COL333/671: Introduction to AI

Semester I, 2022-23

Learning - I

Rohan Paul

Outline

- Last Class
    - Reinforcement Learning
- This Class
    - Bayesian Learning, MLE/MAP, Learning in Probabilistic Models.
- Reference Material
    - Please follow the notes as the primary reference on this topic. Supplementary reading on topics covered in class from AIMA Ch 20 sections 20.1 – 20.2.4.

Acknowledgement

These slides are intended for teaching purposes only. Some material has been used/adapted from web sources and from slides by Doina Precup, Dorsa Sadigh, Percy Liang, Mausam, Parag, Emma Brunskill, Alexander Amini, Dan Klein, Anca Dragan, Nicholas Roy and others.

Learning Probabilistic Models

- Models are useful for making optimal decisions.
  - Probabilistic models express a theory about the domain and can be used for decision making.
- How to acquire these models in the first place?
  - Solution: data or experience can be used to build these models
- Key question: how to learn from data?
  - Bayesian view of learning (learning task itself is probabilistic inference)
  - Learning with complete and incomplete data.
  - Essentially, rely on counting.

Example

Suppose there are five kinds of bags of candies:

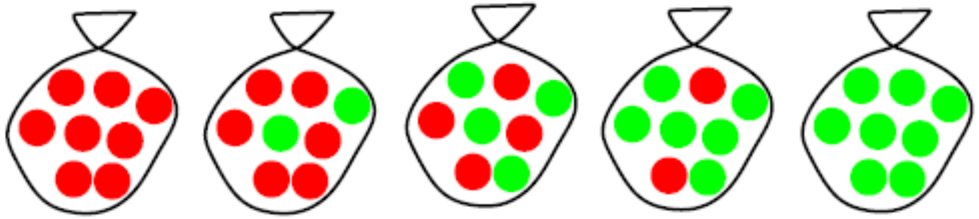
10% are h_1: 100% cherry candies

20% are h_2: 75% cherry candies + 25% lime candies

40% are h_3: 50% cherry candies + 50% lime candies

20% are h_4: 25% cherry candies + 75% lime candies

10% are h_5: 100% lime candies



Then we observe candies drawn from some bag: ● ● ● ● ● ● ● ● ● ●

What kind of bag is it? What flavour will the next candy be?

Statistics

Probability

Bayesian Learning

View learning as Bayesian updating of a probability distribution over the hypothesis space

H is the hypothesis variable, values $h_1, h_2, \ldots$, prior $\mathbf{P}(H)$

jth observation d_j gives the outcome of random variable D_j
training data $\mathbf{d} = d_1, \ldots, d_N$

Given the data so far, each hypothesis has a posterior probability:

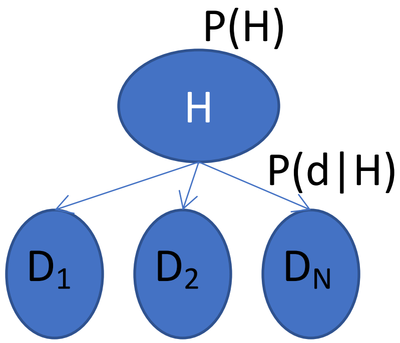
$$P(h_i|\mathbf{d}) = \alpha P(\mathbf{d}|h_i) P(h_i)$$

where $P(\mathbf{d}|h_i)$ is called the likelihood

Predictions use a likelihood-weighted average over the hypotheses:

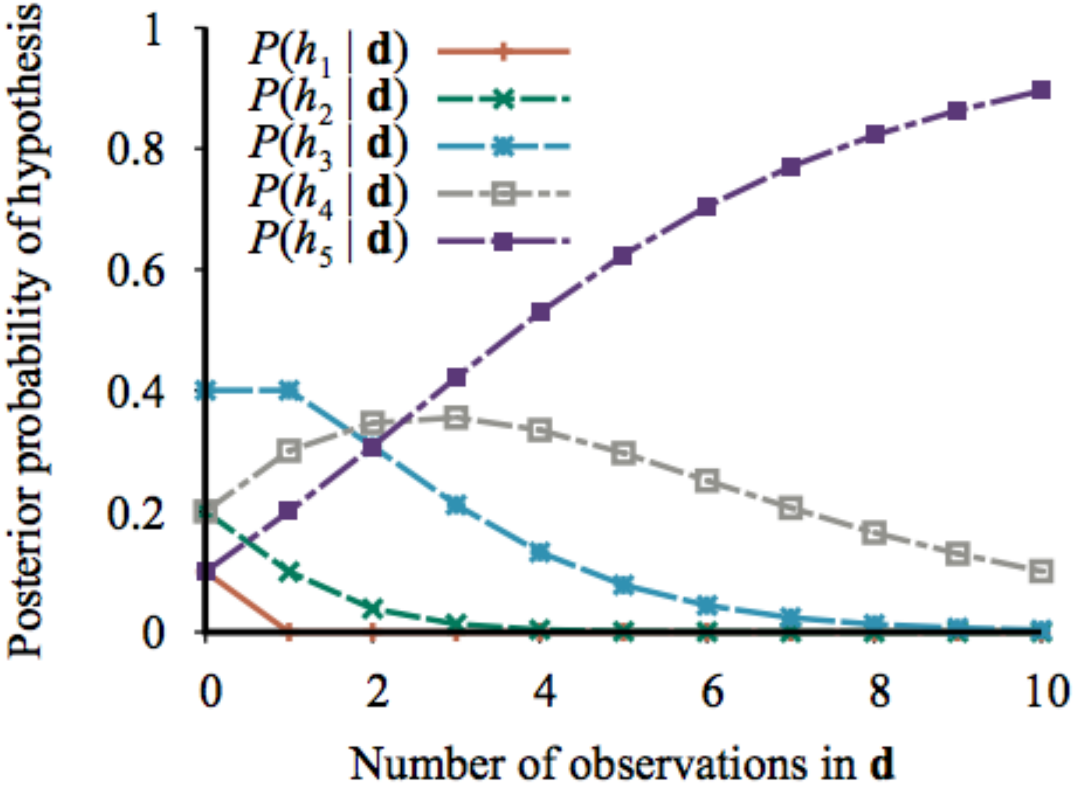
$$\mathbf{P}(X|\mathbf{d}) = \Sigma_i \, \mathbf{P}(X|\mathbf{d}, h_i) P(h_i|\mathbf{d}) = \Sigma_i \, \mathbf{P}(X|h_i) P(h_i|\mathbf{d})$$

No need to pick one best-guess hypothesis!



i.i.d

Posterior Probability of Hypothesis



Probability of a bag of a certain type given observations.

$$P(h_i \,|\, d_1, \ldots, d_N)$$

Bayes Rule

$$P(h_i|\mathbf{d}) = \alpha P(\mathbf{d}\,|\,h_i)P(h_i)$$

IID assumption

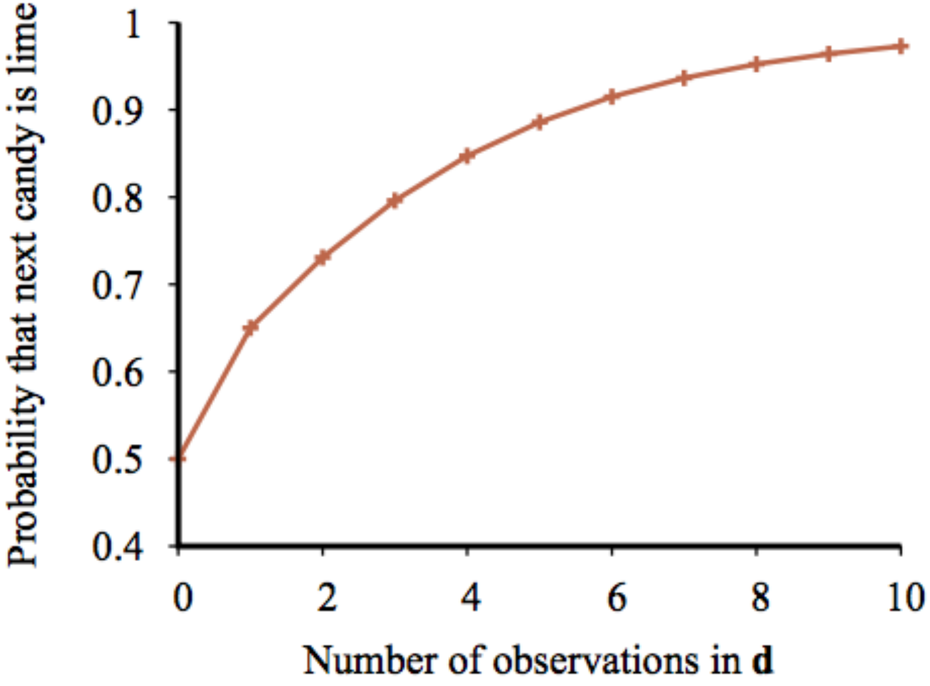
$$P(d\,|\,h_i) = \prod_j P(d_j\,|\,h_i)$$

True hypothesis eventually dominates. Probability of indefinitely producing uncharacteristic data →0

Bayesian Prediction

- Predictions are weighted average over the predictions of the individual hypothesis.
- Bayesian prediction eventually agrees with the true hypothesis.
- For any fixed prior that does not rule out the true hypothesis, the posterior probability of any false hypothesis will eventually vanish.
- Why keep all the hypothesis?
    - Learning from small data, early commitment to a hypothesis is risky, later evidence may lead to a different likely hypothesis.
    - Better accounting of uncertainty in making predictions.
    - Problem: maybe slow and intractable, cannot estimate and marginalize out the hypotheses.

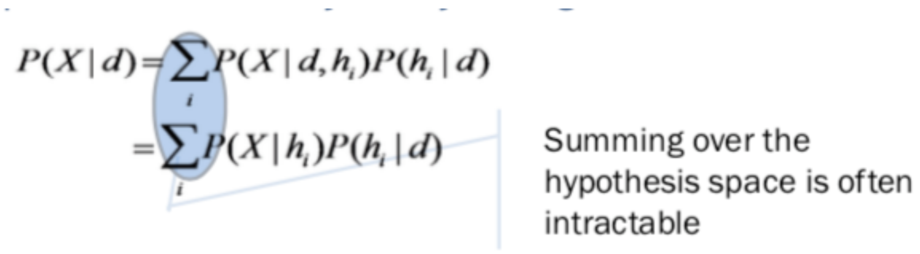
What is the probability that the next candy is of type lime?



Observations

MAP Estimation

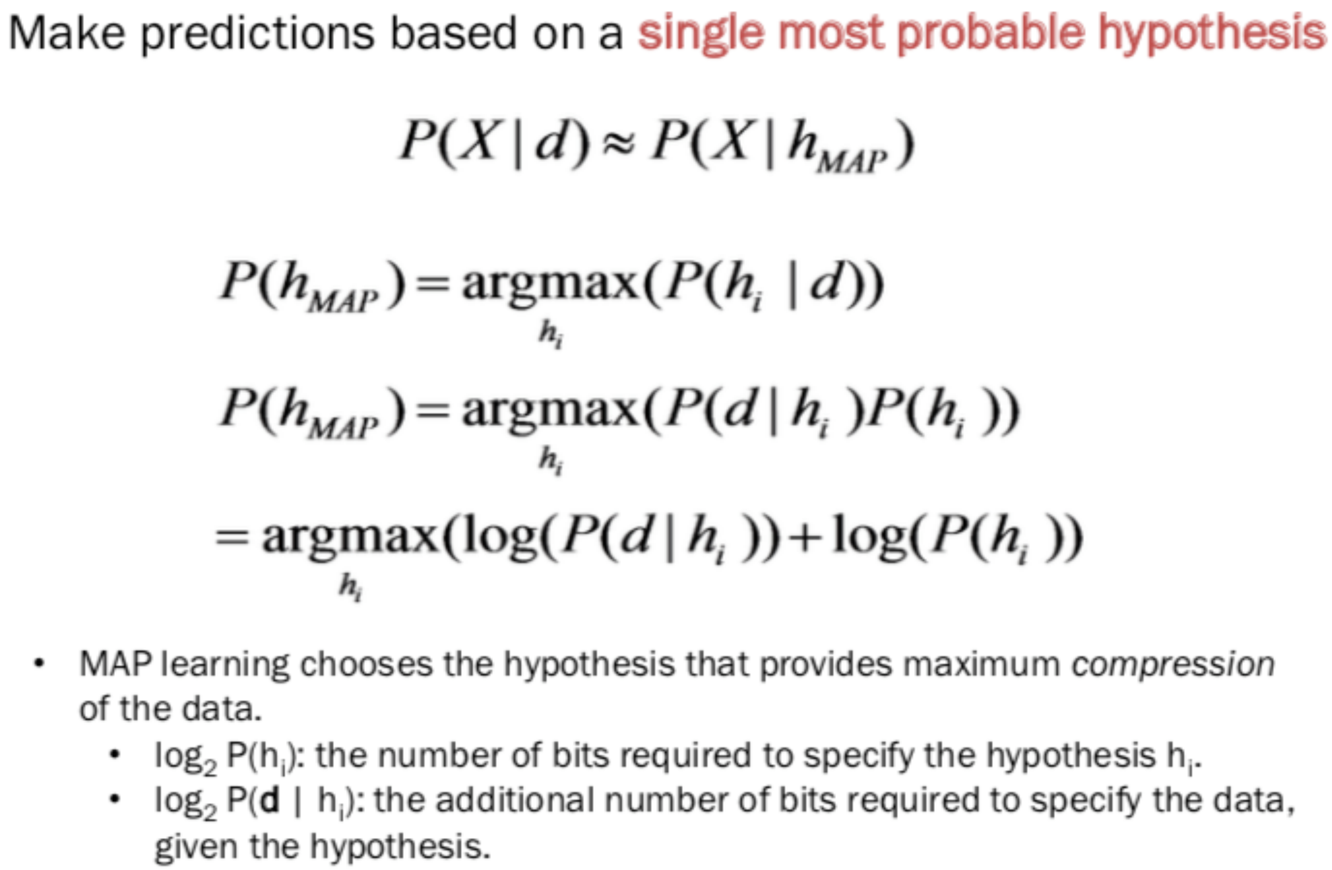
Marginalization over hypothesis may be difficult hence may take only the most probable hypothesis.



$$P(X|d)=\sum_i P(X|d,h_i)P(h_i|d)$$

$$=\sum_i P(X|h_i)P(h_i|d)$$

Summing over the hypothesis space is often intractable



Make predictions based on a single most probable hypothesis

$$P(X|d)\approx P(X|h_{MAP})$$

$$P(h_{MAP})=\underset{h_i}{\operatorname{argmax}}(P(h_i\,|d))$$

$$P(h_{MAP})=\underset{h_i}{\operatorname{argmax}}(P(d\,|h_i\,)P(h_i\,))$$

$$=\underset{h_i}{\operatorname{argmax}}(\log(P(d\,|h_i\,))+\log(P(h_i\,))$$

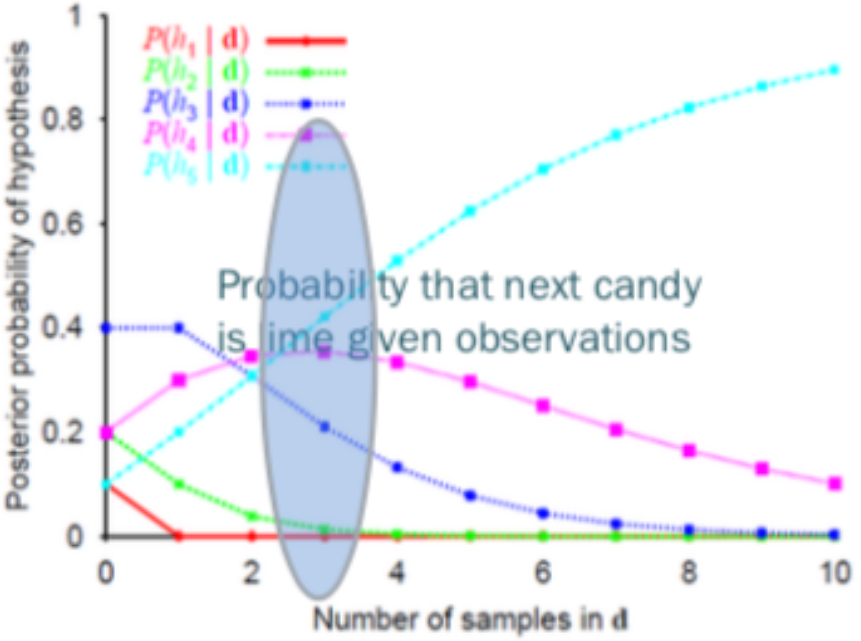
- MAP learning chooses the hypothesis that provides maximum *compression* of the data.
  - $\log_2 P(h_i)$: the number of bits required to specify the hypothesis h_i.
  - $\log_2 P(\mathbf{d}\mid h_i)$: the additional number of bits required to specify the data, given the hypothesis.

MAP Vs. Bayesian Estimation

EX> 🟢 🟢 🟢 After three observations

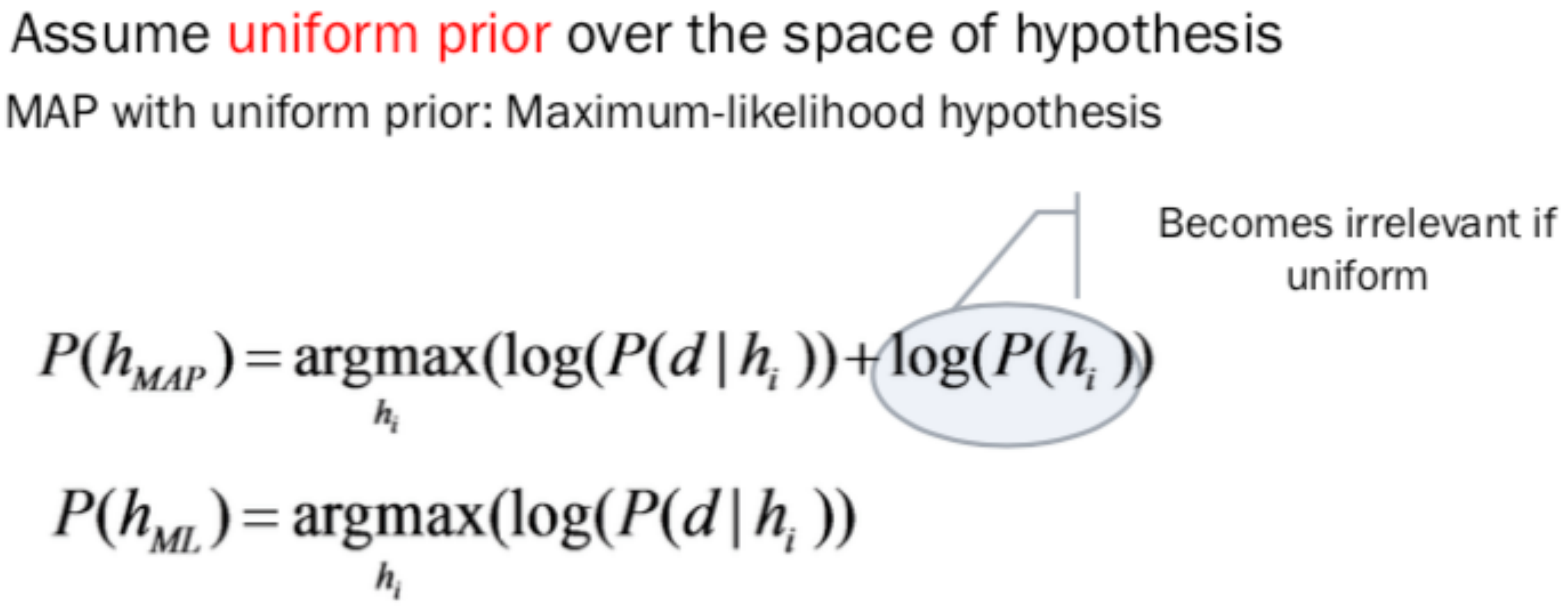
MAP predict with probability 1 that next candy is lime
(pick h5)

Bayes will predict with probability 0.8 that net is lime



Probability that next candy is lime given observations

MLE Estimation



Assume uniform prior over the space of hypothesis

MAP with uniform prior: Maximum-likelihood hypothesis

Becomes irrelevant if uniform

$$P(h_{MAP}) = \operatorname*{argmax}_{h_i}(\log(P(d \mid h_i)) + \log(P(h_i))$$

$$P(h_{ML}) = \operatorname*{argmax}_{h_i}(\log(P(d \mid h_i))$$

Make predictions with the hypothesis that maximizes the data likelihood. Essentially, assuming a uniform prior with no preference of a hypothesis over another.

Maximum Likelihood Estimation (MLE)

- Write the expression for the likelihood of the data given the probabilistic model as characterized by the parameters.
  - Take the log likelihood.
- Optimize the likelihood given the parameters.
  - Write down the derivative of the log likelihood with respect to each parameter.
  - Find the parameters such that the derivatives are zero.

$$\theta_{ML} = \operatorname*{arg\,max}_{\theta} P(\mathbf{X}|\theta)$$

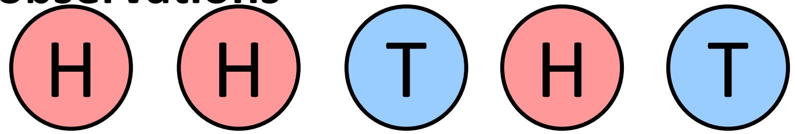
$$= \operatorname*{arg\,max}_{\theta} \prod_{i} P_{\theta}(X_i)$$

$$P_{\text{ML}}(x) = \frac{\text{count}(x)}{\text{total samples}}$$

MLE Example

- Estimating the parameters of a biased coin.
- Setup
  - P(Heads) = θ, P(Tails) = 1-θ
  - Flips are independent and identically distributed according to an unknown distribution.
  - Observe a sequence D of α_H Heads and α_T Tails
  - Learning task: parameter θ.
  - Hypothesis space: Binomial distributions

Data: sequence of coin toss observations



$$D=\{x_i \mid i=1\ldots n\},\ \ P(D\mid\theta)=\Pi_i P(x_i\mid\theta)$$

MLE Steps

- Formulate the objective Function

$$P(\mathcal{D} \mid \theta) = \theta^{\alpha_H}(1-\theta)^{\alpha_T}$$

- MLE of θ that maximizes the probability of D.

$$\widehat{\theta} = \arg\max_{\theta} \quad P(\mathcal{D} \mid \theta)$$

$$\begin{aligned} \widehat{\theta} &= \arg\max_{\theta} \quad \ln P(\mathcal{D} \mid \theta) \\ &= \arg\max_{\theta} \quad \ln \theta^{\alpha_H}(1-\theta)^{\alpha_T} \end{aligned}$$

- Optimize the objective function.

$$\begin{aligned} \frac{d}{d\theta} \ln P(\mathcal{D} \mid \theta) &= \frac{d}{d\theta}\left[\ln \theta^{\alpha_H}(1-\theta)^{\alpha_T}\right] \\ &= \frac{d}{d\theta}\left[\alpha_H \ln \theta + \alpha_T \ln(1-\theta)\right] \\ &= \alpha_H \frac{d}{d\theta} \ln \theta + \alpha_T \frac{d}{d\theta} \ln(1-\theta) \\ &= 0 \end{aligned}$$

$$\boxed{\widehat{\theta}_{MLE} = \frac{\alpha_H}{\alpha_H + \alpha_T}}$$

Maximum a posteriori (MAP) Estimates

- **Maximum likelihood estimate (MLE)**
  - Estimates the parameters that maximizes the data likelihood.
  - Relative counts give MLE estimates

$$\theta_{ML} = \arg\max_{\theta} P(\mathbf{X}|\theta)$$

$$= \arg\max_{\theta} \prod_{i} P_{\theta}(X_i)$$

- **Maximum a posteriori estimate (MAP)**
  - Bayesian parameter estimation
  - Encodes a prior over the parameters (not all parameters are equal prior values).
  - Combines the prior and the likelihood while estimating the parameters.

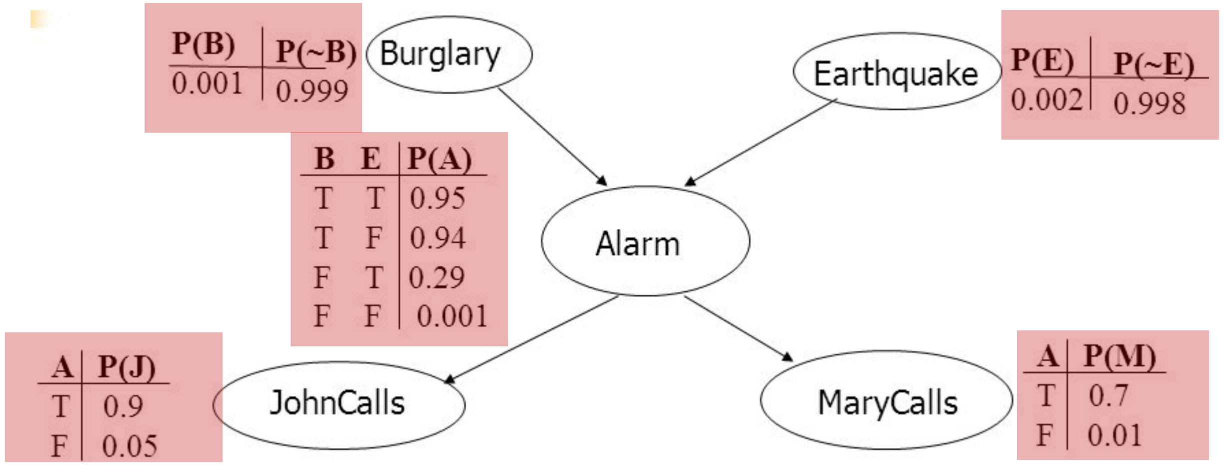
$$\theta_{MAP} = \arg\max_{\theta} P(\theta|\mathbf{X})$$

$$= \arg\max_{\theta} P(\mathbf{X}|\theta)P(\theta)/P(\mathbf{X})$$

$$= \arg\max_{\theta} P(\mathbf{X}|\theta)P(\theta)$$

Learning Parameters for a Probability Model

- Probabilistic models require parameters (numbers in the conditional probability tables).
- We need these values to make predictions.
- Can we learn these from data (i.e., samples from the Bayes Net)?
- How to do this? Counting and averaging.



P(B)	P(~B)
0.001	0.999

P(E)	P(~E)
0.002	0.998

B	E	P(A)
T	T	0.95
T	F	0.94
F	T	0.29
F	F	0.001

A	P(J)
T	0.9
F	0.05

A	P(M)
T	0.7
F	0.01

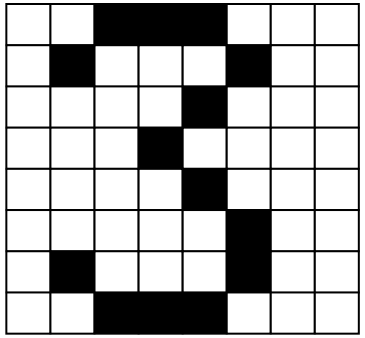
Can we use samples to estimate the values in the tables?

Learning Parameters for a Probability Model

Classification Problem

- Task: given inputs x, predict labels (classes) y
- Examples:
  - Spam detection (input: document, classes: spam / ham)
  - OCR (input: images, classes: characters)
  - Medical diagnosis (input: symptoms, classes: diseases)
  - Fraud detection (input: account activity, classes: fraud / no fraud)

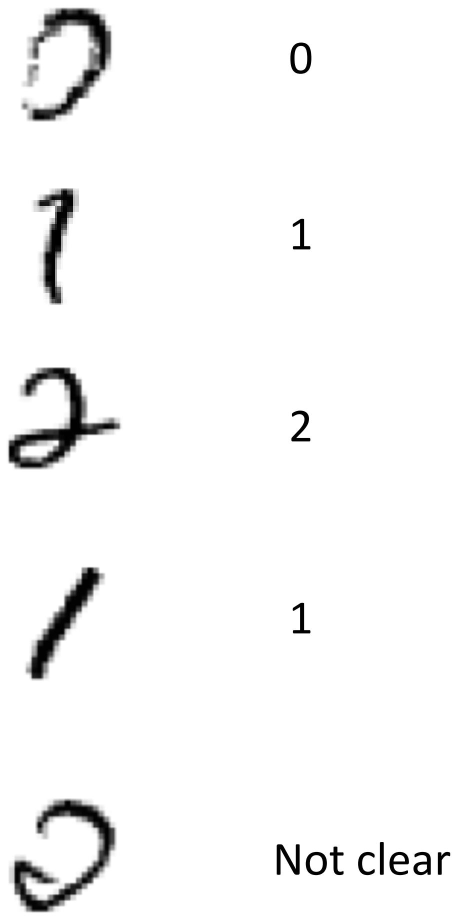
Bayes Net for Classification



- Input: images / pixel grids
- Output: a digit 0-9

- Setup:
  - Get a large collection of example images, each labeled with a digit
  - Note: someone has to hand label all this data!
  - Want to learn to predict labels of new, future digit images

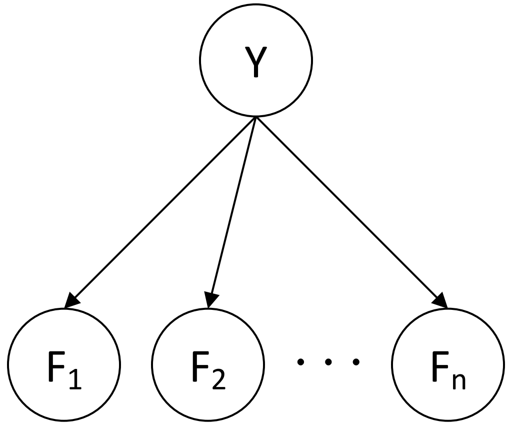
- Features: The attributes used to make the digit decision
  - Pixels: (6,8)=ON
  - Shape Patterns: NumComponents, AspectRatio, NumLoops
  - …



Bayes Net for Classification

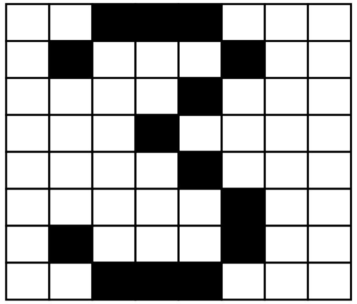
- Naïve Bayes: Assume all features are independent effects of the label

- Simple digit recognition:
  - One feature (variable) F_{ij} for each grid position <i,j>
  - Feature values are on / off, based on whether intensity is more or less than 0.5 in underlying image
  - Each input maps to a feature vector, e.g.



$\rightarrow \langle F_{0,0} = 0 \;\; F_{0,1} = 0 \;\; F_{0,2} = 1 \;\; F_{0,3} = 1 \;\; F_{0,4} = 0 \;\; \ldots F_{15,15} = 0 \rangle$

$$P(Y|F_{0,0} \ldots F_{15,15}) \propto P(Y) \prod_{i,j} P(F_{i,j}|Y)$$

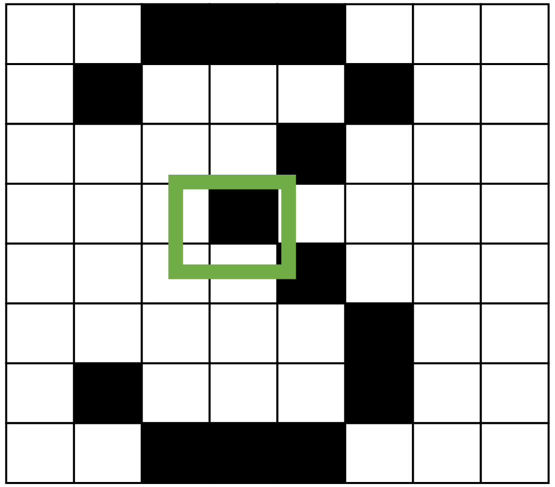


Parameter Estimation

- Need the estimates of local conditional probability tables.
  - P(Y), the prior over labels
  - P(F_i|Y) for each feature (evidence variable)
  - These probabilities are collectively called the ***parameters*** of the model and denoted by θ
    - Till now, the table values were provided.
    - Now, use data to acquire these values.

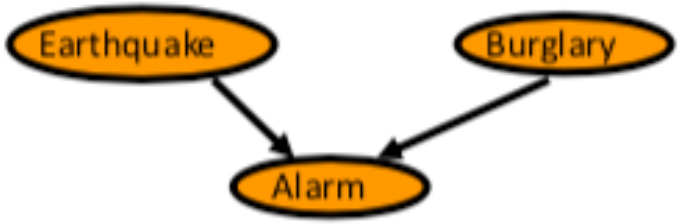
Parameter Estimation

- P(Y) – how frequent is the class-type for digit 3?
  - If you take a sample of images of numbers how frequent is this number
- $P(F_i|Y)$ – for digit 3 what fraction of the time the cell is on?
  - Conditioned on the class type how frequent is the feature
- Use **relative frequencies** from the data to estimate these values.



Parameter Estimation: Complete Data

E	B	A	#
0	0	0	1000
0	0	1	10
0	1	0	20
0	1	1	100
1	0	0	200
1	0	1	50
1	1	0	0
1	1	1	5



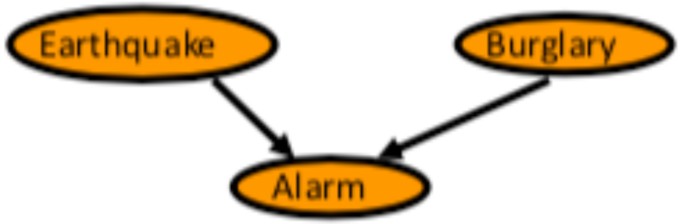
Note: The data is “complete”. Each data point had values observed for “all” the variables in the model.

	Pr(A|E,B)
e,b	
$e,\bar{b}$	
$\bar{e},b$	
$\bar{e},\bar{b}$	

$$P(a|\bar{e}, \bar{b}) = ?$$

$$= 10/1010$$

Parameter Estimation



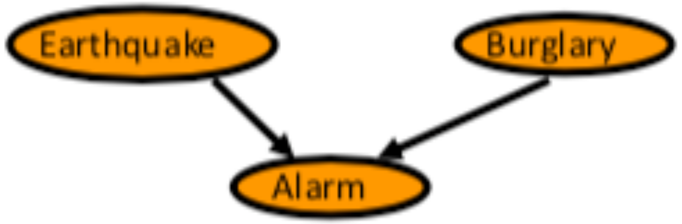
E	B	A	#
0	0	0	1000
0	0	1	10
0	1	0	20
0	1	1	100
1	0	0	200
1	0	1	50
1	1	0	0
1	1	1	5

	Pr(A|E,B)
e,b	
$e,\bar{b}$	
$\bar{e},b$	
$\bar{e},\bar{b}$	~0.01

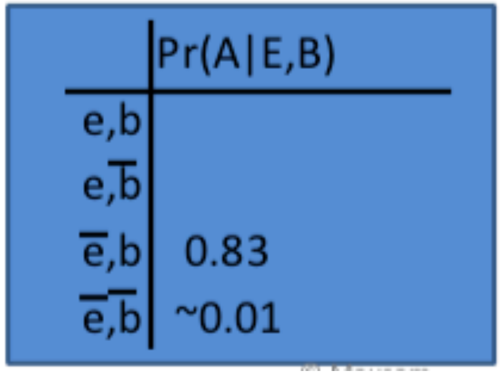
$P(a|\bar{e}, b) = ?$

$= 100/120$

Parameter Estimation



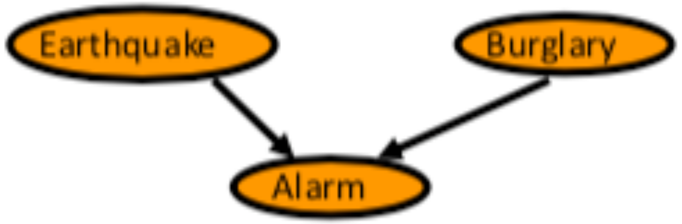
E	B	A	#
0	0	0	1000
0	0	1	10
0	1	0	20
0	1	1	100
1	0	0	200
1	0	1	50
1	1	0	0
1	1	1	5



	Pr(A|E,B)
e,b	
$e,\bar{b}$	
$\bar{e},b$	0.83
$\bar{e},\bar{b}$	~0.01

$$P(a|e,\bar{b}) = ?$$
$$= 50/250$$

Parameter Estimation

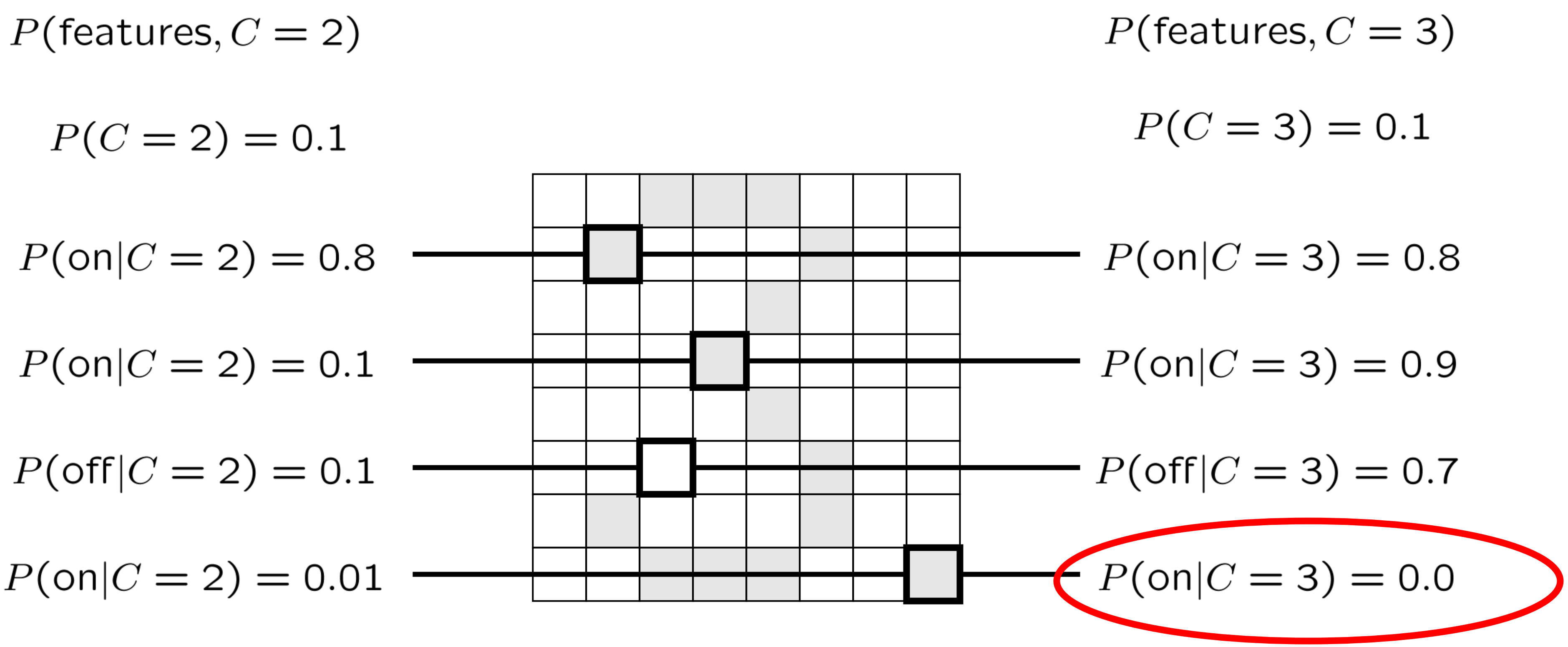


E	B	A	#
0	0	0	1000
0	0	1	10
0	1	0	20
0	1	1	100
1	0	0	200
1	0	1	50
1	1	0	0
1	1	1	5

	Pr(A|E,B)
e,b	
$e,\bar{b}$	0.2
$\bar{e},b$	0.83
$\bar{e},\bar{b}$	~0.01

P(a|e, b) = ?
= 5/5

Problem: values not seen in the training data



$P(\text{features}, C = 2)$

$P(C = 2) = 0.1$

$P(\text{on}|C = 2) = 0.8$

$P(\text{on}|C = 2) = 0.1$

$P(\text{off}|C = 2) = 0.1$

$P(\text{on}|C = 2) = 0.01$

$P(\text{features}, C = 3)$

$P(C = 3) = 0.1$

$P(\text{on}|C = 3) = 0.8$

$P(\text{on}|C = 3) = 0.9$

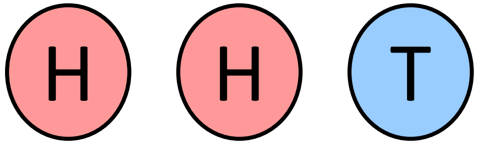
$P(\text{off}|C = 3) = 0.7$

$P(\text{on}|C = 3) = 0.0$

If one feature was not seen in the training data, the likelihood goes to zero. If we did not see this feature in the training data, does not mean we will not see this in training. Essentially overfitting to the training data set.

Laplace Smoothing

- Pretend that every outcome occurs **once more** than it is actually observed.



- If certain counts are **not** seen in training does not mean that they have zero probability of occurring in future.

$$P_{LAP}(x) = \frac{c(x)+1}{\sum_x [c(x)+1]} = \frac{c(x)+1}{N+|X|}$$

- Another version of Laplace smoothing
  - instead of 1, add **k times**
  - k is an adjustable parameter.

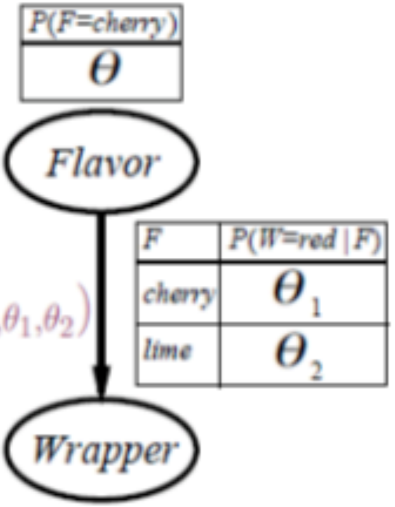
$$P_{LAP,k}(x) = \frac{c(x)+k}{N+k|X|}$$

- Essentially, encodes a **prior** (pseudo-counts).

Learning Multiple Parameters

- Estimate latent parameters using MLE.
- There are two CPTs in this example.
- Observations are of both variables: Flavor and Wrapper.
- Take log likelihood.

Red/green wrapper depends probabilistically on flavor:



Likelihood for, e.g., cherry candy in green wrapper:

$$\begin{aligned} &P(F=cherry, W=green|h_{\theta,\theta_1,\theta_2}) \\ &= P(F=cherry|h_{\theta,\theta_1,\theta_2})P(W=green|F=cherry, h_{\theta,\theta_1,\theta_2}) \\ &= \theta \cdot (1-\theta_1) \end{aligned}$$

N candies, r_c red-wrapped cherry candies, etc.:

$$P(\mathbf{d}|h_{\theta,\theta_1,\theta_2}) = \theta^c(1-\theta)^\ell \cdot \theta_1^{r_c}(1-\theta_1)^{g_c} \cdot \theta_2^{r_\ell}(1-\theta_2)^{g_\ell}$$

N candies unwrapped, c are cherries and *l* are limes

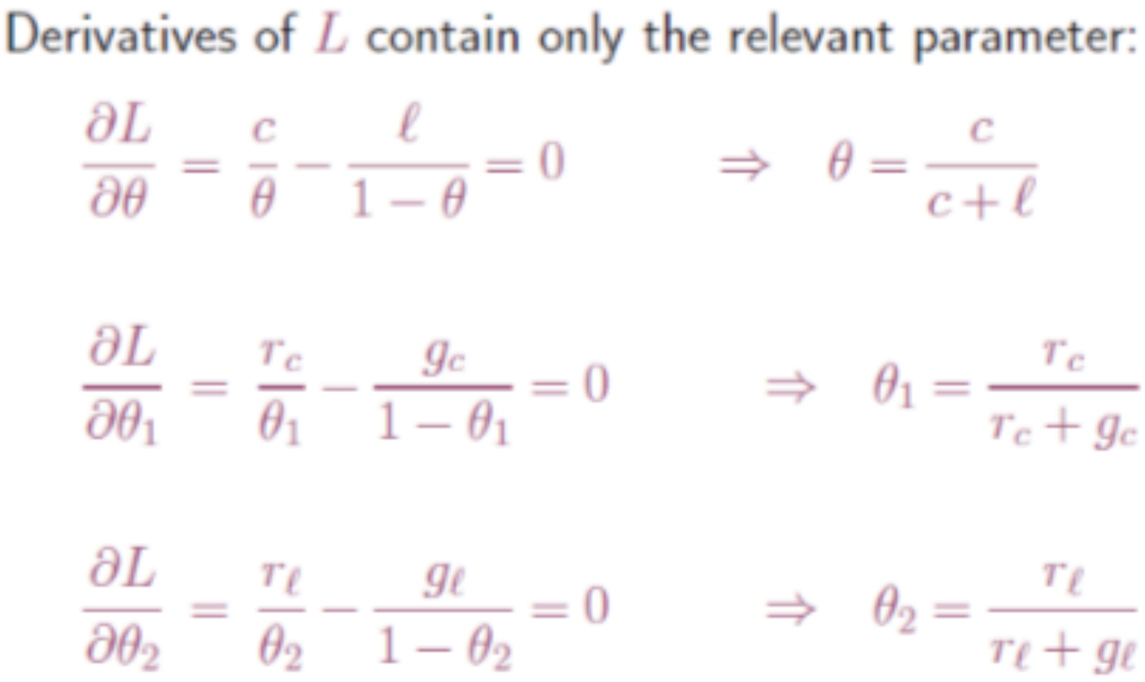
Take logarithm

$$\begin{aligned} L = &\ [c\log\theta + \ell\log(1-\theta)] \\ &+ [r_c\log\theta_1 + g_c\log(1-\theta_1)] \\ &+ [r_\ell\log\theta_2 + g_\ell\log(1-\theta_2)] \end{aligned}$$

With complete data, the ML parameter learning problem for a Bayesian network decomposes into separate learning problems, one for each parameter

Learning Multiple Parameters

- Minimize data likelihood to estimate the parameters.



Derivatives of L contain only the relevant parameter:

$$\frac{\partial L}{\partial \theta} = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 \qquad \Rightarrow \quad \theta = \frac{c}{c+\ell}$$

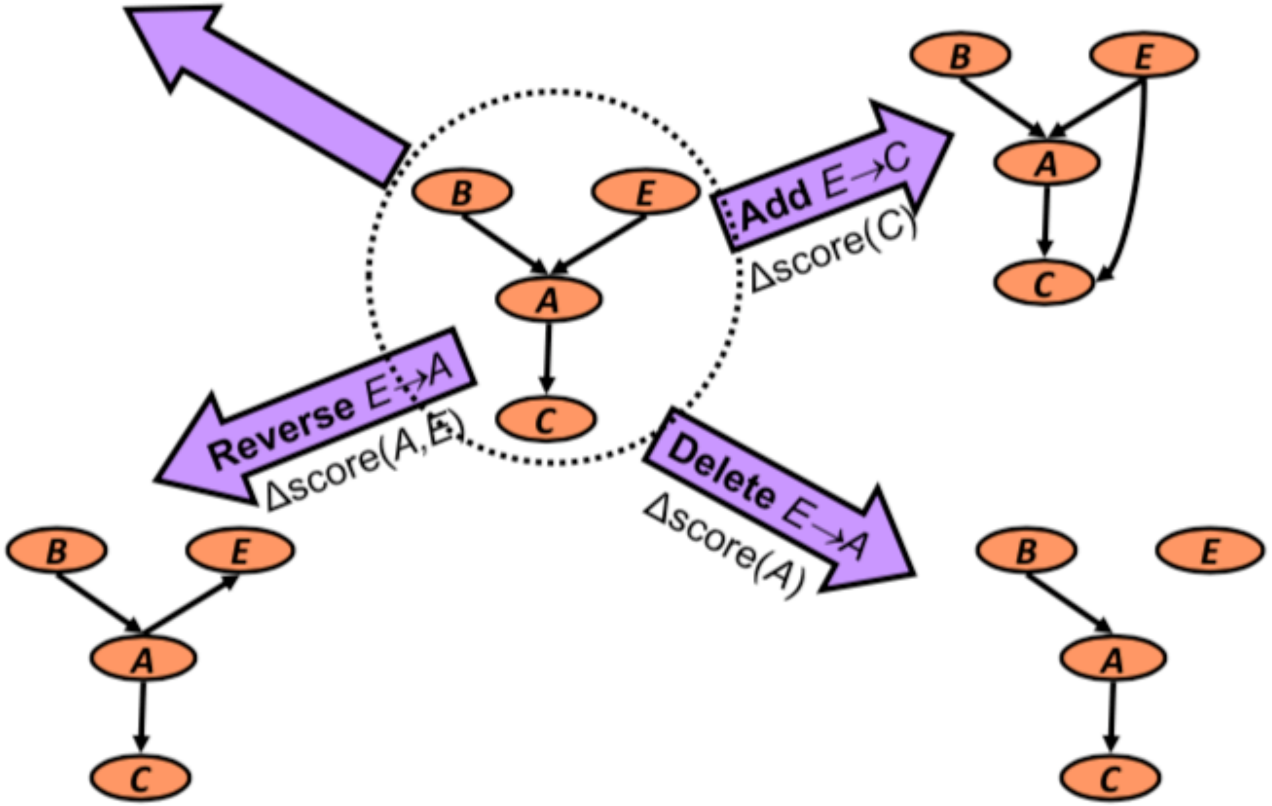
$$\frac{\partial L}{\partial \theta_1} = \frac{r_c}{\theta_1} - \frac{g_c}{1-\theta_1} = 0 \qquad \Rightarrow \quad \theta_1 = \frac{r_c}{r_c+g_c}$$

$$\frac{\partial L}{\partial \theta_2} = \frac{r_\ell}{\theta_2} - \frac{g_\ell}{1-\theta_2} = 0 \qquad \Rightarrow \quad \theta_2 = \frac{r_\ell}{r_\ell+g_\ell}$$

Maximum Likelihood Parameter Learning with complete data for a Bayes Net decomposes into separate learning problems, one for each parameter.

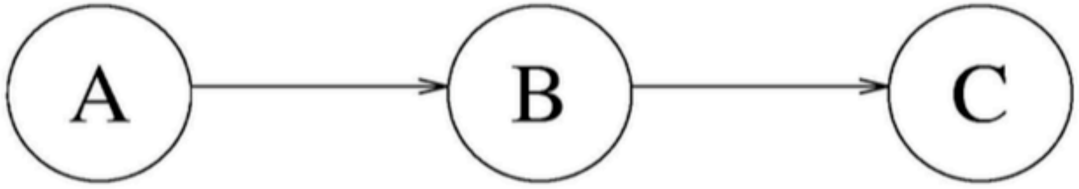
How to learn the structure of the Bayes Net?

- Problem: Estimate/learn the structure of the model
- Setup a search process (like local search, hill climbing etc.)
- For each structure, learn the parameters.
- How to score a solution?
    - Use Max. likelihood estimation.
    - Penalize complexity of the structure (don’t want a fully connected model).
    - Additionally check for validity of the conditional independences.



Parameter Learning when some variables are not observed

- If we knew the missing value for B. Then we can estimate the CPTs.
- If we knew the CPTs then we can infer the probability of the missing value of B.
- It is a *chicken and egg* problem.



Examples:	0	1	1
	1	0	0
	1	1	1
	1	?	0

Initialization: $P(B|A) =$ $P(C|B) =$

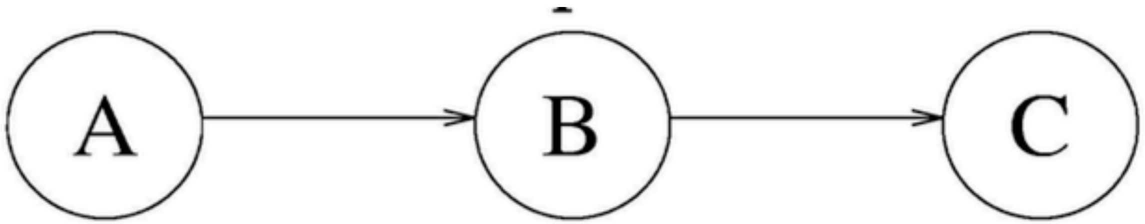
$P(A) =$ $P(B|\neg A) =$ $P(C|\neg B) =$

Data is incomplete. One sample has (A = 1, **B= ?** and C = 0)

Expectation Maximization

- Initialization
  - Initialize CPT parameter values (ignoring missing information)
- Expectation
  - Compute expected values of unobserved variables assuming current parameters values.
  - Involves BayesNet inference (exact or approximate)
- Maximization
  - Compute new parameters (of the CPTs) to maximize the probability of data (observed and estimated)
- Alternate the EM steps until convergence. Convergence is guaranteed.

Expectation Maximization



Examples:	0	1	1
	1	0	0
	1	1	1
	1	?	0

Initialization: $P(B|A) = 0$ $P(C|B) = 0$

$P(A) = 0.75$ $P(B|\neg A) = 0$ $P(C|\neg B) = 0$

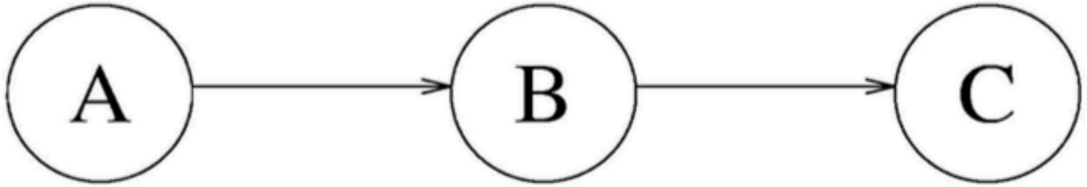
E-step: $P(? = 1) = P(B|A, \neg C) = \frac{P(A,B,\neg C)}{P(A,\neg C)} = \ldots = 0$

M-step: $P(B|A) =$ $P(C|B) =$

$P(A) =$ $P(B|\neg A) =$ $P(C|\neg B) =$

E-step: $P(? = 1) =$

Expectation Maximization



Examples:	0	1	1
	1	0	0
	1	1	1
	1	0	0

Initialization: $P(B|A) = 0$ $P(C|B) = 0$

$P(A) = 0.75$ $P(B|\neg A) = 0$ $P(C|\neg B) = 0$

E-step: $P(? = 1) = P(B|A, \neg C) = \frac{P(A,B,\neg C)}{P(A,\neg C)} = \ldots = 0$

M-step: $P(B|A) = 0.33$ $P(C|B) = 1$

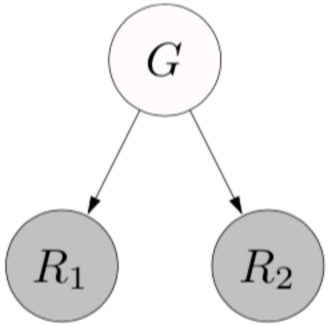
$P(A) = 0.75$ $P(B|\neg A) = 1$ $P(C|\neg B) = 0$

E-step: $P(? = 1) =$

Parameter Learning with Missing Data

Consider a problem of inferring the genre of a movie (Comedy or Drama) from the ratings given by two film reviewers R1 and R2.

Setting where only the ratings are observed.



What if we **don't observe** some of the variables?

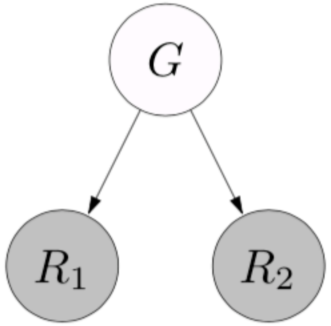
$$\mathcal{D}_{\text{train}} = \{(?,4,5),(?,4,4),(?,5,3),(?,1,2),(?,5,4)\}$$

Slide adapted from Dorsa Sadigh and Percy Liang

Maximum Marginal Likelihood

Variables: H is hidden, $E = e$ is observed

Example:



$H = G \quad E = (R_1, R_2) \quad e = (1, 2)$

$\theta = (p_G, p_R)$

Marginalize over the latent variables in the likelihood

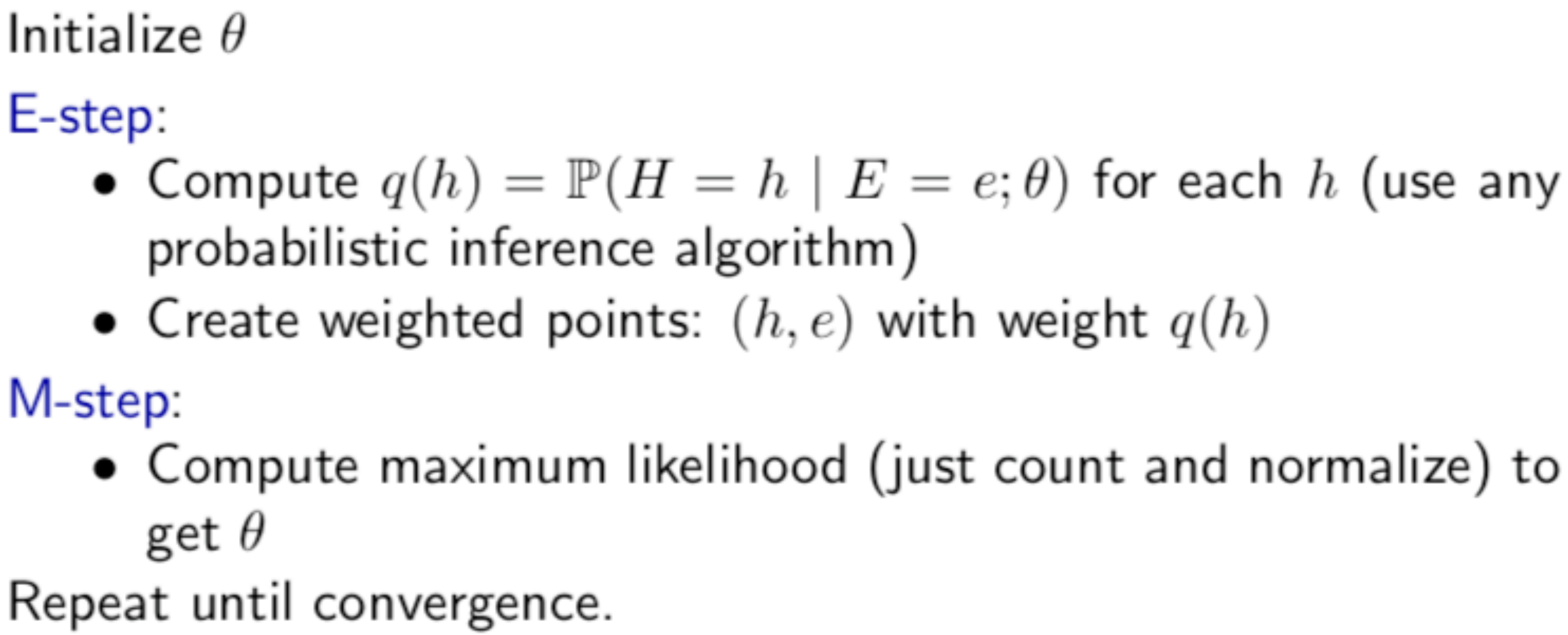
Maximum marginal likelihood objective:

$$\max_{\theta} \prod_{e \in \mathcal{D}_{\text{train}}} \mathbb{P}(E = e; \theta)$$

$$= \max_{\theta} \prod_{e \in \mathcal{D}_{\text{train}}} \sum_{h} \mathbb{P}(H = h, E = e; \theta)$$

Slide adapted from Dorsa Sadigh and Percy Liang

Expectation Maximization



Initialize θ

E-step:

- Compute $q(h) = \mathbb{P}(H = h \mid E = e; \theta)$ for each h (use any probabilistic inference algorithm)
- Create weighted points: (h, e) with weight $q(h)$

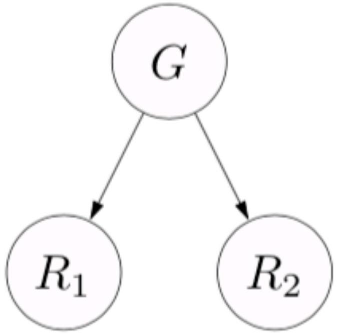
M-step:

- Compute maximum likelihood (just count and normalize) to get θ

Repeat until convergence.

Slide adapted from Dorsa Sadigh and Percy Liang

Expectation Maximization



$$\mathcal{D}_{\text{train}} = \{(?, 2, 2), (?, 1, 2)\}$$

θ:

g	$p_G(g)$
c	0.5
d	0.5

g	r	$p_R(r \mid g)$
c	1	0.4
c	2	0.6
d	1	0.6
d	2	0.4

E-step

(r_1, r_2)	g	$\mathbb{P}(G = g, R_1 = r_1, R_2 = r_2)$	$q(g)$
(2, 2)	c	$0.5 \cdot 0.6 \cdot 0.6 = 0.18$	$\frac{0.18}{0.18+0.08} = 0.69$
(2, 2)	d	$0.5 \cdot 0.4 \cdot 0.4 = 0.08$	$\frac{0.08}{0.18+0.08} = 0.31$
(1, 2)	c	$0.5 \cdot 0.4 \cdot 0.6 = 0.12$	$\frac{0.12}{0.12+0.12} = 0.5$
(1, 2)	d	$0.5 \cdot 0.6 \cdot 0.4 = 0.12$	$\frac{0.12}{0.12+0.12} = 0.5$

Estimated Fractional samples

(g=c, r_1=2, r_2=2)	prob: 0.69
(g=d, r_1=2, r_2=2)	prob: 0.31
(g=c, r_1=1, r_2=2)	prob: 0.5
(g=d, r_1=1, r_2=2)	prob: 0.5

M-step

θ:

g	count	$p_G(g)$
c	0.69 + 0.5	0.59
d	0.31 + 0.5	0.41

g	r	count	$p_R(r \mid g)$
c	1	0.5	0.21
c	2	0.5 + 0.69 + 0.69	0.79
d	1	0.5	0.31
d	2	0.5 + 0.31 + 0.31	0.69

Revising probabilities based on fractional samples.

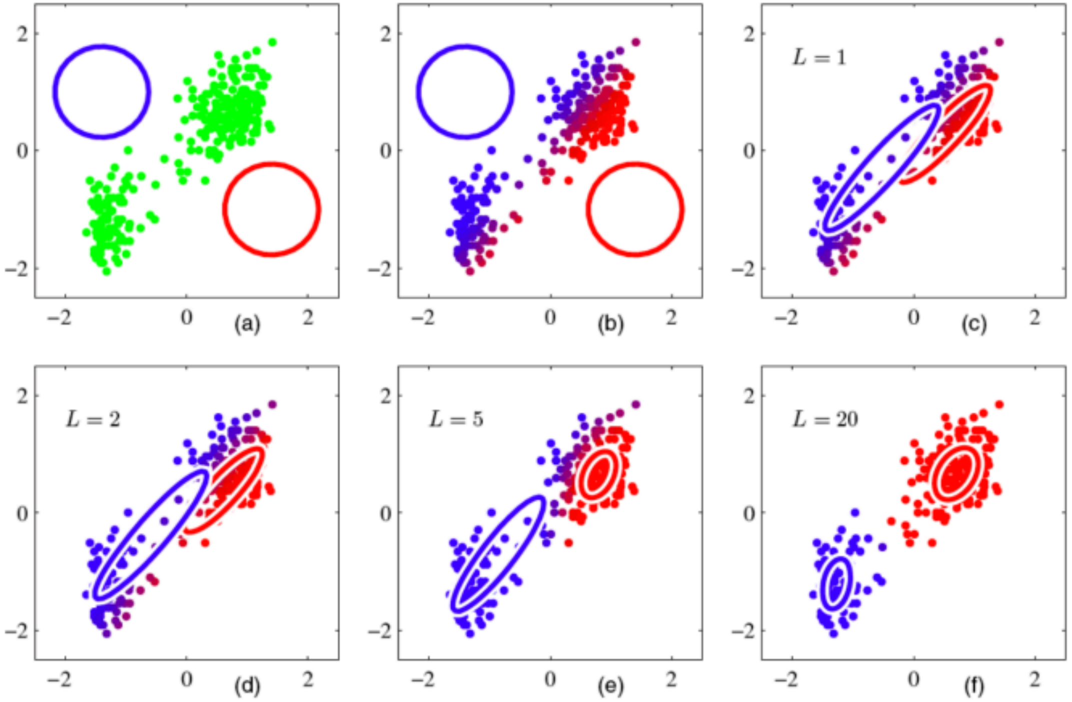
The CPTs for the two reviewers is the same.

Slide adapted from Dorsa Sadigh and Percy Liang

EM in Continuous Space: Gaussian Mixture Modeling

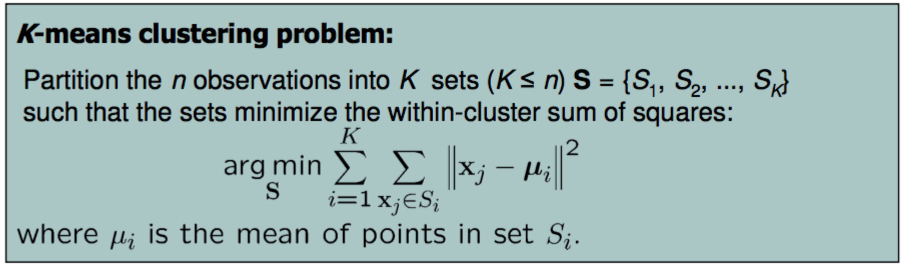
- Problem: clustering task where we want to discern multiple category in a collection of given points.
- Assume a mixture of components (Gaussian)
- Don’t know which data point comes from which component.
- Use EM to iteratively determine the assignments and the parameters of the Gaussian components.

$$P(x) = \sum_{i=1}^{k} P(C=i)P(x|C=i)$$



Web link: https://lukapopijac.github.io/gaussian-mixture-model/

Closely related: K-Means Clustering



***K*-means clustering problem:**

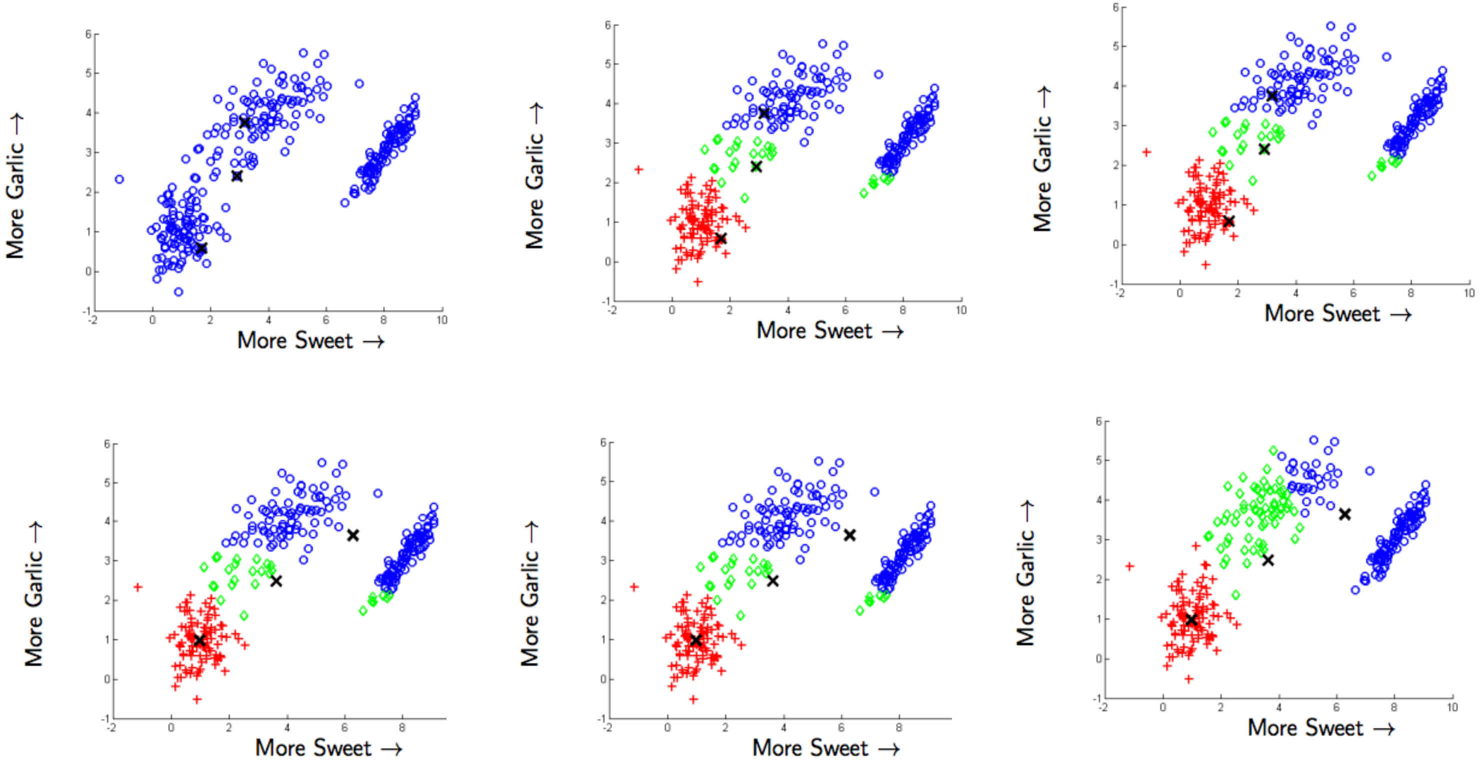
Partition the n observations into K sets ($K \le n$) **S** = {S_1, S_2, ..., S_K} such that the sets minimize the within-cluster sum of squares:

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{K} \sum_{\mathbf{x}_j \in S_i} \left\| \mathbf{x}_j - \boldsymbol{\mu}_i \right\|^2$$

where μ_i is the mean of points in set S_i.

A GMM yields a probability distribution over the cluster assignment for each point; whereas K-Means gives a single hard assignment

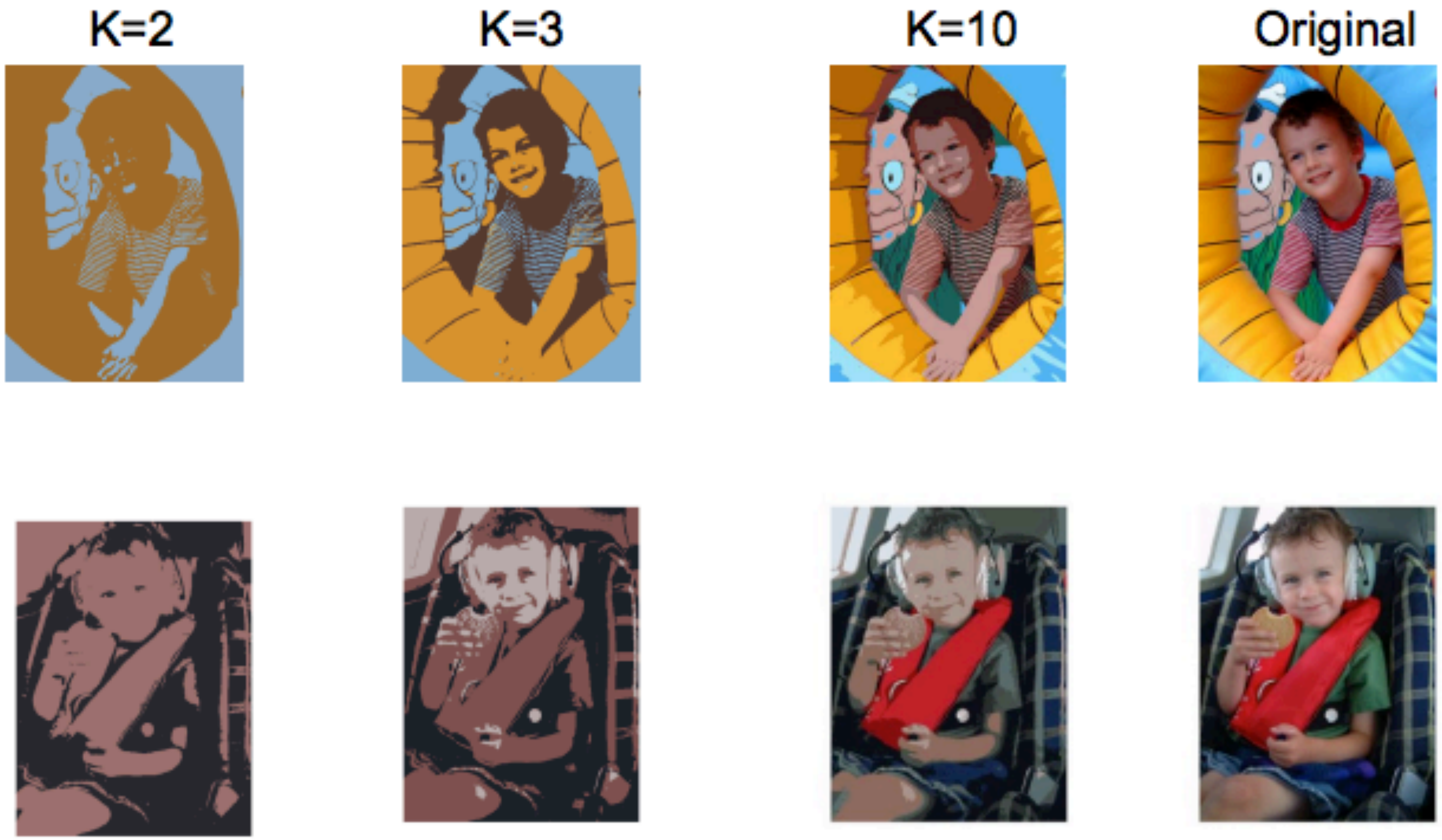
- How many different sauces should the company make?
- How sweet/garlicy should these sauces be?
- Idea: We will segment the consumers into groups (in this case 3), we will then find the best sauce for each group



K-Means: Application

Goal of Segmentation is to partition an image into regions each of which has reasonably homogenous visual appearance.

Apply K-Means in the colour space.

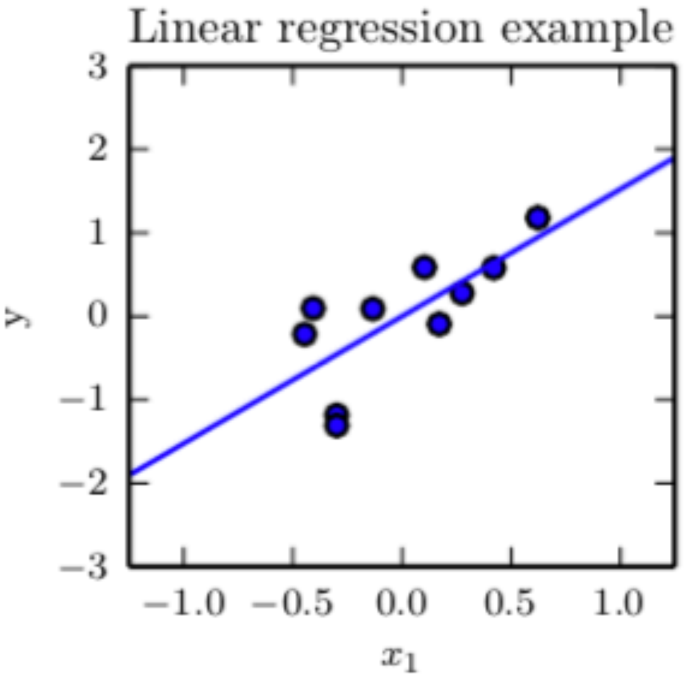


Learning a Best Fit Hypothesis: Linear Regression Task

Linear regression (no bias) $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}$

Linear regression (with bias) $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x} + b$

Error term/Loss term $\text{MSE}_{\text{test}} = \frac{1}{m} ||\hat{\boldsymbol{y}}^{(\text{test})} - \boldsymbol{y}^{(\text{test})}||_2^2,$



Learning: Optimizing the loss will estimate the model parameters w and b.
Online: Given the trained model, we can predict a value.

Examples:

- Predicting pollution levels from visibility
- Predicting reactivity of a molecule from structural data.

Material from https://www.deeplearningbook.org/ Chapter 5

Linear Regression Example

Learning/Training:
Optimize the error w.r.t. the model parameters, w

$$\nabla_{\boldsymbol{w}} \mathrm{MSE}_{\mathrm{train}} = 0$$

$$\Rightarrow \nabla_{\boldsymbol{w}} \frac{1}{m} ||\hat{\boldsymbol{y}}^{(\mathrm{train})} - \boldsymbol{y}^{(\mathrm{train})}||_2^2 = 0$$

$$\Rightarrow \frac{1}{m} \nabla_{\boldsymbol{w}} ||\boldsymbol{X}^{(\mathrm{train})}\boldsymbol{w} - \boldsymbol{y}^{(\mathrm{train})}||_2^2 = 0$$

$$\Rightarrow \nabla_{\boldsymbol{w}} \left(\boldsymbol{X}^{(\mathrm{train})}\boldsymbol{w} - \boldsymbol{y}^{(\mathrm{train})}\right)^\top \left(\boldsymbol{X}^{(\mathrm{train})}\boldsymbol{w} - \boldsymbol{y}^{(\mathrm{train})}\right) = 0 \tag{5.9}$$

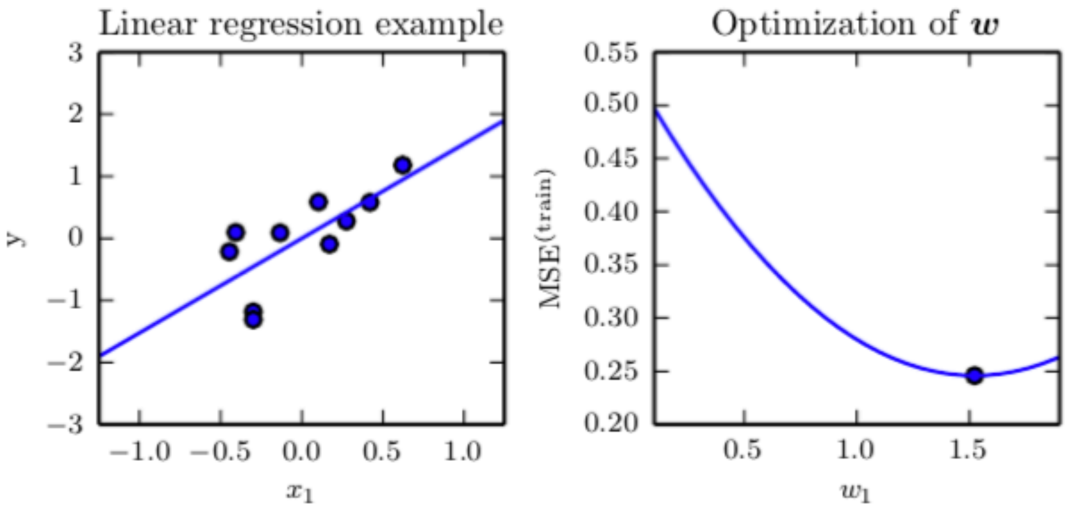
$$\Rightarrow \nabla_{\boldsymbol{w}} \left(\boldsymbol{w}^\top \boldsymbol{X}^{(\mathrm{train})\top}\boldsymbol{X}^{(\mathrm{train})}\boldsymbol{w} - 2\boldsymbol{w}^\top \boldsymbol{X}^{(\mathrm{train})\top}\boldsymbol{y}^{(\mathrm{train})} + \boldsymbol{y}^{(\mathrm{train})\top}\boldsymbol{y}^{(\mathrm{train})}\right) = 0 \tag{5.10}$$

$$\Rightarrow 2\boldsymbol{X}^{(\mathrm{train})\top}\boldsymbol{X}^{(\mathrm{train})}\boldsymbol{w} - 2\boldsymbol{X}^{(\mathrm{train})\top}\boldsymbol{y}^{(\mathrm{train})} = 0 \tag{5.11}$$

$$\Rightarrow \boldsymbol{w} = \left(\boldsymbol{X}^{(\mathrm{train})\top}\boldsymbol{X}^{(\mathrm{train})}\right)^{-1}\boldsymbol{X}^{(\mathrm{train})\top}\boldsymbol{y}^{(\mathrm{train})} \tag{5.12}$$

Inference:
Use the trained model i.e. w, to perform predictions

Optimal w and the implied linear model



Function/model space **Parameter space**

Material from
https://www.deeplearningbook.org/
Chapter 5

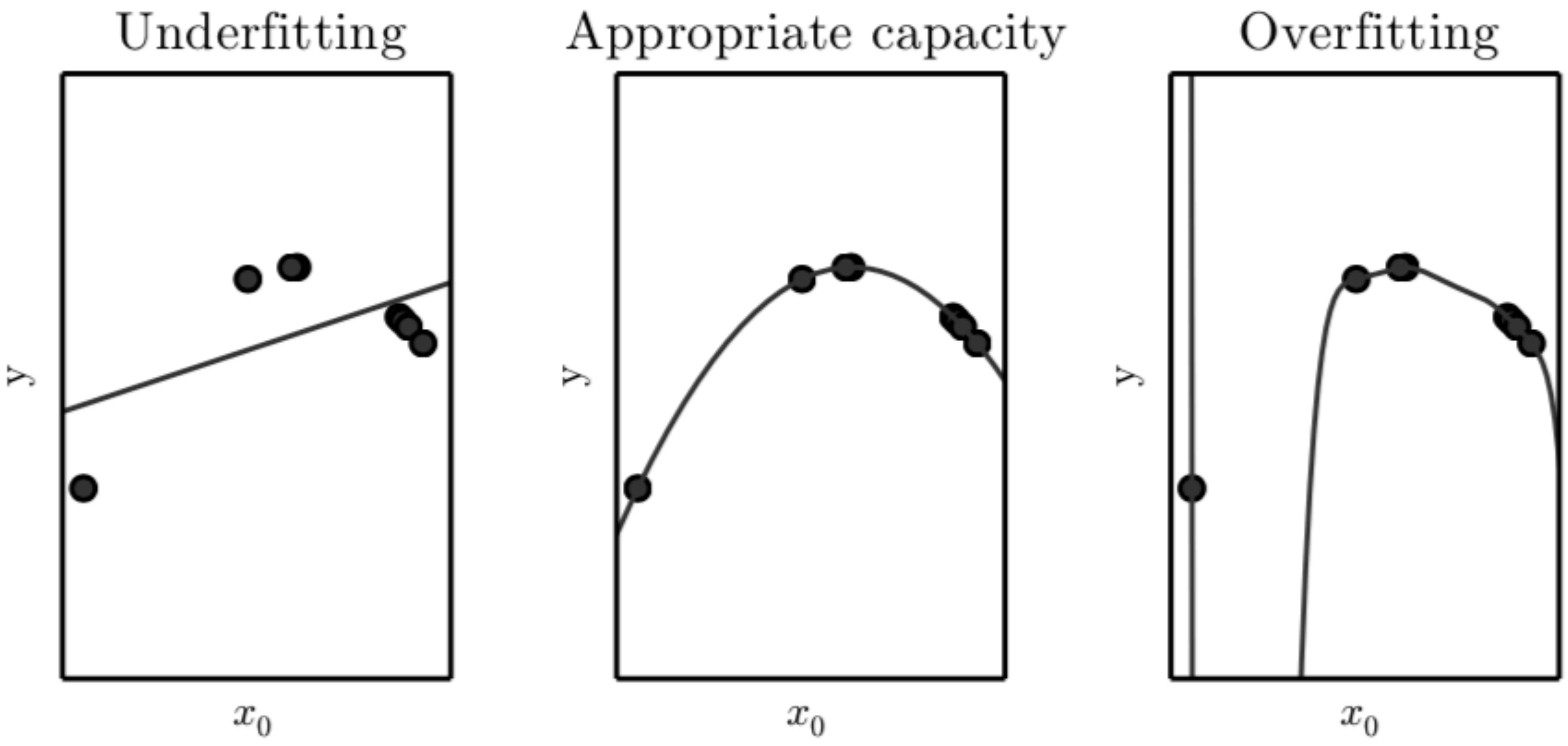
Model Fitting to Data

Overfitting and underfitting with polynomial functions

We seek a reasonable model that is neither underfitting nor over fitting.

That means, we prefer certain types of models.

How to "regularize" or solution, incorporate certain preferences?



Material from
https://www.deeplearningbook.org/
Chapter 5

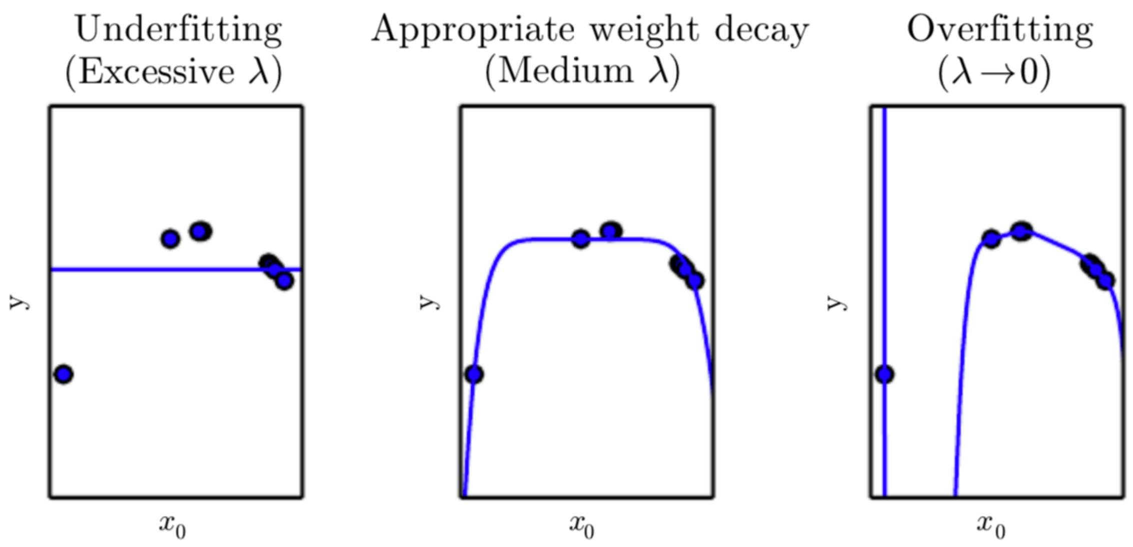
Regularization

- Adding a **preference** for one solution in its hypothesis space to another.
  - Incorporate that preference in the objective function we are optimization.

- **Weight term**
  - Adding a term to the loss function that prefers smaller squared sum of weights. A prior over the parameters.
  - Penalize a very complex model to explain the date.

- **Lambda parameter**
  - Selected ahead of time that controls the strength of our preference for smaller weights.
  - It is a “hyper”-parameter that needs tuning, expressing our trade off.

$$J(\boldsymbol{w}) = \mathrm{MSE}_{\mathrm{train}} + \lambda \boldsymbol{w}^\top \boldsymbol{w}$$



Underfitting (Excessive λ)

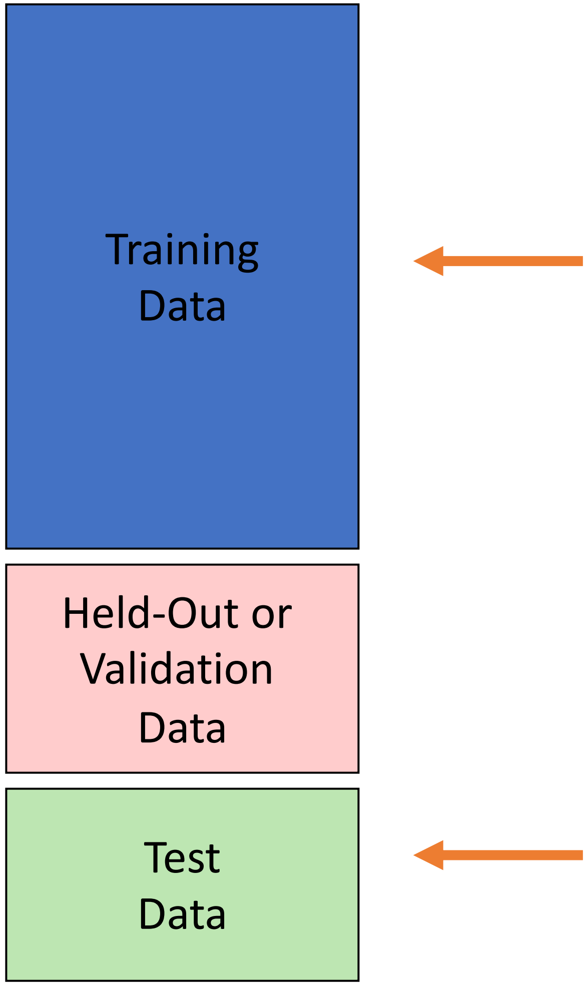
Appropriate weight decay (Medium λ)

Overfitting ($\lambda \to 0$)

Material from
https://www.deeplearningbook.org/
Chapter 5

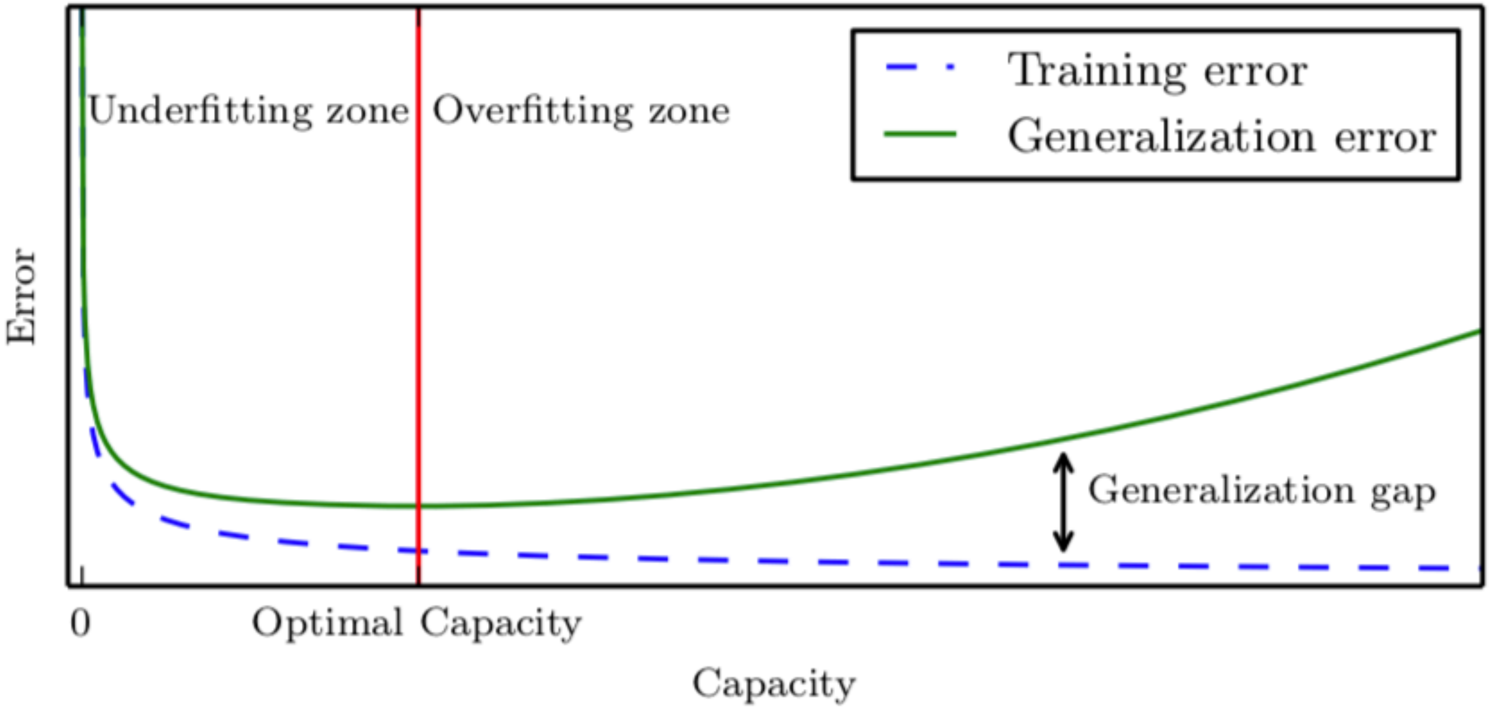
Training and Generalization Errors

- **Goal**
  - Doing learning to estimate parameters of a model.
  - Optimizing a loss function that measures the good ness of a model.
- **Data set consists of labeled instances**
  - Digit images and digits
  - Emails with labels such as spam or no spam
- **Need for Generalization**
  - A machine learning algorithm must perform well on new, previously unseen inputs
  - Note just those on which the model was trained.
  - Essentially, it should not memorize the data.
- **Training and Test sets**
  - The learner should never look at the test data.
  - Training set -> training error
  - Test set -> “generalization” error
- **Central Challenge for a Learning Algorithm**
  - Make the training error small
  - Make the gap between the training and the test error small



Relation between model capacity and error

- **Underfitting regime**
    - Training error and generalization error are both high.
- **Increase capacity**
    - Training error decreases
    - Gap between training and generalization error increases.
- **Overfitting**
    - The size of this gap outweighs the decrease in training error,
    - capacity is too large, above the optimal capacity.



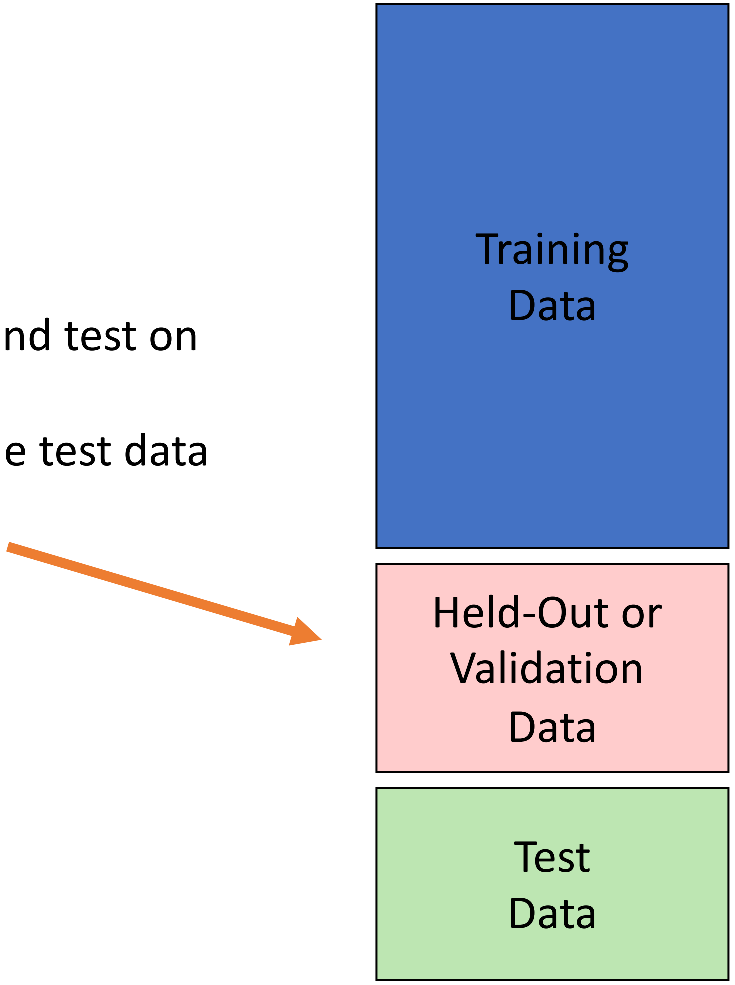
We can train a model only on the training set. The test set is not available during training.
How do we know the generalization error when we cannot use the test set?

Material from
https://www.deeplearningbook.org/
Chapter 5

Hyper-parameters and Validation Sets

- Parameters: P(X), P(Y|X)
- Hyper-parameters: k, lambda etc.

- Selecting hyper-parameters
  - For each value of the hyperparameters, train and test on the held-out data or the validation data set.
  - Choose the best value and do a final test on the test data



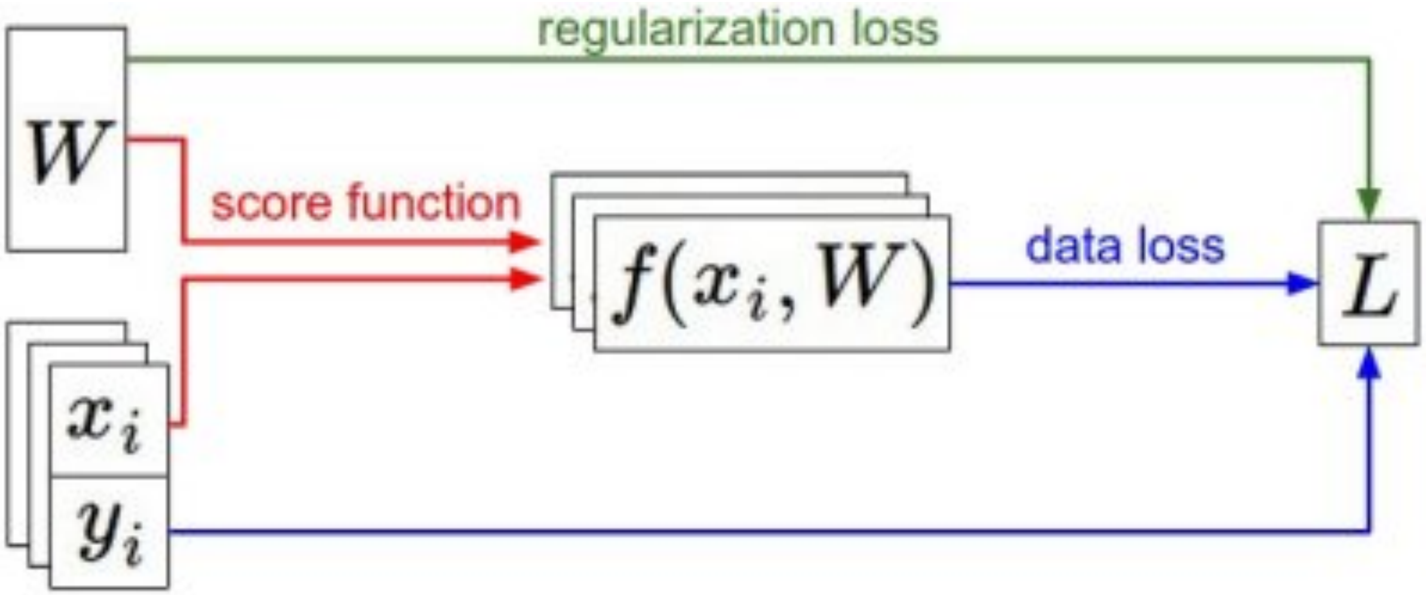
Training Data

Held-Out or Validation Data

Test Data

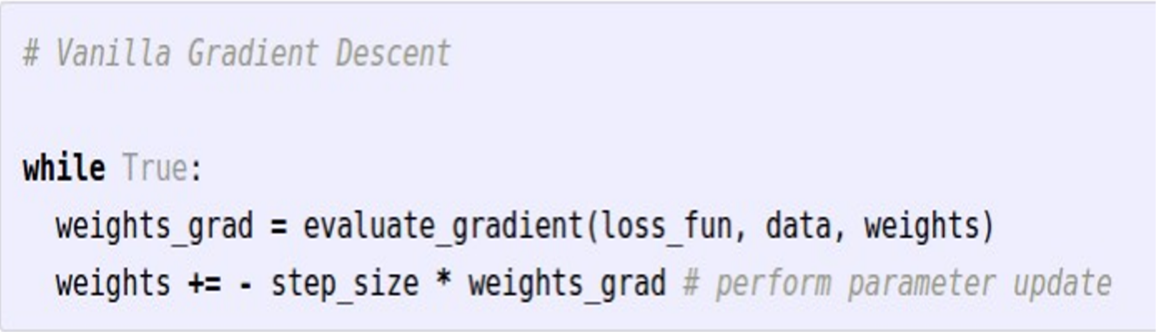
Optimizing the loss

- The loss term is composed of the predictions under the weights and the regularization term.
- The goal is to optimize the loss function given the parameters

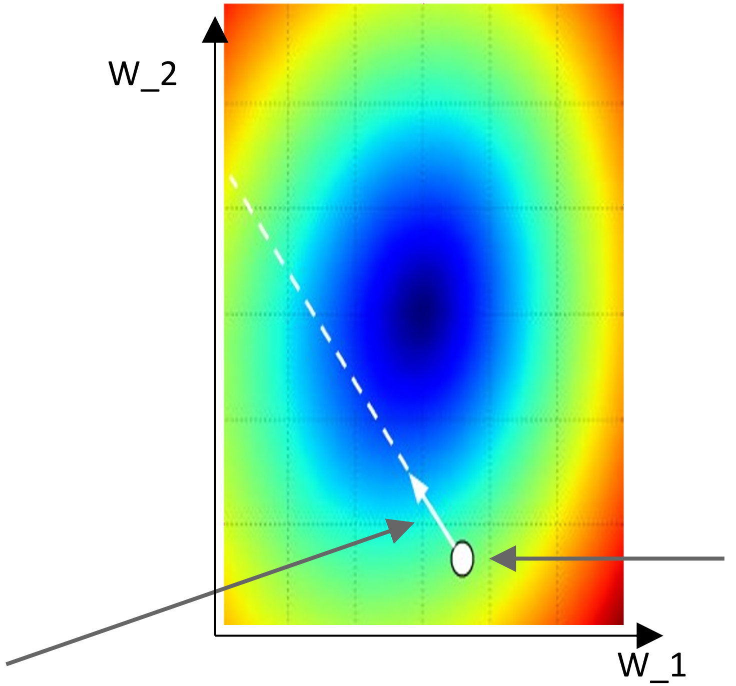


Gradient Descent

- Compute the gradient and take the step in that direction.
- Gradient computation
  - Analytic
  - Numerical



```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```



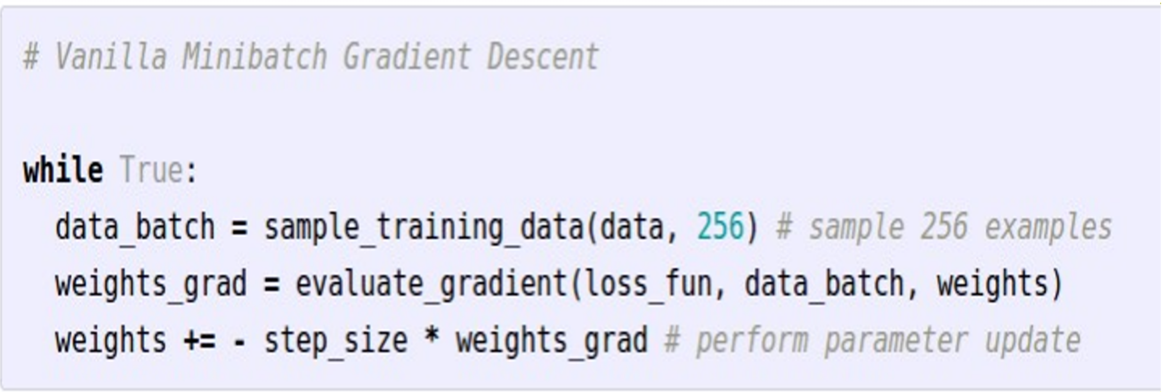
original W

negative gradient direction

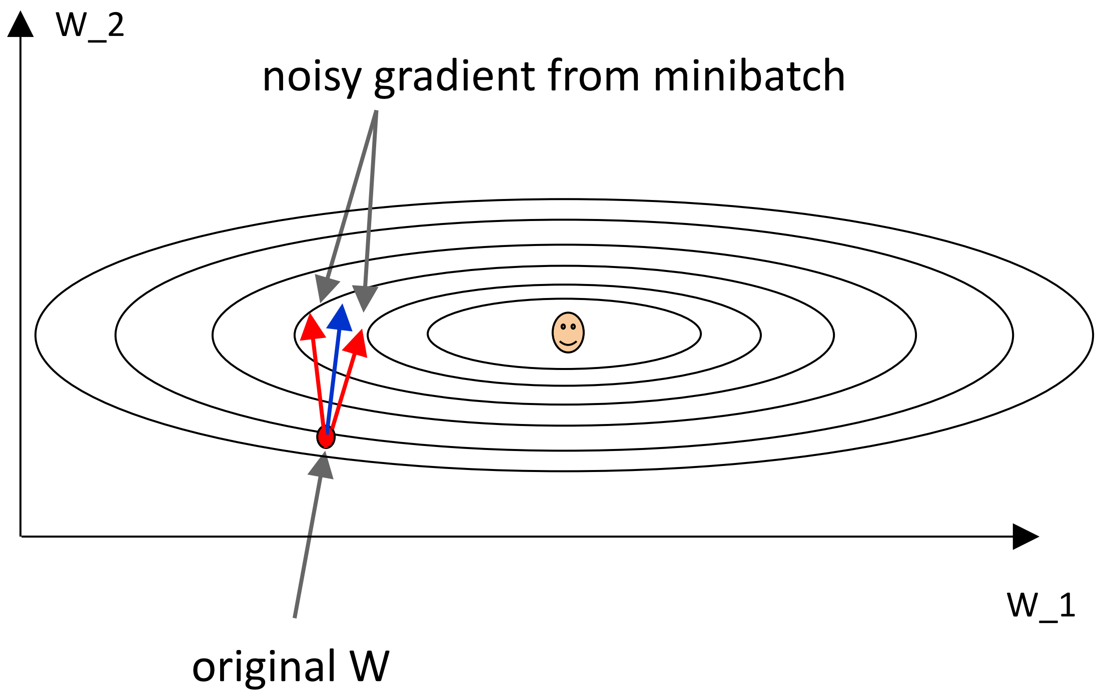
Mini-batch Gradient Descent

Problem: Large data sets. Difficult to compute the full loss function over the entire training set in order to perform only a single parameter update

Solution: Only use a small portion of the training set to compute the gradient.



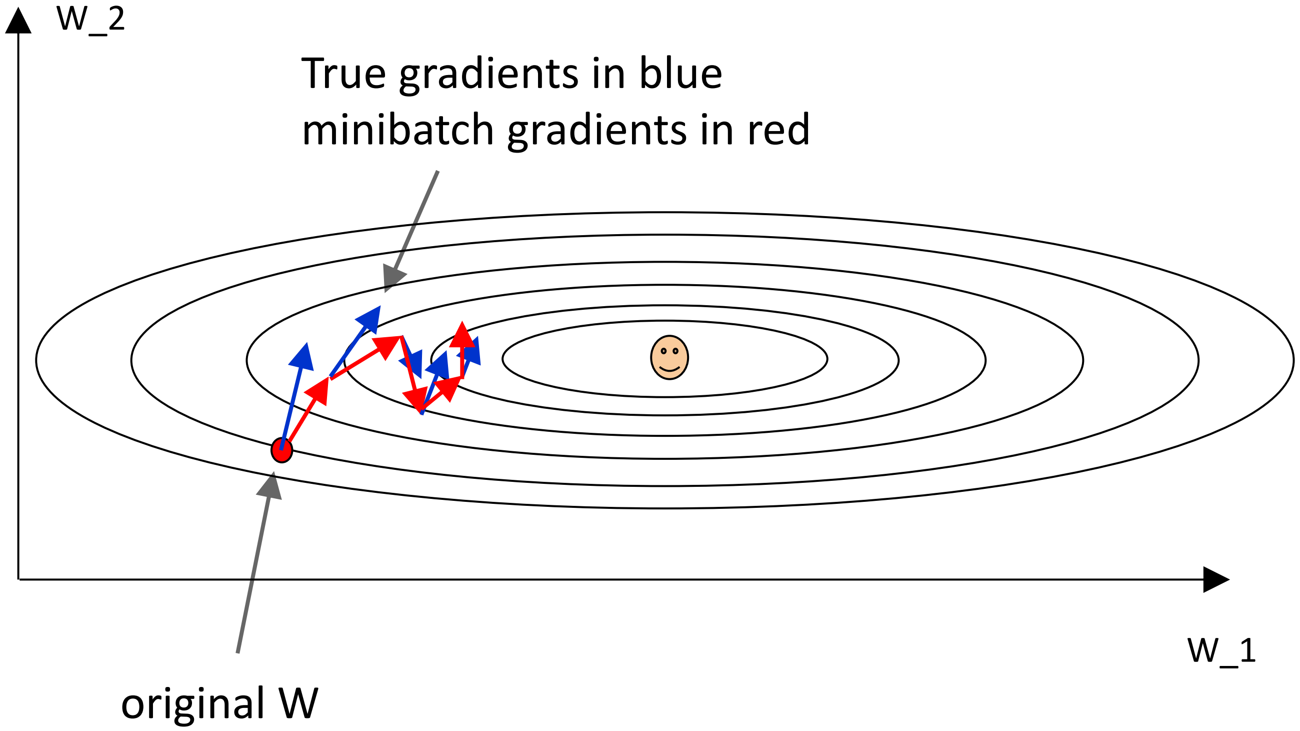
```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```



Stochastic Gradient Descent

Setting the mini-batch to contain only a single example.

Gradients are noisy but still make good progress on average.



Gradient vs Mini-batch Gradient Descent

Cost function decomposes as a sum over training examples of per-example loss function

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{\text{data}}} L(\boldsymbol{x}, y, \boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

$$L(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\log p(y \mid \boldsymbol{x}; \boldsymbol{\theta})$$

Gradient for an additive cost function

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

Mini-batch m’, where m’ is kept constant while m is increasing.

Update the parameters with the estimated gradient from the mini-batch.

$$\boldsymbol{g} = \frac{1}{m'}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) \qquad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{g}$$

Theta denotes the parameters. Same as w used in the previous slides.