sum of n kernel functions centered at each sample points

RBF kernel $k(x_i, x) = \exp\left\{ -\frac{\|x_i - x\|_2^2}{2\sigma^2} \right\}$

- $\mathcal{L}(\alpha) = \|\mathbf{K}\alpha - \mathbf{y}\|_2^2 + \lambda\|w\|_2^2$
- The bandwidth σ^2 of the kernel regularizes the predictor, and the regularization coefficient λ also regularizes the predictor

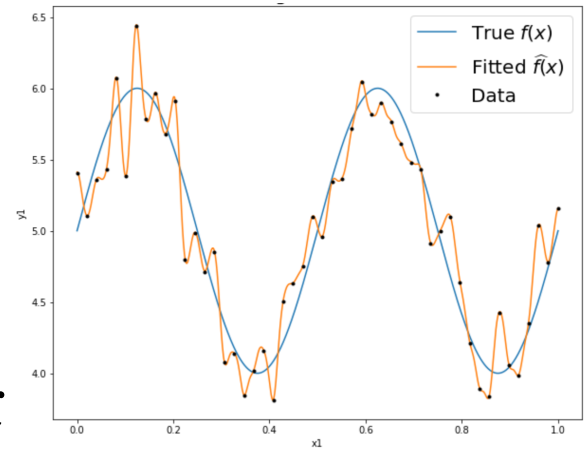
y

$\sigma = 10^{-3}$ $\lambda = 10^{-4}$

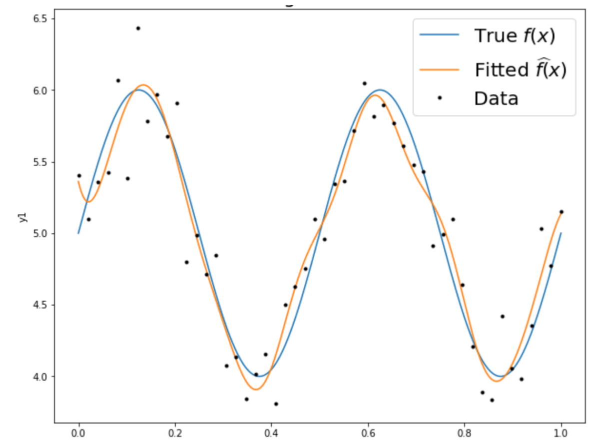


x

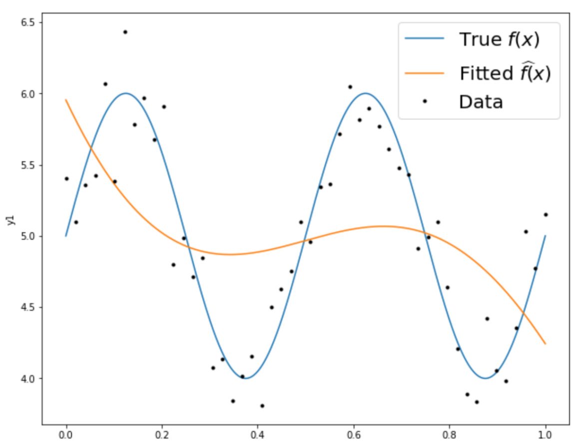
$\sigma = 10^{-2}$ $\lambda = 10^{-4}$



$\sigma = 10^{-1}$ $\lambda = 10^{-4}$

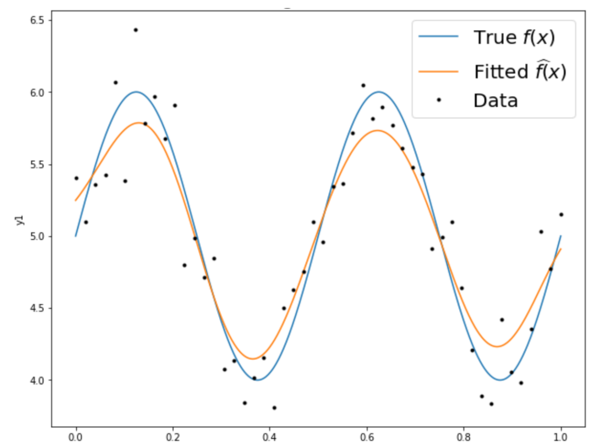


$\sigma = 10^{-0}$ $\lambda = 10^{-4}$



$$\widehat{f}(x) = \sum_{i=1}^{n} \widehat{\alpha}_i K(x_i, x)$$

$\sigma = 10^{-1}$ $\lambda = 10^{-0}$



Fixed Feature V.S. Learned Feature

Can we learn the feature mapping $\phi : \mathbb{R}^d \to \mathbb{R}^p$ from data also?