Text Categorization

Categorization

- Given:
  - A description of an instance, $x \in X$, where X is the *instance language* or *instance space*.
  - A fixed set of categories: $C = \{c_1, c_2, \ldots c_n\}$
- Determine:
  - The category of x: $c(x) \in C$, where $c(x)$ is a categorization function whose domain is X and whose range is C.

Learning for Categorization

- A training example is an instance $x \in X$, paired with its correct category $c(x)$: $<x, c(x)>$ for an unknown categorization function, c.
- Given a set of training examples, D.
- Find a hypothesized categorization function, $h(x)$, such that:

$$\forall < x, c(x) > \in D : h(x) = c(x)$$

Consistency

Sample Category Learning Problem

- Instance language: <size, color, shape>
  - size ∈ {small, medium, large}
  - color ∈ {red, blue, green}
  - shape ∈ {square, circle, triangle}
- C = {positive, negative}
- D:

Example	Size	Color	Shape	Category
1	small	red	circle	positive
2	large	red	circle	positive
3	small	red	triangle	negative
4	large	blue	circle	negative

General Learning Issues

- Many hypotheses are usually consistent with the training data.
- Bias
  - Any criteria other than consistency with the training data that is used to select a hypothesis.
- Classification accuracy (% of instances classified correctly).
  - Measured on independent test data.
- Training time (efficiency of training algorithm).
- Testing time (efficiency of subsequent classification).

Generalization

- Hypotheses must generalize to correctly classify instances not in the training data.
- Simply memorizing training examples is a consistent hypothesis that does not generalize.
- *Occam's razor*:
  - Finding a *simple* hypothesis helps ensure generalization.

Text Categorization

- Assigning documents to a fixed set of categories.
- Applications:
  - Web pages
    - Recommending
    - Yahoo-like classification
  - Newsgroup Messages
    - Recommending
    - spam filtering
  - News articles
    - Personalized newspaper
  - Email messages
    - Routing
    - Prioritizing
    - Folderizing
    - spam filtering

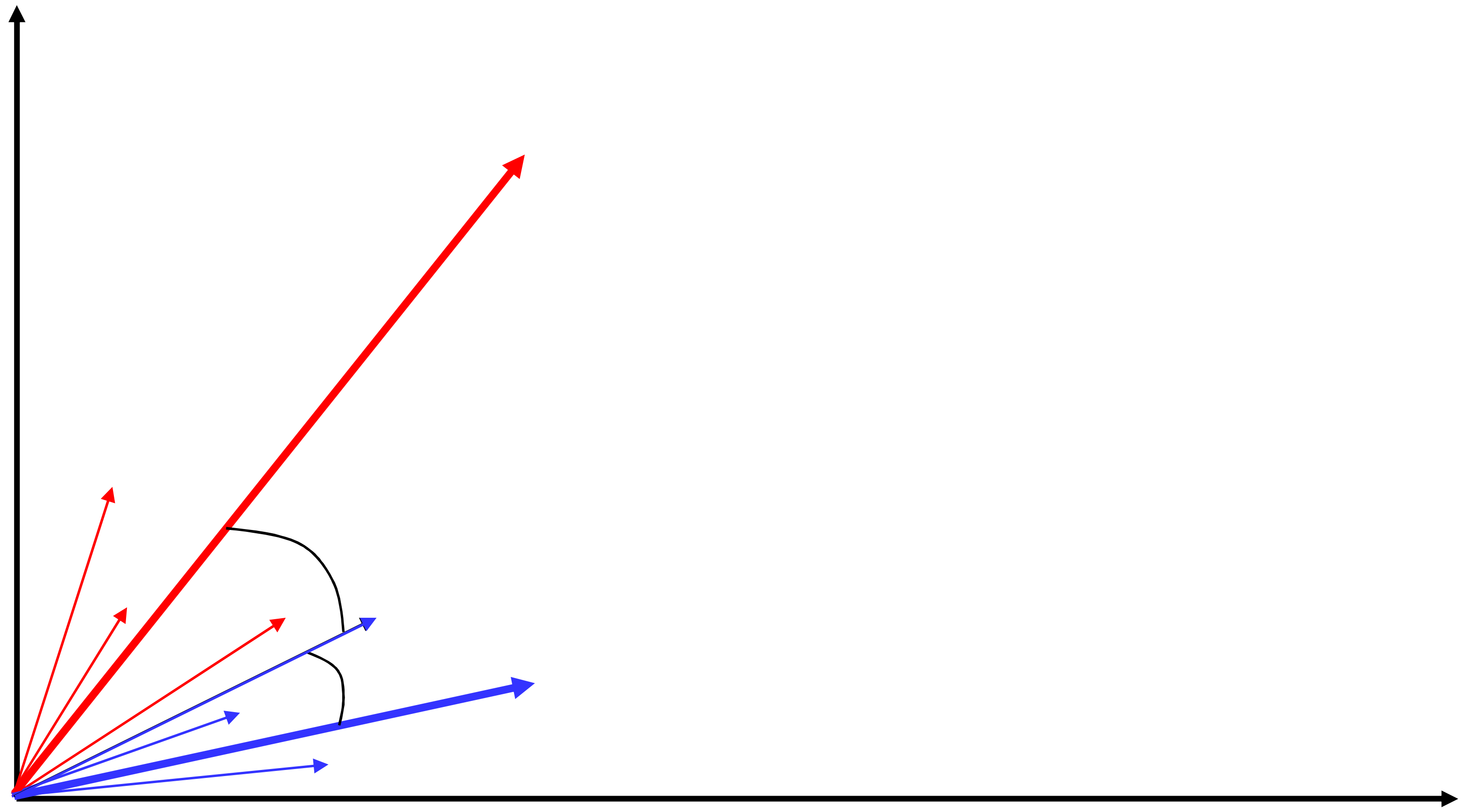
Learning for Text Categorization

- Manual development of text categorization functions is difficult.
- Learning Algorithms:
  - **Bayesian (naïve)**
  - Neural network
  - **Relevance Feedback (Rocchio)**
  - Rule based (Ripper)
  - **Nearest Neighbor (case based)**
  - Support Vector Machines (SVM)

Using Relevance Feedback (Rocchio)

- Relevance feedback methods can be adapted for text categorization.
- Use standard TF/IDF weighted vectors to represent text documents (normalized by maximum term frequency).
- For each category, compute a *prototype* vector by summing the vectors of the training documents in the category.
- Assign test documents to the category with the closest prototype vector based on cosine similarity.

Illustration of Rocchio Text Categorization



Rocchio Text Categorization Algorithm (Training)

```
Assume the set of categories is $\{c_1, c_2,\ldots c_n\}$
For $i$ from 1 to $n$ let $\mathbf{p}_i = \langle 0, 0,\ldots,0\rangle$  (init. prototype vectors)
For each training example $\langle x, c(x)\rangle \in D$
    Let $\mathbf{d}$ be the frequency normalized TF/IDF term vector for doc $x$
    Let $i = j: (c_j = c(x))$
    (sum all the document vectors in $c_i$ to get $\mathbf{p}_i$)
    Let $\mathbf{p}_i = \mathbf{p}_i + \mathbf{d}$
```

Rocchio Text Categorization Algorithm (Test)

```
Given test document $x$
Let $\mathbf{d}$ be the TF/IDF weighted term vector for $x$
Let $m = -2$        (init. maximum cosSim)
For $i$ from 1 to $n$:
    (compute similarity to prototype vector)
    Let $s = \text{cosSim}(\mathbf{d}, \mathbf{p}_i)$
    if $s > m$
        let $m = s$
        let $r = c_i$ (update most similar class prototype)
Return class $r$
```

Rocchio Properties

- Does not guarantee a consistent hypothesis.
- 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.

Rocchio Time Complexity

- Note: The time to add two sparse vectors is proportional to minimum number of non-zero entries in the two vectors.
- Training Time: $O(|D|(L_d + |V_d|)) = O(|D|\ L_d)$
  where L_d is the average length of a document in D and $|V_d|$ is the average vocabulary size for a document in D.
- Test Time: $O(L_t + |C||V_t|)$
  where L_t is the average length of a test document and $|V_t|$ is the average vocabulary size for a test document.
  – Assumes lengths of $\mathbf{p}_i$ vectors are computed and stored during training, allowing cosSim($\mathbf{d}$, $\mathbf{p}_i$) to be computed in time proportional to the number of non-zero entries in $\mathbf{d}$ (i.e. $|V_t|$)

Nearest-Neighbor Learning Algorithm

- Learning is just storing the representations of the training examples in *D*.
- Testing instance *x*:
  - 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
  - Memory-based
  - Lazy learning

K Nearest-Neighbor

- Using only the closest example to determine categorization is subject to errors due to:
  - A single atypical example.
  - Noise (i.e. 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.

Similarity Metrics

- Nearest neighbor method depends on a similarity (or distance) metric.
- Simplest for continuous *m*-dimensional instance space is *Euclidian 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.

3 Nearest Neighbor Illustration (Euclidian Distance)

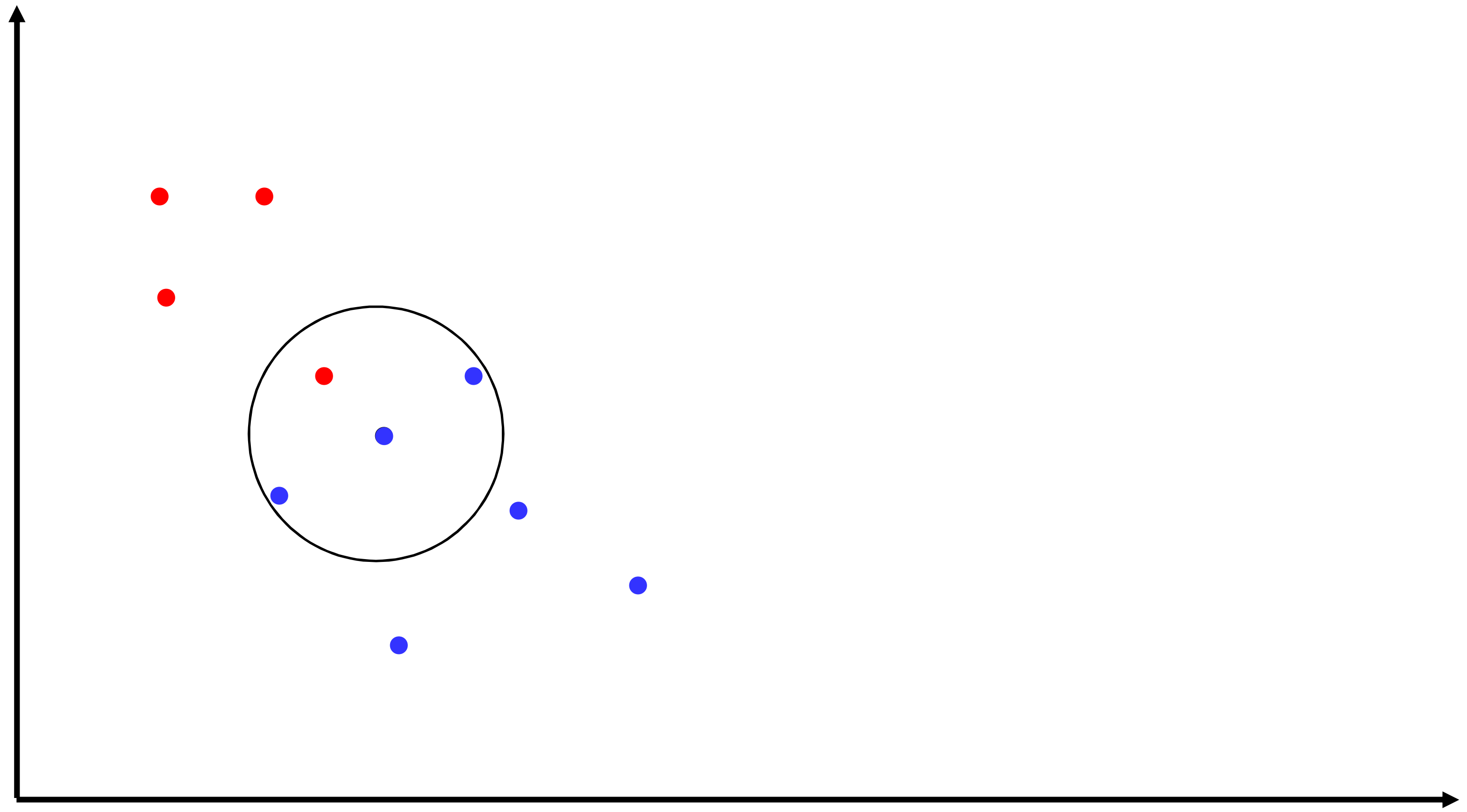
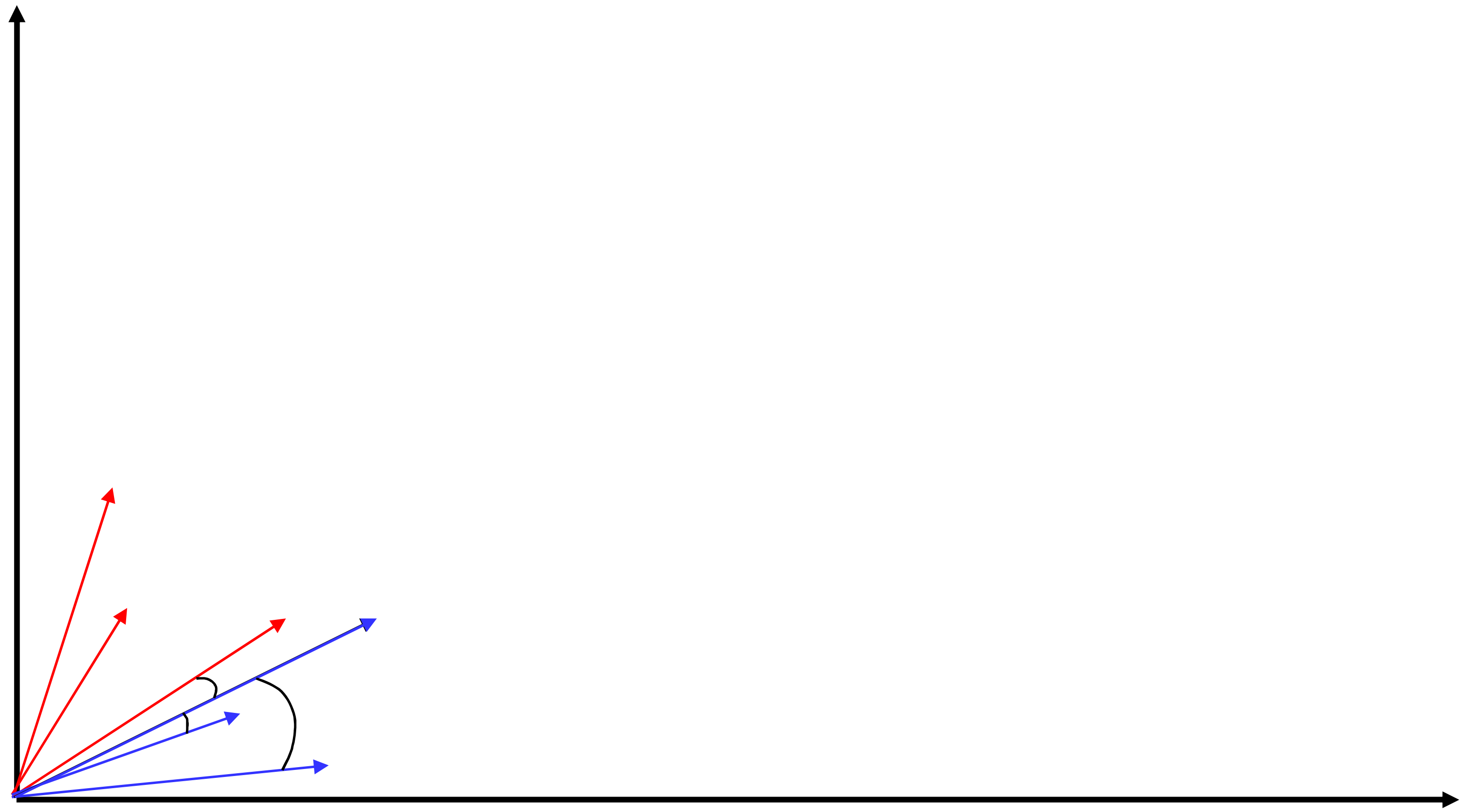


Illustration of 3 Nearest Neighbor for Text



K Nearest Neighbor for Text

Training:

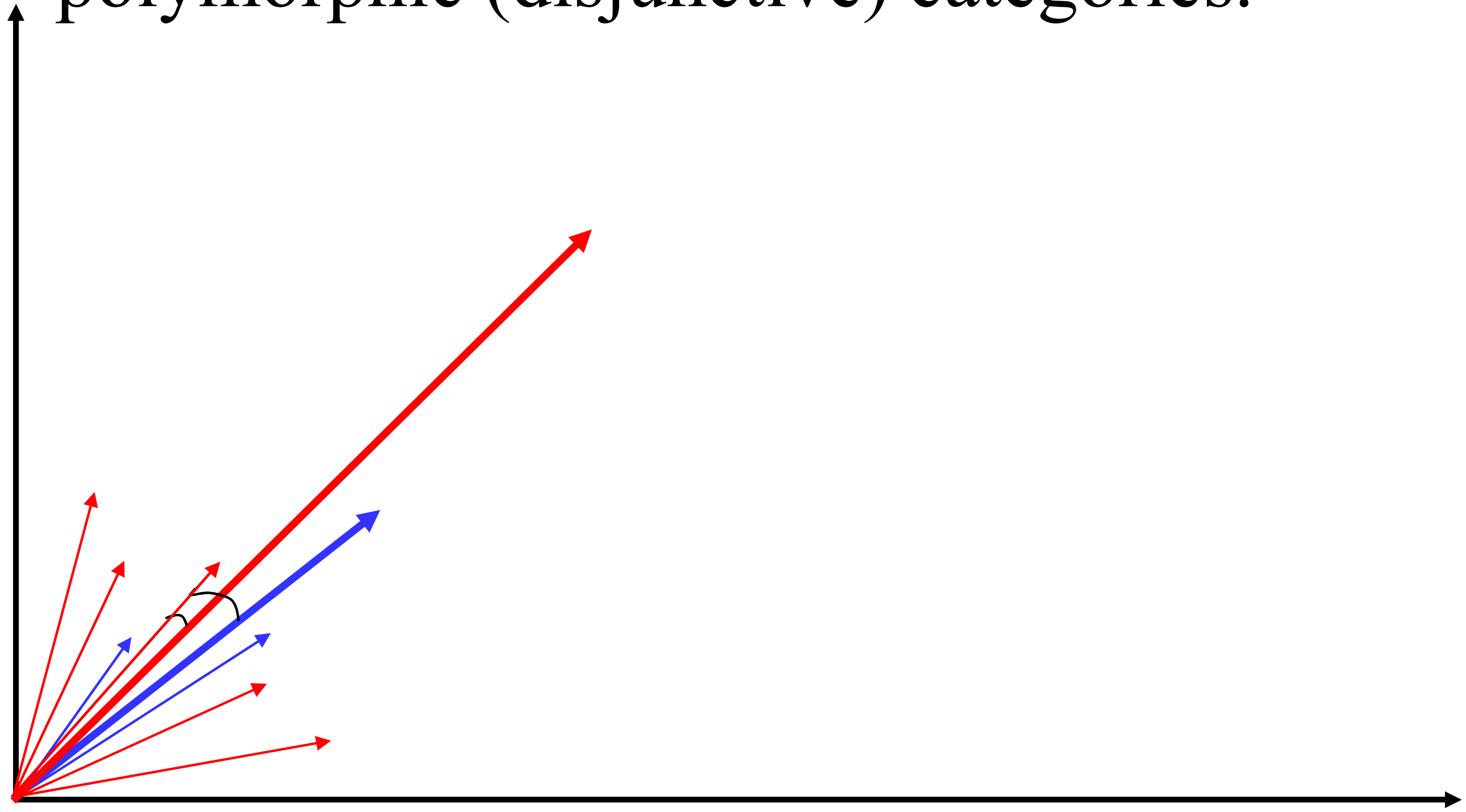
```
For each each training example <$x$, $c(x)$> $\in D$
    Compute the corresponding TF-IDF vector, $\mathbf{d}_x$, for document $x$
```

Test instance *y*:

```
Compute TF-IDF vector $\mathbf{d}$ for document $y$
For each <$x$, $c(x)$> $\in D$
    Let $s_x = \text{cosSim}(\mathbf{d}, \mathbf{d}_x)$
Sort examples, $x$, in $D$ by decreasing value of $s_x$
Let $N$ be the first $k$ examples in D.      (get most similar neighbors)
Return the majority class of examples in $N$
```

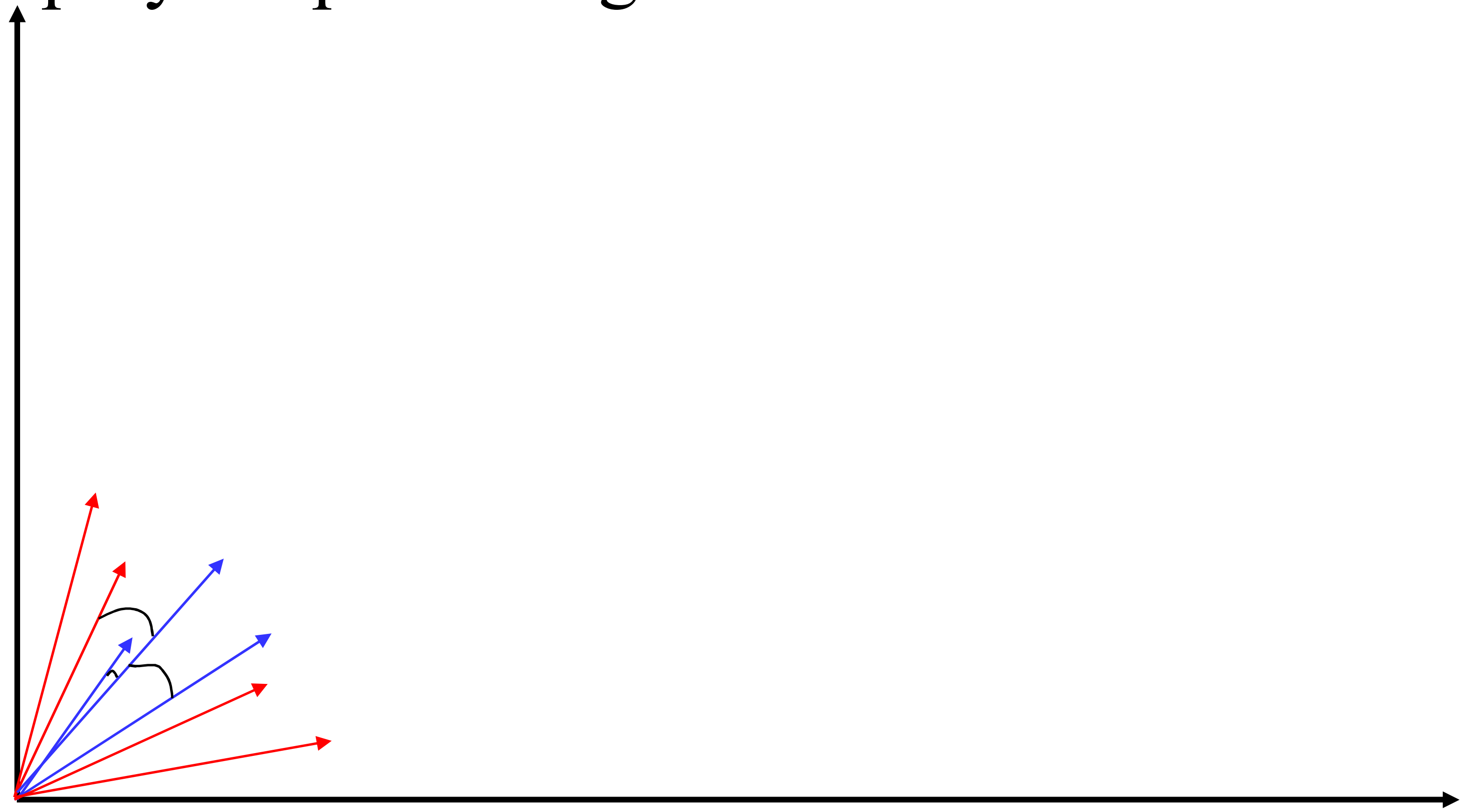
Rocchio Anomoly

- Prototype models have problems with polymorphic (disjunctive) categories.



3 Nearest Neighbor Comparison

- Nearest Neighbor tends to handle polymorphic categories better.



Nearest Neighbor Time Complexity

- Training Time: O($|D|\ L_d$) to compose TF-IDF vectors.
- Testing Time: O($L_t + |D||V_t|$) to compare to all training vectors.
  - Assumes lengths of $\mathbf{d}_x$ vectors are computed and stored during training, allowing cosSim($\mathbf{d}$, $\mathbf{d}_x$) to be computed in time proportional to the number of non-zero entries in $\mathbf{d}$ (i.e. $|V_t|$)
- Testing time can be high for large training sets.

Nearest Neighbor with Inverted Index

- Determining *k* nearest neighbors is the same as determining the *k* best retrievals using the test document as a query to a database of training documents.
- Use standard VSR inverted index methods to find the *k* nearest neighbors.
- Testing Time: $O(B|V_t|)$
  where B is the average number of training documents in which a test-document word appears.
- Therefore, overall classification is $O(L_t + B|V_t|)$
  - Typically $B << |D|$

Bayesian Methods

- Learning and classification methods based on probability theory.
- Bayes theorem plays a critical role in probabilistic learning and classification.
- 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.

Axioms of Probability Theory

- All probabilities between 0 and 1

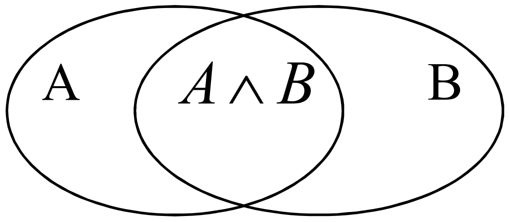
$$0 \le P(A) \le 1$$

- True proposition has probability 1, false has probability 0.

$$P(\text{true}) = 1 \qquad P(\text{false}) = 0.$$

- The probability of disjunction is:

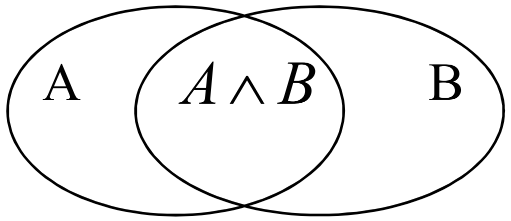
$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$



Conditional Probability

- P(*A* | *B*) is the probability of *A* given *B*
- Assumes that *B* is all and only information known.
- Defined by:

$$P(A \mid B) = \frac{P(A \wedge B)}{P(B)}$$



Independence

- A and B are *independent* iff:

$$P(A \mid B) = P(A)$$

$$P(B \mid A) = P(B)$$

These two constraints are logically equivalent

- Therefore, if A and B are independent:

$$P(A \mid B) = \frac{P(A \wedge B)}{P(B)} = P(A)$$

$$P(A \wedge B) = P(A)P(B)$$

Joint Distribution

- The joint probability distribution for a set of random variables, $X_1,\ldots,X_n$ gives the probability of every combination of values (an n-dimensional array with v^n values if all variables are discrete with v values, all v^n values must sum to 1): $P(X_1,\ldots,X_n)$

positive

	circle	square
red	0.20	0.02
blue	0.02	0.01

negative

	circle	square
red	0.05	0.30
blue	0.20	0.20

- The probability of all possible conjunctions (assignments of values to some subset of variables) can be calculated by summing the appropriate subset of values from the joint distribution.

$$P(red \wedge circle) = 0.20 + 0.05 = 0.25$$
$$P(red) = 0.20 + 0.02 + 0.05 + 0.3 = 0.57$$

- Therefore, all conditional probabilities can also be calculated.

$$P(positive \mid red \wedge circle) = \frac{P(positive \wedge red \wedge circle)}{P(red \wedge circle)} = \frac{0.20}{0.25} = 0.80$$

Probabilistic Classification

- Let Y be the random variable for the class which takes values $\{y_1, y_2, \ldots y_m\}$.
- Let X be the random variable describing an instance consisting of a vector of values for n features $<X_1, X_2 \ldots X_n>$, let x_k be a possible value for X and x_{ij} a possible value for X_i.
- For classification, we need to compute $P(Y=y_i \mid X=x_k)$ for $i=1\ldots m$
- However, given no other assumptions, this requires a table giving the probability of each category for each possible instance in the instance space, which is impossible to accurately estimate from a reasonably-sized training set.
  - Assuming Y and all X_i are binary, we need 2^n entries to specify $P(Y=\text{pos} \mid X=x_k)$ for each of the 2^n possible x_k's since $P(Y=\text{neg} \mid X=x_k) = 1 - P(Y=\text{pos} \mid X=x_k)$
  - Compared to $2^{n+1} - 1$ entries for the joint distribution $P(Y, X_1, X_2 \ldots X_n)$

Bayes Theorem

$$P(H \mid E) = \frac{P(E \mid H)P(H)}{P(E)}$$

Simple proof from definition of conditional probability:

$$P(H \mid E) = \frac{P(H \wedge E)}{P(E)}$$ (Def. cond. prob.)

$$P(E \mid H) = \frac{P(H \wedge E)}{P(H)}$$ (Def. cond. prob.)

$$P(H \wedge E) = P(E \mid H)P(H)$$

QED: $$P(H \mid E) = \frac{P(E \mid H)P(H)}{P(E)}$$

Bayesian Categorization

- Determine category of x_k by determining for each y_i

$$P(Y = y_i \mid X = x_k) = \frac{P(Y = y_i)P(X = x_k \mid Y = y_i)}{P(X = x_k)}$$

- P(X=x_k) can be determined since categories are complete and disjoint.

$$\sum_{i=1}^{m} P(Y = y_i \mid X = x_k) = \sum_{i=1}^{m} \frac{P(Y = y_i)P(X = x_k \mid Y = y_i)}{P(X = x_k)} = 1$$

$$P(X = x_k) = \sum_{i=1}^{m} P(Y = y_i)P(X = x_k \mid Y = y_i)$$

Bayesian Categorization (cont.)

- Need to know:
  - Priors: $P(Y=y_i)$
  - Conditionals: $P(X=x_k \mid Y=y_i)$
- $P(Y=y_i)$ are easily estimated from data.
  - If n_i of the examples in D are in y_i then $P(Y=y_i) = n_i / |D|$
- Too many possible instances (e.g. 2^n for binary features) to estimate all $P(X=x_k \mid Y=y_i)$.
- Still need to make some sort of independence assumptions about the features to make learning tractable.

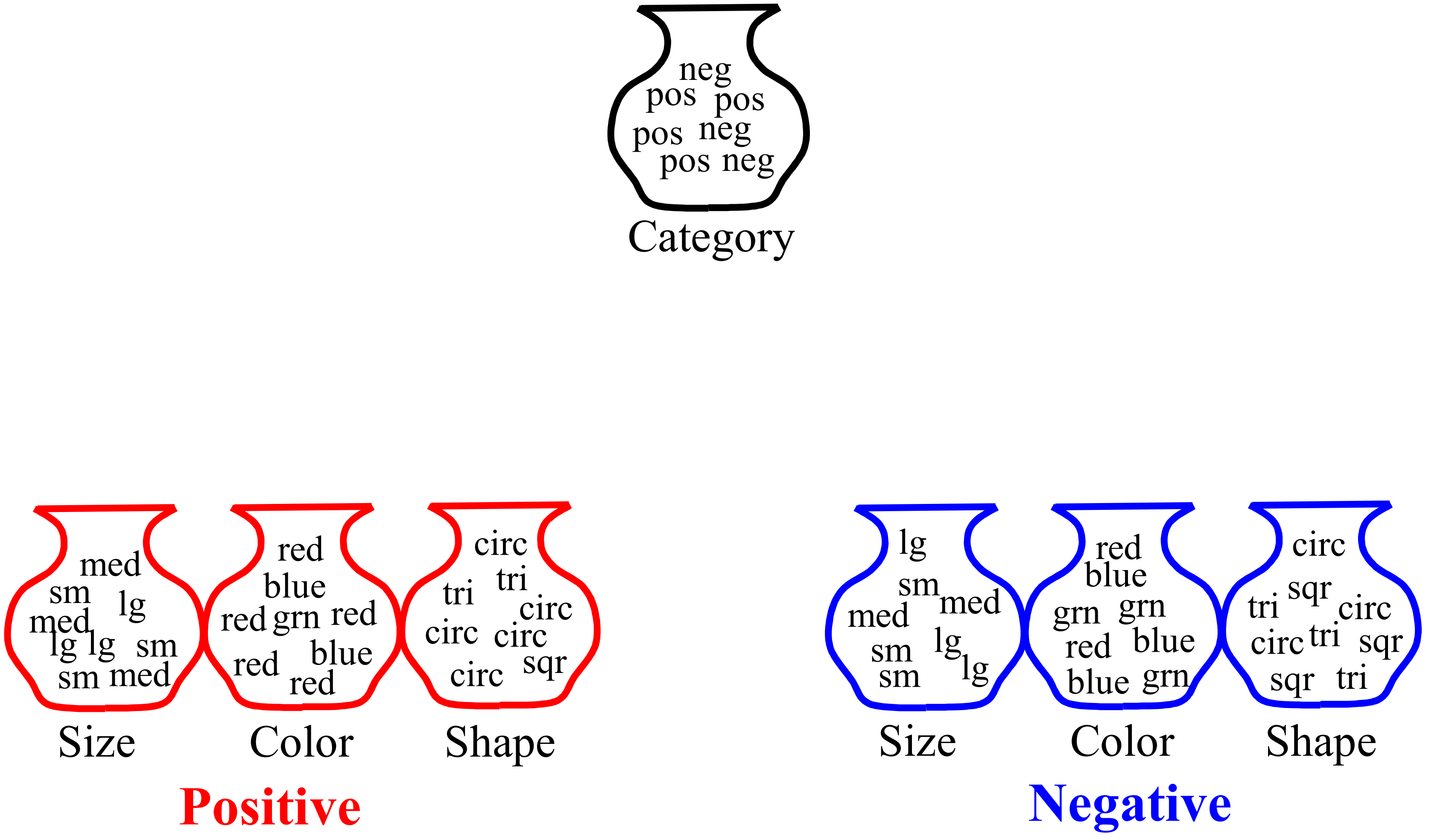
Generative Probabilistic Models

- Assume a simple (usually unrealistic) probabilistic method by which the data was generated.
- For categorization, each category has a different parameterized generative model that characterizes that category.
- **Training**: Use the data for each category to estimate the parameters of the generative model for that category.
  - **Maximum Likelihood Estimation (MLE)**: Set parameters to maximize the probability that the model produced the given training data.
  - If M_λ denotes a model with parameter values λ and D_k is the training data for the *k*th class, find model parameters for class *k* (λ_k) that maximize the likelihood of D_k:

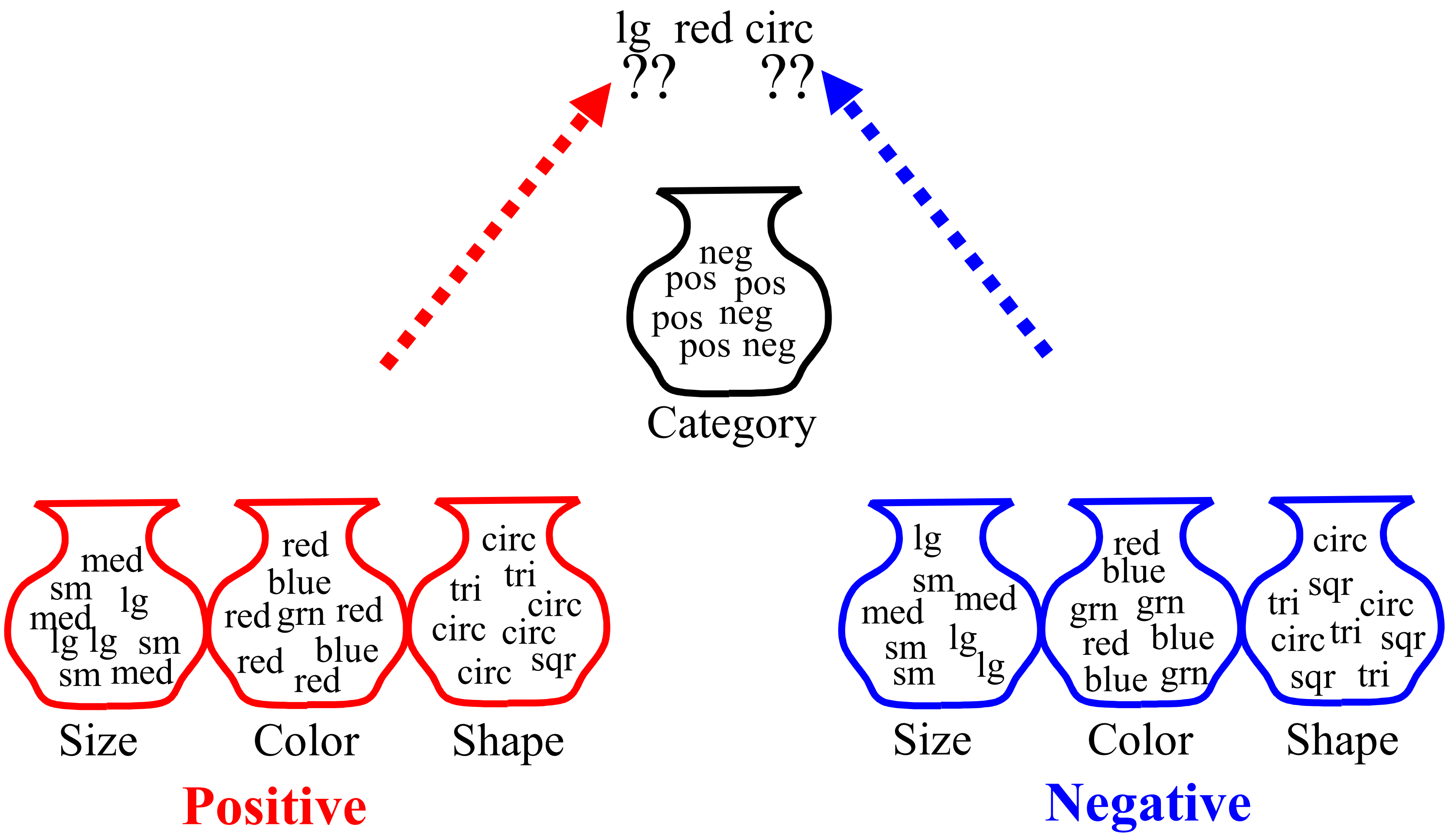
$$\lambda_k = \underset{\lambda}{\operatorname{argmax}} P(D_k \mid M_\lambda)$$

- **Testing**: Use Bayesian analysis to determine the category model that most likely generated a specific test instance.

Naïve Bayes Generative Model



Naïve Bayes Inference Problem



Naïve Bayesian Categorization

- If we assume features of an instance are independent **given the category** (***conditionally independent***).

$$P(X \mid Y) = P(X_1, X_2, \cdots X_n \mid Y) = \prod_{i=1}^{n} P(X_i \mid Y)$$

- Therefore, we then only need to know P(X_i | Y) for each possible pair of a feature-value and a category.
- If Y and all X_i and binary, this requires specifying only 2n parameters:
  - P(X_i=true | Y=true) and P(X_i=true | Y=false) for each X_i
  - P(X_i=false | Y) = 1 – P(X_i=true | Y)

- Compared to specifying 2^n parameters without any independence assumptions.

Naïve Bayes Example

Probability	positive	negative
P(Y)	0.5	0.5
P(small | Y)	0.4	0.4
P(medium | Y)	0.1	0.2
P(large | Y)	0.5	0.4
P(red | Y)	0.9	0.3
P(blue | Y)	0.05	0.3
P(green | Y)	0.05	0.4
P(square | Y)	0.05	0.4
P(triangle | Y)	0.05	0.3
P(circle | Y)	0.9	0.3

Test Instance:
<medium ,red, circle>

Naïve Bayes Example

Probability	positive	negative
P(*Y*)	0.5	0.5
P(medium | *Y*)	0.1	0.2
P(red | *Y*)	0.9	0.3
P(circle | *Y*)	0.9	0.3

Test Instance:
<medium, red, circle>

P(positive | *X*) = P(positive)*P(medium | positive)*P(red | positive)*P(circle | positive) / P(*X*)
0.5 * 0.1 * 0.9 * 0.9
= 0.0405 / P(*X*) = 0.0405 / 0.0495 = 0.8181

P(negative | *X*) = P(negative)*P(medium | negative)*P(red | negative)*P(circle | negative) / P(*X*)
0.5 * 0.2 * 0.3 * 0.3
= 0.009 / P(*X*) = 0.009 / 0.0495 = 0.1818

P(positive | *X*) + P(negative | *X*) = 0.0405 / P(*X*) + 0.009 / P(*X*) = 1

P(*X*) = (0.0405 + 0.009) = 0.0495

Estimating Probabilities

- Normally, probabilities are estimated based on observed frequencies in the training data.
- If D contains n_k examples in category y_k, and n_{ijk} of these n_k examples have the jth value for feature X_i, x_{ij}, then:

$$P(X_i = x_{ij} \mid Y = y_k) = \frac{n_{ijk}}{n_k}$$

- However, estimating such probabilities from small training sets is error-prone.
- If due only to chance, a rare feature, X_i, is always false in the training data, $\forall y_k$:P(X_i=true | Y=y_k) = 0.
- If X_i=true then occurs in a test example, X, the result is that $\forall$y$_k$: P(X | Y=y$_k$) = 0 and $\forall$y$_k$: P(Y=y$_k$ | X) = 0

Probability Estimation Example

Ex	Size	Color	Shape	Category
1	small	red	circle	positive
2	large	red	circle	positive
3	small	red	triangle	negative
4	large	blue	circle	negative

Probability	positive	negative
P(*Y*)	0.5	0.5
P(small | *Y*)	0.5	0.5
P(medium | *Y*)	0.0	0.0
P(large | *Y*)	0.5	0.5
P(red | *Y*)	1.0	0.5
P(blue | *Y*)	0.0	0.5
P(green | *Y*)	0.0	0.0
P(square | *Y*)	0.0	0.0
P(triangle | *Y*)	0.0	0.5
P(circle | *Y*)	1.0	0.5

Test Instance *X*:
<medium, red, circle>

$P(\text{positive} \mid X) = 0.5 * 0.0 * 1.0 * 1.0 / P(X) = 0$

$P(\text{negative} \mid X) = 0.5 * 0.0 * 0.5 * 0.5 / P(X) = 0$

Smoothing

- To account for estimation from small samples, probability estimates are adjusted or *smoothed*.
- Laplace smoothing using an *m*-estimate assumes that each feature is given a prior probability, *p*, that is assumed to have been previously observed in a "virtual" sample of size *m*.

$$P(X_i = x_{ij} \mid Y = y_k) = \frac{n_{ijk} + mp}{n_k + m}$$

- For binary features, *p* is simply assumed to be 0.5.

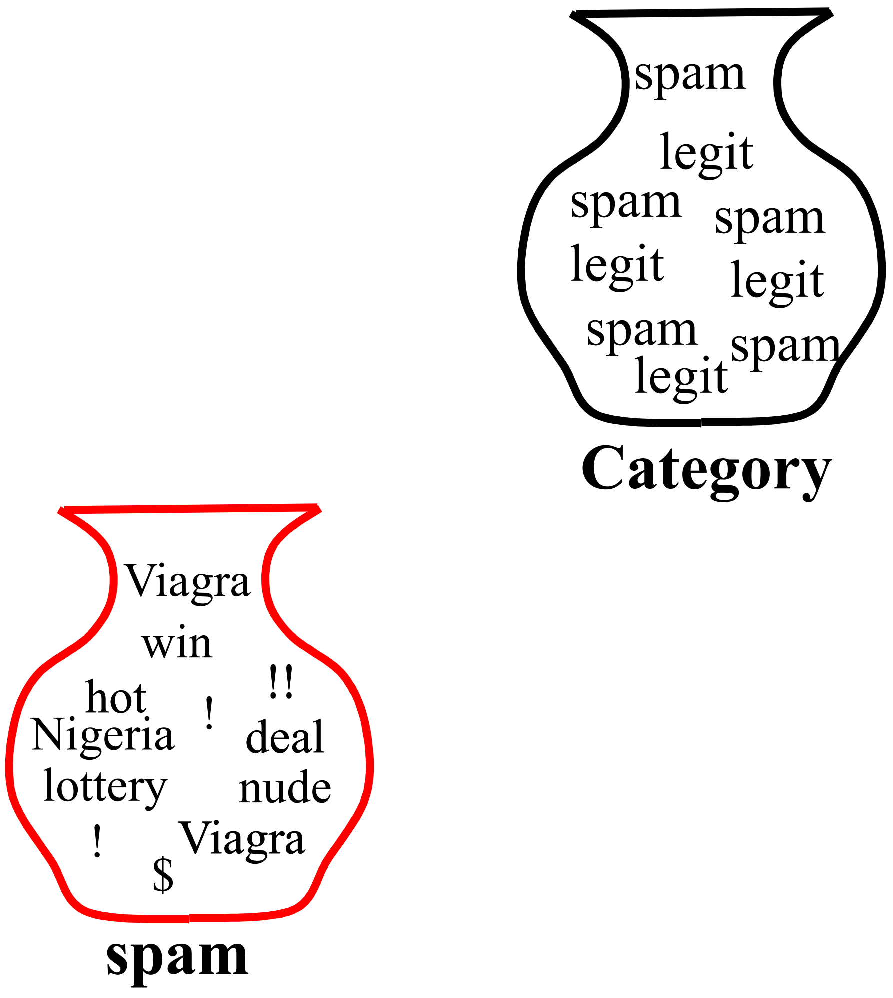
Laplace Smothing Example

- Assume training set contains 10 positive examples:
  - 4: small
  - 0: medium
  - 6: large
- Estimate parameters as follows (if *m*=1, *p*=1/3)
  - P(small | positive) = (4 + 1/3) / (10 + 1) = 0.394
  - P(medium | positive) = (0 + 1/3) / (10 + 1) = 0.03
  - P(large | positive) = (6 + 1/3) / (10 + 1) = 0.576
  - P(small or medium or large | positive) = 1.0

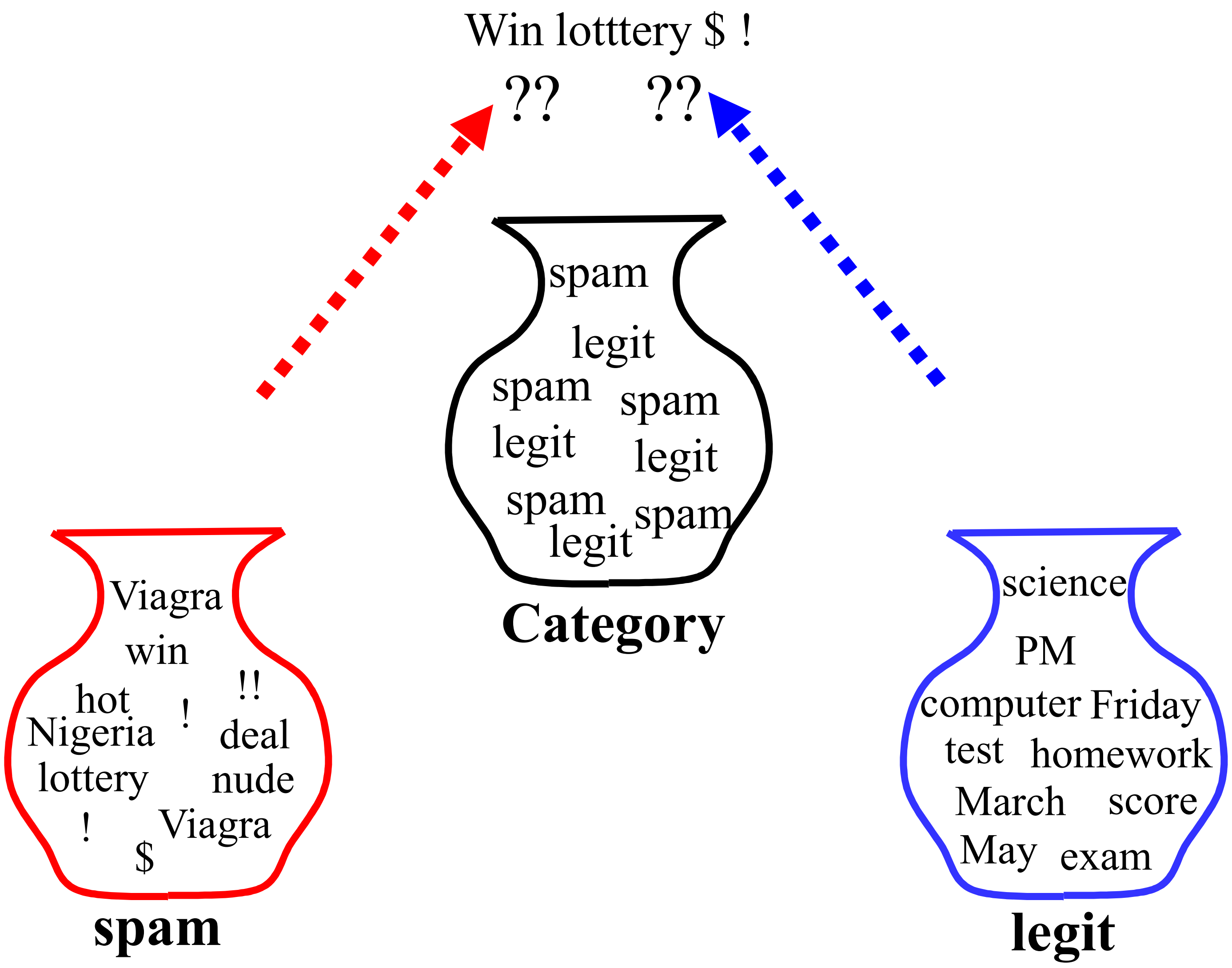
Naïve Bayes for Text

- Modeled as generating a bag of words for a document in a given category by repeatedly sampling with replacement from a vocabulary $V = \{w_1, w_2, \ldots w_m\}$ based on the probabilities $P(w_j \mid c_i)$.
- Smooth probability estimates with Laplace m-estimates assuming a uniform distribution over all words ($p = 1/|V|$) and $m = |V|$
  - Equivalent to a virtual sample of seeing each word in each category exactly once.

Naïve Bayes Generative Model for Text



Naïve Bayes Classification



Text Naïve Bayes Algorithm (Train)

```
Let $V$ be the vocabulary of all words in the documents in $D$
For each category $c_i \in C$
    Let $D_i$ be the subset of documents in $D$ in category $c_i$
    $P(c_i) = |D_i| / |D|$
    Let $T_i$ be the concatenation of all the documents in $D_i$
    Let $n_i$ be the total number of word occurrences in $T_i$
    For each word $w_j \in V$
        Let $n_{ij}$ be the number of occurrences of $w_j$ in $T_i$
        Let $P(w_j \mid c_i) = (n_{ij} + 1) / (n_i + |V|)$
```

Text Naïve Bayes Algorithm (Test)

Given a test document X

Let n be the number of word occurrences in X

Return the category:

$$\underset{c_i \in C}{\operatorname{argmax}}\, P(c_i) \prod_{i=1}^{n} P(a_i \mid c_i)$$

where a_i is the word occurring the ith position in X

Underflow Prevention

- 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.

Naïve Bayes Posterior Probabilities

- Classification results of naïve Bayes (the class with maximum posterior probability) are usually fairly accurate.
- However, due to the inadequacy of the conditional independence assumption, the actual posterior-probability numerical estimates are not.
  – Output probabilities are generally very close to 0 or 1.

Evaluating Categorization

- Evaluation must be done on test data that are independent of the training data (usually a disjoint set of instances).
- *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.
- Results can vary based on sampling error due to different training and test sets.
- Average results over multiple training and test sets (splits of the overall data) for the best results.

N-Fold Cross-Validation

- Ideally, test and training sets are independent on each trial.
  - But this would require too much labeled data.
- Partition data into *N* equal-sized disjoint segments.
- Run *N* trials, each time using a different segment of the data for testing, and training on the remaining *N*–1 segments.
- This way, at least test-sets are independent.
- Report average classification accuracy over the *N* trials.
- Typically, $N = 10$.

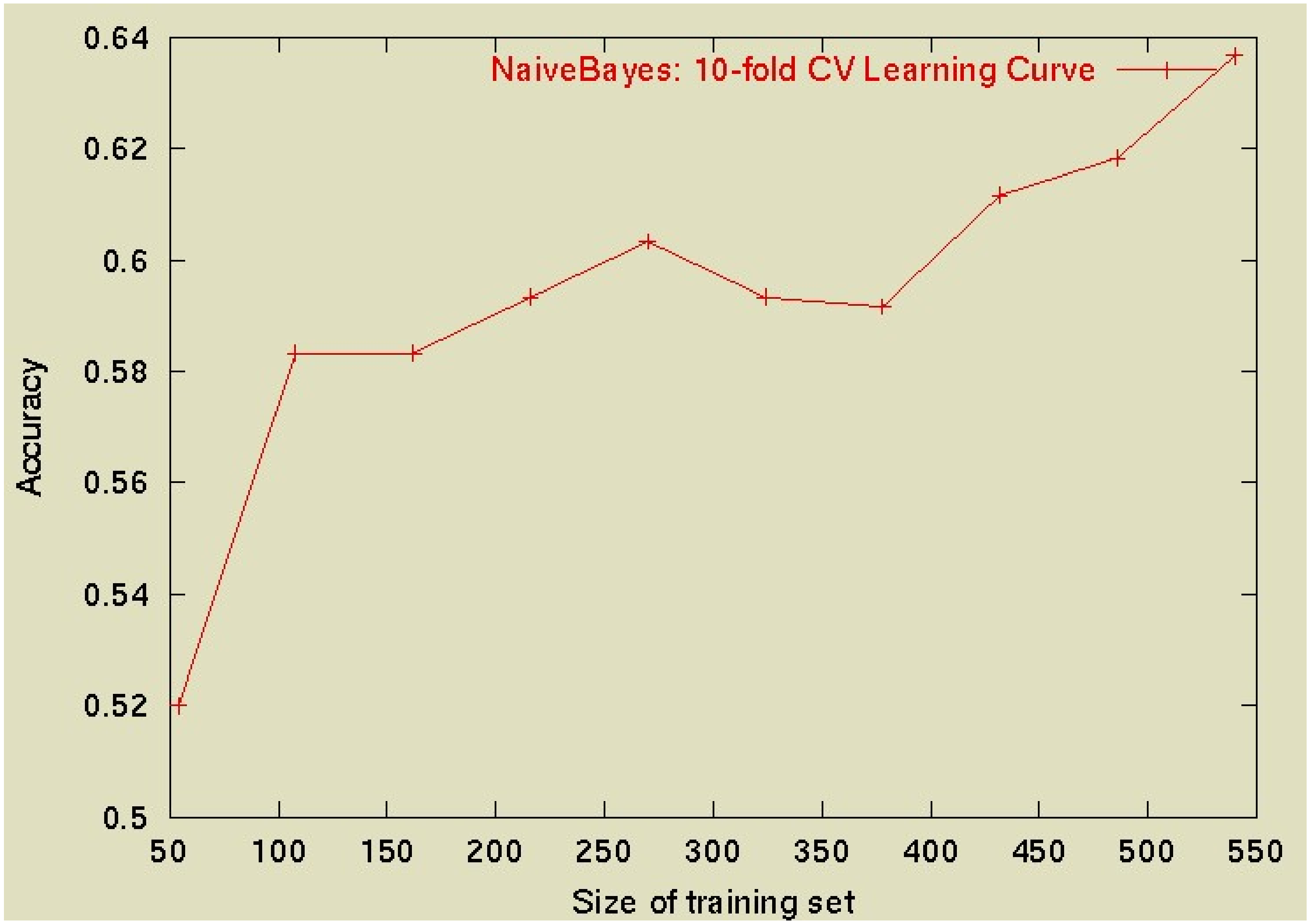
Learning Curves

- In practice, labeled data is usually rare and expensive.
- Would like to know how performance varies with the number of training instances.
- *Learning curves* plot classification accuracy on independent test data (*Y* axis) versus number of training examples (*X* axis).

N-Fold Learning Curves

- Want learning curves averaged over multiple trials.
- Use *N*-fold cross validation to generate *N* full training and test sets.
- For each trial, train on increasing fractions of the training set, measuring accuracy on the test data for each point on the desired learning curve.

Sample Learning Curve (Yahoo Science Data)


NaiveBayes: 10-fold CV Learning Curve
Accuracy
Size of training set