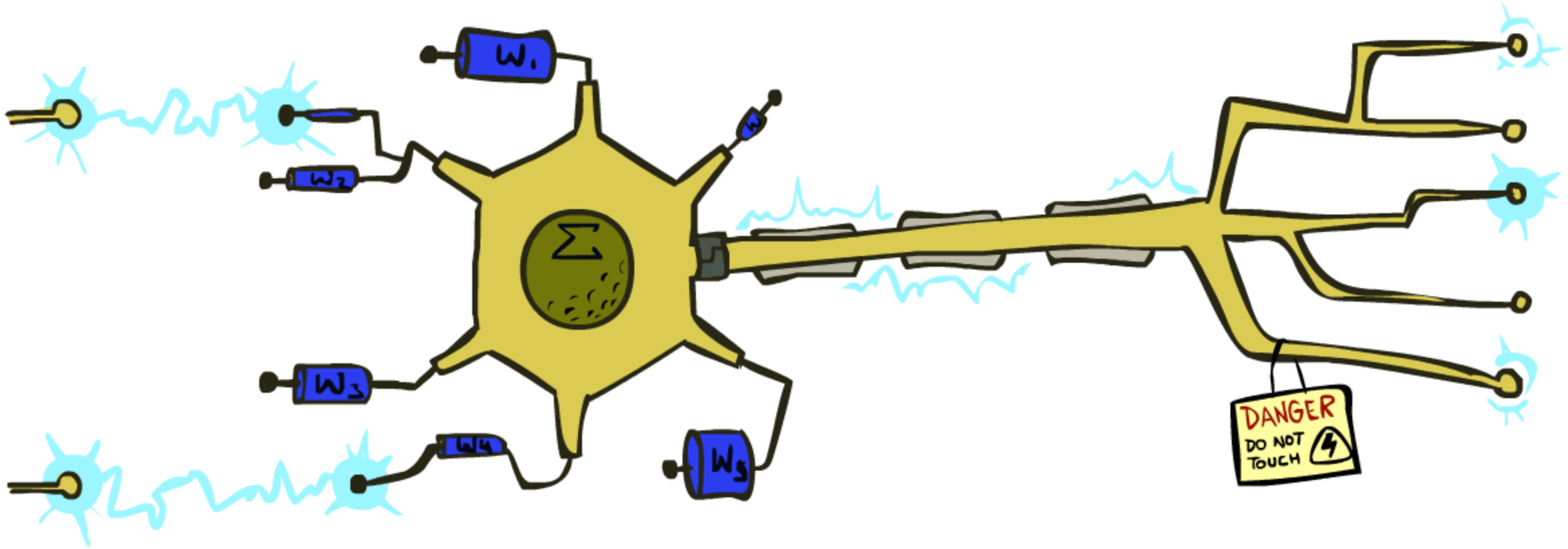
Announcements

- **Homework 9** due **next Monday** (Nov 14) at 11:59pm PT
- **Project 4** due **today** (Nov 8) at 11:59pm PT
- **Project 5** will be released soon
- TA strike next week (starting Nov 14)
  - Lectures will still continue
  - Peyrin and I will run basic version of the class and support you all as much as we can
  - Assignments may be postponed (we'll make sure this is fair to you)

CS 188: Artificial Intelligence

Naïve Bayes and Perceptrons



[These slides were created by Dan Klein, Pieter Abbeel, Anca Dragan, Sergey Levine. All CS188 materials are at http://ai.berkeley.edu.]

Today's Topics

- Naïve Bayes
  - Review: making predictions / inference
  - Learning parameters
- Machine Learning Workflow
- Perceptrons
  - Making predictions
  - Learning parameters

Last Time

- **Classification:** given inputs x, predict labels (classes) y
  - Convert input x into a collection of *features* $f_1, \dots, f_n$

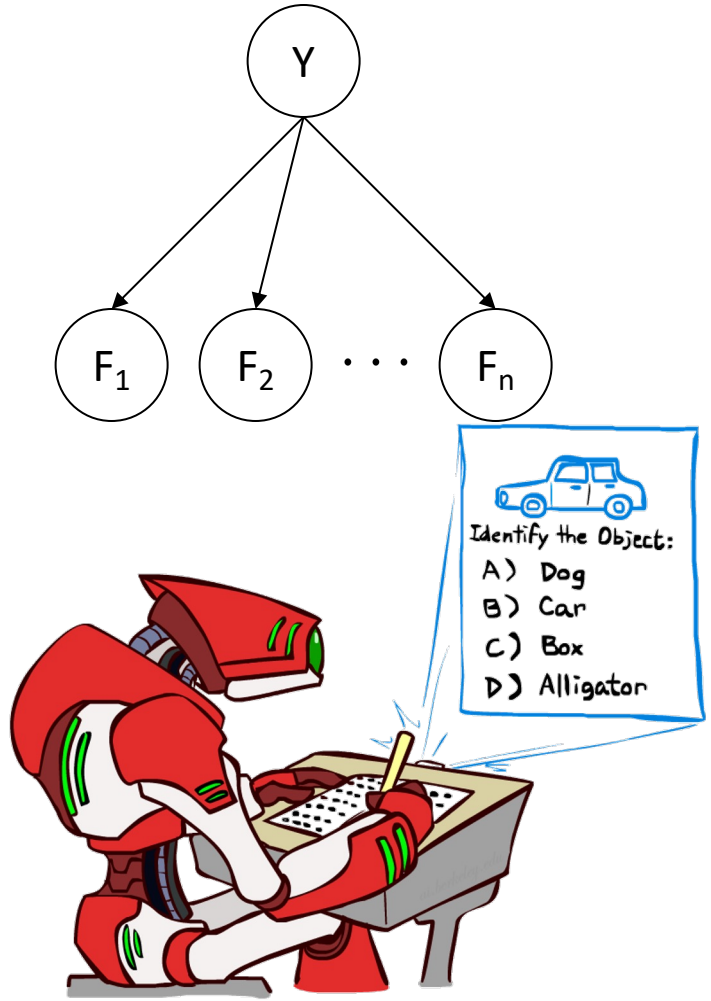
- Naïve Bayes model: $P(Y, F_1, \dots, F_n) = P(Y) \prod_i P(F_i|Y)$
  - Input features $F_1, \dots, F_n$ are conditionally independent given label Y

- Inference for Naïve Bayes:
  - Inference by enumeration
  - Given input x features $f_1, \dots, f_n$ probability over class labels is:

$$P(Y|f_1, \dots, f_n) = P(Y, f_1, \dots, f_n)/Z = P(Y) \prod_i P(f_i|Y) /Z$$

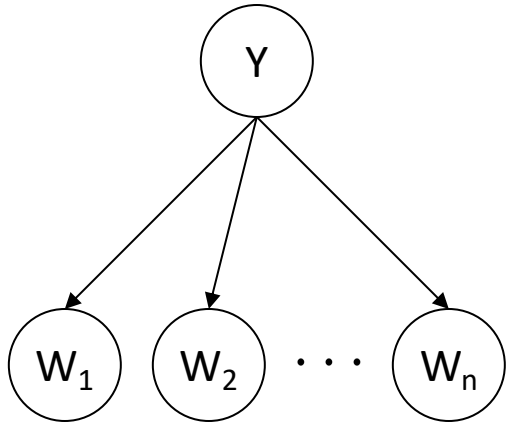
Where $Z = \sum_k P(y_k) \prod_i P(f_i|y_k)$

- Naïve Bayes model parameters:
  - Probability tables $P(Y), P(F_1|Y), \dots, P(F_n|Y)$
  - $|Y| + |F| * |Y| * n$ parameter values



Example: Naïve Bayes for Spam Filtering

- Predict if an email is spam or not
  - Features: W_i is the i'th word in the email (domain: words in the dictionary)
  - Labels: $Y \in \{spam, ham\}$
- Estimated parameters:



$P(Y)$

Y	spam	ham
P(Y)	2/6	4/6

$P(W|Y)$

W	"now"	"the"	"buy"
P(W | Y=spam)	2/6	1/6	3/6
P(W | Y=ham)	1/6	4/6	1/6

- What is $P(Y|"buy", "now")$?
  - $P(spam|"buy", "now") \propto P(spam)P("buy"|spam)P("now"|spam) = \frac{2}{6} \cdot \frac{3}{6} \cdot \frac{2}{6} = \frac{12}{6^3}$
  - $P(ham|"buy", "now") \propto P(ham)P("buy"|ham)P("now"|ham) = \frac{4}{6} \cdot \frac{1}{6} \cdot \frac{1}{6} = \frac{4}{6^3}$
  - Renormalize:

$P(Y|"buy", "now")$

spam	ham
12/16	4/16

- Prediction: pick label with highest probability, so predict ***spam*** for text *"buy now"*

Example: Naïve Bayes for Spam Filtering

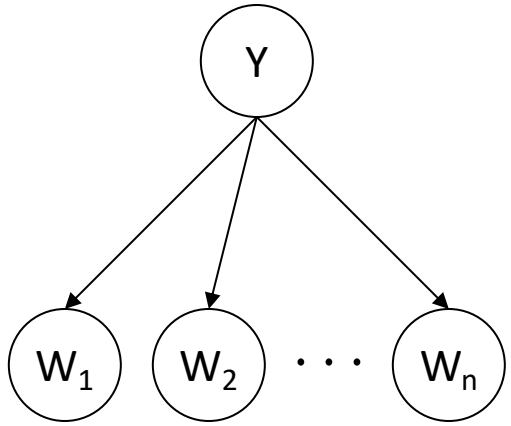
- Predict if an email is spam or not
  - Features: W_i is the i'th word in the email (domain: words in the dictionary)
  - Labels: $Y \in \{spam, ham\}$
- Estimated parameters:

$P(Y)$

Y	spam	ham
P(Y)	2/6	4/6

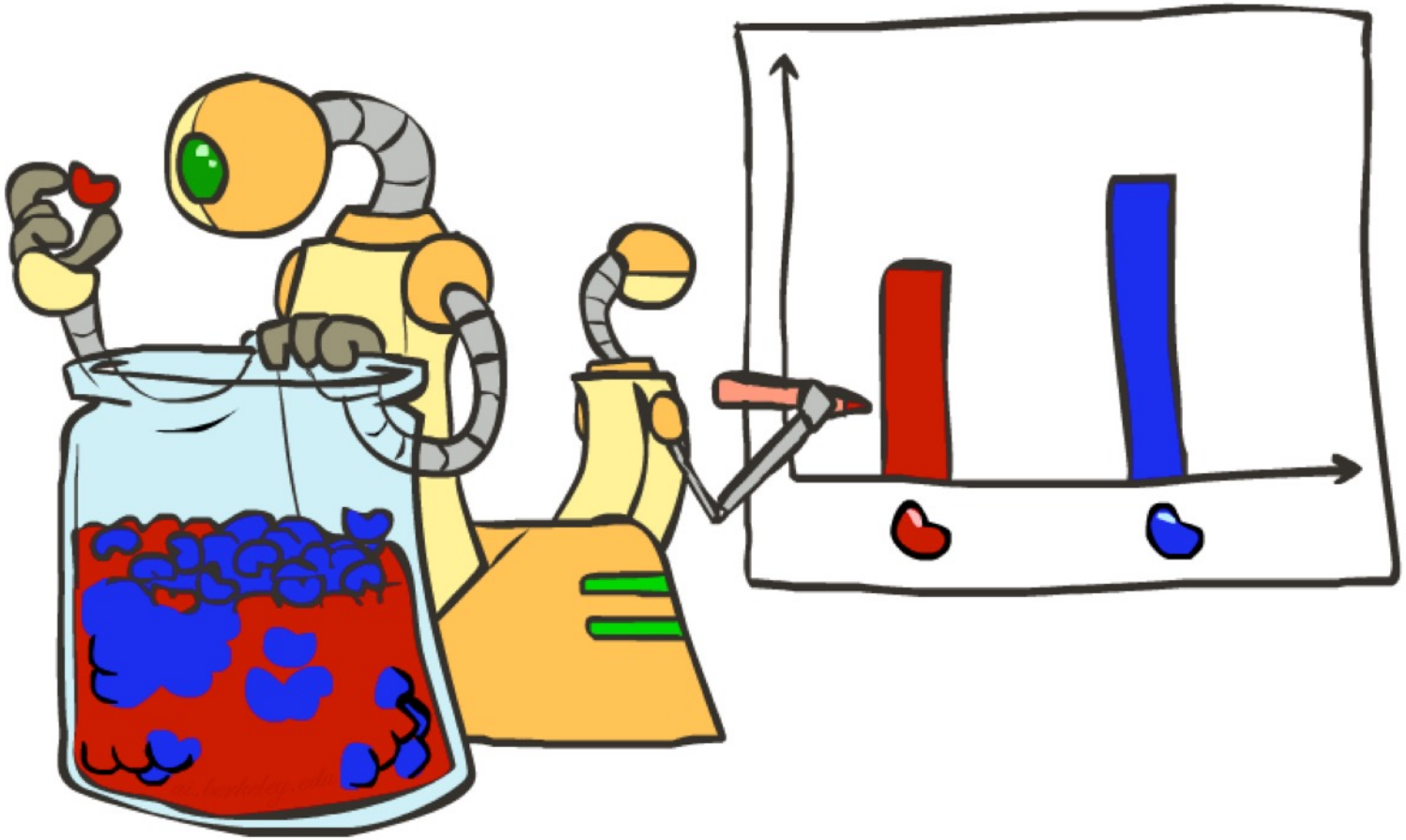
$P(W|Y)$

W	"now"	"the"	"buy"
P(W | Y=spam)	2/6	1/6	3/6
P(W | Y=ham)	1/6	4/6	1/6



- How can we estimate these model parameters?

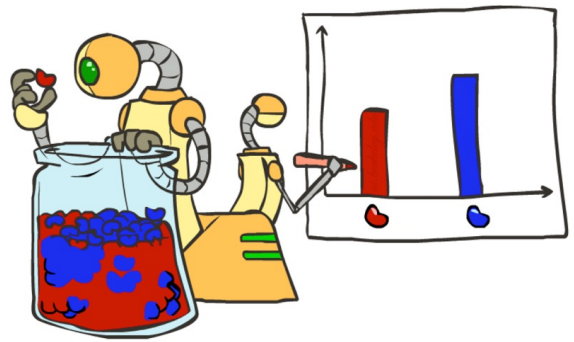
Parameter Estimation



Parameter Estimation with Maximum Likelihood

- Estimating the distribution of a random variable
- Use training data (learning!)
  - For each outcome x, look at the ***empirical rate*** of that value:

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



- Example: probability of x=red given training data (r), (r), (b)

$$P_{\text{ML}}(\text{r}) = 2/3$$

X	red	blue
$P_\theta(X)$	θ	$1-\theta$

- This estimate maximizes the ***likelihood of the data*** for parametric model:

$$L(\theta) = P(red, red, blue \mid \theta) = P_\theta(red) \cdot P_\theta(red) \cdot P_\theta(blue) = \theta^2 \cdot (1-\theta)$$

  - Why?

$$\hat{\theta} = \underset{\theta}{\text{argmax}}\, L(\theta) = \underset{\theta}{\text{argmax}}\, \log L(\theta)$$

$$2\log(\theta) + \log(1-\theta)$$

Take derivative and set to 0:

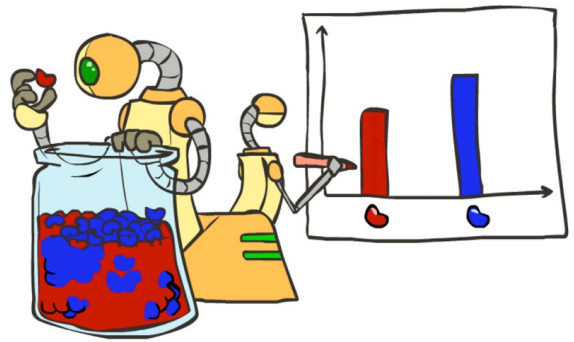
$$\frac{\partial \log L(\theta)}{\partial \theta} = \frac{2}{\theta} + \frac{1}{1-\theta} \cdot -1 = 2(1-\theta) - \theta = 2 - 3\theta = 0$$

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

Parameter Estimation with Maximum Likelihood (General Case)

- **Model**:

X	red	blue
$P_\theta(X)$	θ	$1-\theta$



- **Data:** draw N balls, N_r come up red and N_b come up blue
  - Dataset $D = \{x_1, \ldots, x_N\}$
  - Ball draws are independent and identically distributed (i.i.d):

$$P(D|\theta) = \prod_i P(x_i|\theta) = \prod_i P_\theta(x_i) = \theta^{N_r} \cdot (1-\theta)^{N_b}$$

- **Maximum Likelihood Estimation**: find θ that maximizes $P(D|\theta)$:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, P(D|\theta) = \underset{\theta}{\operatorname{argmax}}\, \log P(D|\theta) \leftarrow N_r \log(\theta) + N_b \log(1-\theta)$$

Take derivative and set to 0:

$$\frac{\partial \log P(D|\theta)}{\partial \theta} = \frac{N_r}{\theta} - \frac{N_b}{1-\theta} = 0$$

$$\rightarrow \hat{\theta} = \frac{N_r}{N_r + N_b} = \frac{\#\text{ of red balls}}{\text{total }\#\text{ of balls}}$$

Parameter Estimation with Maximum Likelihood (General Case)

- **Maximum Likelihood Estimation**: find θ that maximizes $\mathrm{P}(\mathrm{D}|\theta)$:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, P(D|\theta) = \underset{\theta}{\operatorname{argmax}}\, \log P(D|\theta)$$ ← $N_r \log(\theta) + N_b \log(1-\theta)$

Take derivative and set to 0:

$$\frac{\partial}{\partial\theta}\log P(D|\theta) = \frac{\partial}{\partial\theta}\left[N_r \log(\theta) + N_b \log(1-\theta)\right]$$

$$= N_r \frac{\partial}{\partial\theta}[\log(\theta)] + N_b \frac{\partial}{\partial\theta}[\log(1-\theta)]$$

$$= N_r \frac{1}{\theta} + N_b \frac{1}{1-\theta}\cdot -1$$

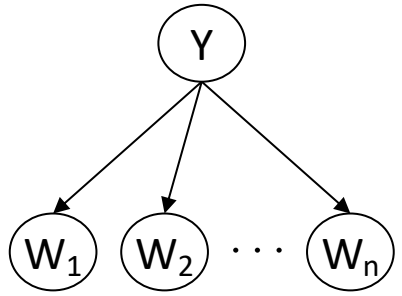
$$= N_r(1-\theta) - N_b\theta$$

$$= N_r - \theta(N_r + N_b) = 0$$

$$\rightarrow \hat{\theta} = \frac{N_r}{N_r + N_b}$$

Example: Spam Filtering Parameter Estimation

- Predict if an email is spam or not
  - Features: W_i is the i'th word in the email (domain: words in the dictionary)
  - Labels: $Y \in \{spam, ham\}$
- Naïve Bayes **model** parameters:



$P_\theta(Y)$

Y	spam	ham
P(Y)	θ_{spam}	$1 - \theta_{spam}$

$P_\theta(W|Y)$

W	"now"	"the"	…
P(W | Y=spam)	$\theta_{"now", spam}$	$\theta_{"the", spam}$	…
P(W | Y=ham)	$\theta_{"now", ham}$	$\theta_{"the", ham}$	…

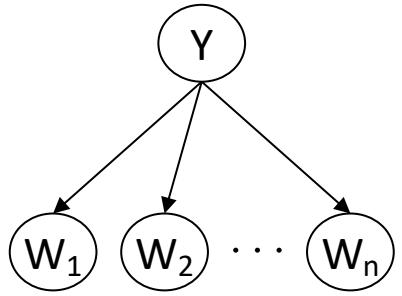
- **Dataset:** M emails where k'th email contains words $\{w_1, \ldots, w_{N_k}\}$ and label y_k
  - Emails are independent and identically distributed:

$$P(D|\theta) = \prod_k^M P(y_k, w_1, \ldots, w_{N_k}) = \prod_k^M P(y_k) \prod_i^{N_k} P(w_i|y_k)$$

- **Maximum Likelihood Estimation**: find θ that maximizes $P(D|\theta)$

Example: Spam Filtering Parameter Estimation

- Predict if an email is spam or not
  - Features: W_i is the i'th word in the email (domain: words in the dictionary)
  - Labels: $Y \in \{spam, ham\}$
- Naïve Bayes **model** parameters:



$P_\theta(Y)$

Y	spam	ham
P(Y)	θ_{spam}	$1 - \theta_{spam}$

$P_\theta(W|Y)$

W	"now"	"the"	…
P(W | Y=spam)	$\theta_{"now", spam}$	$\theta_{"the", spam}$	…
P(W | Y=ham)	$\theta_{"now", ham}$	$\theta_{"the", ham}$	…

- **Maximum Likelihood Estimation Result:**
  - $\theta_{spam} = \frac{\text{\# of spam emails}}{\text{total \# of emails}}$
  - $\theta_{w,spam} = \frac{\text{\# of occurences of word w in spam emails}}{\text{total \# of spam emails}}$
  - $\theta_{w,ham} = \frac{\text{\# of occurences of word w in ham emails}}{\text{total \# of ham emails}}$

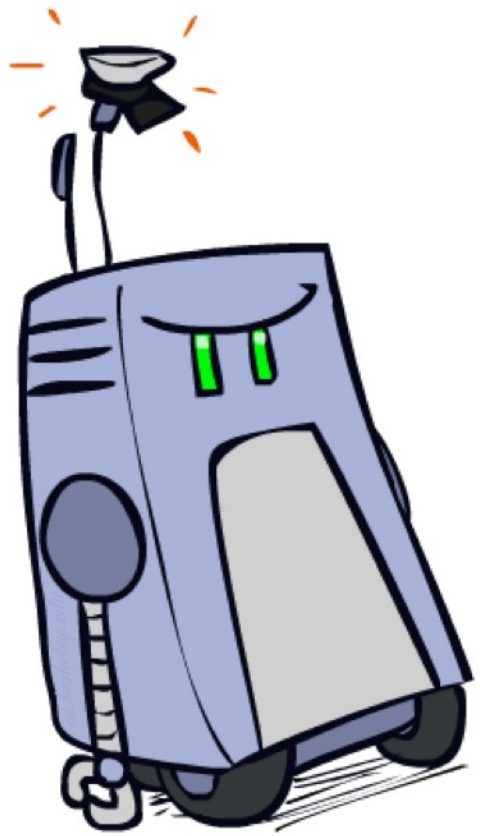
More generally: parameters for model $P_\theta(F|Y)$ are:

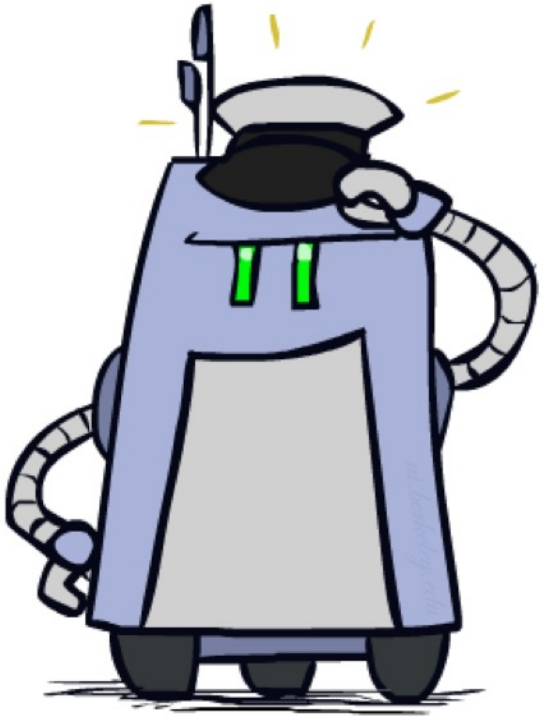
$$\theta_{f,y} = \frac{\text{\# of occurences of feature f and class y}}{\text{total \# of occurences of class y}}$$

Parameter Estimation with Maximum Likelihood

- How do we estimate the conditional probability tables?
  - Maximum Likelihood, which corresponds to counting
- Need to be careful though … let's see what can go wrong..

Underfitting and Overfitting





Example: Overfitting

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

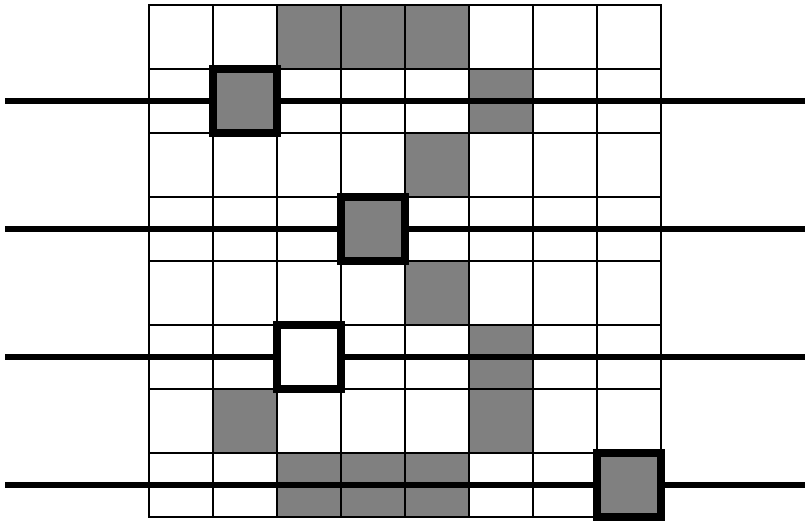
$P(C = 2) = 0.1$

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

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

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

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



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

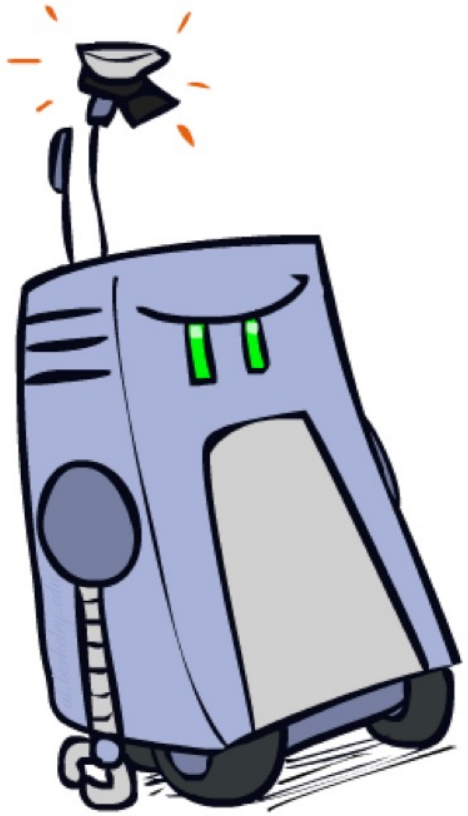
$P(C = 3) = 0.1$

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

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

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

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



2 wins!!

Example: Overfitting

- *relative* probabilities (odds ratios):

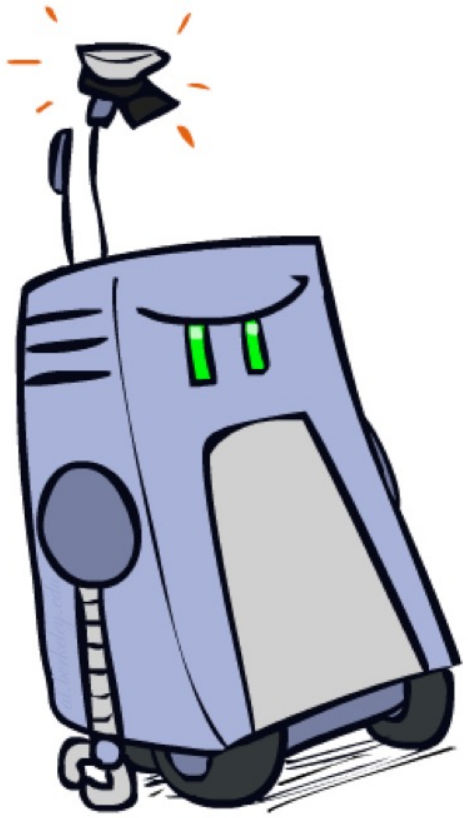
$$\frac{P(W|\text{ham})}{P(W|\text{spam})}$$

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

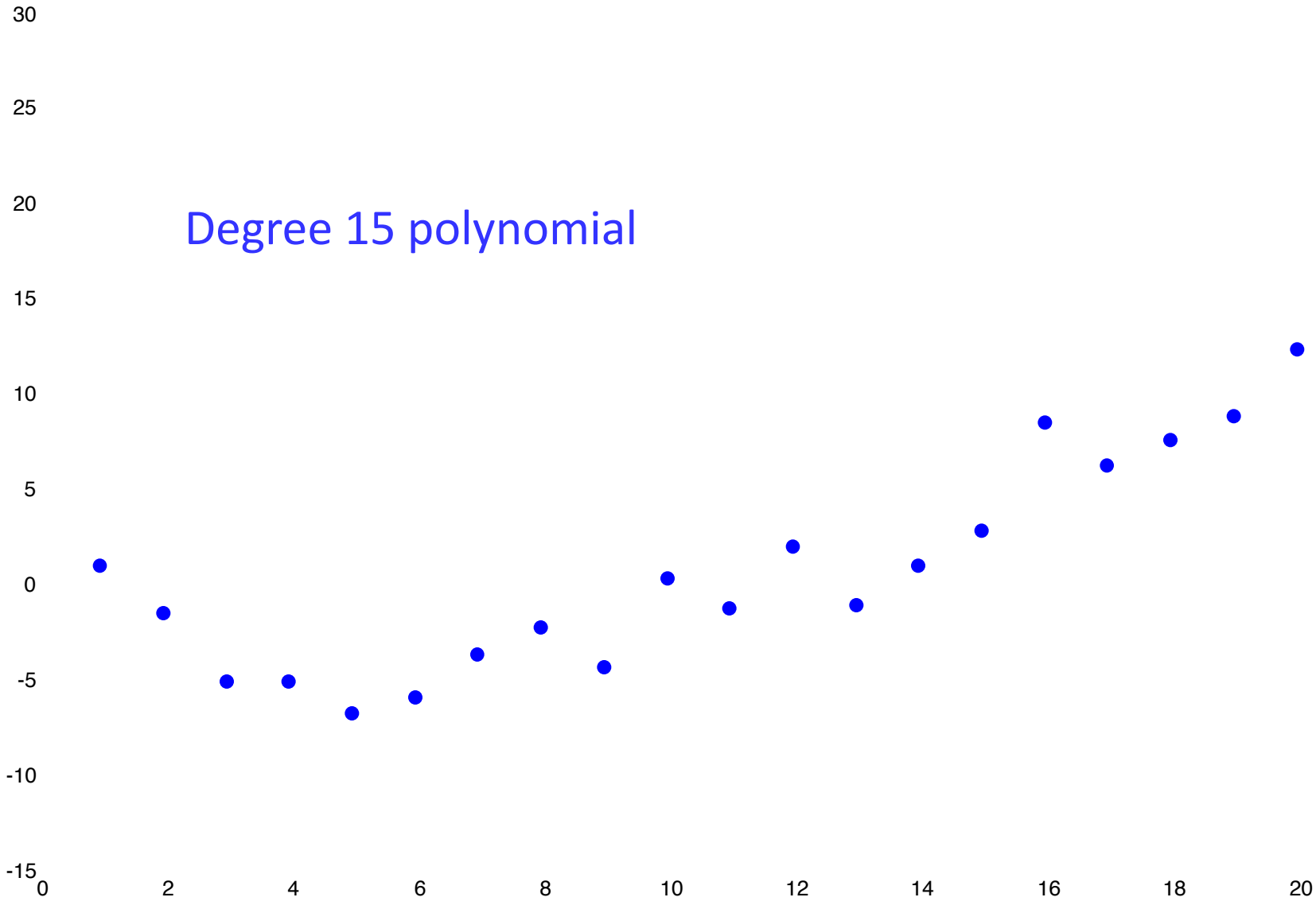
$$\frac{P(W|\text{spam})}{P(W|\text{ham})}$$

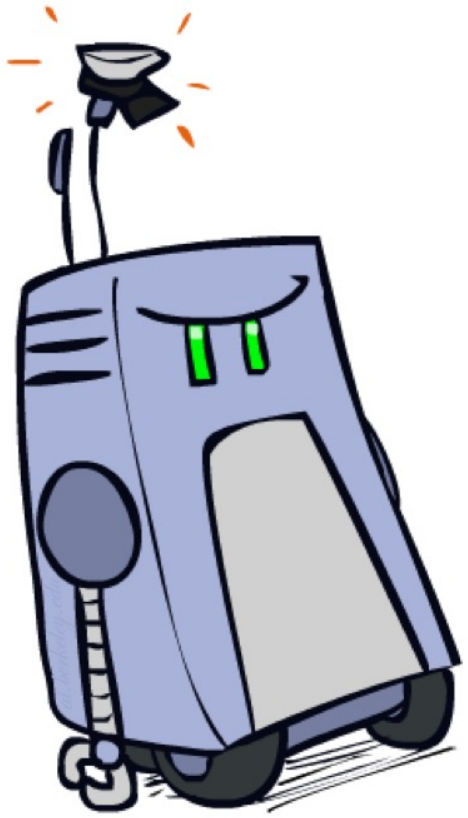
```
screens    : inf
minute     : inf
guaranteed : inf
$205.00    : inf
delivery   : inf
signature  : inf
...
```

What went wrong here?



Overfitting





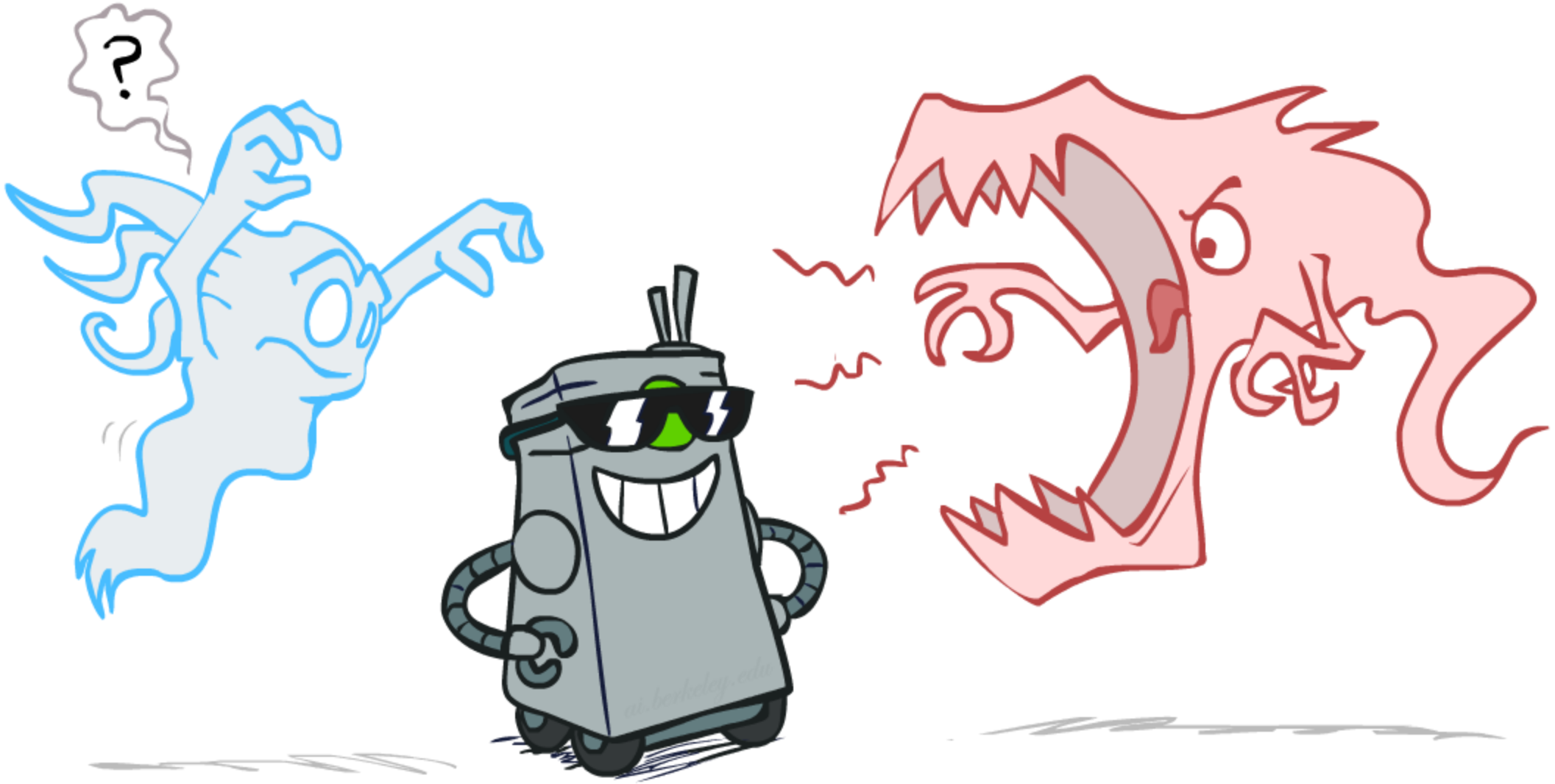
Generalization and Overfitting

- Relative frequency parameters 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
  - Unlikely that every occurrence of "minute" is 100% spam
  - Unlikely that every occurrence of "seriously" is 100% ham
  - What about all the words that don't occur in the training set at all?
  - In general, we can't go around giving unseen events zero probability

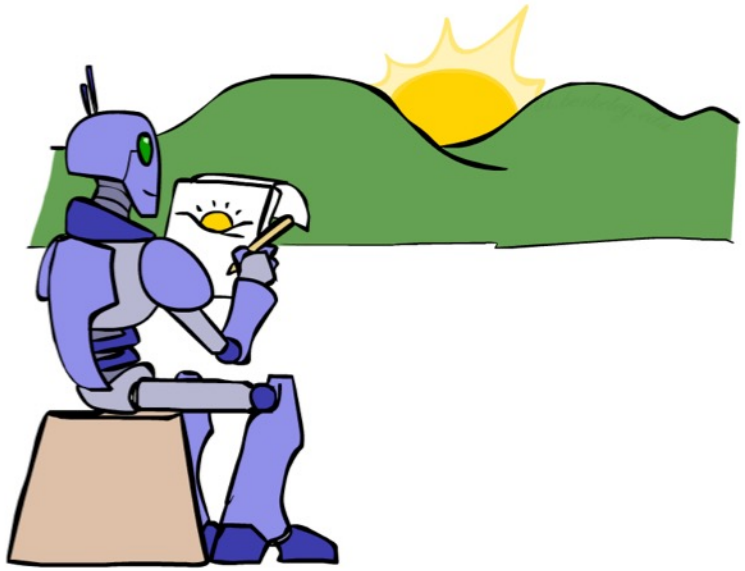
- As an extreme case, imagine using the entire email as the only feature
  - Would get the training data perfect (if deterministic labeling)
  - Wouldn't *generalize* at all
  - Just making the bag-of-words assumption gives us some generalization, but isn't enough

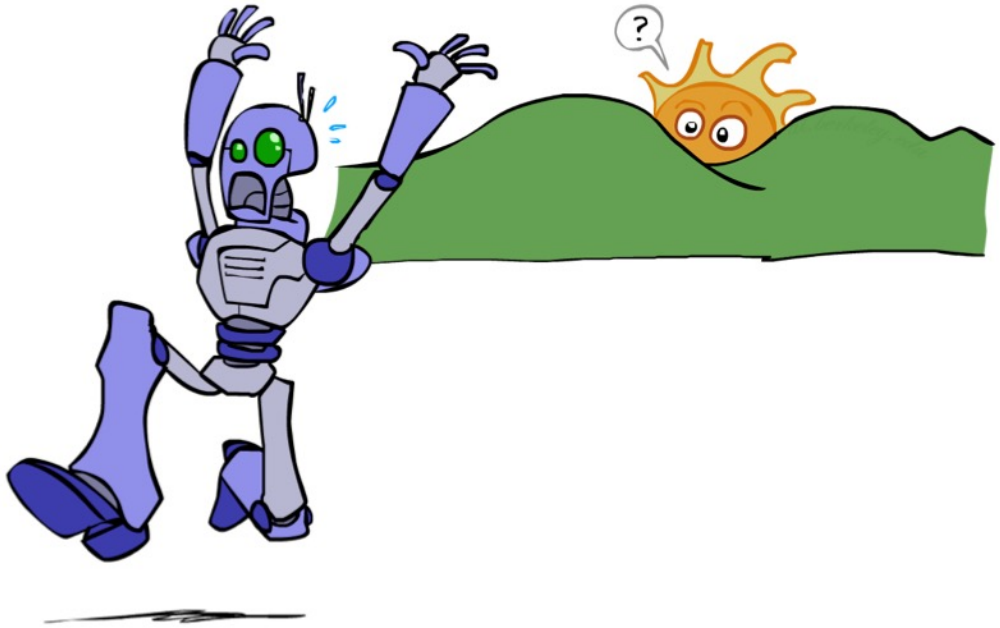
- To generalize better: we need to smooth or regularize the estimates

Smoothing



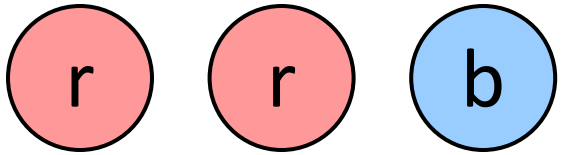
Unseen Events





Laplace Smoothing

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



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

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

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

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

  - Can derive this estimate with *Dirichlet priors* (see cs281a)

Laplace Smoothing

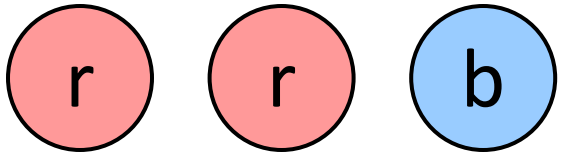
- Laplace's estimate (extended):
  - Pretend you saw every outcome k extra times

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

  - 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 NB: Smoothing

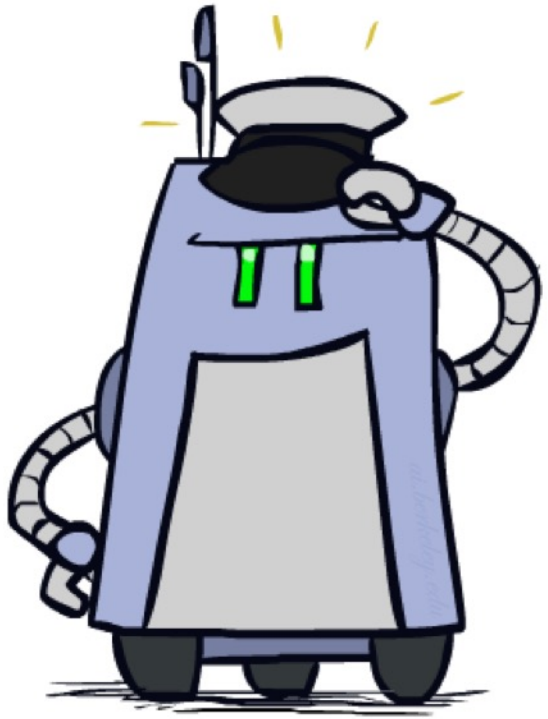
- For real classification problems, smoothing is critical
- New odds ratios:

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

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

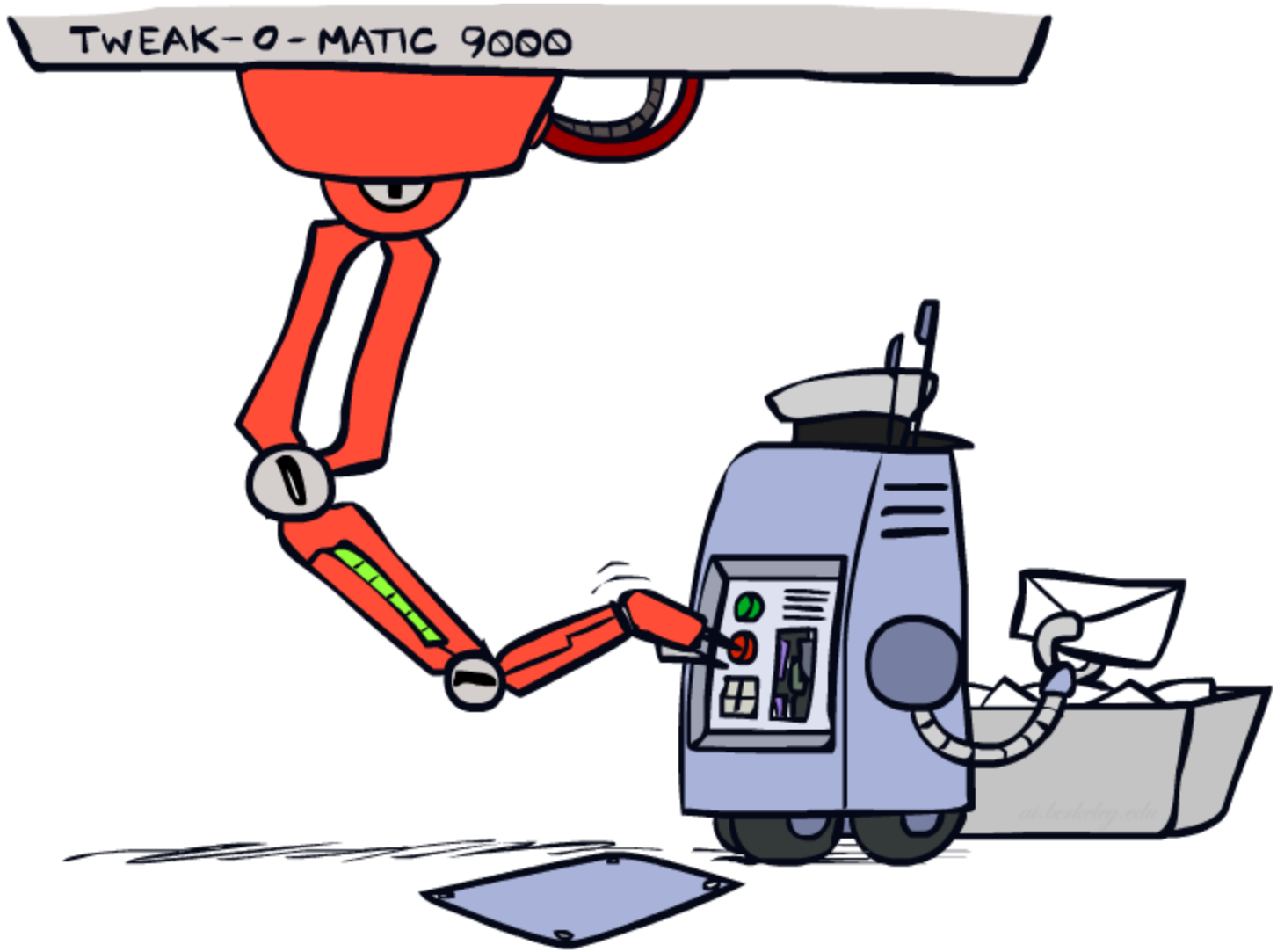
$$\frac{P(W|\text{spam})}{P(W|\text{ham})}$$

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



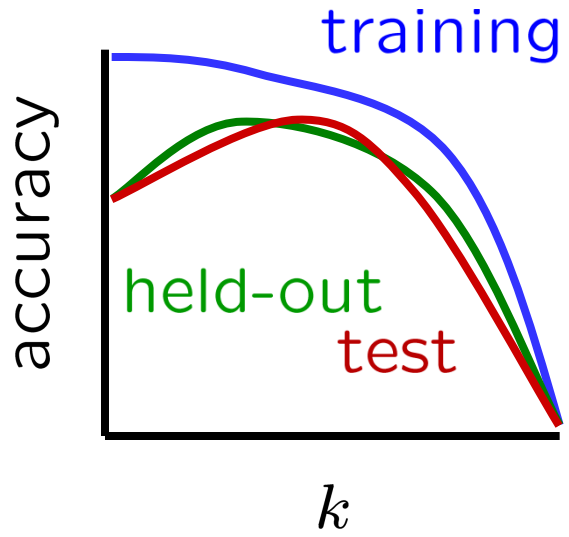
Do these make more sense?

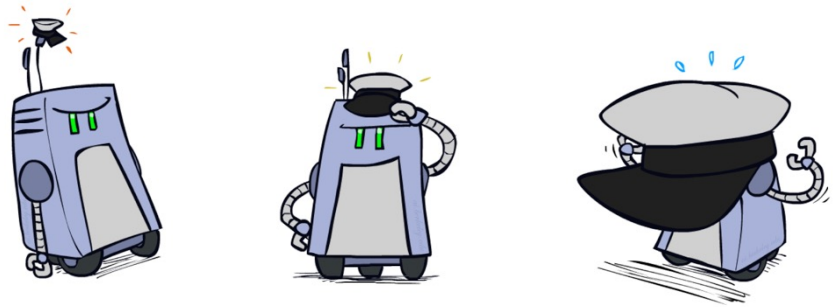
Tuning



Tuning on Held-Out Data

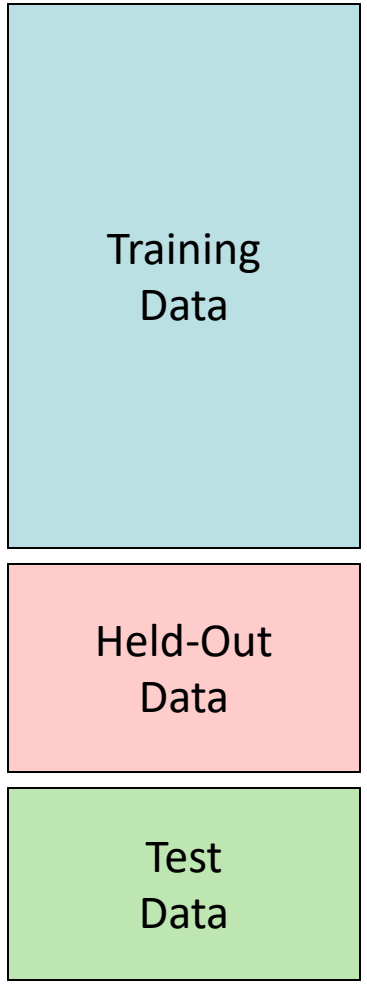
- Now we've got two kinds of unknowns
  - Parameters: the probabilities P(X|Y), P(Y)
  - Hyperparameters: e.g. the amount of smoothing *k*
- What should we learn where?
  - Learn parameters from training data
  - Tune hyperparameters on different data
    - Why?
  - For each value of the hyperparameters,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



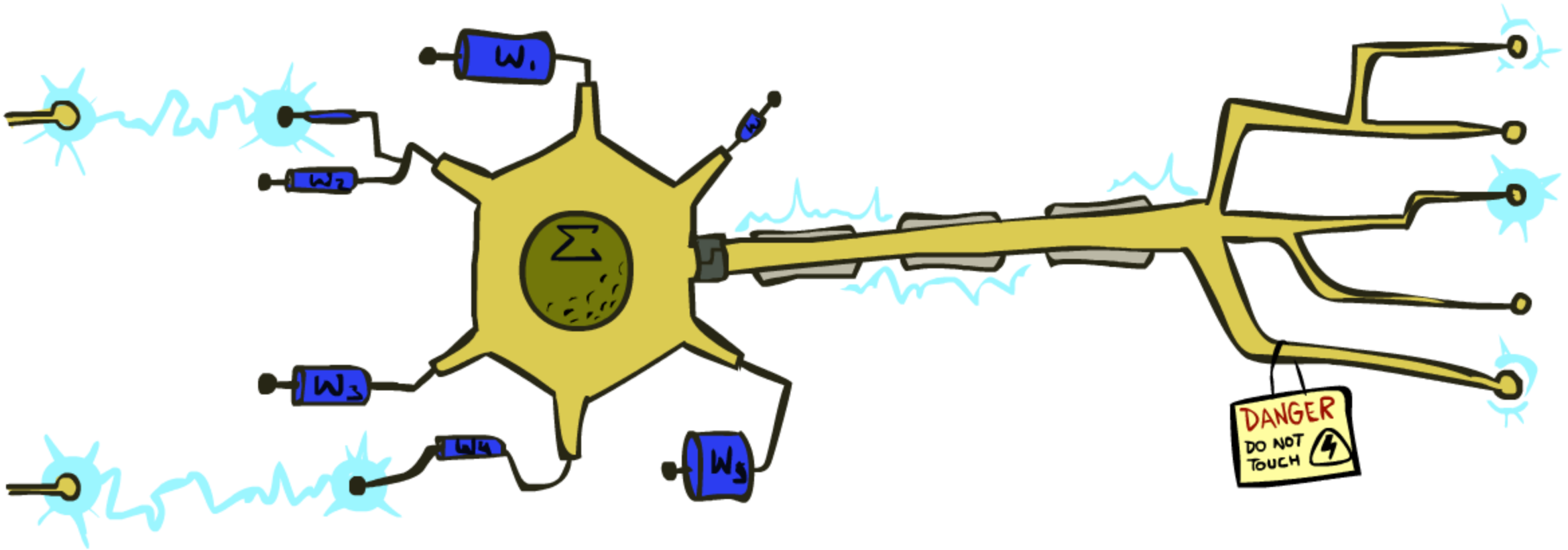


Important Concepts

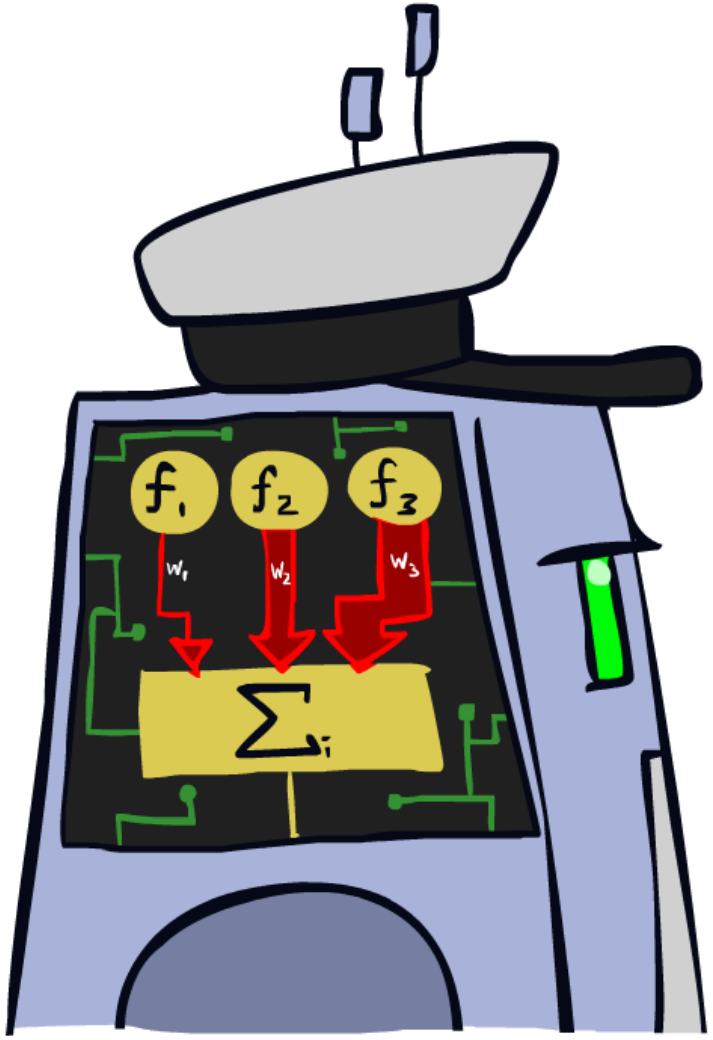
- Data: labeled instances, e.g. emails marked spam/ham
  - Training set
  - Held out set
  - Test set
- Features: attribute-value pairs which characterize each input
- Experimentation cycle
  - Learn parameters (e.g. model probabilities) on training set
  - (Tune hyperparameters on held-out set)
  - Compute accuracy on test set
  - Very important: never "peek" at the test set!
- Evaluation
  - 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
  - Underfitting: fits the training set poorly



Perceptrons



Linear Classifiers



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
...
```

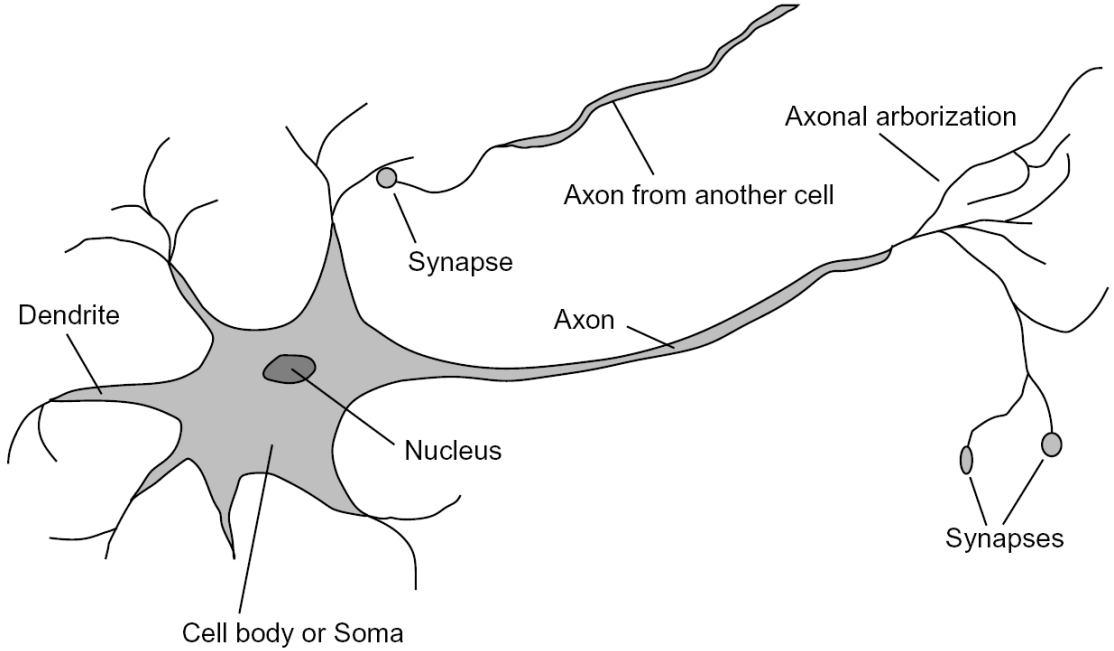
SPAM
or
HAM

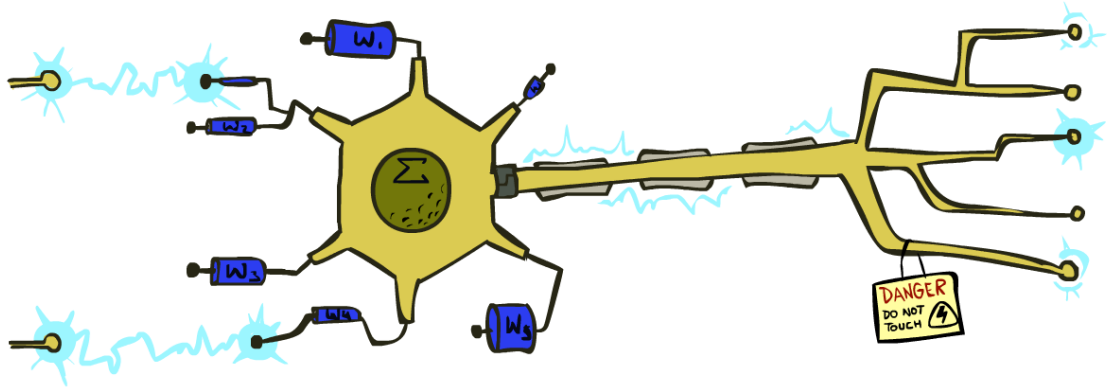
```
PIXEL-7,12   : 1
PIXEL-7,13   : 0
...
NUM_LOOPS    : 1
...
```

“2”

Some (Simplified) Biology

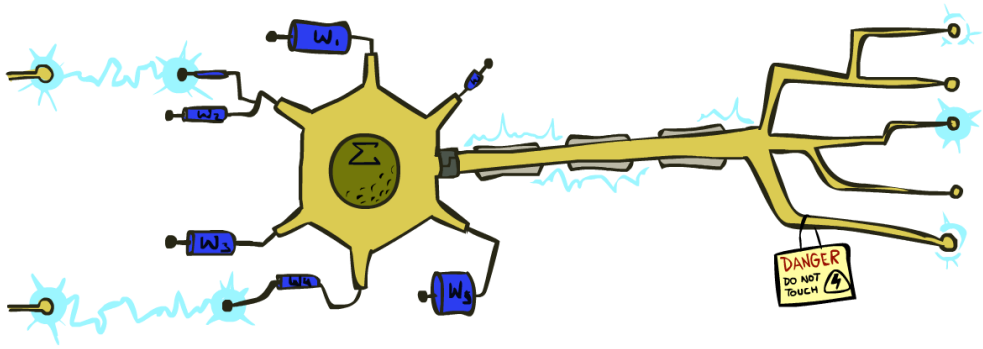
- Very loose inspiration: human neurons





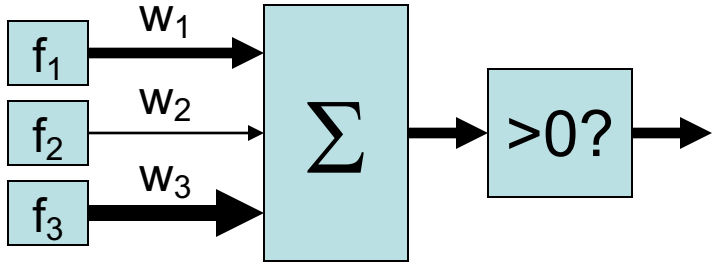
Linear Classifiers

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



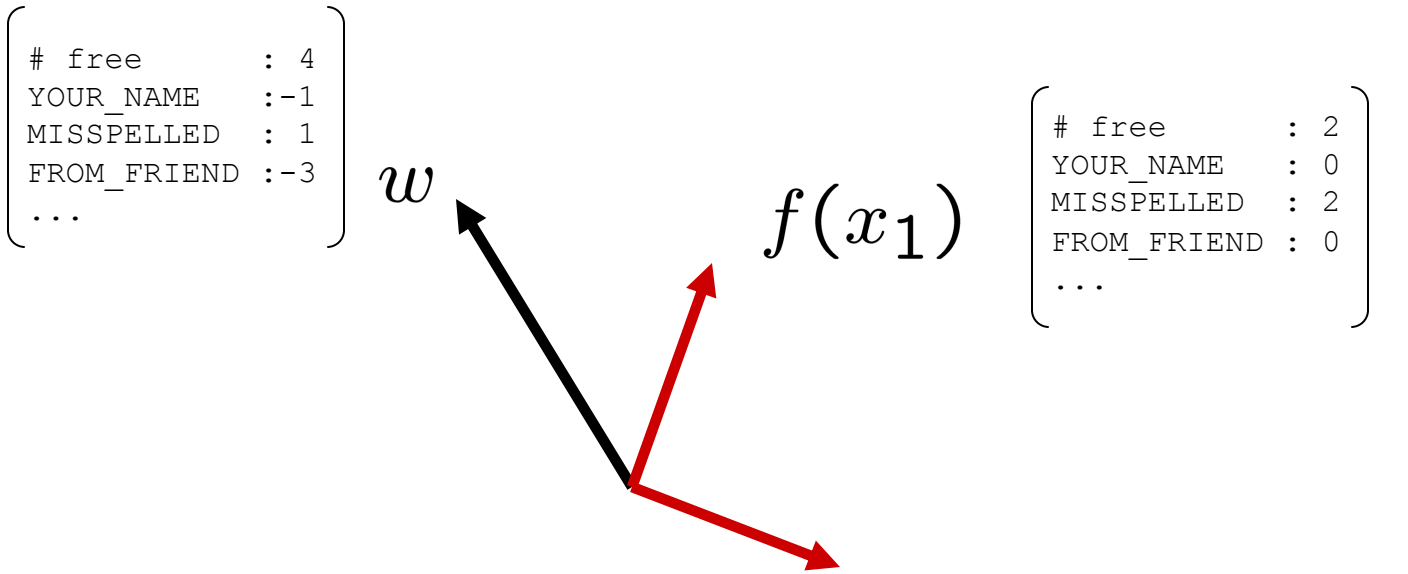
$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- If the activation is:
  - Positive, output +1
  - Negative, output -1



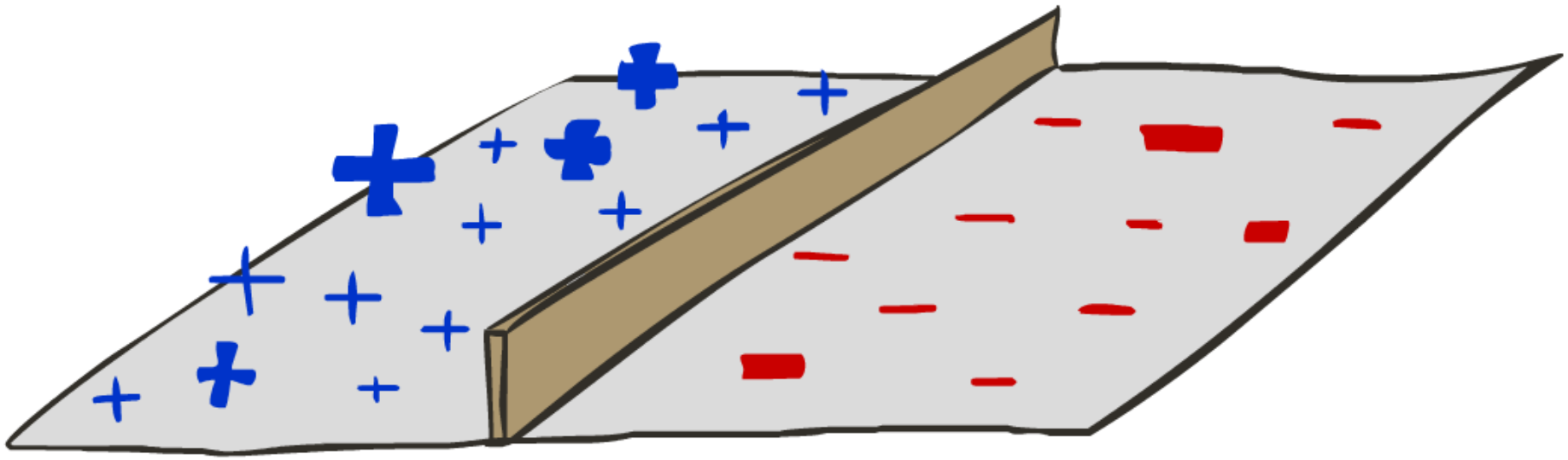
Weights

- Binary case: compare features to a weight vector
- Learning: figure out the weight vector from examples



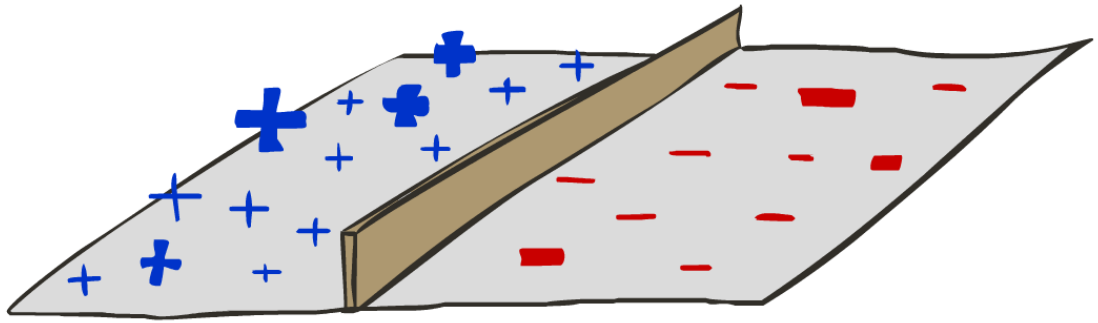
Dot product $w \cdot f$ positive means the positive class

Decision Rules



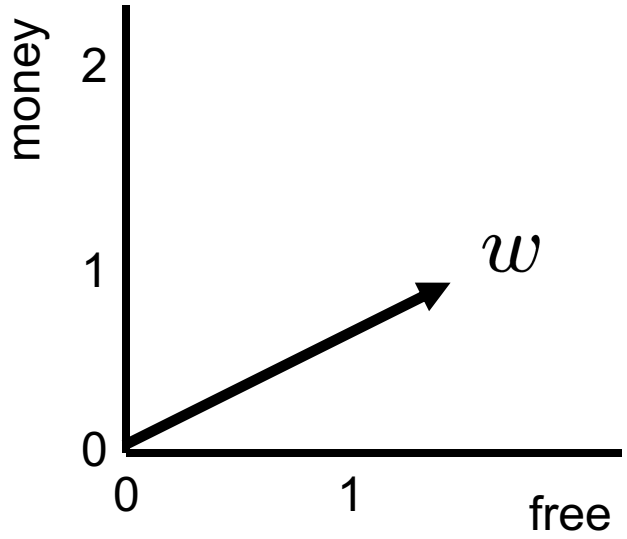
Binary Decision Rule

- In the space of feature vectors
  - Examples are points
  - Any weight vector is a hyperplane
  - One side corresponds to Y=+1
  - Other corresponds to Y=-1



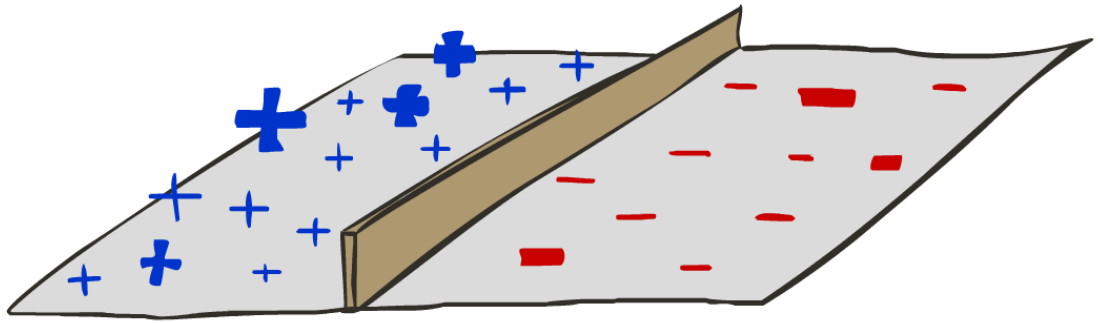
w

```
free   :   4
money  :   2
```



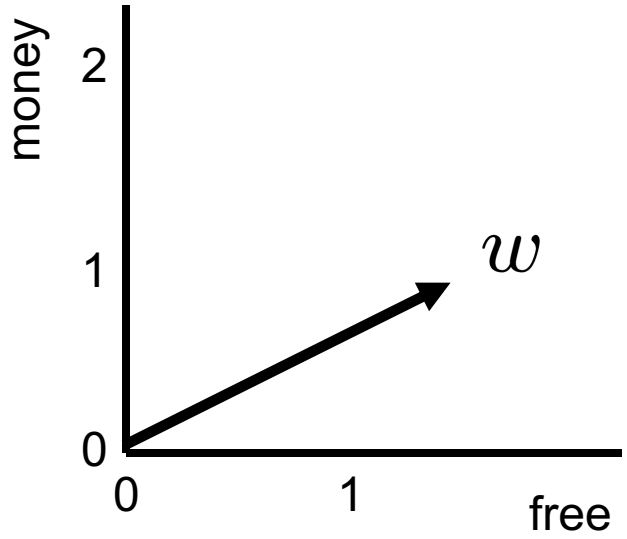
Binary Decision Rule

- In the space of feature vectors
  - Examples are points
  - Any weight vector is a hyperplane
  - One side corresponds to Y=+1
  - Other corresponds to Y=-1



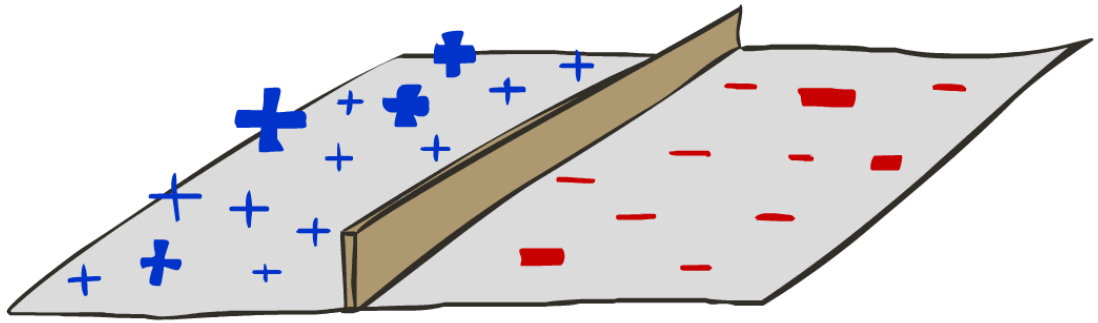
w

```
BIAS   : -3
free   :  4
money  :  2
...
```



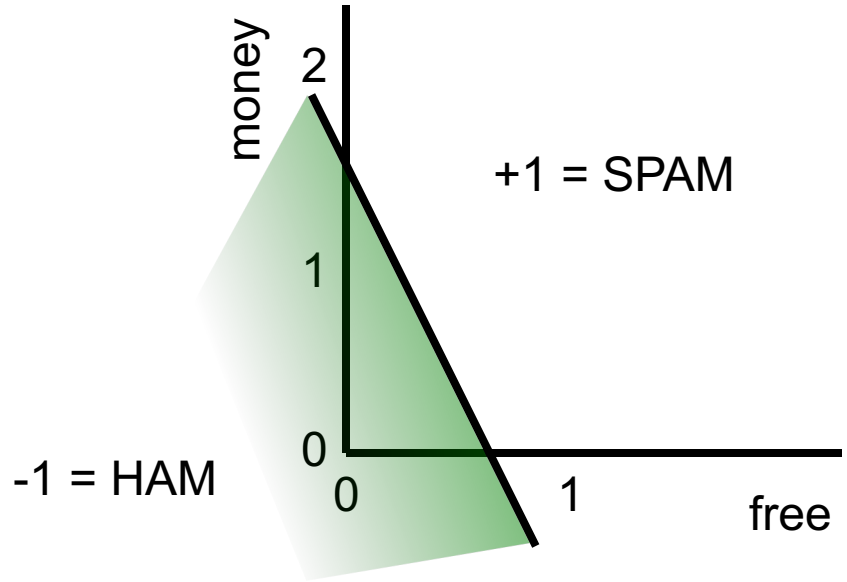
Binary Decision Rule

- In the space of feature vectors
  - Examples are points
  - Any weight vector is a hyperplane
  - One side corresponds to Y=+1
  - Other corresponds to Y=-1



w

```
BIAS   : -3
free   :  4
money  :  2
...
```

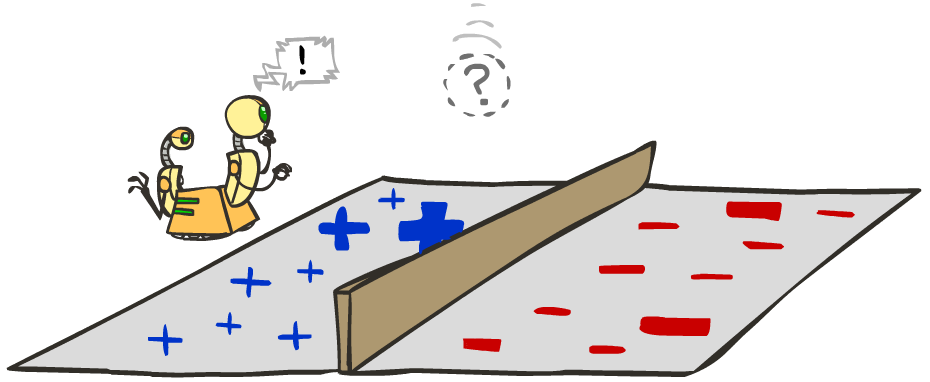


$f \cdot w = 0$

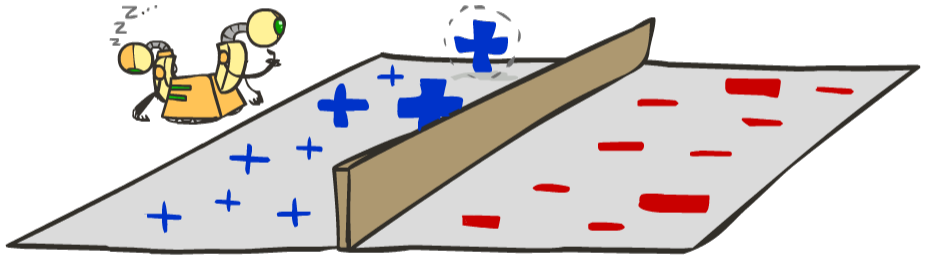
Weight Updates

Learning: Binary Perceptron

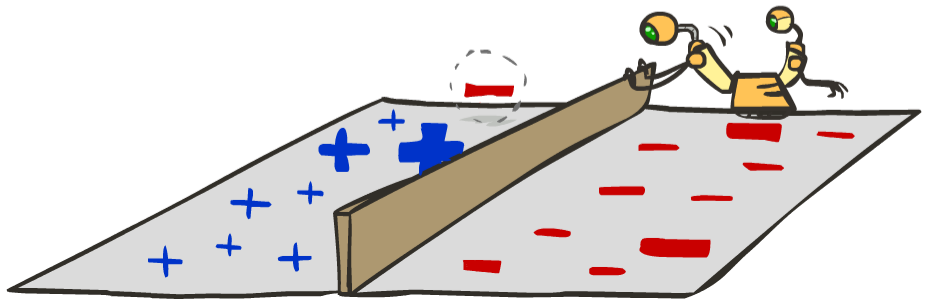
- Start with weights w = 0
- For each training instance f(x), y*:
  - Classify with current weights



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



  - If wrong: adjust the weight vector



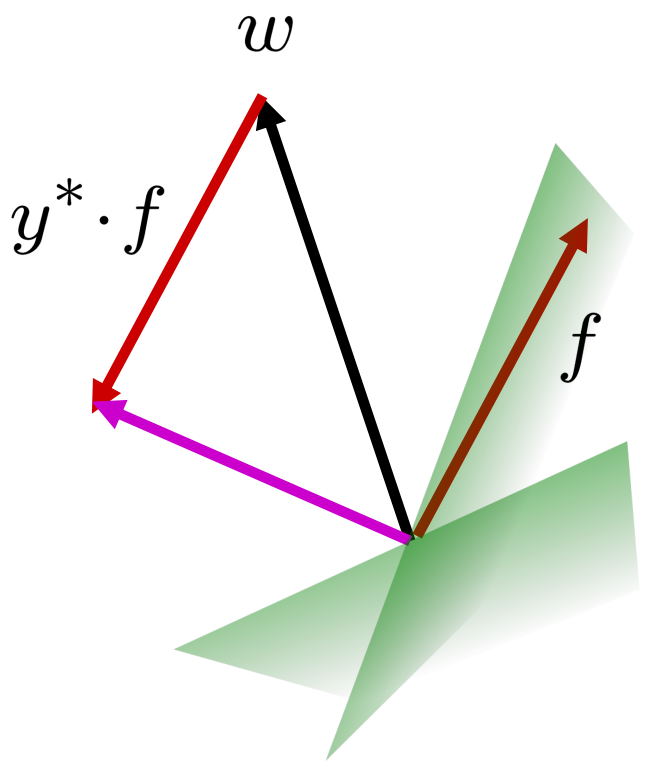
Learning: Binary Perceptron

- Start with weights w = 0
- For each training instance f(x), y*:
  - 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. Subtract if y* is -1.

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



Before: $w f$
After: $wf + y^* f f$
$f f >= 0$

Example: Perceptron

Iteration 0: x: *"win the vote"* f(x): [1 1 0 1 1] y*: -1

Iteration 1: x: *"win the election"* f(x): [1 1 0 0 1] y*: -1

Iteration 2: x: *"win the game"* f(x): [1 1 1 0 1] y*: +1

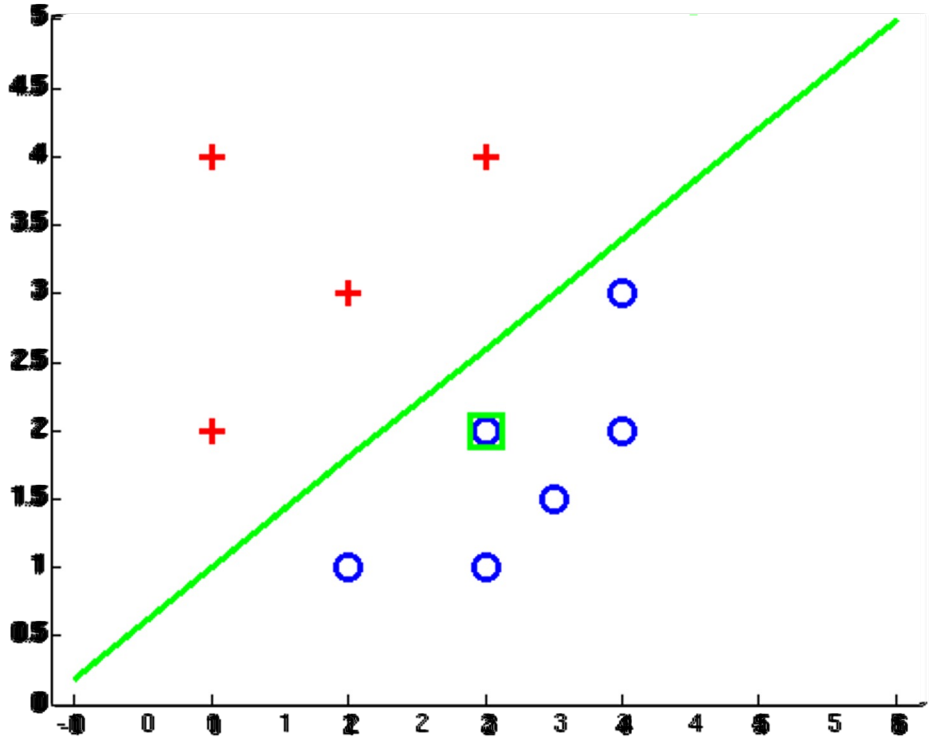
Iteration 3: x: *"win the game"* f(x): [1 1 1 0 1] y*: +1

w

BIAS	1	0	0	1
win	0	-1	-1	0
game	0	0	0	1
vote	0	-1	-1	-1
the	0	-1	-1	0
$w \cdot f(x)$:	1	-2	-2	2

Example: Perceptron

- Separable Case



Multiclass Decision Rule

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

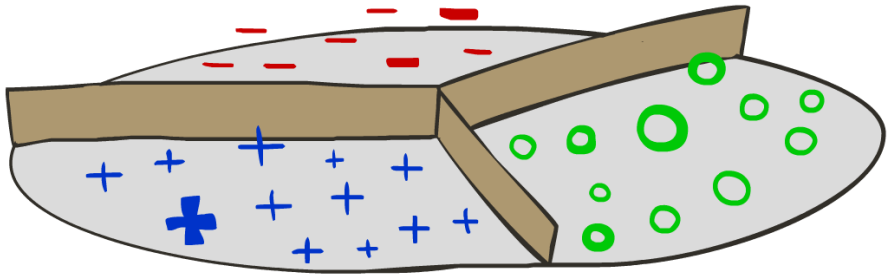
$$w_y$$

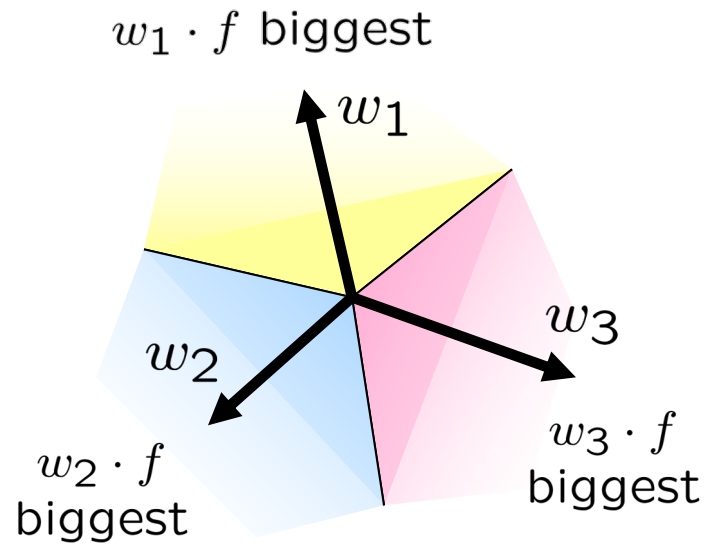
  - Score (activation) of a class y:

$$w_y \cdot f(x)$$

  - 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

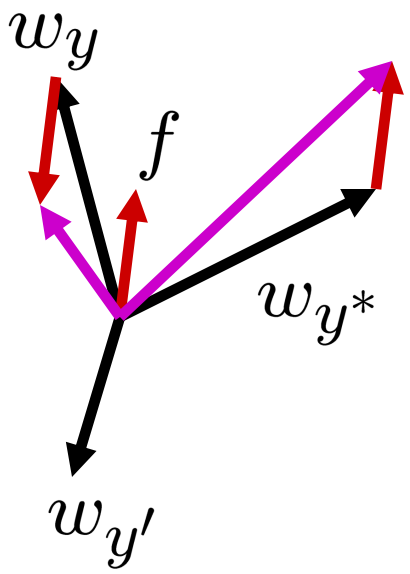
- Start with all weights = 0
- Pick up training examples f(x), y* one by one
- Predict with current weights

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

- 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

Iteration 0: x: *"win the vote"* f(x): [1 1 0 1 1] y*: politics

Iteration 1: x: *"win the election"* f(x): [1 1 0 0 1] y*: politics

Iteration 2: x: *"win the game"* f(x): [1 1 1 0 1] y*: sports

w_{SPORTS}

BIAS	1	0	0	1
win	0	-1	-1	0
game	0	0	0	1
vote	0	-1	-1	-1
the	0	-1	-1	0

$w \cdot f(x)$: 1 -2 -2

$w_{POLITICS}$

BIAS	0	1	1	0
win	0	1	1	0
game	0	0	0	-1
vote	0	1	1	1
the	0	1	1	0

$w \cdot f(x)$: 0 3 3

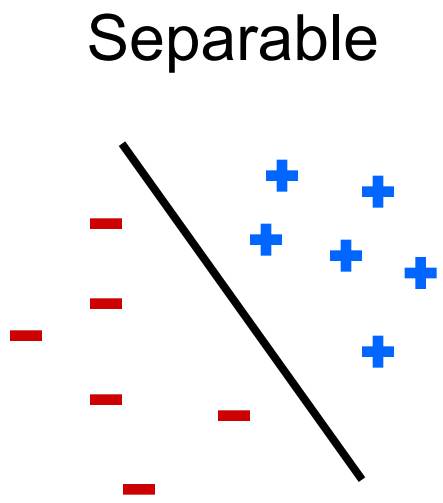
w_{TECH}

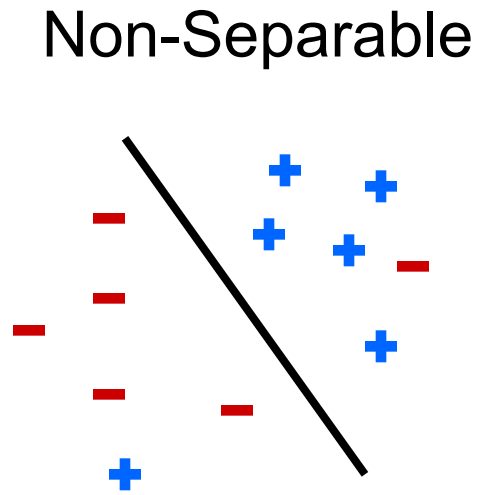
BIAS	0	0	0	0
win	0	0	0	0
game	0	0	0	0
vote	0	0	0	0
the	0	0	0	0

$w \cdot f(x)$: 0 0 0

Properties of Perceptrons

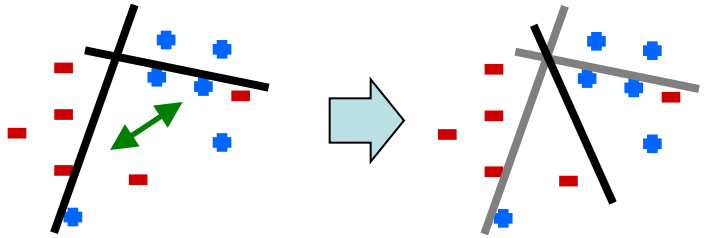
- Separability: true if some parameters get the training set perfectly correct
- Convergence: if the training is separable, perceptron will eventually converge (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

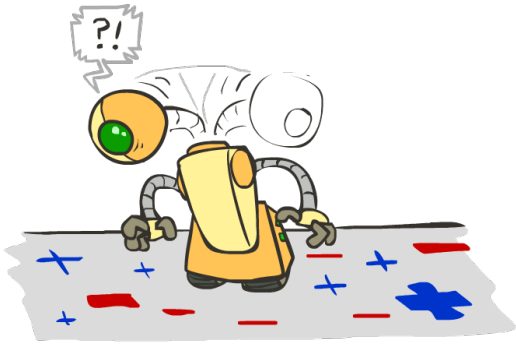




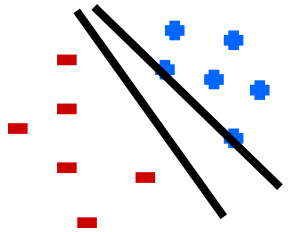
Problems with the Perceptron

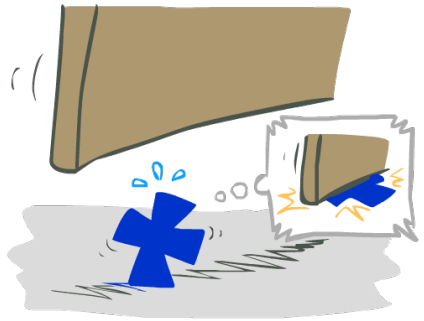
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



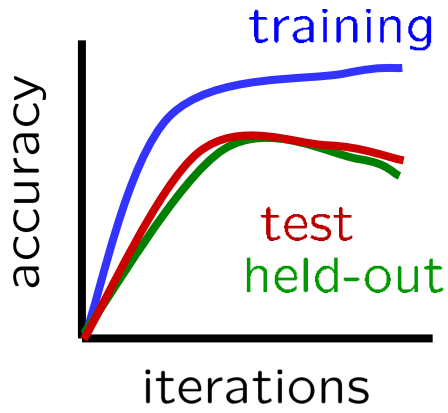


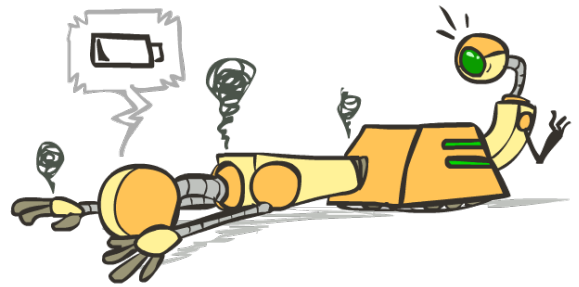
- Mediocre generalization: finds a "barely" separating solution





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





Next Lecture: Improving Perceptron & Optimization