

# Discriminative Models for Text Classification

Mausam

(Slides by Michael Collins, Emily Fox, Dan Jurafsky, Dan Klein, Chris Manning, Ray Mooney, Dan Weld, Alex Yates, Luke Zettlemoyer)

# Introduction

- So far we've looked at “generative models”
  - Naive Bayes
- But there is now much use of conditional or discriminative probabilistic models in NLP, Speech, IR (and ML generally)
- Because:
  - They give high accuracy performance
  - They make it easy to incorporate lots of linguistically important features
  - They allow automatic building of language independent, retargetable NLP modules

# Joint vs. Conditional Models

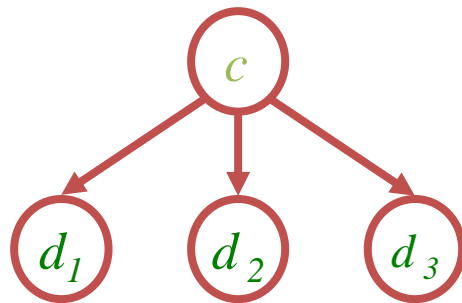
- We have some data  $\{(d, c)\}$  of paired observations  $d$  and hidden classes  $c$ .
- **Joint (generative) models** place probabilities over both observed data and the hidden stuff (generate the observed data from hidden stuff):
  - All the classic Stat-NLP models:
    - $n$ -gram models, Naive Bayes classifiers, hidden Markov models, probabilistic context-free grammars, IBM machine translation alignment models

# Joint vs. Conditional Models

- Discriminative (conditional) models take the data as given, and put a probability over hidden structure given the data:
  - Logistic regression, conditional loglinear or maximum entropy models, conditional random fields
  - Also, SVMs, (averaged) perceptron, etc. are discriminative classifiers (but not directly probabilistic)

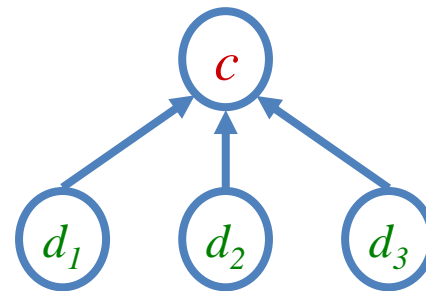
# Bayes Net/Graphical Models

- Bayes net diagrams draw circles for random variables, and lines for direct dependencies
- Some variables are observed; some are hidden
- Each node is a little classifier (conditional probability table) based on incoming arcs



Naive Bayes

Generative



Logistic Regression

Discriminative

# Conditional vs. Joint Likelihood

- A *joint* model gives probabilities  $P(d,c)$  and tries to maximize this joint likelihood.
  - It turns out to be trivial to choose weights: just relative frequencies.
- A *conditional* model gives probabilities  $P(c|d)$ . It takes the data as given and models only the conditional probability of the class.
  - We seek to maximize conditional likelihood.
  - Harder to do (as we'll see...)
  - More closely related to classification error.

# Case Study: Word Senses

- Words have multiple distinct meanings, or senses:
  - Plant: living plant, manufacturing plant, ...
  - Title: name of a work, ownership document, form of address, material at the start of a film, ...
- Many levels of sense distinctions
  - Homonymy: totally unrelated meanings (river bank, money bank)
  - Polysemy: related meanings (star in sky, star on tv)
  - Systematic polysemy: productive meaning extensions (metonymy such as organizations to their buildings) or metaphor
  - Sense distinctions can be extremely subtle (or not)
- Granularity of senses needed depends a lot on the task
- Why is it important to model word senses?
  - Translation, parsing, information retrieval?

# Word Sense Disambiguation

- Example: living plant vs. manufacturing plant
- How do we tell these senses apart?
  - “context”

The manufacturing **plant** which had previously sustained the town's economy shut down after an extended labor strike.

- Maybe it's just text categorization
  - Each word sense represents a class
  - Run a naive-bayes classifier?
- Bag-of-words classification works OK for noun senses
  - 90% on classic, shockingly easy examples (line, interest, star)
  - 80% on senseval-1 nouns
  - 70% on senseval-1 verbs



# Verb WSD

- Why are verbs harder?
  - Verbal senses less topical
  - More sensitive to structure, argument choice
- Verb Example: “Serve”
  - [function] The tree stump serves as a table
  - [enable] The scandal served to increase his popularity
  - [dish] We serve meals for the homeless
  - [enlist] She served her country
  - [jail] He served six years for embezzlement
  - [tennis] It was Agassi's turn to serve
  - [legal] He was served by the sheriff

# Better Features

- There are smarter features:
  - Argument selectional preference:
    - serve NP[meals] vs. serve NP[papers] vs. serve NP[country]
  - Subcategorization:
    - [function] serve PP[as]
    - [enable] serve VP[to]
    - [tennis] serve <intransitive>
    - [food] serve NP {PP[to]}
  - Can be captured poorly (but robustly) with modified Naïve Bayes approach
- Other constraints (Yarowsky 95)
  - One-sense-per-discourse (only true for broad topical distinctions)
  - One-sense-per-collocation (pretty reliable when it kicks in: manufacturing plant, flowering plant)

# Complex Features with NB?

- **Example:** Washington County jail **served** 11,166 meals last month - a figure that translates to feeding some 120 people three times daily for 31 days.
- **So we have a decision to make based on a set of cues:**
  - context:jail, context:county, context:feeding, context:meals, ...
  - subcat:NP, direct-object-head:meals
- **Not clear how build a generative derivation for these:**
  - Choose topic, then decide on having a transitive usage, then pick “meals” to be the object’s head, then generate other words?
  - Hard to make this work (though maybe possible)
  - No real reason to try

# A Discriminative Approach

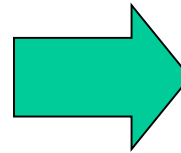
- View WSD as a discrimination task, directly estimate:

$P(\text{sense} \mid \text{context:jail, context:county,}$   
 $\text{context:feeding, context:meals, ...}$   
 $\text{subcat:NP, direct-object-head:meals, ....})$

- Have to estimate multinomial (over senses) where there are a huge number of things to condition on
- Many feature-based classification techniques out there
  - Log-linear models extremely popular in the NLP community!

# Feature Representations

Washington County jail **served**  
11,166 meals last month - a  
figure that translates to feeding  
some 120 people three times  
daily for 31 days.



context:jail = 1  
context:county = 1  
context:feeding = 1  
context:game = 0  
...  
local-context:jail = 1  
local-context:meals = 1  
...  
subcat:NP = 1  
subcat:PP = 0  
...  
object-head:meals = 1  
object-head:ball = 0

- Features are indicator functions which count the occurrences of certain patterns in the input
- *We will have different feature values for every pair of input  $x$  and class  $y$*

# Features

- In NLP uses, usually a feature specifies

1. an indicator function – a yes/no boolean matching function – of properties of the input and
2. a particular class

$$\phi_i(x, y) \equiv [\Phi(x) \wedge y = y_j] \quad [\text{Value is 0 or 1}]$$

- Each feature picks out a data subset and suggests a label for it

# Example of Features

- context:jail & served:functional
  - context:jail & served:dish
  - ...
- 
- subcat:NP & served:functional
  - subcat:NP & served:dish
  - ...

# Feature-Based Linear Classifiers

- Linear classifiers at classification time:
  - Linear function from feature sets  $\{\phi_i\}$  to classes  $\{y\}$ .
  - Assign a weight  $w_i$  to each feature  $\phi_i$ .
  - We consider each class for an observed datum  $x$
- For a pair  $(x,y)$ , features vote with their weights:
  - $\text{vote}(y) = \sum w_i \phi_i(x,y)$
  - Choose the class  $y$  which maximizes  $\sum w_i \phi_i(x,y)$



# Exponential Models (log-linear, maxent, Logistic, Gibbs)

- Model: use the scores as probabilities:

$$p(y|x; w) = \frac{\exp(w \cdot \phi(x, y))}{\sum_{y'} \exp(w \cdot \phi(x, y'))}$$

← Make positive  
← Normalize

- Learning: maximize the (log) conditional likelihood of training data  $\{(x_i, y_i)\}_{i=1}^n$

$$L(w) = \sum_{i=1}^n \log p(y_i|x_i; w) \quad w^* = \arg \max_w L(w)$$

- Prediction: output  $\arg \max_y p(y|x; w)$

# Derivative of Log-linear Model

$$p(y|x; w) = \frac{\exp(w \cdot \phi(x, y))}{\sum_{y'} \exp(w \cdot \phi(x, y'))}$$

- Unfortunately,  $\operatorname{argmax}_w L(w)$  doesn't have a close formed solution
- We will have to differentiate and use gradient ascent

$$L(w) = \sum_{i=1}^n \log p(y_i|x_i; w)$$

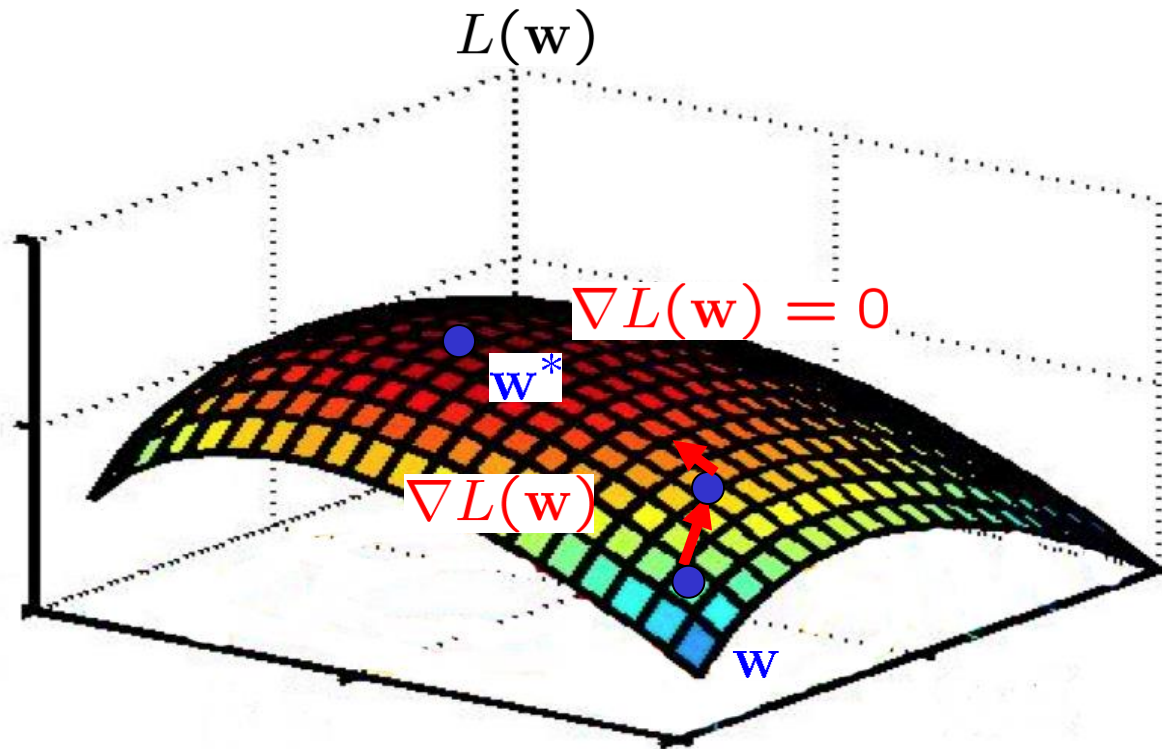
$$L(w) = \sum_{i=1}^n \left( w \cdot \phi(x_i, y_i) - \log \sum_y \exp(w \cdot \phi(x_i, y)) \right)$$

$$\frac{\partial}{\partial w_j} L(w) = \sum_{i=1}^n \left( \phi_j(x_i, y_i) - \sum_y p(y|x_i; w) \phi_j(x_i, y) \right)$$

Total count of feature j  
in correct candidates

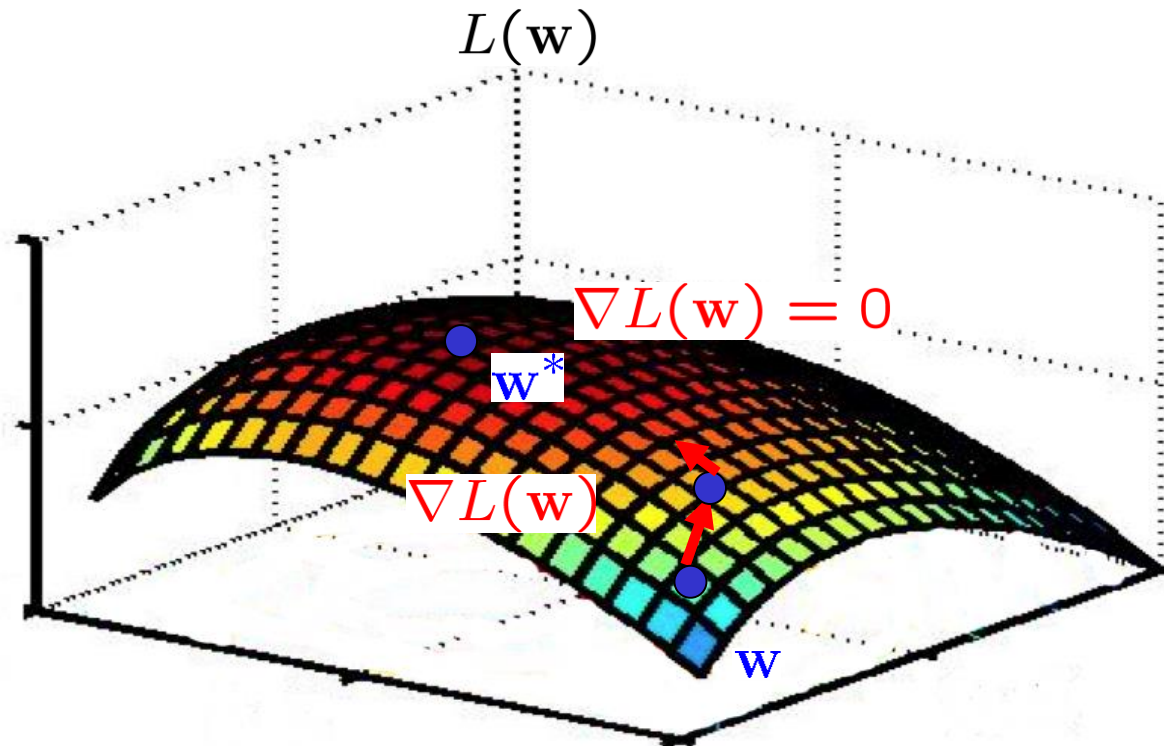
Expected count of  
feature j in predicted  
candidates

# Unconstrained Optimization



- Basic idea: move uphill from current guess
- Gradient ascent / descent follows the gradient incrementally
- At local optimum, derivative vector is zero
- Will converge if step sizes are small enough, but not efficient
- All we need is to be able to evaluate the function and its derivative

# Unconstrained Optimization



- For convex functions, a local optimum will be global
- Basic gradient ascent isn't very efficient, but there are simple enhancements which take into account previous gradients: conjugate gradient, L-BFGS
- There are special-purpose optimization techniques for maxent, like iterative scaling, but they aren't better

# What About Overfitting?

- For Naïve Bayes, we were worried about zero counts in MLE estimates
  - Can that happen here?
- Regularization (smoothing) for Log-linear models
  - Instead, we worry about large feature weights
  - Add a regularization term to the likelihood to push weights towards zero

$$L(w) = \sum_{i=1}^n \log p(y_i | x_i; w) - \frac{\lambda}{2} ||w||^2$$

# Derivative for Regularized Maximum Entropy


- Unfortunately,  $\operatorname{argmax}_w L(w)$  still doesn't have a close formed solution
- We will have to differentiate and use gradient ascent

$$L(w) = \sum_{i=1}^n \left( w \cdot \phi(x_i, y_i) - \log \sum_y \exp(w \cdot \phi(x_i, y)) \right) - \frac{\lambda}{2} ||w||^2$$

$$\frac{\partial}{\partial w_j} L(w) = \sum_{i=1}^n \left( \phi_j(x_i, y_i) - \sum_y p(y|x_i; w) \phi_j(x_i, y) \right) - \lambda w_j$$



Total count of feature j  
in correct candidates



Expected count of  
feature j in predicted  
candidates



Big weights  
are bad

# Word Sense Disambiguation Results

[Suarez and Palomar, 2002]

- With clever features, small variations on simple log-linear (Maximum Entropy – ME) models did very well in an word sense competition:

Figure 1: List of types of features

- **0**: ambiguous-word shape
- **s**: words at positions  $\pm 1, \pm 2, \pm 3$
- **p**: POS-tags of words at positions  $\pm 1, \pm 2, \pm 3$
- **b**: lemmas of collocations at positions  $(-2, -1), (-1, +1), (+1, +2)$
- **c**: collocations at positions  $(-2, -1), (-1, +1), (+1, +2)$
- **km**: lemmas of nouns at any position in context, occurring at least  $m\%$  times with a sense
- **r**: grammatical relation of the ambiguous word
- **d**: the word that the ambiguous word depends on
- **L**: lemmas of content-words at positions  $\pm 1, \pm 2, \pm 3$  (“relaxed” definition)
- **W**: content-words at positions  $\pm 1, \pm 2, \pm 3$  (“relaxed” definition)
- **S, B, C, P, and D**: “relaxed” versions

Table 5: Comparing with SENSEVAL-2 systems

ALL		Nouns		Verbs		Adjectives	
0.713	jhu(R)	0.702	jhu(R)	0.643	jhu(R)	0.802	jhu(R)
0.684	<b>vME+SM</b>	0.702	<b>vME+SM</b>	0.609	jhu	0.774	<b>vME</b>
0.682	jhu	0.683	<b>MEbfs.pos</b>	0.595	css244	0.772	<b>MEbfs.pos</b>
0.677	<b>MEbfs.pos</b>	0.681	jhu	0.584	umd-sst	0.772	css244
0.676	<b>vME</b>	0.678	<b>vME</b>	0.583	<b>vME</b>	0.771	<b>MEbfs</b>
0.670	css244	0.661	<b>MEbfs</b>	0.583	<b>MEbfs.pos</b>	0.764	jhu
0.667	<b>MEbfs</b>	0.652	css244	0.583	<b>MEfix</b>	0.756	<b>MEfix</b>
0.658	<b>MEfix</b>	0.646	<b>MEfix</b>	0.580	<b>MEbfs</b>	0.725	duluth 8
0.627	umd-sst	0.621	duluth 8	0.515	duluth 10	0.712	duluth 10
0.617	duluth 8	0.612	duluth Z	0.513	duluth 8	0.706	duluth 7
0.610	duluth 10	0.611	duluth 10	0.511	ua	0.703	umd-sst
0.595	duluth Z	0.603	umd-sst	0.498	duluth 7	0.689	duluth 6
0.595	duluth 7	0.592	duluth 6	0.490	duluth Z	0.689	duluth Z
0.582	duluth 6	0.590	duluth 7	0.478	duluth X	0.687	ua
0.578	duluth X	0.586	duluth X	0.477	duluth 9	0.678	duluth X
0.560	duluth 9	0.557	duluth 9	0.474	duluth 6	0.655	duluth 9
0.548	ua	0.514	duluth Y	0.431	duluth Y	0.637	duluth Y
0.524	duluth Y	0.464	ua				

- The winning system is a famous semi-supervised learning approach by Yarowsky
- The other systems include many different approaches: Naïve Bayes, SVMs, etc

# Features

- In NLP uses, usually a feature specifies

1. an indicator function – a yes/no boolean matching function – of properties of the input and
2. a particular class

$$\phi_i(x, y) \equiv [\Phi(x) \wedge y = y_j] \quad [\text{Value is 0 or 1}]$$

- Each feature picks out a data subset and suggests a label for it



# Exponential Models (log-linear, maxent, Logistic, Gibbs)

- Model: use the scores as probabilities:

$$p(y|x; w) = \frac{\exp(w \cdot \phi(x, y))}{\sum_{y'} \exp(w \cdot \phi(x, y'))}$$

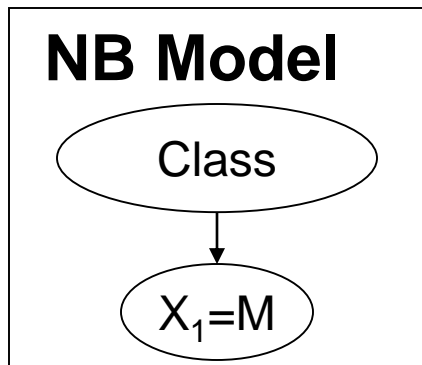
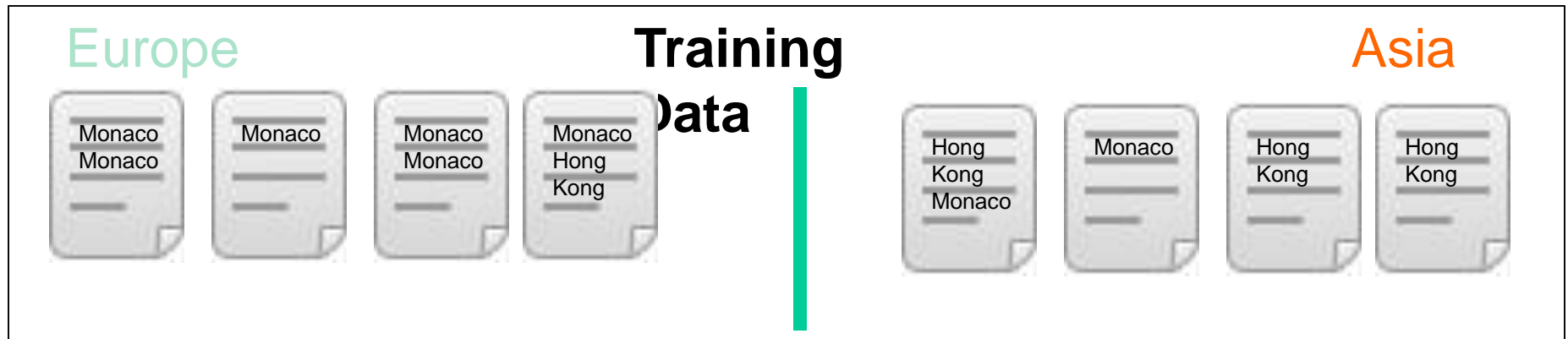
← Make positive  
← Normalize

- Learning: maximize the (log) conditional likelihood of training data  $\{(x_i, y_i)\}_{i=1}^n$

$$L(w) = \sum_{i=1}^n \log p(y_i|x_i; w) \quad w^* = \arg \max_w L(w)$$

- Prediction: output  $\arg \max_y p(y|x; w)$

# Text classification: Asia or Europe



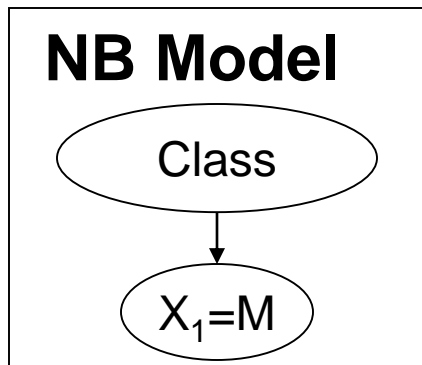
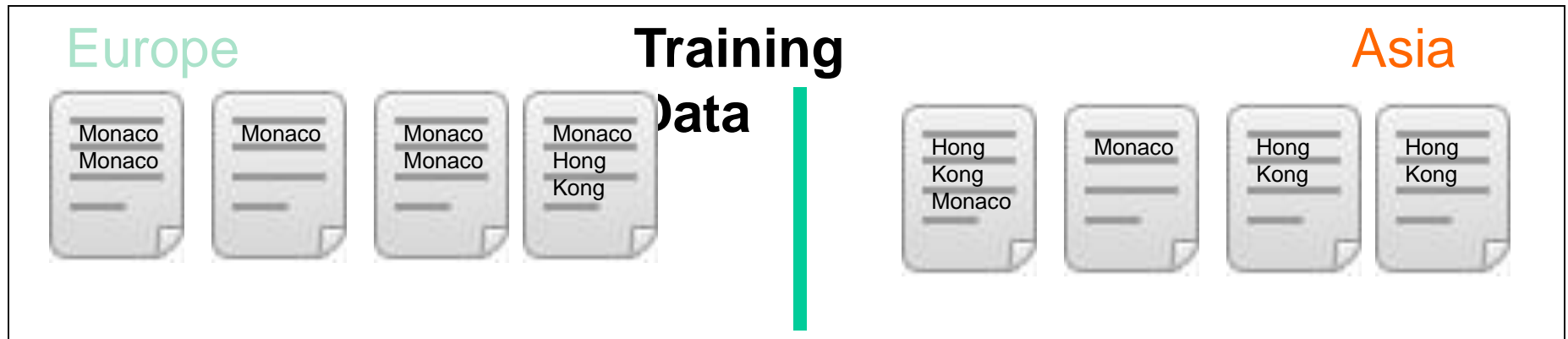
## NB FACTORS:

- $P(A) = P(E) =$
- $P(M|A) =$
- $P(M|E) =$

## PREDICTIONS:

- $P(A,M) =$
- $P(E,M) =$
- $P(A|M) =$
- $P(E|M) =$

# Text classification: Asia or Europe



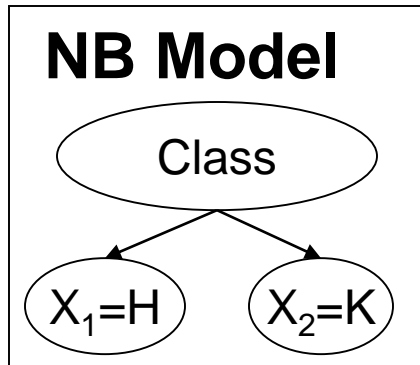
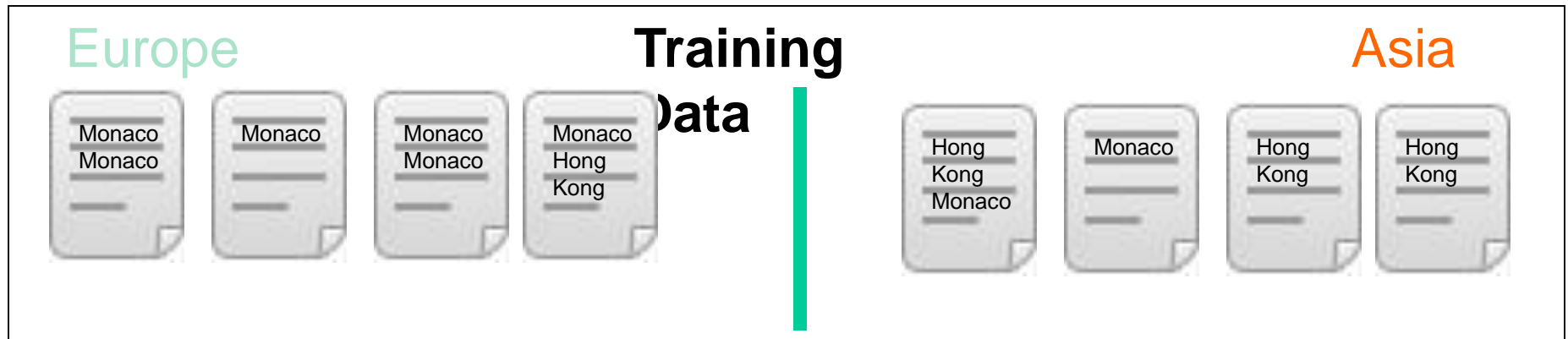
## NB FACTORS:

- $P(A) = P(E) =$
- $P(M|A) =$
- $P(M|E) =$

## PREDICTIONS:

- $P(A,M) =$
- $P(E,M) =$
- $P(A|M) =$
- $P(E|M) =$

# Text classification: Asia or Europe



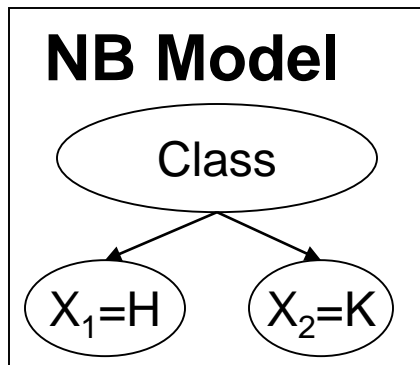
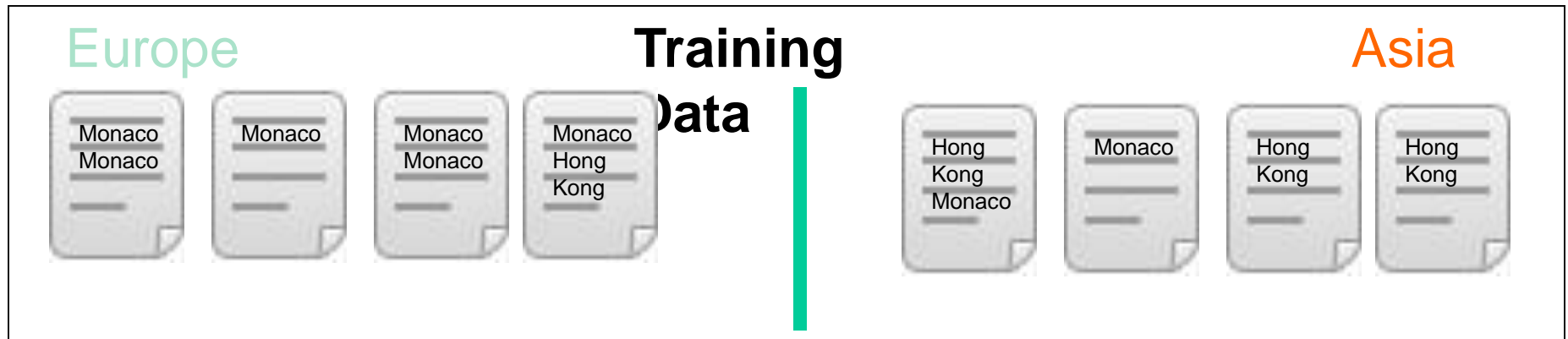
## NB FACTORS:

- $P(A) = P(E) =$
- $P(H|A) = P(K|A) =$
- $P(H|E) = P(K|E) =$

## PREDICTIONS:

- $P(A,H,K) =$
- $P(E,H,K) =$
- $P(A|H,K) =$
- $P(E|H,K) =$

# Text classification: Asia or Europe



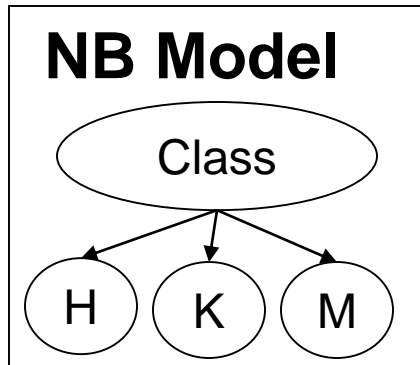
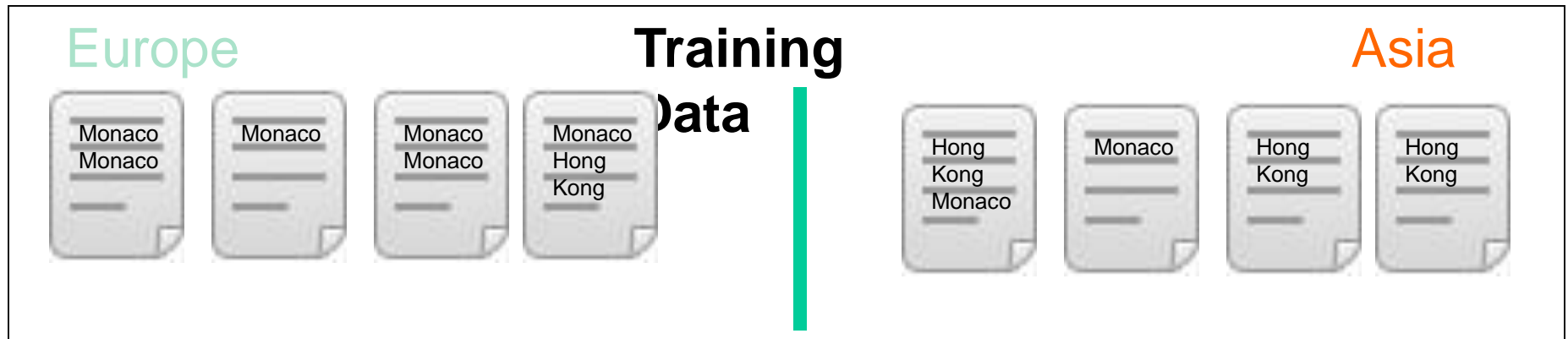
## NB FACTORS:

- $P(A) = P(E) =$
- $P(H|A) = P(K|A) =$
- $P(H|E) = P(K|E) =$

## PREDICTIONS:

- $P(A,H,K) =$
- $P(E,H,K) =$
- $P(A|H,K) =$
- $P(E|H,K) =$

# Text classification: Asia or Europe



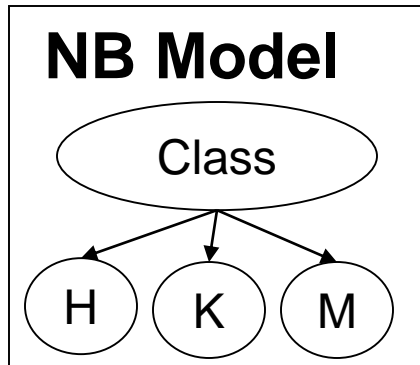
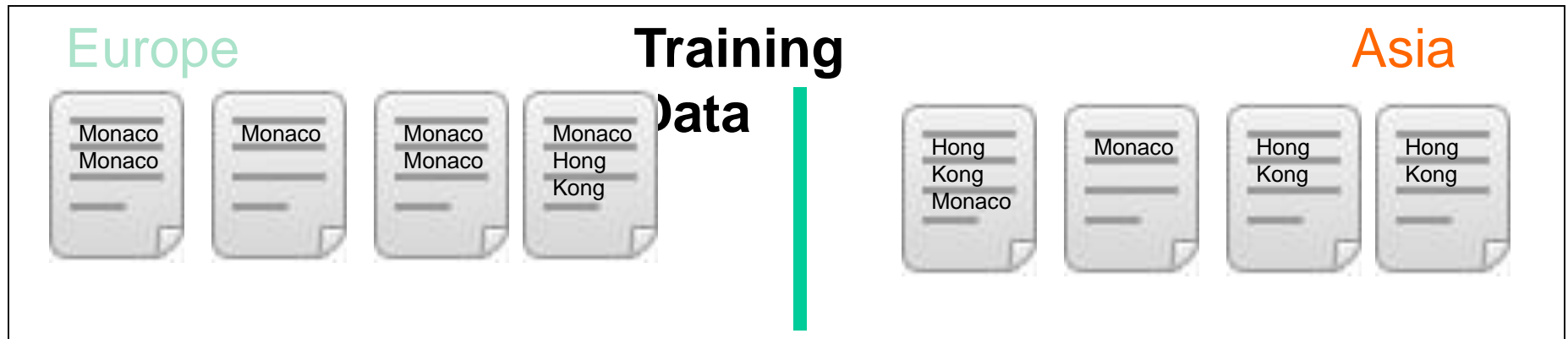
## NB FACTORS:

- $P(A) = P(E) =$
- $P(M|A) =$
- $P(M|E) =$
- $P(H|A) = P(K|A) =$
- $P(H|E) = P(K|E) =$

## PREDICTIONS:

- $P(A, H, K, M) =$
- $P(E, H, K, M) =$
- $P(A|H, K, M) =$
- $P(E|H, K, M) =$

# Text classification: Asia or Europe



## NB FACTORS:

- $P(A) = P(E) =$
- $P(M|A) =$
- $P(M|E) =$
- $P(H|A) = P(K|A) =$
- $P(H|E) = P(K|E) =$

## PREDICTIONS:

- $P(A,H,K,M) =$
- $P(E,H,K,M) =$
- $P(A|H,K,M) =$
- $P(E|H,K,M) =$

# Naive Bayes vs. Maxent Models

- Naive Bayes models multi-count correlated evidence
  - Each feature is multiplied in, even when you have multiple features telling you the same thing
- Maximum Entropy models (pretty much) solve this problem
  - weight features so that model expectations match the observed (empirical) expectations
  - by dividing the weights into all features



# Proof

## (Conditional Likelihood Derivative)

- Recall

$$p(y|x; w) = \frac{\exp(w \cdot \phi(x, y))}{\sum_{y'} \exp(w \cdot \phi(x, y'))}$$

$$P(Y | X, w) = \prod_{(x,y) \in D} p(y | x, w)$$

- We can separate this into two components:

$$\log P(Y | X, w) = \sum_{(x,y) \in D} \log \exp \sum_i w_i \phi_i(x, y) - \sum_{(x,y) \in D} \log \sum_{y'} \exp \sum_i w_i \phi_i(x, y')$$

- The derivative is the difference between the derivatives of each component

$$\log P(Y | X, w) = N(w) - D(w)$$

# Proof: Numerator

$$\begin{aligned}\frac{\partial N(w)}{\partial w_i} &= \frac{\partial \sum_{(x,y) \in D} \log \exp \sum_i w_i \phi_i(x, y)}{\partial w_i} = \frac{\partial \sum_{(x,y) \in D} \sum_i w_i \phi_i(x, y)}{\partial w_i} \\ &= \sum_{(x,y) \in D} \frac{\partial \sum_i w_i \phi_i(x, y)}{\partial w_i} \\ &= \sum_{(x,y) \in D} \phi_i(x, y)\end{aligned}$$

Derivative of the numerator is: the empirical count( $\phi_i, y$ )

# Proof: Denominator

$$\begin{aligned}
 \frac{\partial D(w)}{\partial w_i} &= \frac{\partial \sum_{(x,y) \in D} \log \sum_{y'} \exp \sum_i w_i \phi_i(x, y')}{\partial w_i} \\
 &= \sum_{(x,y) \in D} \frac{1}{\sum_{y''} \exp \sum_i w_i \phi_i(x, y'')} \frac{\partial \sum_{y'} \exp \sum_i w_i \phi_i(x, y')}{\partial w_i} \\
 &= \sum_{(x,y) \in D} \frac{1}{\sum_{y''} \exp \sum_i w_i \phi_i(x, y'')} \sum_{y'} \frac{\exp \sum_i w_i \phi_i(x, y')}{1} \frac{\partial \sum_i w_i \phi_i(x, y')}{\partial w_i} \\
 &= \sum_{(x,y) \in D} \sum_{y'} \frac{\exp \sum_i w_i \phi_i(x, y')}{\sum_{y''} \exp \sum_i w_i \phi_i(x, y'')} \frac{\partial \sum_i w_i \phi_i(x, y')}{\partial w_i} \\
 &= \sum_{(x,y) \in D} \sum_{y'} P(y' | x, w) \phi_i(x, y') \quad = \text{predicted count}(\phi_i, w)
 \end{aligned}$$

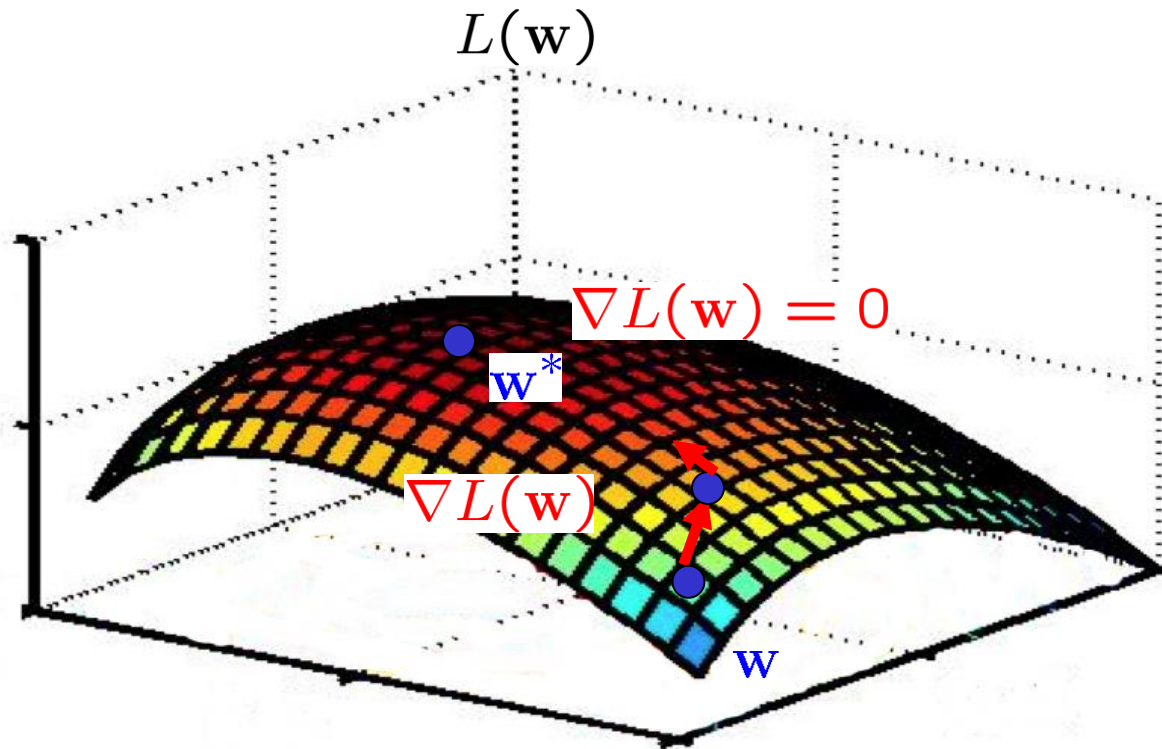
# Proof (concluded)

$$\frac{\partial \log P(Y | X, w)}{\partial w_i} = \text{actual count}(\phi_i, Y) - \text{predicted count}(\phi_i, w)$$

- The optimum parameters are the ones for which each feature's predicted expectation equals its empirical expectation. The optimum distribution is:
  - Always unique (but parameters may not be unique)
  - Always exists (if feature counts are from actual data).
- These models are also called maximum entropy models because we find the model having maximum entropy and satisfying the constraints:

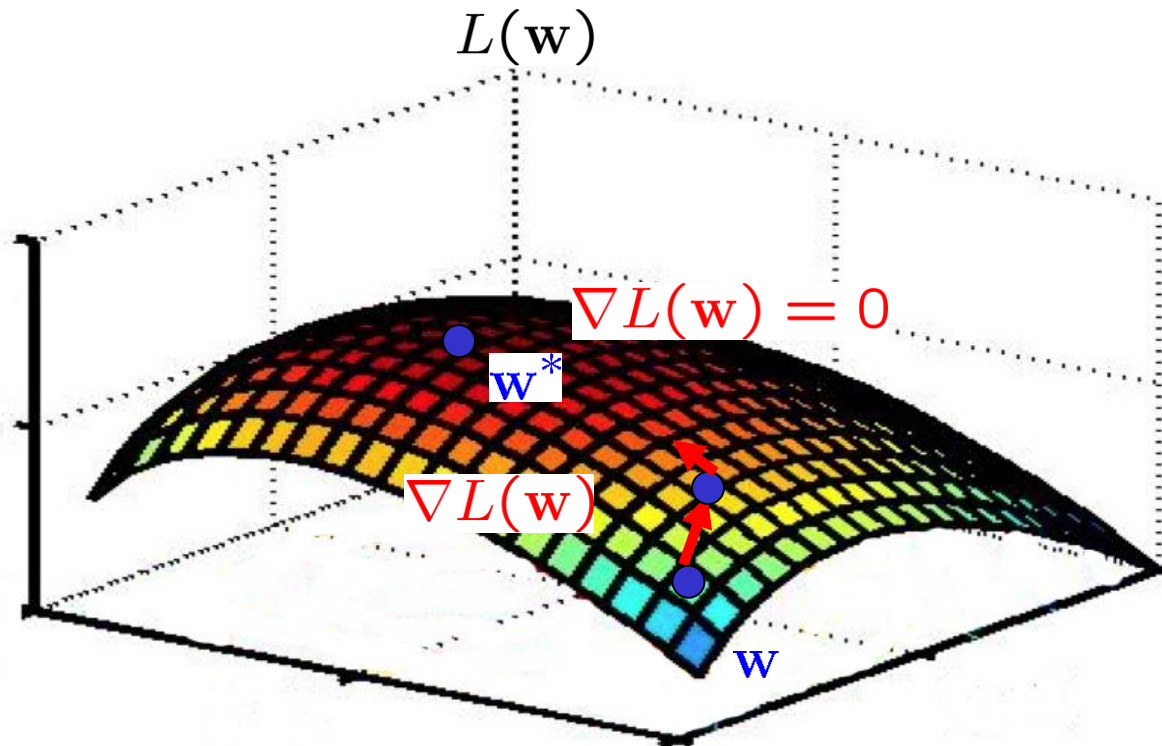
$$E_p(\phi_i) = E_{\tilde{p}}(\phi_i), \forall i$$

# Unconstrained Optimization



- Basic idea: move uphill from current guess
- Gradient ascent / descent follows the gradient incrementally
- At local optimum, derivative vector is zero
- Will converge if step sizes are small enough, but not efficient
- All we need is to be able to evaluate the function and its derivative

# Unconstrained Optimization



- For convex functions, a local optimum will be global
- Basic gradient ascent isn't very efficient, but there are simple enhancements which take into account previous gradients: conjugate gradient, L-BFGS
- There are special-purpose optimization techniques for maxent, like iterative scaling, but they aren't better

# Gradient Descent & Large Training Data

repeat

$$w^{(t+1)} \leftarrow w^{(t)} + \eta \frac{\partial L}{\partial w}$$

$$w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \left[ \frac{1}{n} \sum_{i=1}^N \left( \phi_j(x_i, y_i) - \sum_y p(y | x_i, w^{(t)}) \phi_j(x_i, y) \right) - \lambda w_j^{(t)} \right]$$

until convergence



Prohibitive for large datasets

# Stochastic Gradient Descent

repeat

$$\cancel{w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta \left[ \frac{1}{N} \sum_{i=1}^N \left( \phi_j(x_i, y_i) - \sum_y p(y | x_i, w^{(t)}) \phi_j(x_i, y) \right) - \lambda w_j^{(t)} \right]}$$

until convergence

Use gradient at current point as approx. for avg gradient!

repeat

$$w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta^{(t)} \left[ \phi_j(x_i, y_i) - \sum_y p(y | x_i, w^{(t)}) \phi_j(x_i, y) - \lambda w_j^{(t)} \right]$$

until convergence

Reduce learning rate slowly (e.g., as  $\eta/t$ )



# L1 and L2 Regularization

## L2 Regularization for Log-linear models

- Instead, we worry about large feature weights
- Add a regularization term to the likelihood to push weights towards zero

$$L(w) = \sum_{i=1}^n \log p(y_i | x_i; w) - \frac{\lambda}{2} \|w\|^2$$

Regularization Constant

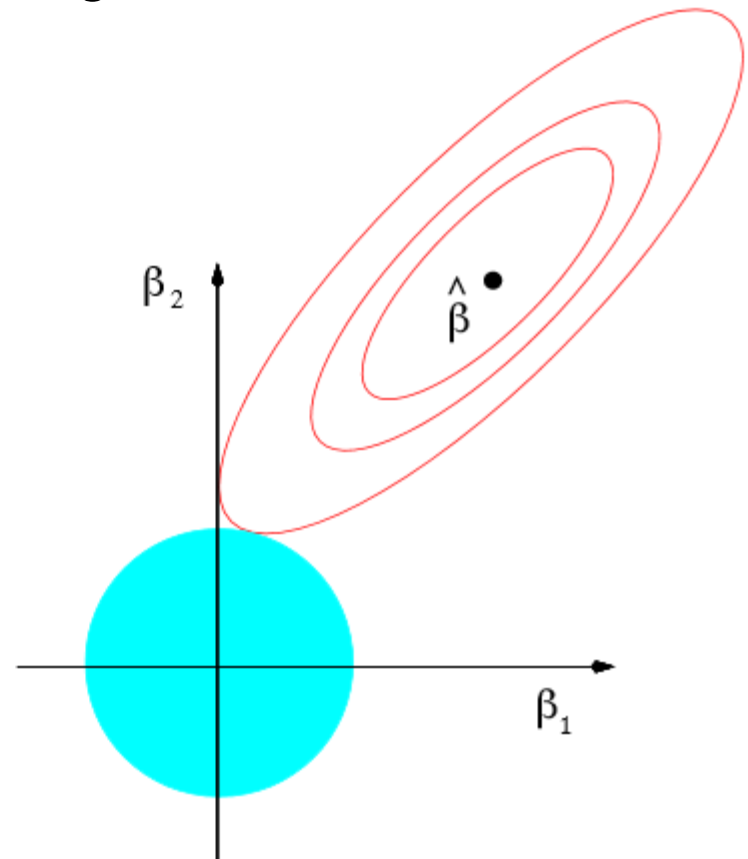
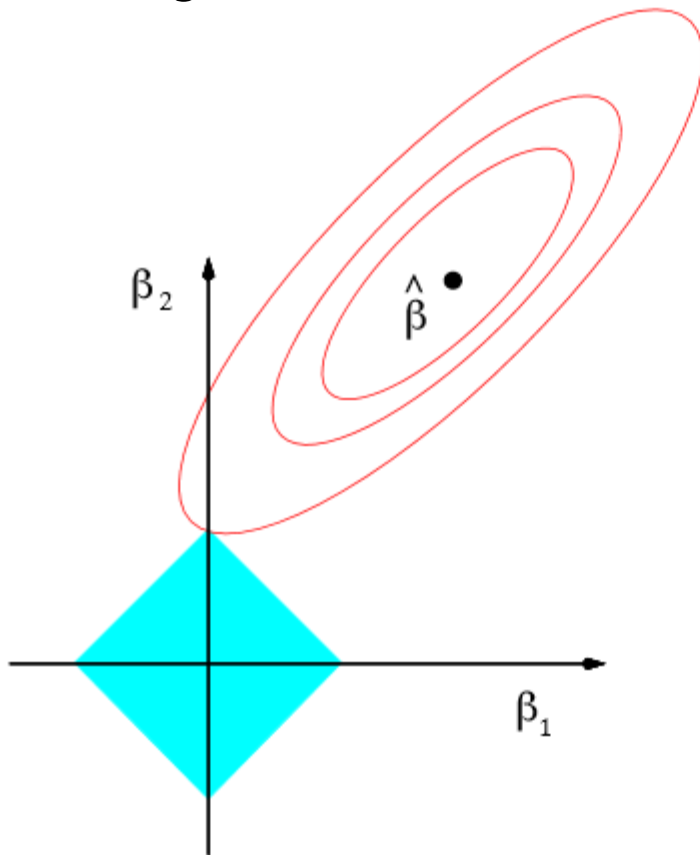
## L1 Regularization for Log-linear models

- Instead, we worry about number of active features
- Add a regularization term to the likelihood to push weights to zero

$$L(w) = \sum_{i=1}^n \log p(y_i | x_i; w) - \lambda \|w\|$$

# L1 vs L2

- Optimizing L1 harder
  - Discontinuous objective function
  - Subgradient descent versus gradient descent



# How to pick weights?

- Goal: choose “best” vector  $w$  given training data
  - For now, we mean “best for classification”
- The ideal: the weights which have greatest test set accuracy / F1 / whatever
  - But, don't have the test set
  - Must compute weights from training set
- Maybe we want weights which give best training set accuracy?
  - Hard discontinuous optimization problem
  - May not (does not) generalize to test set
  - Easy to overfit

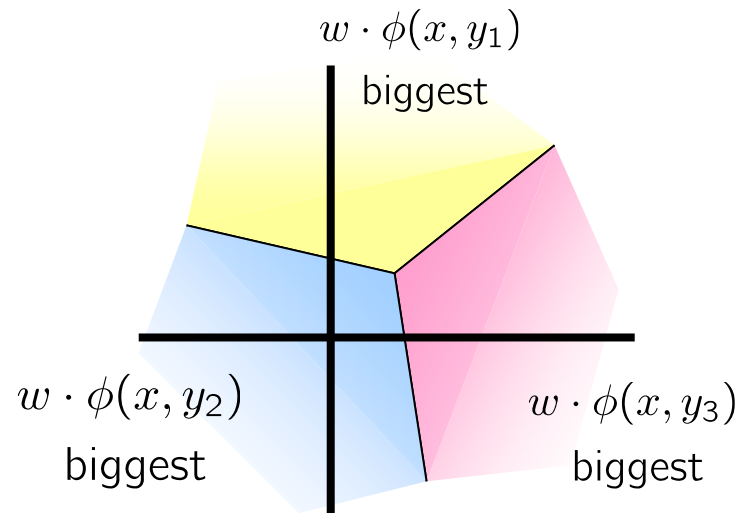
# Learning Classifiers

- Two probabilistic approaches to predicting classes  $y^*$ 
  - **Joint:** work with a *joint* probabilistic model of the data, weights are (often) local conditional probabilities
    - E.g., represent  $p(y,x)$  as Naïve Bayes model, compute  $y^* = \operatorname{argmax}_y p(y,x)$
  - **Conditional:** work with *conditional* probability  $p(y|x)$ 
    - We can then directly compute  $y^* = \operatorname{argmax}_y p(y|x)$  Can develop *feature rich* models for  $p(y|x)$ .
- But, why estimate a distribution at all?
  - Linear predictor:  $y^* = \operatorname{argmax}_y w \cdot \phi(x,y)$
  - Perceptron algorithm
    - Online
    - Error driven
    - Simple, additive updates

# Multiclass Perceptron Decision Rule

- Compare all possible outputs

- Highest score wins
- Boundaries are more complex
- Harder to visualize



$$y^* = \arg \max_y w \cdot \phi(x, y)$$

# Linear Models: Perceptron

- The perceptron algorithm

- Iteratively processes the training set, reacting to training errors
- Can be thought of as trying to drive down training error

- The (online) perceptron algorithm:

- Start with zero weights
- Visit training instances  $(x_i, y_i)$  one by one
  - Make a prediction

$$y^* = \arg \max_y w \cdot \phi(x_i, y)$$

- If correct ( $y^* == y_i$ ): no change, goto next example!
- If wrong: adjust weights

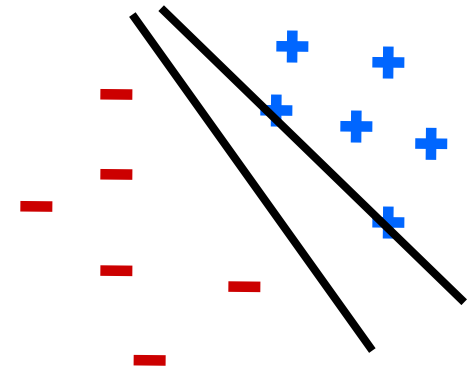
$$w = w + \eta \{ \phi(x_i, y_i) - \phi(x_i, y^*) \}$$

# Properties of Perceptrons

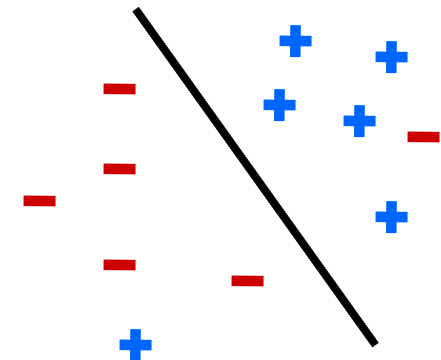
- **Separability:** some parameters get the training set perfectly correct
- **Convergence:** if the training is separable, perceptron will eventually converge
- **Mistake Bound:** the maximum number of mistakes (binary case) related to the *margin* or degree of separability

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

Separable

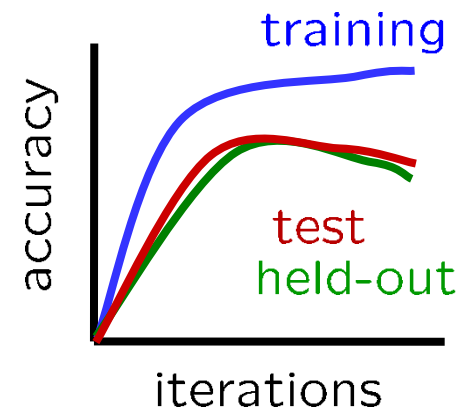
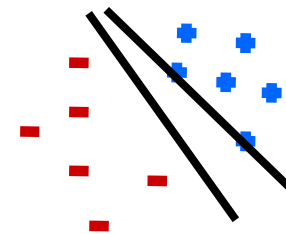
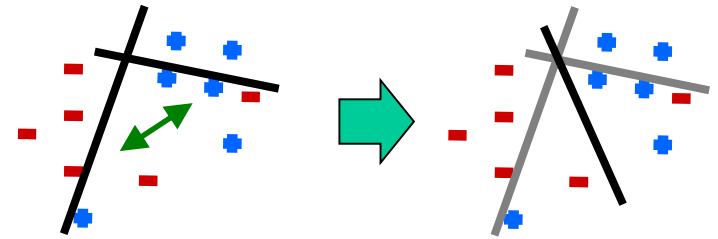


Non-Separable



# Problems with the Perceptron

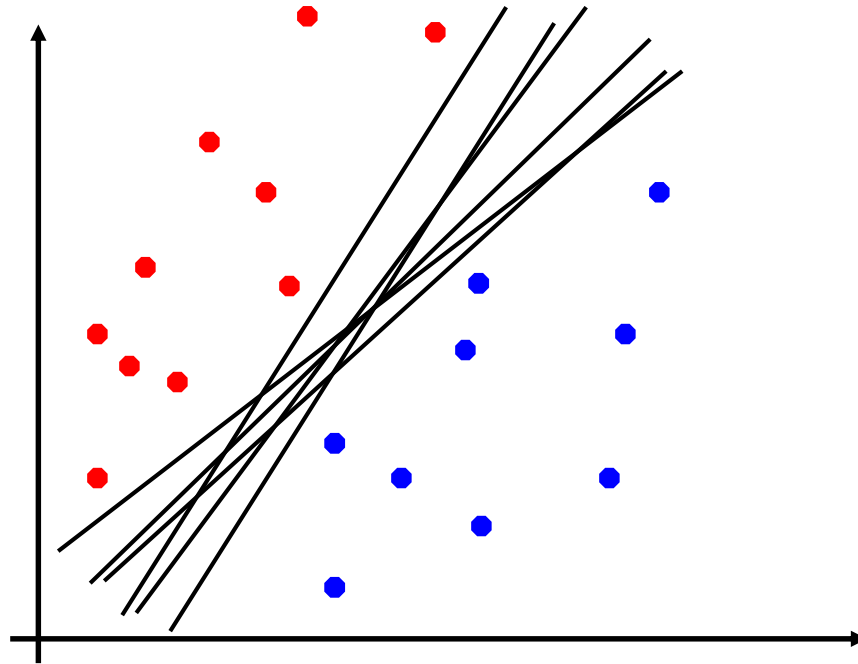
- **Noise:** if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)
- **Mediocre generalization:** finds a “barely” separating solution
- **Overtraining:** test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting





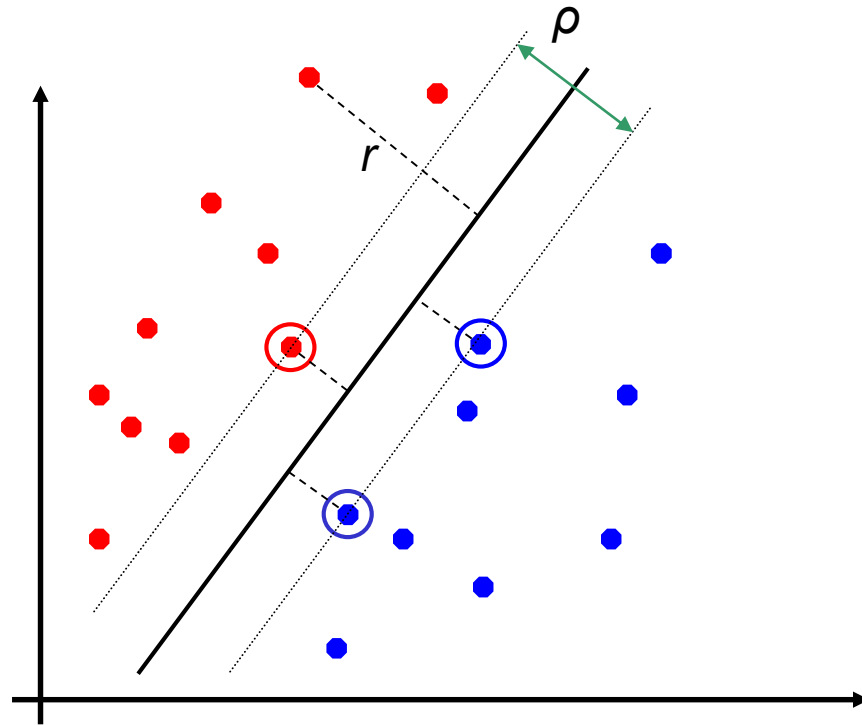
# Perceptrons look for any separator

- Which of the linear separators is optimal?



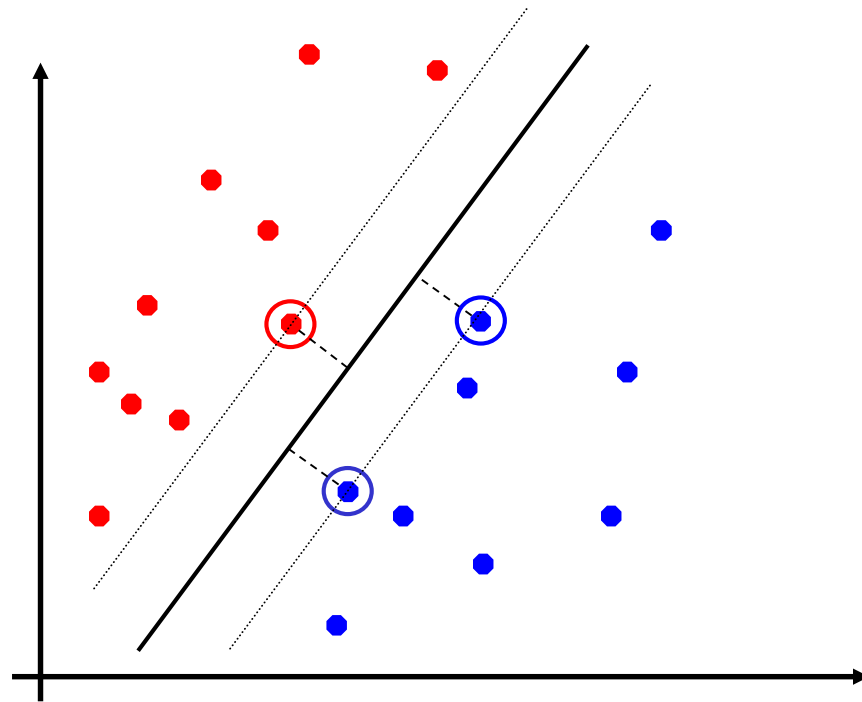
# Classification Margin

- Distance from example  $\mathbf{x}_i$  to the separator is  $r = \frac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are **support vectors**.
- **Margin**  $\rho$  of the separator is the distance between support vectors.



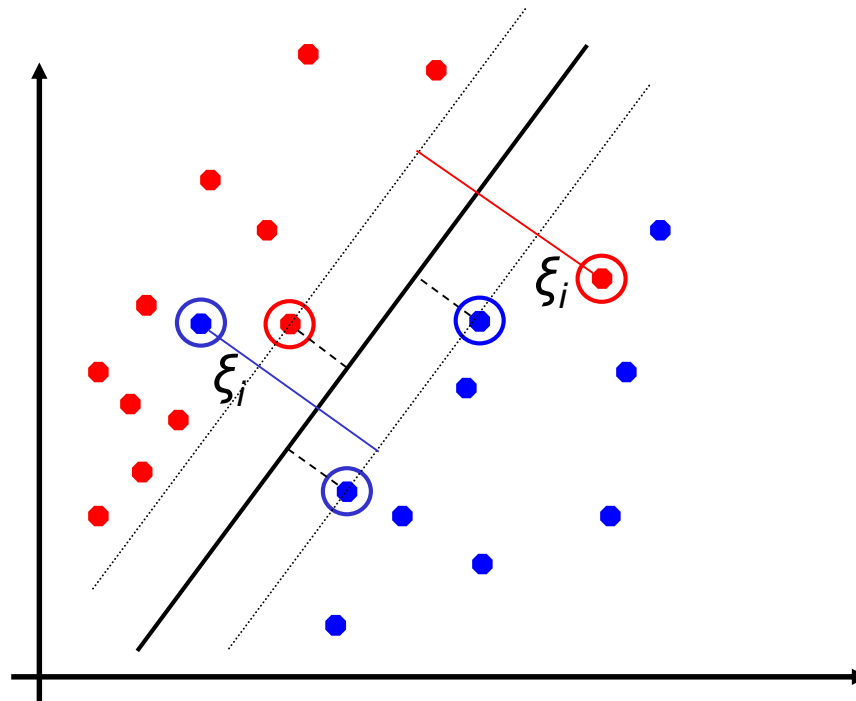
# Maximum Margin Classification

- Maximizing the margin is good according to intuition and PAC theory.
- Implies that only support vectors matter; other training examples are ignorable.



# Soft Margin Classification

- What if the training set is not linearly separable?
- *Slack variables*  $\xi_i$  can be added to allow misclassification of difficult or noisy examples, resulting margin called *soft*.



# Linear SVMs: Overview

- The classifier is a *separating hyperplane*.
- Most “important” training points are support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points  $\mathbf{x}_i$  are support vectors with non-zero Lagrangian multipliers  $\alpha_i$ .
- Both in the dual formulation of the problem and in the solution training points appear only inside inner products:

Find  $\alpha_1 \dots \alpha_N$  such that

$Q(\boldsymbol{\alpha}) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$  is maximized and

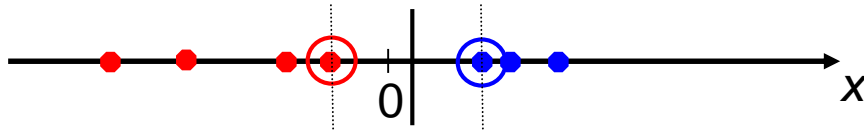
(1)  $\sum \alpha_i y_i = 0$

(2)  $0 \leq \alpha_i \leq C$  for all  $\alpha_i$

$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$$

# Non-linear SVMs

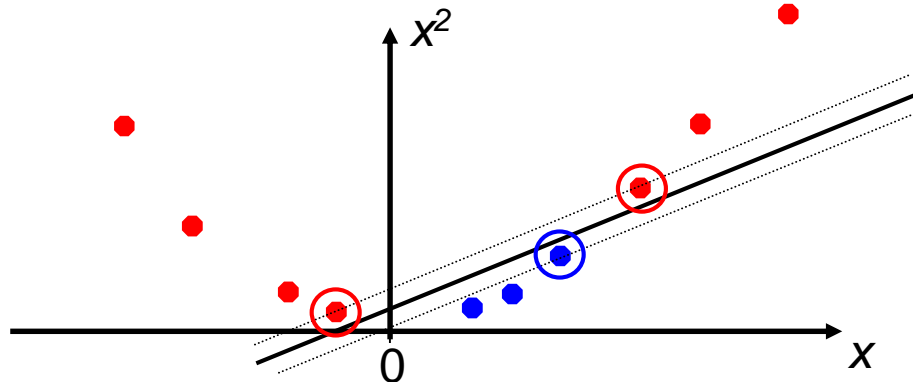
- Datasets that are linearly separable with some noise work out great:



- But what are we going to do if the dataset is just too hard?

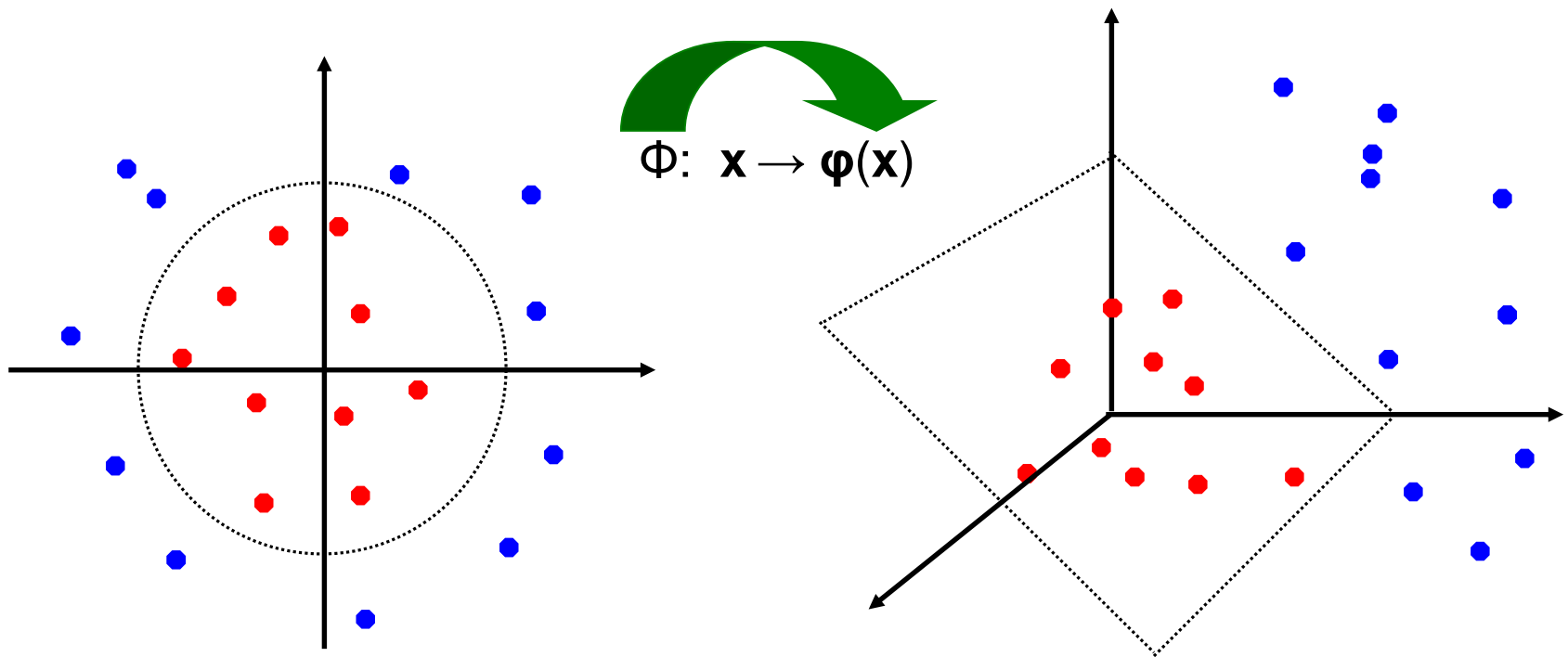


- How about... mapping data to a higher-dimensional space:



# Non-linear SVMs: Feature spaces

- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



# Examples of Kernel Functions

- Linear:  $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$ 
  - Mapping  $\Phi: \mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$ , where  $\boldsymbol{\varphi}(\mathbf{x})$  is  $\mathbf{x}$  itself
- Polynomial of power  $p$ :  $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$ 
  - Mapping  $\Phi: \mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$ , where  $\boldsymbol{\varphi}(\mathbf{x})$  has  $\binom{d+p}{p}$  dimensions
- Gaussian (radial-basis function):  $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}}$ 
  - Mapping  $\Phi: \mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$ , where  $\boldsymbol{\varphi}(\mathbf{x})$  is *infinite-dimensional*: every point is mapped to a *function* (a Gaussian); combination of functions for support vectors is the separator.
- Higher-dimensional space still has *intrinsic* dimensionality  $d$  (the mapping is not *onto*), but linear separators in it correspond to *non-linear* separators in original space.



# SVM applications

- SVMs were originally proposed by Boser, Guyon and Vapnik in 1992 and gained increasing popularity in late 1990s.
- SVMs are currently among the best performers for a number of classification tasks ranging from text to genomic data.
- SVMs can be applied to complex data types beyond feature vectors (e.g. graphs, sequences, relational data) by designing kernel functions for such data.
- SVM techniques have been extended to a number of tasks such as regression [Vapnik *et al.* '97], principal component analysis [Schölkopf *et al.* '99], etc.
- Most popular optimization algorithms for SVMs use *decomposition* to hill-climb over a subset of  $\alpha_i$ 's at a time, e.g. SMO [Platt '99] and [Joachims '99]
- Tuning SVMs remains a black art: selecting a specific kernel and parameters is usually done in a try-and-see manner.

# Performance Comparison

					linear SVM		rbf-SVM
	NB	Rocchio	Dec. Trees	kNN	C=0.5	C=1	
earn	96.0	96.1	96.1	97.8	98.0	<b>98.2</b>	98.1
acq	90.7	92.1	85.3	91.8	95.5	<b>95.6</b>	94.7
money-fx	59.6	67.6	69.4	75.4	<b>78.8</b>	78.5	74.3
grain	69.8	79.5	89.1	82.6	91.9	93.1	<b>93.4</b>
crude	81.2	81.5	75.5	85.8	89.4	<b>89.4</b>	88.7
trade	52.2	77.4	59.2	77.9	79.2	<b>79.2</b>	76.6
interest	57.6	72.5	49.1	<b>76.7</b>	75.6	74.8	69.1
ship	80.9	83.1	80.9	79.8	<b>87.4</b>	86.5	85.8
wheat	63.4	79.4	85.5	72.9	86.6	<b>86.8</b>	82.4
corn	45.2	62.2	87.7	71.4	87.5	<b>87.8</b>	84.6
microavg.	<b>72.3</b>	<b>79.9</b>	<b>79.4</b>	<b>82.6</b>	<b>86.7</b>	<b>87.5</b>	<b>86.4</b>

SVM classifier break-even F from ([Joachims, 2002a](#), p. 114). Results are shown for the 10 largest categories and for microaveraged performance over all 90 categories on the Reuters-21578 data set.

# Choosing a classifier

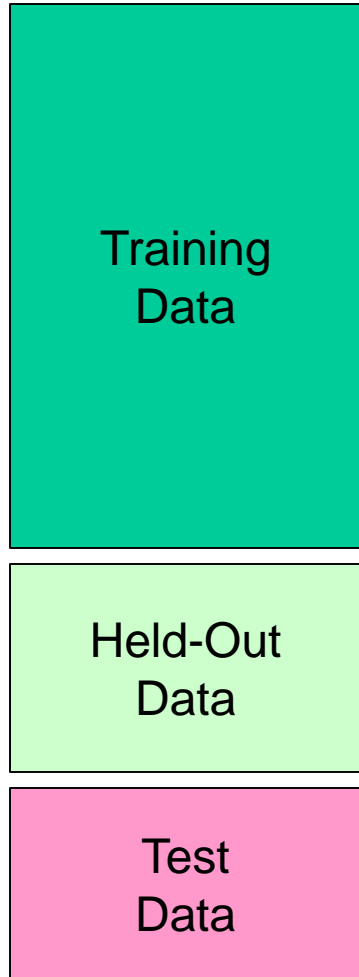
Technique	Train time	Test time	“Accuracy”	Interpre-tability	Bias-Variance	Data Complexity
Naïve Bayes	$ W  +  C   V $	$ C  *  V_d $	Medium-low	Medium	High-bias	Low
k-NN	$ W $	$ V  *  V_d $	Medium	Low	High-variance	High
SVM	$ C   D ^3  V _{ave}$	$ C  *  V_d $	High	Low	Mixed	Medium-low
Neural Nets	?	$ C  *  V_d $	High	Low	High-variance	High
Log-linear	?	$ C  *  V_d $	High	Medium	High-variance/ mixed	Medium

“Accuracy” – reputation for accuracy in experimental settings.

Note that it is impossible to say beforehand which classifier will be most accurate on any given problem.

$C$  = set of classes.  $W$  = bag of training tokens.  $V$  = set of training types.  $D$  = set of train docs.  $V_d$  = types in test document  $d$ .  $V_{ave}$  = average number of types per doc in training.

# Summary: Three Views of Classification



- Joint prob. Models (Naïve Bayes):
  - Parameters from data statistics
  - Parameters: probabilistic interpretation
  - Training: one pass through the data
- Conditional prob. Models (Log-linear):
  - Parameters from gradient ascent
  - Parameters: linear, probabilistic model, and discriminative
  - Training: gradient ascent (usually batch), regularize to stop overfitting
- Discriminative non-prob. models:
  - Perceptrons: Parameters from reactions to mistakes; linear
  - SVMs: parameters from gradient ascent; can be any shape depending on kernel; regularize to stop overfitting

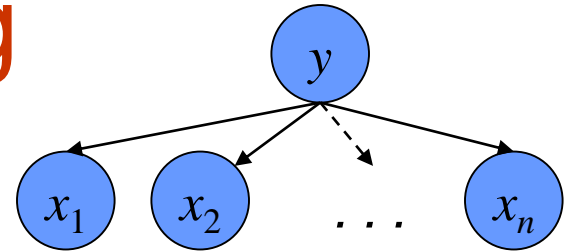
# Clustering vs. Classification

- **Classification:** we specify which pattern we want, features uncorrelated with that pattern are idle

$P(w \text{sports})$	$P(w \text{politics})$	$P(w \text{headline})$	$P(w \text{story})$
the 0.1	the 0.1	the 0.05	the 0.1
game 0.02	game 0.005	game 0.01	game 0.01
win 0.02	win 0.01	win 0.01	win 0.01

- **Clustering:** the clustering procedure locks on to whichever pattern is most salient, statistically
  - $P(\text{content words} \mid \text{class})$  will learn topics
  - $P(\text{length, function words} \mid \text{class})$  will learn style
  - $P(\text{characters} \mid \text{class})$  will learn “language”

# Model-Based Clustering



- Clustering with probabilistic models:

Unobserved (Y)

Observed (X)

??	LONDON -- Soccer team wins match
??	NEW YORK – Stocks close up 3%
??	Investing in the stock market has ...
??	The first game of the world series ...

Build a model of the domain:  $P(x, y, \theta)$

Often: find  $\theta$  to maximize:  $P(x|\theta) = \sum_y P(x, y|\theta)$

- Problem 2: The relationship between the structure of your model and the kinds of patterns it will detect is complex.

# Chicken & Egg Problem

- If we knew the cluster ids/
  - It would be easy to learn CPT/centroids
- If we knew the CPT/centroids
  - Then it'd be easy to infer the (probability of) cluster id
- But we do not know either!

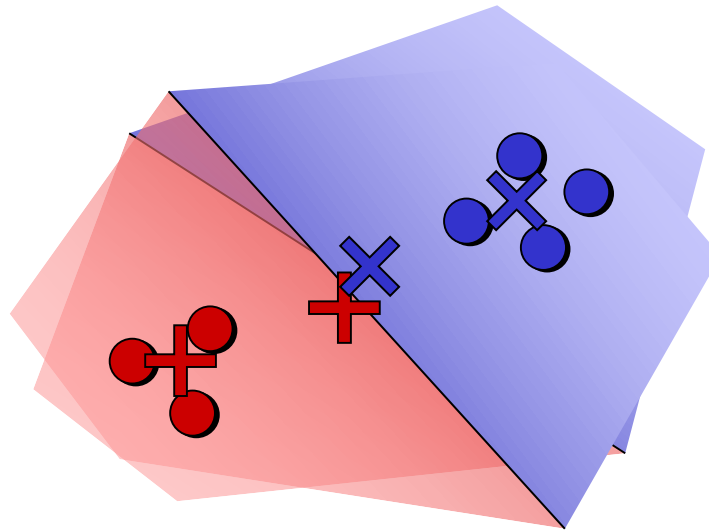
# Learning Models with EM

- Hard EM:  
alternate between

E-step: Find best “completions”  $Y$  for fixed  $\theta$

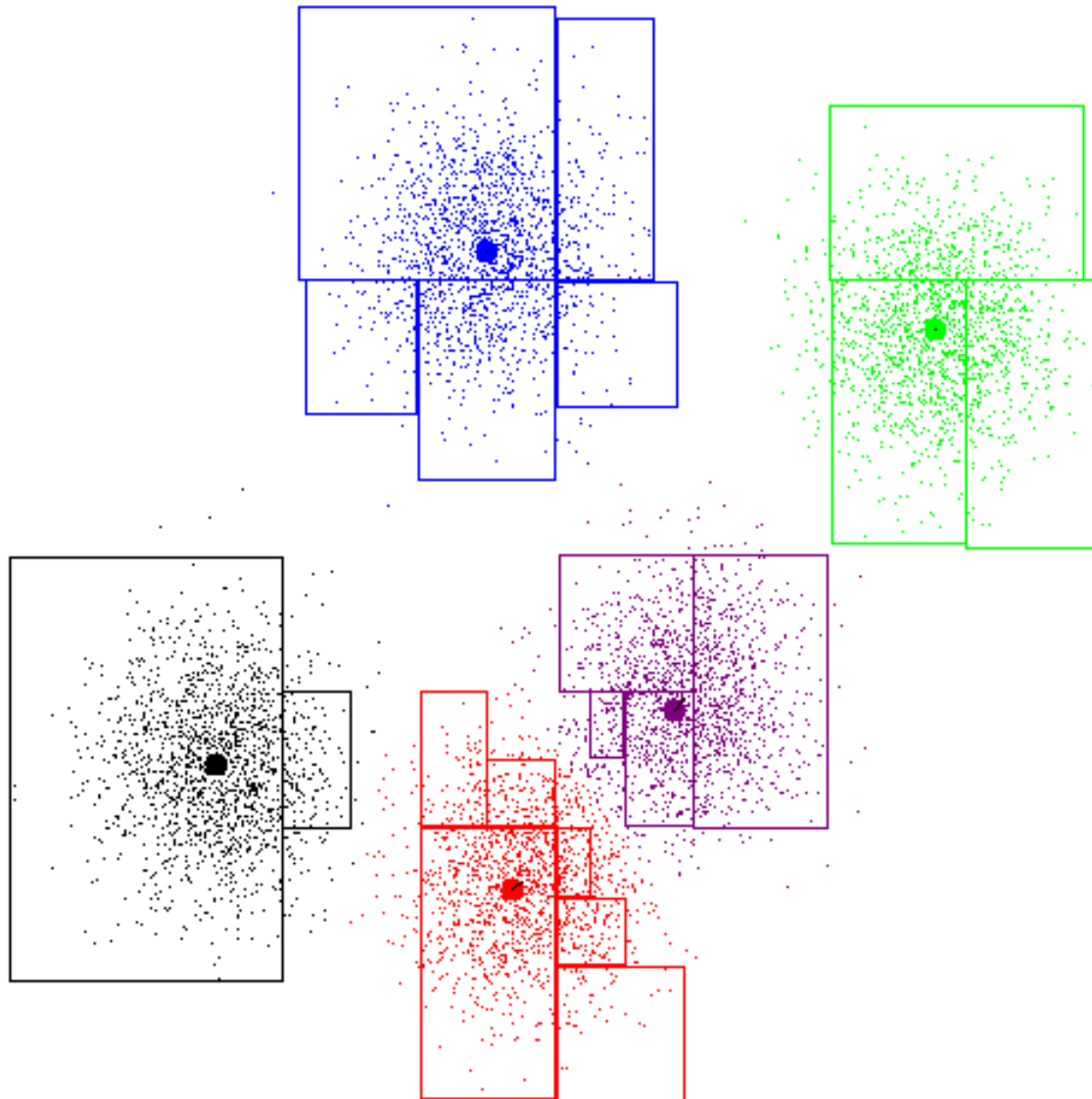
M-step: Find best parameters  $\theta$  for fixed  $Y$

- Example: K-Means





# K-Means Example



# Hard EM for Naïve-Bayes

- Procedure: (1) we calculate best completions:

$$y^* = \arg \max_y P(y) \prod_i P(x_i|y)$$

- (2) compute relevant counts from the completed data:

$$c(w, y) = \sum_{x \in D} \sum_i [1(x_i = w, y^* = y)]$$

- (3) compute new parameters from these counts (divide)
- (4) repeat until convergence

# (Soft) EM for Naïve-Bayes

- Procedure: (1) calculate posteriors (soft completions):

$$P(y|x) = \frac{P(y) \prod_i P(x_i|y)}{\sum_{y'} P(y') \prod_i P(x_i|y')}$$

- (2) compute expected counts under those posteriors:

$$c(w, y) = \sum_{x \in D} P(y|x) \sum_i [1(x_i = w, y)]$$

- (3) compute new parameters from these counts (divide)
- (4) repeat until convergence
- Can also do this when some docs are labeled

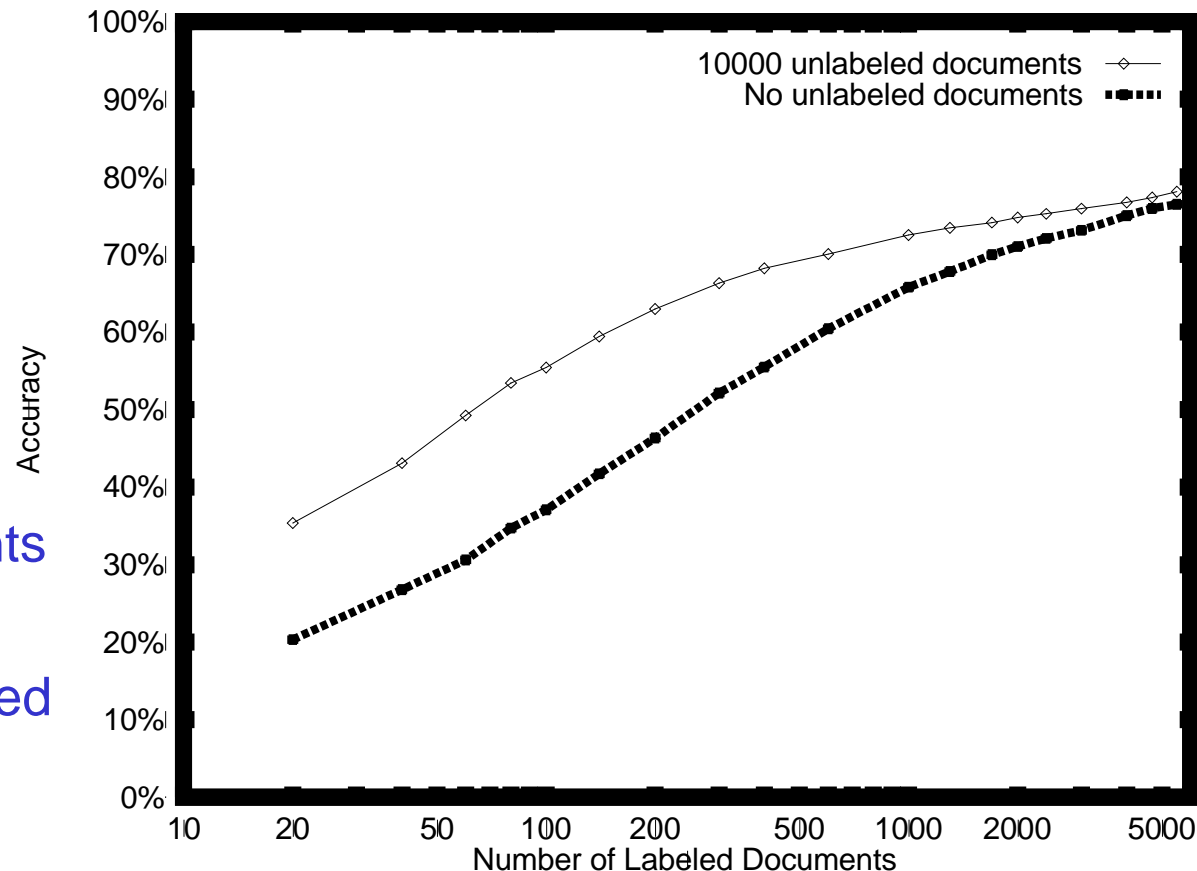
# EM in General

- We'll use EM over and over again to fill in missing data, e.g. we want  $P(x,y)$  but our training data has only  $x$ s (no  $y$ s labeled)
  - Convenience Scenario: we want  $P(x)$ , including  $y$  just makes the model simpler (e.g. mixing weights for language models)
  - Induction Scenario: we actually want to know  $y$  (e.g. clustering)
  - NLP differs from much of statistics / machine learning in that we often want to interpret or use the induced variables (which is tricky at best)
- General approach: alternately update  $y$  and  $\theta$ 
  - E-step: compute posteriors  $P(y|x,\theta)$ 
    - This means scoring all completions with the current parameters
    - Usually, we do this implicitly with dynamic programming
  - M-step: fit  $\theta$  to these completions
    - This is usually the easy part – treat the completions as (fractional) complete data
  - Initialization: start with some noisy labelings and the noise adjusts into patterns based on the data and the model
  - We'll see lots of examples in this course
- EM is only locally optimal (why?)

# EM for Semi-supervised Learning

[Nigam, McCallum, Mitchel, 2006]

- Define data log likelihood to be  $L(y,x) + L(x)$ , computed on labeled and unlabeled data. Find parameters that maximize the total likelihood.
- Paper also presents a number of other fancier models where the unlabeled data helps more.



**Figure 3.1** Classification accuracy on the 20 Newsgroups data set, both with and without 10,000 unlabeled documents. With small amounts of training data, using EM yields more accurate classifiers. With large amounts of labeled training data, accurate parameter estimates can be obtained without the use of unlabeled data, and classification accuracies of the two methods begin to converge.