

# **CS/INFS 780**

# **Data Mining for Multimedia Data**

## **Text Classification**



Dr. Jessica Lin

The slides are from Christopher Manning and Prabhakar Raghavan from Stanford University

# Standing queries

- The path from IR to text classification:
  - You have an information need to monitor, say:
    - iPhone 5s
  - You want to rerun an appropriate query periodically to find new news items on this topic
  - You will be sent new documents that are found
- Such queries are called *standing queries*
  - Long used by “information professionals”
  - A modern mass instantiation is **Google Alerts**
- Standing queries are (hand-written) text classifiers

# Spam filtering: Another text classification task

From: "" <takworld@hotmail.com>

Subject: real estate is the only way... gem oalvgkay

Anyone can buy real estate with no money down

Stop paying rent TODAY !

There is no need to spend hundreds or even thousands for similar courses

I am 22 years old and I have already purchased 6 properties using the methods outlined in this truly INCREDIBLE ebook.

Change your life NOW !

=====

Click Below to order:

<http://www.wholesaledaily.com/sales/nmd.htm>

=====

# Text classification

## ■ Today:

### ★ Introduction to Text Classification

✓ Also widely known as “text categorization”. Same thing.

### ★ Naïve Bayes text classification

### ★ Text classification for vector space models

# Categorization/Classification

## ■ Given:

- ★ A description of an instance,  $d \in X$ 
  - ✓  $X$  is the *instance language* or *instance space*.
    - Issue: how to represent text documents.
    - Usually some type of high-dimensional space

- ★ A fixed set of classes:

$$C = \{c_1, c_2, \dots, c_J\}$$

## ■ Determine:

- ★ The category of  $d$ :  $\gamma(d) \in C$ , where  $\gamma(d)$  is a *classification function* whose domain is  $X$  and whose range is  $C$ .
  - ✓ We want to know how to build classification functions (“classifiers”).

# Supervised Classification

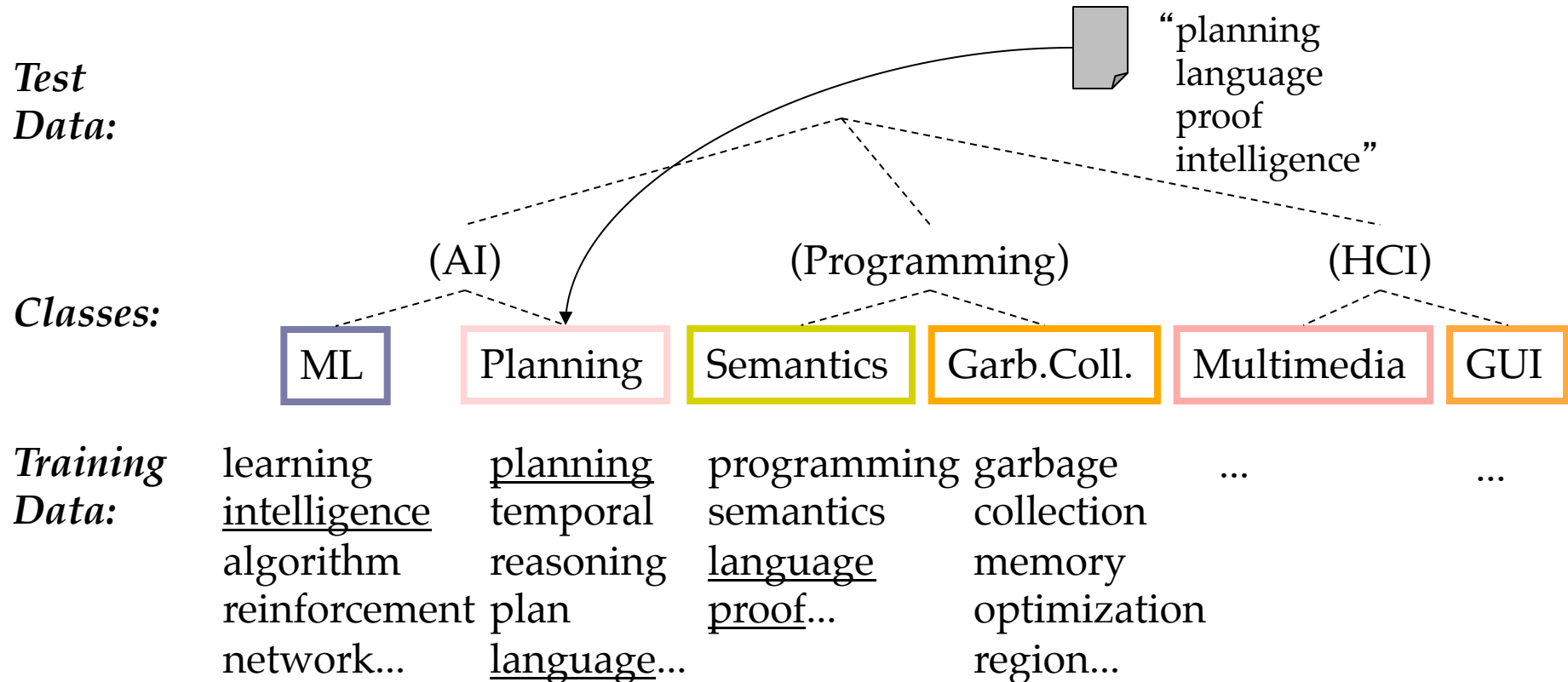
## ■ Given:

- ★ A description of an instance,  $d \in X$ 
  - ✓  $X$  is the *instance language* or *instance space*.
- ★ A fixed set of classes:  
 $C = \{c_1, c_2, \dots, c_J\}$
- ★ A training set  $D$  of labeled documents with each labeled document  $\langle d, c \rangle \in X \times C$

## ■ Determine:

- ★ A learning method or algorithm which will enable us to learn a classifier  $\gamma: X \rightarrow C$
- ★ For a test document  $d$ , we assign it the class  $\gamma(d) \in C$

# Document Classification



# More Text Classification Examples

## Many search engine functionalities use classification

- Assigning labels to documents or web-pages:
- Labels are most often topics such as Yahoo-categories
  - *"finance," "sports," "news>world>asia>business"*
- Labels may be genres
  - *"editorials" "movie-reviews" "news"*
- Labels may be opinion on a person/product
  - *"like", "hate", "neutral"*
- Labels may be domain-specific
  - *"interesting-to-me" : "not-interesting-to-me"*
  - *"contains adult language" : "doesn't"*
  - *language identification: English, French, Chinese, ...*
  - *search vertical: about Linux versus not*
  - *"link spam" : "not link spam"*



# Classification Methods (1)

## ■ Manual classification

- ★ Used by the original Yahoo! Directory
- ★ Looksmart, about.com, ODP, PubMed
- ★ Very accurate when job is done by experts
- ★ Consistent when the problem size and team is small
- ★ Difficult and expensive to scale
  - ✓ Means we need automatic classification methods for big problems

# Classification Methods (2)

## ■ Automatic document classification

### ★ Hand-coded rule-based systems

- ✓ One technique used by CS dept's spam filter, Reuters, CIA, etc.
- ✓ It's what Google Alerts is doing
  - Widely deployed in government and enterprise
- ✓ Companies provide “IDE” for writing such rules
  - e.g., assign category if document contains a given boolean combination of words
- ✓ Standing queries: Commercial systems have complex query languages (everything in IR query languages +score accumulators)
- ✓ Accuracy is often very high if a rule has been carefully refined over time by a subject expert
- ✓ Building and maintaining these rules is expensive

# Classification Methods (3)

- Supervised learning of a document-label assignment function
  - ★ Many systems partly rely on machine learning (Autonomy, Microsoft, Enkata, Yahoo!, ...)
    - ✓ k-Nearest Neighbors (simple, powerful)
    - ✓ Naive Bayes (simple, common method)
    - ✓ Support-vector machines (new, more powerful)
    - ✓ ... plus many other methods
    - ✓ No free lunch: requires hand-classified training data
    - ✓ But data can be built up (and refined) by amateurs
- Many commercial systems use a mixture of methods

# Bayesian Methods

- Learning and classification methods based on probability theory.
- Bayes theorem plays a critical role in probabilistic learning and classification.
- Builds a *generative model* that approximates how data is produced
- Uses *prior* probability of each category given no information about an item.
- Categorization produces a *posterior* probability distribution over the possible categories given a description of an item.

# Naïve Bayes Classifier



Thomas Bayes  
1702 - 1761

We will start off with a visual intuition, before looking at the math...

# Bayes Classifiers

- Bayesian classifiers use **Bayes theorem**, which says

$$p(c_j | d) = \frac{p(d | c_j) p(c_j)}{p(d)}$$

- $p(c_j | d)$  = probability of instance  $d$  being in class  $c_j$ ,  
This is what we are trying to compute
- $p(d | c_j)$  = probability of generating instance  $d$  given class  $c_j$ ,  
We can imagine that being in class  $c_j$ , causes you to have feature  $d$  with some probability
- $p(c_j)$  = probability of occurrence of class  $c_j$ ,  
This is just how frequent the class  $c_j$ , is in our database
- $p(d)$  = probability of instance  $d$  occurring

Assume that we have two classes

$C_1 = \text{male}$ , and  $C_2 = \text{female}$ .

We have a person whose sex we do not know, say “*drew*” or *d*.

Classifying *drew* as male or female is equivalent to asking is it more probable that *drew* is **male** or **female**, i.e which is greater  $p(\text{male} \mid \text{drew})$  or  $p(\text{female} \mid \text{drew})$

(Note: “Drew can be a male or female name”)



Drew Barrymore



Drew Carey

What is the probability of being called “*drew*” given that you are a **male**?

$$p(\text{male} \mid \text{drew}) = \frac{p(\text{drew} \mid \text{male}) p(\text{male})}{p(\text{drew})}$$

What is the probability of being a **male**?

What is the probability of being named “*drew*”? (actually irrelevant, since it is the same for all



**Officer Drew**

This is Officer Drew. Is Officer Drew a **Male** or **Female**?

Luckily, we have a small database with names and sex.

We can use it to apply Bayes rule...

$$p(c_j | d) = \frac{p(d | c_j) p(c_j)}{p(d)}$$

Name	Sex
Drew	Male
Claudia	Female
Drew	Female
Drew	Female
Alberto	Male
Karin	Female
Nina	Female
Sergio	Male





**Officer Drew**

$$p(c_j | d) = \frac{p(d | c_j) p(c_j)}{p(d)}$$

Name	Sex
Drew	Male
Claudia	Female
Drew	Female
Drew	Female
Alberto	Male
Karin	Female
Nina	Female
Sergio	Male

$$p(\text{male} | \text{drew}) = \frac{1/3 * 3/8}{3/8} = \frac{0.125}{3/8}$$

$$p(\text{female} | \text{drew}) = \frac{2/5 * 5/8}{3/8} = \frac{0.250}{3/8}$$

Officer Drew is more likely to be a **Female**.

So far we have only considered Bayes Classification when we have one attribute (the “*antennae length*”, or the “*name*”). But we may have many features.  
How do we use all the features?

$$p(c_j | d) = \frac{p(d | c_j) p(c_j)}{p(d)}$$

Name	Over 170cm	Eye	Hair length	Sex
Drew	No	Blue	Short	Male
Claudia	Yes	Brown	Long	Female
Drew	No	Blue	Long	Female
Drew	No	Blue	Long	Female
Alberto	Yes	Brown	Short	Male
Karin	No	Blue	Long	Female
Nina	Yes	Brown	Short	Female
Sergio	Yes	Blue	Long	Male

- To simplify the task, **naïve Bayesian classifiers** assume attributes have independent distributions, and thereby estimate

$$p(d|c_j) = p(d_1|c_j) * p(d_2|c_j) * \dots * p(d_n|c_j)$$

↑  
The probability of class  $c_j$   
generating instance  $d$ ,  
equals....

↑  
The probability of class  
 $c_j$  generating the  
observed value for  
feature 1, multiplied by..

↑  
The probability of class  
 $c_j$  generating the  
observed value for  
feature 2, multiplied by..

- To simplify the task, **naïve Bayesian classifiers** assume attributes have independent distributions, and thereby estimate

$$p(d|c_j) = p(d_1|c_j) * p(d_2|c_j) * \dots * p(d_n|c_j)$$

$$p(\text{officer drew}|c_j) = p(\text{over\_170}_{\text{cm}} = \text{yes}|c_j) * p(\text{eye} = \text{blue}|c_j) * \dots$$



Officer Drew  
is blue-eyed,  
over 170<sub>cm</sub>  
tall, and has  
long hair

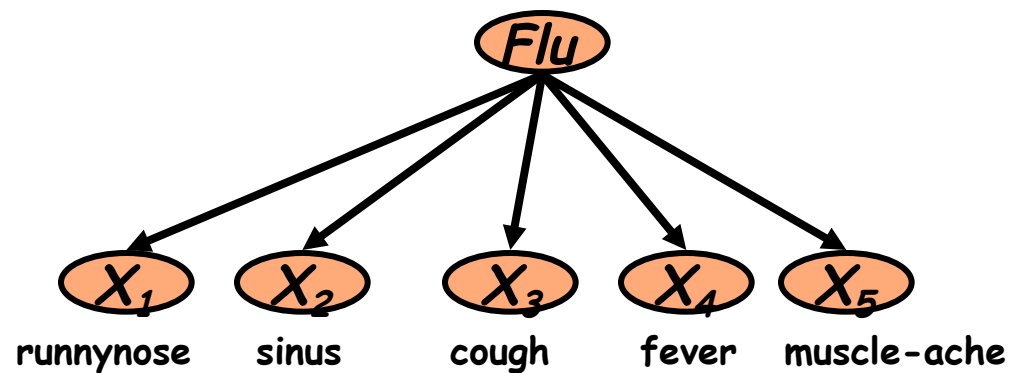
$$p(\text{officer drew} | \text{Female}) = 2/5 * 3/5 * \dots$$

$$p(\text{officer drew} | \text{Male}) = 2/3 * 2/3 * \dots$$

# Advantages/Disadvantages of Naïve Bayes

- Advantages:
  - Fast to train (single scan). Fast to classify
  - Not sensitive to irrelevant features
  - Handles real and discrete data
  - Handles streaming data well
- Disadvantages:
  - Assumes independence of features

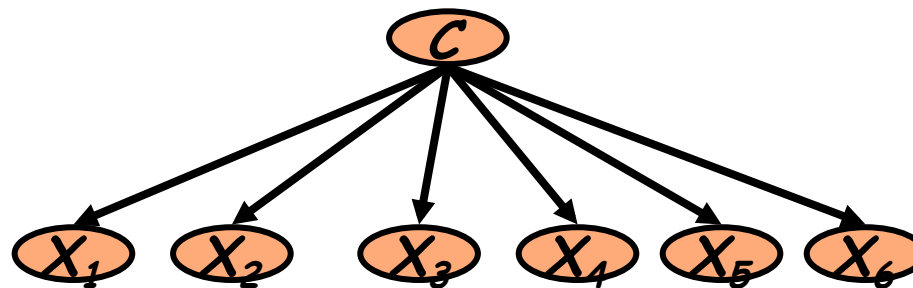
# The Naïve Bayes Classifier for Text Classification



- **Conditional Independence Assumption:** features detect term presence and are independent of each other given the class:

$$P(X_1, \dots, X_5 | C) = P(X_1 | C) \cdot P(X_2 | C) \cdot \dots \cdot P(X_5 | C)$$

# Learning the Model



- First attempt: maximum likelihood estimates
  - ★ simply use the frequencies in the data

$$\hat{P}(c_j) = \frac{N(C = c_j)}{N}$$

$$\hat{P}(x_i | c_j) = \frac{N(X_i = x_i, C = c_j)}{N(C = c_j)}$$

# Using Multinomial Naive Bayes Classifiers to Classify Text: Basic method

- Attributes are text positions, values are words.

$$\begin{aligned}c_{NB} &= \operatorname{argmax}_{c_j \in C} P(c_j) \prod_i P(x_i | c_j) \\ &= \operatorname{argmax}_{c_j \in C} P(c_j) P(x_1 = \text{"our"} | c_j) \cdots P(x_n = \text{"text"} | c_j)\end{aligned}$$

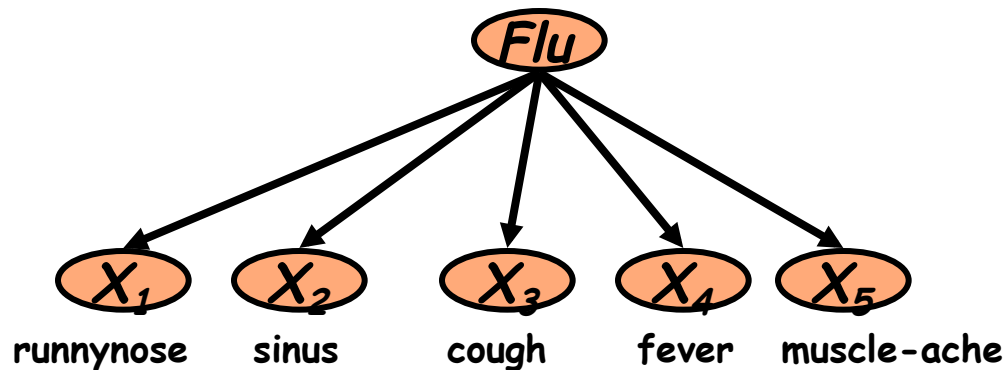
- Still too many possibilities
- Assume that classification is *independent* of the positions of the words
  - Result is bag of words model (over *tokens* not *types*)



# Naive Bayes: Learning

- From training corpus, extract *Vocabulary*
- Calculate required  $P(c_j)$  and  $P(x_k | c_j)$  terms
  - ★ For each  $c_j$  in  $C$  do
    - ✓  $docs_j \leftarrow$  subset of documents for which the target class is  $c_j$
    - ✓
 
$$P(c_j) \leftarrow \frac{|docs_j|}{|\text{total \# documents}|}$$
  - $Text_j \leftarrow$  single document containing all  $docs_j$
  - for each word  $t$  in *Vocabulary*  $V$ 
    - $T_{ct} \leftarrow$  number of occurrences of  $t$  in  $Text_j$
    - $$P(t | c) \leftarrow \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

# Problem with Maximum Likelihood



$$P(X_1, \dots, X_5 | C) = P(X_1 | C) \cdot P(X_2 | C) \cdot \dots \cdot P(X_5 | C)$$

- What if we have seen no training documents with the word ***muscle-ache*** and classified in the topic ***Flu***?

$$\hat{P}(X_5 = t | C = nf) = \frac{N(X_5 = t, C = nf)}{N(C = nf)} = 0$$

- Zero probabilities cannot be conditioned away, no matter the other evidence!

$$\ell = \arg \max_c \hat{P}(c) \prod_i \hat{P}(x_i | c)$$

# Add-one or Laplace Smoothing

- To eliminate zeros, use add-one or Laplace smoothing, which simply adds one to each count

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V} (T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B'}$$

where  $T_{ct}$  is the # of term  $t$  in topic  $c$ , and  $B$  is the vocabulary size

# Naive Bayes: Learning

- From training corpus, extract *Vocabulary*
- Calculate required  $P(c_j)$  and  $P(x_k | c_j)$  terms
  - ★ For each  $c_j$  in  $C$  do
    - ✓  $docs_j \leftarrow$  subset of documents for which the target class is  $c_j$
    - ✓
 
$$P(c_j) \leftarrow \frac{|docs_j|}{|\text{total \# documents}|}$$
  - $Text_j \leftarrow$  single document containing all  $docs_j$
  - for each word  $t$  in *Vocabulary*  $V$ 
    - $T_{ct} \leftarrow$  number of occurrences of  $t$  in  $Text_j$

**This becomes:**

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V} (T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B'}$$

# Naive Bayes: Classifying

- $positions \leftarrow$  all word positions in current document which contain tokens found in *Vocabulary*
- Return  $c_{NB}$ , where

$$c_{NB} = \operatorname{argmax}_{c_j \in C} P(c_j) \prod_{i \in positions} P(x_i | c_j)$$

## Example

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V} (T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B'}$$

	docID	words in document	in $c = \text{China?}$
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

$$\hat{P}(\text{Chinese}|c) = (5 + 1) / (8 + 6) = 6/14 = 3/7$$

$$\hat{P}(\text{Tokyo}|c) = \hat{P}(\text{Japan}|c) = (0 + 1) / (8 + 6) = 1/14$$

$$\hat{P}(\text{Chinese}|\bar{c}) = (1 + 1) / (3 + 6) = 2/9$$

$$\hat{P}(\text{Tokyo}|\bar{c}) = \hat{P}(\text{Japan}|\bar{c}) = (1 + 1) / (3 + 6) = 2/9$$

$$\hat{P}(c) = 3/4$$

$$\hat{P}(\bar{c}) = 1/4$$

$$\hat{P}(c|d_5) \propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003.$$

$$\hat{P}(\bar{c}|d_5) \propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001.$$

# Naive Bayes: Time Complexity

- **Training Time:**  $O(|D|L_{ave} + |C||V|)$ , where  $L_{ave}$  is the average length of a document in  $D$ .
  - ★ Assumes all counts are pre-computed in  $O(|D|L_{ave})$  time during one pass through all of the data.
  - ★ Generally just  $O(|D|L_{ave})$  since usually  $|C||V| < |D|L_{ave}$
- **Test Time:**  $O(|C| L_t)$ , where  $L_t$  is the average length of a test document.
- Very efficient overall, linearly proportional to the time needed to just read in all the data.

## Underflow Prevention: using logs

- Multiplying lots of probabilities, which are between 0 and 1 by definition, can result in floating-point underflow.
- Since  $\log(xy) = \log(x) + \log(y)$ , it is better to perform all computations by summing logs of probabilities rather than multiplying probabilities.
- Class with highest final un-normalized log probability score is still the most probable.

$$c_{NB} = \operatorname{argmax}_{c_j \in C} [\log P(c_j) + \sum_{i \in \text{positions}} \log P(x_i | c_j)]$$

- Note that model is now just max of sum of weights...



# Naive Bayes Classifier

$$c_{NB} = \operatorname{argmax}_{c_j \in C} [\log P(c_j) + \sum_{i \in \text{positions}} \log P(x_i | c_j)]$$

- Simple interpretation: Each conditional parameter  $\log P(x_i | c_j)$  is a weight that indicates how good an indicator  $x_i$  is for  $c_j$ .
- The prior  $\log P(c_j)$  is a weight that indicates the relative frequency of  $c_j$ .
- The sum is then a measure of how much evidence there is for the document being in the class.
- We select the class with the most evidence for it

# Two Models

## ■ Model 1: Multinomial (what we have seen)

- ★ One feature  $X_i$  for each word pos in document
  - ✓ feature's values are all words in dictionary
- ★ Value of  $X_i$  is the word in position  $i$
- ★ Naïve Bayes assumption:
  - ✓ Given the document's topic, word in one position in the document tells us nothing about words in other positions
- ★ Second assumption:
  - ✓ Word appearance does not depend on position

$$P(X_i = w | c) = P(X_j = w | c)$$

for all positions  $i, j$ , word  $w$ , and class  $c$

- ✓ Just have one multinomial feature predicting all words

# Two Naive Bayes Models

## ■ Model 2: Multivariate Bernoulli

- ★ One feature  $X_w$  for each word in dictionary
- ★  $X_w = \text{true}$  in document  $d$  if  $w$  appears in  $d$
- ★ Naive Bayes assumption:
  - ✓ Given the document's topic, appearance of one word in the document tells us nothing about chances that another word appears

# Parameter estimation

## ■ Multivariate Bernoulli model:

$$\hat{P}(X_w = t \mid c_j) =$$

fraction of documents of topic  $c_j$   
in which word  $w$  appears

$$P(t \mid c) \leftarrow \frac{N_{ct} + 1}{N_c + 2}$$

← (B = 2)

## ■ Multinomial model:

$$\hat{P}(X_i = w \mid c_j) =$$

fraction of times in which  
word  $w$  appears among all  
words in documents of topic  $c_j$

- ★ Can create a mega-document for topic  $j$  by concatenating all documents in this topic
- ★ Use frequency of  $w$  in mega-document

# Classification

- Multinomial vs Multivariate Bernoulli?
- Multinomial model is almost always more effective in text applications!
  - ✓ See results figures later

## Feature Selection: Why?

- Text collections have a large number of features
  - ★ 10,000 – 1,000,000 unique words ... and more
- May make using a particular classifier feasible
  - ★ Some classifiers can't deal with 100,000 of features
- Reduces training time
  - ★ Training time for some methods is quadratic or worse in the number of features
- Can improve generalization (performance)
  - ★ Eliminates noise features
  - ★ Avoids overfitting

# Feature selection: how?

## ■ Two ideas:

### ★ Hypothesis testing statistics:

- ✓ Are we confident that the value of one categorical variable is associated with the value of another
- ✓ Chi-square test ( $\chi^2$ )

### ★ Information theory:

- ✓ How much information does the value of one categorical variable give you about the value of another
- ✓ Mutual information

## ■ They're similar, but $\chi^2$ measures confidence in association, (based on available statistics), while MI measures extent of association (assuming perfect knowledge of probabilities)

## $\chi^2$ statistic (CHI)

- $\chi^2$  is interested in  $(f_o - f_e)^2 / f_e$  summed over all table entries: is the observed number what you'd expect given the marginals?

$$\chi^2(j, a) = \sum (O - E)^2 / E = (2 - .25)^2 / .25 + (3 - 4.75)^2 / 4.75 + (500 - 502)^2 / 502 + (9500 - 9498)^2 / 9498 = 12.9 \quad (p < .001)$$

- The null hypothesis is rejected with confidence .999,
- since  $12.9 > 10.83$  (the value for .999 confidence).

	<i>Term = jaguar</i>	<i>Term ≠ jaguar</i>	
<i>Class = auto</i>	2 (0.25)	500 (502)	expected: $f_e$
<i>Class ≠ auto</i>	3 (4.75)	9500 (9498)	observed: $f_o$



$p$	$\chi^2$ critical value
0.1	2.71
0.05	3.84
0.01	6.63
0.005	7.88
0.001	10.83

- This table says that, for example, if two events are independent, then  $P(\chi^2 > 6.63) < 0.01$ . So for  $\chi^2 > 6.63$  the assumption of independence can be rejected with 99% confidence.

# $\chi^2$ statistic (CHI)

There is a simpler formula for 2x2  $\chi^2$ :

$$\chi^2(t, c) = \frac{N \times (AD - CB)^2}{(A + C) \times (B + D) \times (A + B) \times (C + D)}$$

$A = \#(t, c)$	$C = \#(\neg t, c)$
$B = \#(t, \neg c)$	$D = \#(\neg t, \neg c)$

$$N = A + B + C + D$$

# Feature selection via Mutual Information

- In training set, choose  $k$  words which best discriminate (give most info on) the categories.
- The Mutual Information between a word, class is:

$$I(w, c) = \sum_{e_w \in \{0,1\}} \sum_{e_c \in \{0,1\}} p(e_w, e_c) \log \frac{p(e_w, e_c)}{p(e_w)p(e_c)}$$

★ For each word  $w$  and each category  $c$

## Feature selection via MI (contd.)

- For each category we build a list of  $k$  most discriminating terms.
- For example (on 20 Newsgroups):
  - ★ ***sci.electronics***: circuit, voltage, amp, ground, copy, battery, electronics, cooling, ...
  - ★ ***rec.autos***: car, cars, engine, ford, dealer, mustang, oil, collision, autos, tires, toyota, ...
- Greedy: does not account for correlations between terms

# Feature Selection

- Mutual Information
  - ★ Clear information-theoretic interpretation
  - ★ May select rare uninformative terms
- Chi-square
  - ★ Statistical foundation
  - ★ May select very slightly informative frequent terms that are not very useful for classification
- Just use the most common terms?
  - ★ No particular foundation
  - ★ In practice, this is often 90% as good

# Feature selection for NB

- In general feature selection is *necessary* for multivariate Bernoulli NB.
- Otherwise you suffer from noise, multi-counting
- “Feature selection” really means something different for multinomial NB. It means dictionary truncation
- This “feature selection” normally isn’t needed for multinomial NB, but may help a fraction with quantities that are badly estimated

# Evaluating Categorization

- Evaluation must be done on test data that are independent of the training data (usually a disjoint set of instances).
  - ★ Sometimes use cross-validation (averaging results over multiple training and test splits of the overall data)
- It's easy to get good performance on a test set that was available to the learner during training (e.g., just memorize the test set).
- Measures: precision, recall, F1, classification accuracy
- *Classification accuracy*:  $c/n$  where  $n$  is the total number of test instances and  $c$  is the number of test instances correctly classified by the system.
  - ★ Adequate if one class per document
  - ★ Otherwise F measure for each class

# Performance Measures

	label $y = +1$	label $y = -1$
prediction $h(\mathbf{x}) = +1$	$f_{++}$	$f_{+-}$
prediction $h(\mathbf{x}) = -1$	$f_{-+}$	$f_{--}$

## Contingency Table

$$Err(h) = \Pr(h(\mathbf{x}) \neq y) = \frac{f_{+-} + f_{-+}}{f_{++} + f_{+-} + f_{-+} + f_{--}} \quad \text{Error rate}$$



# Recall

- Is the probability that a document with label  $y=1$  is classified correctly:

$$\text{Recall}(h) = \Pr(h(\mathbf{x}) = 1 \mid y = 1, h)$$

- Estimation of the recall:

$$\text{Recall}_{\text{test}}(h) = \frac{f_{++}}{f_{++} + f_{-+}}$$

Total number of  
documents with label  
 $y = 1$

# Precision

- Is the probability that a document classified as 1 is classified correctly:

$$\text{Prec}(h) = \Pr(y = 1 \mid h(\mathbf{x}) = 1, h)$$

- Estimation of the precision:

$$\text{Prec}_{\text{test}}(h) = \frac{f_{++}}{f_{++} + f_{+-}}$$

Total number of  
documents predicted  
as positives

# Precision and Recall combined

- Between high precision and high recall exists a trade-off;
- Single performance measure: F measure

$$F_{\beta}(h) = \frac{(1 + \beta^2) \text{Prec}(h) \text{Recall}(h)}{\text{Prec}(h) + \text{Recall}(h)}$$

usually  $\beta = 1$

# Macroaveraging vs. Microaveraging

- *Macroaveraging computes a simple average over classes.*
- *Microaveraging pools per-document decisions across classes, and then computes an effectiveness measure on the pooled contingency table.*

	class 1			class 2			pooled table	
	truth: yes	truth: no		truth: yes	truth: no		truth: yes	truth: no
call: yes	10	10	call: yes	90	10	call: yes	100	20
call: no	10	970	call: no	10	890	call: no	20	1860

Macroaveraged precision =  $(10/(10+10) + 90/(90+10))/2 = (0.5+0.9)/2 = 0.7$

Microaveraged precision =  $100/(100+20) = 5/6 = 0.83$

# Naive Bayes vs. other methods

## ■ Reuters datasets:

★ <http://www.daviddlewis.com/resources/testcollections/>

(a)	NB	Rocchio	kNN	SVM	
micro-avg-L (90 classes)	80	85	86	89	
macro-avg (90 classes)	47	59	60	60	

(b)	NB	Rocchio	kNN	trees	SVM
earn	96	93	97	98	98
acq	88	65	92	90	94
money-fx	57	47	78	66	75
grain	79	68	82	85	95
crude	80	70	86	85	89
trade	64	65	77	73	76
interest	65	63	74	67	78
ship	85	49	79	74	86
wheat	70	69	77	93	92
corn	65	48	78	92	90
micro-avg (top 10)	82	65	82	88	92
micro-avg-D (118 classes)	75	62	n/a	n/a	87

Evaluation measure:  $F_1$

Naive Bayes does pretty well, but some methods beat it consistently (e.g., SVM).

# WebKB Experiment (1998)

- Classify webpages from CS departments into:

- ★ student, faculty, course, project

- ★ Dataset available at:

<http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/>

- Train on ~5,000 hand-labeled web pages

- ★ Cornell, Washington, U.Texas, Wisconsin

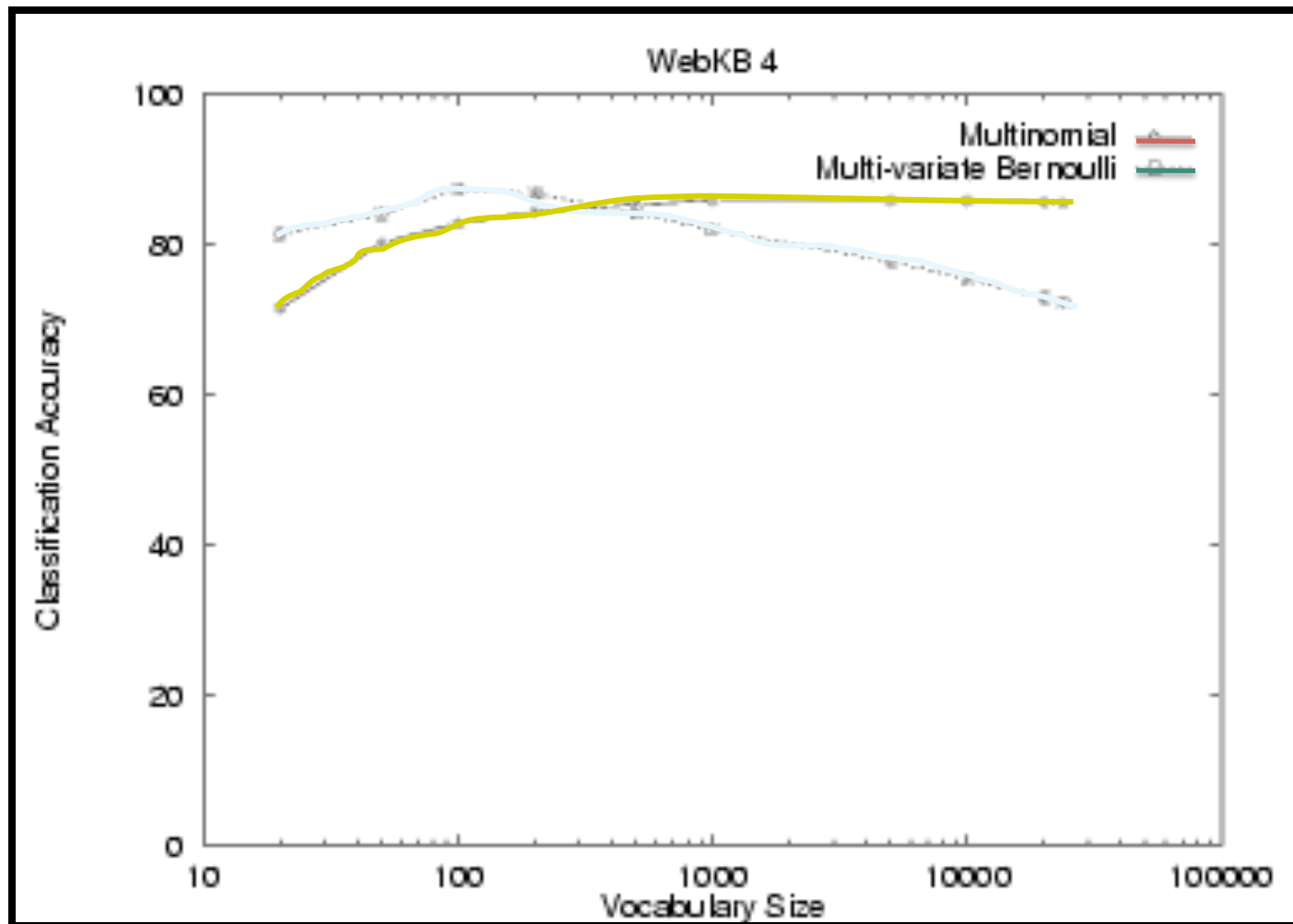
- Crawl and classify a new site (CMU)

- Results:

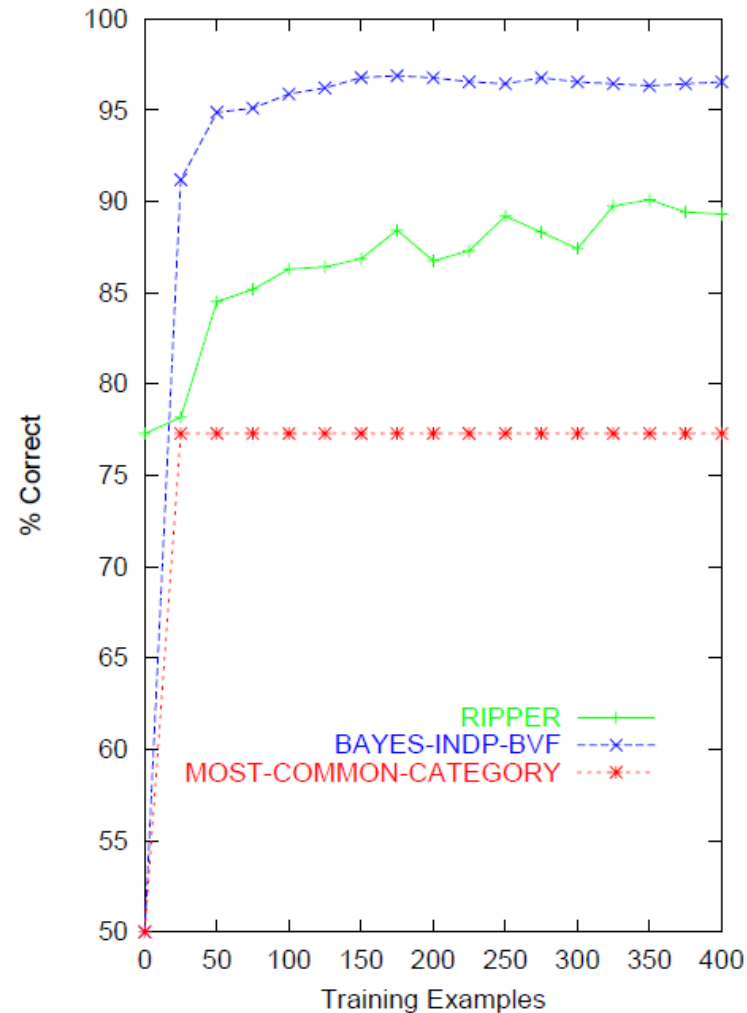


	Student	Faculty	Person	Project	Course	Department
Extracted	180	66	246	99	28	1
Correct	130	28	194	72	25	1
Accuracy:	72%	42%	79%	73%	89%	100%

# NB Model Comparison: WebKB



# Naïve Bayes on spam email



<http://www.cs.utexas.edu/users/jp/research/email.paper.pdf>  
**Naïve-Bayes vs. Rule-Learning in Classification of Email**  
Jefferson Provost, UT Austin



# SpamAssassin

- Naïve Bayes has found a home in spam filtering
  - ★ Paul Graham's *A Plan for Spam*
    - ✓ <http://www.paulgraham.com/spam.html>
  - ★ Widely used in spam filters
    - ✓ Classic Naive Bayes superior when appropriately used
      - According to David D. Lewis
  - ★ But also many other things: black hole lists, etc.
- Many email topic filters also use NB classifiers

# Naive Bayes is Not So Naive

- Naive Bayes won 1<sup>st</sup> and 2<sup>nd</sup> place in KDD-CUP 97 competition out of 16 systems

Goal: Financial services industry direct mail response prediction model: Predict if the recipient of mail will actually respond to the advertisement – 750,000 records.

- More robust to irrelevant features than many learning methods

Irrelevant Features cancel each other without affecting results

Decision Trees can suffer **heavily** from this.

- More robust to concept drift (changing class definition over time)

- Very good in domains with many equally important features

Decision Trees suffer from *fragmentation* in such cases – especially if little data

- A good dependable baseline for text classification (but not the best)!

- Optimal if the Independence Assumptions hold: **Bayes Optimal Classifier**

Never true for text, but possible in some domains

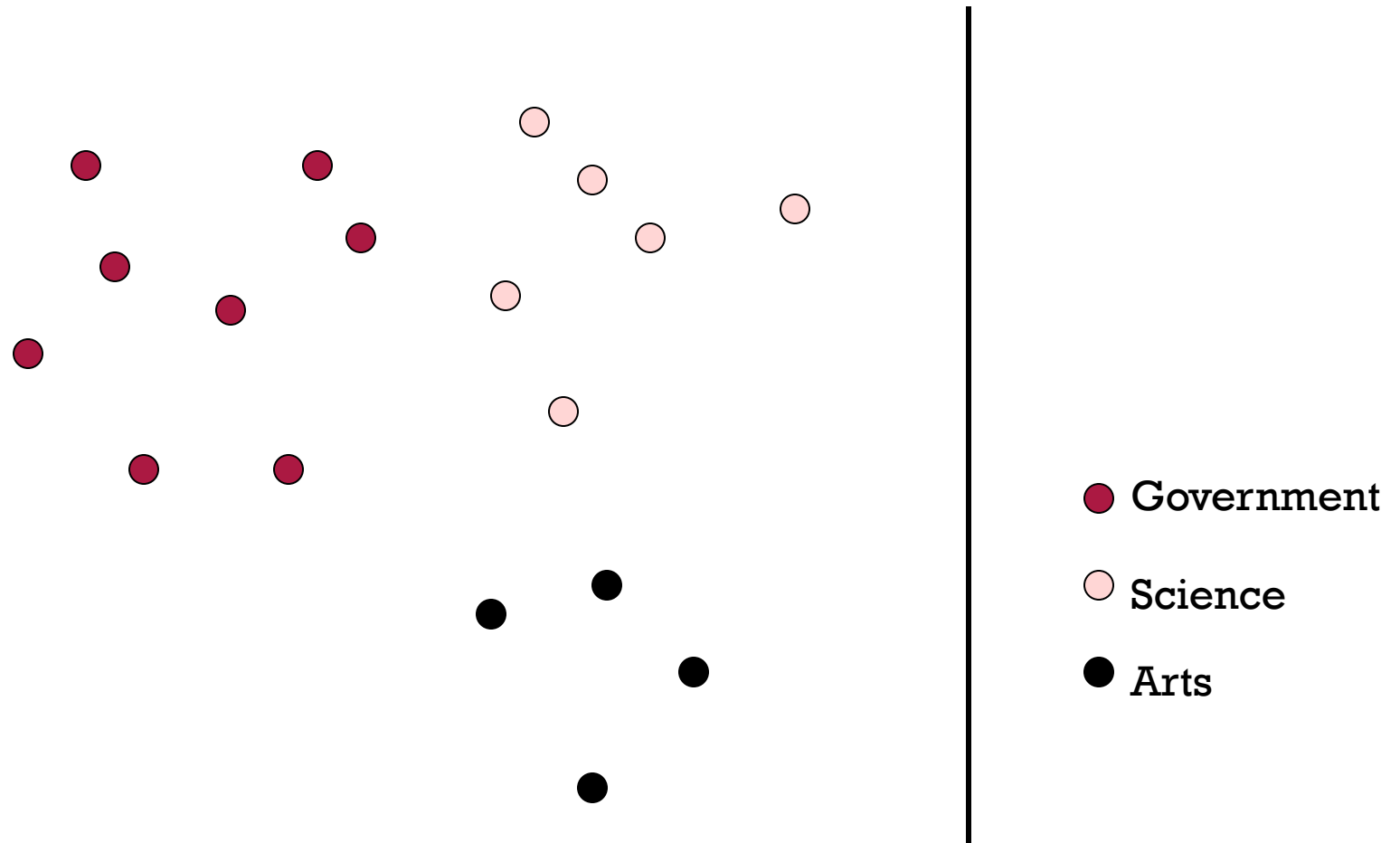
- Very Fast Learning and Testing (basically just count the data)

- Low Storage requirements

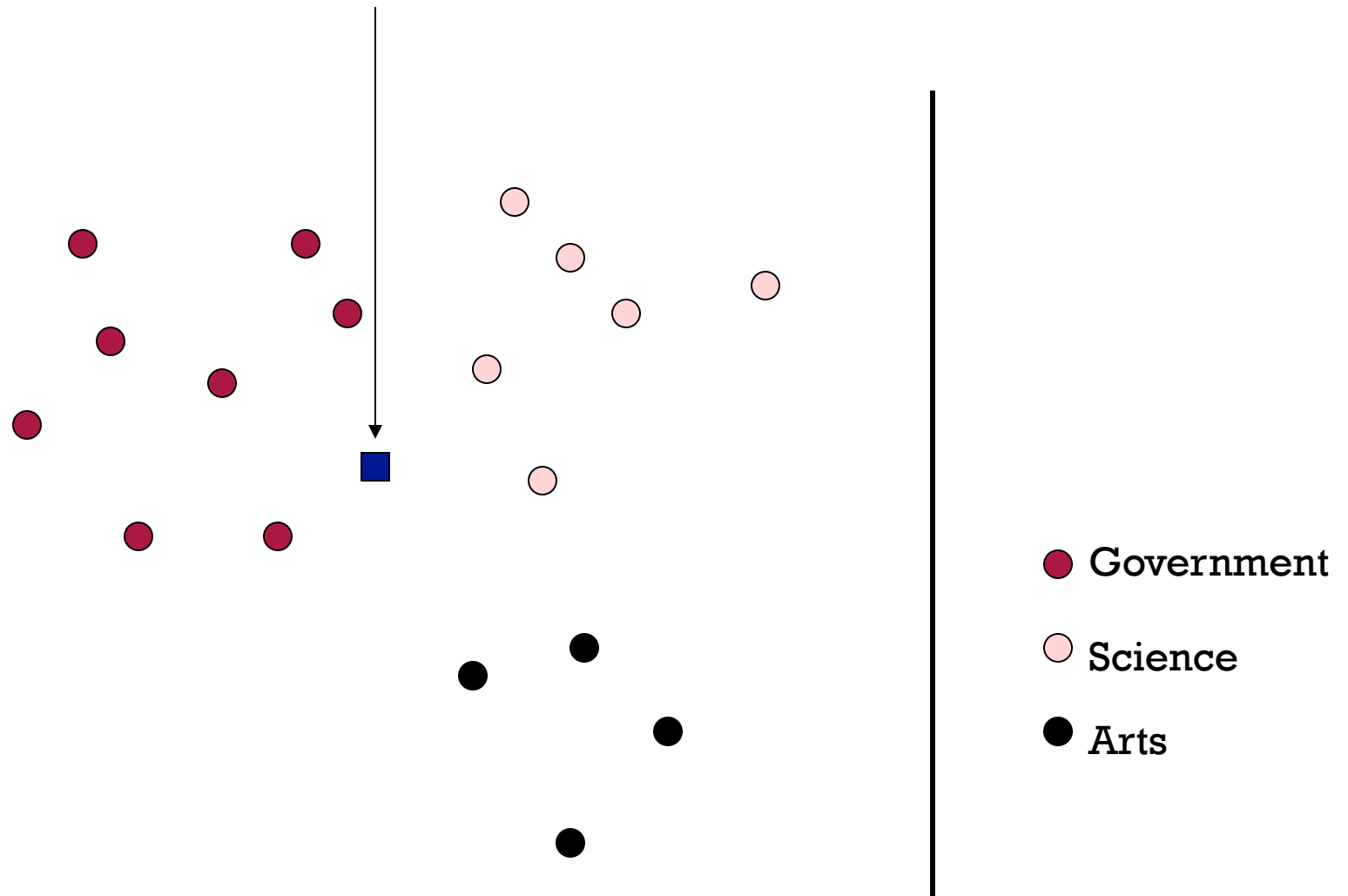
# Classification Using Vector Spaces

- As before, the training set is a set of documents, each labeled with its class (e.g., topic)
- In vector space classification, this set corresponds to a labeled set of points (or, equivalently, vectors) in the vector space
- **Premise 1:** Documents in the same class form a contiguous region of space
- **Premise 2:** Documents from different classes don't overlap (much)
- We define surfaces to delineate classes in the space

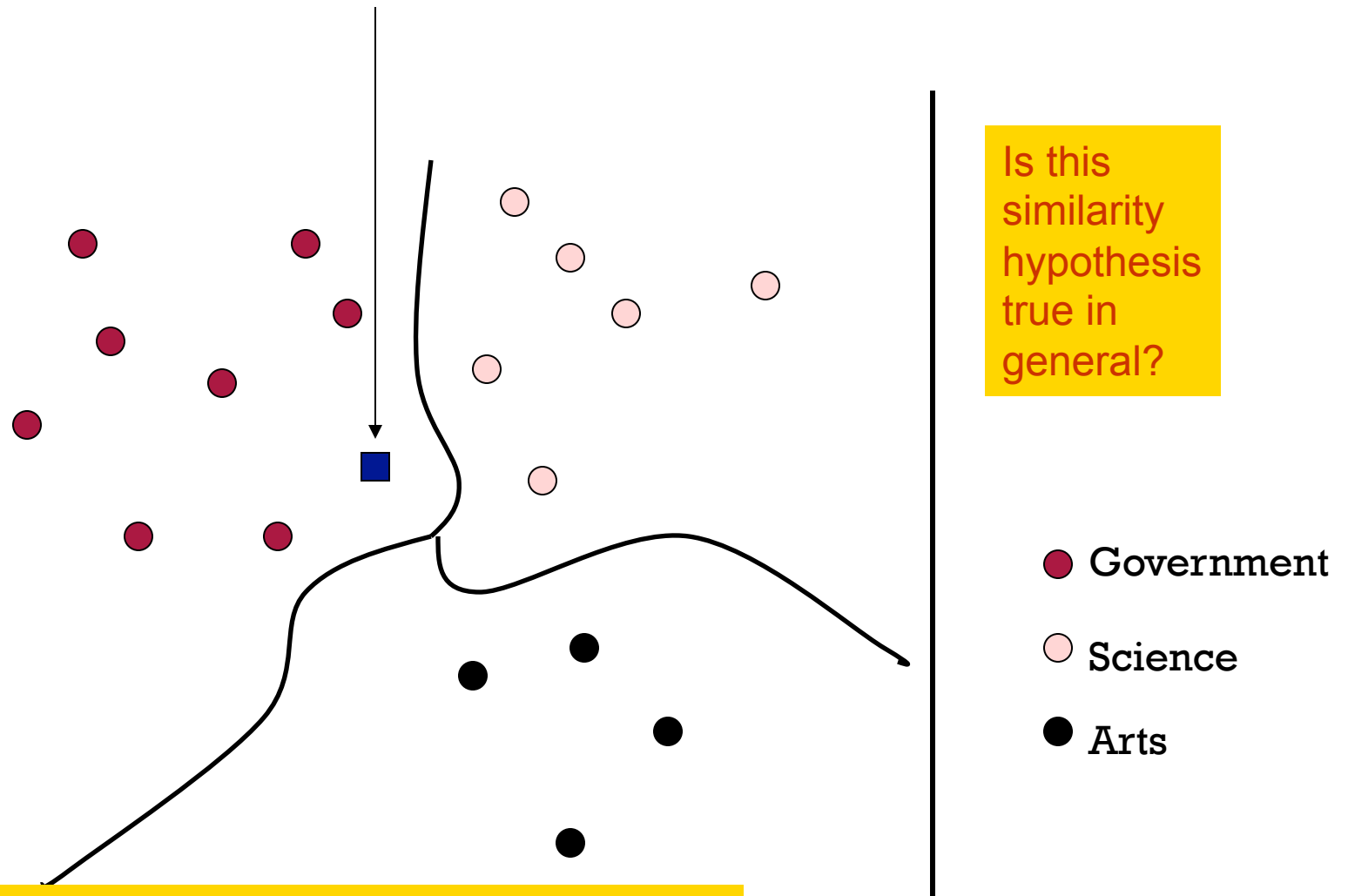
# Documents in a Vector Space



# Test Document of what class?

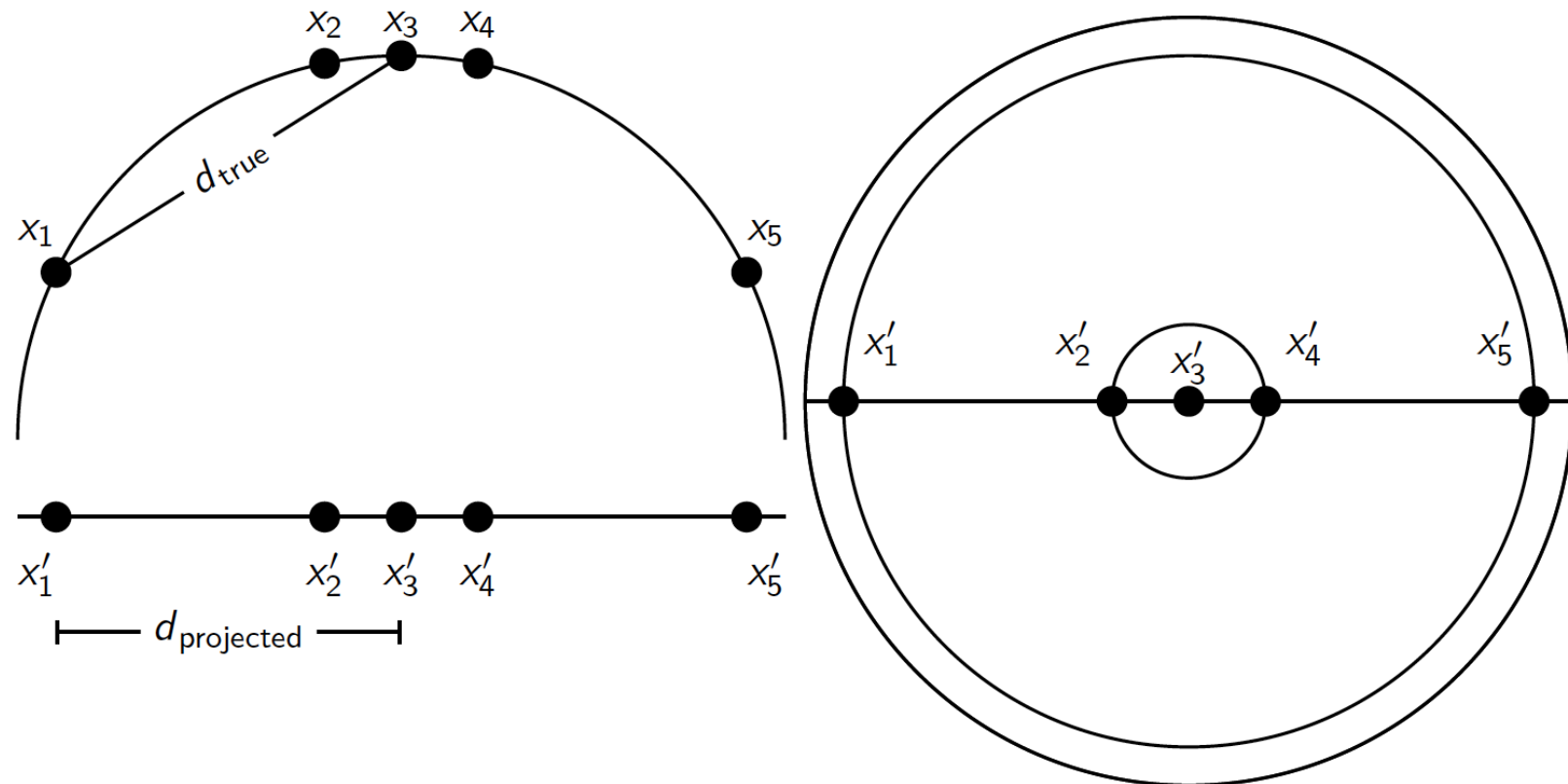


# Test Document = Government



The next topic is how to find good separators

# Aside: 2D/3D graphs can be misleading



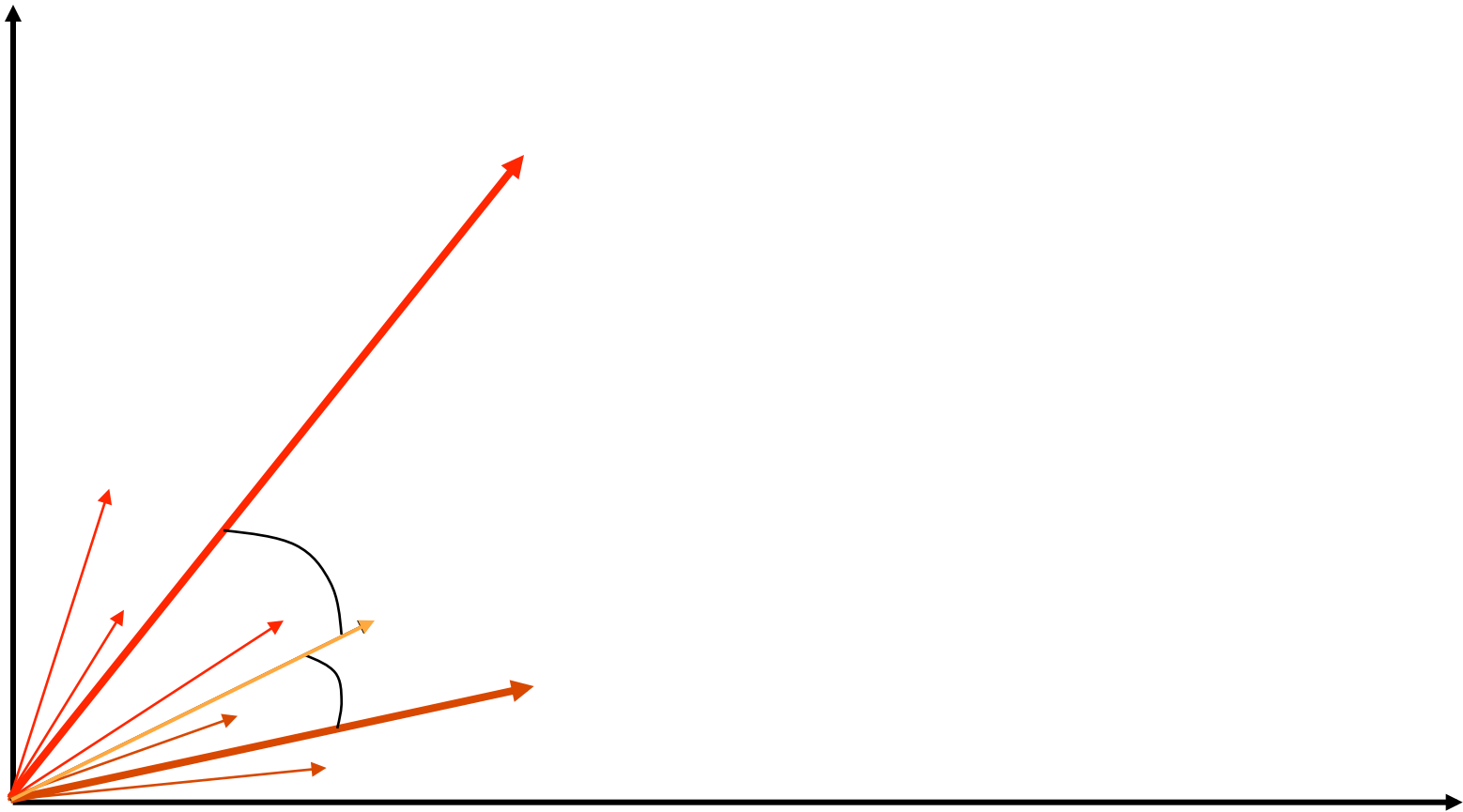
*Left:* A projection of the 2D semicircle to 1D. For the points  $x_1, x_2, x_3, x_4, x_5$  at  $x$  coordinates  $-0.9, -0.2, 0, 0.2, 0.9$  the distance  $|x_2 x_3| \approx 0.201$  only differs by 0.5% from  $|x'_2 x'_3| = 0.2$ ; but  $|x_1 x_3|/|x'_1 x'_3| = d_{\text{true}}/d_{\text{projected}} \approx 1.06/0.9 \approx 1.18$  is an example of a large distortion (18%) when projecting a large area. *Right:* The corresponding projection of the 3D hemisphere to 2D.

# Using Rocchio for text classification

- Relevance feedback methods can be adapted for text categorization
  - Relevance feedback can be viewed as 2-class classification
    - Relevant vs. nonrelevant documents
- Use standard tf-idf weighted vectors to represent text documents
- For training documents in each category, compute a prototype vector by summing the vectors of the training documents in the category.
  - Prototype = centroid of members of class
- Assign test documents to the category with the closest prototype vector based on cosine similarity.



# Illustration of Rocchio Text Categorization



## Definition of centroid

$$\vec{\mu}(c) = \frac{1}{|D_c|} \sum_{d \in D_c} \vec{v}(d)$$

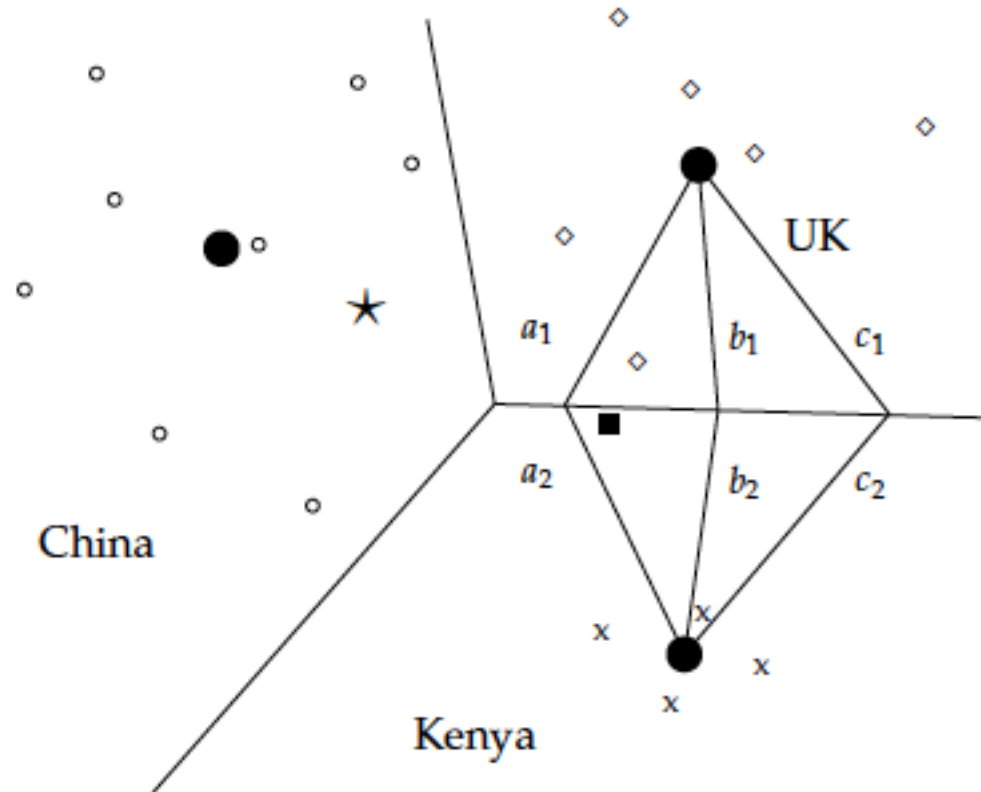
- Where  $D_c$  is the set of all documents that belong to class  $c$  and  $v(d)$  is the vector space representation of  $d$ .
- *Note that centroid will in general not be a unit vector even when the inputs are unit vectors.*

# Rocchio Properties

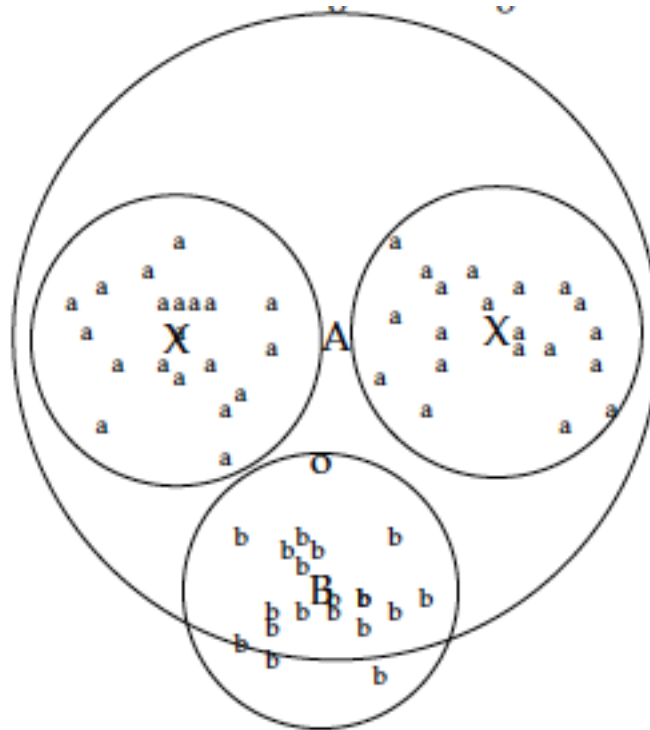
- Forms a simple generalization of the examples in each class (a *prototype*).
- Prototype vector does not need to be averaged or otherwise normalized for length since cosine similarity is insensitive to vector length.
- Classification is based on similarity to class prototypes.
- Does not guarantee classifications are consistent with the given training data.

Why not?

# Problem with Non-Similar Radii



# Problem with Multimodal Classes



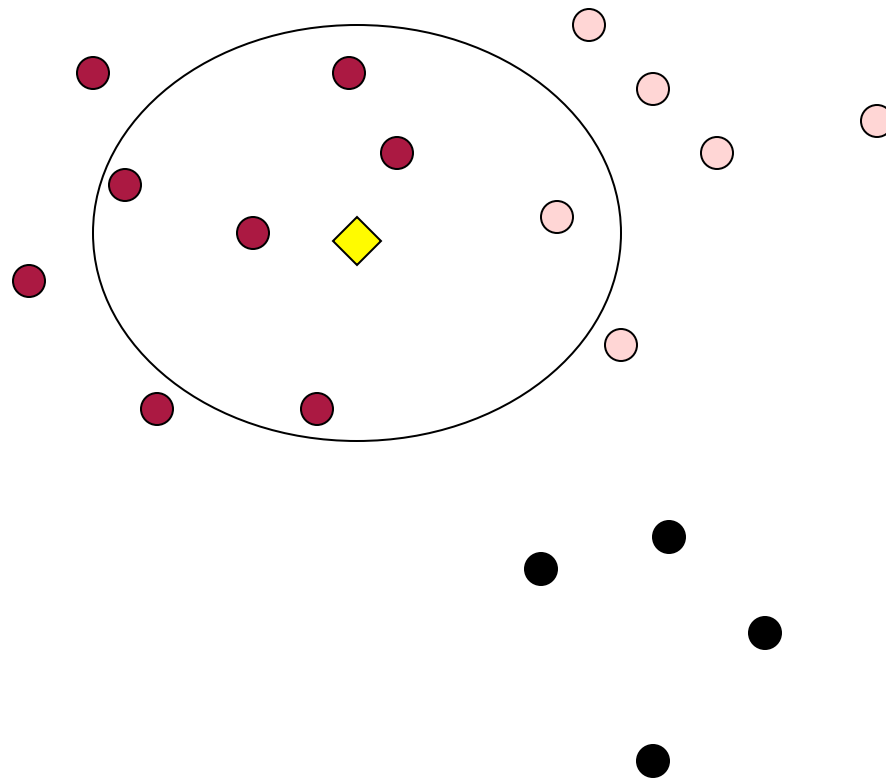
# Rocchio classification

- Rocchio forms a simple representation for each class: the centroid/prototype
- Classification is based on similarity to / distance from the prototype/centroid
- It does not guarantee that classifications are consistent with the given training data
- It is little used outside text classification
  - ★ It has been used quite effectively for text classification
  - ★ But in general worse than Naïve Bayes
- Again, cheap to train and test documents

# k Nearest Neighbor Classification

- kNN = k Nearest Neighbor
- To classify a document  $d$  into class  $c$ :
- Define  $k$ -neighborhood  $N$  as  $k$  nearest neighbors of  $d$
- Count number of documents  $i$  in  $N$  that belong to  $c$
- Estimate  $P(c|d)$  as  $i/k$
- Choose as class  $\operatorname{argmax}_c P(c|d)$  [ = majority class]

## Example: $k=6$ (6NN)



$P(\text{science} | \text{ })?$  

 Government

 Science

 Arts



# Nearest-Neighbor Learning Algorithm

- Learning is just storing the representations of the training examples in  $D$ .
- Testing instance  $x$  (*under 1NN*):
  - ★ Compute similarity between  $x$  and all examples in  $D$ .
  - ★ Assign  $x$  the category of the most similar example in  $D$ .
- Does not explicitly compute a generalization or category prototypes.
- Also called:
  - ★ Case-based learning
  - ★ Memory-based learning
  - ★ Lazy learning
- Rationale of kNN: contiguity hypothesis

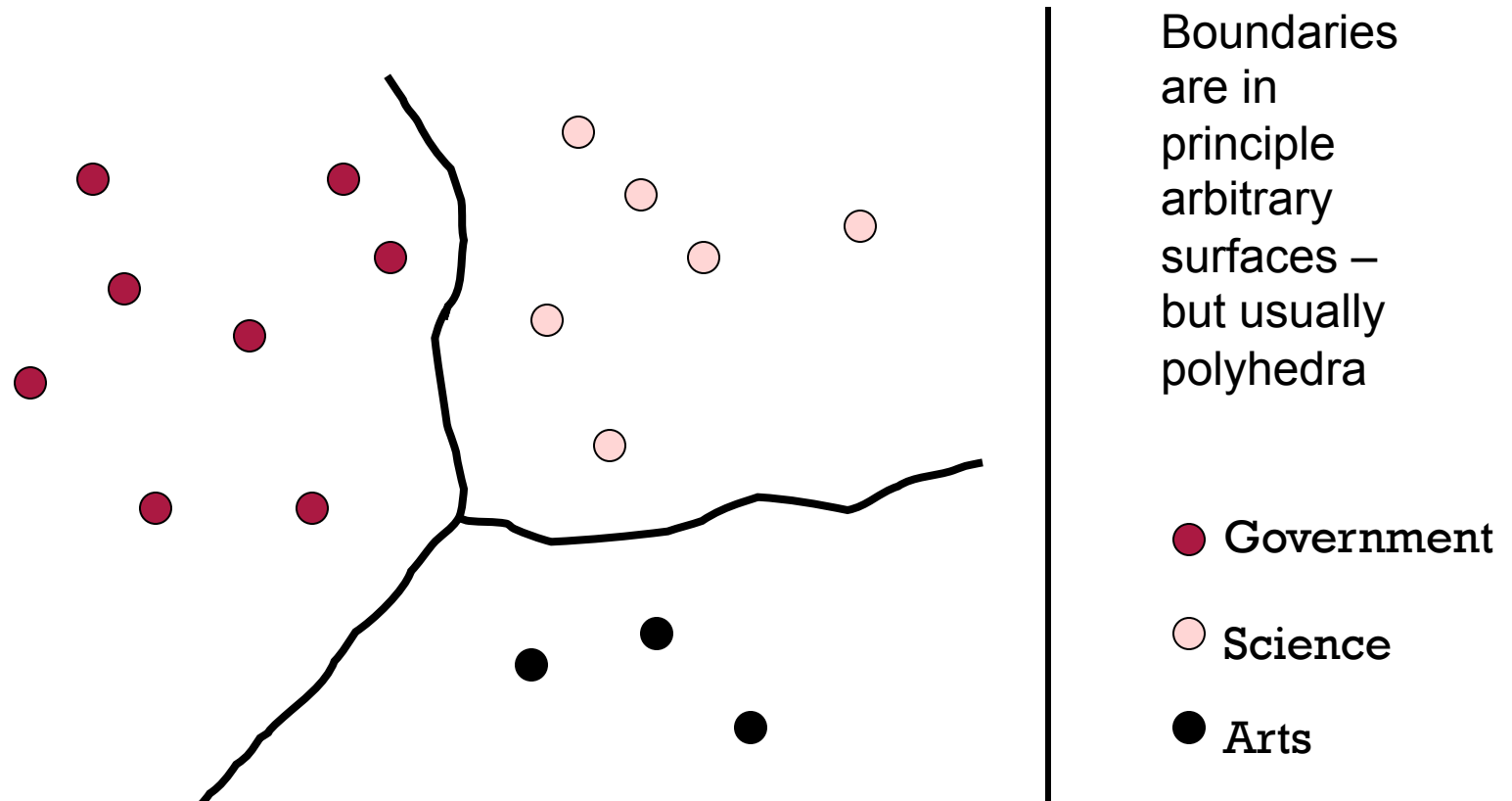
# kNN Is Close to Optimal

- Asymptotically, the error rate of 1-nearest-neighbor classification is less than twice the Bayes rate [error rate of classifier knowing model that generated data]
- In particular, asymptotic error rate is 0 if Bayes rate is 0.
- Assume: query point coincides with a training point.
- Both query point and training point contribute error  $\rightarrow$  2 times Bayes rate

## **k Nearest Neighbor**

- Using only the closest example (1NN) to determine the class is subject to errors due to:
  - ★ A single atypical example.
  - ★ Noise (i.e., an error) in the category label of a single training example.
- More robust alternative is to find the  $k$  most-similar examples and return the majority category of these  $k$  examples.
- Value of  $k$  is typically odd to avoid ties; 3 and 5 are most common.

# kNN decision boundaries



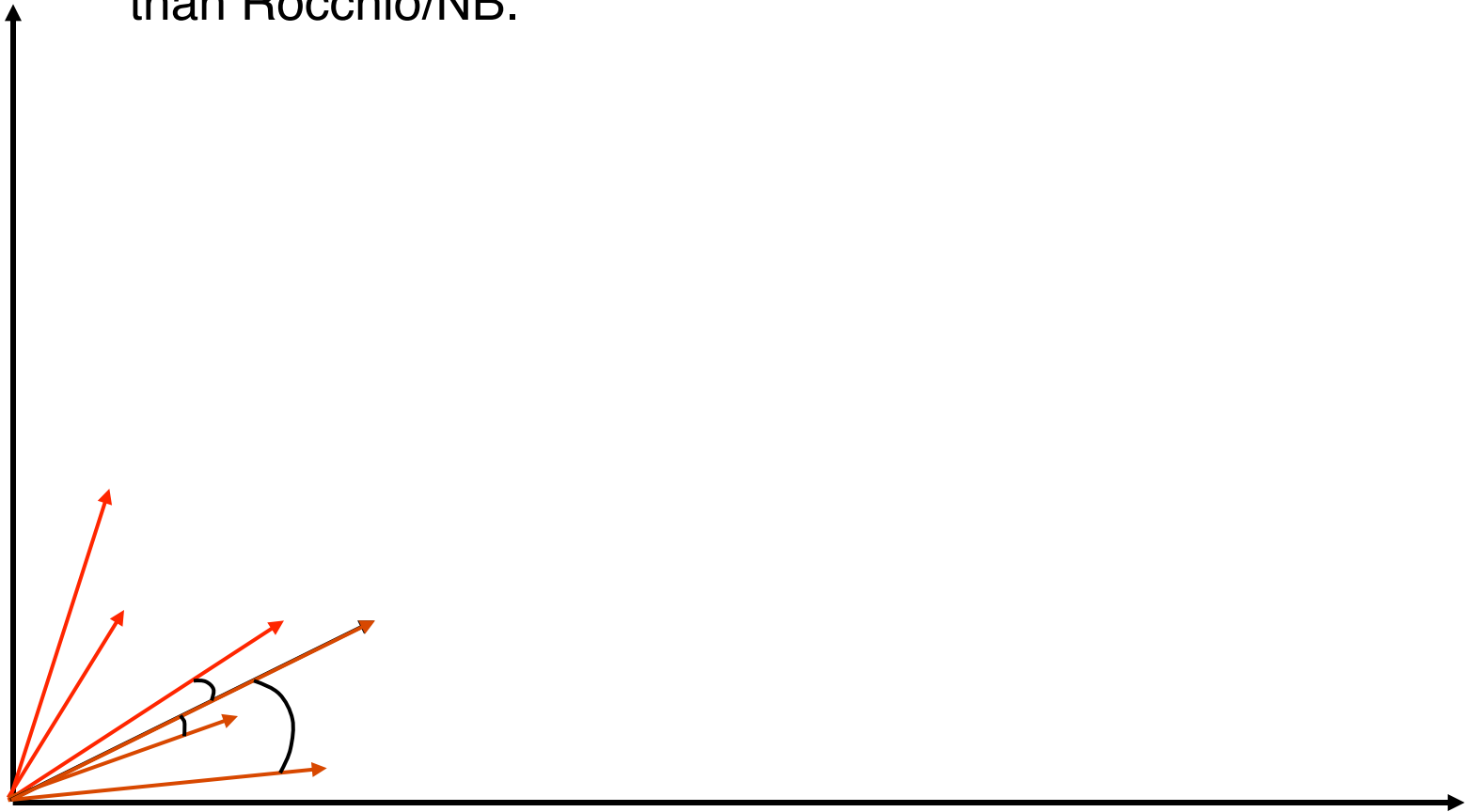
kNN gives locally defined decision boundaries between classes – far away points do not influence each classification decision (unlike in Naïve Bayes, Rocchio, etc.)

# Similarity Metrics

- Nearest neighbor method depends on a similarity (or distance) metric.
- Simplest for continuous  $m$ -dimensional instance space is *Euclidean distance*.
- Simplest for  $m$ -dimensional binary instance space is *Hamming distance* (number of feature values that differ).
- For text, cosine similarity of tf.idf weighted vectors is typically most effective.

# Illustration of 3 Nearest Neighbor for Text Vector Space

- Nearest Neighbor tends to handle polymorphic categories better than Rocchio/NB.



## kNN: Discussion

- No feature selection necessary
- Scales well with large number of classes
  - ★ Don't need to train  $n$  classifiers for  $n$  classes
- Classes can influence each other
  - ★ Small changes to one class can have ripple effect
- Scores can be hard to convert to probabilities
- No training necessary
- May be expensive at test time
- In most cases it's more accurate than NB or Rocchio

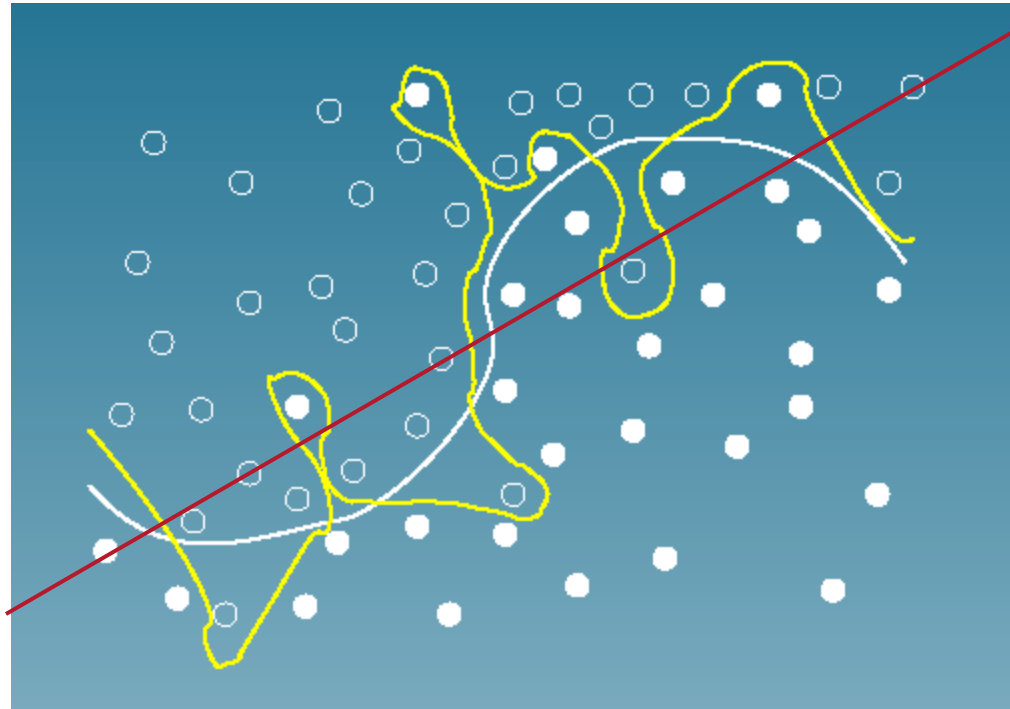
# kNN vs. Naive Bayes

- Bias/Variance tradeoff
  - ★ Variance  $\approx$  Capacity
- kNN has **high variance** and **low bias**.
  - ★ Infinite memory
- NB has **low variance** and **high bias**.
  - ★ Decision surface has to be linear (hyperplane – see later)
- Consider asking a botanist: **Is an object a tree?**
  - ★ Too much capacity/variance, low bias
    - ✓ Botanist who memorizes
    - ✓ Will always say “no” to new object (e.g., different # of leaves)
  - ★ Not enough capacity/variance, high bias
    - ✓ Lazy botanist
    - ✓ Says “yes” if the object is green
  - ★ You want the middle ground

(Example due to C. Burges)



# Bias vs. variance: Choosing the correct model capacity

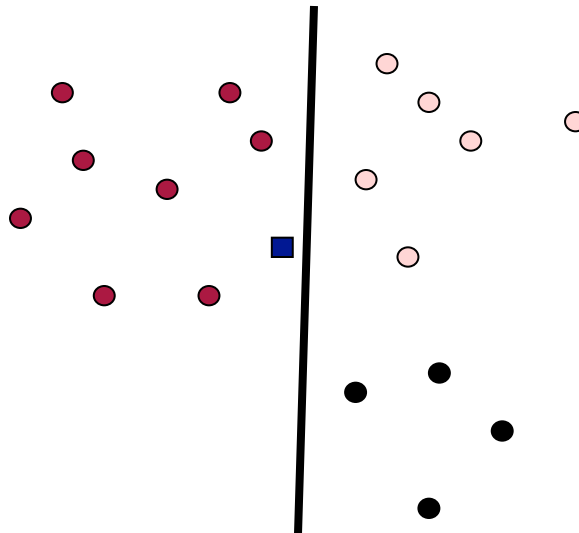


# Linear classifiers and binary and multiclass classification

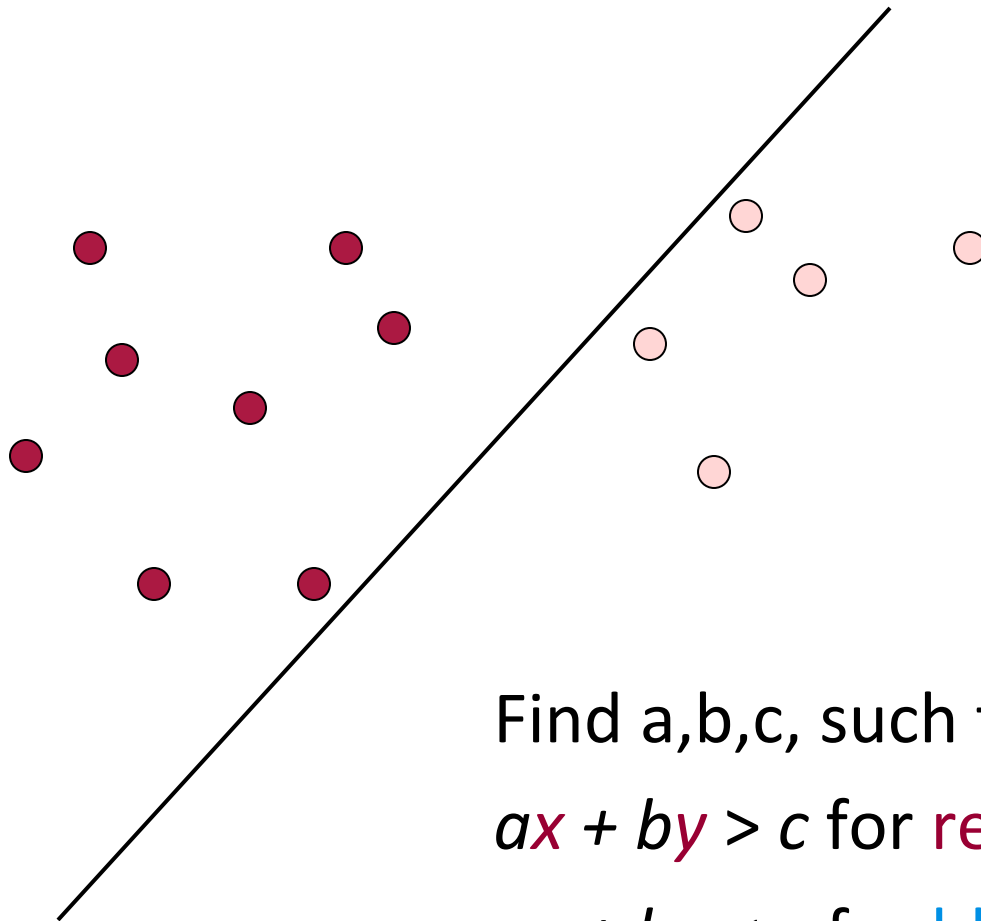
- Consider 2 class problems
  - ★ Deciding between two classes, perhaps, government and non-government
    - ✓ One-versus-rest classification
- How do we define (and find) the separating surface?
- How do we decide which region a test doc is in?

# Separation by Hyperplanes

- A strong high-bias assumption is *linear separability*:
  - ★ in 2 dimensions, can separate classes by a line
  - ★ in higher dimensions, need hyperplanes
- Can find separating hyperplane by *linear programming*  
(or can iteratively fit solution via perceptron):
  - ★ separator can be expressed as  $ax + by = c$



# Linear programming / Perceptron

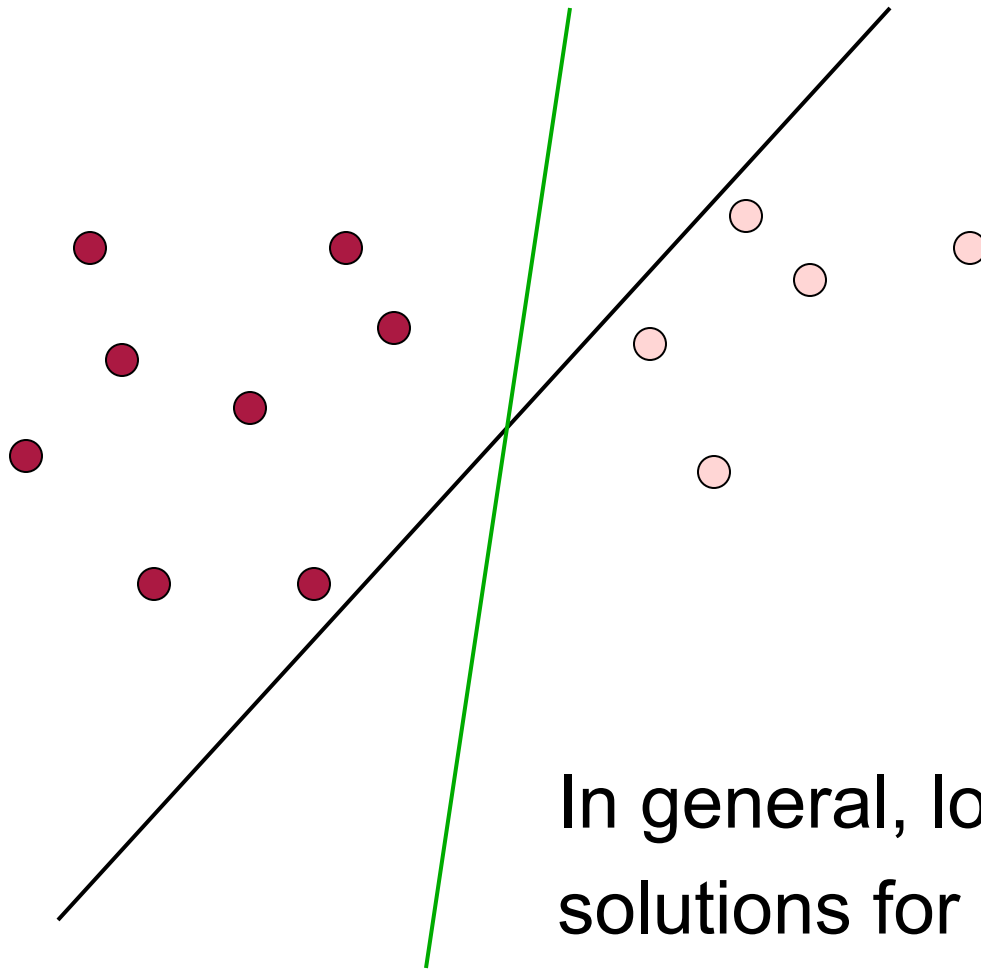


Find  $a, b, c$ , such that

$ax + by > c$  for red points

$ax + by < c$  for blue points.

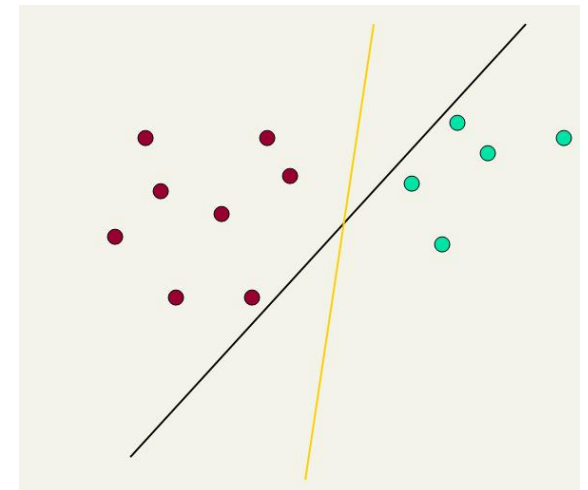
# Which Hyperplane?



In general, lots of possible solutions for  $a, b, c$ .

## Which Hyperplane?

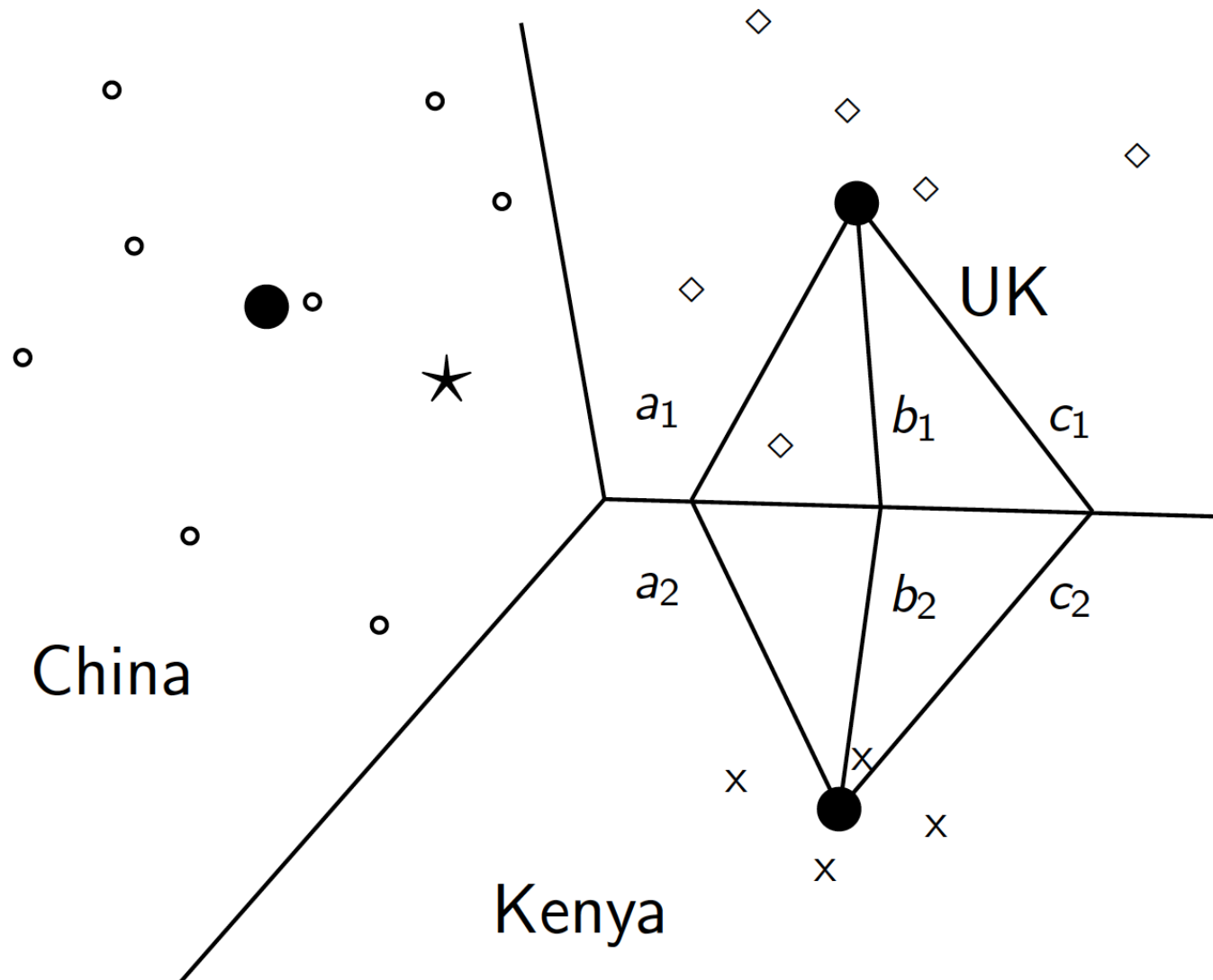
- Lots of possible solutions for  $a, b, c$ .
- Some methods find a separating hyperplane, but not the optimal one  
[according to some criterion of expected goodness]
  - ★ E.g., perceptron
- Most methods find an optimal separating hyperplane
- Which points should influence optimality?
  - ★ All points
    - ✓ Linear/logistic regression
    - ✓ Naïve Bayes
  - ★ Only “difficult points” close to decision boundary
    - ✓ Support vector machines



# Linear Classifiers

- Many common text classifiers are linear classifiers
  - ★ Naïve Bayes
  - ★ Perceptron
  - ★ Rocchio
  - ★ Logistic regression
  - ★ Support vector machines (with linear kernel)
  - ★ Linear regression with threshold
- Despite this similarity, noticeable performance differences
  - ★ For separable problems, there is an infinite number of separating hyperplanes. Which one do you choose?
  - ★ What to do for non-separable problems?
  - ★ Different training methods pick different hyperplanes
- Classifiers more powerful than linear often don't perform better on text problems. Why?

# Rocchio is a linear classifier





# Naive Bayes is a linear classifier

- Two-class Naive Bayes. We compute:

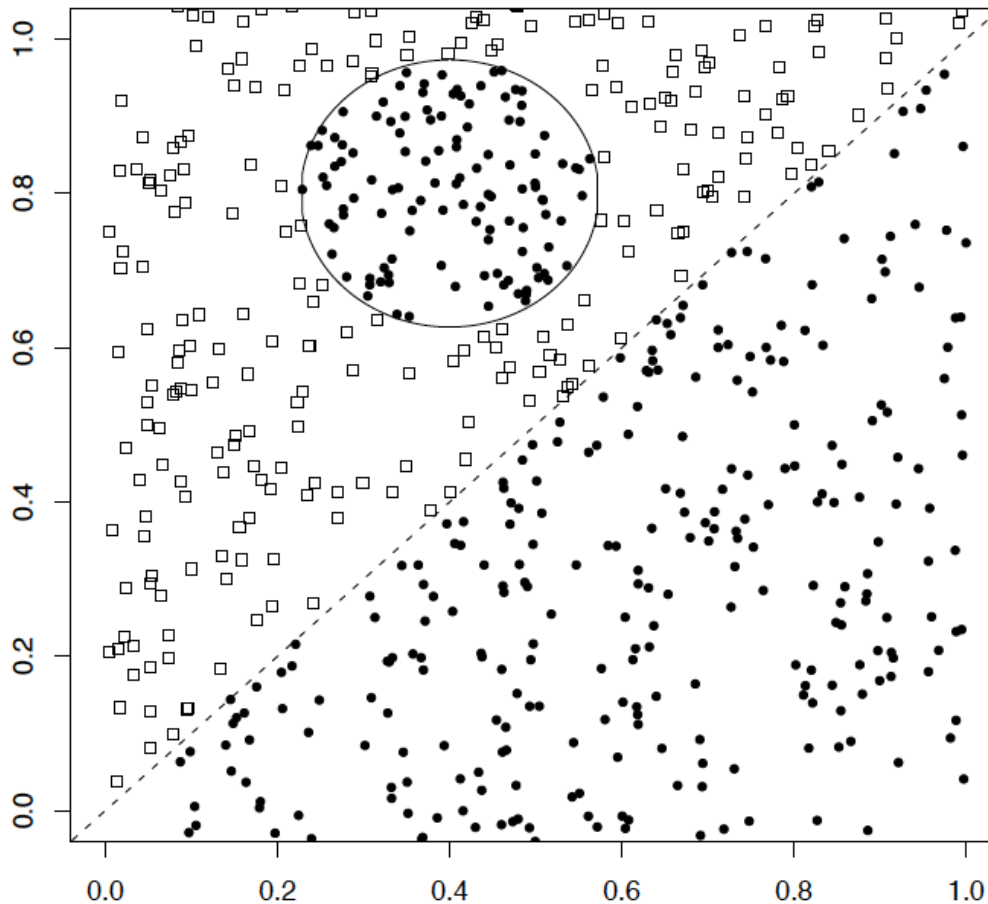
$$\log \frac{P(C | d)}{P(\bar{C} | d)} = \log \frac{P(C)}{P(\bar{C})} + \sum_{w \in d} \log \frac{P(w | C)}{P(w | \bar{C})}$$

- Decide class  $C$  if the odds is greater than 1, i.e., if the log odds is greater than 0.
- So decision boundary is hyperplane:

$$\alpha + \sum_{w \in V} \beta_w \times n_w = 0 \quad \text{where } \alpha = \log \frac{P(C)}{P(\bar{C})};$$

$$\beta_w = \log \frac{P(w | C)}{P(w | \bar{C})}; \quad n_w = \# \text{ of occurrences of } w \text{ in } d$$

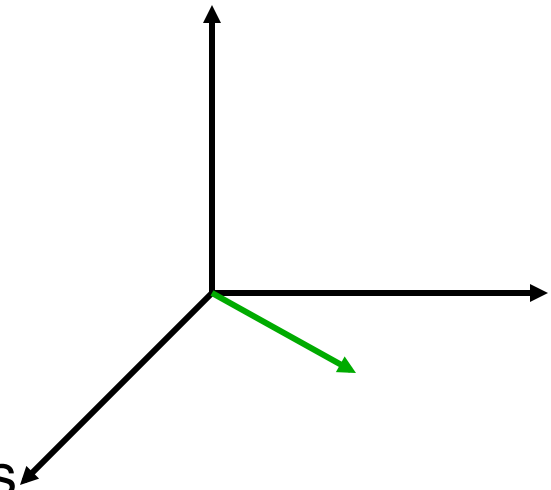
# A nonlinear problem



- A linear classifier like Naïve Bayes does badly on this task
- kNN will do very well (assuming enough training data)

## High Dimensional Data

- Pictures like the one at right are absolutely misleading!
- Most document pairs are very far apart (i.e., not strictly orthogonal, but only share very common words and a few scattered others)
- In classification terms: often document sets are separable, for most any classification
- This is part of why linear classifiers are quite successful in this domain



# More Than Two Classes

- **Any-of** or **multivalued** classification
  - ★ Classes are independent of each other.
  - ★ A document can belong to 0, 1, or  $>1$  classes.
  - ★ Decompose into  $n$  binary problems
  - ★ Quite common for documents
- **One-of** or **multinomial** or **polytomous** classification
  - ★ Classes are mutually exclusive.
  - ★ Each document belongs to exactly one class
  - ★ E.g., digit recognition is polytomous classification
    - ✓ Digits are mutually exclusive
  - ★ True one-of problems are less common in text classification than any-of problems

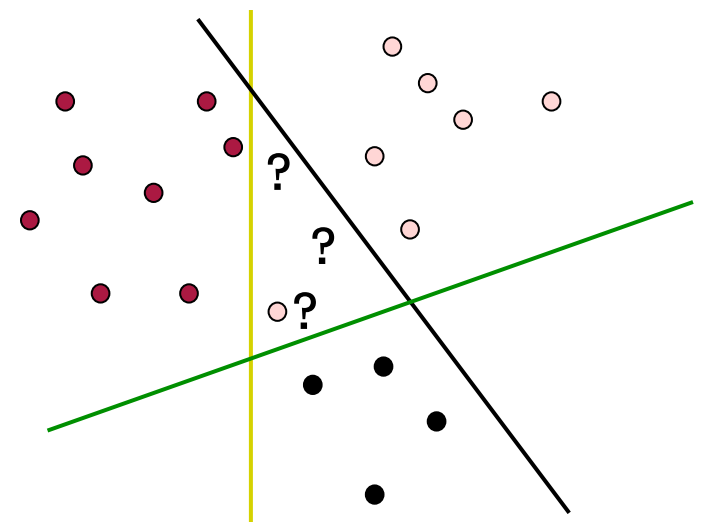
# Set of Binary Classifiers: Any of

- Build a separator between each class and its complementary set (docs from all other classes).
- Given test doc, evaluate it for membership in each class.
- Apply decision criterion of classifiers independently
- Done

★ Though maybe you could do better by considering dependencies between categories

# Set of Binary Classifiers: One of

- Build a separator between each class and its complementary set (docs from all other classes).
- Given test doc, evaluate it for membership in each class.
- Assign document to class with:
  - ★ maximum score
  - ★ maximum confidence
  - ★ maximum probability



# Summary: Representation of Text Categorization Attributes

- Representations of text are usually very high dimensional (one feature for each word)
- High-bias algorithms that prevent overfitting in high-dimensional space should generally work best\*
- For most text categorization tasks, there are many relevant features and many irrelevant ones
- Methods that combine evidence from many or all features (e.g. naive Bayes, kNN) often tend to work better than ones that try to isolate just a few relevant features\*

\*Although the results are a bit more mixed than often thought

# Which classifier do I use for a given text classification problem?

- Is there a learning method that is optimal for all text classification problems?
  - ★ No, because there is a tradeoff between bias and variance.
- Factors to take into account:
  - ★ How much training data is available?
  - ★ How simple/complex is the problem? (linear vs. nonlinear decision boundary)
  - ★ How noisy is the data?
  - ★ How stable is the problem over time?
    - ✓ For an unstable problem, it's better to use a simple and robust classifier.