# Announcement

- Project proposal presentation
  - On Dec. 4 & 6, in class
- Final project presentation
  - On Jan. 16 & 17, 2025
  - Week 18, Thursday & Friday night
  - No class on Dec. 11, 13, 18
- Project evaluation criteria
  - Soundness, substance and depth
  - Relevance to this course
  - Quality of report and presentation

# Announcement

## 上海科技大学2024-2025学年校历

| | 八月 | | 九月 | | | | 十月 | | | | | 十一月 | | | | 十二月 | | | | 一月 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 星期一 | 19 | 26 | 2 | 9 | 16 | 23 | 30 | 7 | 14 | 21 | 28 | 4 | 11 | 18 | 25 | 2 | 9 | 16 | 23 | 30 | 6 | 13 | 20 | 27 | 3 |
| 星期二 | 20 | 27 | 3 | 10 | 17 中秋节 | 24 | 1 国庆节 | 8 | 15 | 22 | 29 | 5 | 12 | 19 | 26 | 3 | 10 | 17 | 24 | 31 | 7 | 14 | 21 | 28 | 4 |
| 星期三 | 21 | 28 | 4 | 11 | 18 | 25 | 2 | 9 | 16 | 23 | 30 | 6 | 13 | 20 | 27 | 4 | | | 25 | 1 元旦 | 8 | 15 | 22 | 29 春节 | 5 |
| 星期四 | 22 | 29 | 5 | 12 | 19 | 26 | 3 | 10 | 17 | 24 | 31 | 7 | 14 | 21 | 28 | 5 | 12 | 19 | 26 | 2 | 9 | 16+ | 23 | 30 | 6 |
| 星期五 | 23 | 30 | 6 | 13 | 20 | 27 | 4 | 11 | 18 | 25 | 1 | 8 | 15 | 22 | 29 | 6 | | 20 | 27 | 3 | 10 | 17+ | 24 | 31 | 7 |
| 星期六 | 24 | 31 | 7 | 14 | 21 | 28 | 5 | 12 | 19 | 26 | 2 | 9 | 16 | 23 | 30 | 7 | 14 | 21 | 28 | 4 | 11 | 18 | 25 | 1 | 8 |
| 星期日 | 25 | 1 | 8 | 15 | 22 | 29 | 6 | 13 | 20 | 27 | 3 | 10 | 17 | 24 | 1 | 8 | 15 | 22 | 29 | 5 | 12 | 19 | 26 | 2 | 9 |
| 周数 | 4 | 5 | 6 | 7 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 1 | 2 | 3 |
| 学期 | 暑假 | | | | 秋学期 | | | | | | | | | | | | | | | | | | 寒假 | | |

○ Project proposal presentation　　○+ Project final presentation　　／ No class

# Proposal Presentation

- Proposal presentation
  - 4-5 min presentation: topic, motivation, possible methods
  - Dec. 4, 6, in class
  - Presentation schedule will be sent out later
- If you have not formed/joined a group, please do so ASAP
- Project evaluation criteria
  - Soundness, substance and depth
  - Relevance to this course
  - Quality of report and presentation

# Supervised Machine Learning

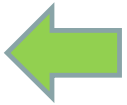

AIMA Chapter 18, 20

[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 for reinforcement learning
- Machine learning: how to acquire a model from data / experience
- Related courses
  - CS182 Introduction to Machine Learning
  - CS282 Machine Learning
  - CS280 Deep Learning

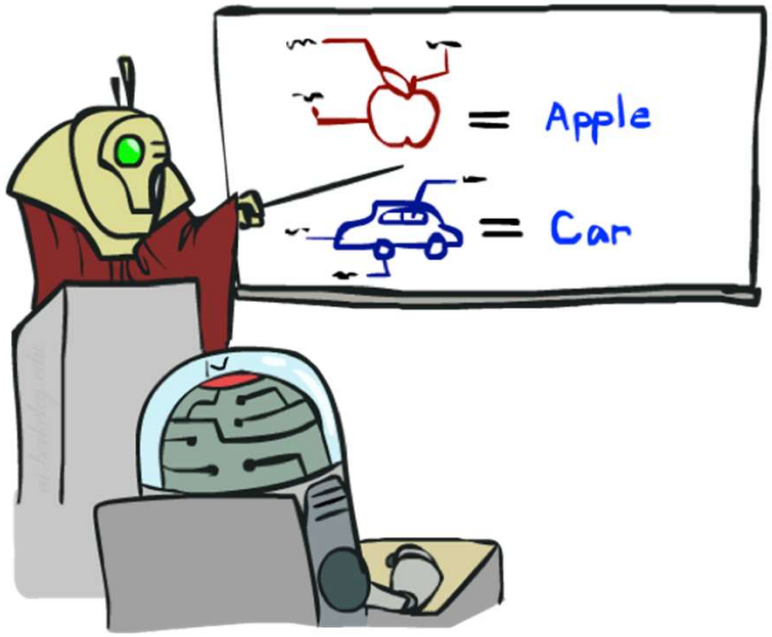
# Types of Learning

- 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

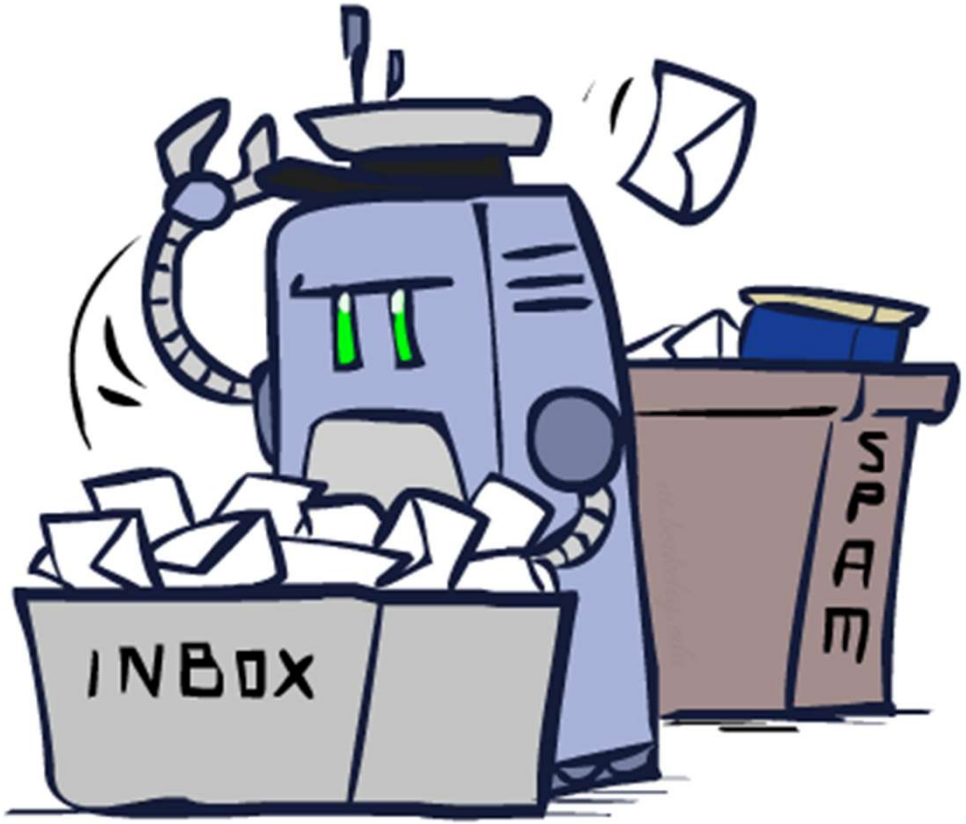
- To learn an unknown *target function* f
- Input: a *training set* of *labeled examples* $(x_j, y_j)$ where $y_j = f(x_j)$
- Output: *hypothesis* h that is "close" to f

Human annotation



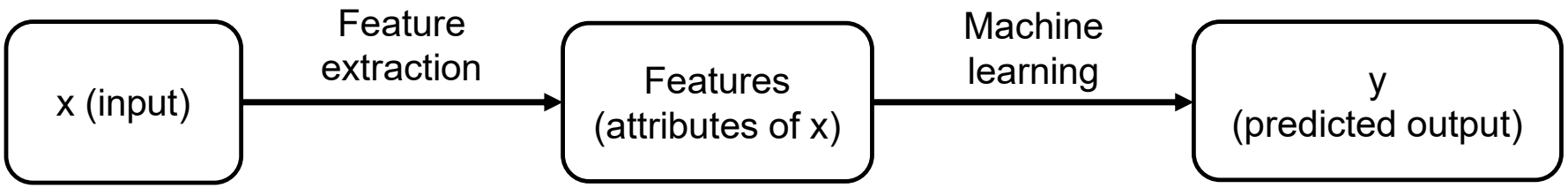

- Types of supervised learning
  - Classification = learning f with discrete output value
  - Regression = learning f with real-valued output value
  - Structured prediction = learning f with structured output

# Classification



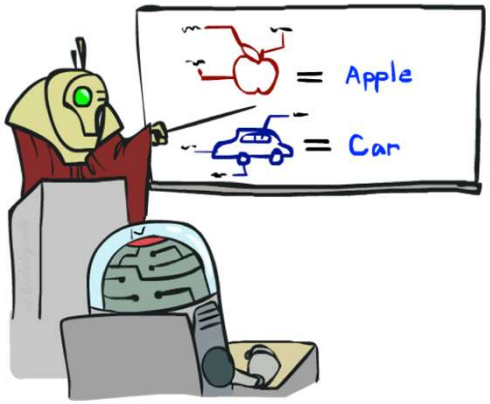
# Classification and Machine Learning

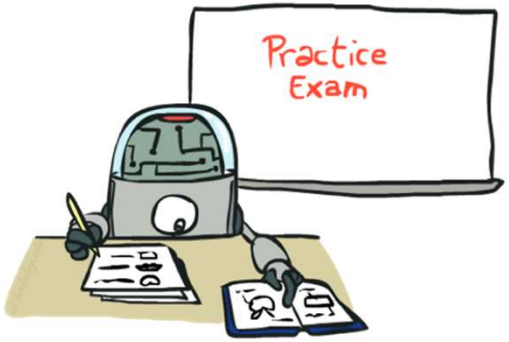
- Dataset: each data point, x, is associated with some label (aka class), y
- Goal of classification: given inputs x, write an algorithm to predict labels y
- Workflow of classification process:
  - Input is provided to you
  - Extract **features** from the input: attributes of the input that characterize each x and hopefully help with classification
  - Run some machine learning algorithm on the features: e.g., Naïve Bayes
  - Output a predicted label y



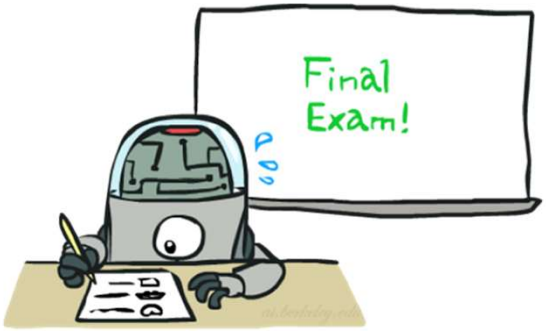
# Training and Machine Learning

- Big idea: ML algorithms learn patterns between features and labels from *data*
  - You don't have to reason about the data yourself
  - You're given **training data**: lots of example datapoints and their actual labels




Training: Learn patterns from labeled data, and periodically test how well you're doing




Eventually, use your algorithm to predict labels for unlabeled data

# 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

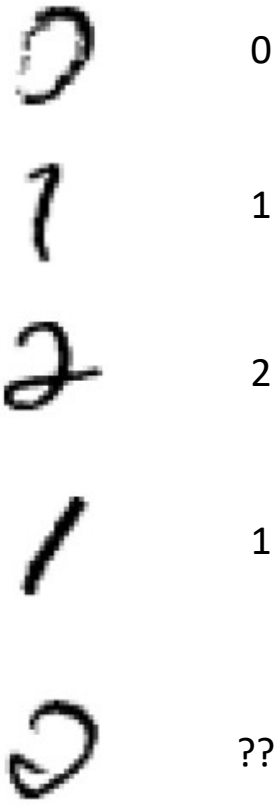
- Features: The attributes used to make the ham / spam decision
  - Words: FREE!
  - Text Patterns: $dd, CAPS
  - Non-text: SenderInContacts
  - …

# 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
  - Note: someone has to hand label all this data!
  - Want to learn to predict labels of new, future digit images

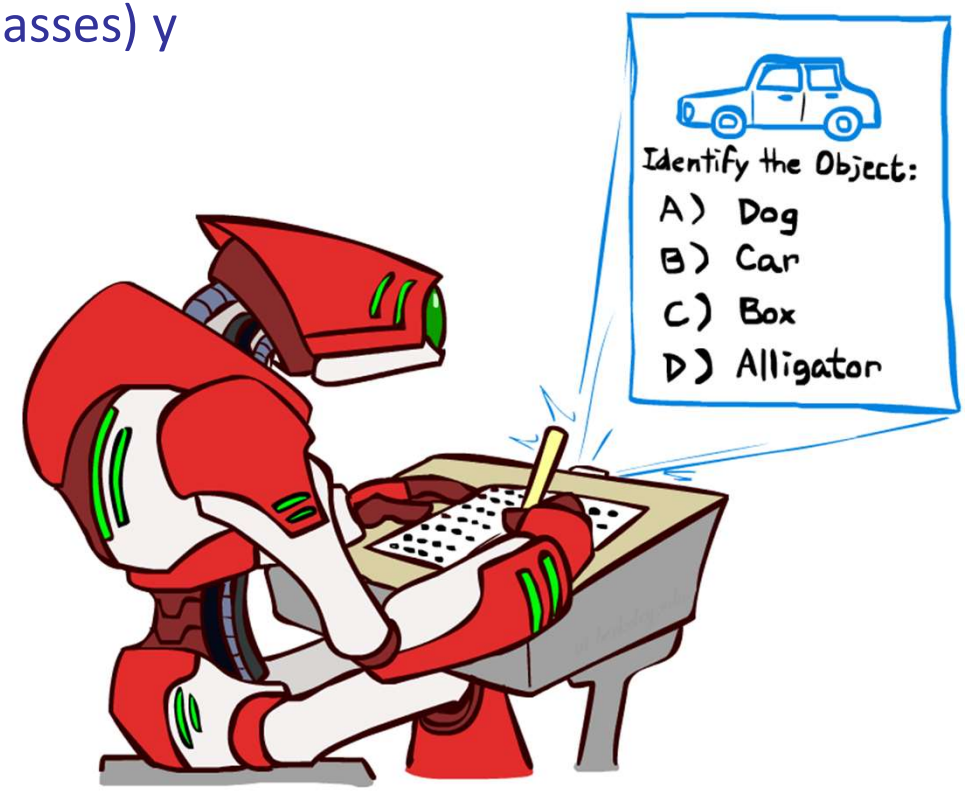
- Features: The attributes used to make the digit decision
  - Pixels: (6,8)=ON
  - Shape Patterns: NumComponents, AspectRatio, NumLoops
  - …
  - Features are increasingly induced rather than crafted



# Other Classification Tasks

- Classification: given inputs x, predict labels (classes) y

- Examples:
  - Medical diagnosis (input: symptoms, classes: diseases)
  - Fraud detection (input: account activity, classes: fraud / no fraud)
  - Automatic essay grading (input: document, classes: grades)
  - Customer service email routing
  - Review sentiment
  - Language ID
  - … many more




- Classification is an important commercial technology!

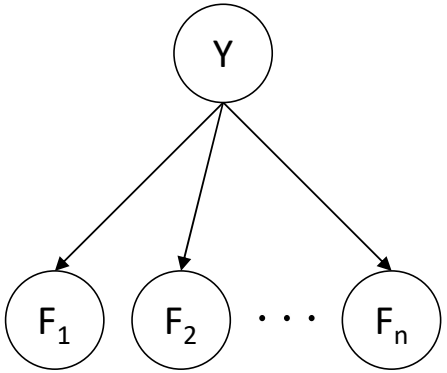
# Model-Based Classification

# Model-Based Classification

- Model-based approach
  - Build a model (e.g. Bayes net) where both the output label and input 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?

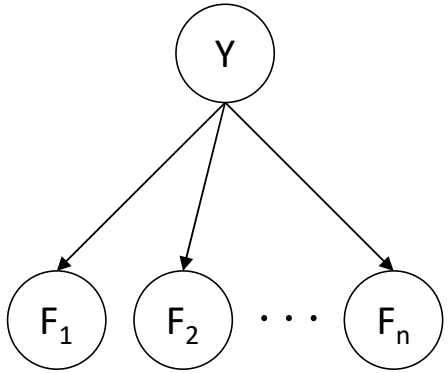
# Naïve Bayes Model

- Random variables in this Bayes net:
  - Y = The label
  - $F_1$, $F_2$, …, $F_n$ = The n features
- Probability tables in this Bayes net:
  - P(Y) = Probability of each label occurring, given no information about the features. Sometimes called the *prior*.
  - $P(F_i|Y)$ = One table per feature. Probability distribution over a feature, given the label.



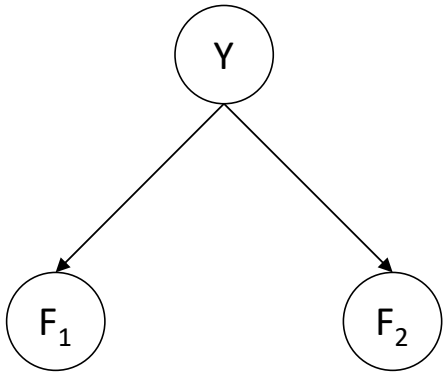
# Naïve Bayes Model

- To perform training:
  - Use the training dataset to estimate the probability tables.
  - Estimate P(Y) = how often does each label occur?
  - Estimate $P(F_i|Y)$ = how does the label affect the feature?
- To perform classification:
  - Instantiate all features. You know the input features, so they're your evidence.
  - Query for $P(Y|f_1, f_2, \ldots, f_n)$. Probability of label, given all the input features. Use an inference algorithm (e.g. variable elimination) to compute this.



# Example: Naïve Bayes for Spam Filter

- Step 1: Select a ML algorithm. We choose to model the problem with Naïve Bayes.
- Step 2: Choose features to use.




| Y: The label (spam or ham) | |
|---|---|
| Y | P(Y) |
| ham | ? |
| spam | ? |

| $F_1$: A feature (do I know the sender?) | | |
|---|---|---|
| $F_1$ | Y | $P(F_1 \mid Y)$ |
| yes | ham | ? |
| no | ham | ? |
| yes | spam | ? |
| no | spam | ? |

| $F_2$: Another feature (# of occurrences of FREE) | | |
|---|---|---|
| $F_2$ | Y | $P(F_2 \mid Y)$ |
| 0 | ham | ? |
| 1 | ham | ? |
| 2 | ham | ? |
| 0 | spam | ? |
| 1 | spam | ? |
| 2 | spam | ? |

# Example: Naïve Bayes for Spam Filter

- Step 3: Training: Use training data to fill in the probability tables.

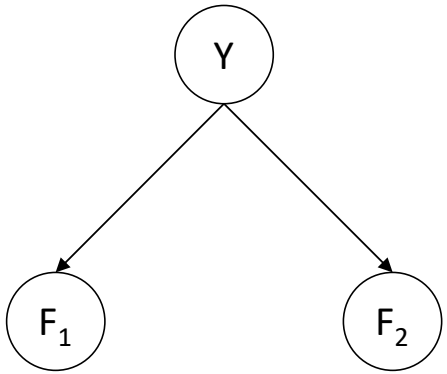
| $F_2$: # of occurrences of FREE | | |
|---|---|---|
| $F_2$ | Y | $P(F_2 \mid Y)$ |
| 0 | ham | 0.5 |
| 1 | ham | 0.5 |
| 2 | ham | 0.0 |
| 0 | spam | 0.25 |
| 1 | spam | 0.50 |
| 2 | spam | 0.25 |

| Training Data | | |
|---|---|---|
| # | Email Text | Label |
| 1 | Attached is my portfolio. | ham |
| 2 | Are you **free** for a meeting tomorrow? | ham |
| 3 | **Free** unlimited credit cards!!!! | spam |
| 4 | Mail $10,000 check to this address | spam |
| 5 | Sign up now for 1 **free** Bitcoin | spam |
| 6 | **Free** money **free** money | spam |

Row 4: $P(F_2=0 \mid Y=\text{spam}) = 0.25$ because 1 out of 4 spam emails contains “free” 0 times.
Row 5: $P(F_2=1 \mid Y=\text{spam}) = 0.50$ because 2 out of 4 spam emails contains “free” 1 time.
Row 6: $P(F_2=2 \mid Y=\text{spam}) = 0.25$ because 1 out of 4 spam emails contains “free” 2 times.

# Example: Naïve Bayes for Spam Filter

- Model trained on a larger dataset:



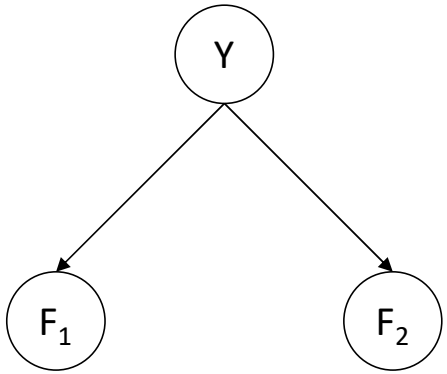

| Y: The label (spam or ham) | |
|---|---|
| Y | P(Y) |
| ham | 0.6 |
| spam | 0.4 |

| $F_1$: A feature (do I know the sender?) | | |
|---|---|---|
| $F_1$ | Y | $P(F_1 \mid Y)$ |
| yes | ham | 0.7 |
| no | ham | 0.3 |
| yes | spam | 0.1 |
| no | spam | 0.9 |

| $F_2$: Another feature (# of occurrences of FREE) | | |
|---|---|---|
| $F_2$ | Y | $P(F_2 \mid Y)$ |
| 0 | ham | 0.85 |
| 1 | ham | 0.07 |
| 2 | ham | 0.08 |
| 0 | spam | 0.75 |
| 1 | spam | 0.12 |
| 2 | spam | 0.13 |

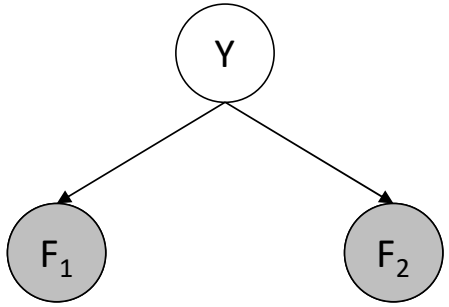
# Example: Naïve Bayes for Spam Filter

- Step 4: Classification
- Suppose you want to label this email from a known sender: "**Free** food in Soda 430 today"
- Step 4.1: Feature extraction:
  - $F_1$ = yes, known sender
  - $F_2$ = 1 occurrence of "free"



# Example: Naïve Bayes for Spam Filter

- Step 4.2: Inference
- Instantiate features (evidence):
  - $F_1$ = yes
  - $F_2$ = 1
- Compute joint probabilities:
  - P(Y = spam, $F_1$ = yes, $F_2$ = 1) = P(Y = spam) P($F_1$ = yes | spam) P($F_2$ = 1 | spam)
    = 0.4 * 0.1 * 0.12 = 0.0048
  - P(Y = ham, $F_1$ = yes, $F_2$ = 1) = P(Y = ham) P($F_1$ = yes | ham) P($F_2$ = 1 | ham)
    = 0.6 * 0.7 * 0.07 = 0.0294
- Normalize:
  - P(Y = spam | $F_1$ = yes, $F_2$ = 1) = 0.0048 / (0.0048+0.0294) = 0.14
  - P(Y = ham | $F_1$ = yes, $F_2$ = 1) = 0.0294 / (0.0048+0.0294) = 0.86
- Classification result:
  - 14% chance the email is spam. 86% chance it's ham.
  - Or, if you don't need probabilities, note that 0.0294 > 0.0048 and guess ham.




| Y: The label (spam or ham) | |
|---|---|
| Y | P(Y) |
| ham | 0.6 |
| spam | 0.4 |

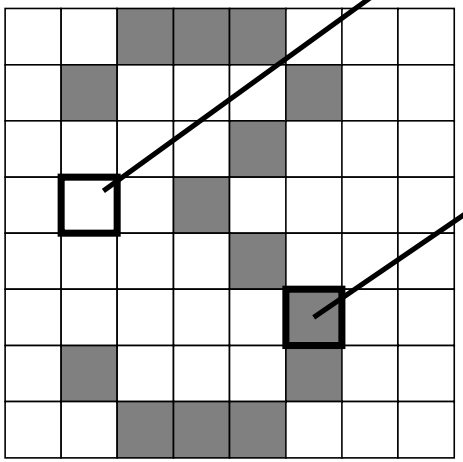
| $F_1$: do I know the sender? | | |
|---|---|---|
| $F_1$ | Y | P($F_1$\|Y) |
| yes | ham | 0.7 |
| no | ham | 0.3 |
| yes | spam | 0.1 |
| no | spam | 0.9 |

| $F_2$: # of occurrences of FREE | | |
|---|---|---|
| $F_2$ | Y | P($F_2$\|Y) |
| 0 | ham | 0.85 |
| 1 | ham | 0.07 |
| 2 | ham | 0.08 |
| 0 | spam | 0.75 |
| 1 | spam | 0.12 |
| 2 | spam | 0.13 |

# Example: Conditional Probabilities

$P(Y)$

| | |
|---|---|
| 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 |



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

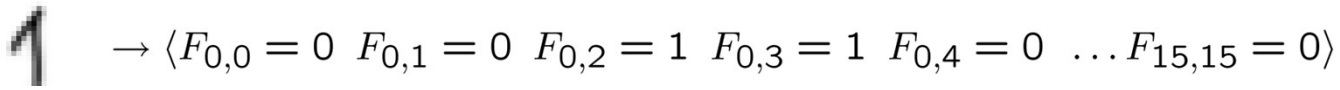
| | |
|---|---|
| 1 | 0.01 |
| 2 | 0.05 |
| 3 | 0.05 |
| 4 | 0.30 |
| 5 | 0.80 |
| 6 | 0.90 |
| 7 | 0.05 |
| 8 | 0.60 |
| 9 | 0.50 |
| 0 | 0.80 |

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

| | |
|---|---|
| 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 |

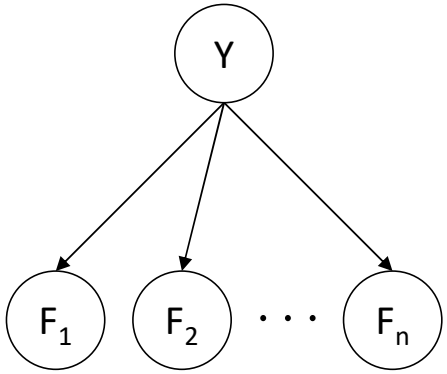
# Naïve Bayes for Digits

- 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.



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

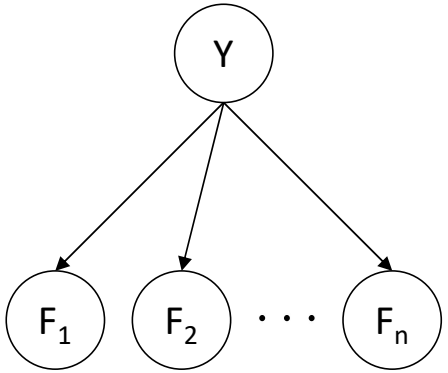
  - Here: lots of features, each is binary valued




- Naïve Bayes model: $P(Y|F_{0,0} \ldots F_{15,15}) \propto P(Y) \prod_{i,j} P(F_{i,j}|Y)$
- What do we need to learn?

# General Naïve Bayes

- Naïve Bayes assumes that all features are independent effects of the label
- A general Naive Bayes model:




|Y| parameters

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

|Y| x |F|$^n$ values

n x |F| x |Y| parameters

- We only have to specify how each feature depends on the class
- Total number of parameters is *linear* in n
- Model is very simplistic, but often works anyway

# Inference for Naïve Bayes

- Goal: compute posterior distribution over label variable Y
  - Step 1: get joint probability of label and evidence for each label

$$P(Y, f_1 \dots f_n) = \begin{bmatrix} P(y_1, f_1 \dots f_n) \\ P(y_2, f_1 \dots f_n) \\ \vdots \\ P(y_k, f_1 \dots f_n) \end{bmatrix} \Rightarrow \begin{bmatrix} P(y_1) \prod_i P(f_i|y_1) \\ P(y_2) \prod_i P(f_i|y_2) \\ \vdots \\ P(y_k) \prod_i P(f_i|y_k) \end{bmatrix}$$

$$\frac{\begin{bmatrix} P(y_1) \prod_i P(f_i|y_1) \\ P(y_2) \prod_i P(f_i|y_2) \\ \vdots \\ P(y_k) \prod_i P(f_i|y_k) \end{bmatrix}}{P(f_1 \dots f_n)}$$

+

  - Step 2: sum to get probability of evidence

$$\Downarrow$$

  - Step 3: normalize by dividing Step 1 by Step 2

$$P(Y|f_1 \dots f_n)$$

# Naïve Bayes for Text

- Naïve Bayes spam filter
- Data:
  - Collection of emails, labeled spam or ham
  - Note: someone has to hand label all this data!
  - Split into training, held-out, test sets
- Classifiers
  - Learn on the training set
  - (Tune it on a held-out set)
  - Test it on new emails

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.

# Naïve Bayes for Text

- Bag-of-words Naïve Bayes:
  - Features: $W_i$ is the word at position 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!*

- "Tied" distributions and bag-of-words
  - Usually, each variable gets its own conditional probability distribution P(F|Y)
  - In a bag-of-words model
    - Each position is identically distributed
    - All positions share the same conditional probs P(W|Y)
    - Why make this assumption?
  - Called "bag-of-words" because model is insensitive to word order or reordering

# Example: Spam Filtering

- Model: $P(Y, W_1 \dots W_n) = P(Y) \prod_i P(W_i|Y)$
- What are the parameters?

$P(Y)$

| | |
|---|---|
| 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 |
| ... | |

- Where do these tables come from?

# Spam Example

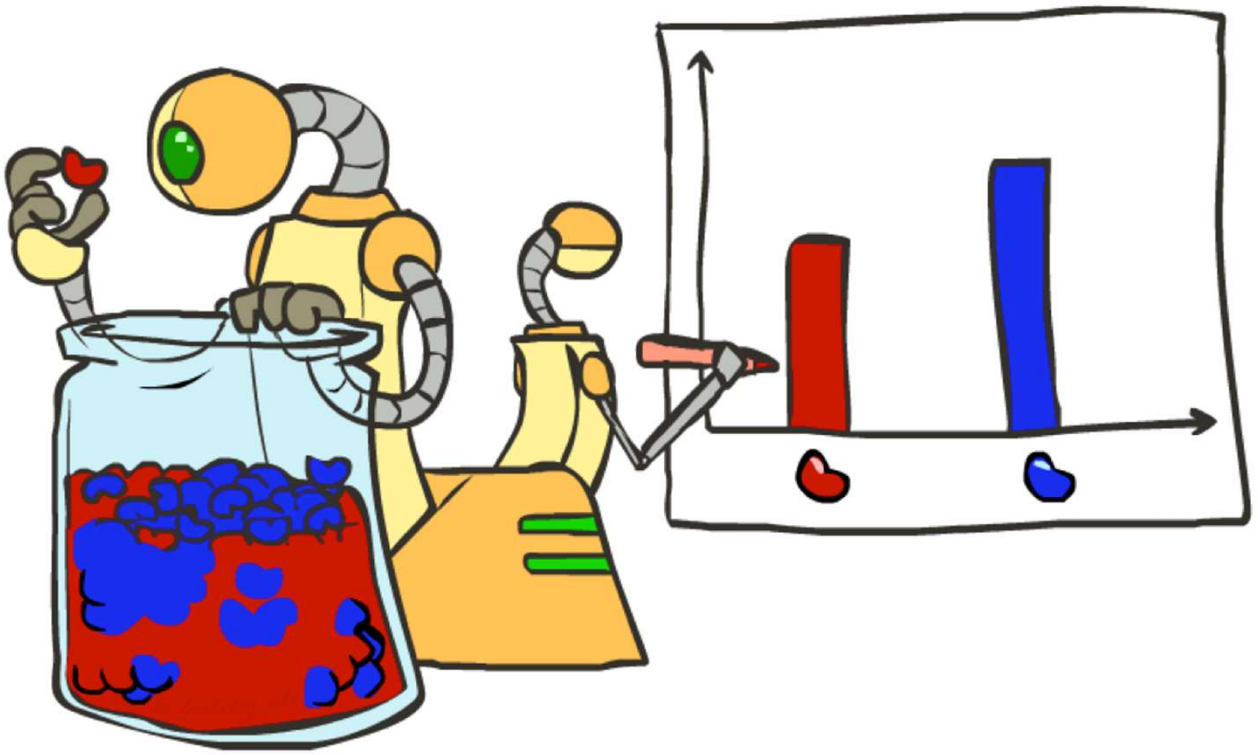
| Word | P(w\|spam) | P(w\|ham) | Tot Spam | Tot Ham |
|---|---|---|---|---|
| (prior) | 0.33333 | 0.66666 | -1.1 | -0.4 |

P(spam | w) = 98.9

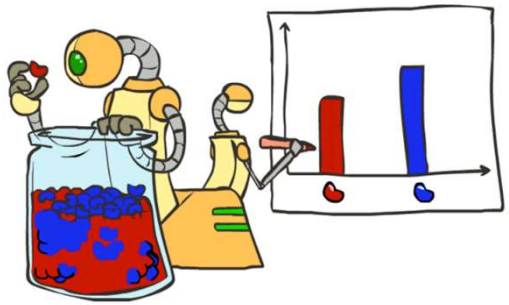
# General Naïve Bayes

- What do we need in order to use Naïve Bayes?
  - Inference method
    - Start with a bunch of probabilities: P(Y) and the $P(F_i|Y)$ tables
    - Use standard inference to compute $P(Y|F_1 \ldots F_n)$
    - Nothing new here
  - 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 $\theta$
    - Up until now, we assumed these appeared by magic, but they typically come from *training data counts*

# Parameter Estimation



# Parameter Estimation



- Estimating the distribution of a random variable
- *Elicitation:* ask a human (why is this hard?)
- *Empirically:* use training data (learning!)
  - Example: The parameter θ is the true fraction of red beans in the jar. You don't know θ but would like to estimate it.
  - Collecting training data: You randomly pull out 3 beans:

    r r b

  - Estimating θ using counts, you guess 2/3 of beans in the jar are red.
  - Can we mathematically show that using counts is the "right" way to estimate θ?

# Parameter Estimation with Maximum Likelihood

- **Can we mathematically show that using counts is the "right" way to estimate θ?**
- Maximum likelihood estimation: Choose the θ value that maximizes the probability of the observation
  - In other words, choose the θ value that maximizes P(observation | θ)
  - For our problem:

    P(observation | θ)

    = P(randomly selected 2 red and 1 blue | θ of beans are red)

    = P(red | θ) P(red | θ) P(blue | θ)

    = $\theta^2 (1- \theta)$

  - We want to compute:

    $\underset{\theta}{\text{argmax}}\ \theta^2 (1- \theta)$

# Parameter Estimation with Maximum Likelihood

- We want to compute:

  $$\underset{\theta}{\text{argmax}}\ \theta^2 (1- \theta)$$

- Set derivative to 0, and solve!
  - Common issue: The likelihood (expression we're maxing) is the product of a lot of probabilities. This can lead to complicated derivatives.
  - Solution: Maximize the log-likelihood instead. Useful fact:

    $$\underset{\theta}{\text{argmax}}\ f(\theta) = \underset{\theta}{\text{argmax}}\ \ln f(\theta)$$

# Parameter Estimation with Maximum Likelihood

$$\operatorname*{argmax}_{\theta} \theta^2(1-\theta)$$

Find θ that maximizes likelihood

$$= \operatorname*{argmax}_{\theta} \ln\left(\theta^2(1-\theta)\right)$$

Find θ that maximizes log-likelihood (will be the same θ)

$$\frac{d}{d\theta} \ln\left(\theta^2(1-\theta)\right) = 0$$

Set derivative to 0

$$\frac{d}{d\theta}\left[\ln(\theta^2) + \ln(1-\theta)\right] = 0$$

Logarithm rule: products become sums

$$\frac{d}{d\theta}\left[2\ln(\theta) + \ln(1-\theta)\right] = 0$$

Logarithm rule: exponentiation becomes multiplication

$$\frac{d}{d\theta} 2\ln(\theta) + \frac{d}{d\theta}\ln(1-\theta) = 0$$

Now we can derive each term of the original product separately

$$\frac{2}{\theta} - \frac{1}{1-\theta} = 0$$

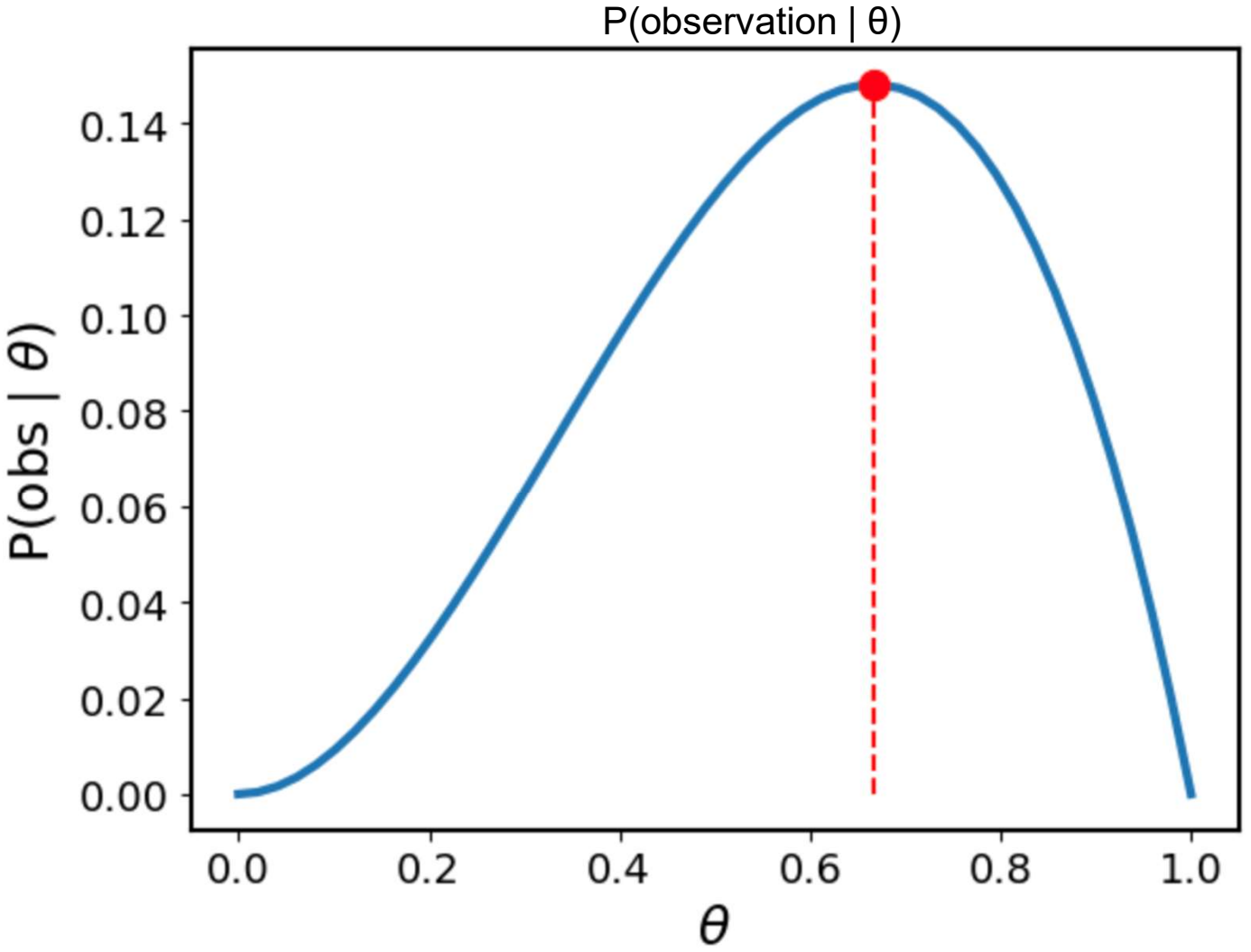
Reminder: Derivative of ln(θ) is 1/θ

$$\theta = \frac{2}{3}$$

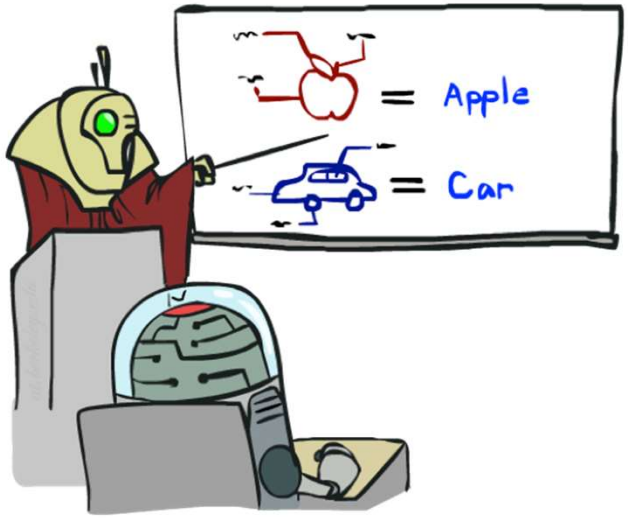
Use algebra to solve for θ. If we used arbitrary red and blue counts r and b instead of r=2 and b=1, we'd get θ = r / (r+b), the count estimate.

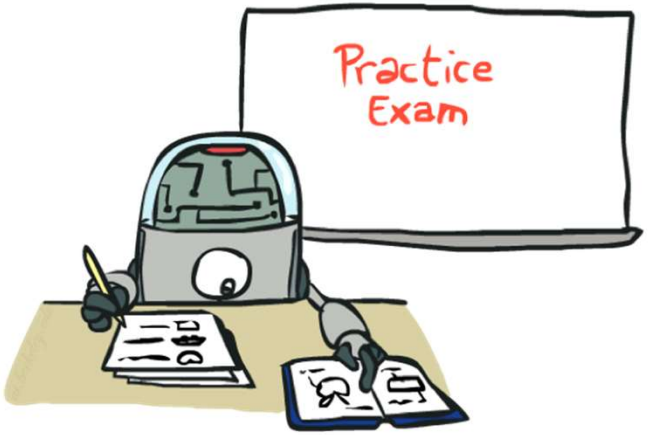
# Parameter Estimation with Maximum Likelihood

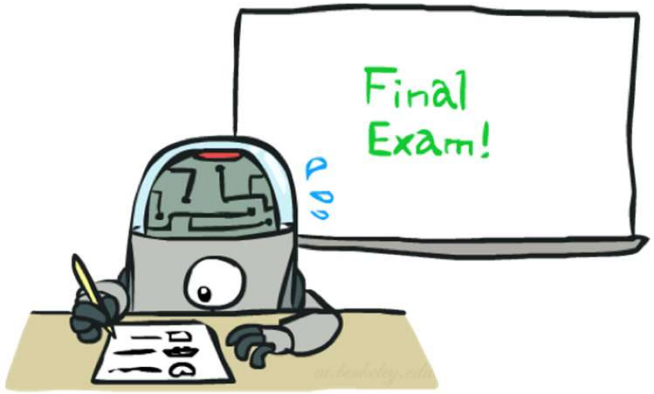
$$
\begin{aligned}
& \underset{\theta}{\operatorname{argmax}}\, \theta^2(1-\theta) \\
= \; & \underset{\theta}{\operatorname{argmax}} \ln\left(\theta^2(1-\theta)\right) \\
& \frac{d}{d\theta} \ln\left(\theta^2(1-\theta)\right) = 0 \\
& \frac{d}{d\theta}\left[\ln(\theta^2) + \ln(1-\theta)\right] = 0 \\
& \frac{d}{d\theta}\left[2\ln(\theta) + \ln(1-\theta)\right] = 0 \\
& \frac{d}{d\theta} 2\ln(\theta) + \frac{d}{d\theta}\ln(1-\theta) = 0 \\
& \frac{2}{\theta} - \frac{1}{1-\theta} = 0 \\
& \theta = \frac{2}{3}
\end{aligned}
$$



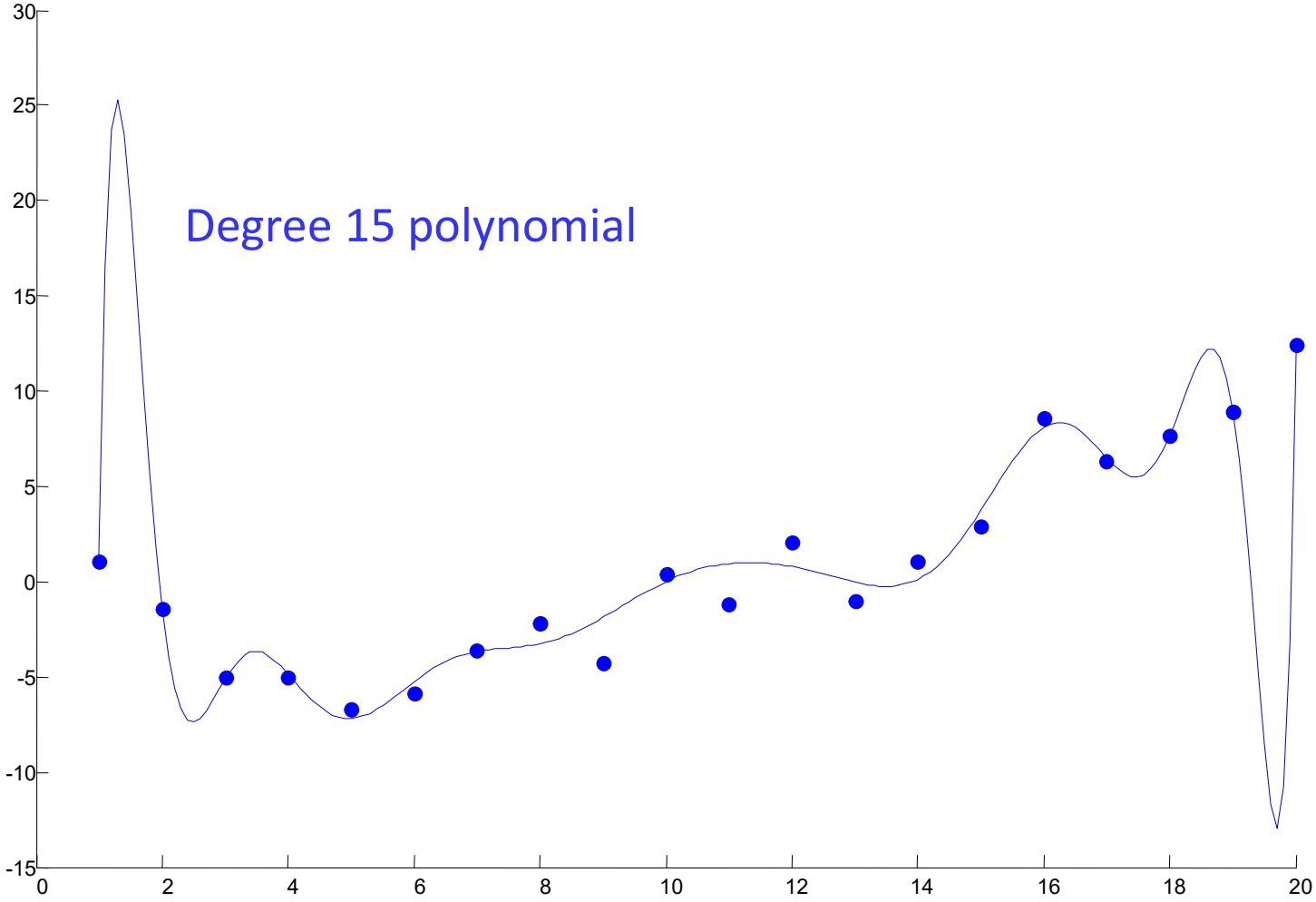
# Training and Testing

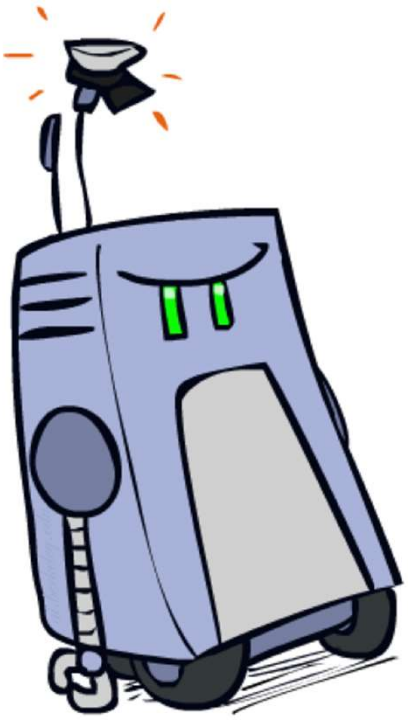


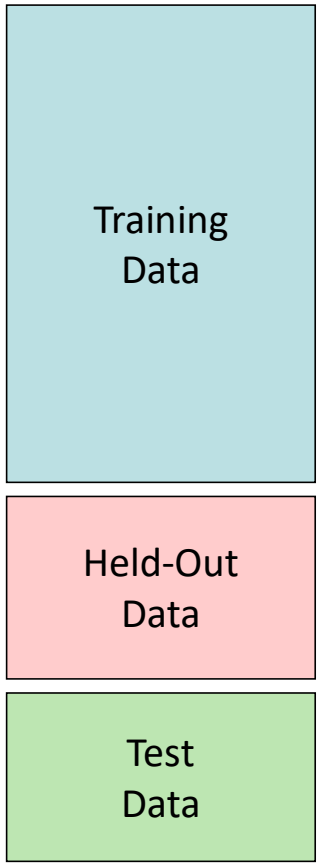
# Overfitting

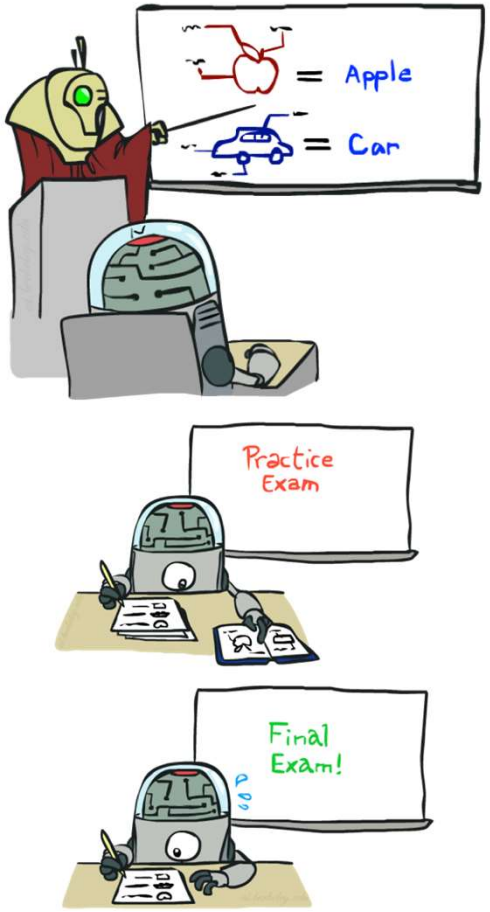


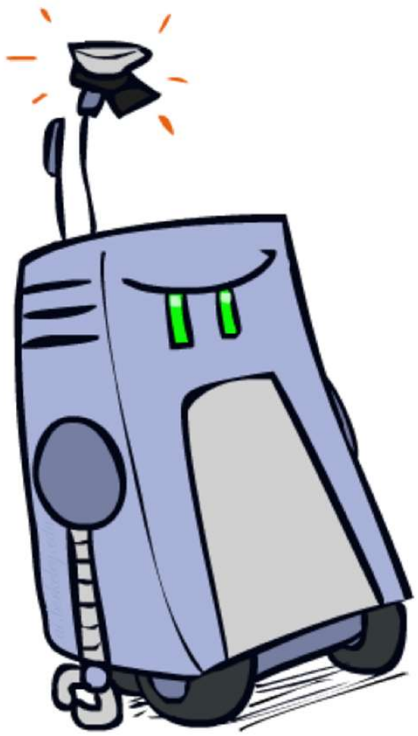
# Important Concepts

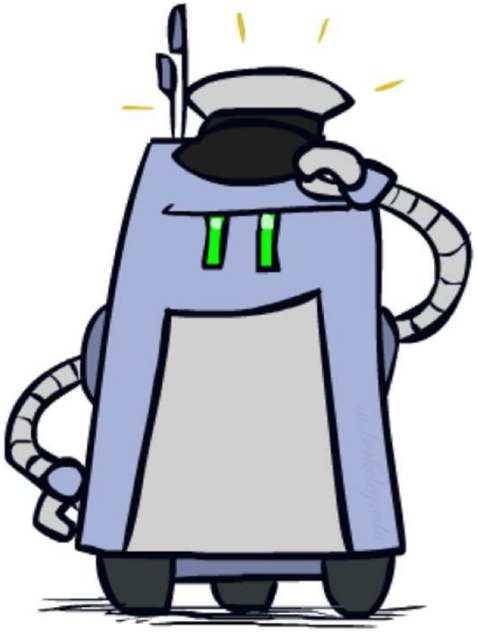
- How do we check that we're not overfitting during training?
- Split training data into 3 different sets:
  - Training set
  - Held out set (more on this later)
  - Test set
- Experimentation cycle
  - Learn parameters (e.g. model probabilities) on training set
  - Compute accuracy of test set
  - Very important: never "peek" at the test set!
- Evaluation (many metrics possible, e.g. accuracy)
  - Accuracy: fraction of instances predicted correctly
- Overfitting and generalization
  - Want a classifier which does well on *test* data
  - Overfitting: fitting the training data very closely, but not generalizing well
  - We'll investigate overfitting and generalization formally in a few lectures



# Generalization and Overfitting





# Example: Overfitting

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

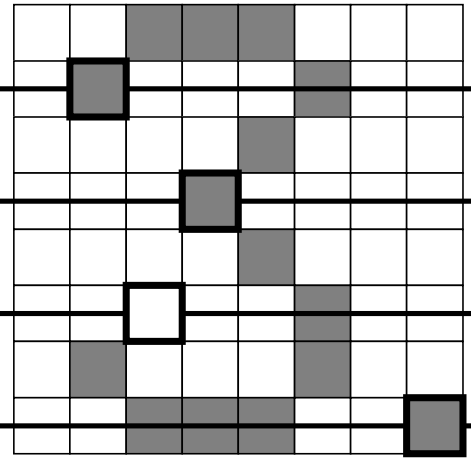
$P(C = 2) = 0.1$

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

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

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

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



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

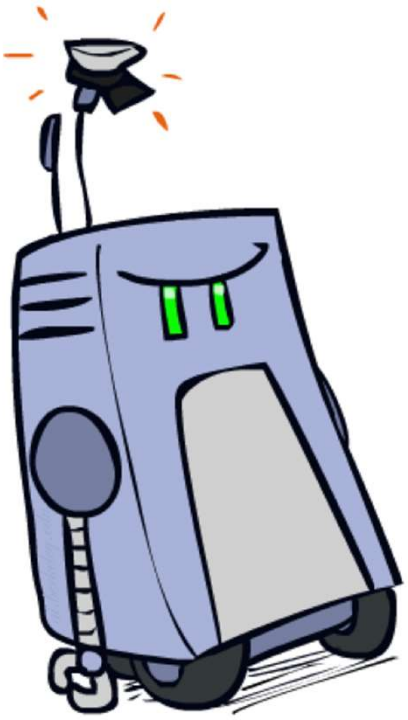
$P(C = 3) = 0.1$

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

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

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

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



*2 wins!!*

# Example: Overfitting

- Posteriors determined by *relative* probabilities (odds ratios):

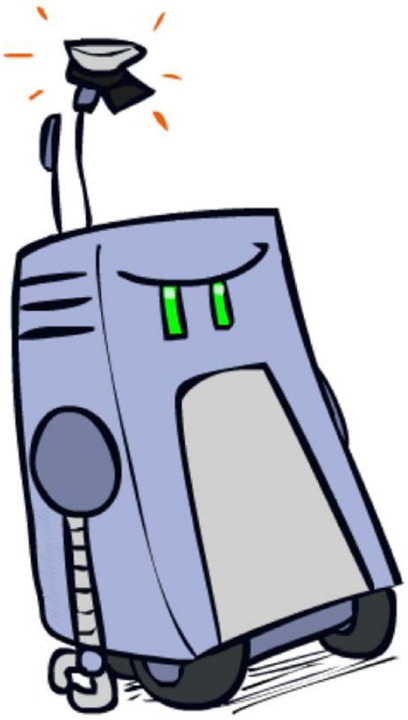
$$\frac{P(W|\text{ham})}{P(W|\text{spam})}$$

```
south-west : inf
nation     : inf
morally    : inf
nicely     : inf
extent     : inf
seriously  : inf
...
```

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

```
screens    : inf
minute     : inf
guaranteed : inf
$205.00    : inf
delivery   : inf
signature  : inf
...
```

*What went wrong here?*



# Generalization and Overfitting

- Relative empirical rate will overfit the training data!
  - Just because we never *saw a 3 with pixel (8,8) 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
  - In general, we can't go around giving unseen events zero probability
  - More generally, rates in the training data may not exactly match rates at test time

- As an extreme case, imagine using the entire email as the only feature (e.g. document ID)
  - Would get the training data perfect (if deterministic labeling)
  - Wouldn't *generalize* at all
  - Just making the bag-of-words assumption gives us some generalization, but isn't enough

# Generalization and Overfitting

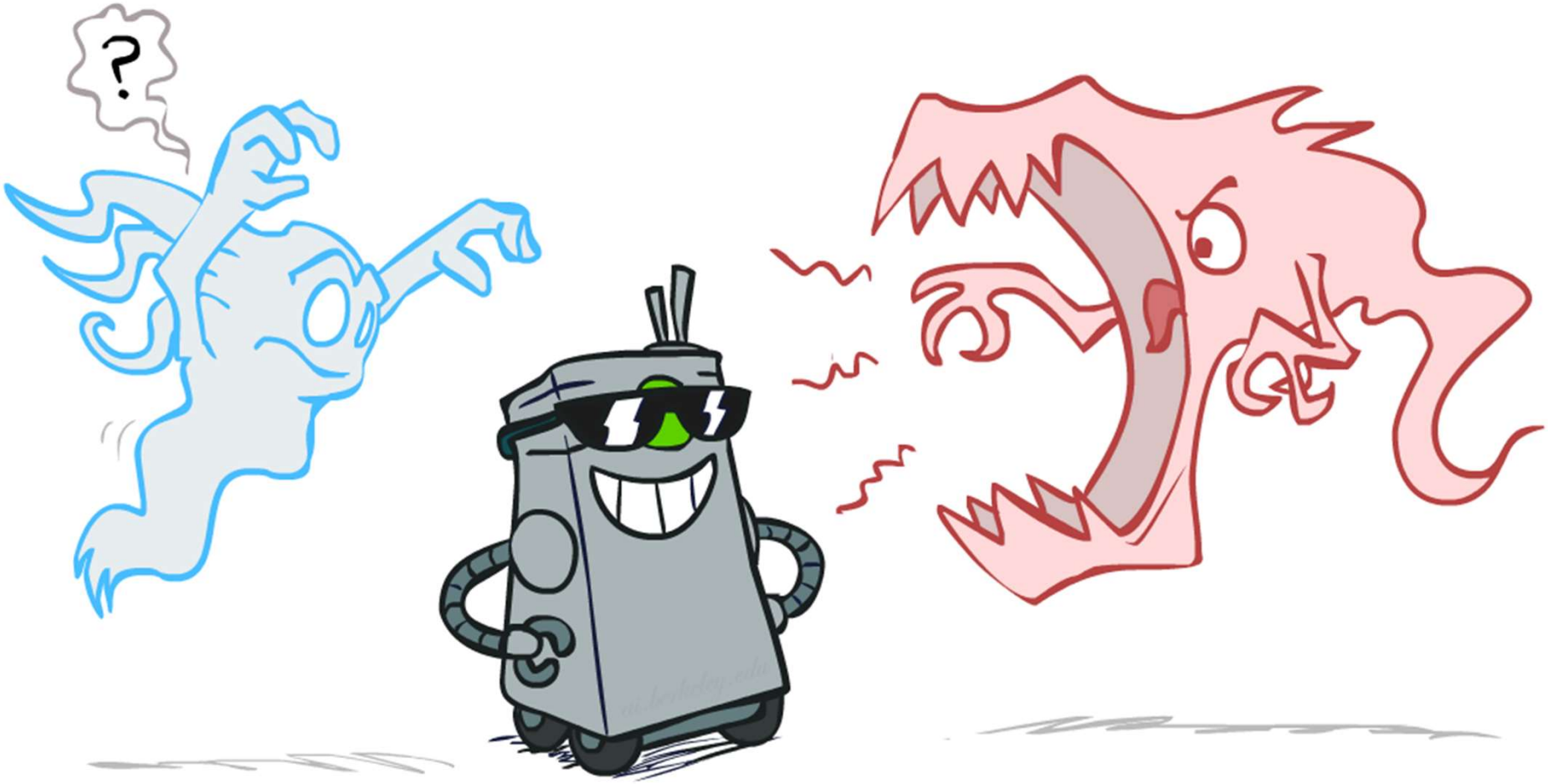
- Overfitting: learn to fit the training data very closely, but fit the test data poorly
  - Generalization: try to fit the test data as well
- Why does overfitting occur?
  - Training data is not representative of the true data distribution
    - Too few training samples
    - Training data is noisy
    - Test set is out of distribution (OOD) of training set
  - Too many attributes, some of them irrelevant to the classification task
  - The model is too expressive
    - Ex: the model is capable of memorizing all the spam emails in the training set

# Generalization and Overfitting

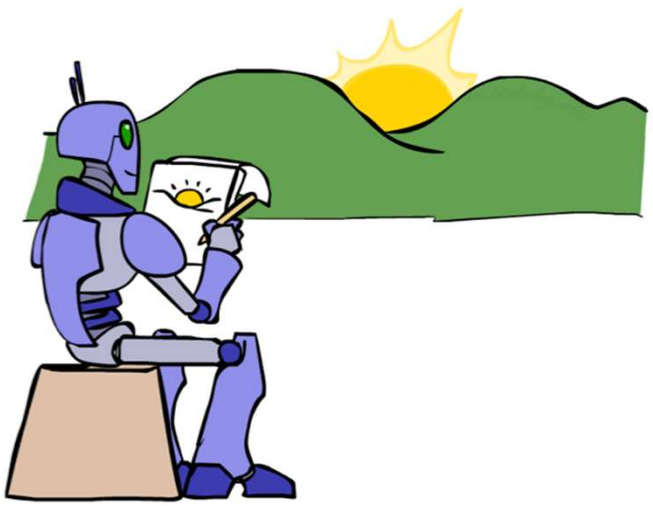
- Avoid overfitting
  - Acquire more training data (not always possible)
  - Remove irrelevant attributes (not always possible)
  - Limit the model expressiveness by regularization, early stopping, pruning, etc.

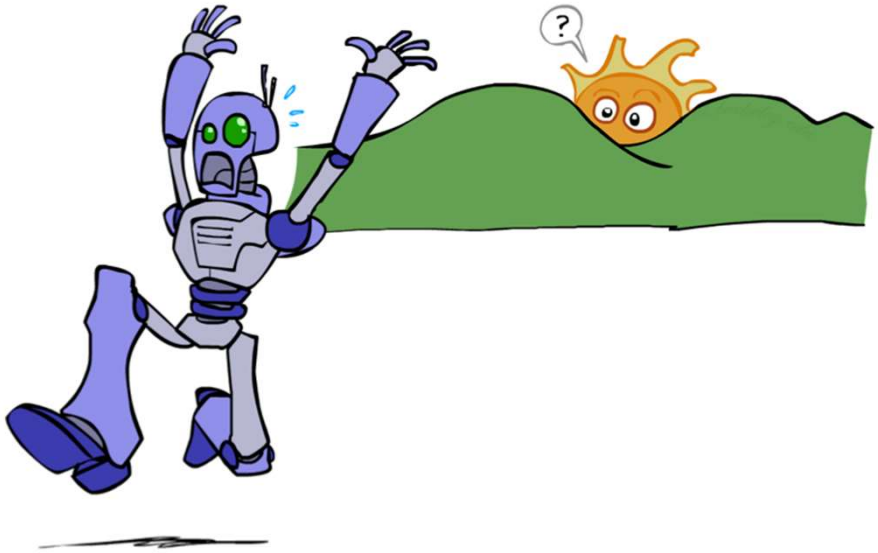
- In our previous example, we may smooth or regularize the empirical estimation to improve generalization

# Smoothing



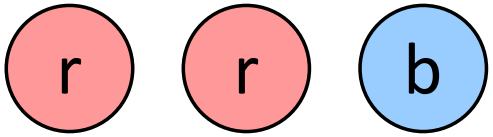
# Unseen Events





# Laplace Smoothing

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




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

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

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

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

# Laplace Smoothing

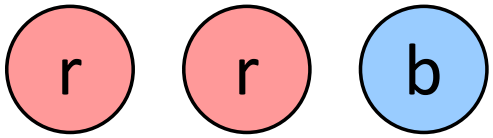
- Laplace's estimate (extended):
  - Pretend you saw every outcome k extra times

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

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

- Laplace for conditionals:
  - Smooth each condition independently:

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




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

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

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

# Real Naïve Bayes: Smoothing

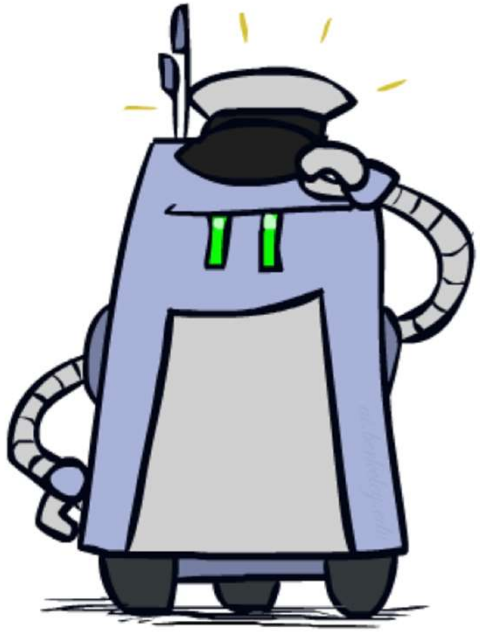
- For real classification problems, smoothing is critical
- New odds ratios:

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

```
helvetica : 11.4
seems     : 10.8
group     : 10.2
ago       :  8.4
areas     :  8.3
...
```

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

```
verdana : 28.8
Credit  : 28.4
ORDER   : 27.2
<FONT>  : 26.9
money   : 26.5
...
```



*Do these make more sense?*

# Maximum Likelihood?

- Empirical rates are the maximum likelihood estimates

$$\theta_{ML} = \arg\max_{\theta} P(\mathbf{X}|\theta) = \arg\max_{\theta} \prod_i P_\theta(X_i) \quad \Rightarrow \quad P_{\text{ML}}(x) = \frac{\text{count}(x)}{\text{total samples}}$$

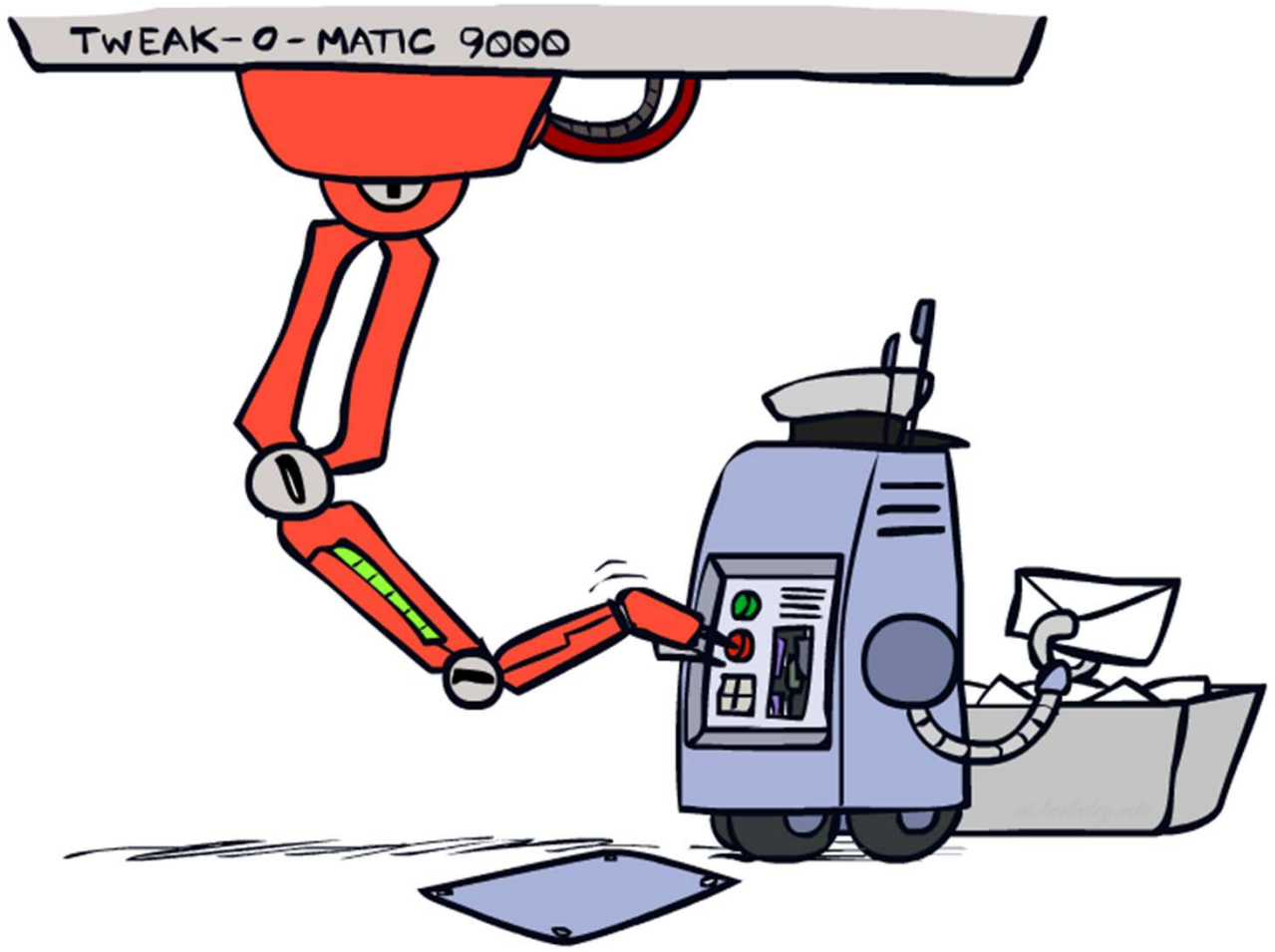
- Laplace's estimate is the most likely parameter value given the data

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

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

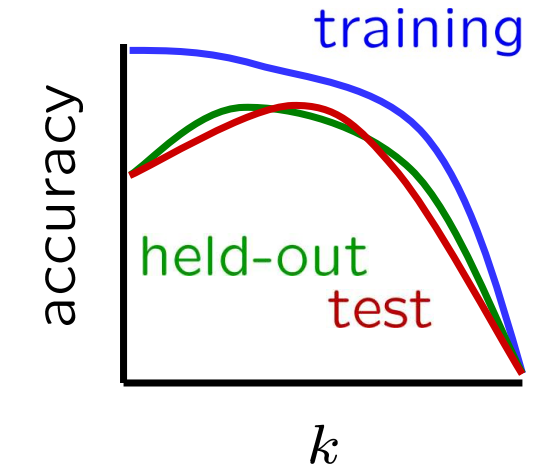
Dirichlet distribution (parameterized by k)

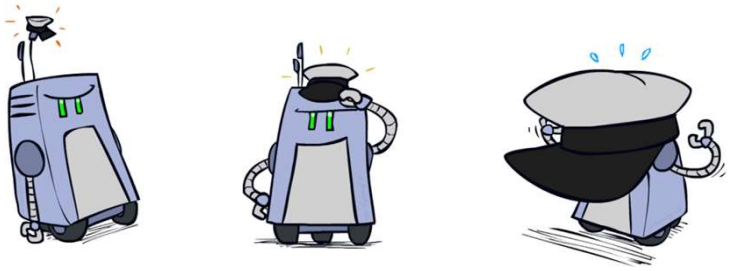
# Tuning



# Tuning on Held-Out Data

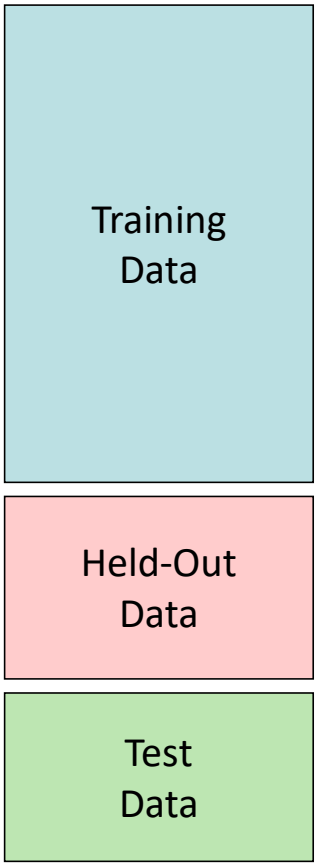
- Now we've got two kinds of unknowns
  - Parameters: the probabilities P(X|Y), P(Y)
  - Hyperparameters: e.g. the amount / type of smoothing to do, k, $\alpha$
- What should we learn where?
  - Learn parameters from training data
  - Tune hyperparameters on different data
    - Why?
  - For each value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

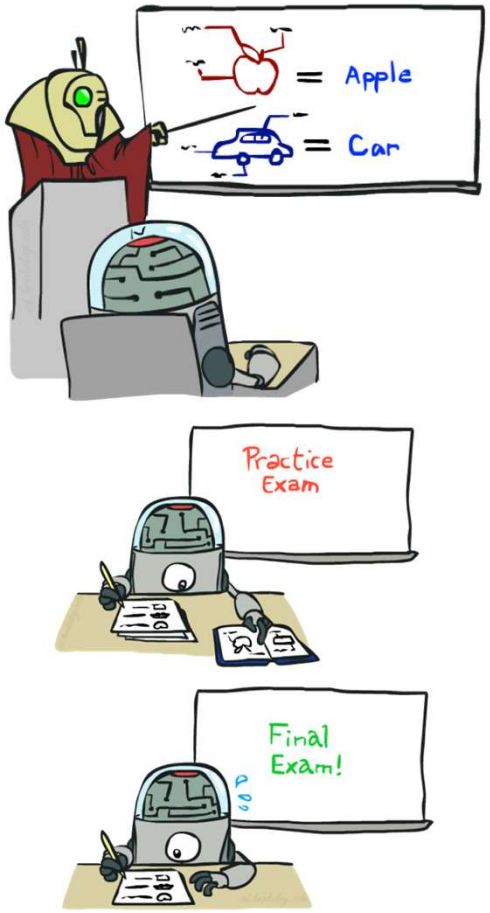


# Tuning for generalization

- Typical problems
  - Underfitting: fitting the training set poorly
  - Overfitting: fitting the training data very well, but not the test data

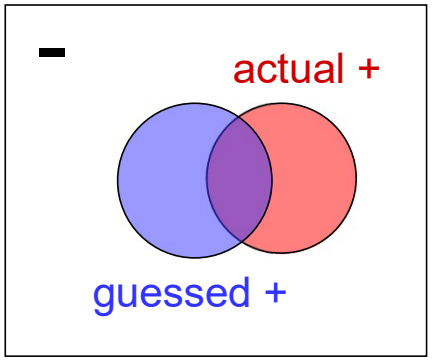



# Baselines

- Is your testing accuracy good or bad?
- First step: get a baseline
  - Baselines are very simple “straw man” procedures
  - Help determine how hard the task is
  - Help know what a “good” accuracy is
- Weak baseline: most frequent label classifier
  - Gives all test instances whatever label was most common in the training set
  - E.g. for spam filtering, might label everything as ham
  - Accuracy might be very high if the problem is skewed
  - E.g. calling everything “ham” gets 66%, so a classifier that gets 70% isn’t very good…
- For real research, usually use previous work as a (strong) baseline

# Precision vs. Recall

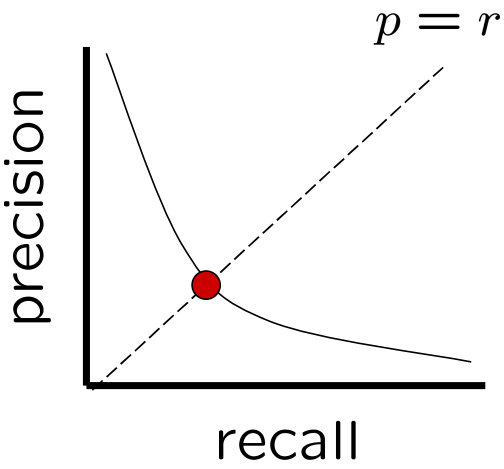
- Let's say we want to classify web pages as homepages or not
  - In a test set of 1K pages, there are 3 homepages
  - Our classifier says they are all non-homepages
  - 99.7 accuracy!
  - Need new measures for rare positive events




- Precision: fraction of guessed positives which were actually positive
- Recall: fraction of actual positives which were guessed as positive
- Say we guess 5 homepages, of which 2 were actually homepages
  - Precision: 2 correct / 5 guessed = 0.4
  - Recall: 2 correct / 3 true = 0.67
- Which is more important in customer support email automation?
- Which is more important in airport baggage security screening?

# Precision vs. Recall

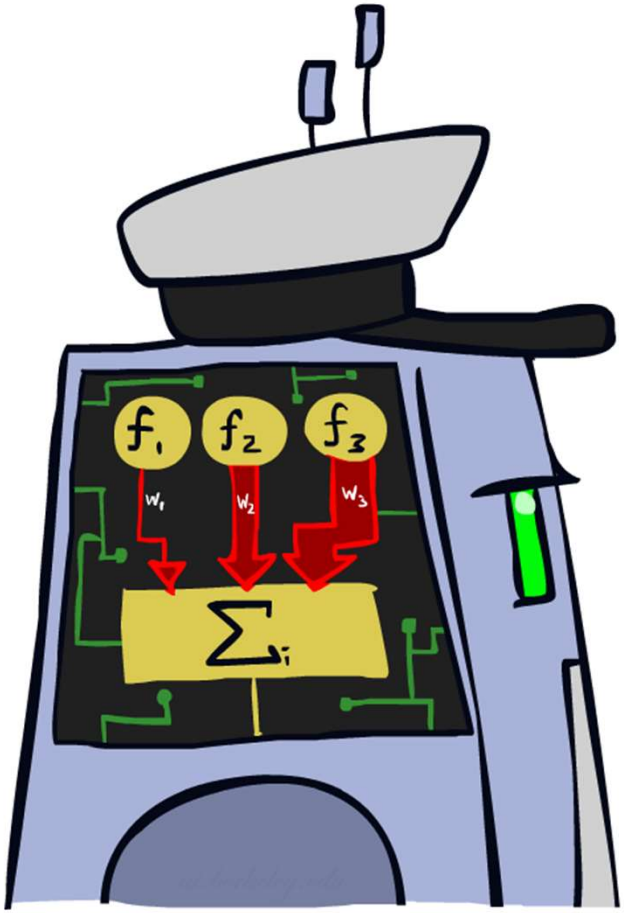
- Precision/recall tradeoff
  - Often, you can trade off precision and recall
  - Only works well with weakly calibrated classifiers




- To summarize the tradeoff:
  - Break-even point: precision value when p = r
  - F-measure: harmonic mean of p and r:

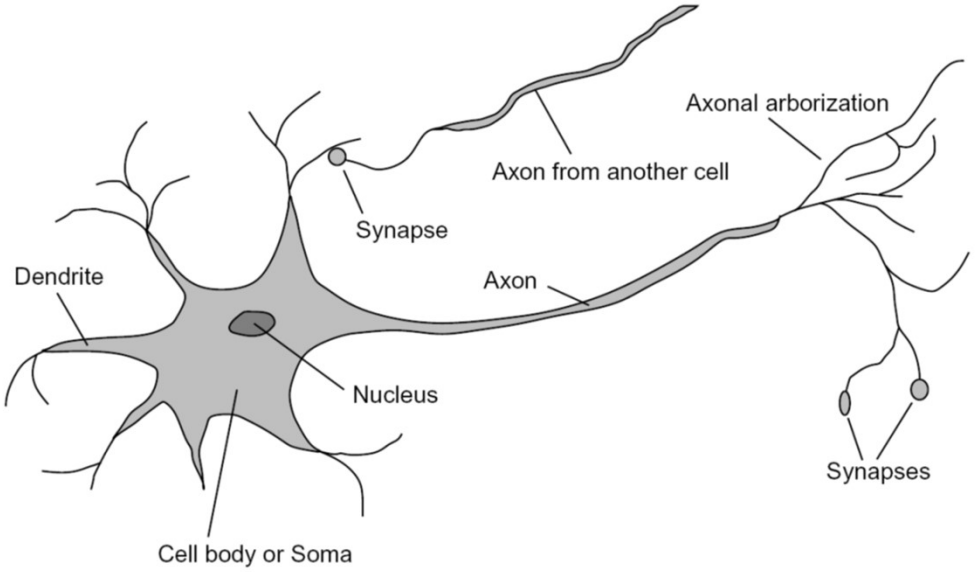
$$F_1 = \frac{2}{1/p + 1/r}$$

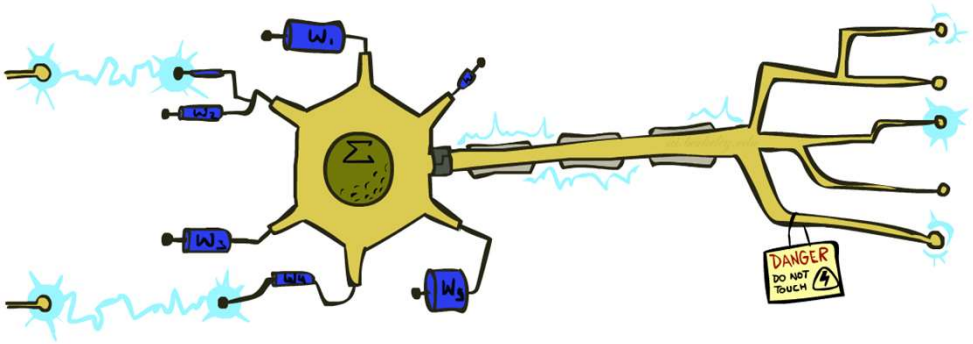
# Linear Classifiers (Perceptrons)



# Some (Simplified) Biology

- Very loose inspiration: human neurons



# Feature Vectors

$x$ $f(x)$ $y$

```
Hello,

Do you want free printr
cartriges?  Why pay more
when you can get them
ABSOLUTELY FREE!  Just
```

```
# free       : 2
YOUR_NAME    : 0
MISSPELLED   : 2
FROM_FRIEND  : 0
...
```

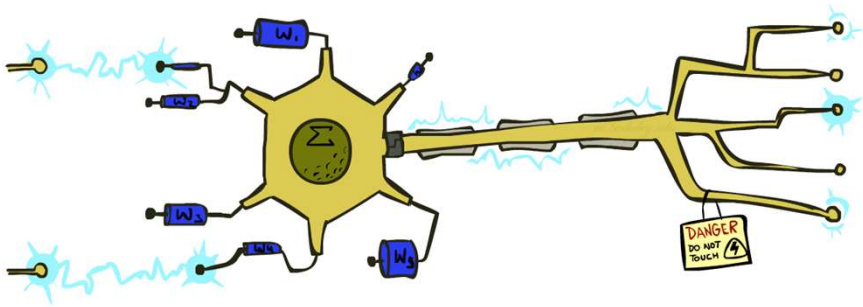
Spam or Ham

```
PIXEL-7,12   : 1
PIXEL-7,13   : 0
...
NUM_LOOPS    : 1
...
```

“2”

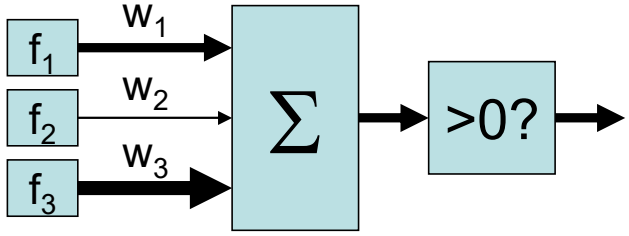
# Linear Classifiers

- Inputs are feature values
- Each feature has a weight
- Sum is the activation




$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- Binary case: if the activation is:
  - Positive, output +1
  - Negative, output -1



# Linear Classifiers

- Sometimes we also add a bias term

$$\sum_i w_i \cdot f_i(x) + b = w \cdot f(x) + b$$

- This is equivalent to appending a constant feature

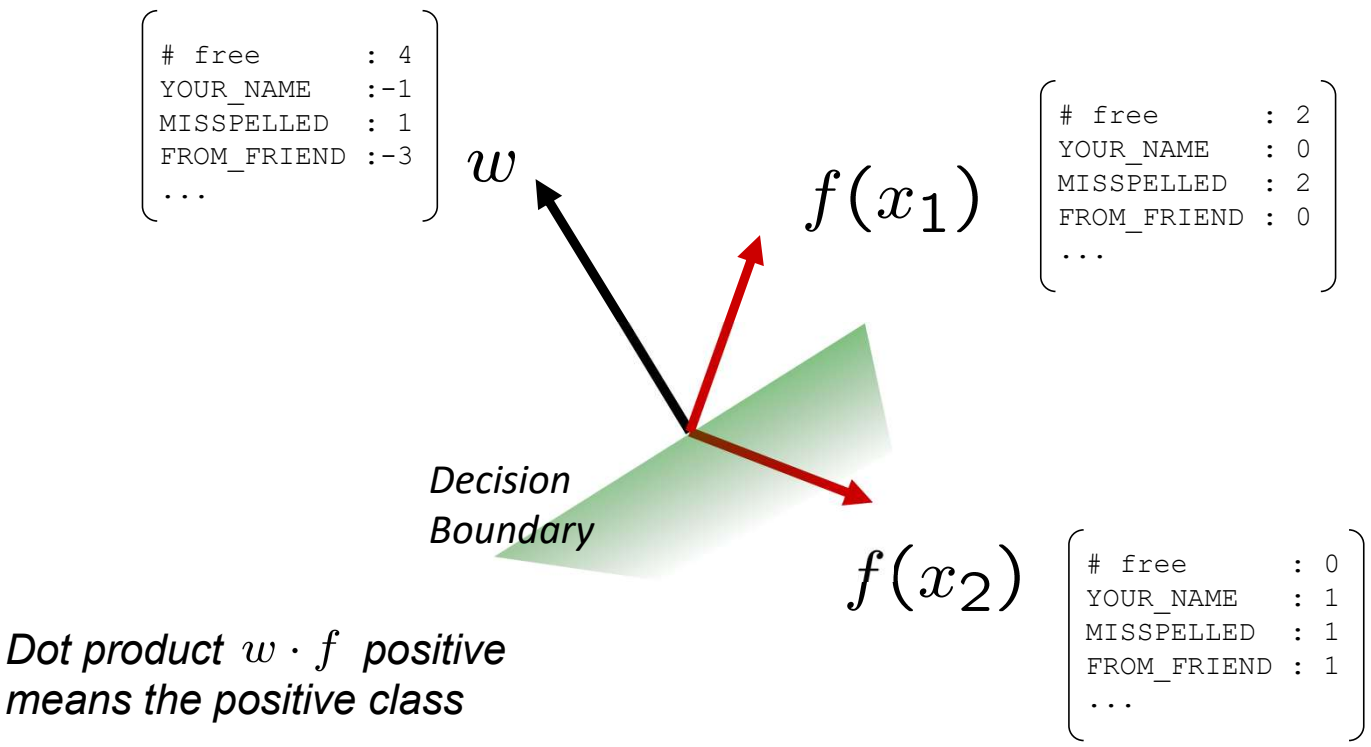
$f(x)$

```
# free       : 2
YOUR_NAME    : 0
MISSPELLED   : 2
FROM_FRIEND  : 0
...
Bias         ≡ 1
```

$$\sum_{i=1}^{n+1} w_i \cdot f_i(x) = \sum_{i=1}^{n} w_i \cdot f_i(x) + w_{n+1}$$

# Decision Boundary

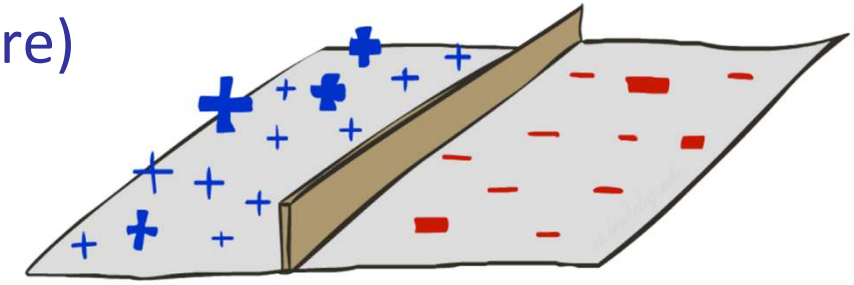
- Binary case: compare features to a weight vector

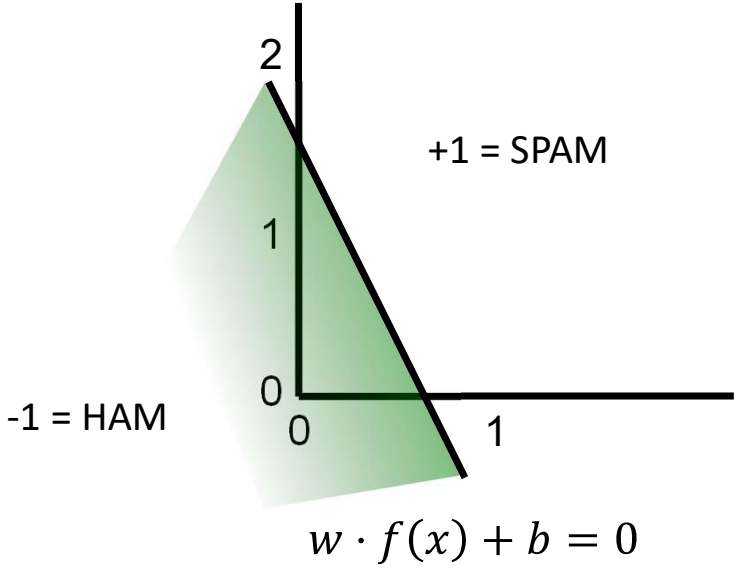



*Dot product $w \cdot f$ positive means the positive class*

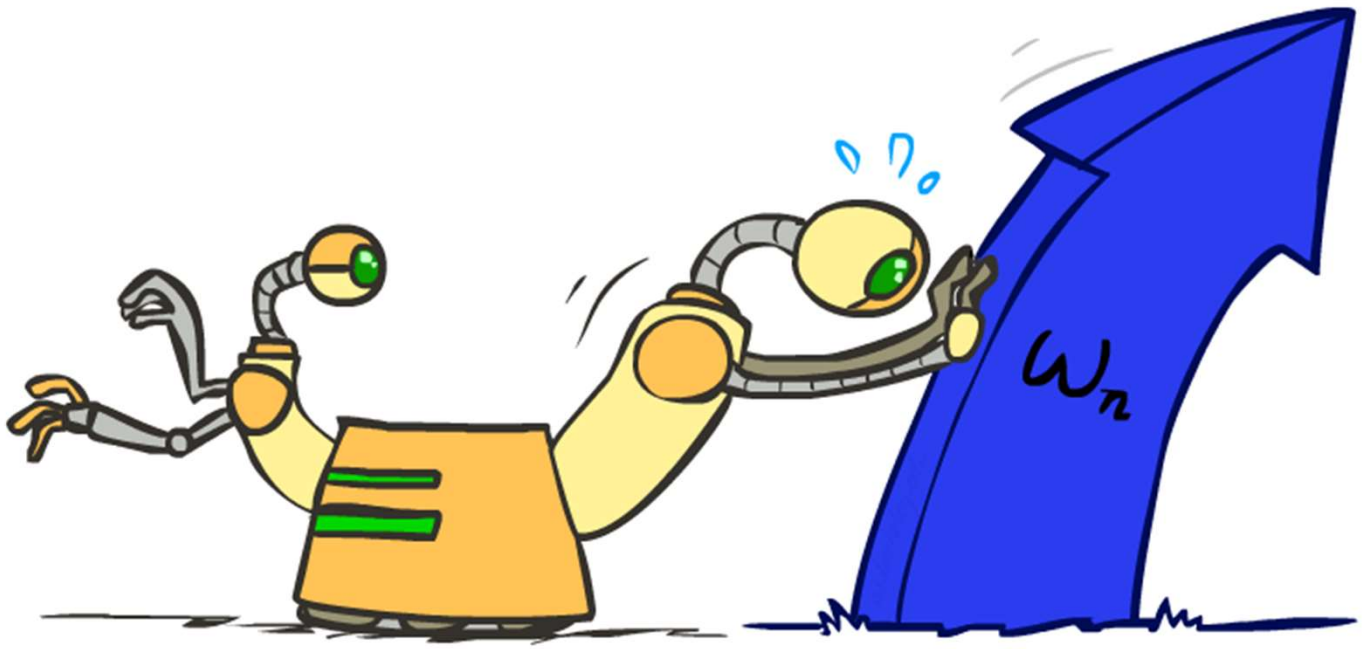
# Decision Boundary

- In the space of feature vectors (no bias feature)
  - Examples are points
  - Weight vector specifies a hyperplane
    - $w \cdot f(x) + b = 0$
  - One side corresponds to Y=+1
  - Other corresponds to Y=-1



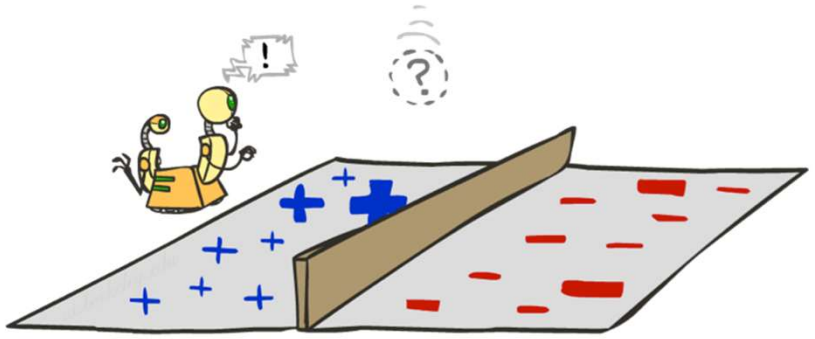


# Weight Updates

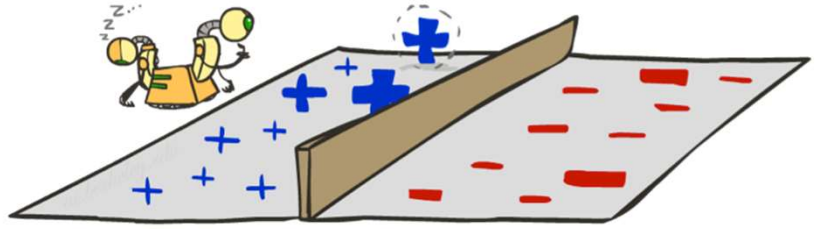


# Learning: Binary Perceptron

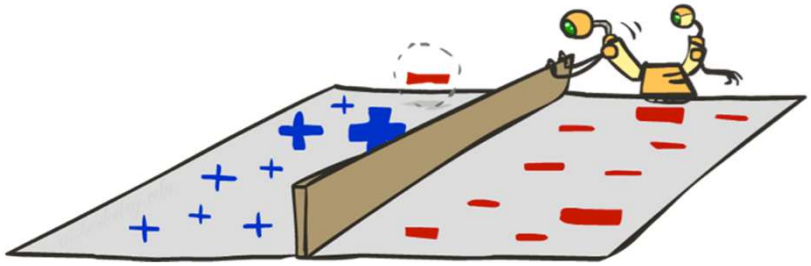
- Start with weights = 0
- For each training instance:
  - Classify with current weights




  - If correct (i.e., y=y*), no change!



  - If wrong: adjust the weight vector



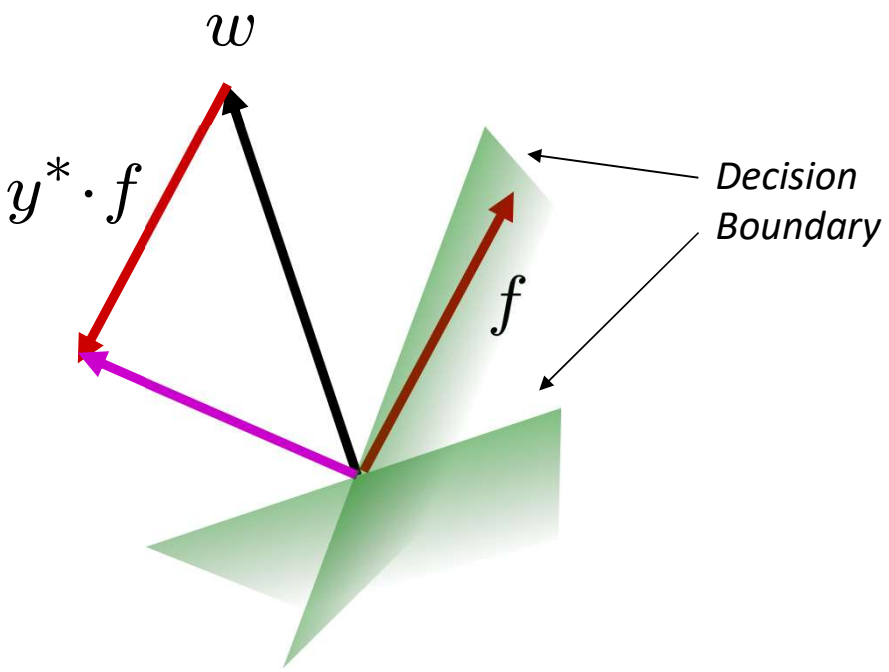
# Learning: Binary Perceptron

- Start with weights = 0
- For each training instance:
  - Classify with current weights

$$y = \begin{cases} +1 & \text{if } w \cdot f(x) \geq 0 \\ -1 & \text{if } w \cdot f(x) < 0 \end{cases}$$

  - If correct (i.e., y=y*), no change!
  - If wrong: adjust the weight vector by adding or subtracting the feature vector.

$$w = w + y^* \cdot f$$

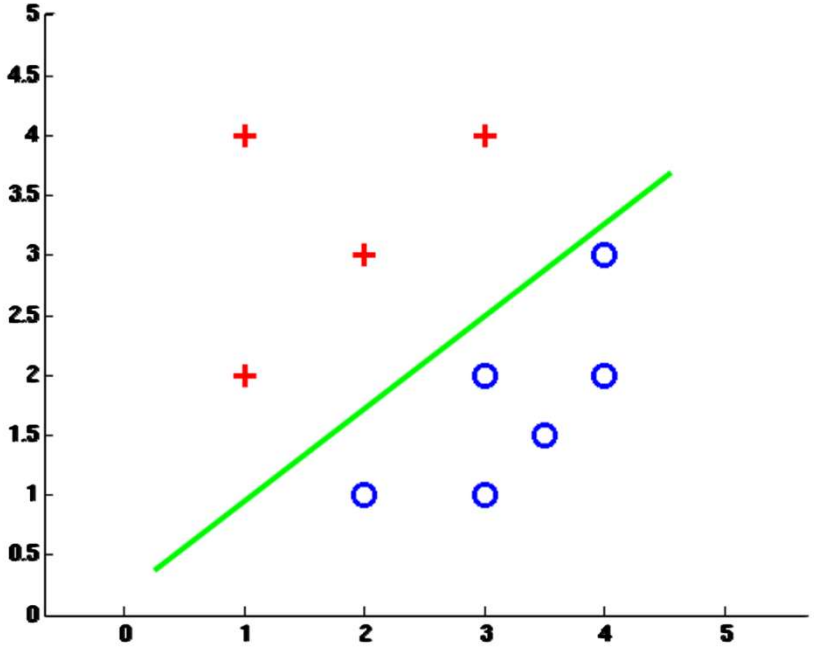



Before: $w \cdot f(x)$

After: $w \cdot f(x) + y^* f(x) \cdot f(x)$ $\geq 0$

# Examples: Perceptron

- Separable Case



# Multiclass Decision Rule

- If we have multiple classes:
  - A weight vector for each class:

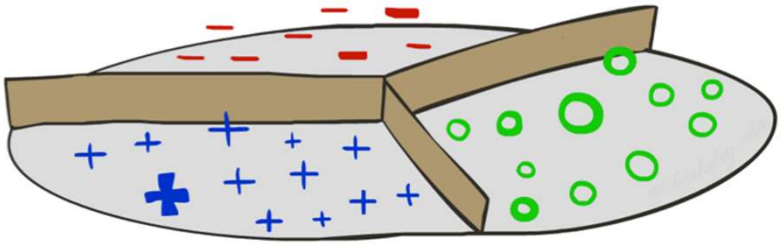
$$w_y$$

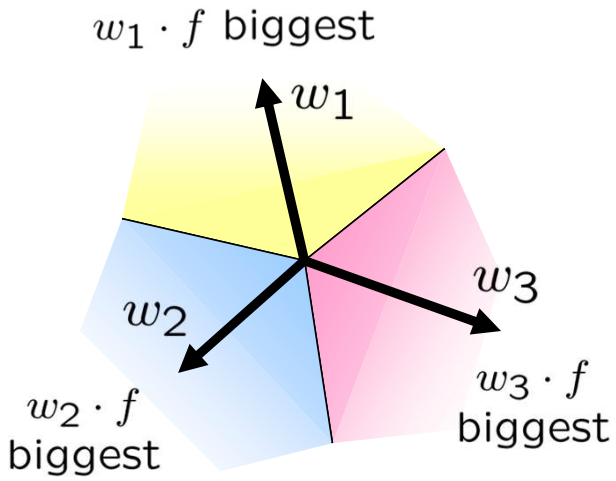
  - Score (activation) of a class y:

$$w_y \cdot f(x)$$

  - Prediction highest score wins

$$y = \arg\max_y \; w_y \cdot f(x)$$






*Binary = multiclass where the negative class has weight zero*

# Learning: Multiclass Perceptron

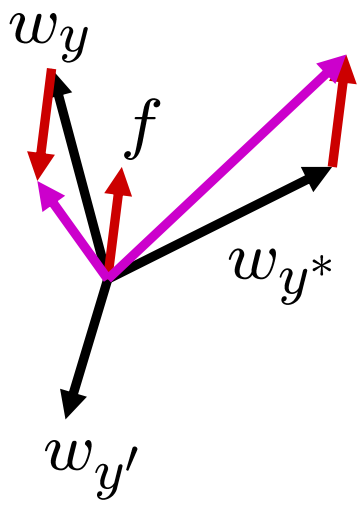
- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights

$$y = \arg\max_y \; w_y \cdot f(x)$$

- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

$$w_y = w_y - f(x)$$

$$w_{y^*} = w_{y^*} + f(x)$$



# Example: Multiclass Perceptron

| | | $y$ |
|---|---|---|
| "win the vote" | [1 1 0 1 1] | Politics |
| "win the election" | [1 1 0 0 1] | Politics |
| "win the game" | [1 1 1 0 1] | Tech |

$w_{SPORTS}$

| | | | |
|---|---|---|---|
| | 1 | -2 | -2 |
| BIAS : 1 | 0 | | 1 |
| win : 0 | -1 | | 0 |
| game : 0 | 0 | | 1 |
| vote : 0 | -1 | | -1 |
| the : 0 | -1 | | 0 |
| ... | | | |

$w_{POLITICS}$

| | | | |
|---|---|---|---|
| | 0 | 3 | 3 |
| BIAS : 0 | 1 | | 0 |
| win : 0 | 1 | | 0 |
| game : 0 | 0 | | -1 |
| vote : 0 | 1 | | 1 |
| the : 0 | 1 | | 0 |
| ... | | | |

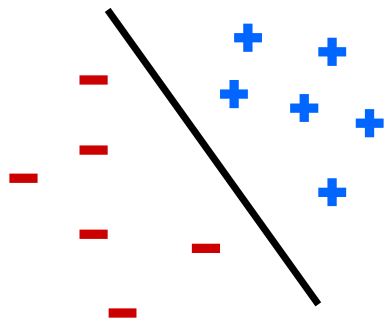
$w_{TECH}$

| | | |
|---|---|---|
| | 0 | 0 |
| BIAS : 0 | | |
| win : 0 | | |
| game : 0 | | |
| vote : 0 | | |
| the : 0 | | |
| ... | | |

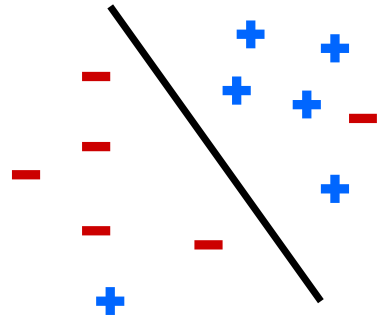
# Properties of Perceptrons

- Separability: true if some parameters get the training set perfectly correct
- Convergence: if the training set is separable, perceptron will eventually converge (binary case)

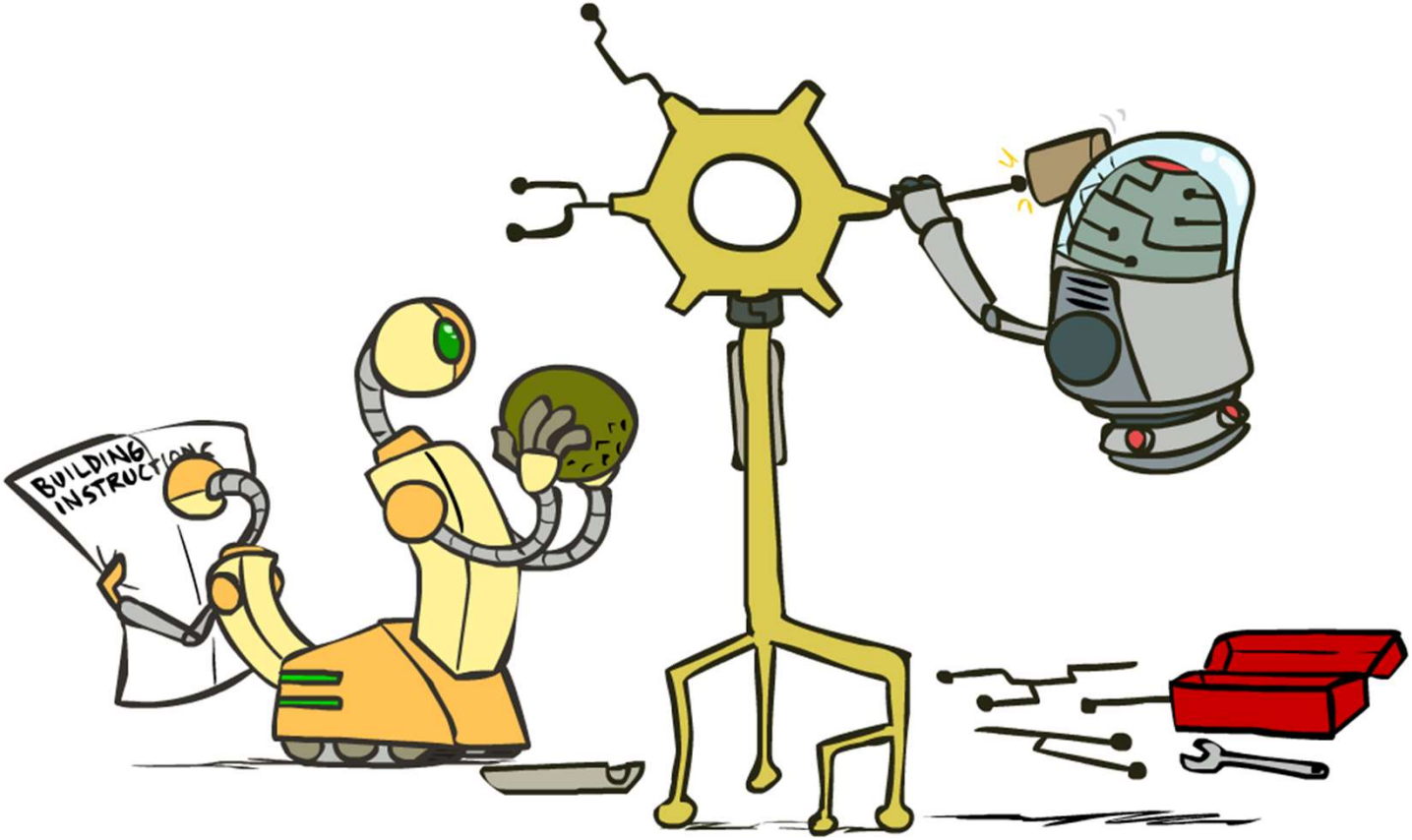
Separable



Non-Separable

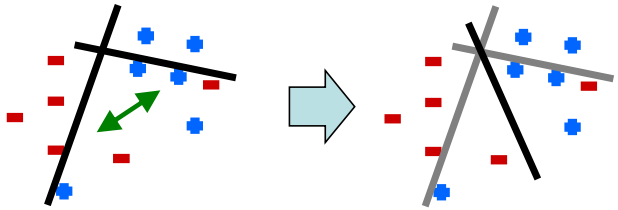


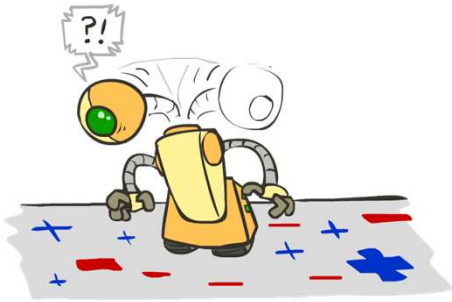
# Improving Perceptrons (Logistic Regression)



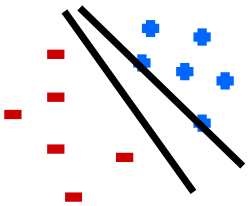
# Problems with Perceptrons

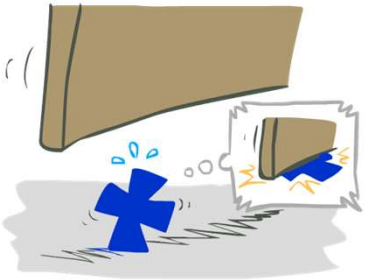
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



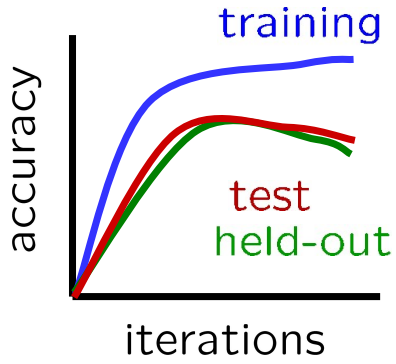


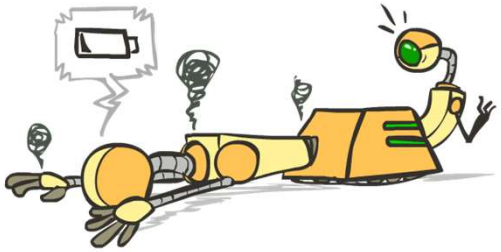

- Mediocre generalization: finds a "barely" separating solution





- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting

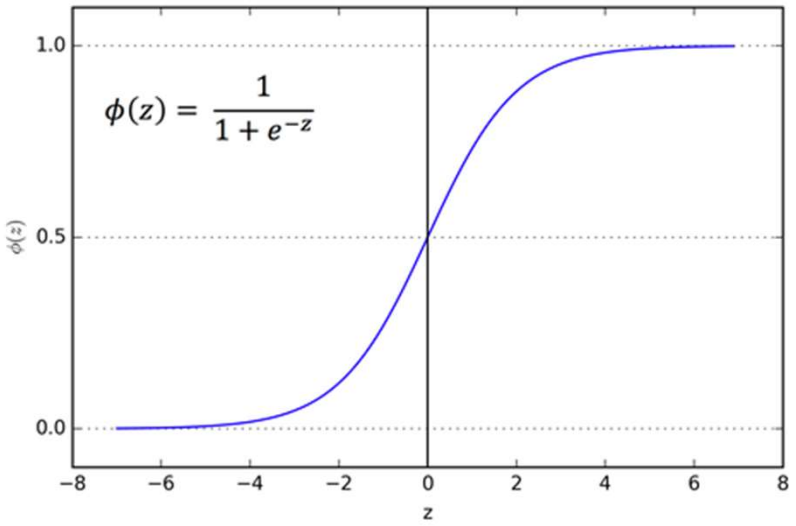


# Probabilistic decisions

- Perceptron scoring: $z = w \cdot f(x)$
  - z=0.1 and z=100 both produces +1
- Probabilistic decisions
  - z very positive → prob of +1 going to 1
  - z close to 0 → prob of +1 close to 0.5
  - z very negative → prob of +1 going to 0
- Sigmoid function

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



# Best w?

- Maximum likelihood estimation:

$$\max_w \; ll(w) = \max_w \; \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

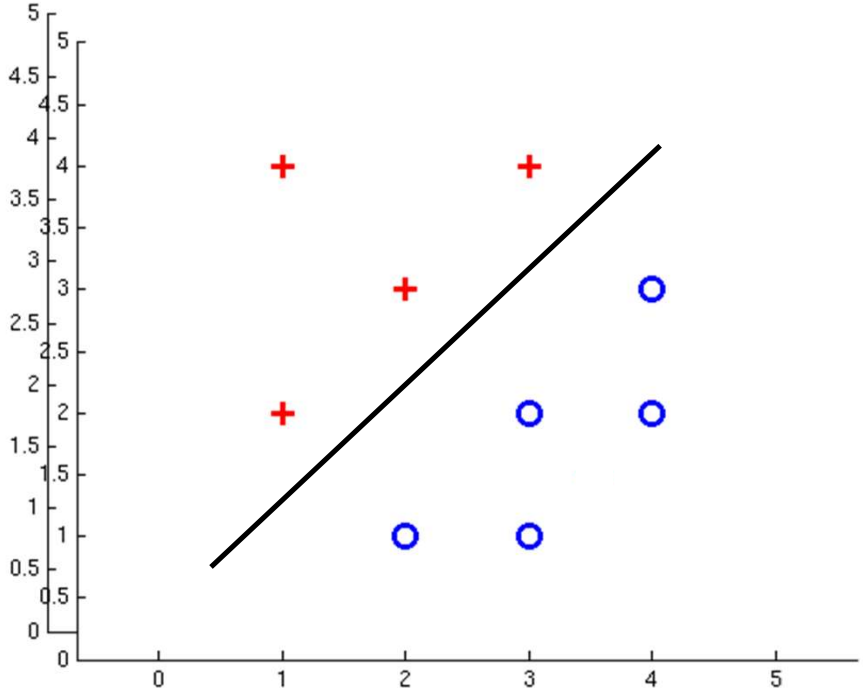
with:

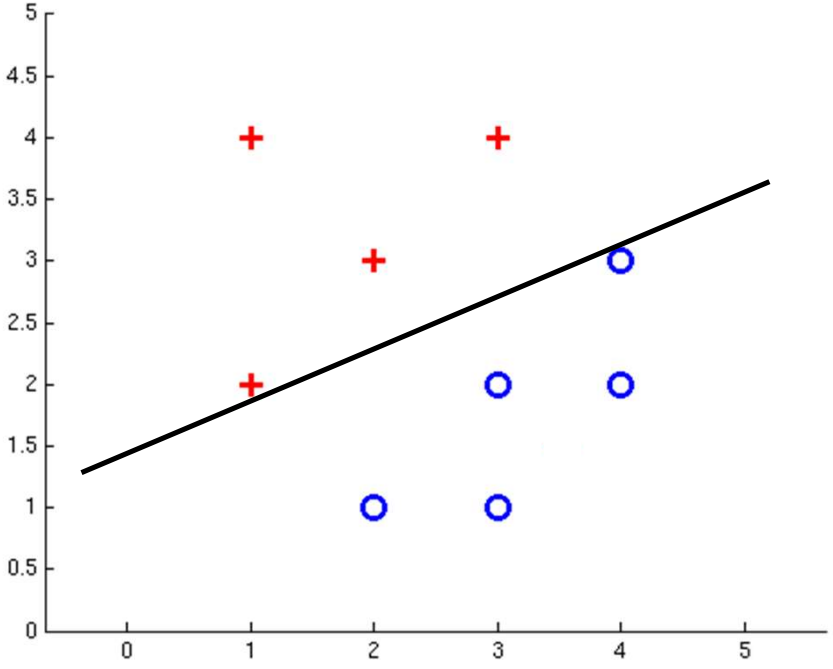
$$P(y^{(i)} = +1|x^{(i)}; w) = \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

$$P(y^{(i)} = -1|x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

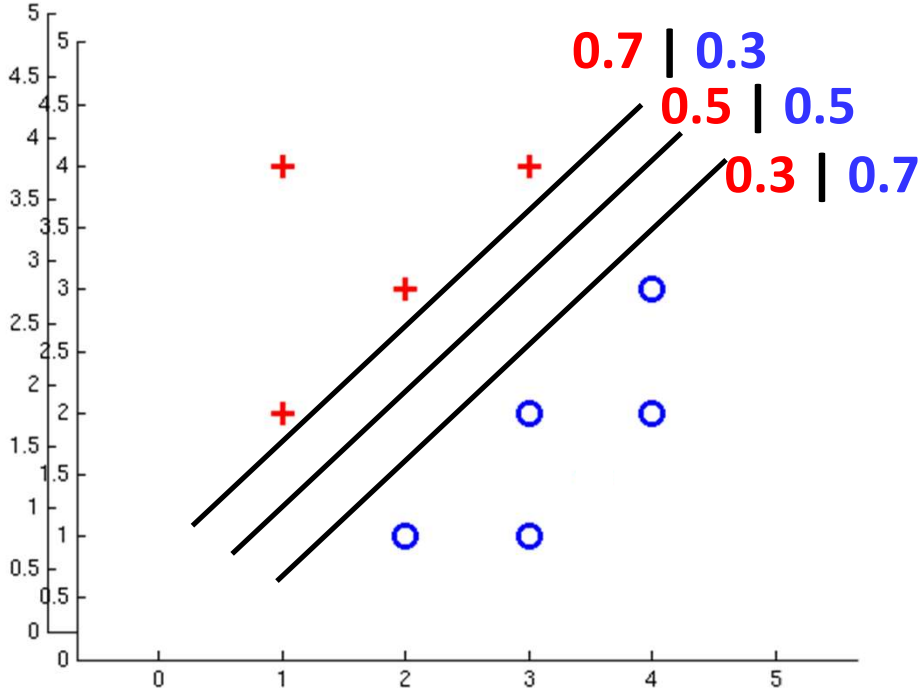
**= Logistic Regression**

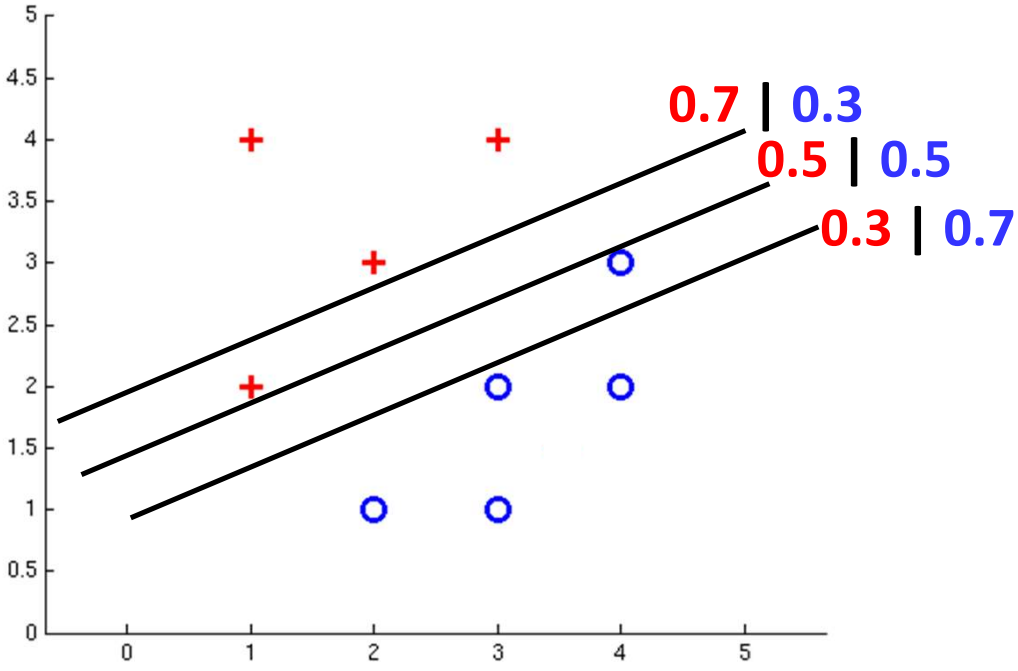
# Separable Case: Deterministic Decision – Many Options



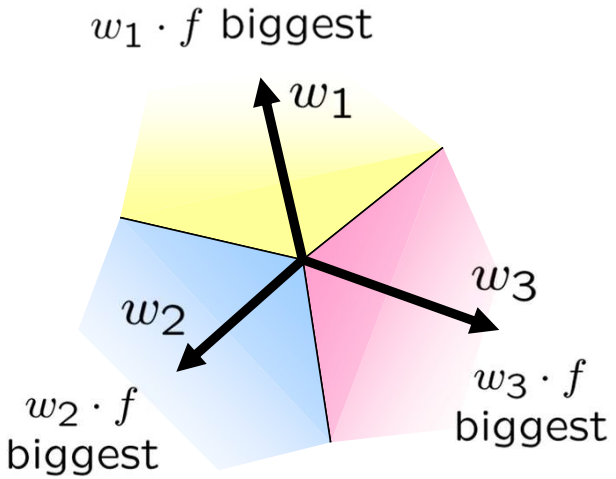
# Separable Case: Probabilistic Decision – Clear Preference



# Multiclass Logistic Regression

- Recall Perceptron:
  - A weight vector for each class: $w_y$
  - Score (activation) of a class y: $w_y \cdot f(x)$
  - Prediction highest score wins $y = \arg\max_y \ w_y \cdot f(x)$




- How to make the scores into probabilities?

$$\underbrace{z_1, z_2, z_3}_{\text{original activations}} \rightarrow \underbrace{\frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}}_{\text{softmax activations}}$$

# Best w?

- Maximum likelihood estimation:

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

with: $$P(y^{(i)}|x^{(i)}; w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$$

**= Multi-Class Logistic Regression**

# Optimization

- How do we solve:

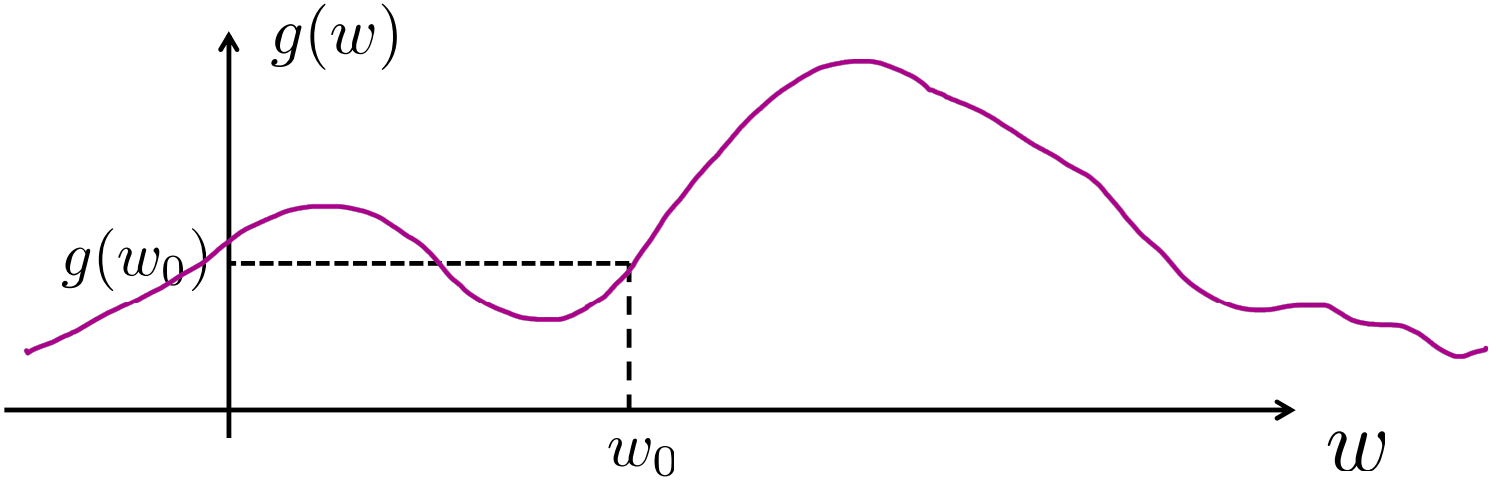
$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

# Hill Climbing

- Recall from CSP lecture: simple, general idea
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit

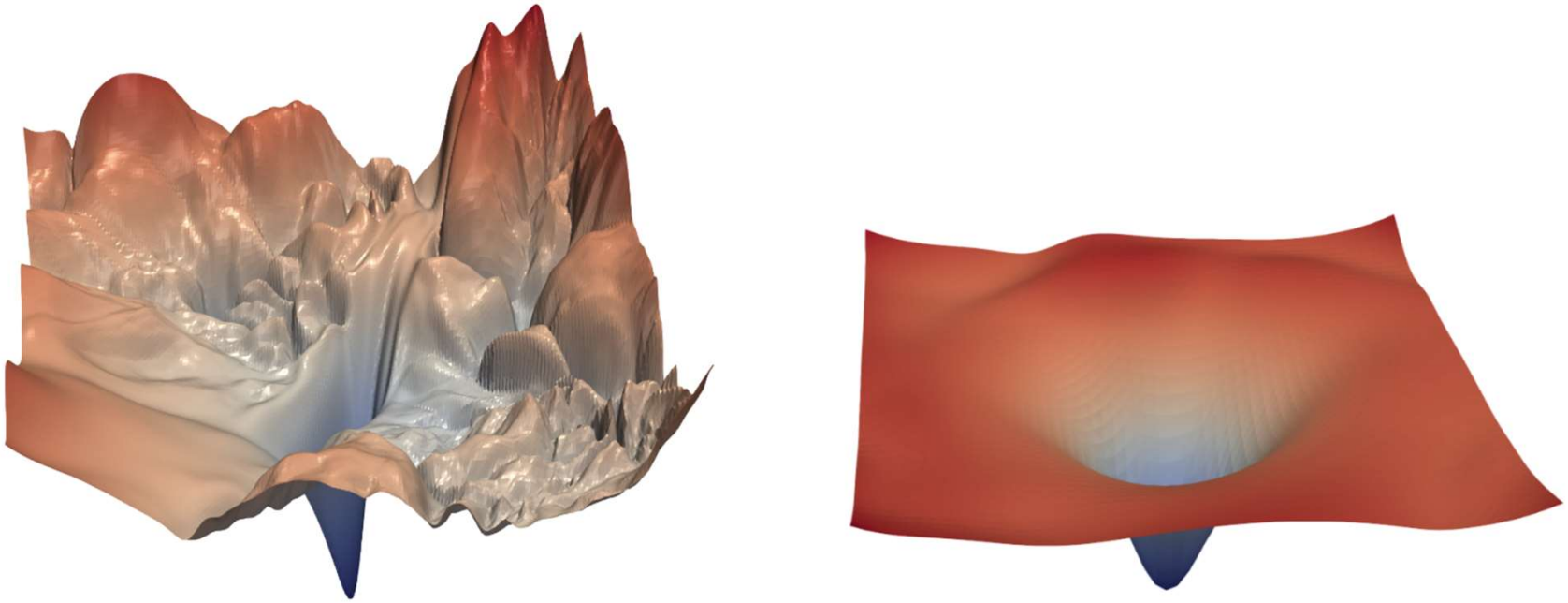
- Can we do hill-climbing for multiclass logistic regression?
  - Optimization over a continuous space
    - Infinitely many neighbors!
    - How to do this efficiently?

# 1-D Optimization




- Could evaluate $g(w_0 + h)$ and $g(w_0 - h)$
  - Then step in best direction
- Or, evaluate derivative: $$\frac{\partial g(w_0)}{\partial w} = \lim_{h \to 0} \frac{g(w_0 + h) - g(w_0 - h)}{2h}$$
  - Tells which direction to step into

# Loss Landscape



(a) without skip connections

(b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

Li, Hao, et al. "Visualizing the loss landscape of neural nets." *Advances in neural information processing systems* 31 (2018).

# Gradient Ascent

- Perform update in uphill direction for each coordinate
- The steeper the slope (i.e. the higher the derivative) the bigger the step for that coordinate
- E.g., consider: $g(w_1, w_2)$
  - Updates:

$$w_1 \leftarrow w_1 + \alpha * \frac{\partial g}{\partial w_1}(w_1, w_2)$$

$$w_2 \leftarrow w_2 + \alpha * \frac{\partial g}{\partial w_2}(w_1, w_2)$$

  - Updates in vector notation:

$$w \leftarrow w + \alpha * \nabla_w g(w)$$

with: $\nabla_w g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_1}(w) \\ \frac{\partial g}{\partial w_2}(w) \end{bmatrix}$ **= gradient**

# Gradient Ascent

- `init` $w$
- `for iter = 1, 2, …`

$$w \leftarrow w + \alpha * \nabla g(w)$$

- $\alpha$: learning rate
  - A hyperparameter that needs to be chosen carefully

# Gradient Ascent

- Start somewhere
- Repeat:  Take a step in the gradient direction

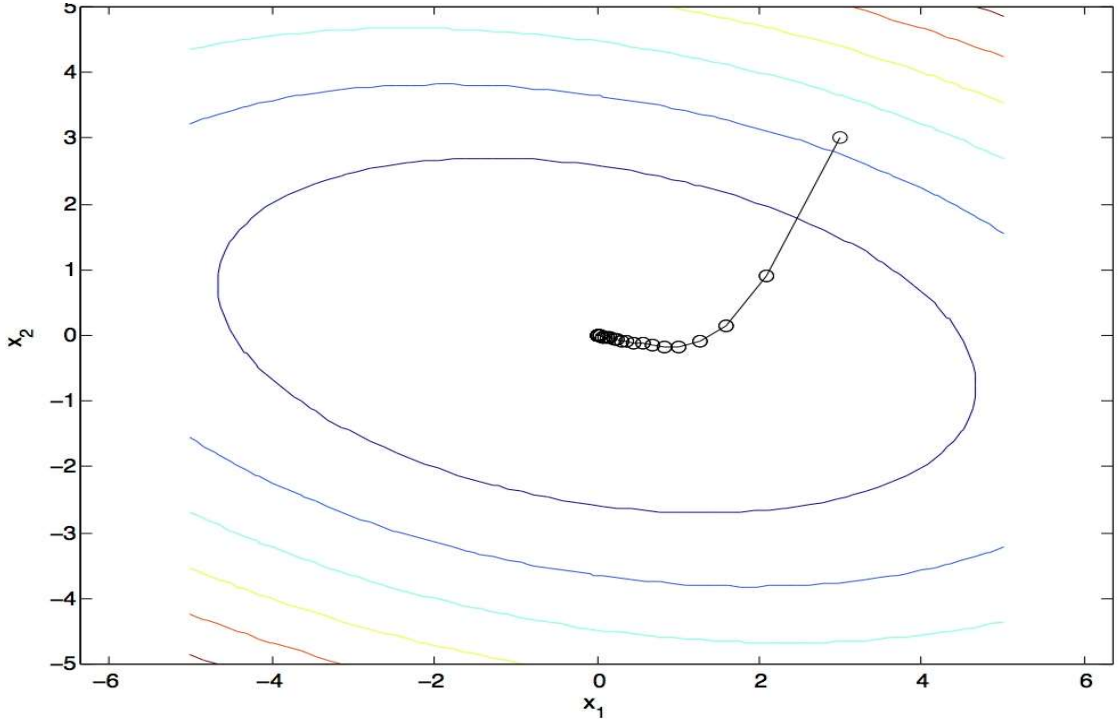



Figure source: Mathworks

# Batch Gradient Ascent on the Log Likelihood Objective

$$\max_w \quad ll(w) = \max_w \quad \underbrace{\sum_i \log P(y^{(i)}|x^{(i)};w)}_{g(w)}$$

- `init` $w$
- `for iter = 1, 2, …`

$$w \leftarrow w + \alpha * \sum_i \nabla \log P(y^{(i)}|x^{(i)};w)$$

# What will gradient ascent do?

$$w \leftarrow w + \alpha * \sum_i \nabla \log P(y^{(i)}|x^{(i)}; w)$$

$$P(y^{(i)}|x^{(i)}; w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$$

$$\nabla w_{y^{(i)}} f(x^{(i)}) - \nabla \log \sum_y e^{w_y f(x^{(i)})}$$

add f to weights of the correct class

Increase the score of the correct class

for y' weights: $\frac{1}{\sum_y e^{w_y f(x^{(i)})}} e^{w_{y'} f(x^{(i)})} f(x^{(i)})$

$= P(y'|x^{(i)}; w) f(x^{(i)})$

subtract f from y' weights in proportion to the current probability of y'

Decrease the scores of all the classes

# Stochastic Gradient Ascent on the Log Likelihood Objective

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

**Idea:** once gradient on one training example has been computed, might as well update before computing next one

- `init` $w$
- `for iter = 1, 2, …`
  - `pick random j`

$$w \leftarrow w + \alpha * \nabla \log P(y^{(j)}|x^{(j)}; w)$$

# Mini-Batch Gradient Ascent on the Log Likelihood Objective

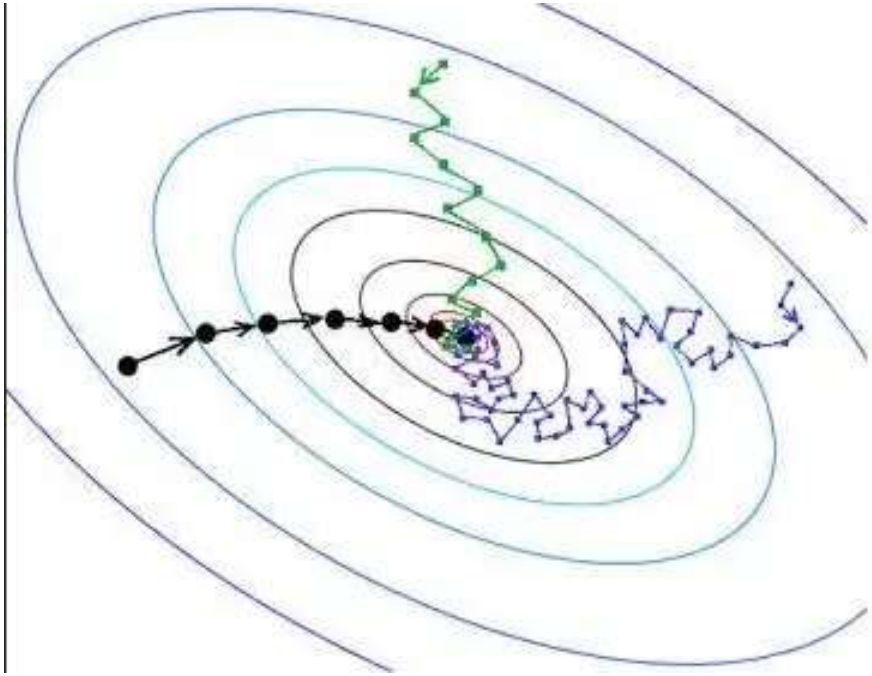
$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

**Idea:** gradient over a small set of training examples (=mini-batch) can be computed in parallel, might as well do that instead of a single one

- `init` $w$
- `for iter = 1, 2, …`
  - `pick random subset of training examples J`

$$w \leftarrow w + \alpha * \sum_{j \in J} \nabla \log P(y^{(j)}|x^{(j)}; w)$$

# Batch vs. Stochastic vs. Mini-batch GD



**Batch GD**
- Slowest
- Perfect gradient

**Stochastic GD**
- Fastest
- Rough-estimate grad

**Mini-batch GD**
- Compromise

https://sweta-nit.medium.com/batch-mini-batch-and-stochastic-gradient-descent-e9bc4cacd461