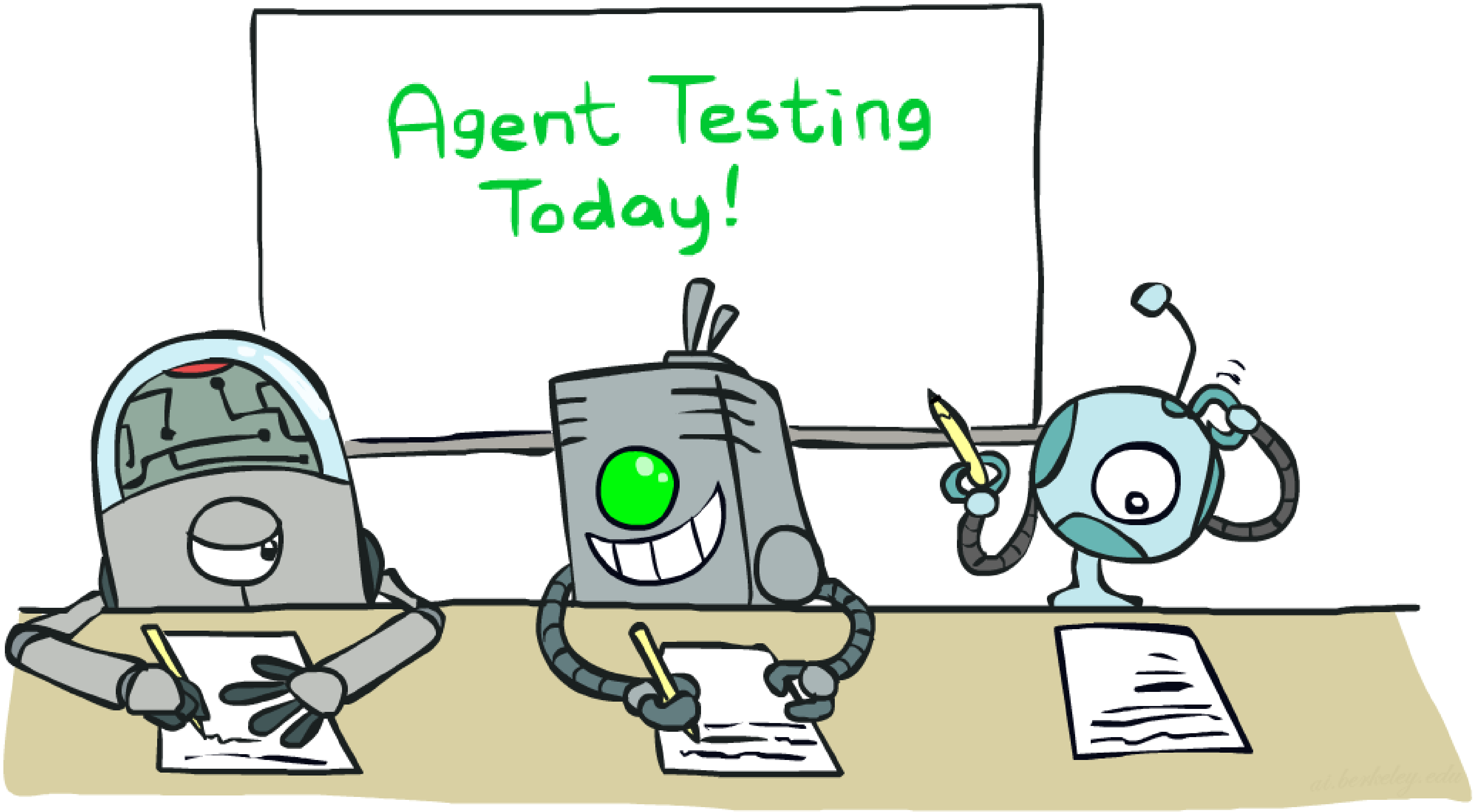
Supervised Machine Learning



AIMA Chapter 18, 20

[Adapted from slides by Dan Klein and Pieter Abbeel at UC Berkeley]

Machine Learning

- Up until now: how to use a model to make optimal decisions
  - Except reinforcement learning

- Machine learning: how to acquire a model from data / experience

- Related courses
  - SI151 Optimization and Machine Learning
  - CS282 Machine Learning
  - CS280 Deep Learning

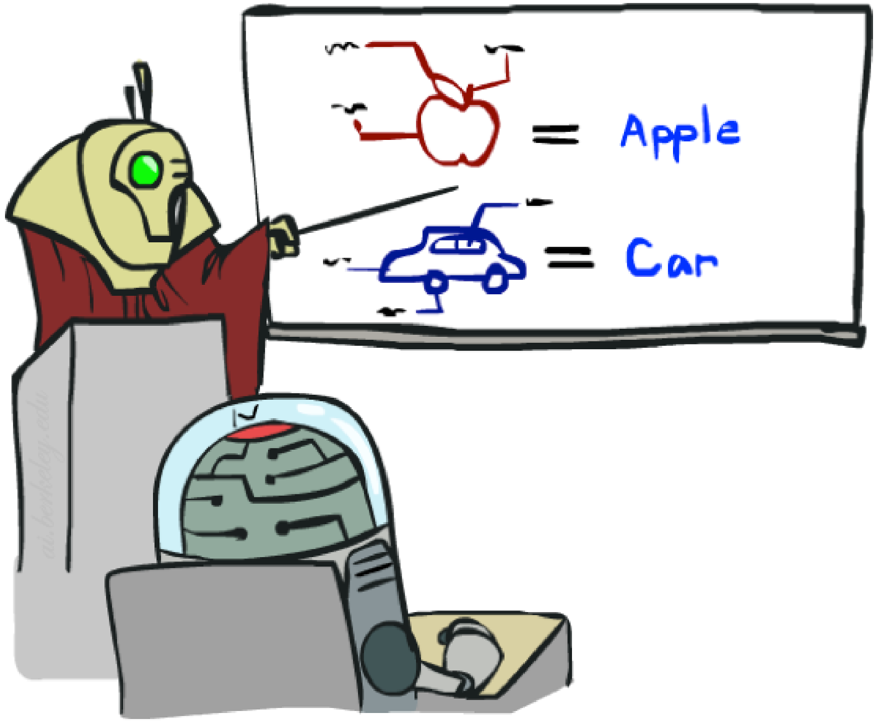
Types of Learning

- Supervised learning
  - Training data includes desired outputs
- Unsupervised learning
  - Training data does not include desired outputs
- Semi-supervised learning
  - Training data includes a few desired outputs
- Reinforcement learning
  - Rewards from sequence of actions

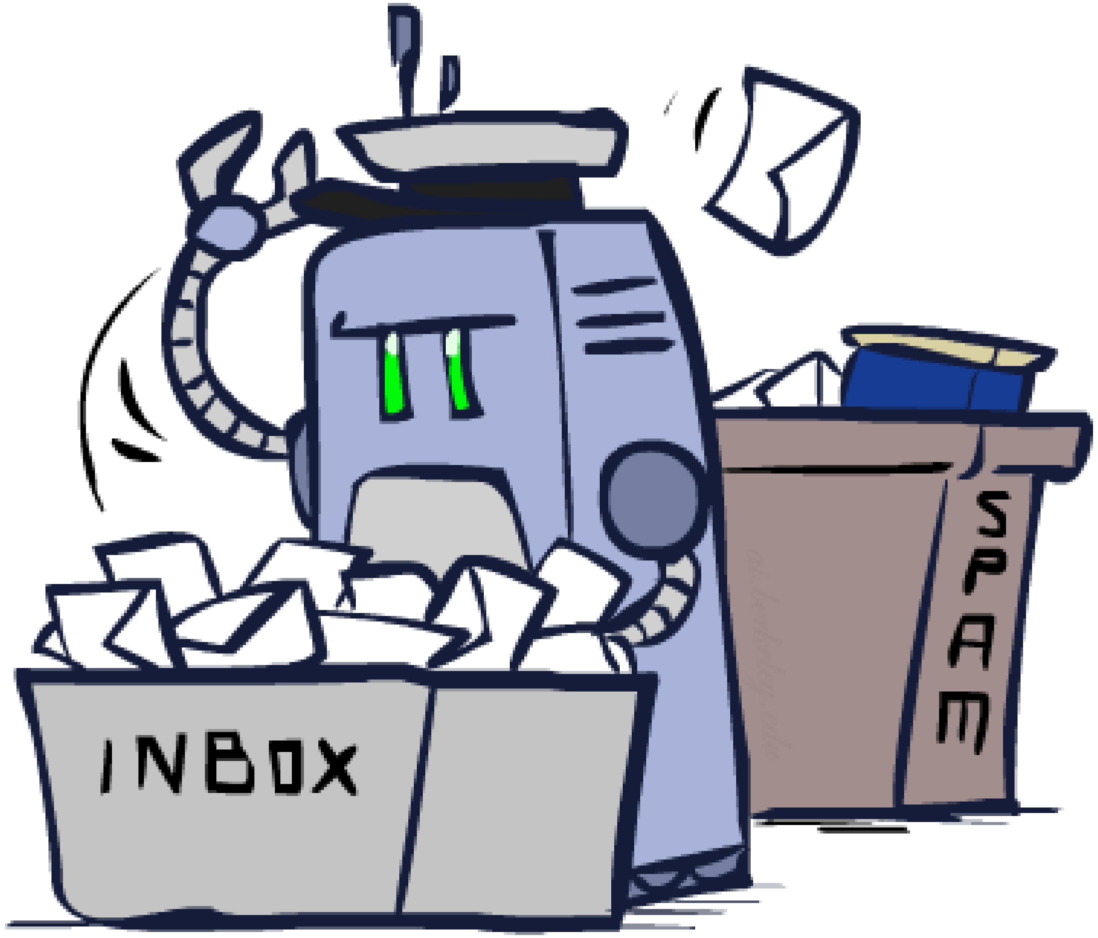
Supervised learning

- To learn an unknown *target function* f
- Input: a *training set* of *labeled examples* (x_j, y_j) where $y_j = f(x_j)$
- Output: *hypothesis* h that is “close” to f

- Two types of supervised learning
  - Classification = learning f with discrete output value
  - Regression = learning f with real-valued output value



Classification



Example: Spam Filter

- Input: an email
- Output: spam/ham

- Setup:
  - Get a large collection of example emails, each labeled “spam” or “ham” (by hand)
  - Want to learn to predict labels of new, future emails

- Features: The attributes used to make the ham / spam decision
  - Words: FREE!
  - Text Patterns: $dd, CAPS
  - Non-text: SenderInContacts
  - …

Dear Sir.

First, I must solicit your confidence in this transaction, this is by virture of its nature as being utterly confidencial and top secret. …

TO BE REMOVED FROM FUTURE MAILINGS, SIMPLY REPLY TO THIS MESSAGE AND PUT "REMOVE" IN THE SUBJECT.

99 MILLION EMAIL ADDRESSES
FOR ONLY $99

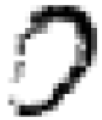
Ok, Iknow this is blatantly OT but I'm beginning to go insane. Had an old Dell Dimension XPS sitting in the corner and decided to put it to use, I know it was working pre being stuck in the corner, but when I plugged it in, hit the power nothing happened.

Example: Digit Recognition

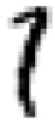
- Input: images / pixel grids
- Output: a digit 0-9

- Setup:
  - Get a large collection of example images, each labeled with a digit
  - Want to learn to predict labels of new, future digit images

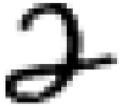
- Features: The attributes used to make the digit decision
  - Pixels: (6,8)=ON
  - Shape Patterns: NumComponents, AspectRatio, NumLoops
  - …



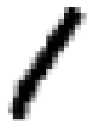
0



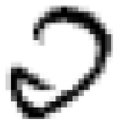
1



2



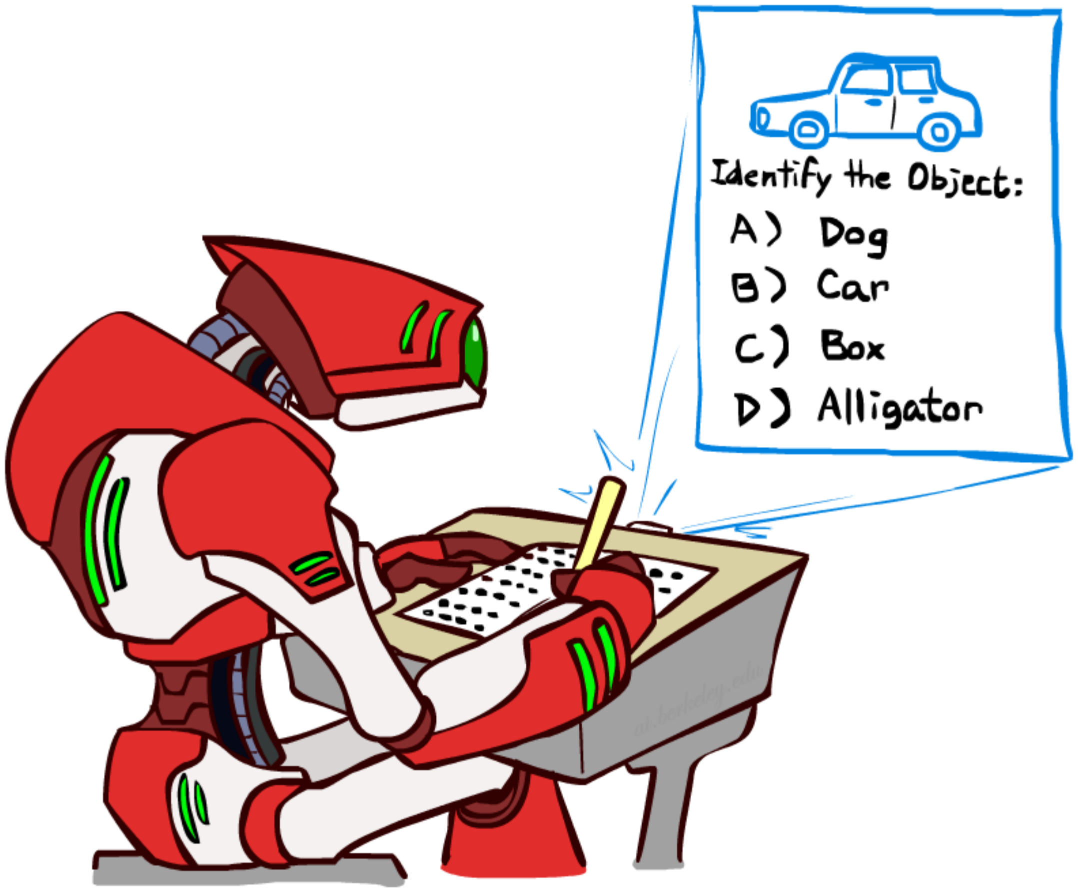
1



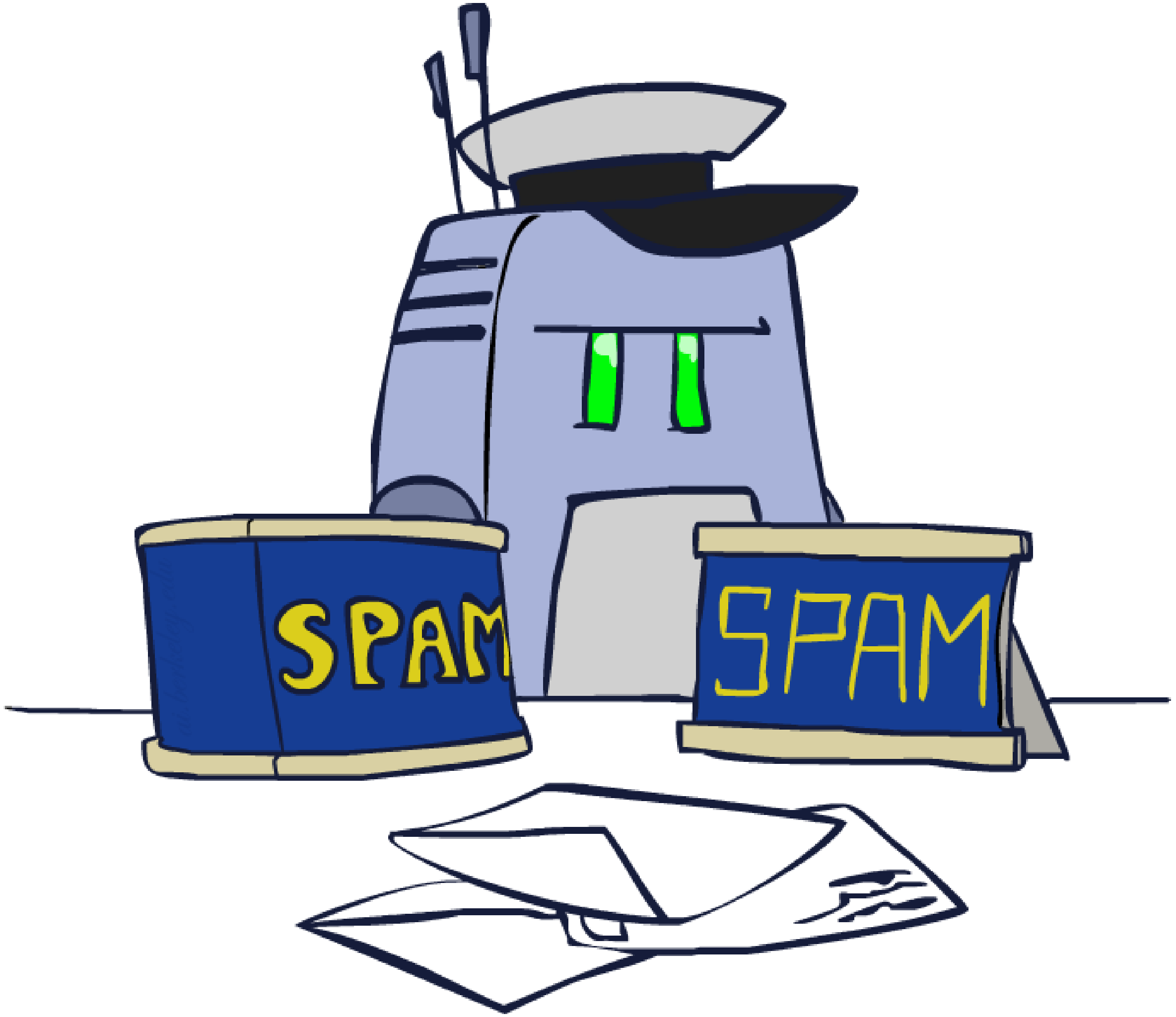
??

Other Classification Tasks

- Medical diagnosis
  - input: symptoms
  - output: disease
- Automatic essay grading
  - input: document
  - output: grades
- Fraud detection
  - input: account activity
  - output: fraud / no fraud
- Email routing
  - input: customer complaint email
  - output: which department needs to ignore this email
- Fruit and vegetable inspection
  - input: image (or gas analysis)
  - output: moldy or OK
- … many more



Model-Based Classification



Model-Based Classification

- Model-based approach
  - Build a model (e.g. Bayes' net) where both the label and features are random variables
  - Instantiate any observed features
  - Query for the distribution of the label conditioned on the features

- Challenges
  - What structure should the BN have?
  - How should we learn its parameters?

Naïve Bayes

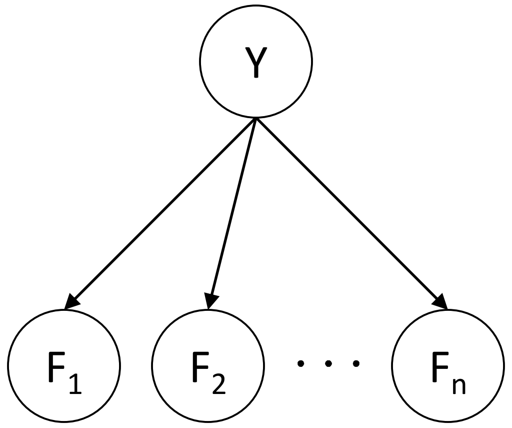
- Naive Bayes model:

|Y| parameters

$$P(\mathsf{Y}, \mathsf{F}_1 \ldots \mathsf{F}_n) = \quad P(\mathsf{Y}) \prod_i P(\mathsf{F}_i|\mathsf{Y})$$

$|Y| \times |F|^n$ values

n x |F| x |Y| parameters



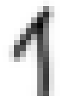
- Assume all features are independent effects of the label
- Total number of parameters is *linear* in n
- Tree-structured: *linear* inference time
- Model is very simplistic, but often works anyway

Learning Naïve Bayes

- What do we need in order to learn a Naïve Bayes?
  - 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
  - Later: how to estimate the parameters

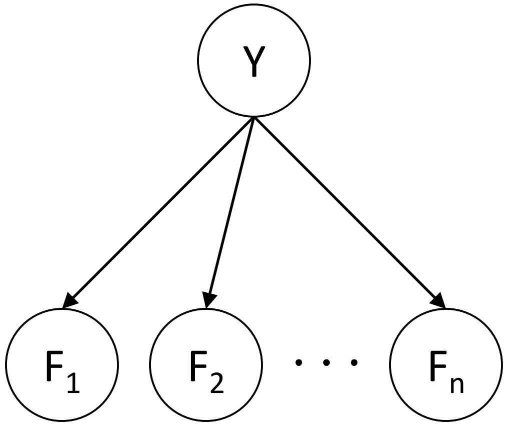
Naïve Bayes for Digits

- Simple digit recognition version:
  - 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.



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

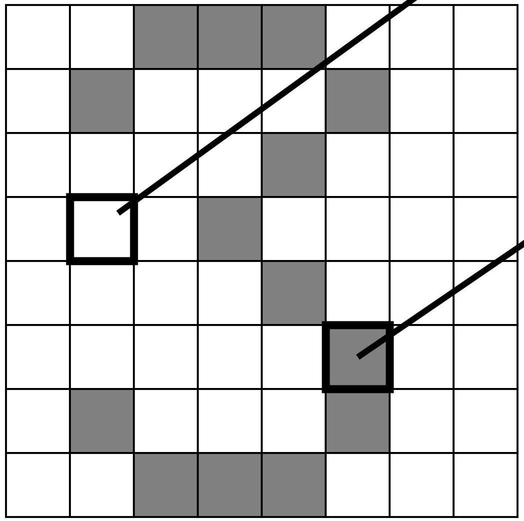
  - Here: lots of features, each is binary valued



Naïve Bayes for Digits

$P(Y)$

1	0.1
2	0.1
3	0.1
4	0.1
5	0.1
6	0.1
7	0.1
8	0.1
9	0.1
0	0.1

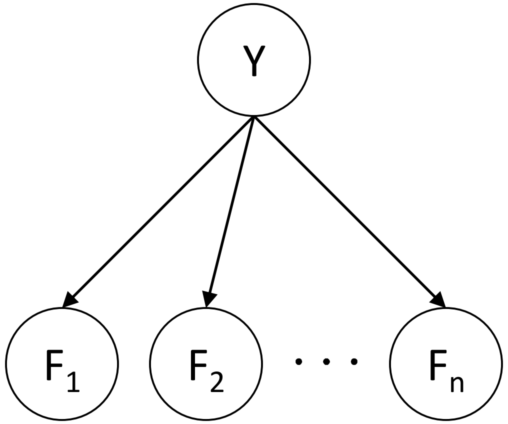


$P(F_{3,1} = on|Y)$

1	0.01
2	0.05
3	0.05
4	0.30
5	0.80
6	0.90
7	0.05
8	0.60
9	0.50
0	0.80

$P(F_{5,5} = on|Y)$

1	0.05
2	0.01
3	0.90
4	0.80
5	0.90
6	0.90
7	0.25
8	0.85
9	0.60
0	0.80



Naïve Bayes for Text

- Bag-of-words Naïve Bayes:
  - Features: W_i is the word at positon i
  - As before: predict label conditioned on feature variables (spam vs. ham)
  - As before: assume features are conditionally independent given label
  - New: each W_i is identically distributed

- Generative model: $P(Y, W_1 \dots W_n) = P(Y) \prod_i P(W_i|Y)$

Word at position i, not i^th word in the dictionary!

  - Usually, each variable gets its own conditional probability distribution P(F|Y)
  - Here
    - Each position is identically distributed
    - All positions share the same conditional probabilities P(W|Y)
  - Called "bag-of-words" because model is insensitive to word order or reordering

Example: Spam Filtering

- Model: $P(Y, W_1 \dots W_n) = P(Y) \prod_i P(W_i|Y)$

$P(Y)$

ham :	0.66
spam:	0.33

$P(W|\text{spam})$

the :	0.0156
to :	0.0153
and :	0.0115
of :	0.0095
you :	0.0093
a :	0.0086
with:	0.0080
from:	0.0075
...	

$P(W|\text{ham})$

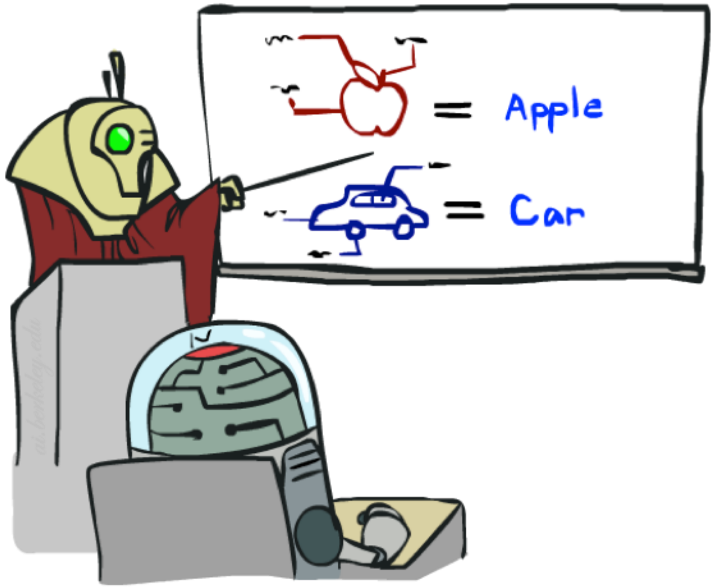
the :	0.0210
to :	0.0133
of :	0.0119
2002:	0.0110
with:	0.0108
from:	0.0107
and :	0.0105
a :	0.0100
...	

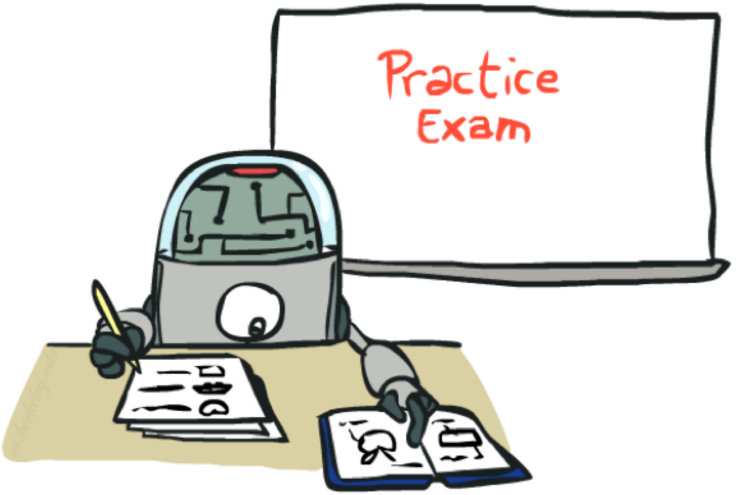
Spam Example

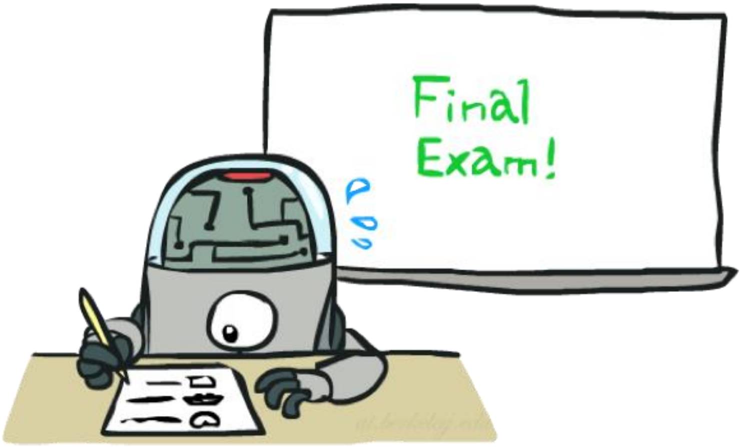
Word	P(w|spam)	P(w|ham)	Tot Spam	Tot Ham
(prior)	0.33333	0.66666	-1.1	-0.4
Gary	0.00002	0.00021	-11.8	-8.9
would	0.00069	0.00084	-19.1	-16.0
you	0.00881	0.00304	-23.8	-21.8
like	0.00086	0.00083	-30.9	-28.9
to	0.01517	0.01339	-35.1	-33.2
lose	0.00008	0.00002	-44.5	-44.0
weight	0.00016	0.00002	-53.3	-55.0
while	0.00027	0.00027	-61.5	-63.2
you	0.00881	0.00304	-66.2	-69.0
sleep	0.00006	0.00001	-76.0	-80.5

P(spam | w) = 98.9

Training and Testing

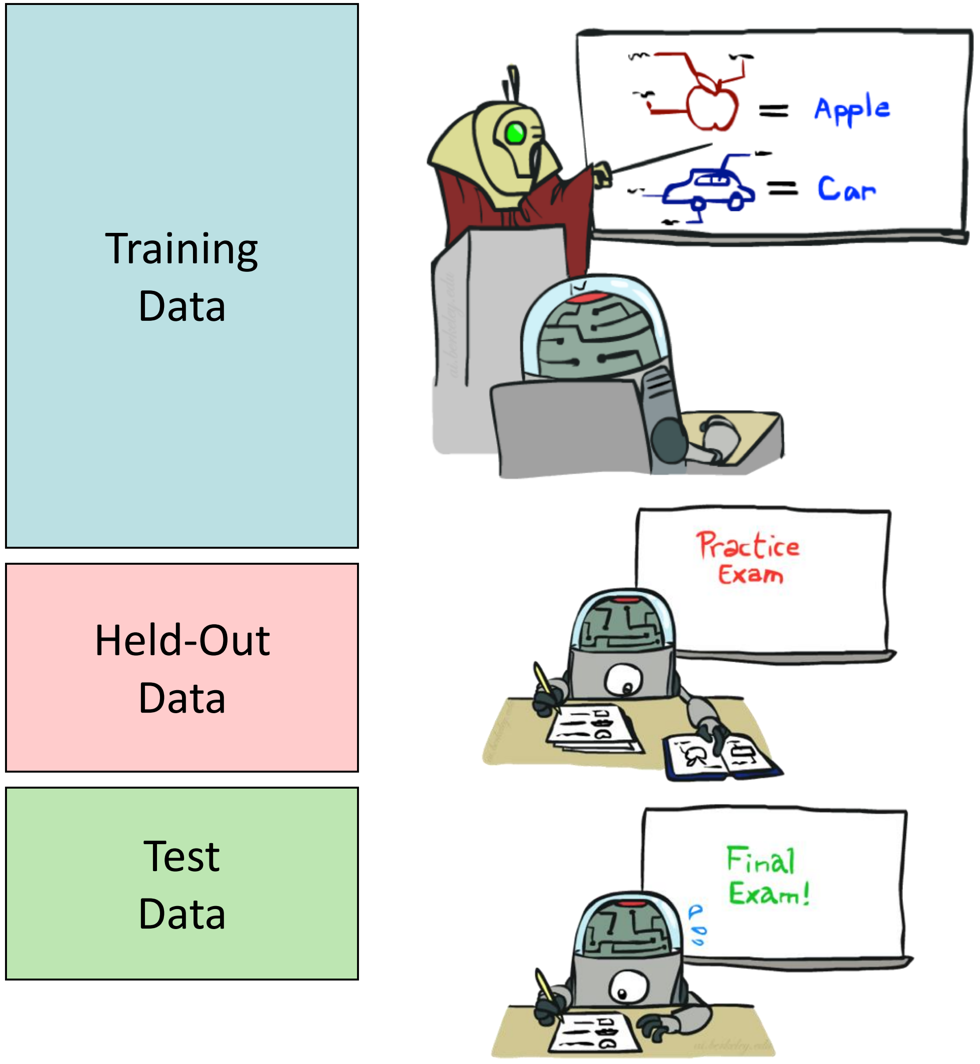




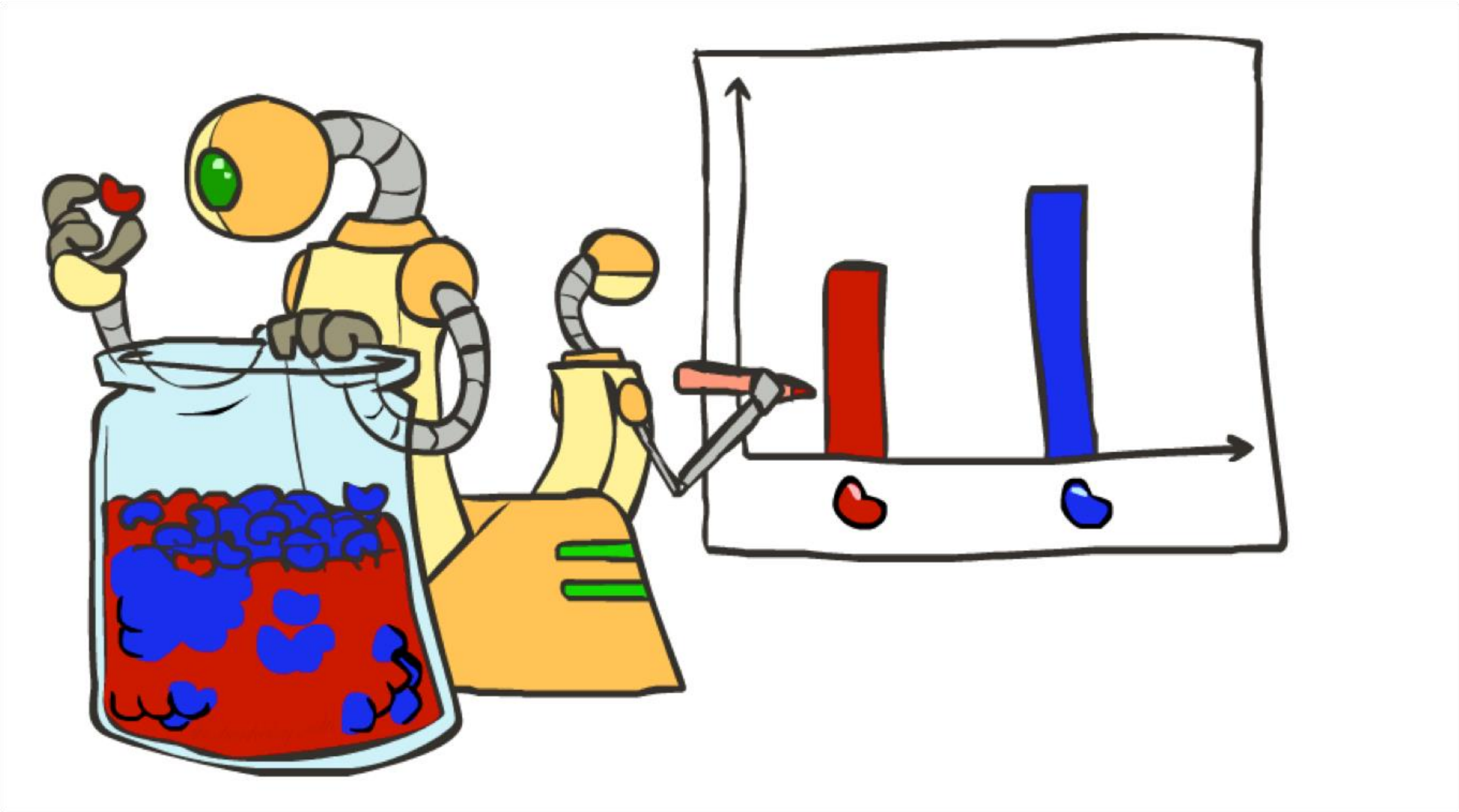


Important Concepts

- Data: labeled instances, e.g. emails marked spam/ham
  - Training set
  - Held out set
  - Test set
- Experimentation cycle
  - Learn parameters (e.g. model probabilities) on training set
  - Tune hyperparameters on held-out set
  - Compute accuracy of test set (fraction of instances predicted correctly)
  - Very important: never “peek” at the test set!



Training

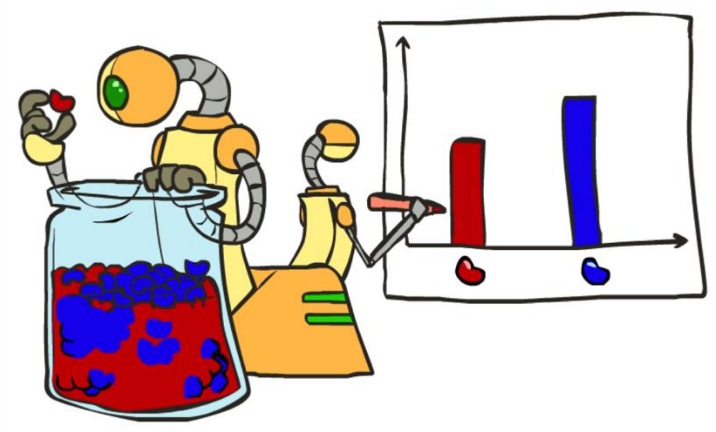


Parameter Estimation

- Estimating the distribution of a random variable
- *Elicitation:* ask a human (why is this hard?)
- *Empirically:* use training data (learning!)
  - For each outcome x, look at the *empirical rate* of that value

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

  - Ex:
    - We've seen 1000 words from spam emails, among which we see "money" for 50 times
    - So we set P(money | spam) = 0.05
  - This is the estimate that maximizes the *likelihood of the data*
    - Likelihood: conditional probability of the data given the parameters



Maximum Likelihood Estimation

- Coin flipping:
  - P(Heads) = θ, P(Tails) = 1-θ
- Flips are *i.i.d.*
  - Independent events
  - Identically distributed according to unknown distribution
- Sequence *D* of α_H Heads and α_T Tails

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

Maximum Likelihood Estimation

- **Data:** Observed set D of α_H Heads and α_T Tails
- **Likelihood:**

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

- **MLE:** Choose θ to maximize probability of *D*

$$\begin{aligned} \widehat{\theta} &= \arg\max_{\theta} \; P(\mathcal{D} \mid \theta) \\ &= \arg\max_{\theta} \; \ln P(\mathcal{D} \mid \theta) \end{aligned}$$

Maximum Likelihood Estimation

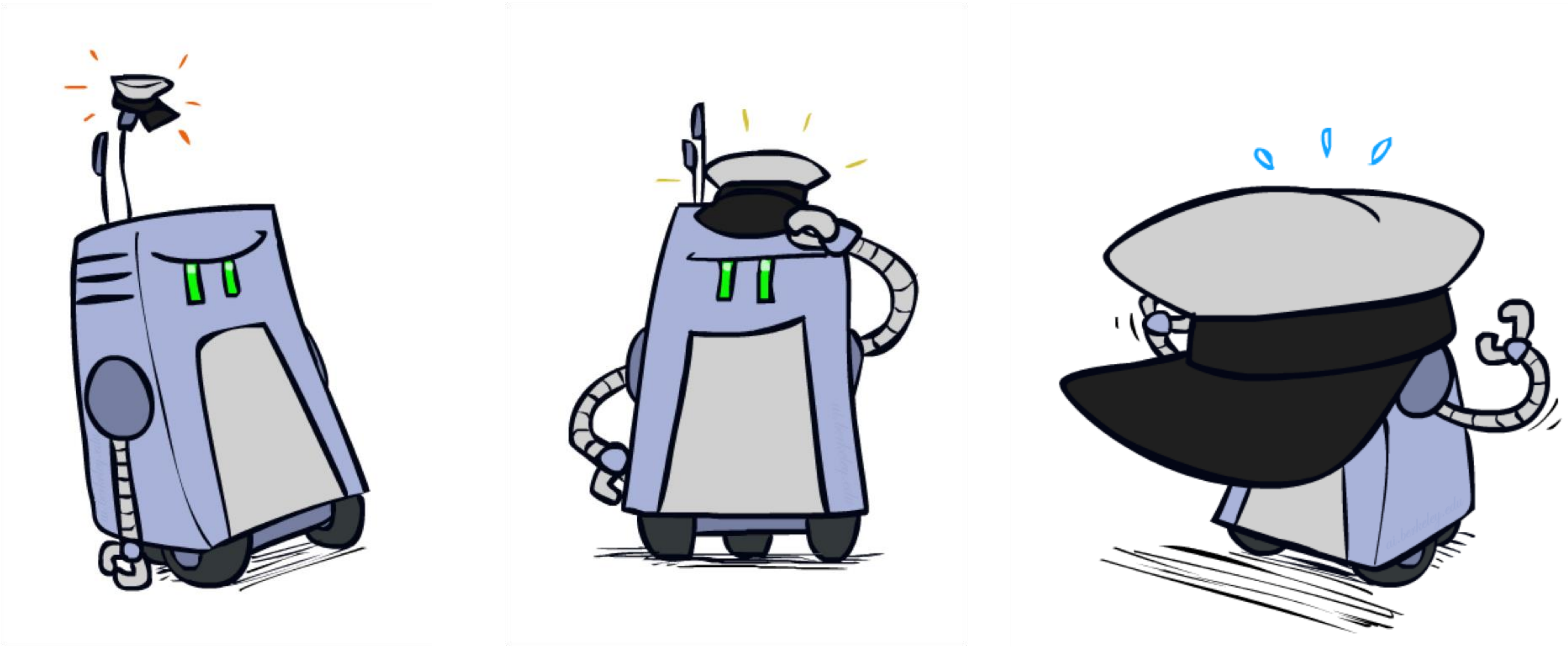
$$\begin{aligned}\widehat{\theta} &= \arg\max_{\theta} \quad \ln P(\mathcal{D} \mid \theta) \\ &= \arg\max_{\theta} \quad \ln \theta^{\alpha_H}(1-\theta)^{\alpha_T}\end{aligned}$$

- Set derivative to zero, and solve!

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

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

Generalization and Overfitting



Example: Overfitting

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

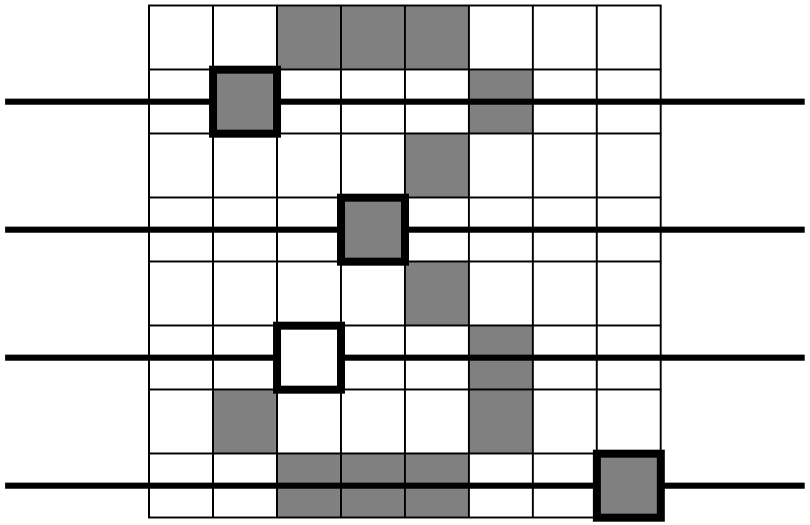
$P(C = 2) = 0.1$

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

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

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

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



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

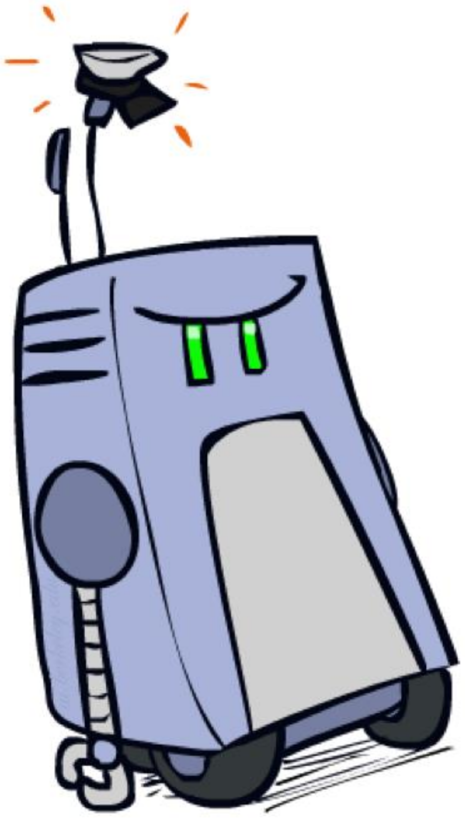
$P(C = 3) = 0.1$

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

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

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

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



2 wins!!

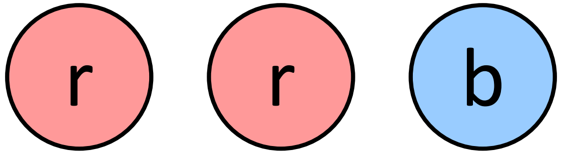
Generalization and Overfitting

- Using empirical rate will overfit the training data!
  - Just because we never saw a 3 with pixel (15,15) on during training doesn't mean we won't see it at test time
  - Just because we never saw a word in spam emails during training doesn't mean we won't see it at test time
  - Therefore, we can't give unseen events zero probability
  - More generally, rates in the training data may not exactly match rates at test time
  - Overfitting: learn to fit the training data very closely, but fit the test data poorly

- To generalize better: we need to smooth or regularize the estimates

Laplace Smoothing

- Laplace's estimate:
  - Pretend you saw every outcome once more than you actually did



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

$$= \frac{c(x) + 1}{N + |X|}$$

$$P_{ML}(X) =$$

$$P_{LAP}(X) =$$

  - Can derive this estimate with *Dirichlet priors*

Laplace Smoothing

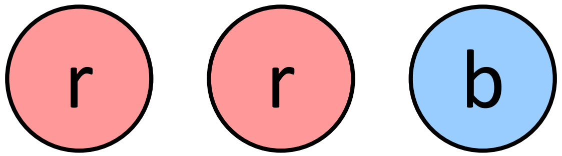
- Laplace's estimate (extended):
  - Pretend you saw every outcome k extra times

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

  - k is the strength of the prior
  - What's Laplace with k = 0?

- Laplace for conditionals:
  - Smooth each condition independently:

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



$$P_{LAP,0}(X) =$$

$$P_{LAP,1}(X) =$$

$$P_{LAP,100}(X) =$$

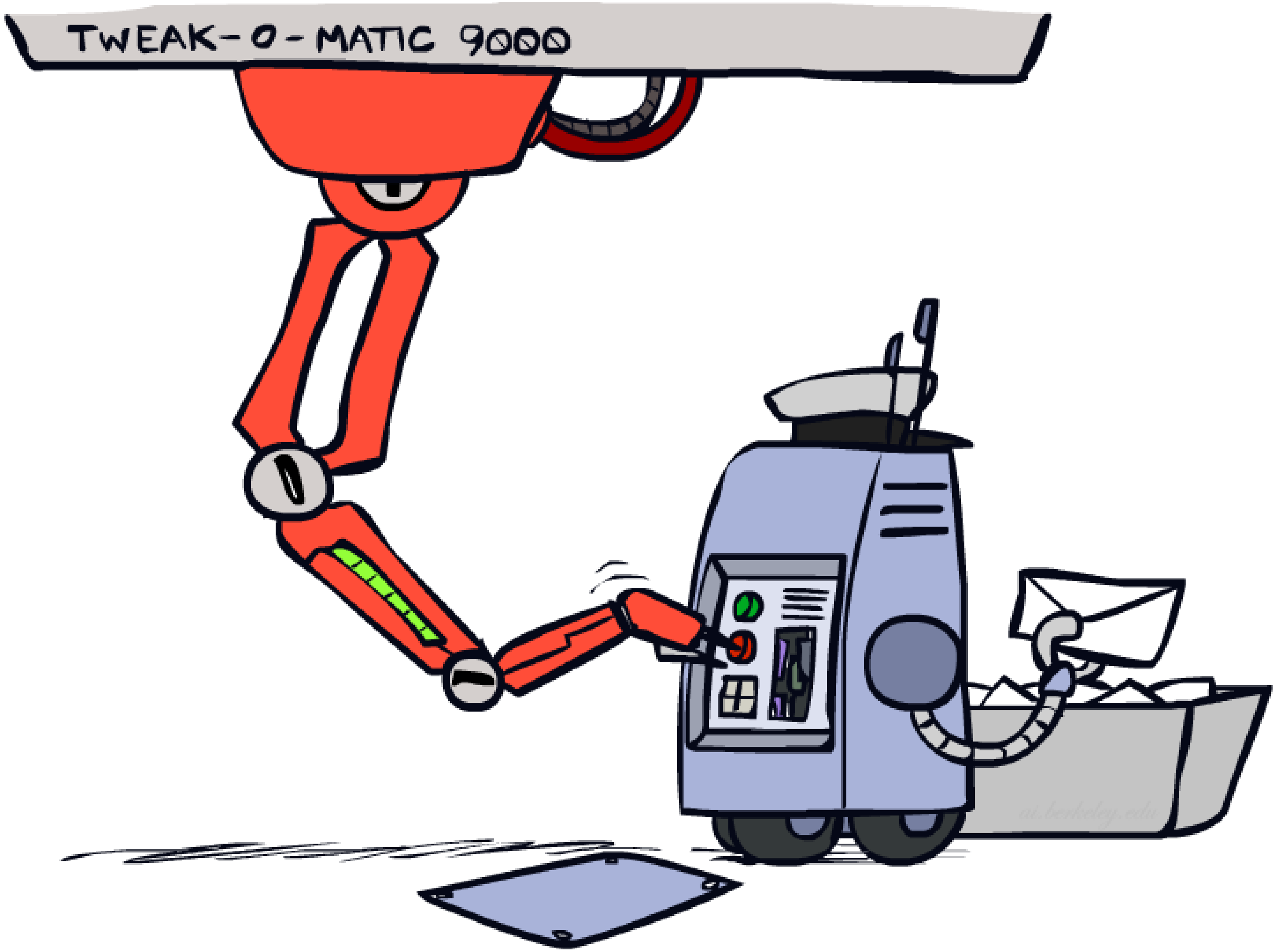
Estimation: Linear Interpolation

- In practice, Laplace often performs poorly for P(X|Y):
  - When |X| is very large
  - When |Y| is very large

- Another option: linear interpolation
  - Also get the empirical P(X) from the data
  - Make sure the estimate of P(X|Y) isn't too different from the empirical P(X)

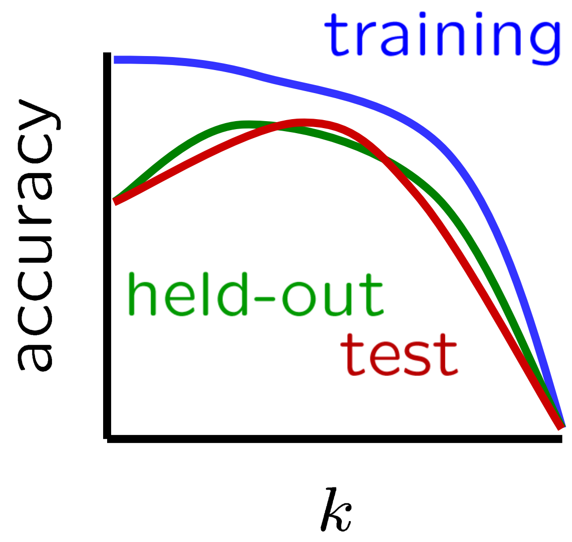
$$P_{LIN}(x|y) = \alpha \widehat{P}(x|y) + (1.0 - \alpha)\widehat{P}(x)$$

Tuning



Tuning on Held-Out Data

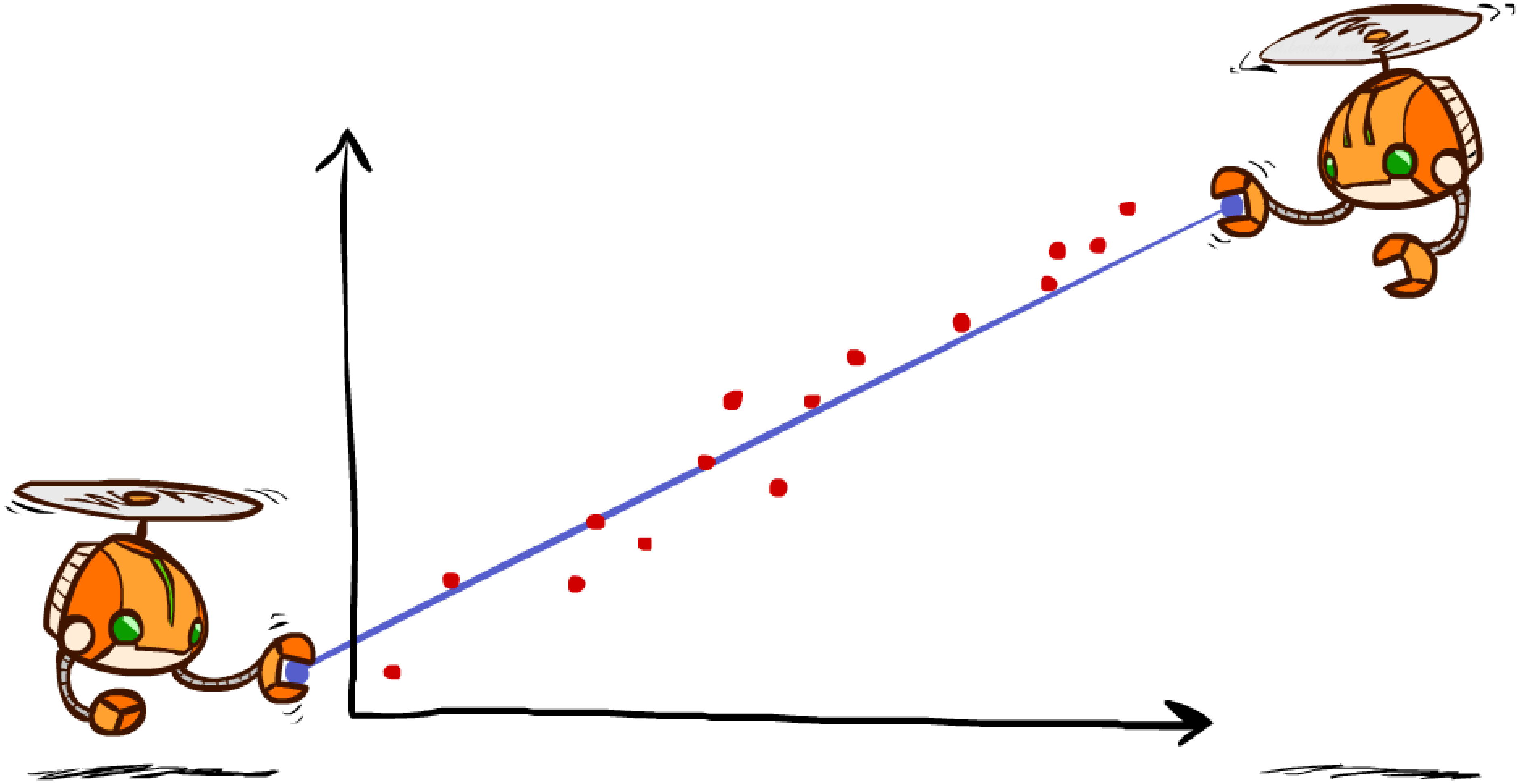
- Now we've got two kinds of unknowns
  - Parameters: the probabilities P(X|Y), P(Y)
  - Hyperparameters: e.g. the amount / type of smoothing to do, k, α
- What should we learn where?
  - Learn parameters from training data
  - Tune hyperparameters on different data
    - Why?
  - For each value of the hyperparameters, train and test on the held-out data
  - Choose the best value and do a final test on the test data



More classification methods

- Decision trees
- Nearest neighbors
- Neural networks
- Support Vector Machines
- Model ensembles
- ……

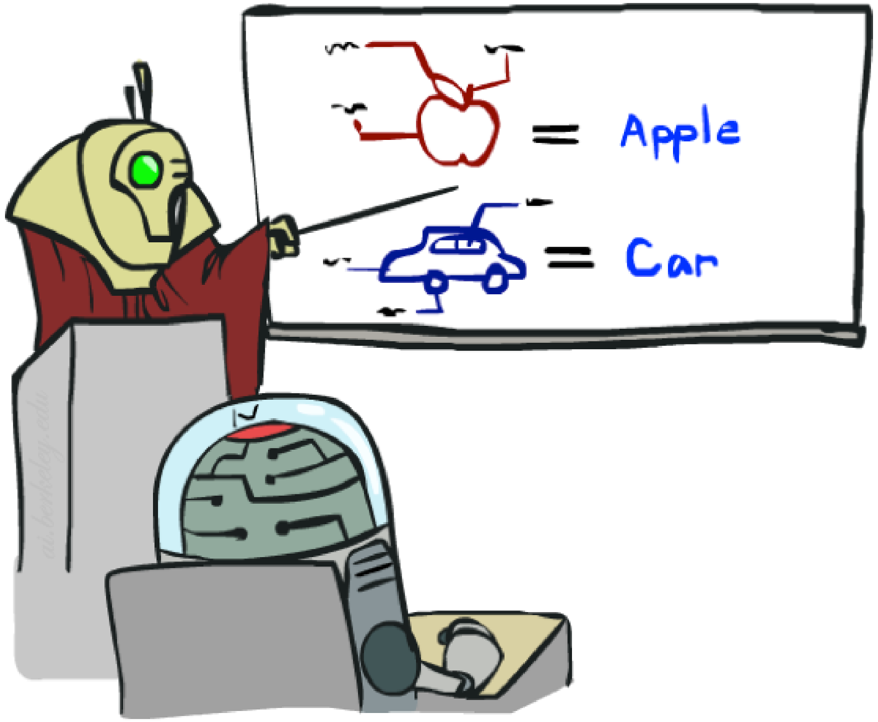
Regression



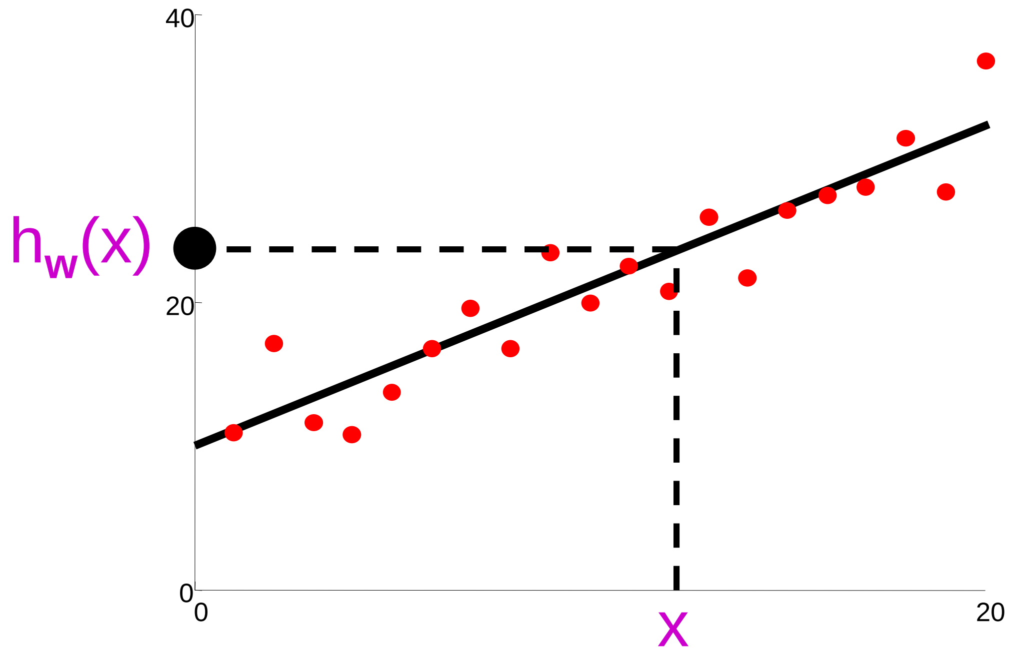
Supervised learning

- To learn an unknown ***target function*** f
- Input: a ***training set*** of ***labeled examples*** (x_j, y_j) where $y_j = f(x_j)$
- Output: ***hypothesis*** h that is "close" to f

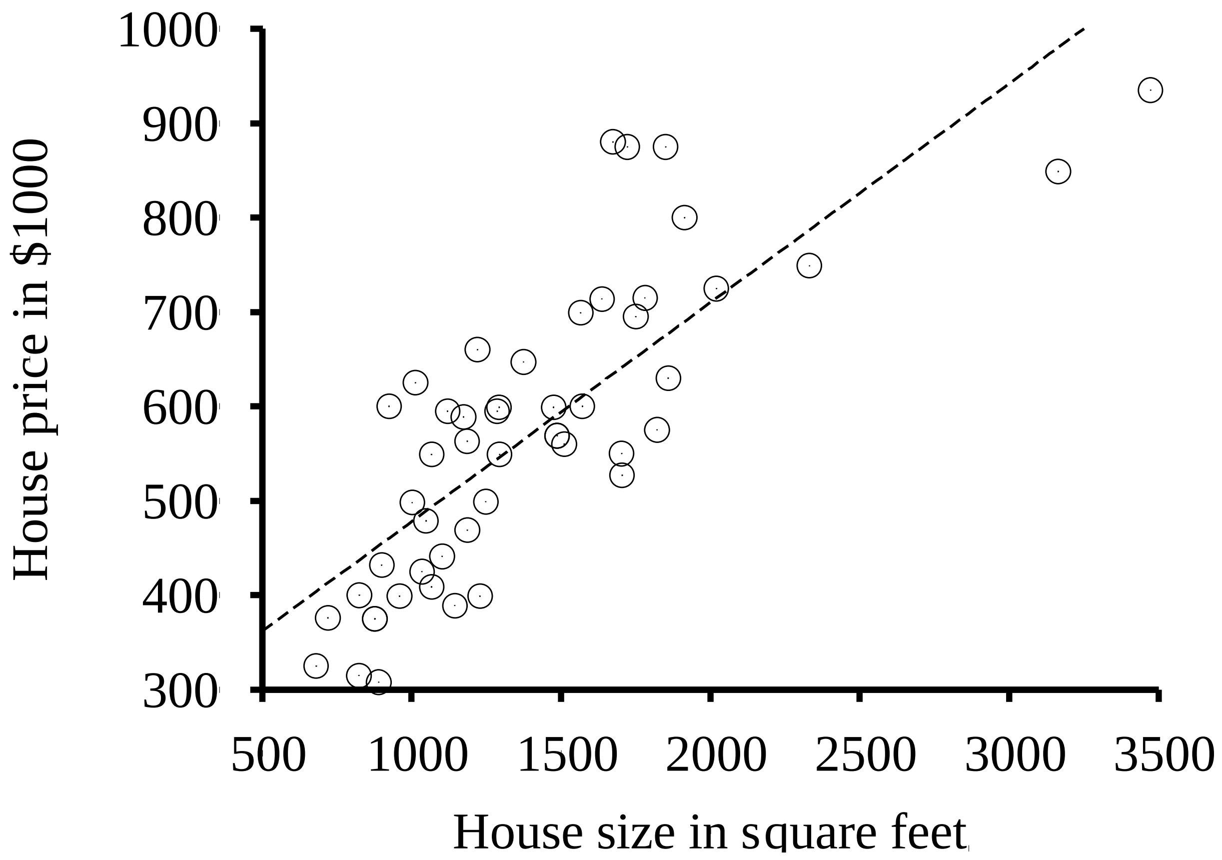
- Two types of supervised learning
  - Classification = learning f with discrete output value
  - Regression = learning f with real-valued output value



Linear regression = fitting a straight line/hyperplane



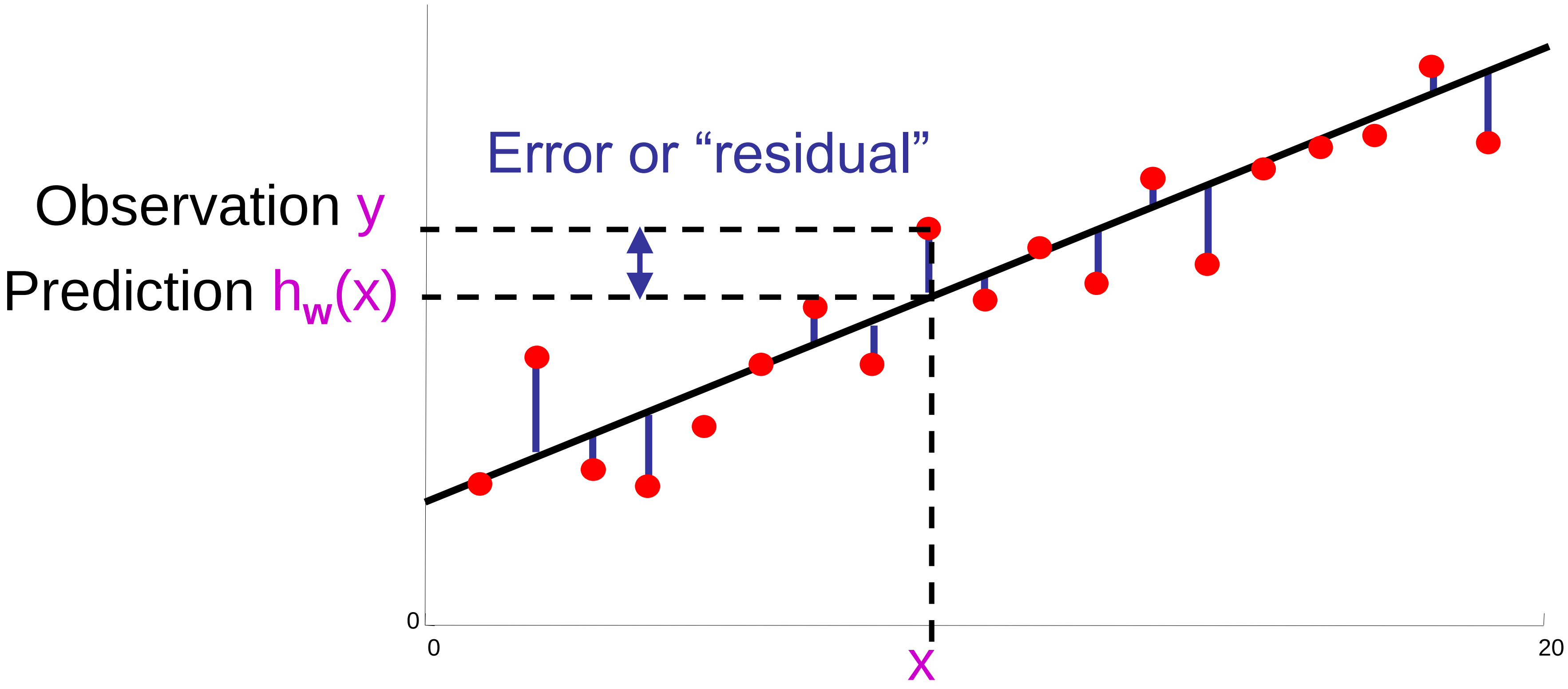
Prediction: $h_w(x) = w_0 + w_1x$



Berkeley house prices, 2009

Prediction error

Error on one instance: $y - h_w(x)$



Least squares: Minimizing squared error

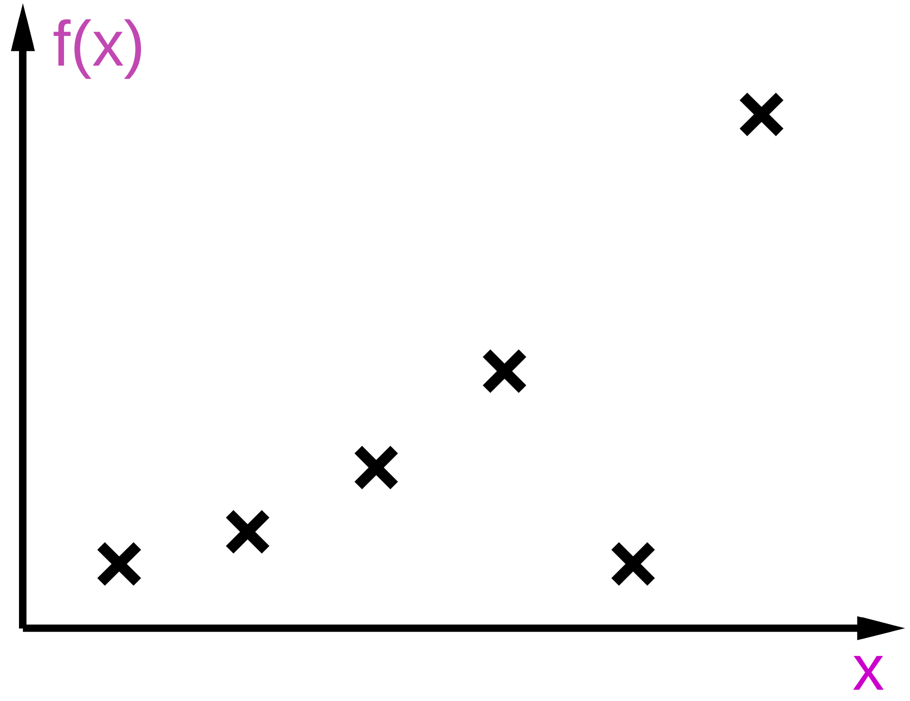
- L2 loss function: sum of squared errors over all examples
  - $Loss = \Sigma_j (y_j - h_{\mathbf{w}}(x_j))^2 = \Sigma_j (y_j - (w_0 + w_1 x_j))^2$

- We want the weights w^* that minimize loss
- At w^* the derivatives of loss w.r.t. each weight are zero:
  - $\partial Loss/\partial w_0 = -2\ \Sigma_j (y_j - (w_0 + w_1 x_j)) = 0$
  - $\partial Loss/\partial w_1 = -2\ \Sigma_j (y_j - (w_0 + w_1 x_j))\, x_j = 0$
- Exact solutions for N examples:
  - $w_1 = [N\Sigma_j x_j y_j - (\Sigma_j x_j)(\Sigma_j y_j)]/[N\Sigma_j x_j^2 - (\Sigma_j x_j)^2]$ and $w_0 = 1/N\ [\Sigma_j y_j - w_1 \Sigma_j x_j]$
- For the general case where $\mathbf{x}$ is an n-dimensional vector
  - $\mathbf{X}$ is the data matrix (all the data, one example per row); $\mathbf{y}$ is the column of labels
  - $\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$

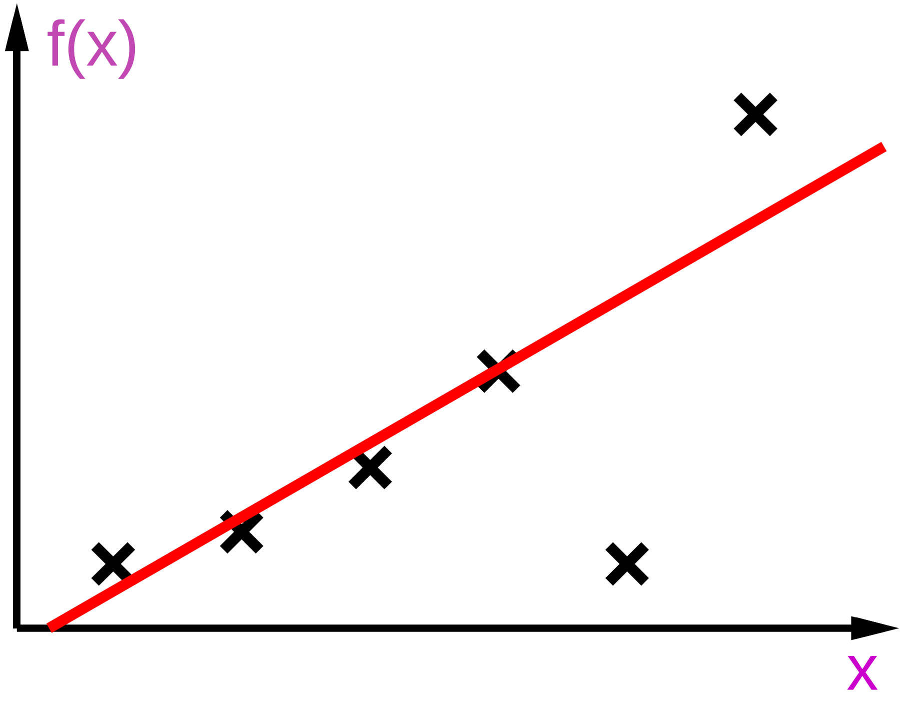
Non-linear least squares

- No closed-form solution in general
- Numerical algorithms are typically used
  - Choose initial values for the parameters and then refine the parameters iteratively
  - Gradient descent
  - Gauss–Newton method
  - Limited-memory BFGS
  - Derivative-free methods
  - etc.

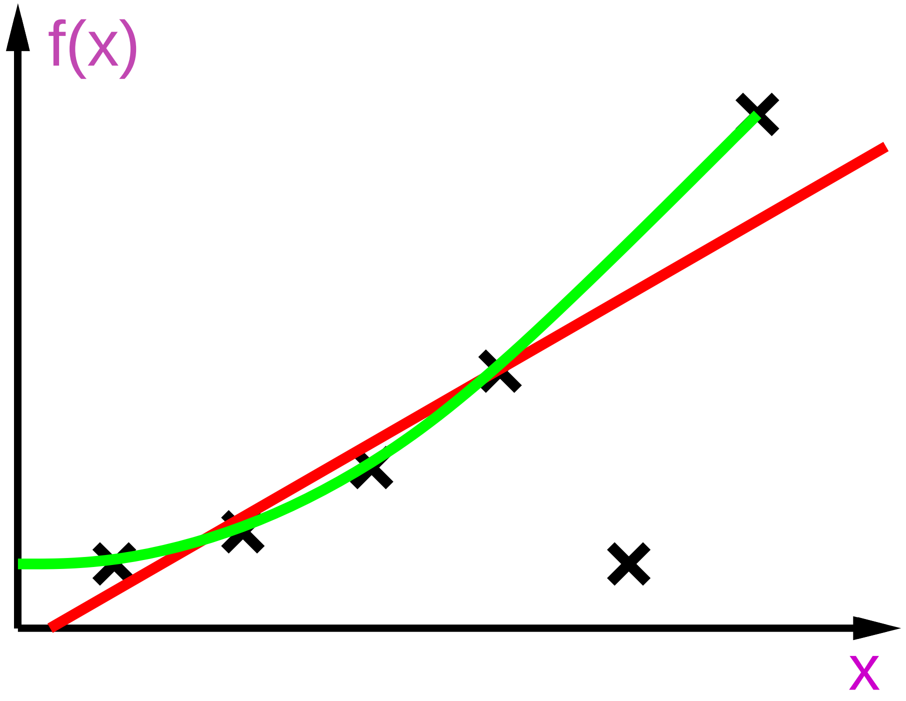
Overfitting in Curve Fitting



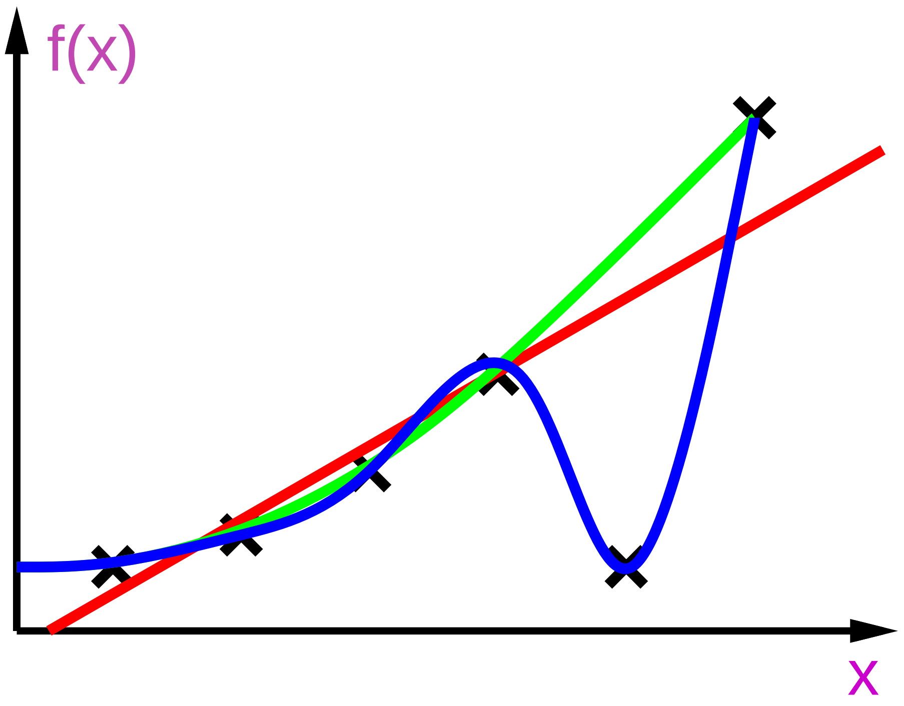
Overfitting in Curve Fitting



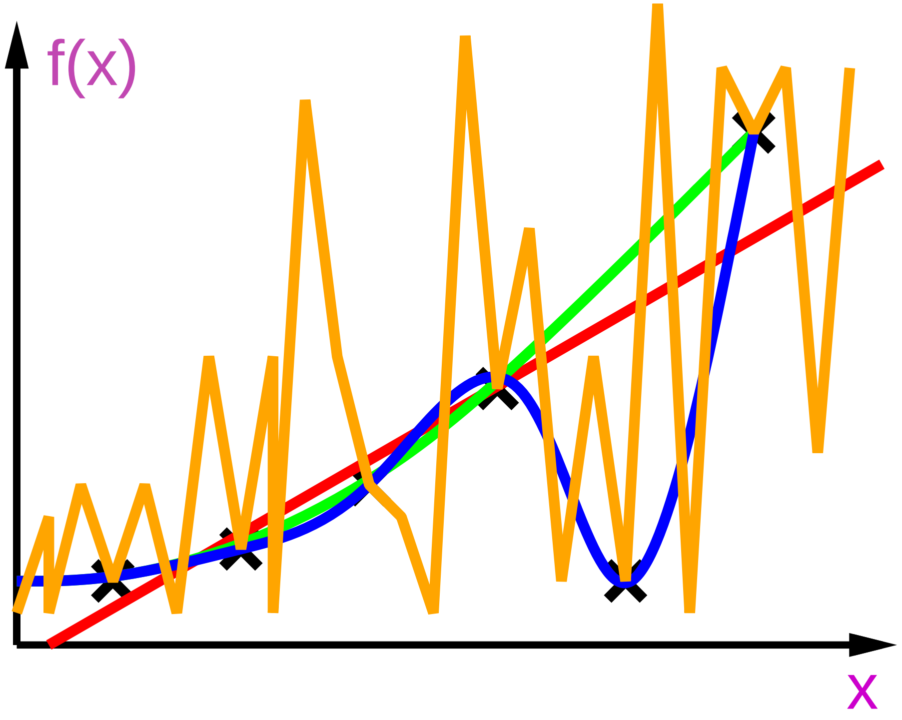
Overfitting in Curve Fitting



Overfitting in Curve Fitting



Overfitting in Curve Fitting



Fit vs. complexity: a tradeoff

"*Ockham's razor*": prefer the *simplest* hypothesis consistent with the data

Summary

- Supervised learning:
  - Learning a function from labeled examples
- Classification: discrete-valued function
  - Naïve Bayes
  - Generalization and overfitting, smoothing
- Regression: real-valued function
  - Linear regression