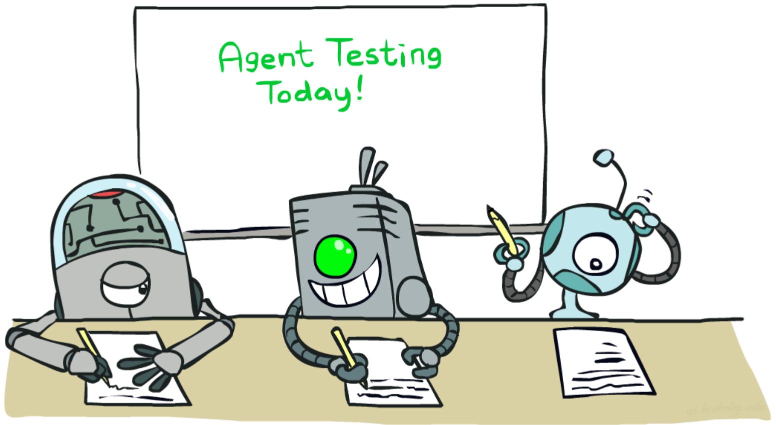
CSE 573 : Artificial Intelligence

Hanna Hajishirzi

Machine Learning, Perceptrons, and Logistic Regression

Part 2



slides adapted from
Dan Klein, Pieter Abbeel ai.berkeley.edu
And Dan Weld, Luke Zettlemoyer

Recap: Machine Learning

- Up until now: how use a model to make optimal decisions
- Machine learning: how to acquire a model from data / experience
  - Learning parameters (e.g. probabilities)
  - Learning structure (e.g. graphs)
  - Learning hidden concepts (e.g. clustering)
- First: model-based classification

Recap: Spam Filter

- Input: an email
- Output: spam/ham

- Setup:
  - Get a large collection of example emails, each labeled "spam" or "ham"
  - Note: someone has to hand label all this data!
  - 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, WidelyBroadcast
  - …

✗

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.

Recap: Feature Vectors in Linear Classifier

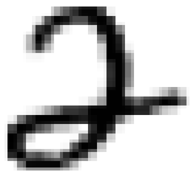
x | $f(x)$ | y

```
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
+



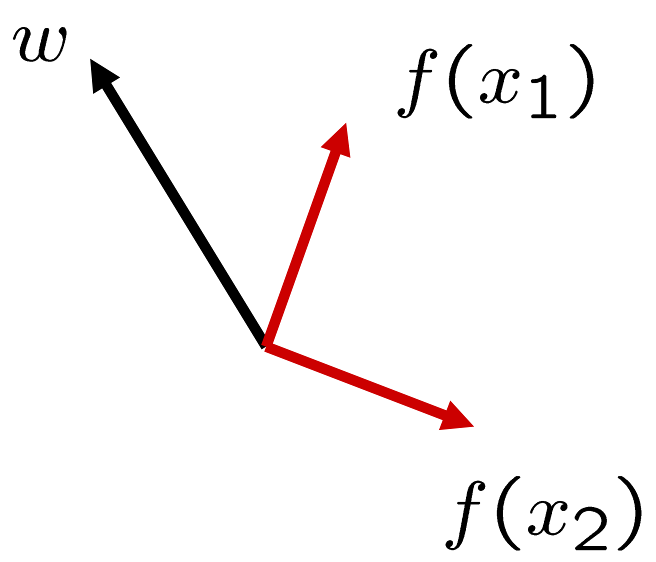
```
PIXEL-7,12  : 1
PIXEL-7,13  : 0
...
NUM_LOOPS   : 1
...
```

“2”

Recap: Weights

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

```
# free       : 4
YOUR_NAME    :-1
MISSPELLED   : 1
FROM_FRIEND  :-3
...
```



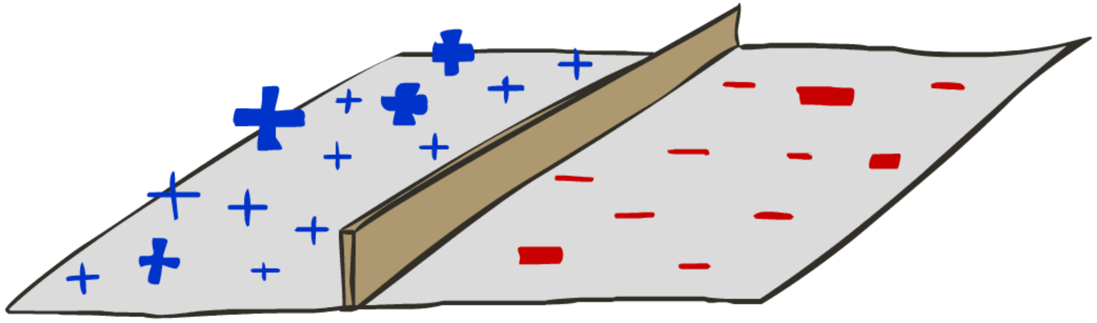

```
# free       : 2
YOUR_NAME    : 0
MISSPELLED   : 2
FROM_FRIEND  : 0
...
```

```
# free       : 0
YOUR_NAME    : 1
MISSPELLED   : 1
FROM_FRIEND  : 1
...
```

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

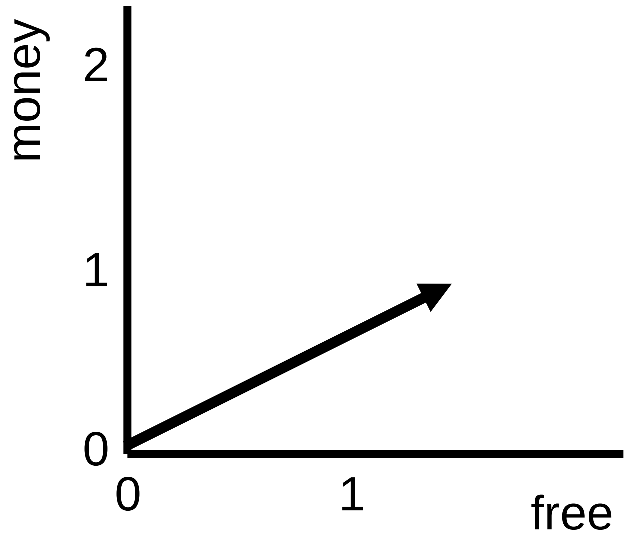
Recap: Binary Decision Rule

- 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
...
```

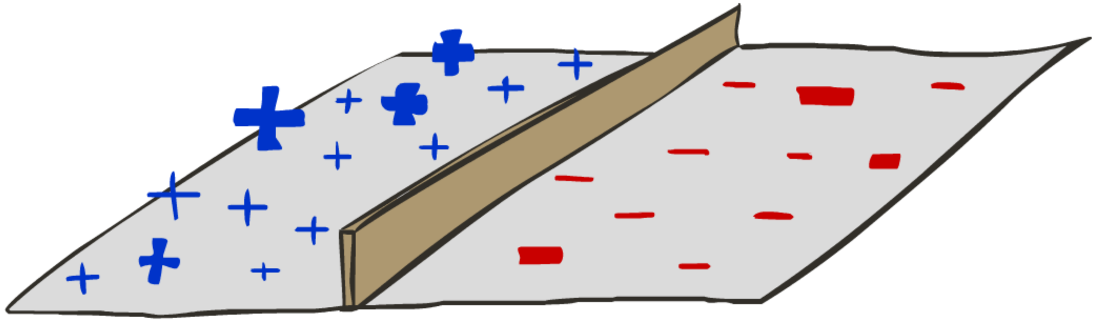


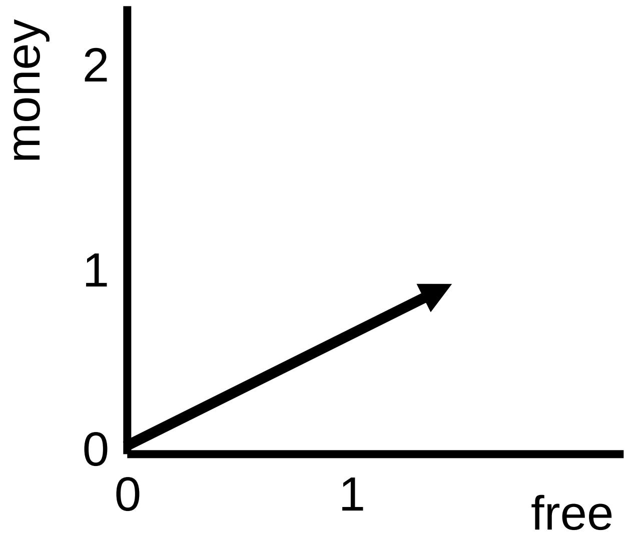
Recap: Binary Decision Rule

- 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

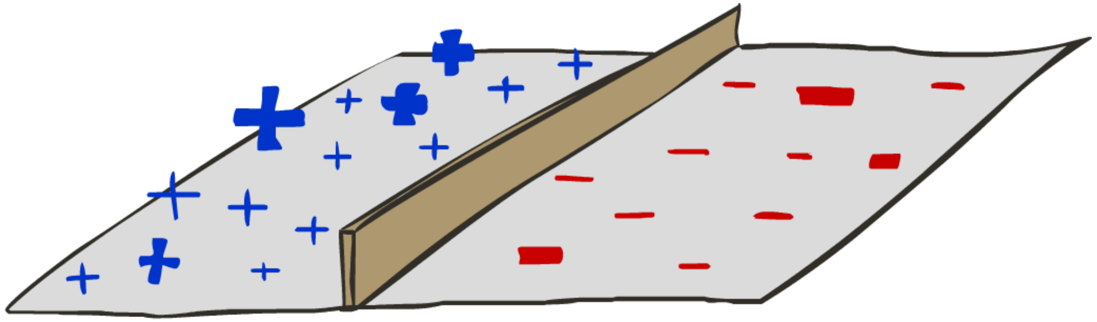
free	:	4
money	:	2





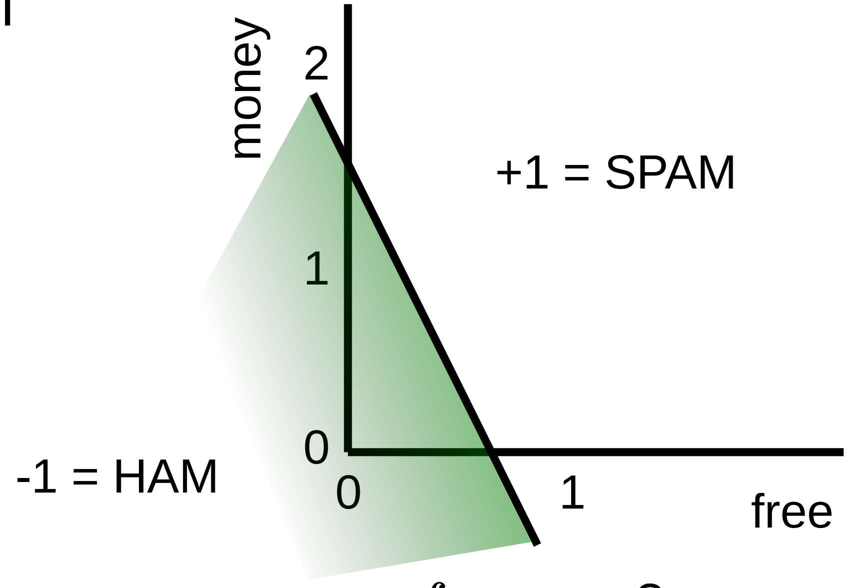
Recap: Binary Decision Rule

- 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

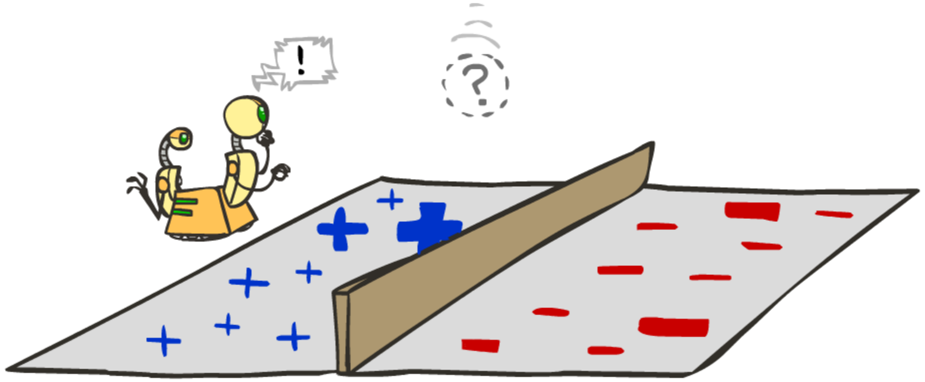
```
BIAS  : -3
free  :  4
money :  2
...
```



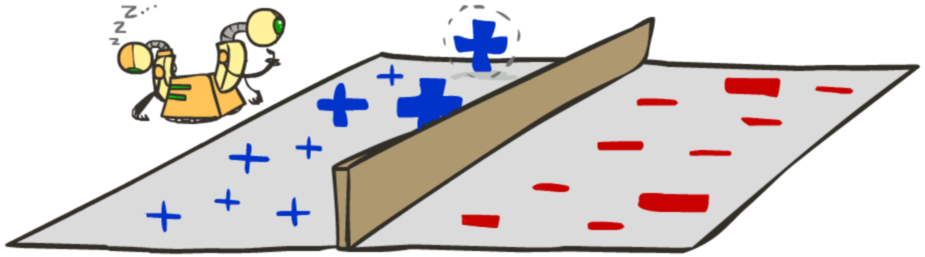
$f \cdot w = 0$

Recap: Binary Perceptron

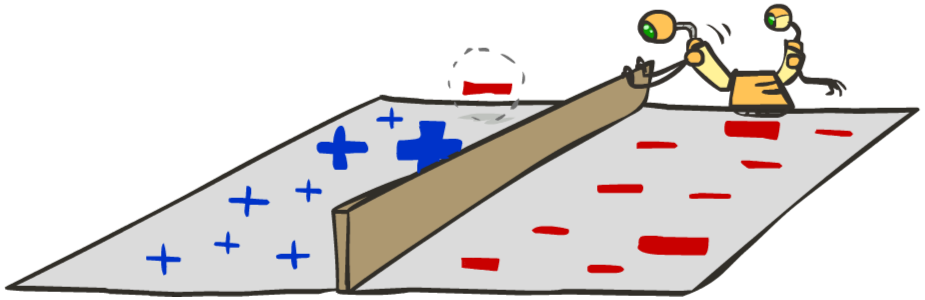
- Start with weights = 0
- For each training instance:
  - Classify with current weights



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



  - If wrong: adjust the weight vector



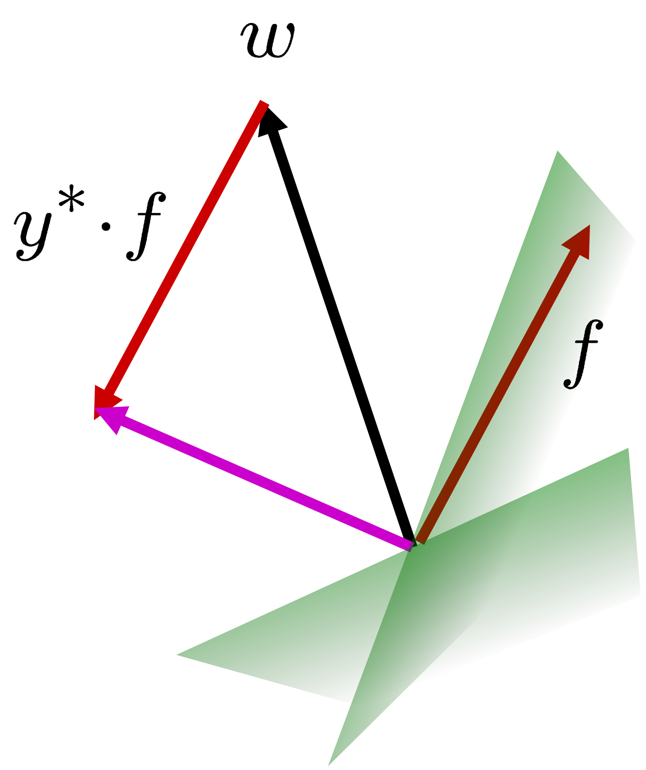
Recap: Binary Perceptron

- 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. Subtract if y* is -1.

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



Recap: Multiclass Decision Rule

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

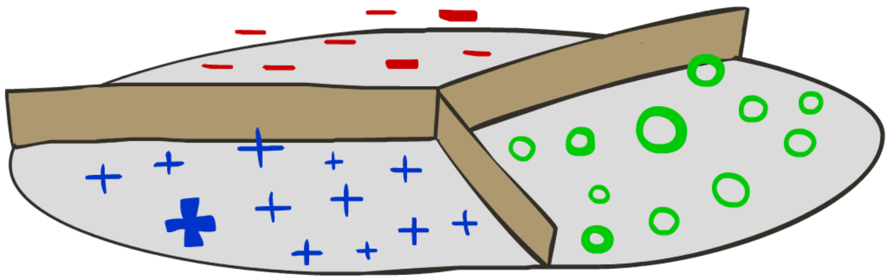
$$w_y$$

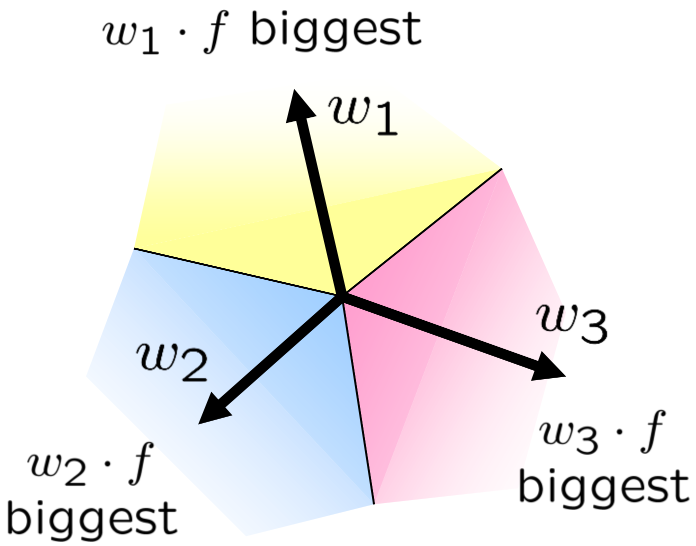
  - Score (activation) of a class y:

$$w_y \cdot f(x)$$

  - Prediction highest score wins

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





Binary = multiclass where the negative class has weight zero

Recap: Multiclass Perceptron

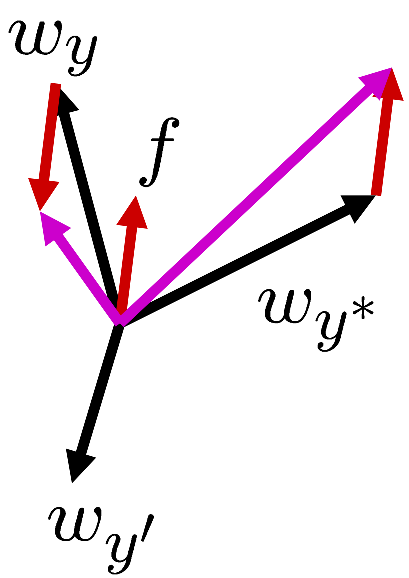
- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights

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

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

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

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



Recap: Multiclass Perceptron

"win the vote" [1 1 0 1 1]

"win the election" [1 1 0 0 1]

"win the game" [1 1 1 0 1]

w_{SPORTS}

1 -2 -2

```
BIAS  : 1
win   : 0
game  : 0
vote  : 0
the   : 0
...
```

0	1
-1	0
0	1
-1	-1
-1	0

$w_{POLITICS}$

0 3 3

```
BIAS  : 0
win   : 0
game  : 0
vote  : 0
the   : 0
...
```

1	0
1	0
0	-1
1	1
1	0

w_{TECH}

0 0

```
BIAS  : 0
win   : 0
game  : 0
vote  : 0
the   : 0
...
```

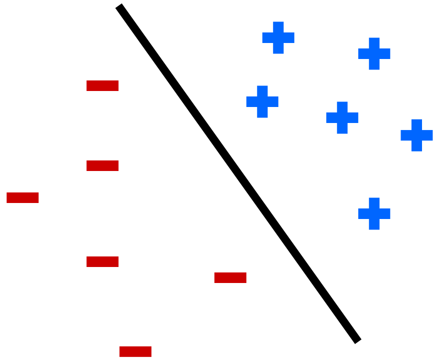
Properties of Perceptrons

- Separability: true if some parameters get the training set perfectly correct

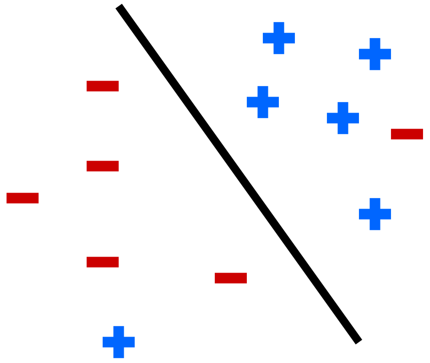
- Convergence: if the training is separable, perceptron will eventually converge (binary case)

- Non-separable?

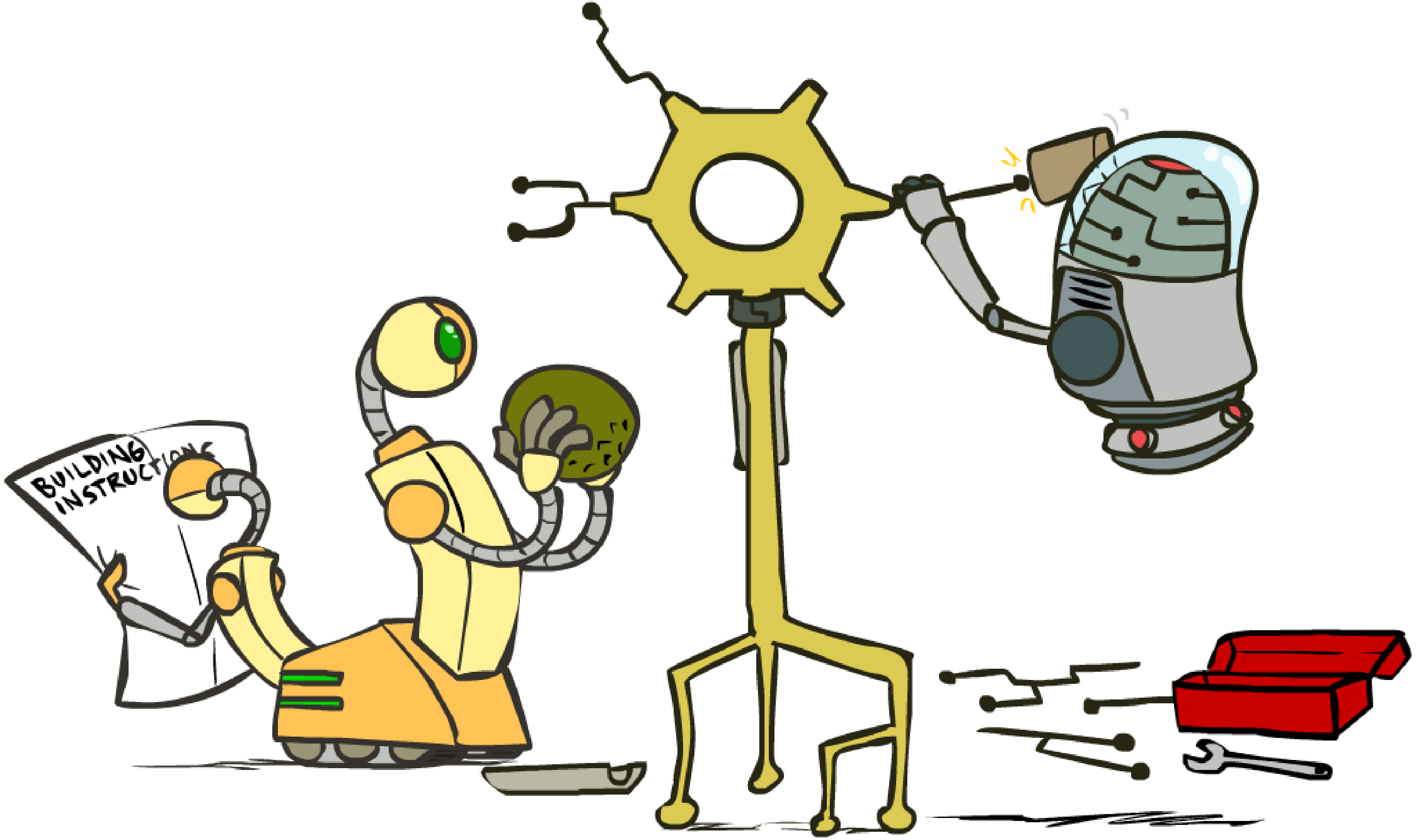
Separable



Non-Separable



Workflow



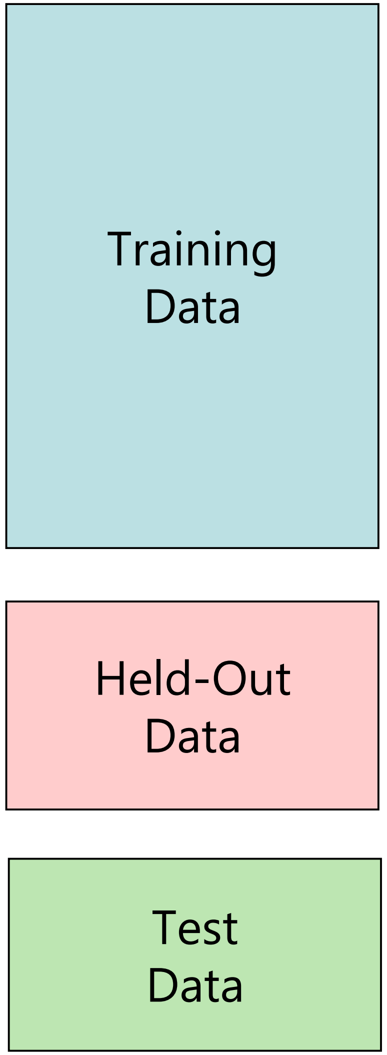
Workflow

- **Phase 1: Train model on Training Data. Choice points for "tuning"**
  - Attributes / Features
  - Model types: Naïve Bayes vs. Perceptron vs. Logistic Regression vs. Neural Net etc..
  - Model hyperparameters
    - E.g. Naïve Bayes – Laplace k
    - E.g. Logistic Regression – weight regularization
    - E.g. Neural Net – architecture, learning rate, …
  - Make sure good performance on training data (why?)

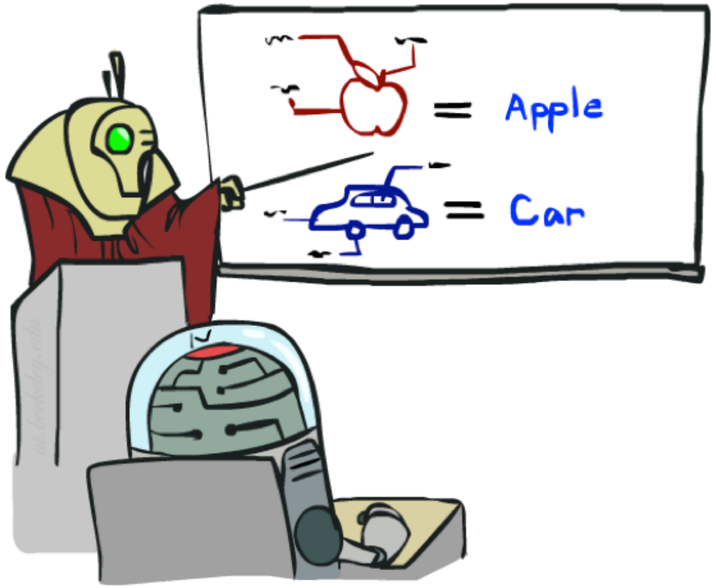
- **Phase 2: Evaluate on Hold-Out Data**
  - If Hold-Out performance is close to Train performance
    - We achieved good generalization, onto Phase 3! ☺
  - If Hold-Out performance is much worse than Train performance
    - We overfitted to the training data! ☹
    - Take inspiration from the errors and:
      - Either: go back to Phase 1 for tuning (typically: make the model less expressive)
      - Or: if we are out of options for tuning while maintaining high train accuracy, collect more data (i.e., let the data drive generalization, rather than the tuning/regularization) and go to Phase 1

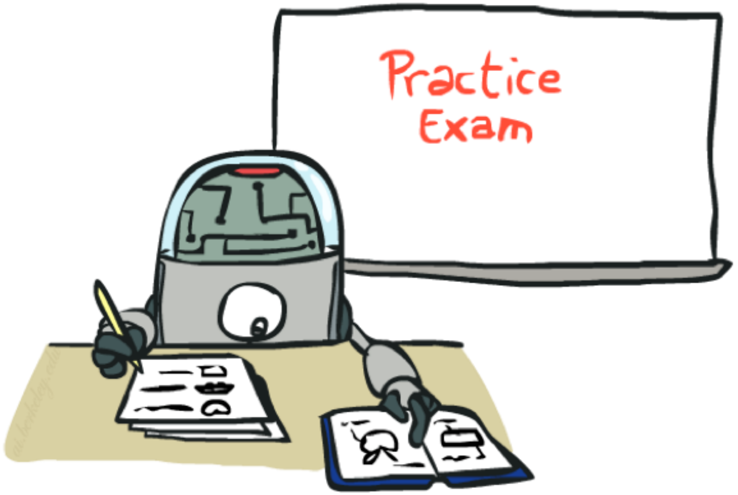
- **Phase 3: Report performance on Test Data**

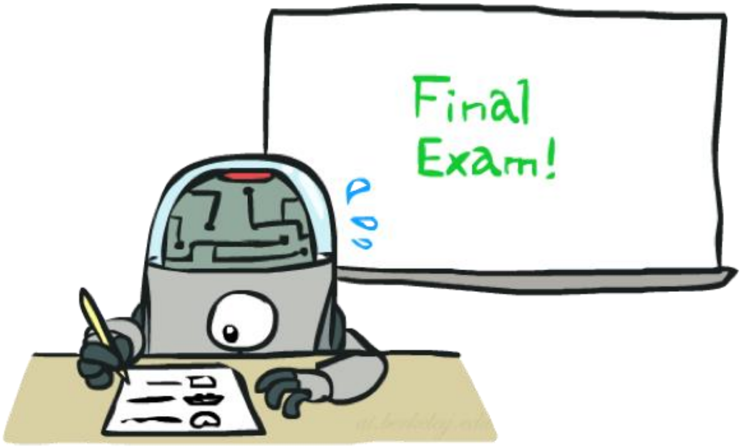
Possible outer-loop: Collect more data ☺



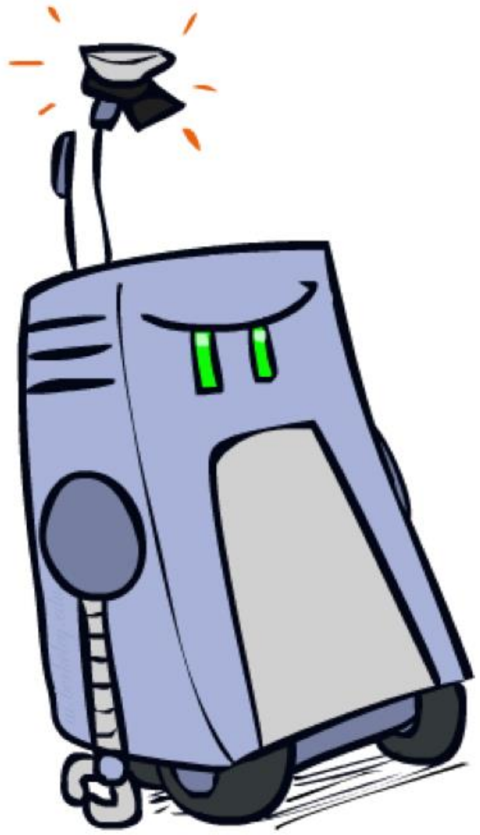
Training and Testing

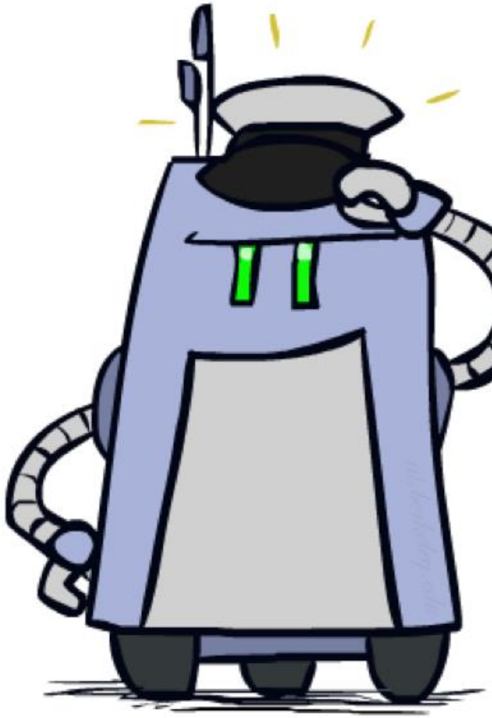






Underfitting and Overfitting





Overfitting

- Too many features
  - Spam if contains “FREE!”
  - Spam if contains $dd, CAPS
  - ...
  - Spam if contains “Sir”
  - Spam if contains address
  - Spam if contains “OT”
  - ...

Dear Sir.

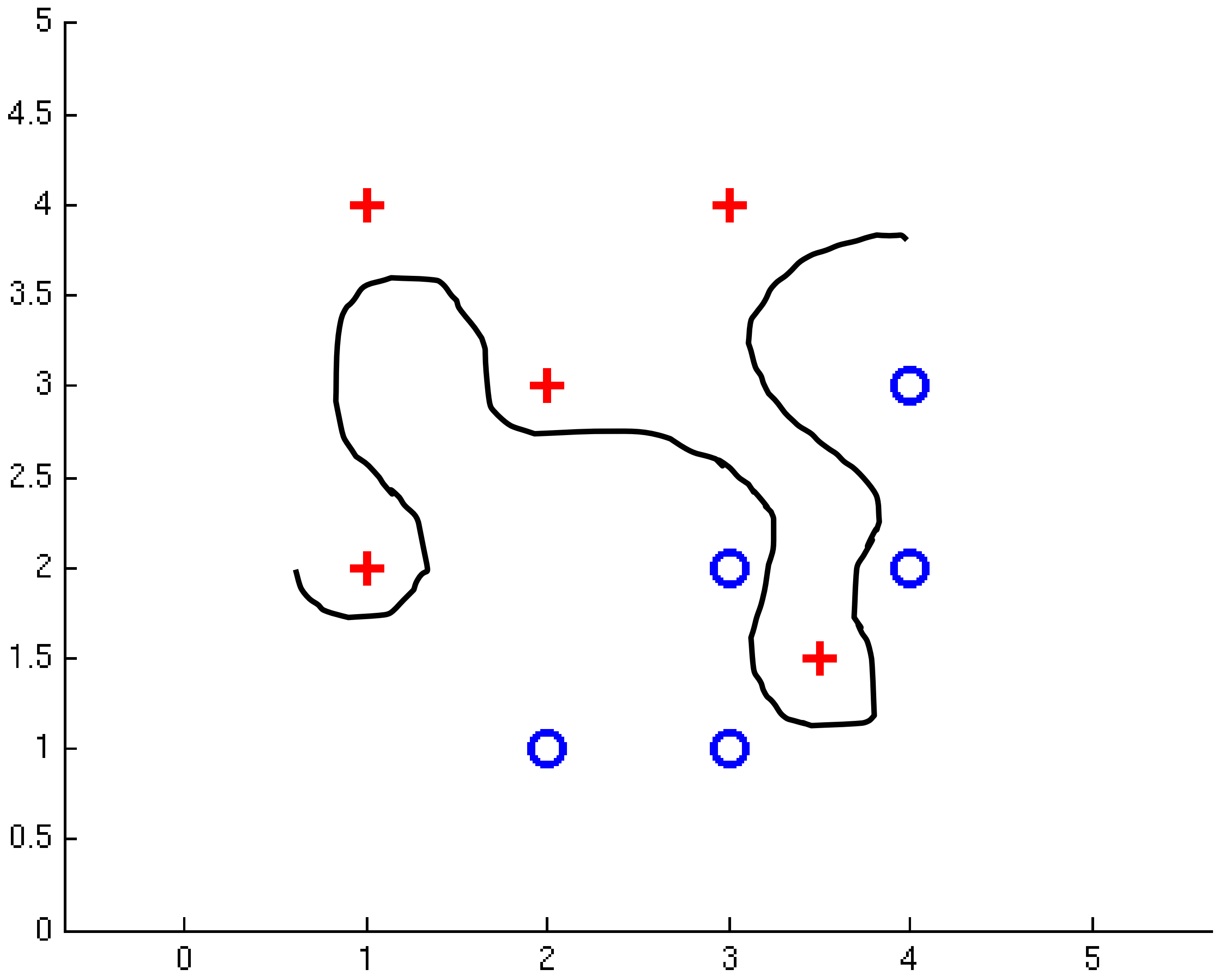
First, I must solicit your confidence in this transaction, this is by virture of its nature as being utterly confidencial and top secret. …

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

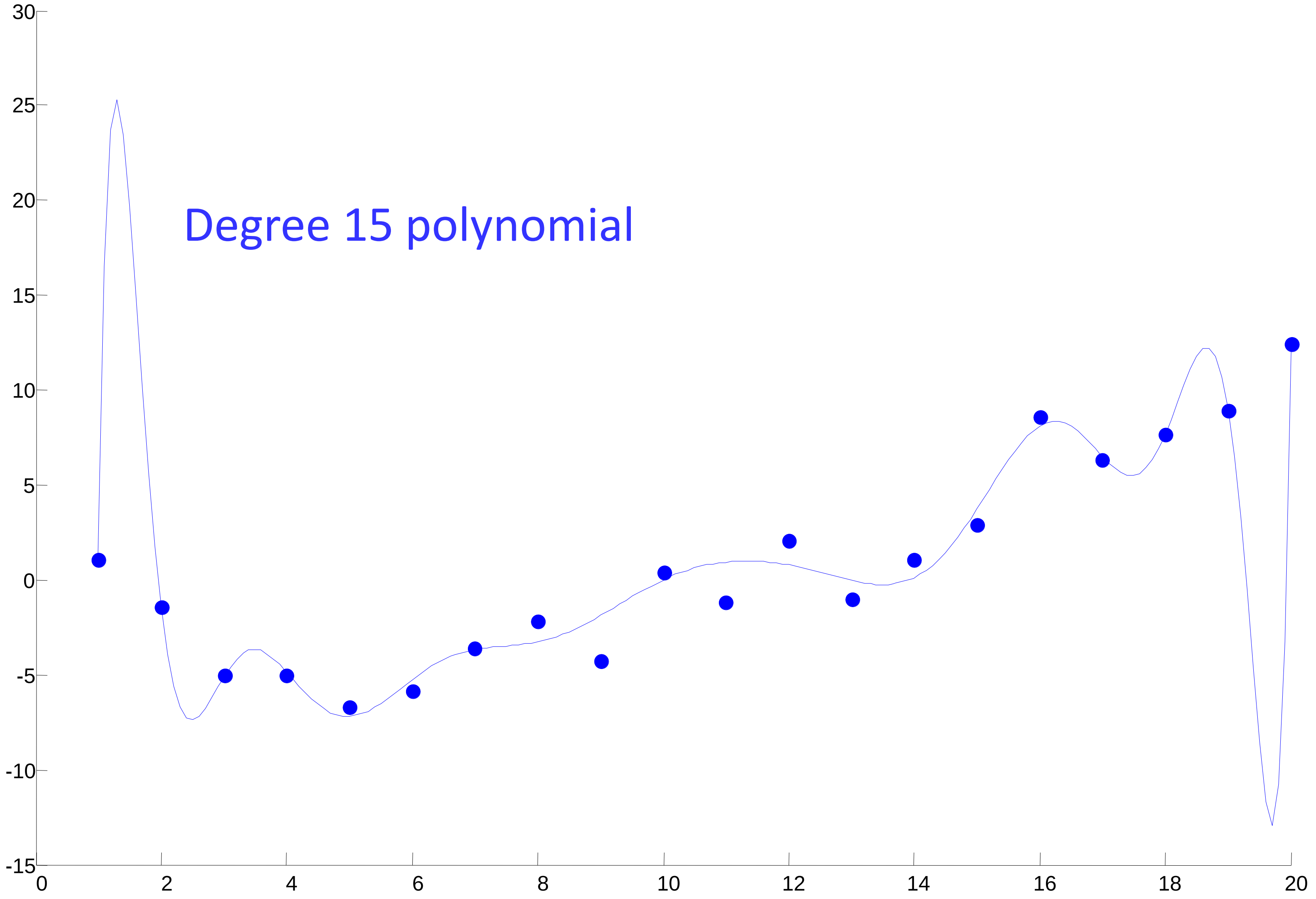
99 MILLION EMAIL ADDRESSES
FOR ONLY $99

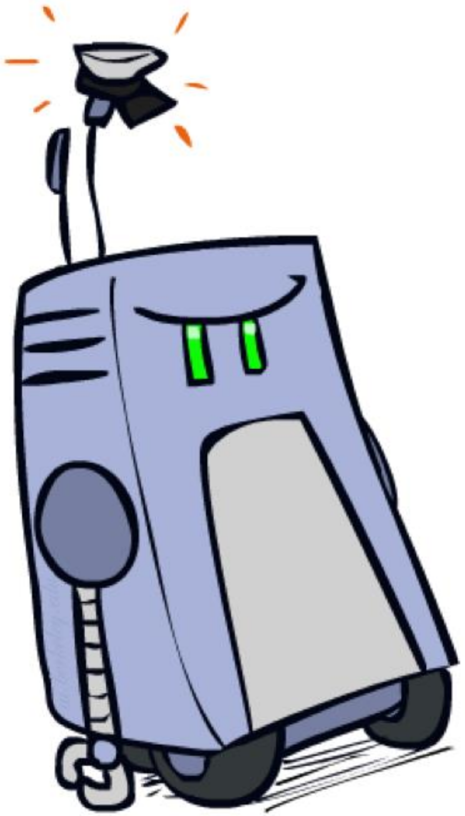
Ok, Iknow this is blatantly OT but I'm beginning to go insane. Had an old Dell Dimension XPS sitting in the corner and decided to put it to use, I know it was working pre being stuck in the corner, but when I plugged it in, hit the power nothing happened.

Overfitting

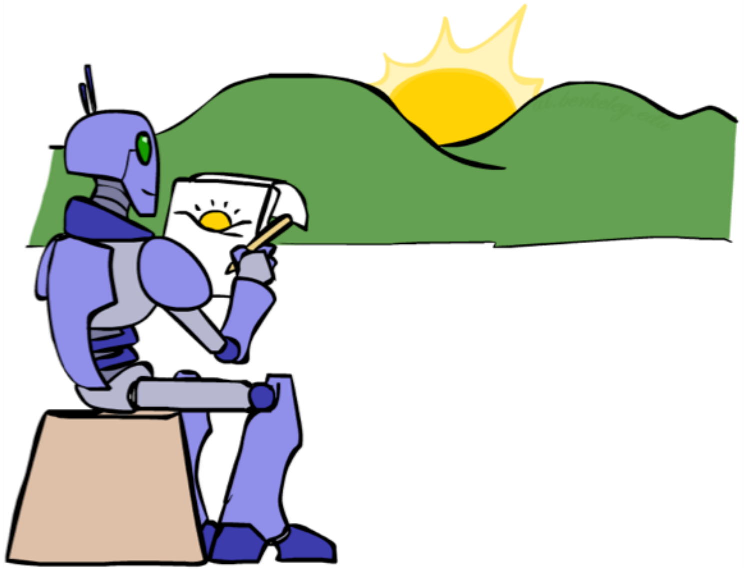


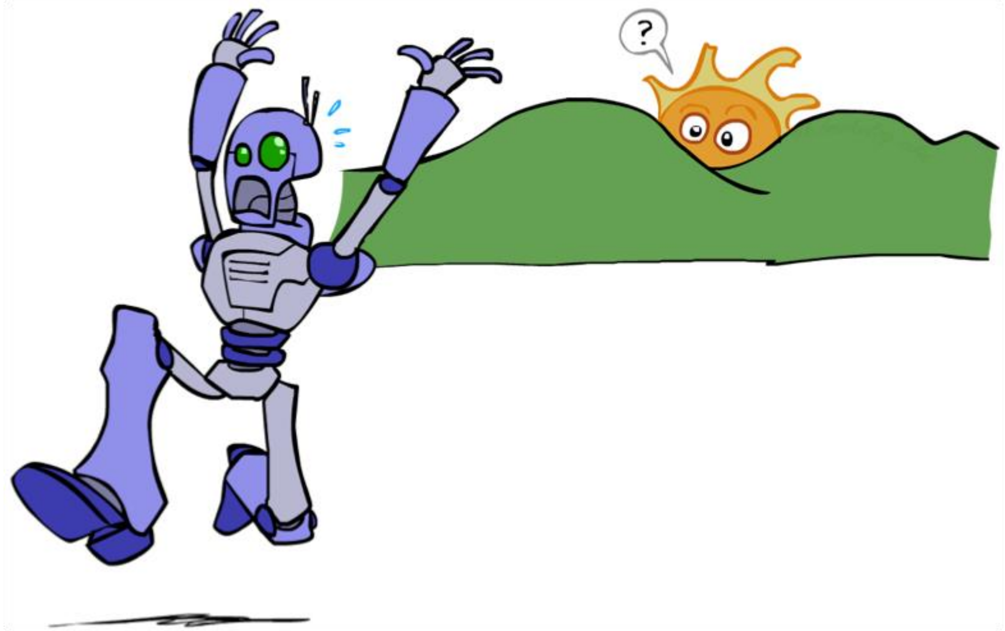
Overfitting





Unseen Events





Generalization and Overfitting

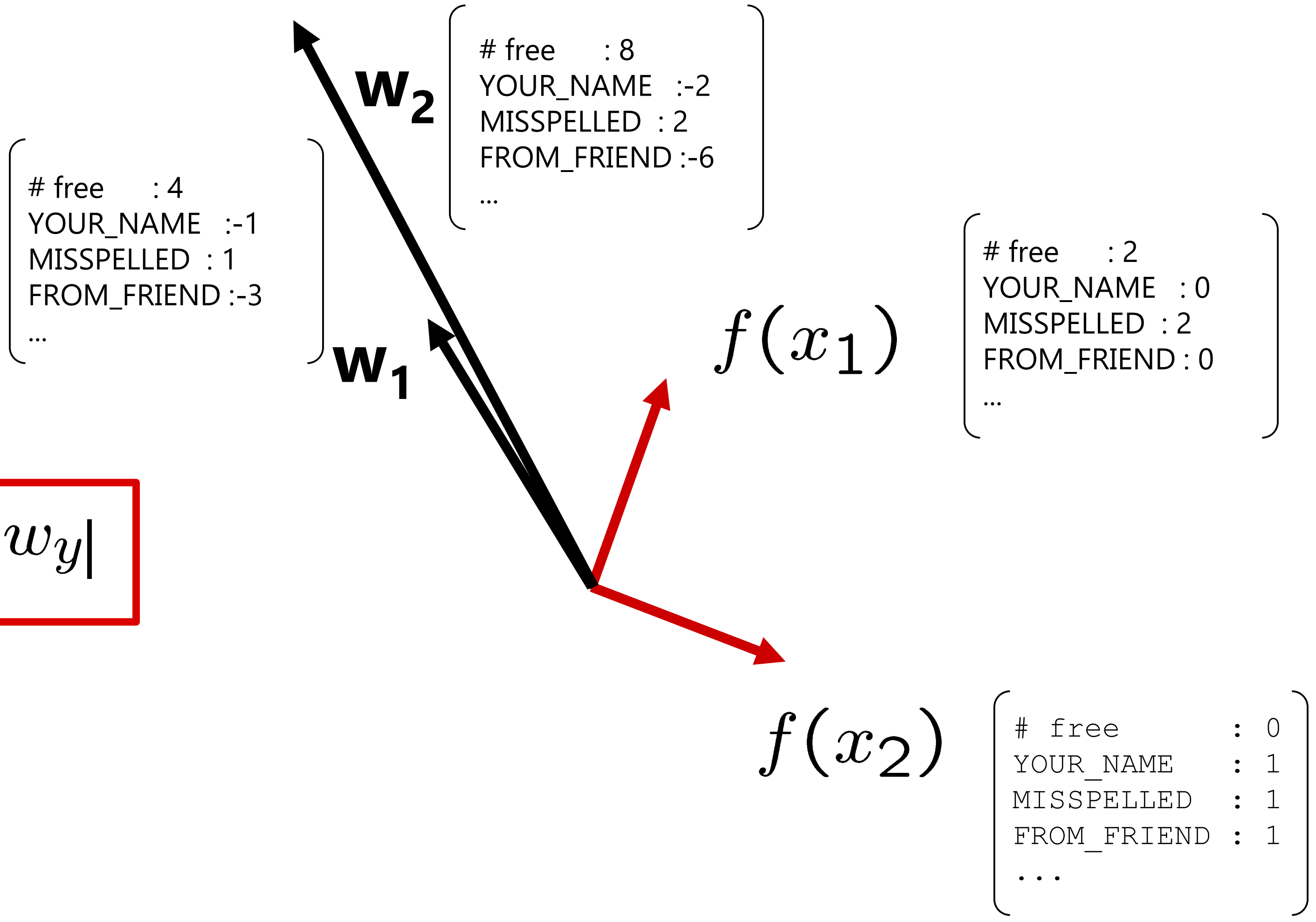
- Relative frequency parameters will overfit the training data!
  - Just because we never saw a non-spam email with an address during training doesn't mean we won't see it at test time
  - Unlikely that every occurrence of "minute" is 100% spam
  - Unlikely that every occurrence of "seriously" is 100% 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

Regularization

- Limit the number of features
- Limit the norm of the vector **w**
  - If $\mathbf{w_1}$ and $\mathbf{w_2}$ are equally good, and $|\mathbf{w_1}|>|\mathbf{w_2}|$, then $\mathbf{w_2}$ is likely to better generalize



$$y = \arg\max_y \boxed{w_y \cdot f(x) - |w_y|}$$

Practical Tip: Baselines

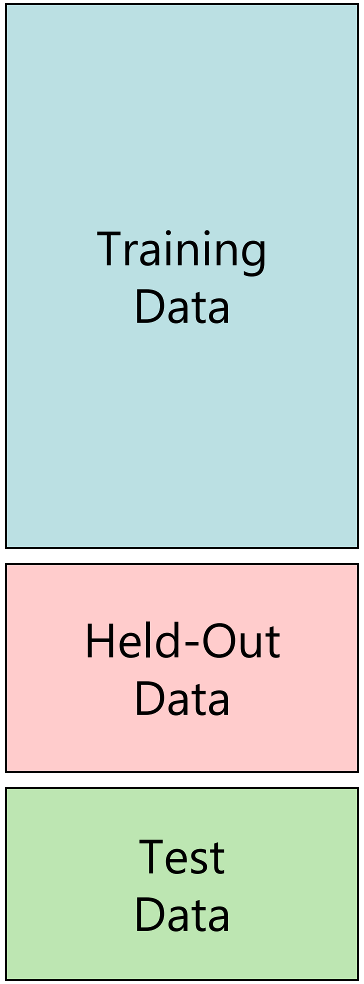
- First step: get a baseline
  - Baselines are very simple “straw man” procedures
  - Help determine how hard the task is
  - Help know what a “good” accuracy is

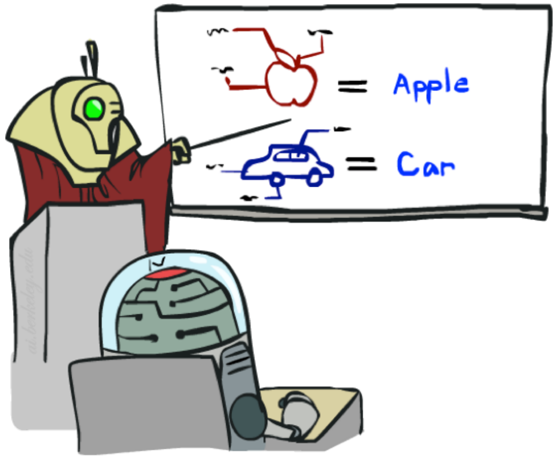
- Weak baseline: most frequent label classifier
  - Gives all test instances whatever label was most common in the training set
  - E.g. for spam filtering, might label everything as ham
  - Accuracy might be very high if the problem is skewed
  - E.g. calling everything “ham” gets 66%, so a classifier that gets 70% isn’t very good…

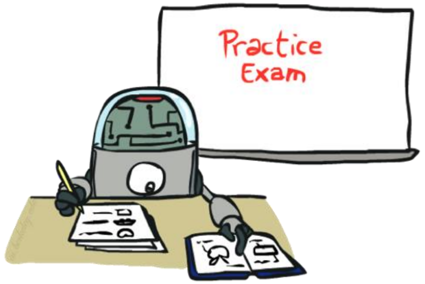
- For real research, usually use previous work as a (strong) baseline

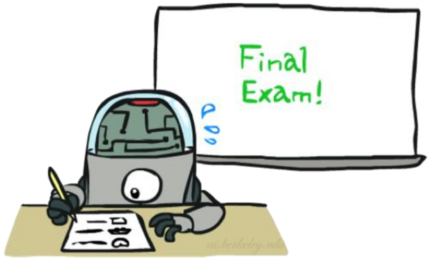
Important Concepts

- Data: labeled instances, e.g. emails marked spam/ham
  - Training set
  - Held out set
  - Test set
- Features: attribute-value pairs which characterize each x
- 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

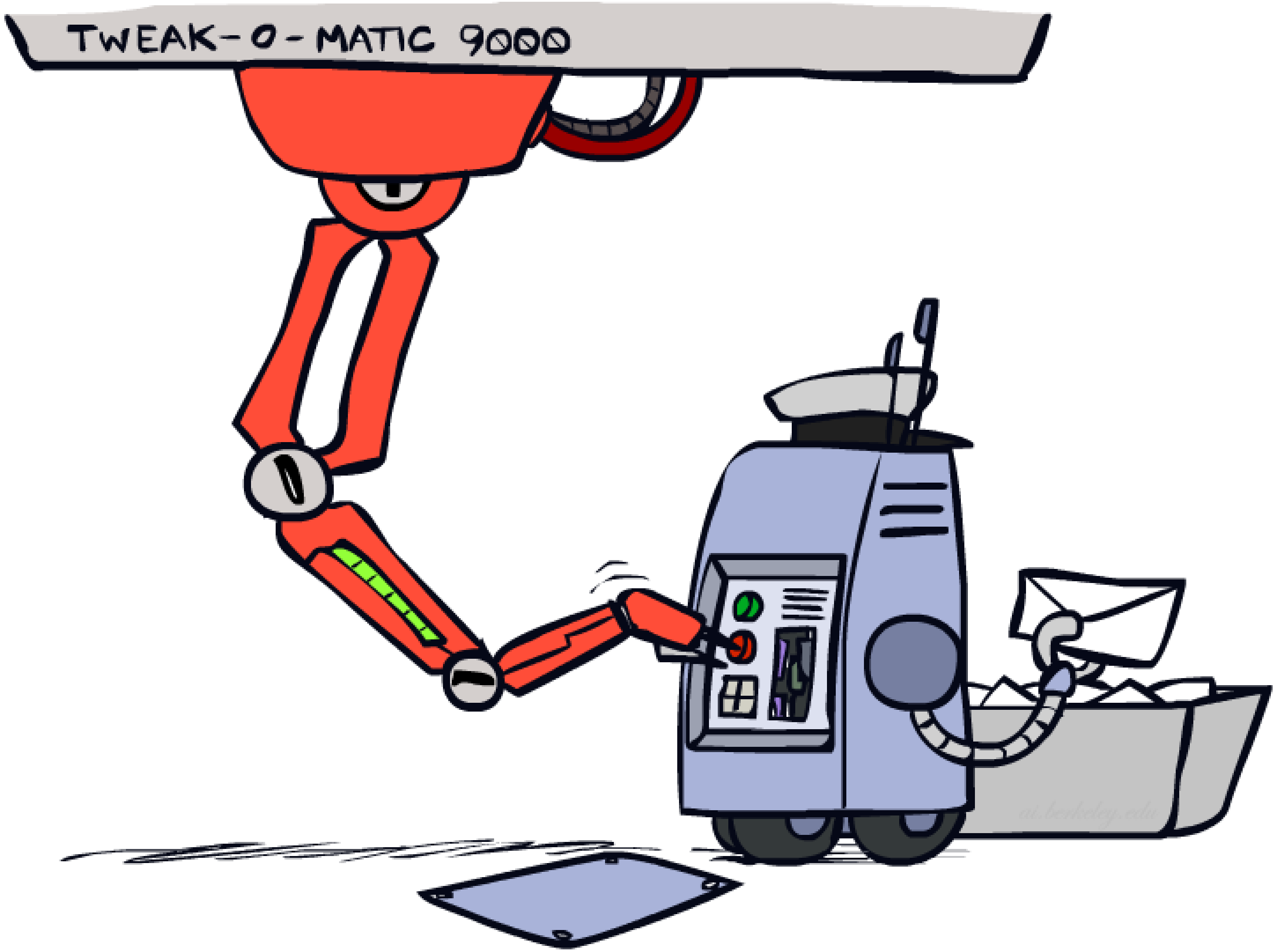








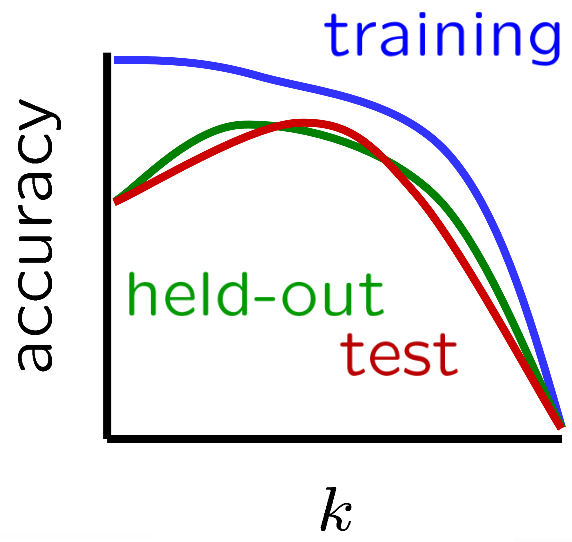
Tuning

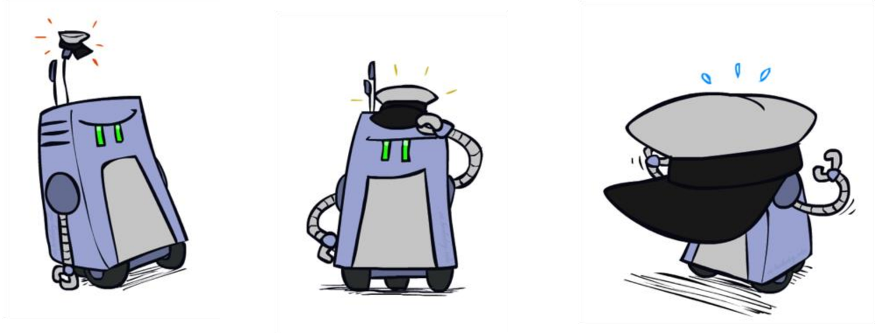


Tuning on Held-Out Data

- 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

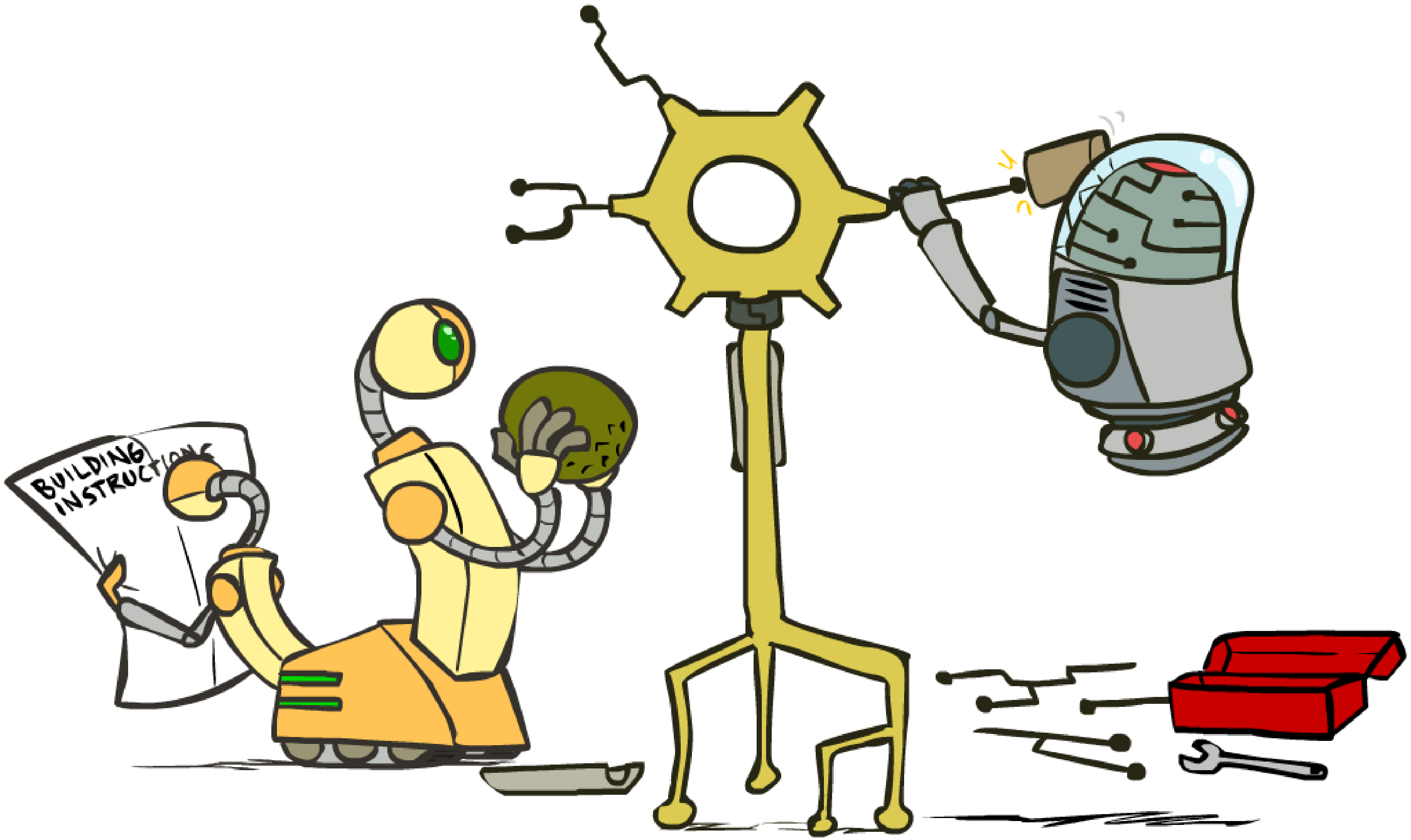




Practical Tip: Baselines

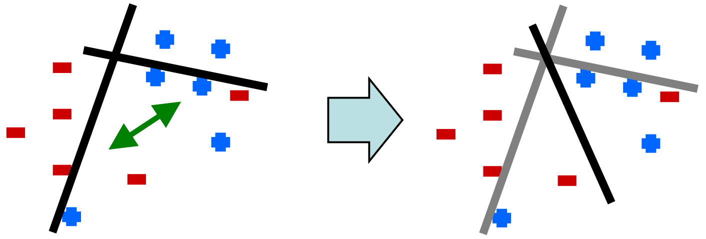
- First step: get a baseline
  - Baselines are very simple “straw man” procedures
  - Help determine how hard the task is
  - Help know what a “good” accuracy is
- Weak baseline: most frequent label classifier
  - Gives all test instances whatever label was most common in the training set
  - E.g. for 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

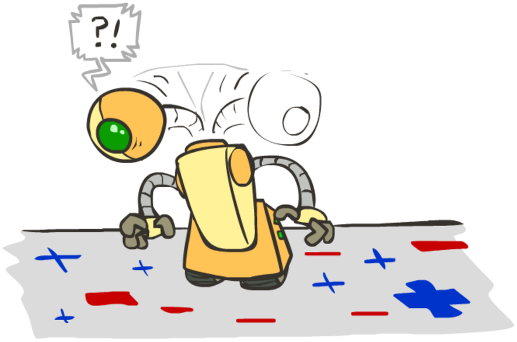
Improving the Perceptron



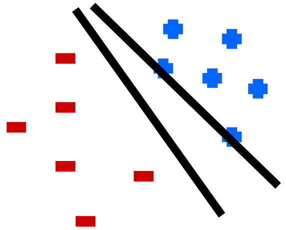
Problems with the Perceptron

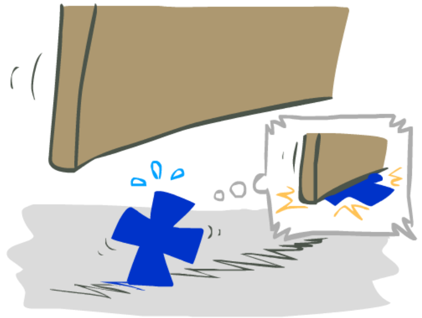
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



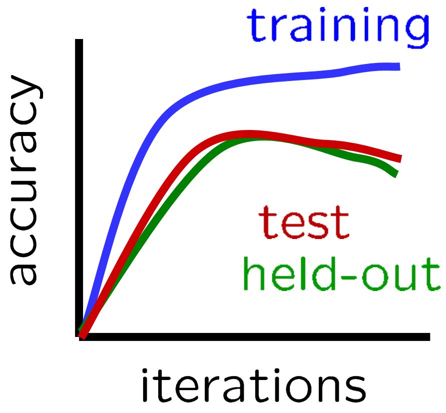


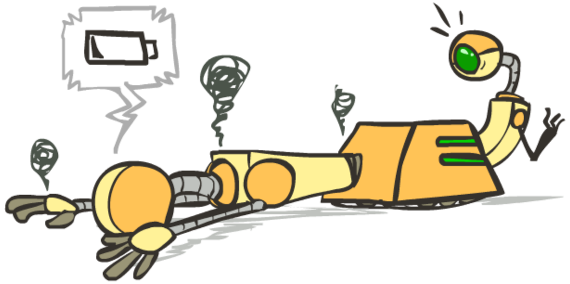
- Mediocre generalization: finds a "barely" separating solution





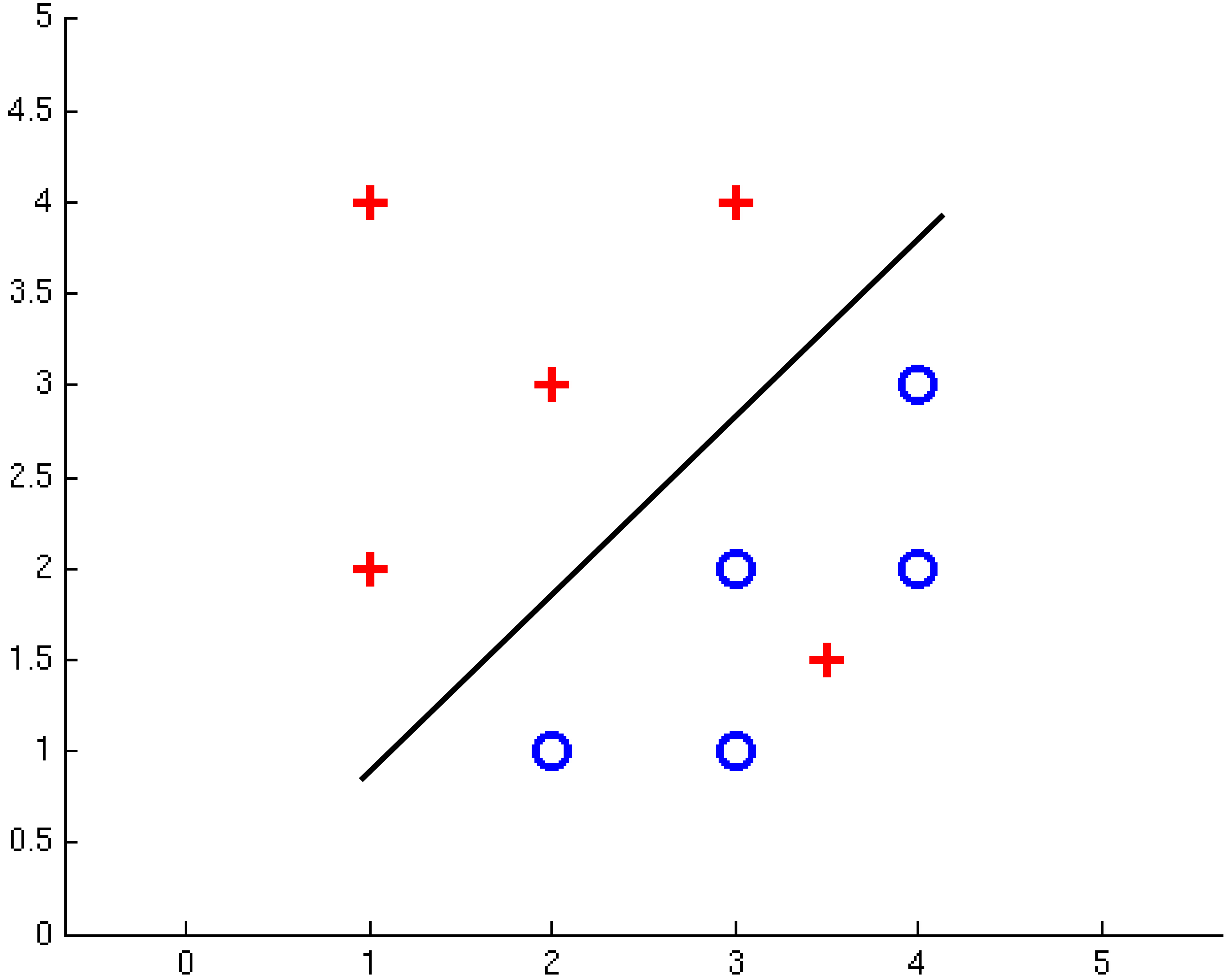
- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting



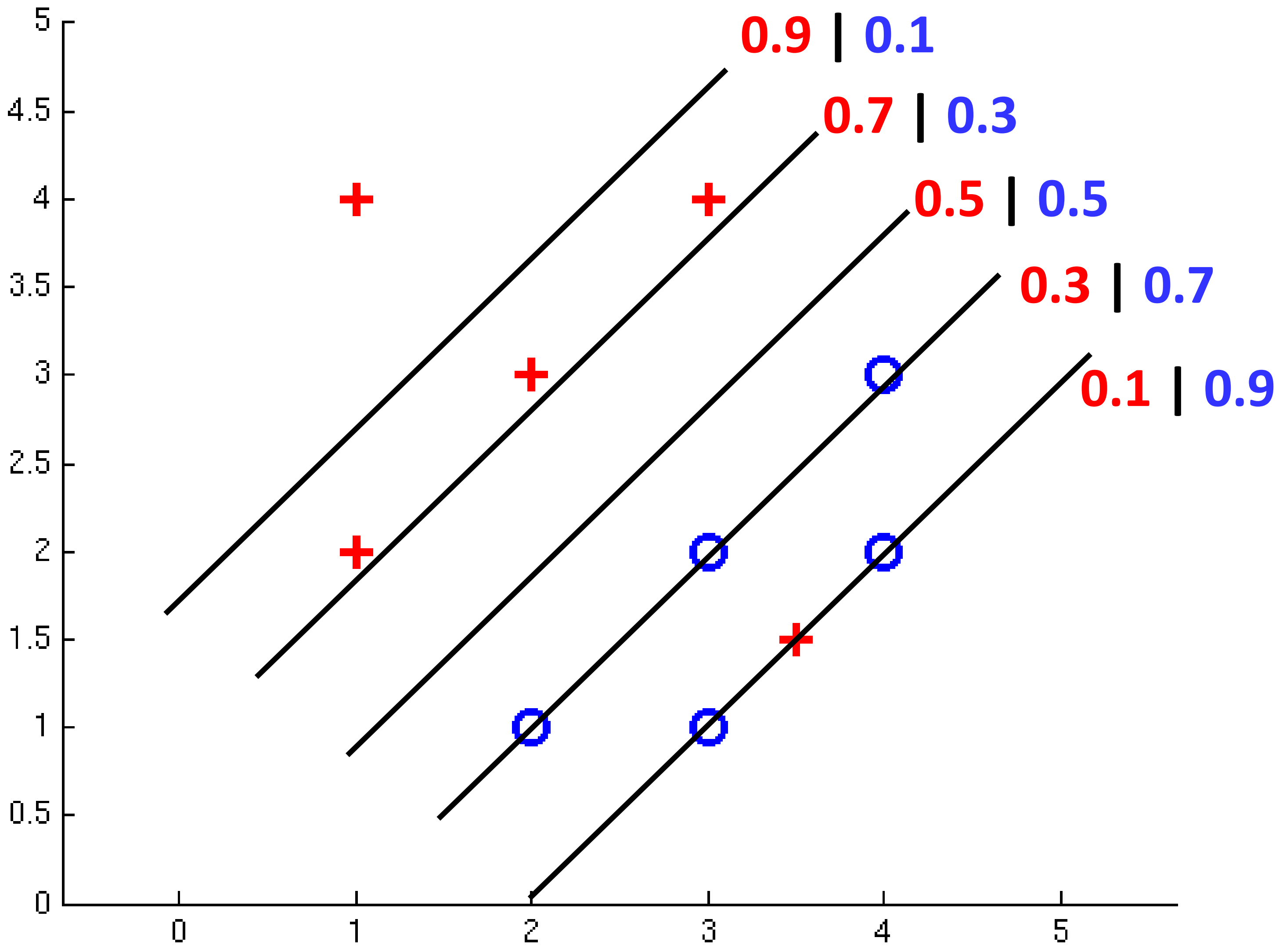


Non-Separable Case: Deterministic Decision

Even the best linear boundary makes at least one mistake



Non-Separable Case: Probabilistic Decision

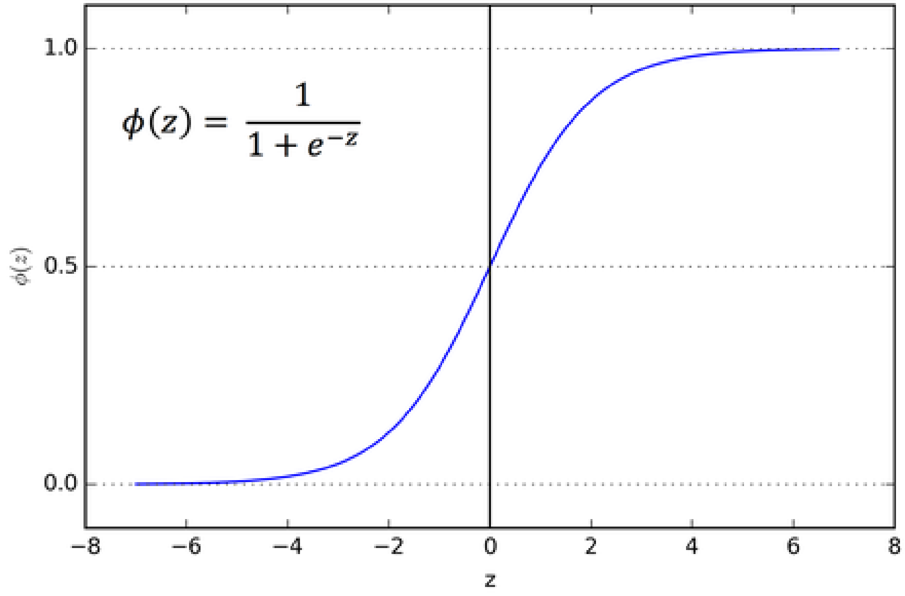


How to get probabilistic decisions?

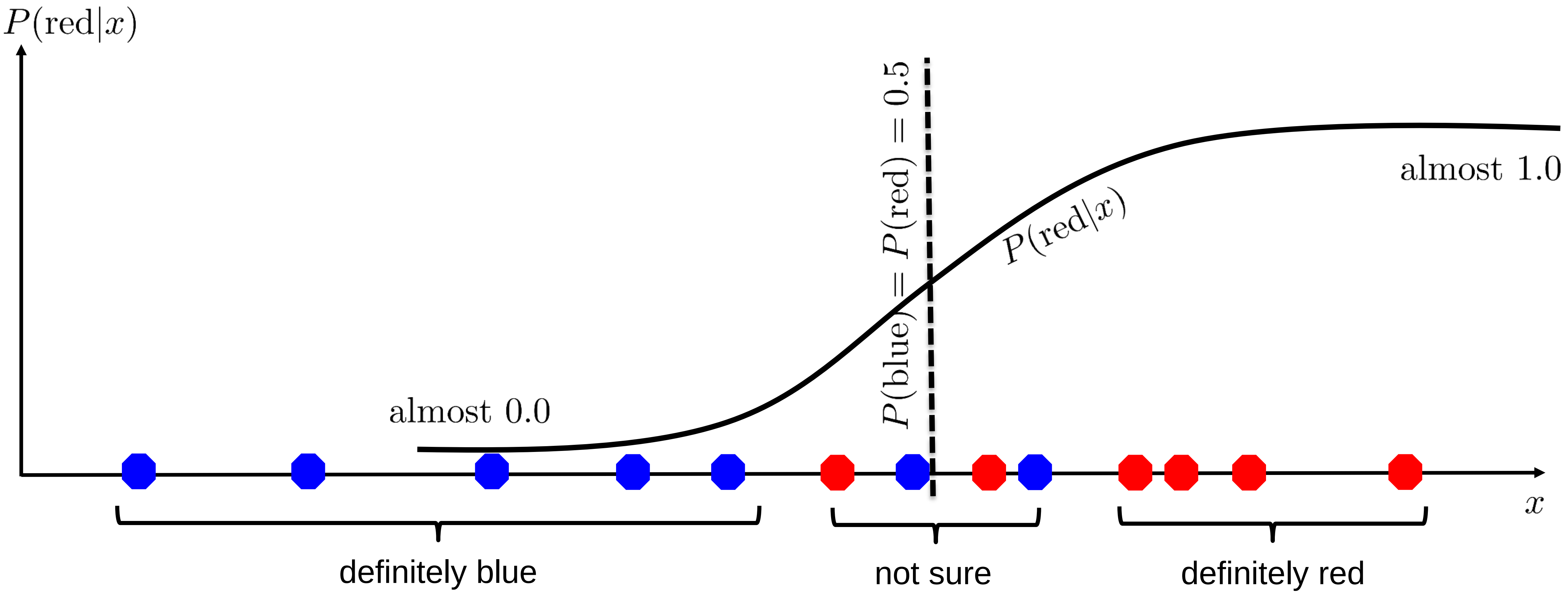
- Perceptron scoring: $z = w \cdot f(x)$
- If $z = w \cdot f(x)$ very positive → want probability going to 1
- If $z = w \cdot f(x)$ very negative → want probability going to 0

- Sigmoid function

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



A 1D Example

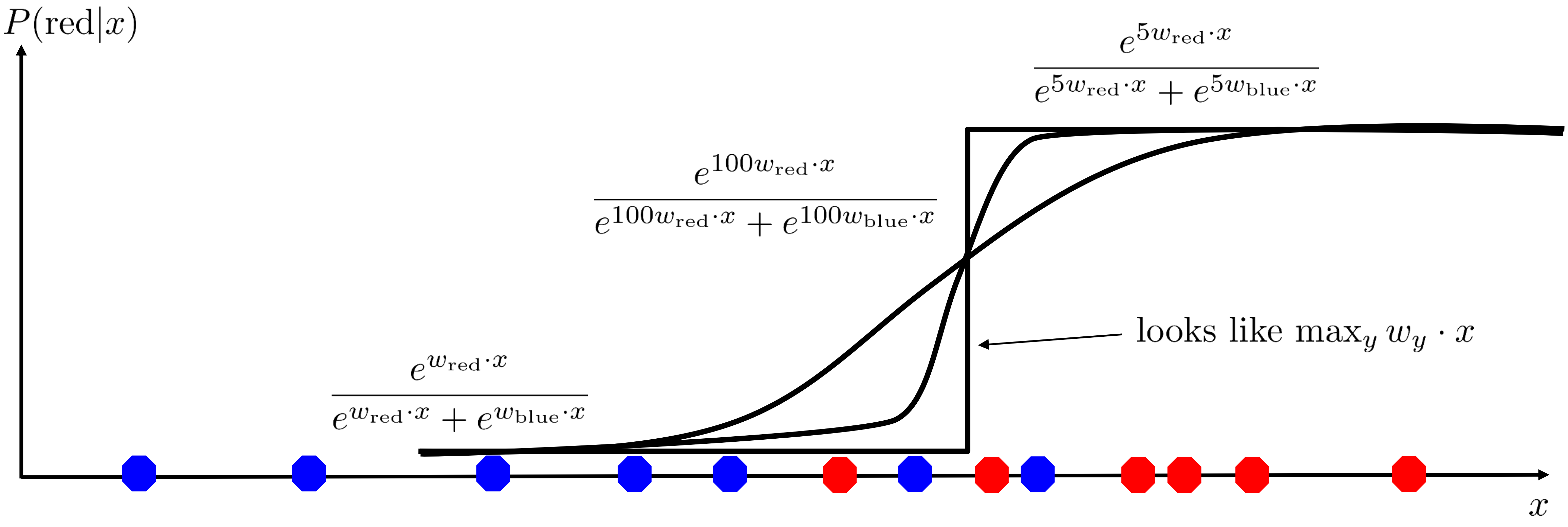


$$P(\mathrm{red}|x) = \frac{e^{w_{\mathrm{red}} \cdot x}}{e^{w_{\mathrm{red}} \cdot x} + e^{w_{\mathrm{blue}} \cdot x}}$$

probability increases exponentially as we move away from boundary

normalizer

The *Soft* Max



$$P(\text{red}|x) = \frac{e^{w_{\text{red}}\cdot x}}{e^{w_{\text{red}}\cdot x}+e^{w_{\text{blue}}\cdot x}}$$

Best w?

- Maximum likelihood estimation:

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

with:

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

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

= Logistic Regression

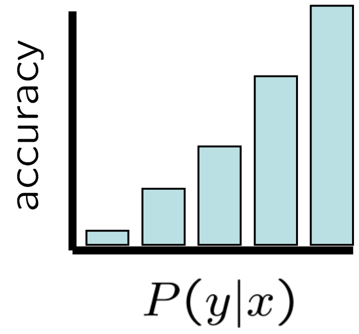
Confidences from a Classifier

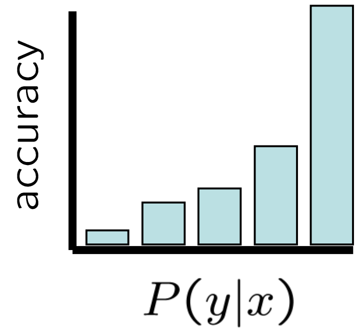
- The confidence of a probabilistic classifier:
  - Posterior over the top label

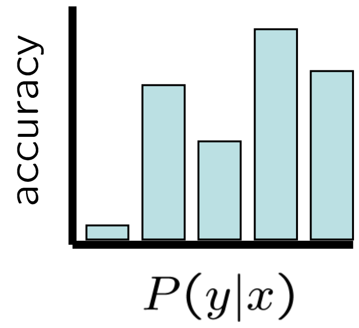
$$\text{confidence}(x) = \max_y P(y|x)$$

  - Represents how sure the classifier is of the classification
  - Any probabilistic model will have confidences
  - No guarantee confidence is correct

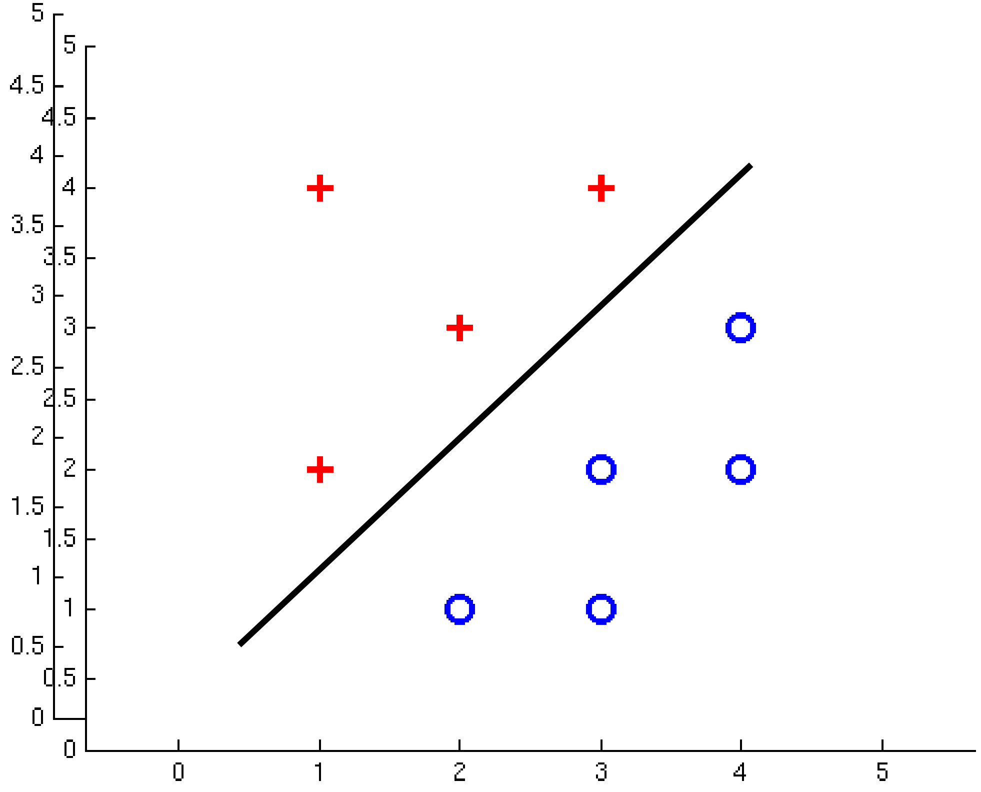
- Calibration
  - Weak calibration: higher confidences mean higher accuracy
  - Strong calibration: confidence predicts accuracy rate
  - What's the value of calibration?

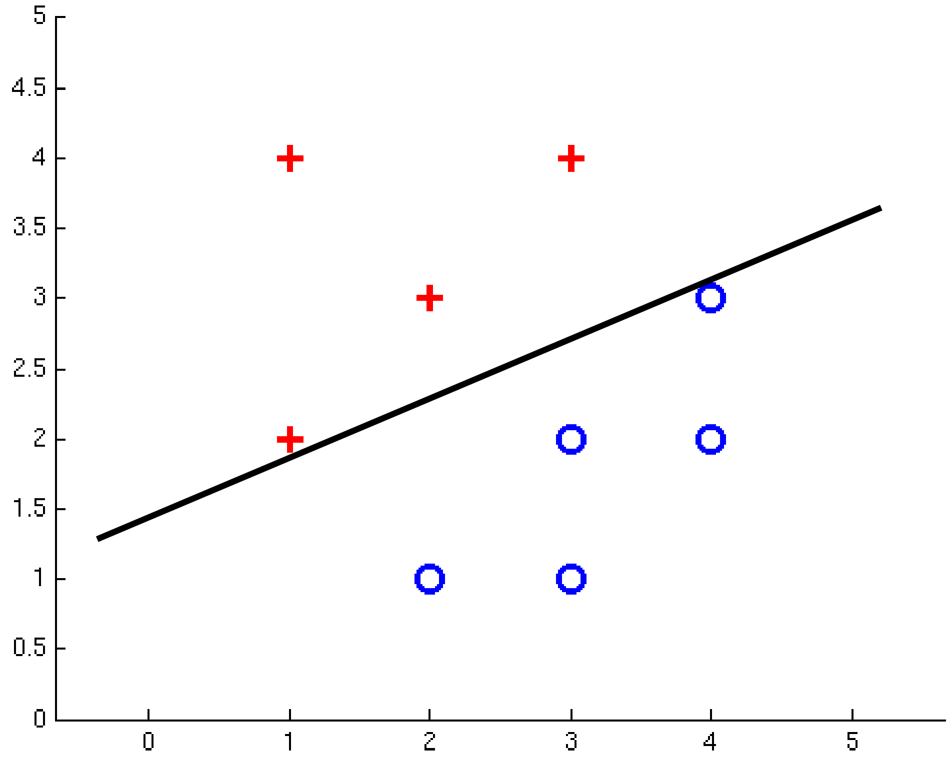




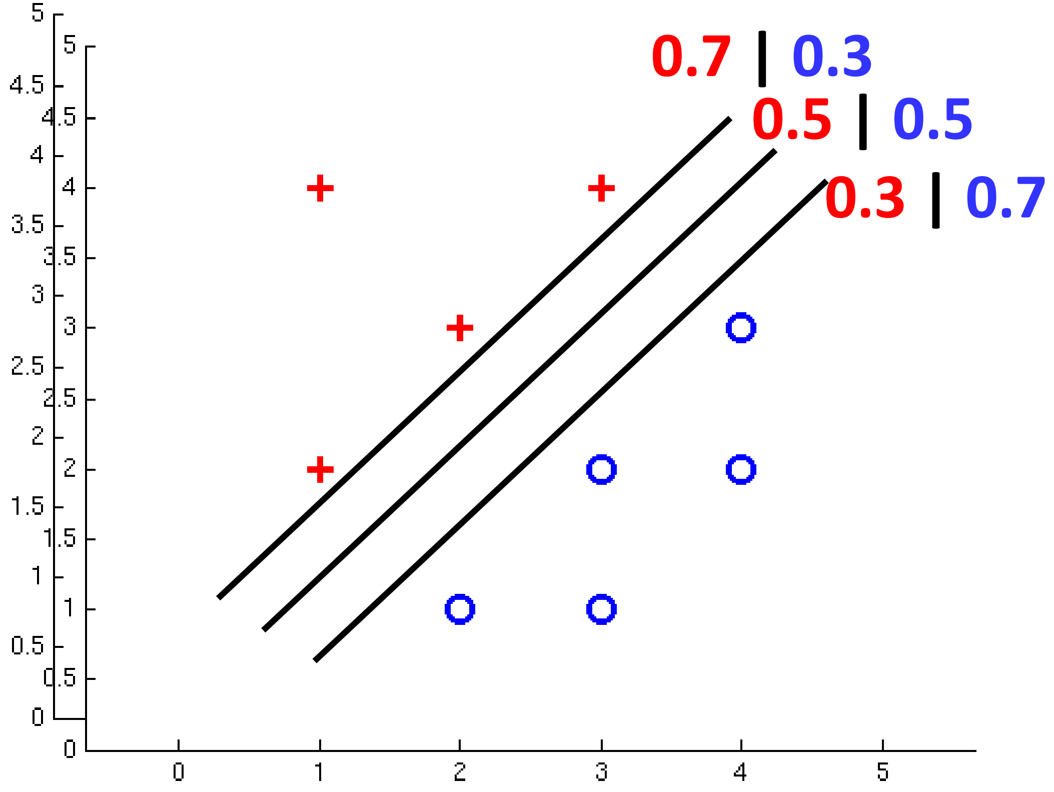


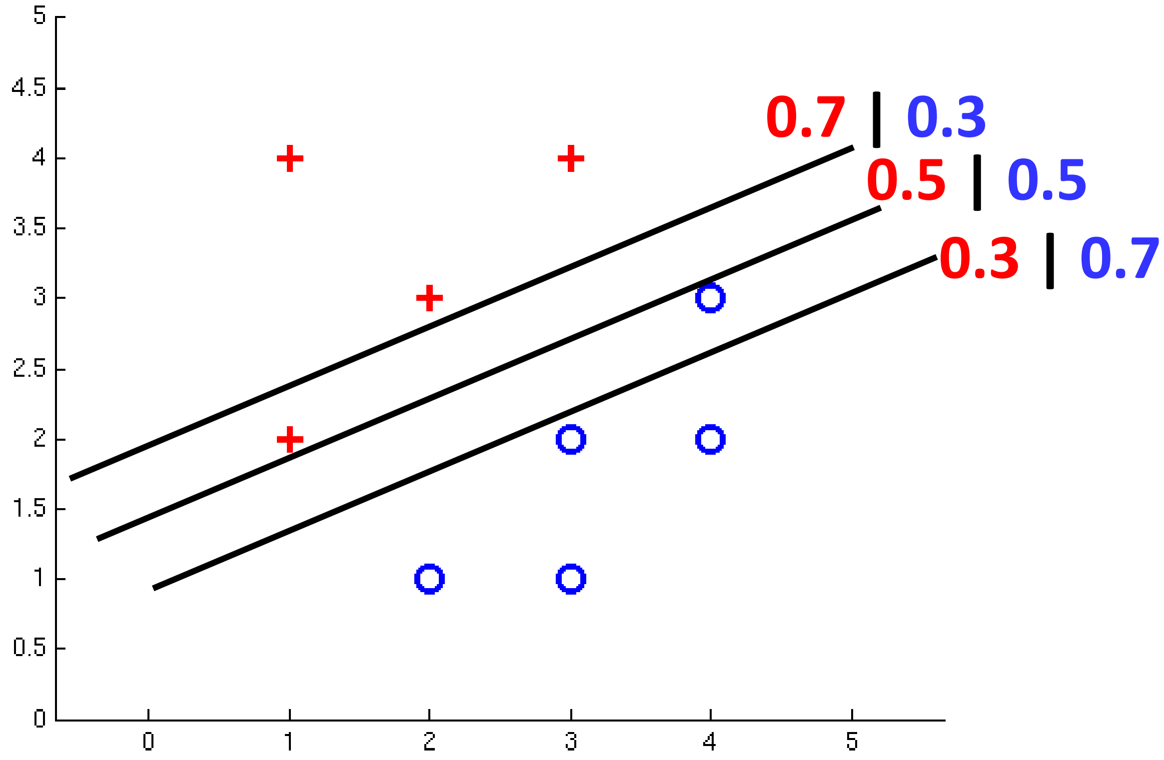
Separable Case: Deterministic Decision – Many Options





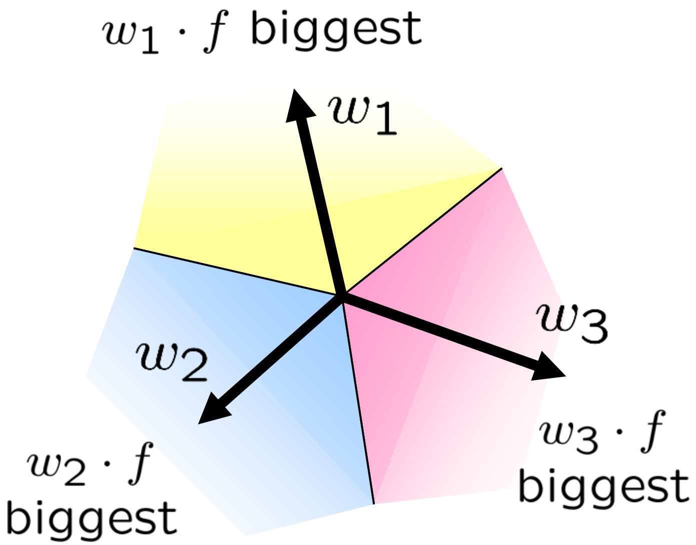
Separable Case: Probabilistic Decision – Clear Preference





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

Best w?

- Optimization
  - i.e., how do we solve:

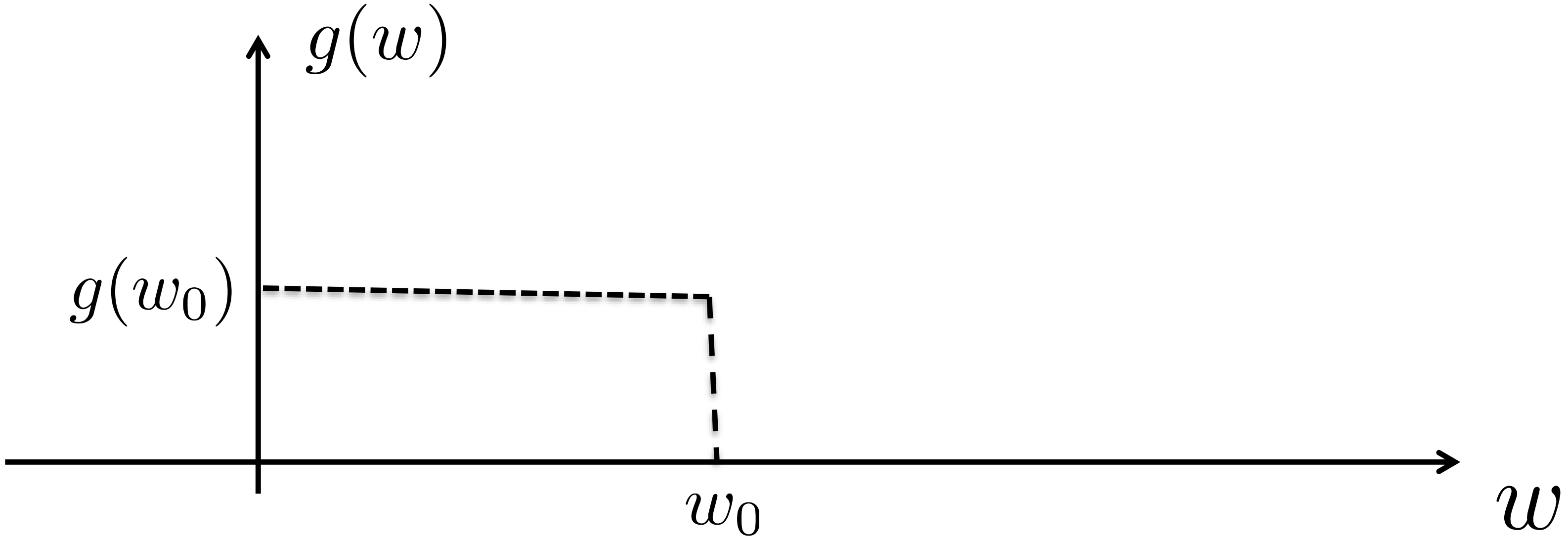
$$\max_{w} \quad ll(w) = \max_{w} \quad \sum_{i} \log P(y^{(i)}|x^{(i)}; w)$$

Hill Climbing

- Simple, general idea
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit

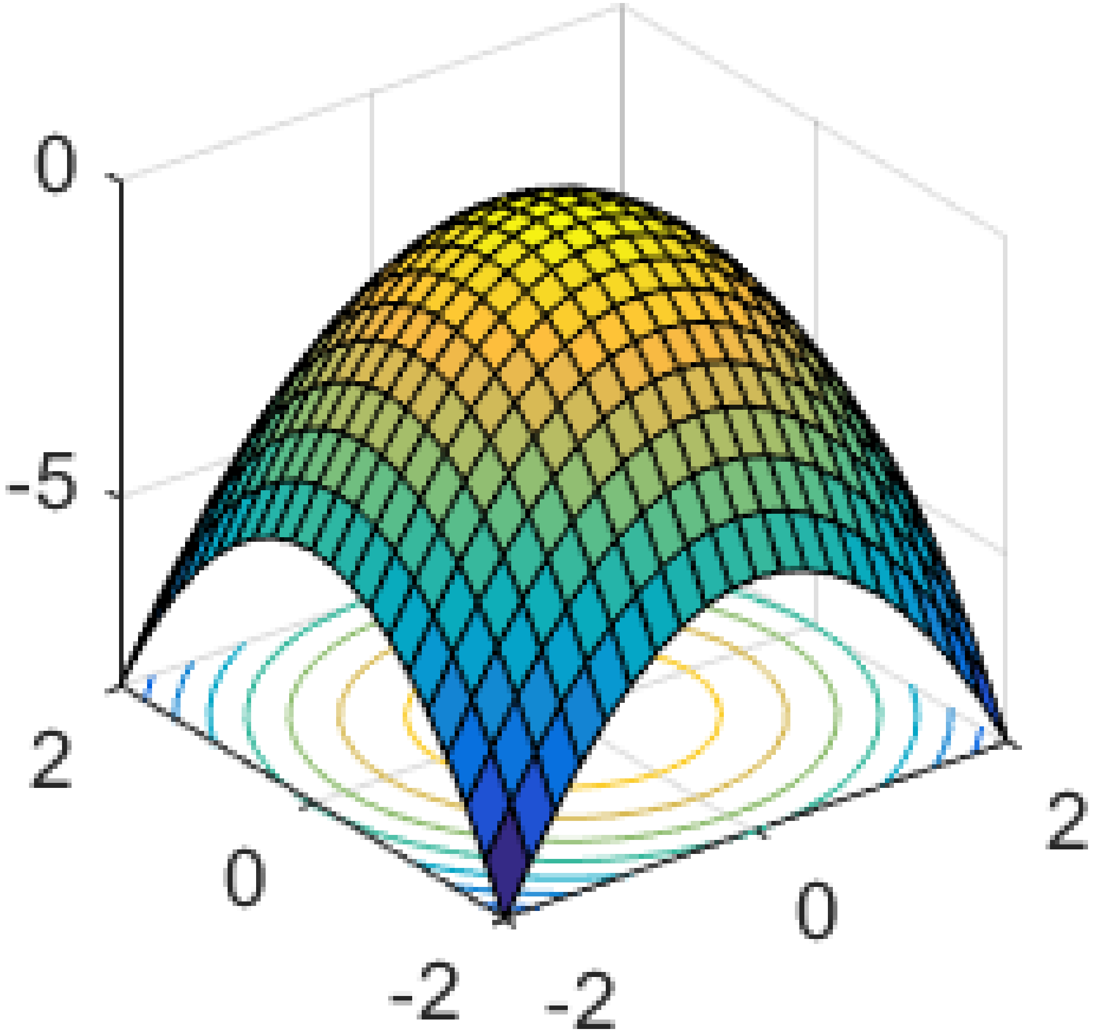
- What's particularly tricky when hill-climbing for multiclass logistic regression?
  - Optimization over a continuous space
    - Infinitely many neighbors!
    - How to do this efficiently?

1-D Optimization



- Could evaluate $g(w_0 + h)$ and $g(w_0 - h)$
  - Then step in best direction
- Or, evaluate derivative:
  - Tells which direction to step into

2-D Optimization



Source: offconvex.org

Gradient Ascent

- Perform update in uphill direction for each coordinate
- E.g., consider:
  - Updates: $g(w_1, w_2)$

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

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

- 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

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

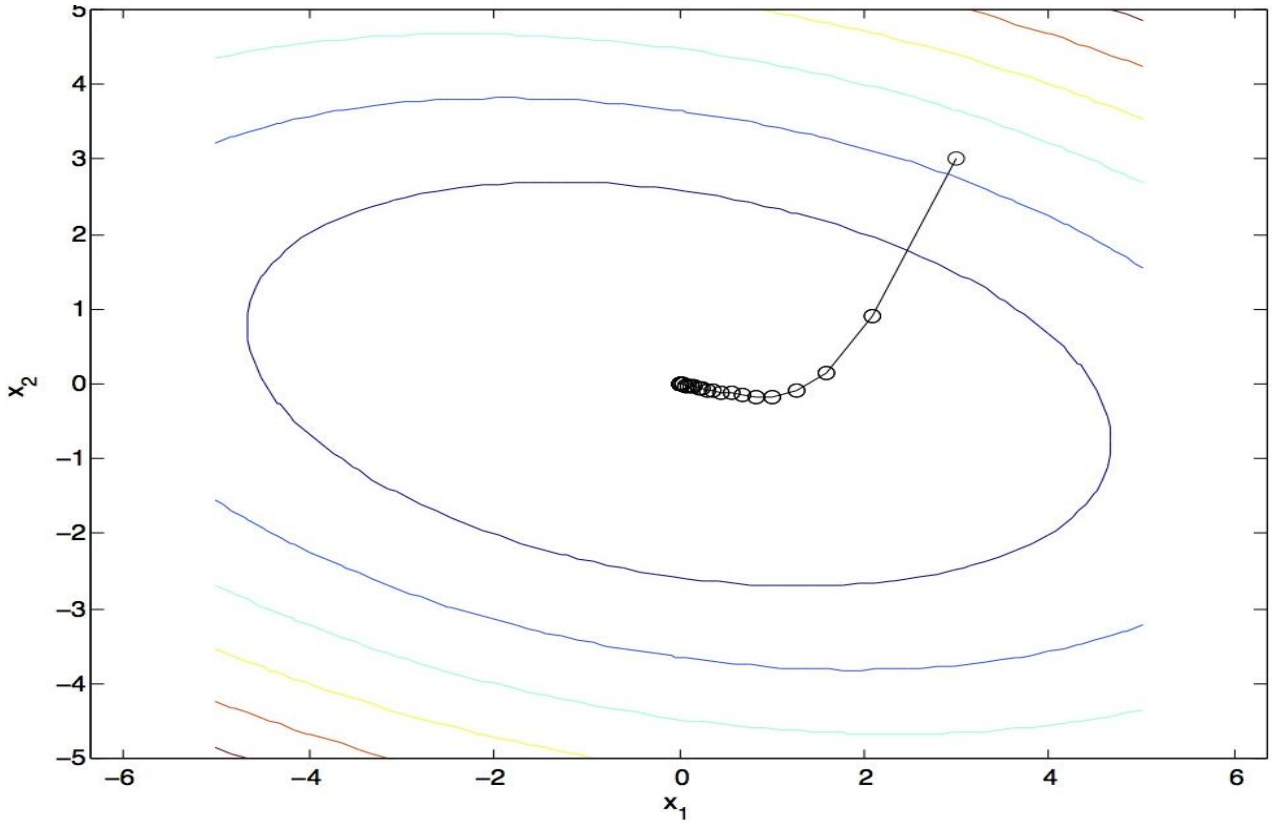


Figure source: Mathworks

Gradient in n dimensions

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \\ \cdots \\ \frac{\partial g}{\partial w_n} \end{bmatrix}$$

Optimization Procedure: Gradient Ascent

- `init` w
- `for iter = 1, 2, …`

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

- α : learning rate --- tweaking parameter that needs to be chosen carefully
- How? Try multiple choices
  - Crude rule of thumb: update changes w about 0.1 – 1 %

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, …

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

Stochastic Gradient Ascent on the Log Likelihood Objective

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

Observation: once gradient on one training example has been computed, might as well incorporate 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

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

Observation: gradient over 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

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

How about computing all the derivatives?

- We’ll talk about that in neural networks, which are a generalization of logistic regression