

# CS 188: Artificial Intelligence

## Fall 2007

### Lecture 24: Perceptrons

11/20/2007

Dan Klein – UC Berkeley

## General Naïve Bayes

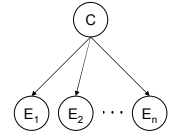
- A general *naive Bayes* model:

$|C| \times |E|^n$   
parameters

$$P(\text{Cause}, \text{Effect}_1 \dots \text{Effect}_n) =$$

$$P(\text{Cause}) \prod_i P(\text{Effect}_i | \text{Cause})$$

$|C|$  parameters       $n \times |E| \times |C|$  parameters



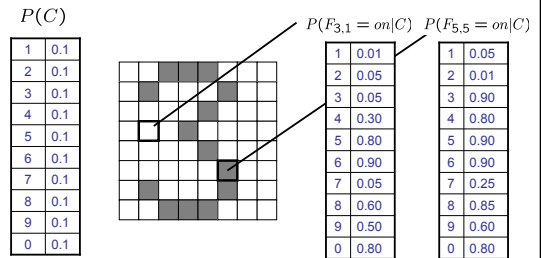
- We only specify how each feature depends on the class
- Total number of parameters is *linear* in  $n$

## Example: Spam Filtering

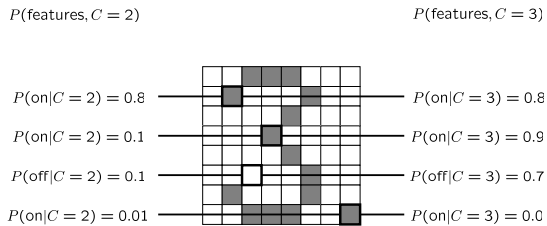
- Model:  $P(C, W_1 \dots W_n) = P(C) \prod_i P(W_i | C)$
- Parameters:

$P(C)$	$P(W \text{spam})$	$P(W \text{ham})$
ham : 0.66	the : 0.016	the : 0.021
spam: 0.33	to : 0.015	to : 0.013
	and : 0.012	and : 0.011
	...	...
	free : 0.001	free : 0.005
	click : 0.001	click : 0.004
	...	...
	morally : 0.001	screens : 0.000
	nicely : 0.001	minute : 0.000
	...	...

## Example: OCR



## Example: Overfitting



2 wins!!

## Generalization and Overfitting

- Relative frequency parameters will overfit the training data!
  - 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?
  - 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

$$\theta_{ML} = \arg \max_{\theta} P(X|\theta) \Rightarrow \hat{P}(x) = \frac{\text{count}(x)}{\text{total samples}}$$

$$= \arg \max_{\theta} \prod_i P_{\theta}(X_i)$$

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

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

$$= \arg \max_{\theta} P(X|\theta)P(\theta)/P(X) \Rightarrow \text{????}$$

$$= \arg \max_{\theta} P(X|\theta)P(\theta)$$

## 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 (see cs281a)

## Estimation: Laplace Smoothing

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



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

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

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

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

- Laplace for conditionals:
  - Smooth each condition independently:

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

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

## 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  $\alpha$  is 0? 1?

- For even better ways to estimate parameters, as well as details of the math see cs281a, cs294

## Real NB: Smoothing

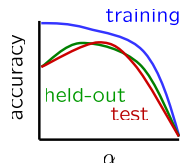
- For real classification problems, smoothing is critical
- New odds ratios:

$\frac{P(W \text{ham})}{P(W \text{spam})}$	$\frac{P(W \text{spam})}{P(W \text{ham})}$
helvetica : 11.4	verdana : 28.8
seems : 10.8	Credit : 28.4
group : 10.2	ORDER : 27.2
ago : 8.4	<FONT> : 26.9
areas : 8.3	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:  $k$ ,  $\alpha$
- 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

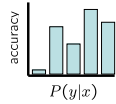
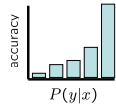
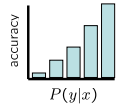


## Baselines

- First task: 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
- 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
$$\text{confidence}(x) = \arg \max_y P(y|x)$$
  - 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?



## Generative vs. Discriminative

- Generative classifiers:
  - E.g. naïve Bayes
  - We build a causal model of the variables
  - We then query that model for causes, given evidence
- Discriminative classifiers:
  - E.g. perceptron (next)
  - No causal model, no Bayes rule, often no probabilities
  - Try to predict output directly
  - Loosely: mistake driven rather than model driven

## 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?

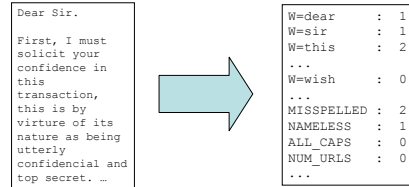
- Need more features— words aren't enough!
  - Have you emailed the sender before?
  - Have 1K other people just gotten the same email?
  - Is the sending information consistent?
  - Is the email in ALL CAPS?
  - Do inline URLs point where they say they point?
  - Does the email address you by (your) name?
- Naïve Bayes models can incorporate a variety of features, but tend to do best in homogeneous cases (e.g. all features are word occurrences)

## Features

- A **feature** is a function which signals a property of the input
- Examples:
  - ALL\_CAPS: value is 1 iff email in all caps
  - HAS\_URL: value is 1 iff email has a URL
  - NUM\_URLS: number of URLs in email
  - VERY\_LONG: 1 iff email is longer than 1K
  - SUSPICIOUS\_SENDER: 1 iff reply-to domain doesn't match originating server
- Features are anything you can think of code to evaluate on an input
  - Some cheap, some very very expensive to calculate
  - Can even be the output of another classifier
  - Domain knowledge goes here!
- In naïve Bayes, how did we encode features?

## Feature Extractors

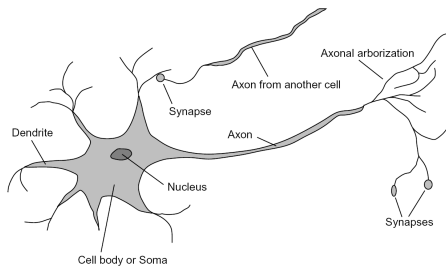
- A **feature extractor** maps inputs to **feature vectors**



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

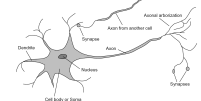
## Some (Vague) Biology

- Very loose inspiration: human neurons



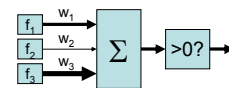
## The Binary Perceptron

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



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

- If the activation is:
  - Positive, output 1
  - Negative, output 0



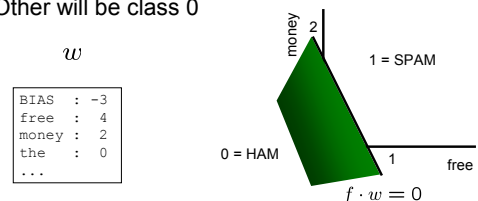
## Example: Spam

- Imagine 4 features:
  - Free (number of occurrences of "free")
  - Money (occurrences of "money")
  - BIAS (always has value 1)

$x$	$f(x)$	$w$	$\sum_i w_i \cdot f_i(x)$
"free money"	BIAS : 1 free : 1 money : 1 the : 0 ...	BIAS : -3 free : 4 money : 2 the : 0 ...	(1)(-3) + (1)(4) + (1)(2) + (0)(0) + ... = 3

## Binary Decision Rule

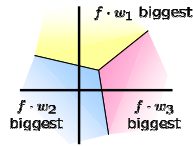
- In the space of feature vectors
  - Any weight vector is a hyperplane
  - One side will be class 1
  - Other will be class 0



## The Multiclass Perceptron

- If we have more than two classes:

- Have a weight vector for each class
- Calculate an activation for each class



$$\text{activation}_w(x, c) = \sum_i w_{c,i} \cdot f_i(x)$$

- Highest activation wins

$$c = \arg \max_c (\text{activation}_w(x, c))$$

## 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
- Pick up training instances one by one
- Try to classify

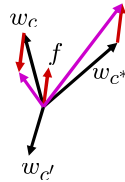
$$c = \arg \max_c w_c \cdot f(x)$$

$$= \arg \max_c \sum_i w_{c,i} \cdot f_i(x)$$

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

$$w_c = w_c - f(x)$$

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



## Example

“win the vote”

“win the election”

“win the game”

$w_{SPORTS}$

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

$w_{POLITICS}$

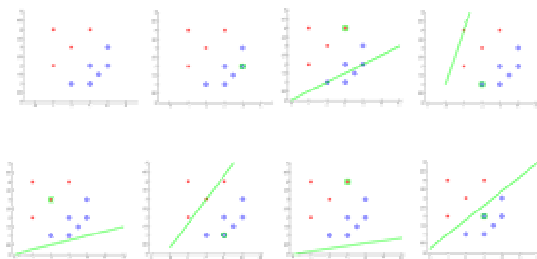
```
BIAS :
win  :
game :
vote :
the  :
...
```

$w_{TECH}$

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

## Examples: Perceptron

- Separable Case



## Mistake-Driven Classification

- In naïve Bayes, parameters:
  - From data statistics
  - Have a causal interpretation
  - One pass through the data
- For the perceptron parameters:
  - From reactions to mistakes
  - Have a discriminative interpretation
  - Go through the data until held-out accuracy maxes out

Training Data

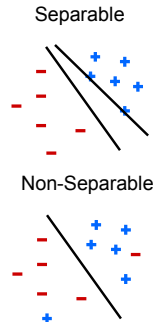
Held-Out Data

Test Data

## Properties of Perceptrons

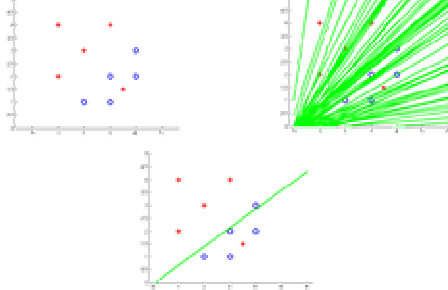
- Separability: some parameters get the training set perfectly correct
- Convergence: if the training is separable, perceptron will eventually converge (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

$$\text{mistakes} < \frac{1}{\delta^2}$$



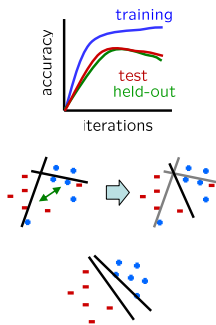
## Examples: Perceptron

- Non-Separable Case



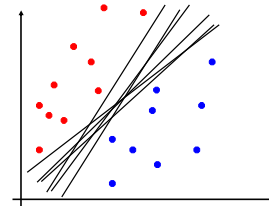
## Issues with Perceptrons

- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining isn't quite as bad as overfitting, but is similar
- Regularization: if the data isn't separable, weights might thrash around
  - Averaging weight vectors over time can help (averaged perceptron)
- Mediocre generalization: finds a "barely" separating solution



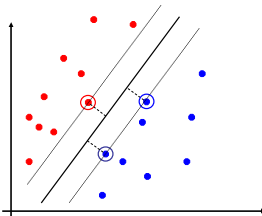
## Linear Separators

- Which of these linear separators is optimal?



## Support Vector Machines

- Maximizing the margin: good according to intuition and PAC theory.
- Only support vectors matter; other training examples are ignorable.
- Support vector machines (SVMs) find the separator with max margin
- Mathematically, gives a quadratic program to solve
- Basically, SVMs are perceptrons with smarter update counts!



## Summary

- Naïve Bayes
  - Build classifiers using model of training data
  - Smoothing estimates is important in real systems
  - Classifier confidences are useful, when you can get them
- Perceptrons:
  - Make less assumptions about data
  - Mistake-driven learning
  - Multiple passes through data