CSE 573: Artificial Intelligence
Autumn 2010

Lecture 14: Smoothing and the Perceptron

12/2/2010

Luke Zettlemoyer

Many slides over the course adapted from Dan Klein.

Announcements

- Syllabus revised
  - Machine learning focus
- We will do mini-project status reports during last class
  - I will email instructions this weekend

Outline

- Learning: Naive Bayes and Perceptron
  - (Recap) Naive Bayes models
  - Parameter Estimation
  - Smoothing
  - Perceptron (binary and multi-class)
  - MIRA
  - SVMs
  - Linear Ranking Models

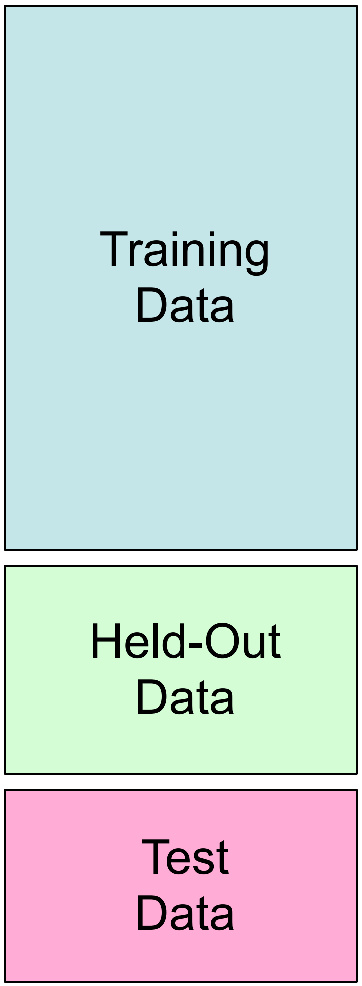
(Recap) Machine Learning

- Up until now: how to reason in a model and how to make optimal decisions

- Machine learning: how to acquire a model on the basis of data / experience
  - Learning parameters (e.g. probabilities)
  - Learning structure (e.g. BN graphs)
  - Learning hidden concepts (e.g. clustering)

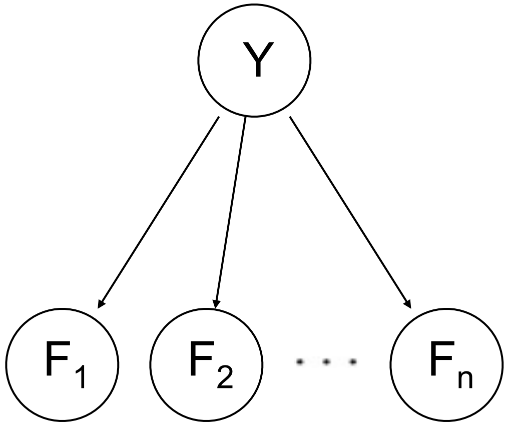
(Recap) Important Concepts

- 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)
  - Very important: never “peek” at the test set!
- Evaluation
  - Compute accuracy of test set
  - 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



General Naïve Bayes

- A general *naive Bayes* model:



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

- We only specify how each feature depends on the class
- Total number of parameters is *linear* in *n*
- Use probabilistic inference to compute most likely Y

$$y = \mathrm{argmax}_y \; P(y | f_1 \dots f_n)$$

(Recap) General Naïve Bayes

- What do we need in order to use naïve Bayes?
  - Inference (you know this part)
    - Start with a bunch of conditionals, 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 θ
    - Up until now, we assumed these appeared by magic, but…
    - …they typically come from training data: we'll look at this now

Parameter Estimation

- Estimating distribution of random variables like X or X | Y

- *Elicitation:* ask a human!
  - Usually need domain experts, and sophisticated ways of eliciting probabilities (e.g. betting games)
  - Trouble calibrating

- *Empirically:* use training data
  - For each outcome x, look at the *empirical rate* of that value:

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

$$P_{\text{ML}}(\text{r}) = 1/3$$

  - This is the estimate that maximizes the *likelihood of the data*

$$L(x, \theta) = \prod_i P_\theta(x_i)$$

Naïve Bayes for Digits

- Simple version:
  - One feature F_{ij} for each grid position <i,j>
  - Possible 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?

Naïve Bayes for Text

- Bag-of-Words Naïve Bayes:
  - Predict unknown class label (spam vs. ham)
  - Assume evidence features (e.g. the words) are independent
  - Warning: subtly different assumptions than before!

- Generative model

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

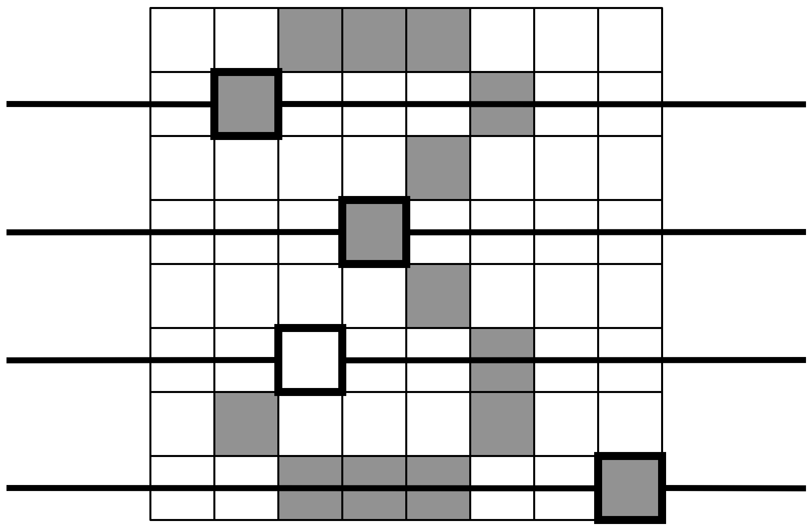
$$P(C, W_1 \dots W_n) = P(C) \prod_i P(W_i|C)$$

- 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|C)
    - Why make this assumption?

Example: Overfitting

$P(\text{features}, C = 2)$ $\quad$ $P(\text{features}, C = 3)$

$P(C = 2) = 0.1$ $\quad$ $P(C = 3) = 0.1$



$P(\text{on}|C = 2) = 0.8$ $\quad$ $P(\text{on}|C = 3) = 0.8$

$P(\text{on}|C = 2) = 0.1$ $\quad$ $P(\text{on}|C = 3) = 0.9$

$P(\text{off}|C = 2) = 0.1$ $\quad$ $P(\text{off}|C = 3) = 0.7$

$P(\text{on}|C = 2) = 0.01$ $\quad$ $P(\text{on}|C = 3) = 0.0$

2 wins!!

Example: Overfitting

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

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

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

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

What went wrong here?

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

Estimation: Smoothing

- Problems with maximum likelihood estimates:
  - If I flip a coin once, and it's heads, what's the estimate for P (heads)?
  - What if I flip 10 times with 8 heads?
  - What if I flip 10M times with 8M heads?

- Basic idea:
  - We have some prior expectation about parameters (here, the probability of heads)
  - Given little evidence, we should skew towards our prior
  - Given a lot of evidence, we should listen to the data

Estimation: Smoothing

- Relative frequencies are the maximum likelihood estimates

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

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

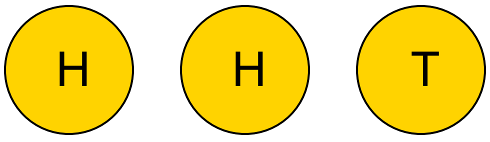
- In Bayesian statistics, we think of the parameters as just another random variable, with its own distribution

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

$\Rightarrow$????

Estimation: Laplace Smoothing

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



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

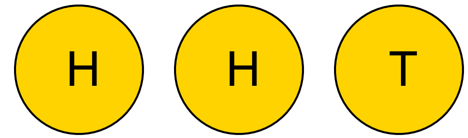
$$= \frac{c(x)+1}{N+|X|}$$

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

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

  - Can derive this as a MAP estimate with *Dirichlet priors* (Bayesian justfication)

Estimation: 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

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

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

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

- Laplace for conditionals:
  - Smooth each condition independently:

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

Estimation: Linear Interpolation

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

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

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

  - What if α is 0? 1? How do we set α?

Real NB: Smoothing

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

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

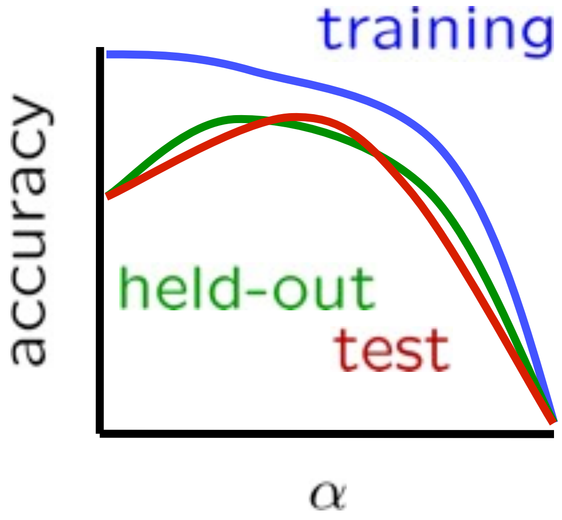
```
helvetica  :  11.4
seems      :  10.8
group      :  10.2
ago        :   8.4
areas      :   8.3
...
```

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

Do these make more sense?

Tuning on Held-Out Data

- Now we've got two kinds of unknowns
  - Parameters: the probabilities P(Y|X), P(Y)
  - Hyperparameters, like the amount of smoothing to do: k, α
- Where to learn?
  - Learn parameters from training data
  - Must 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



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

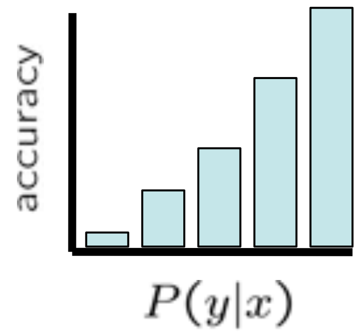
Confidences from a Classifier

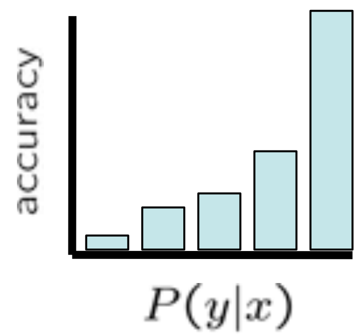
- The confidence of a probabilistic classifier:
  - Posterior over the top label

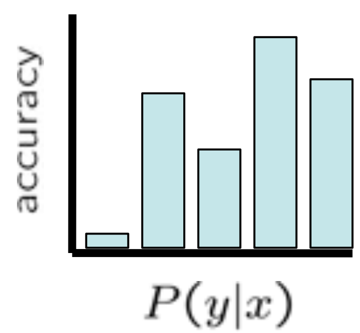
$$\text{confidence}(x) = \max_y P(y|x)$$

  - 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?

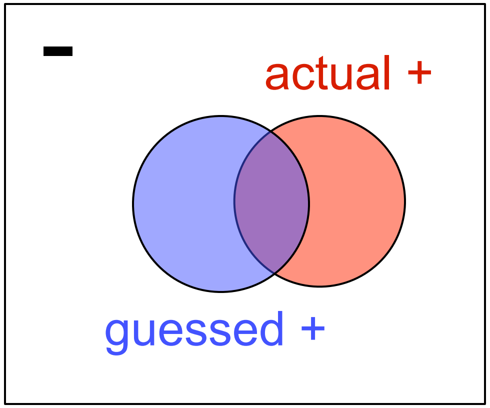






Precision vs. Recall

- 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

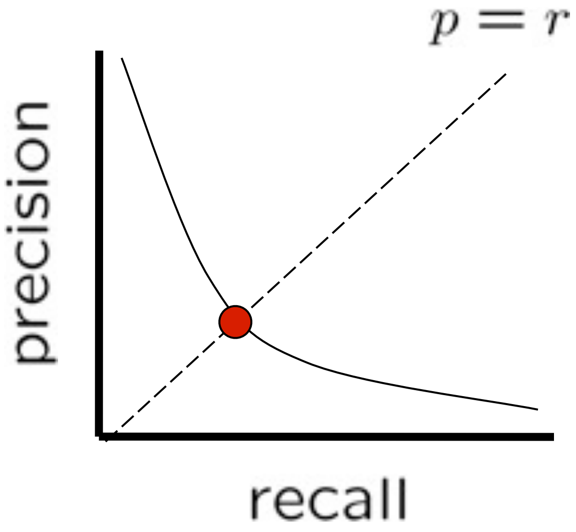


- Precision: fraction of guessed positives which were actually positive
- Recall: fraction of actual positives which were guessed as positive
- Say we detect 5 spam emails, of which 2 were actually spam, and we missed one
  - Precision: 2 correct / 5 guessed = 0.4
  - Recall: 2 correct / 3 true = 0.67
- Which is more important in spam filtering?

Precision vs. Recall

- Precision/recall tradeoff
  - Often, you can trade off precision and recall
  - Only works well with calibrated classifiers
- To summarize the tradeoff:
  - Break-even point: precision value when p = r
  - F-measure: harmonic mean of p and r:

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



Errors, and What to Do

- Examples of errors

```
Dear GlobalSCAPE Customer,

GlobalSCAPE has partnered with ScanSoft to offer you the
latest version of OmniPage Pro, for just $99.99* - the
regular list price is $499! The most common question we've
received about this offer is - Is this genuine? We would like
to assure you that this offer is authorized by ScanSoft, is
genuine and valid. You can get the . . .
```

```
. . . To receive your $30 Amazon.com promotional certificate,
click through to

  http://www.amazon.com/apparel

and see the prominent link for the $30 offer. All details are
there. We hope you enjoyed receiving this message. However,
if you'd rather not receive future e-mails announcing new
store launches, please click . . .
```

What to Do About Errors?

- Need more features– words aren't enough!
  - Have you emailed the sender before?
  - Have 1K other people just gotten the same email?
  - Is the sending information consistent?
  - Is the email in ALL CAPS?
  - Do inline URLs point where they say they point?
  - Does the email address you by (your) name?

- Can add these information sources as new variables in the NB model

- Next class we'll talk about classifiers which let you easily add arbitrary features more easily

Summary

- Bayes rule lets us do diagnostic queries with causal probabilities
- The naïve Bayes assumption takes all features to be independent given the class label
- We can build classifiers out of a naïve Bayes model using training data
- Smoothing estimates is important in real systems
- Classifier confidences are useful, when you can get them

Feature Extractors

- Features: anything you can compute about the input
- A **feature extractor** maps inputs to **feature vectors**

```
Dear Sir.

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

```
W=dear      :   1
W=sir       :   1
W=this      :   2
...
W=wish      :   0
...
MISSPELLED  :   2
YOUR_NAME   :   1
ALL_CAPS    :   0
NUM_URLS    :   0
...
```

- Many classifiers take feature vectors as inputs
- Feature vectors can be very sparse, use sparse encodings (i.e. only represent non-zero keys)

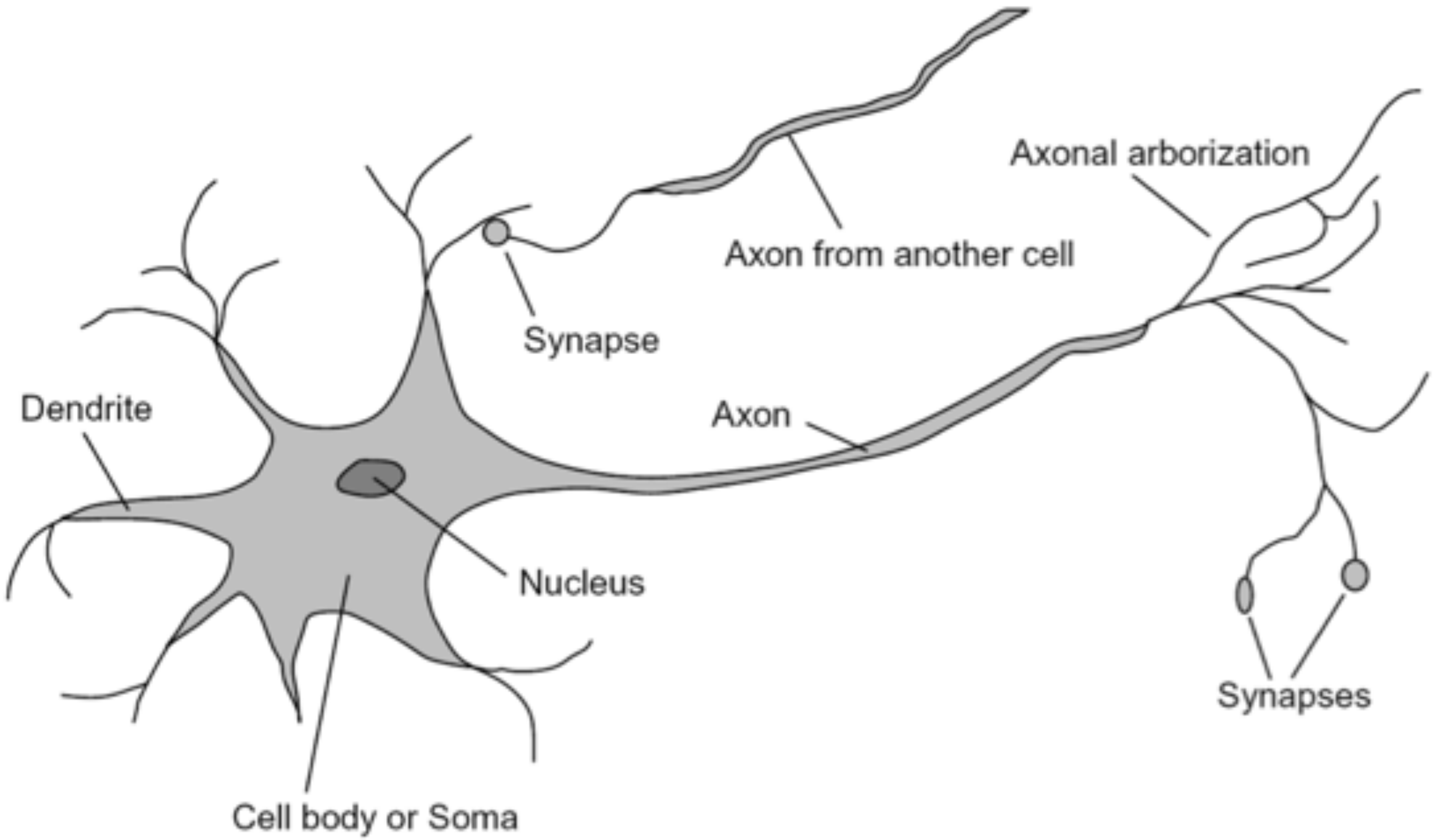
Generative vs. Discriminative

- Generative classifiers:
  - E.g. naïve Bayes
  - A joint probability model with evidence variables
  - Query model for causes given evidence

- Discriminative classifiers:
  - No generative model, no Bayes rule, often no probabilities at all!
  - Try to predict the label Y directly from X
  - Robust, accurate with varied features
  - Loosely: mistake driven rather than model driven

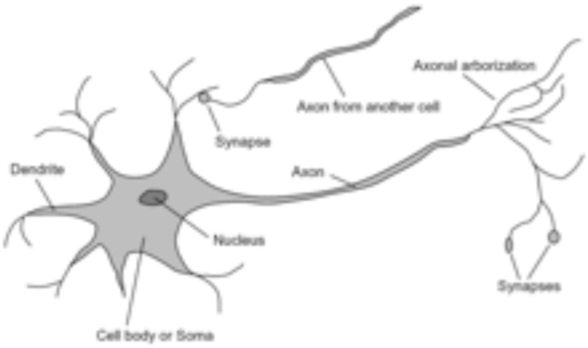
Some (Simplified) Biology

- Very loose inspiration: human neurons



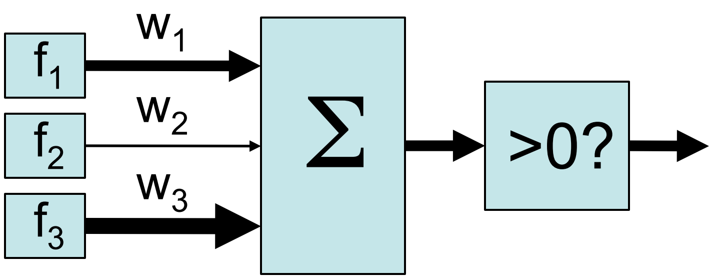
Linear Classifiers

- 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



Example: Spam

- Imagine 4 features (spam is "positive" class):
  - free (number of occurrences of "free")
  - money (occurrences of "money")
  - BIAS (intercept, always has value 1)

$$w \cdot f(x)$$

$\uparrow$

$$\sum_i w_i \cdot f_i(x)$$

x: "free money"

$f(x)$:

```
BIAS  :  1
free  :  1
money :  1
...
```

w:

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

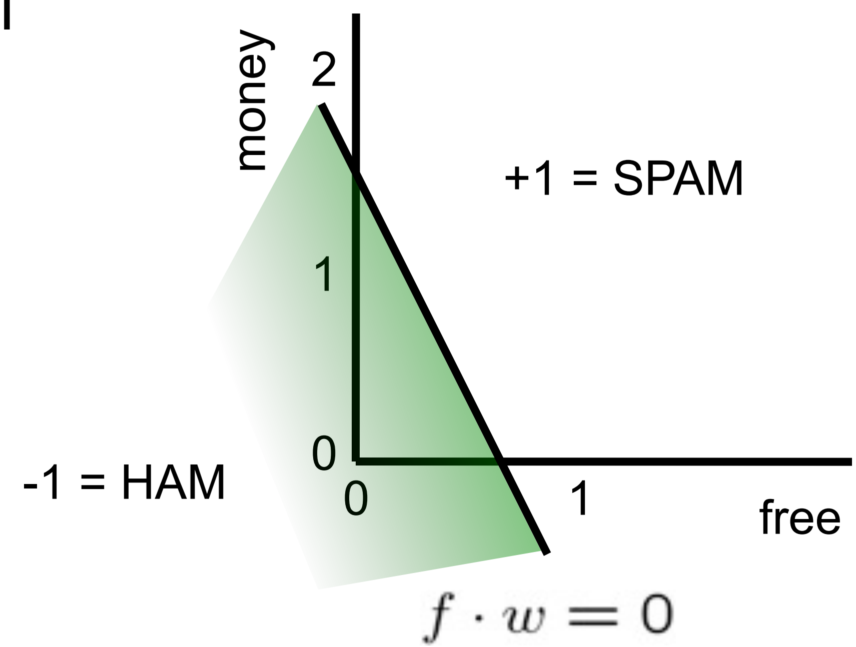
$$(1)(-3) \; + \\ (1)(4) \; + \\ (1)(2) \; + \\ \ldots \\ = 3$$

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
...
```



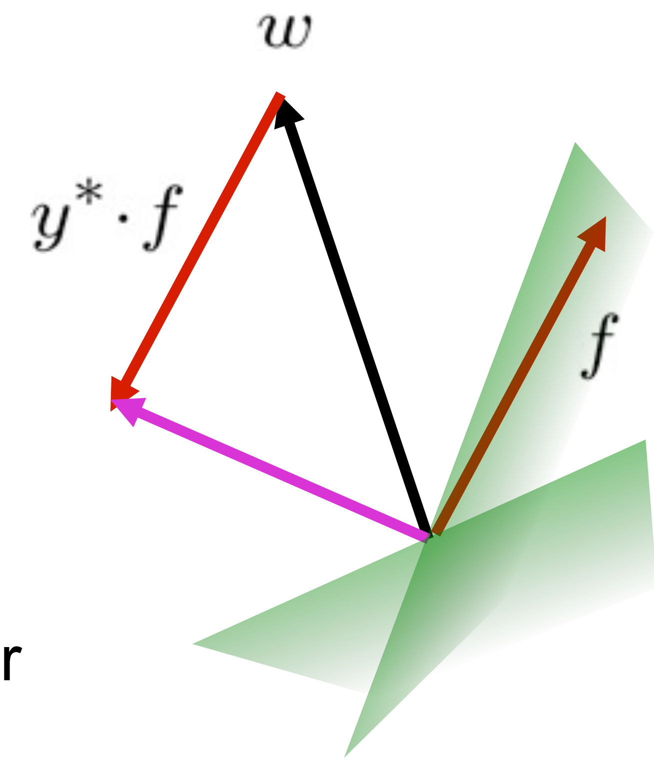
Binary Perceptron Update

- Start with zero weights
- For each training instance:
  - Classify with current weights

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

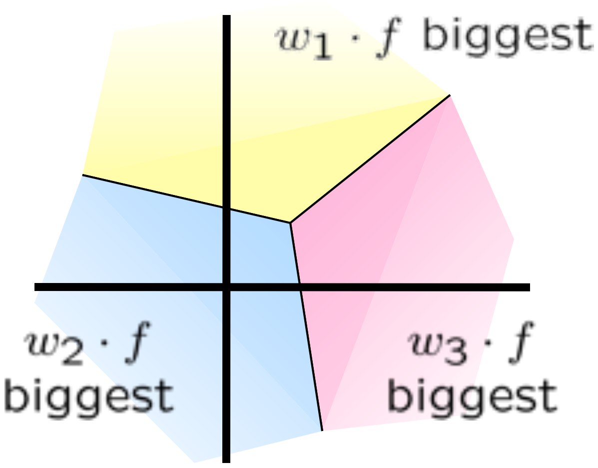
  - If correct (i.e., y=y*), no change!
  - If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if y* is -1.

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



Multiclass Decision Rule

- If we have more than two classes:
  - Have a weight vector for each class: w_y
  - Calculate an activation for each class



$$\text{activation}_w(x, y) = w_y \cdot f(x)$$

  - Highest activation wins

$$y = \arg\max_y \left(\text{activation}_w(x, y)\right)$$

Example

"win the vote"

"win the election"

"win the game"

w_{SPORTS}	$w_{POLITICS}$	w_{TECH}
BIAS :	BIAS :	BIAS :
win :	win :	win :
game :	game :	game :
vote :	vote :	vote :
the :	the :	the :
...	...	...

Example

"win the vote" →

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

w_{SPORTS}

```
BIAS  : -2
win   :  4
game  :  4
vote  :  0
the   :  0
...
```

$w_{POLITICS}$

```
BIAS  :  1
win   :  2
game  :  0
vote  :  4
the   :  0
...
```

w_{TECH}

```
BIAS  :  2
win   :  0
game  :  2
vote  :  0
the   :  0
...
```

The Perceptron Update Rule

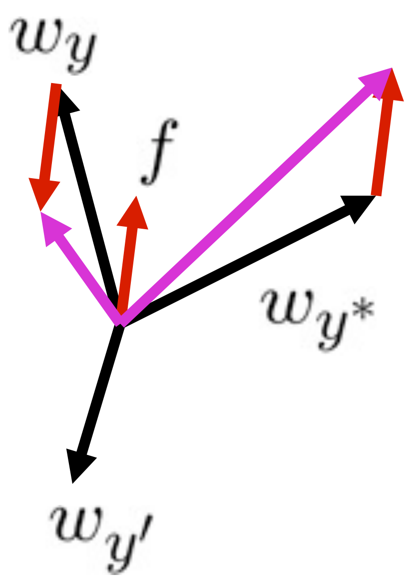
- Start with zero weights
- Iterate training examples
  - Classify with current weights

$$
\begin{aligned}
y &= \arg\max_y \; w_y \cdot f(x) \\
&= \arg\max_y \; \textstyle\sum_i w_{y,i} \cdot f_i(x)
\end{aligned}
$$

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

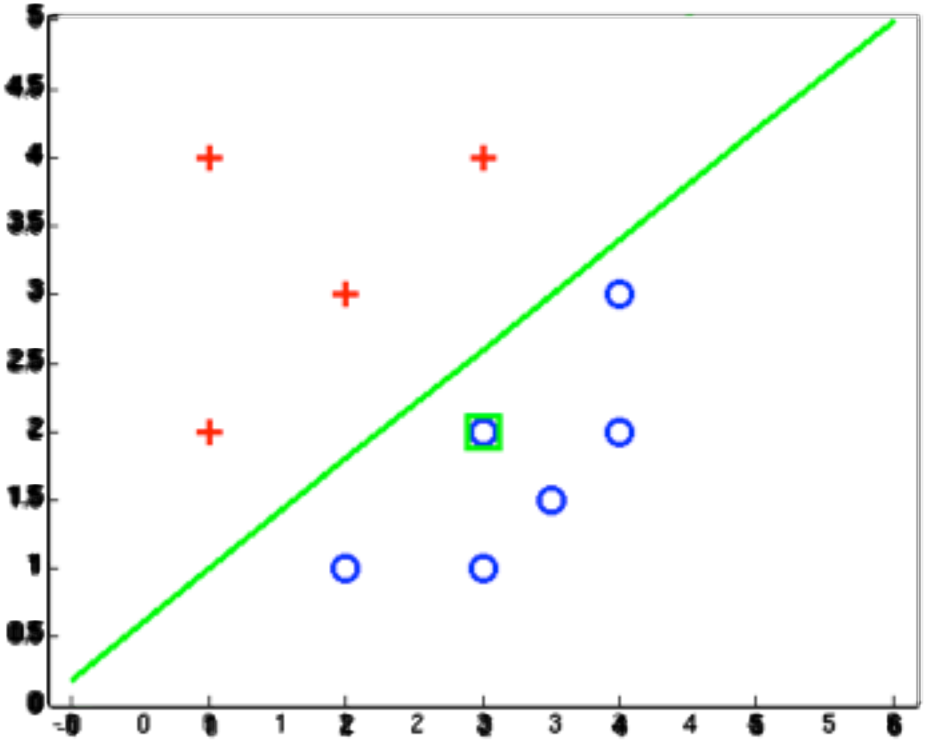
$$w_y = w_y - f(x)$$

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



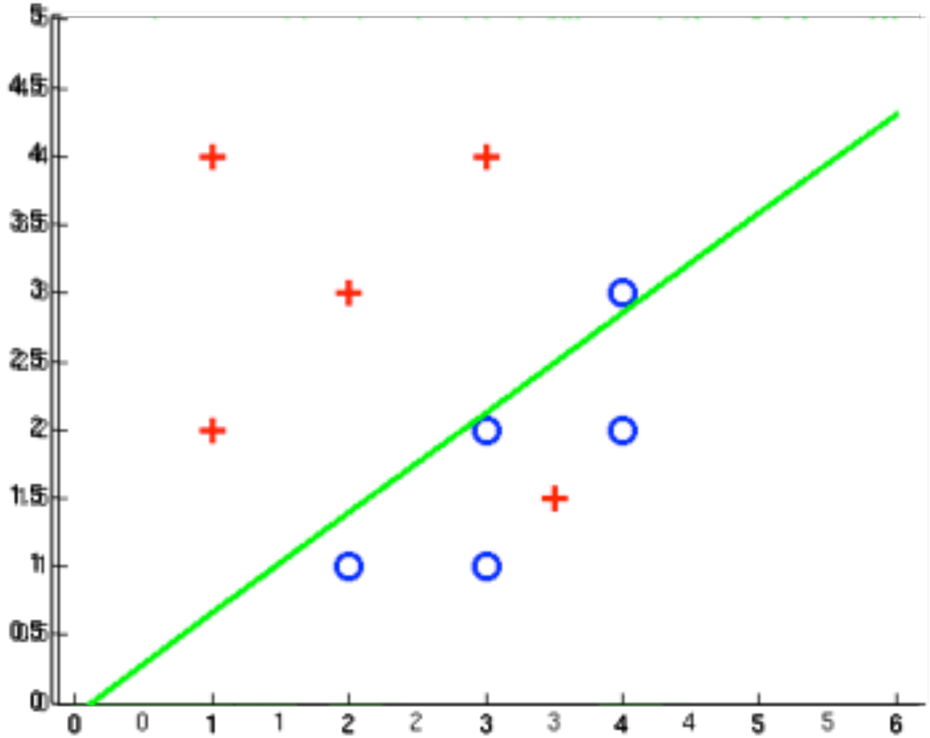
Examples: Perceptron

- Separable Case



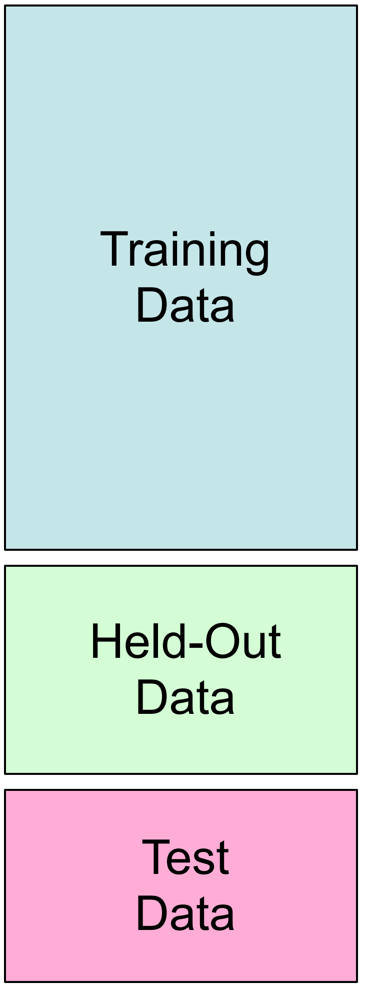
Examples: Perceptron

- Non-Separable Case



Mistake-Driven Classification

- For Naïve Bayes:
  - Parameters from data statistics
  - Parameters: probabilistic interpretation
  - Training: one pass through the data
- For the perceptron:
  - Parameters from reactions to mistakes
  - Parameters: discriminative interpretation
  - Training: go through the data until held-out accuracy maxes out



Properties of Perceptrons

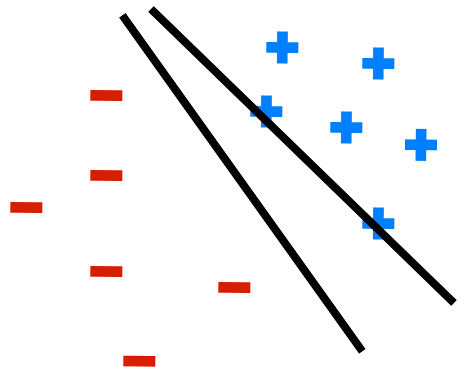
- Separability: some parameters get the training set perfectly correct

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

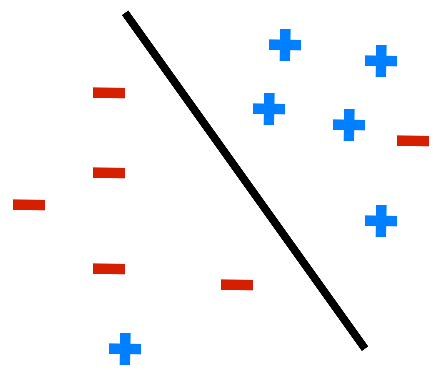
- Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

$$\text{mistakes} < \frac{k}{\delta^2}$$

Separable

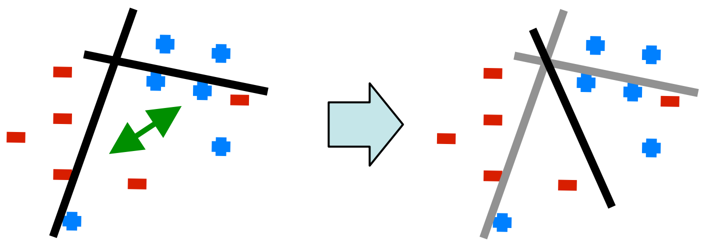


Non-Separable

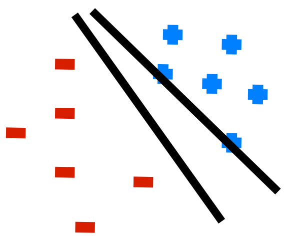


Problems with the Perceptron

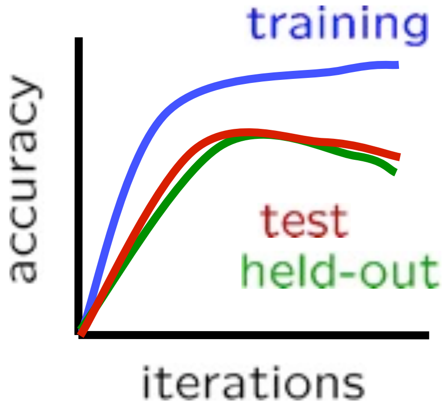
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



- Mediocre generalization: finds a "barely" separating solution



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



Fixing the Perceptron

- Idea: adjust the weight update to mitigate these effects

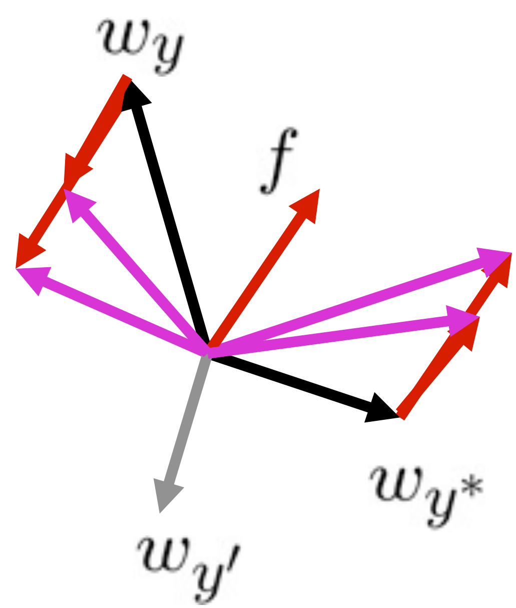
- MIRA*: choose an update size that fixes the current mistake…
- … but, minimizes the change to w

$$\min_{w} \frac{1}{2} \sum_{y} ||w_y - w'_y||^2$$

$$w_{y^*} \cdot f(x) \geq w_y \cdot f(x) + 1$$

- The +1 helps to generalize

* Margin Infused Relaxed Algorithm



Guessed y instead of y^* on example x with features $f(x)$

$$w_y = w'_y - \tau f(x)$$
$$w_{y^*} = w'_{y^*} + \tau f(x)$$

Minimum Correcting Update

$$\min_{w} \frac{1}{2} \sum_{y} ||w_y - w'_y||^2$$

$$w_{y^*} \cdot f \geq w_y \cdot f + 1$$

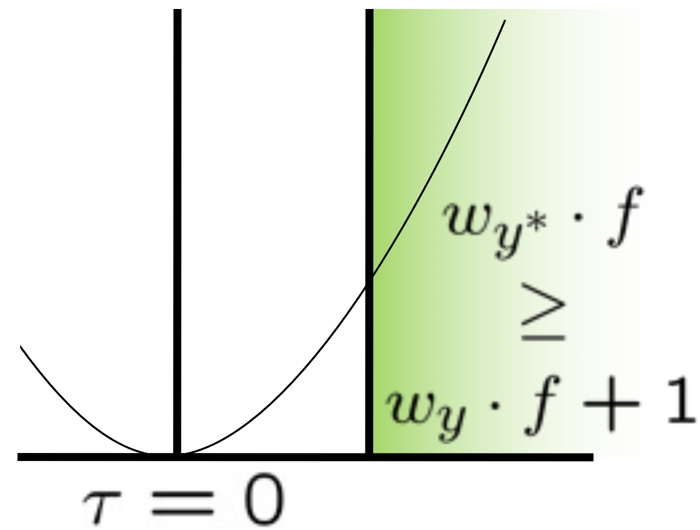
$$\min_{\tau} ||\tau f||^2$$

$$w_{y^*} \cdot f \geq w_y \cdot f + 1$$

$$(w'_{y^*} + \tau f) \cdot f = (w'_y - \tau f) \cdot f + 1$$

$$\tau = \frac{(w'_y - w'_{y^*}) \cdot f + 1}{2 f \cdot f}$$

$$\boxed{\begin{aligned} w_y &= w'_y - \tau f(x) \\ w_{y^*} &= w'_{y^*} + \tau f(x) \end{aligned}}$$

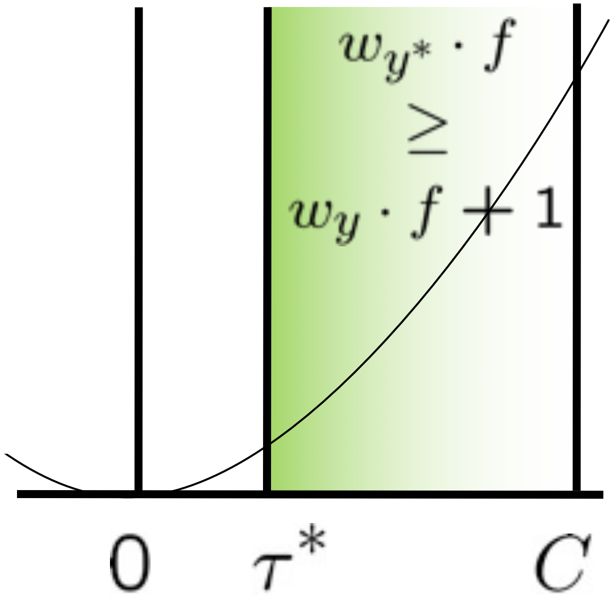


min not τ=0, or would not have made an error, so min will be where equality holds

Maximum Step Size

- In practice, it's also bad to make updates that are too large
  - Example may be labeled incorrectly
  - You may not have enough features
  - Solution: cap the maximum possible value of τ with some constant C

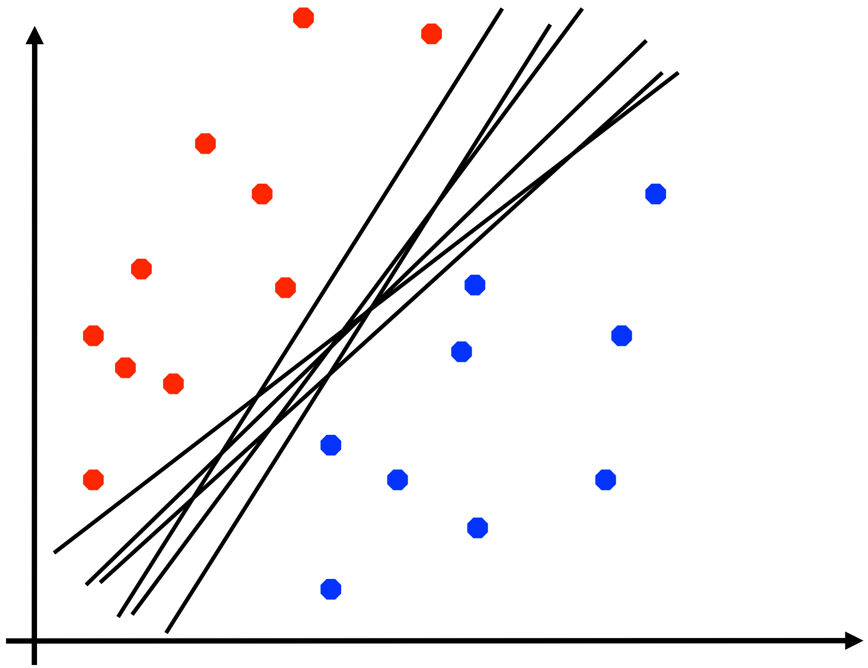
$$\tau^* = \min\left(\frac{(w'_y - w'_{y^*}) \cdot f + 1}{2f \cdot f}, C\right)$$



  - Corresponds to an optimization that assumes non-separable data
  - Usually converges faster than perceptron
  - Usually better, especially on noisy data

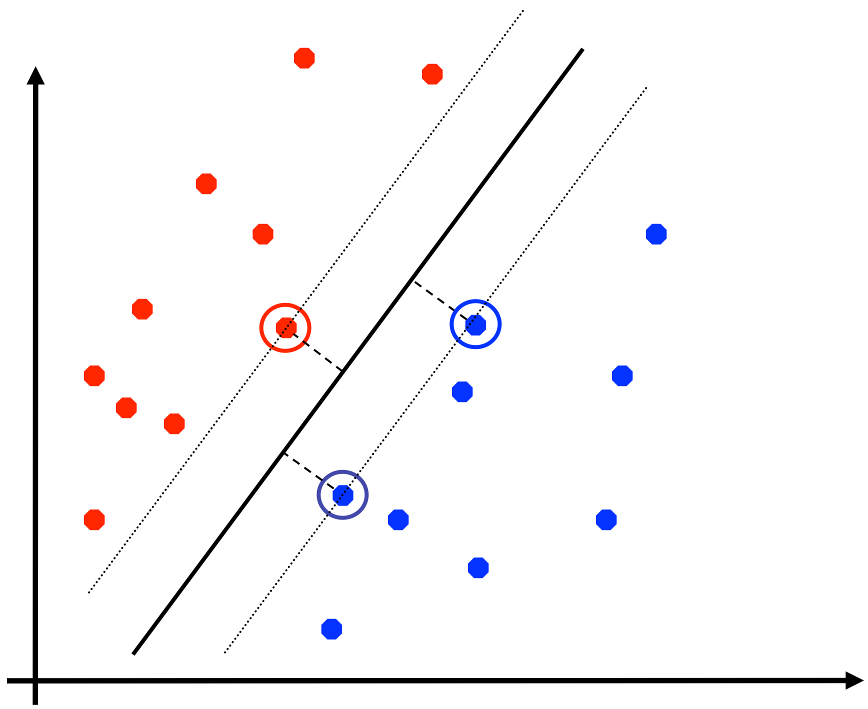
Linear Separators

- Which of these linear separators is optimal?



Support Vector Machines

- Maximizing the margin: good according to intuition, theory, practice
- Only support vectors matter; other training examples are ignorable
- Support vector machines (SVMs) find the separator with max margin
- Basically, SVMs are MIRA where you optimize over all examples at once



MIRA

$$\min_{w} \frac{1}{2}||w - w'||^2$$

$$w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$

SVM

$$\min_{w} \frac{1}{2}||w||^2$$

$$\forall i, y \; w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$

Classification: Comparison

- Naïve Bayes
  - Builds a model training data
  - Gives prediction probabilities
  - Strong assumptions about feature independence
  - One pass through data (counting)

- Perceptrons / MIRA:
  - Makes less assumptions about data
  - Mistake-driven learning
  - Multiple passes through data (prediction)
  - Often more accurate

Extension: Web Search

x = "Apple Computers"

- Information retrieval:
  - Given information needs, produce information
  - Includes, e.g. web search, question answering, and classic IR

- Web search: not exactly classification, but rather ranking

Feature-Based Ranking

x = "Apple Computers"

$f(x,$ 

Apple

From Wikipedia, the free encyclopedia

This article is about the fruit. For the electronics and software company, see Apple Inc.. For other uses, see Apple (disambiguation).

The **apple** is the pomaceous fruit of the apple tree, species *Malus domestica* in the rose family Rosaceae. It is one of the most widely cultivated tree fruits. The tree is small and deciduous, reaching 3 to 12 metres (9.8 to 39 ft) tall, with a broad, often densely twiggy crown.[1] The leaves are alternately arranged simple

Apple

$) = [0.3\ 5\ 0\ 0\ \ldots]$

$f(x,$ 

Apple Inc.

From Wikipedia, the free encyclopedia

(Redirected from Apple Computer)

Apple Inc.,

Apple Inc.

$) = [0.8\ 4\ 2\ 1\ \ldots]$

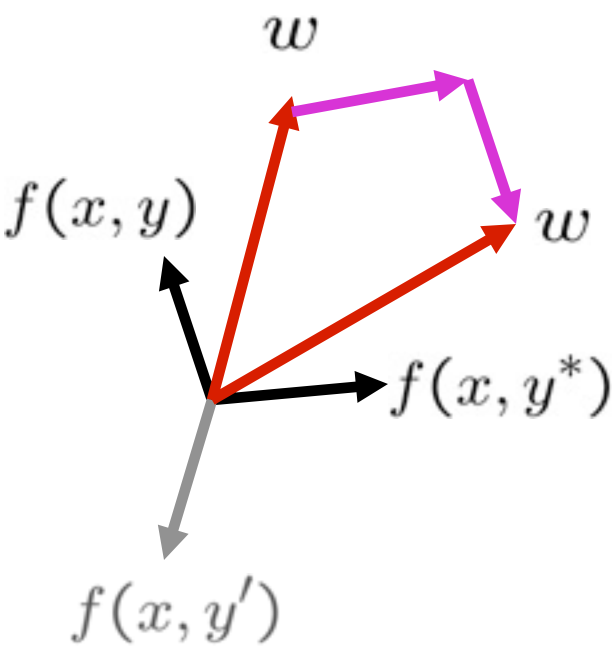
Perceptron for Ranking

- Inputs x
- Candidates y
- Many feature vectors: $f(x, y)$
- One weight vector: w
  - Prediction:

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

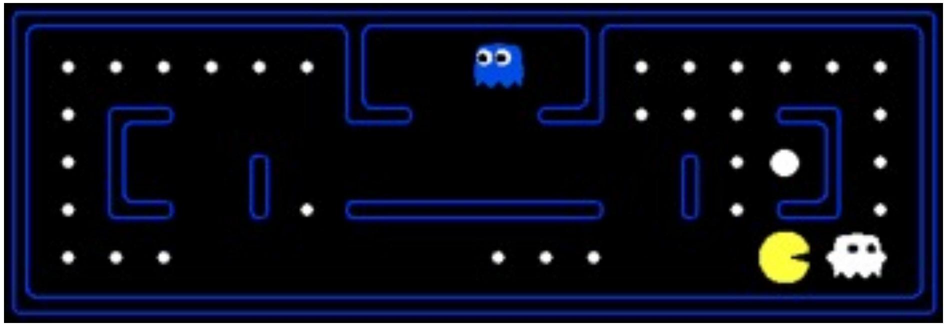
  - Update (if wrong):

$$w = w + f(x, y^*) - f(x, y)$$



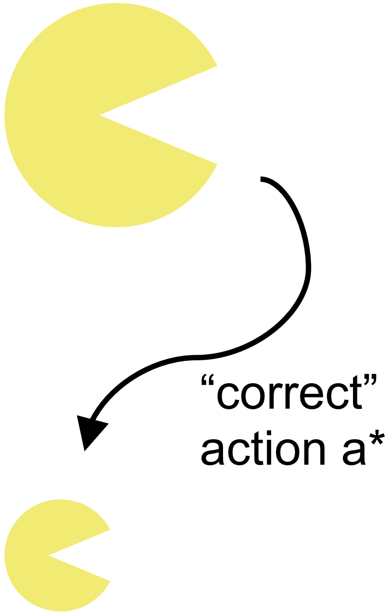
Pacman Apprenticeship!

- Examples are states s



- Candidates are pairs (s,a)
- “Correct” actions: those taken by expert
- Features defined over (s,a) pairs: f(s,a)
- Score of a q-state (s,a) given by:

$$w \cdot f(s, a)$$



$$\forall a \neq a^*,$$
$$w \cdot f(a^*) > w \cdot f(a)$$

- How is this VERY different from reinforcement learning?