# Classification Algorithms

UCSB 290N, 2015. T. Yang

Slides based on R. Mooney (UT Austin)

# Table of Content

# Classification

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

# Learning for Classification

- A training example is an instance $x$, 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 Learning Problem

- Instance space: <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).

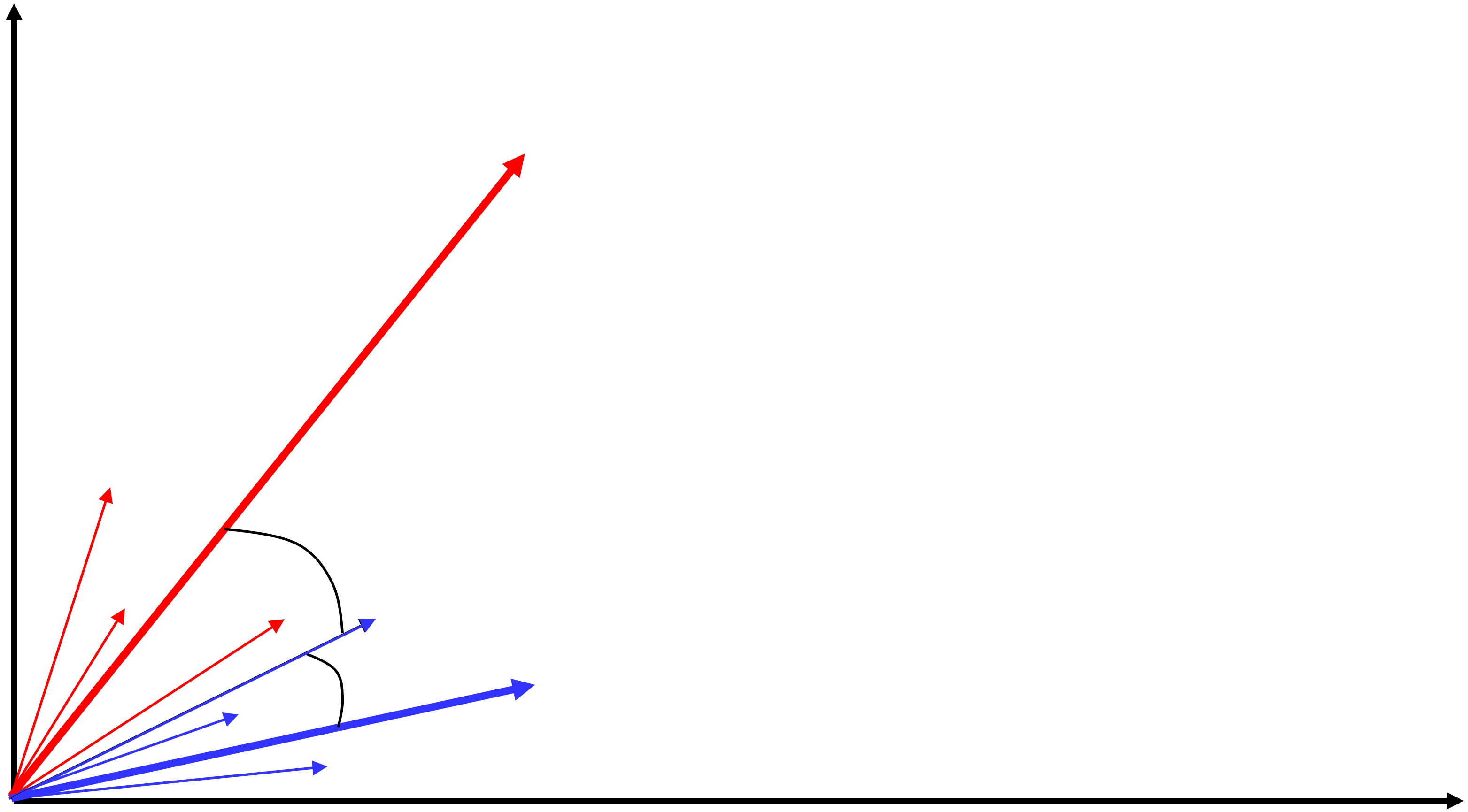
# Text Categorization/Classification

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

# Learning for Classification

- Manual development of text classification functions is difficult.
- Learning Algorithms:
  - **Bayesian (naïve)**
  - Neural network
  - **Rocchio**
  - Rule based (Ripper)
  - **Nearest Neighbor (case based)**
  - Support Vector Machines (SVM)
  - **Decision trees**
  - Boosting algorithms

# Illustration of Rocchio method



# Rocchio Algorithm

Assume the set of categories is $\{c_1, c_2, \ldots c_n\}$

Training:

Each doc vector is the frequency normalized TF/IDF term vector.

For $i$ from 1 to $n$

Sum all the document vectors in $c_i$ to get prototype vector $\mathbf{p}_i$

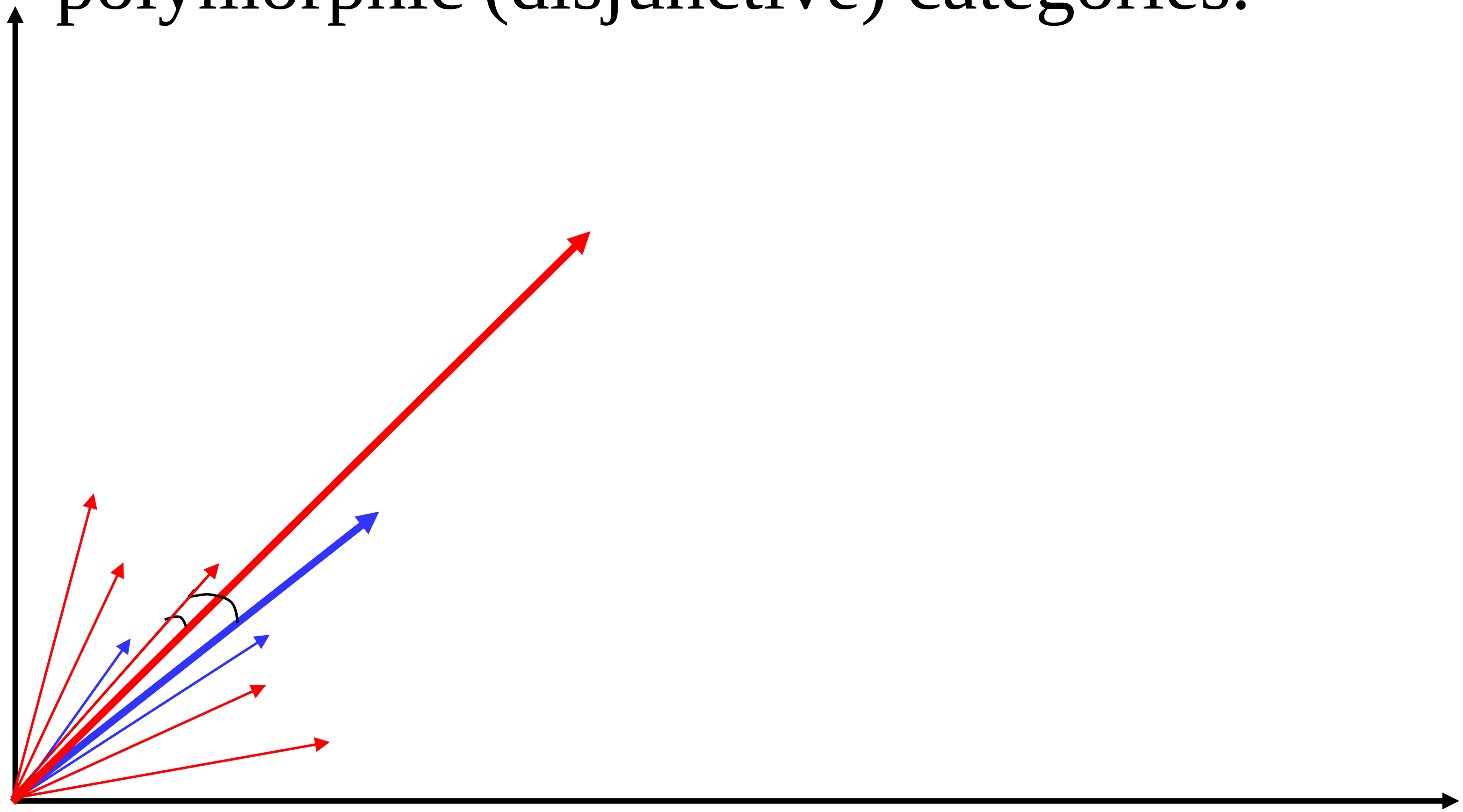
Testing: Given document $x$

Compute the cosine similarity of x with each prototype vector.

Select one with the highest similarity value and return its category

# Rocchio Anomoly

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



# 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

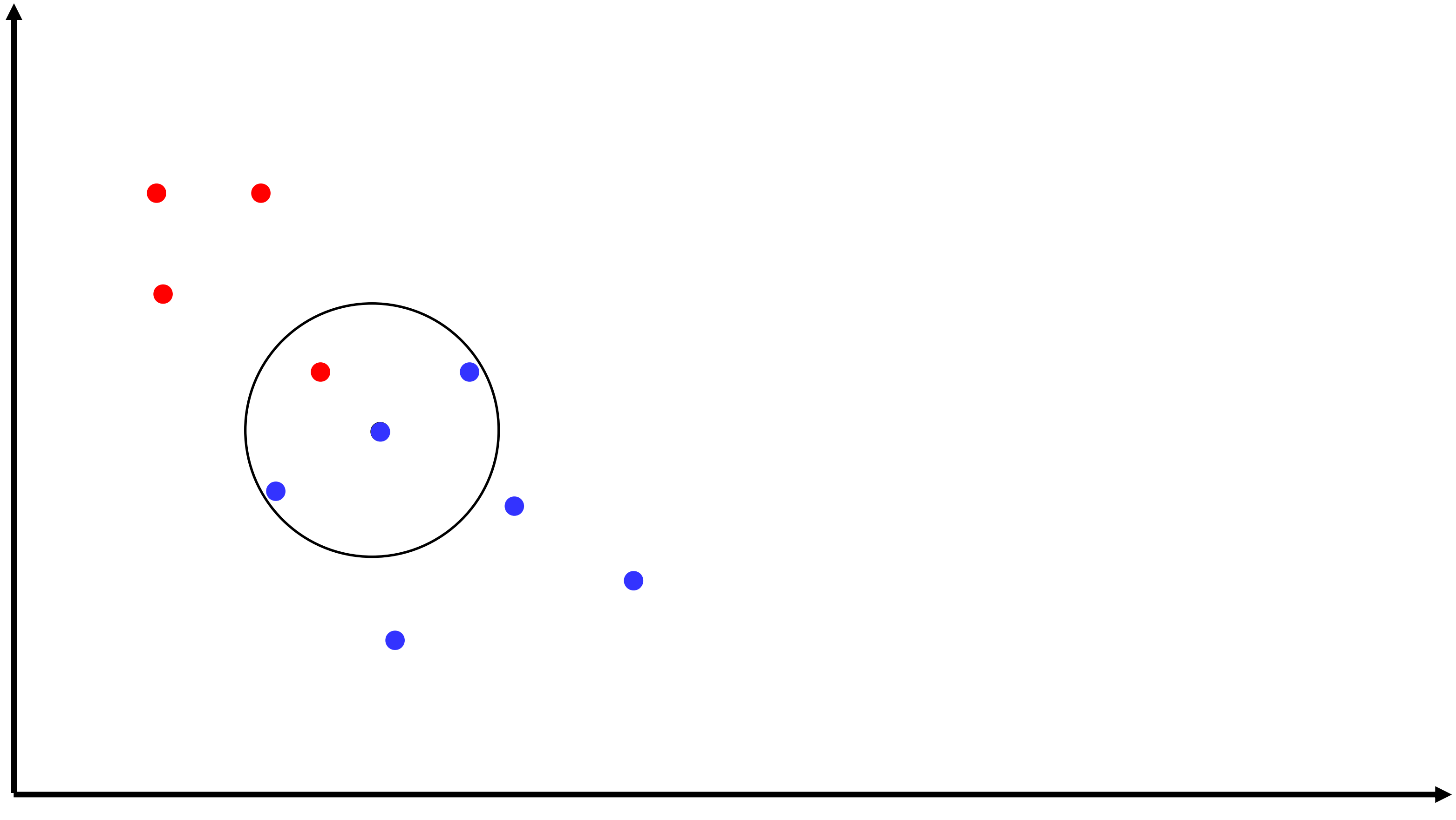
# K Nearest-Neighbor

- 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)



# 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 **d** for document $y$

For each <$x$, $c(x)$> $\in D$

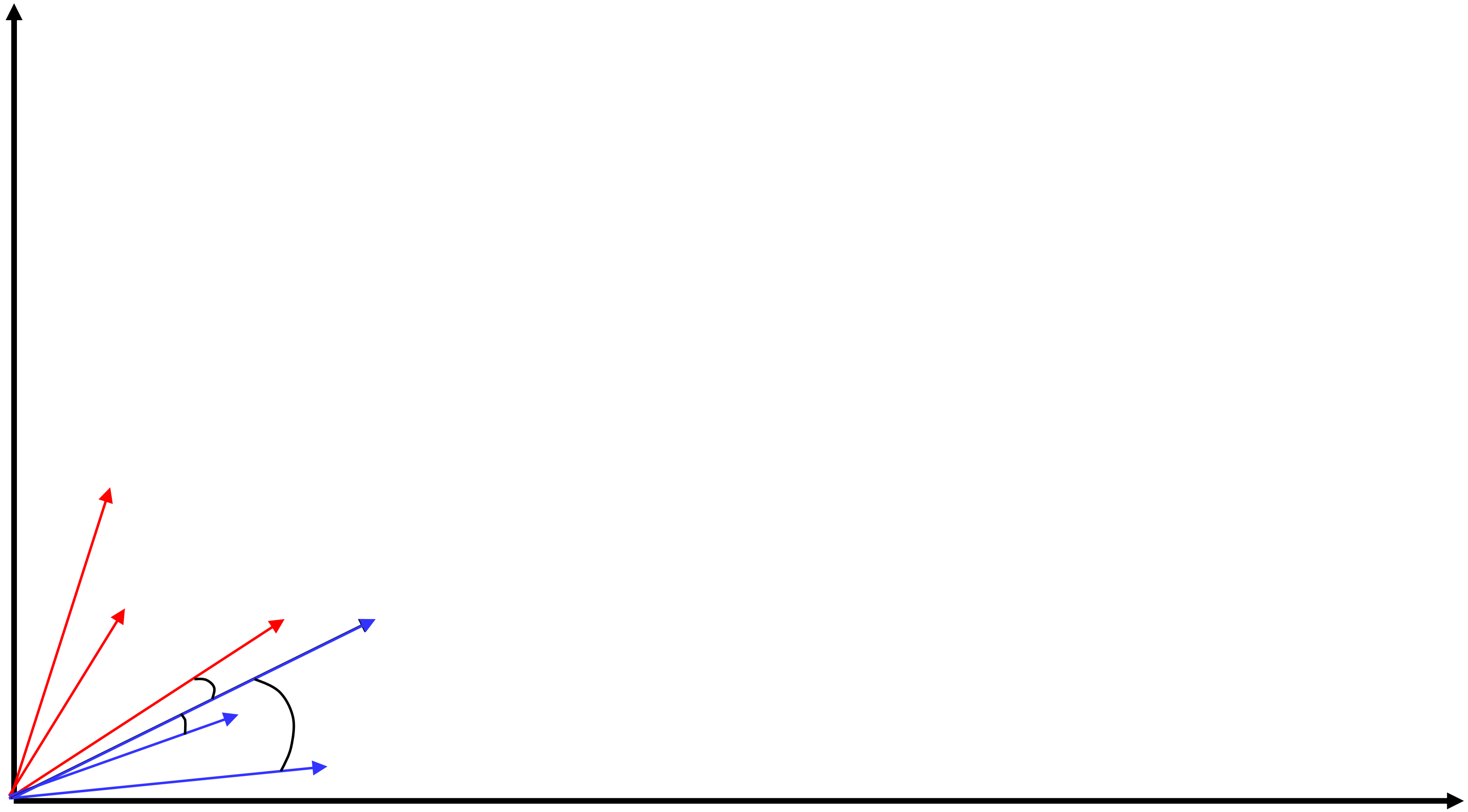
Let $s_x = \text{cosSim}(\mathbf{d}, \mathbf{d}_x)$

Sort examples, $x$, in $D$ by decreasing value of $s_x$

Let $N$ be the first $k$ examples in D. (*get most similar neighbors*)

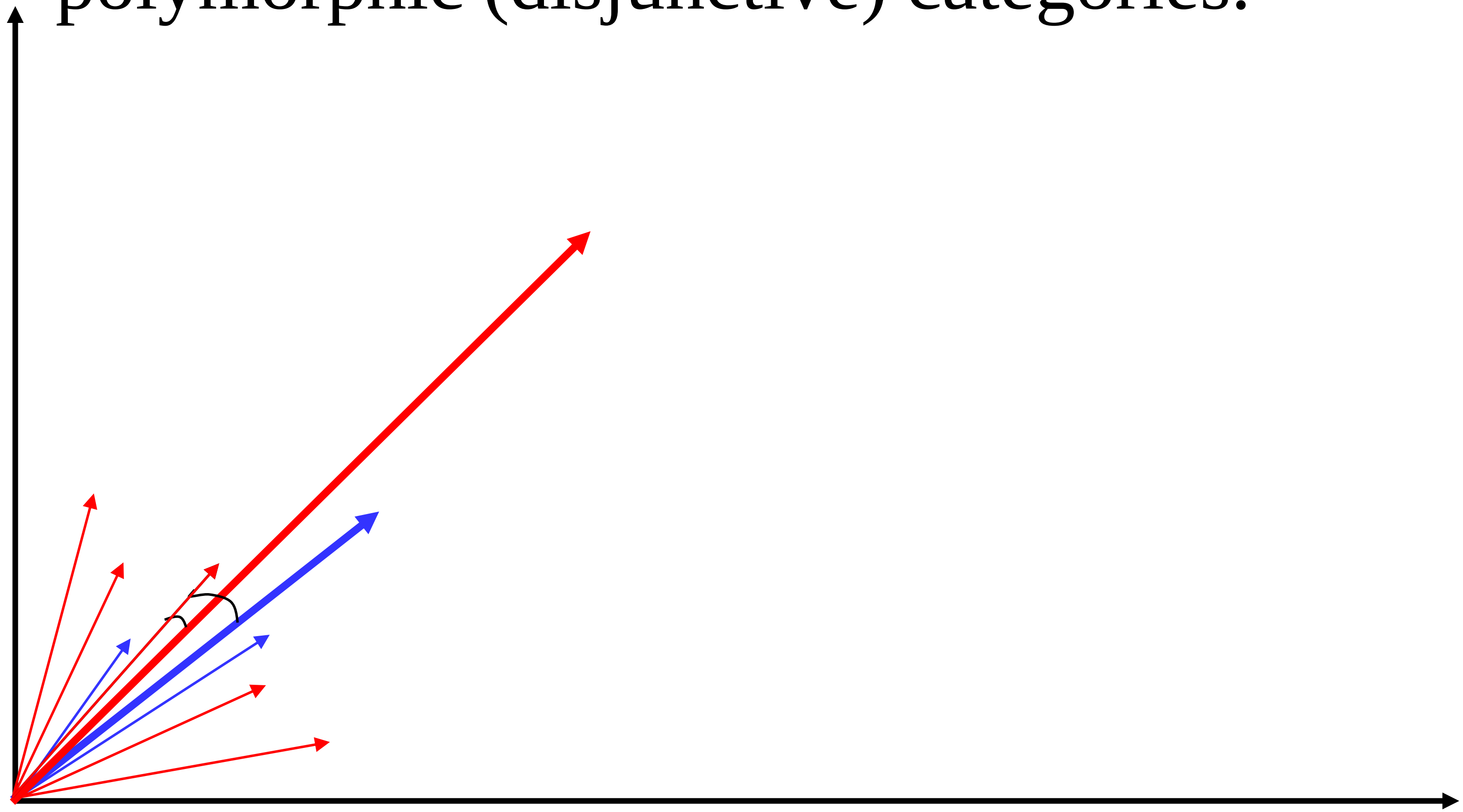
Return the majority class of examples in $N$

# Illustration of 3 Nearest Neighbor for Text



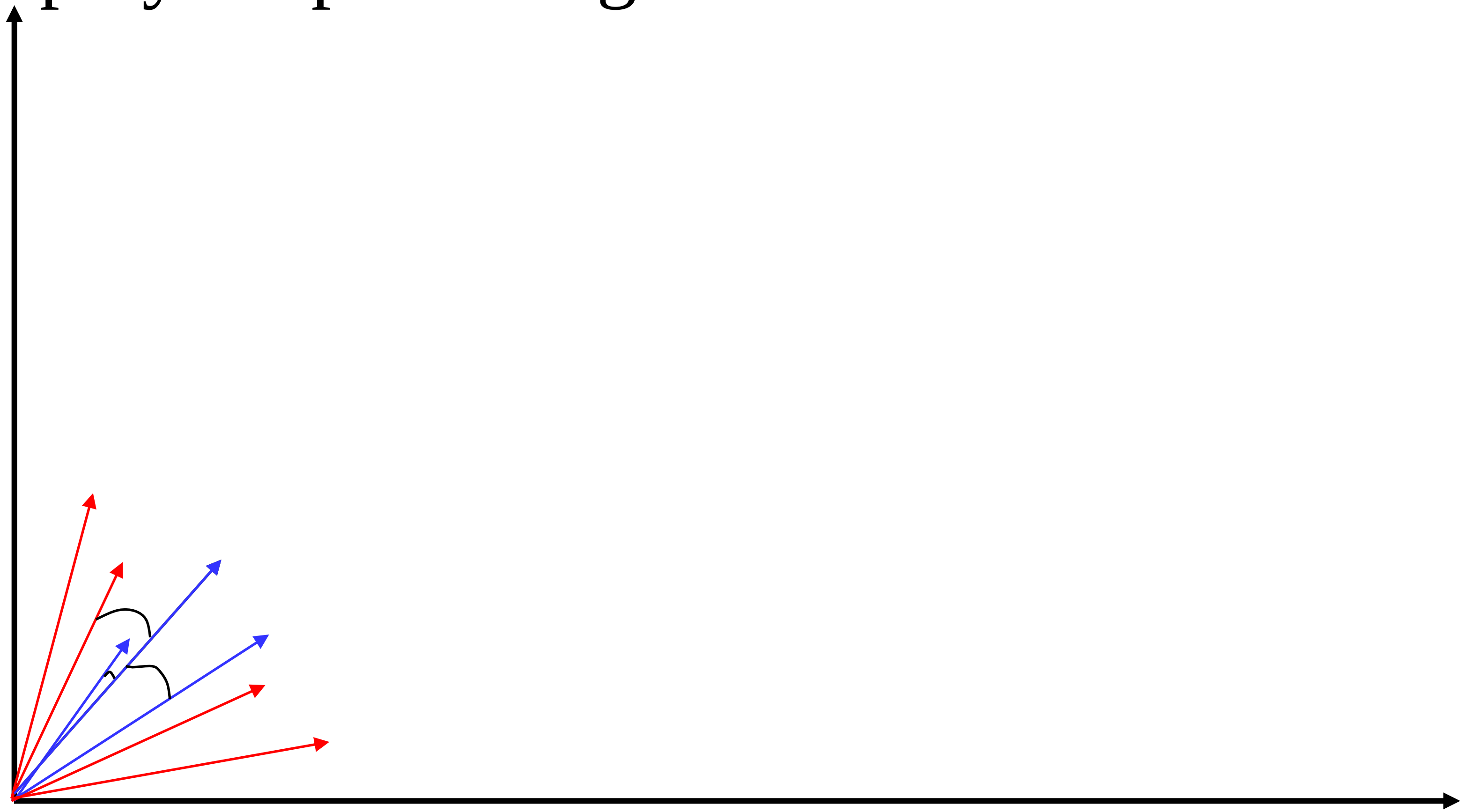
# Rocchio Anomoly

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



# 3 Nearest Neighbor Comparison

- Nearest Neighbor tends to handle polymorphic categories better.



# 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.

# Basic Probability Theory

- All probabilities between 0 and 1

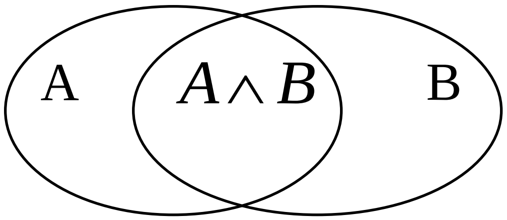
$$0 \leq P(A) \leq 1$$

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

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

- The probability of disjunction is:

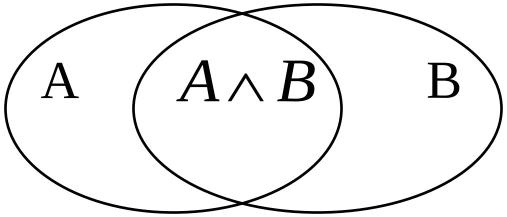
$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$



# Conditional Probability

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

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



# 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

- Joint probability distribution for $X_1,\ldots,X_n$ gives the probability of every combination of values: $P(X_1,\ldots,X_n)$
  - All values must sum to 1.

Category=positive

| Color\shape | 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 |

- Probability for assignments of values to some subset of variables can be calculated by summing the appropriate subset

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

$$P(red) = 0.20 + 0.02 + 0.05 + 0.3 = 0.57$$

- Conditional probabilities can also be calculated.

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

# Computing probability from a training dataset

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

| Probability | Y=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>

# Probabilistic Classification

- Let $Y$ be class variable which takes values $\{y_1, y_2, \ldots y_m\}$.
- Let $X$ describe an instance consisting of $n$ features $<X_1, X_2 \ldots X_n>$, let $x_k$ be a possible value for $X$ and $x_{ij}$ a possible value for $X_i$.
- Given a feature vector $x_k$, classification computes $P(Y=y_i \mid X=x_k)$ for $i=1\ldots m$
- This requires a table giving the probability of each category for each possible instance.
  - Assuming $Y$ and all features $X_i$ 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
    $P(Y=\text{neg} \mid X=x_k) = 1 - P(Y=\text{pos} \mid X=x_k)$

  - How to express prediction concisely?

# Bayes Theorem

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

Simple proof from definition of conditional probability:

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

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

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

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

# 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($X$=$x_k$) estimation is not needed in the algorithm to choose a classification decision via comparison.
- .

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

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

# 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 training data $D$ 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 $P(X=x_k \mid Y=y_i)$ in advance.

# 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.
  - $n_i$ of the examples in training data $D$ are in $y_i$
  - $n_{ij}$ of the examples in $D$ *with category* $y_i$
  - $P(x_{ij} \mid Y{=}y_i) = n_{ij} / n_i$

# Computing probability from a training dataset

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

| Probability | Y=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>

# Naïve Bayes Example

| Probability | Y=positive | Y=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 | Y=positive | Y=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(X/Positive)/P(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

# Error prone prediction with small training data

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

| Probability | Y=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(positive | *X*) = 0.5 * 0.0 * 1.0 * 1.0 = 0

P(negative | *X*) = 0.5 * 0.0 * 0.5 * 0.5 = 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.

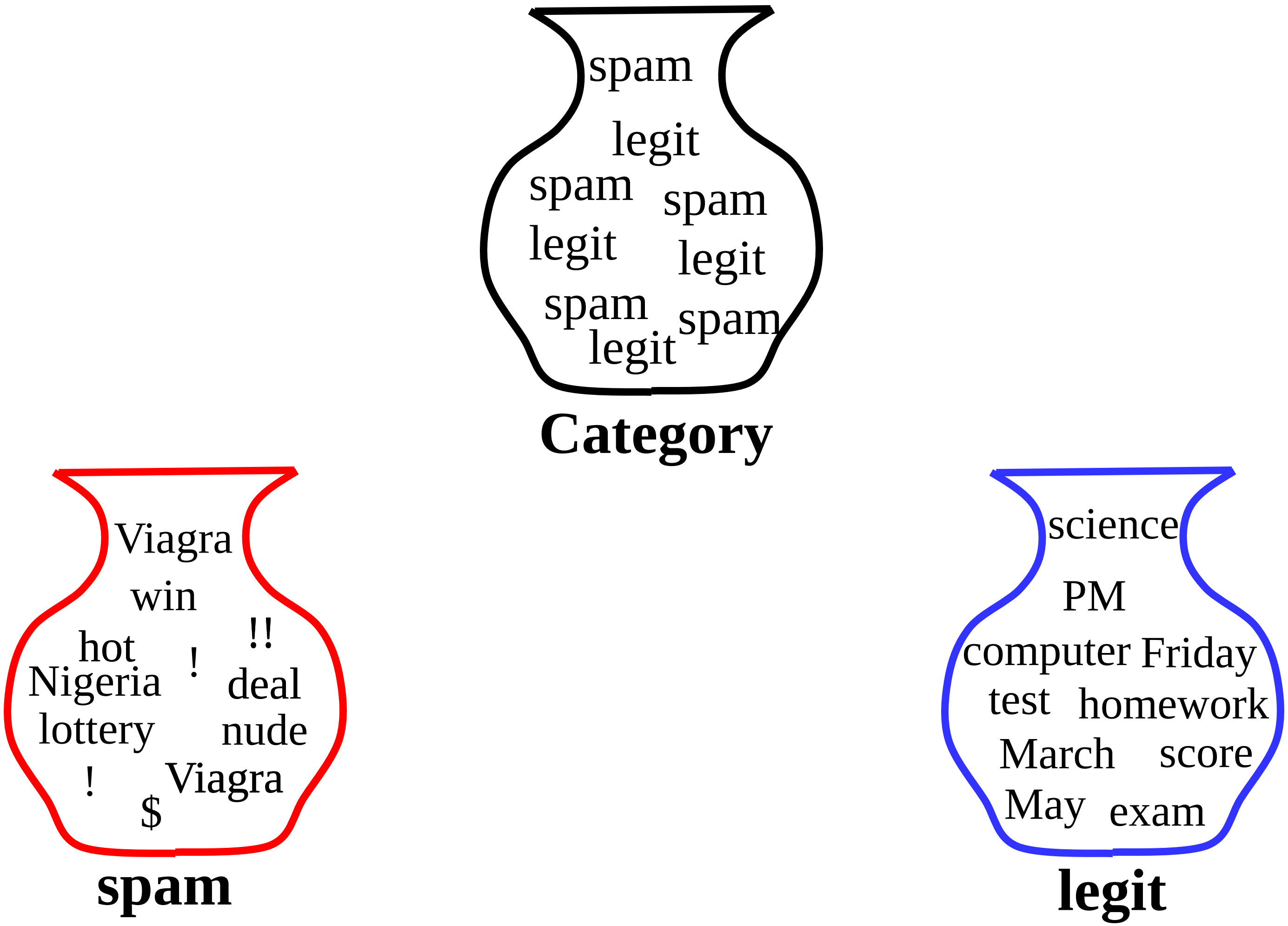
# Laplace Smothing Example

- 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

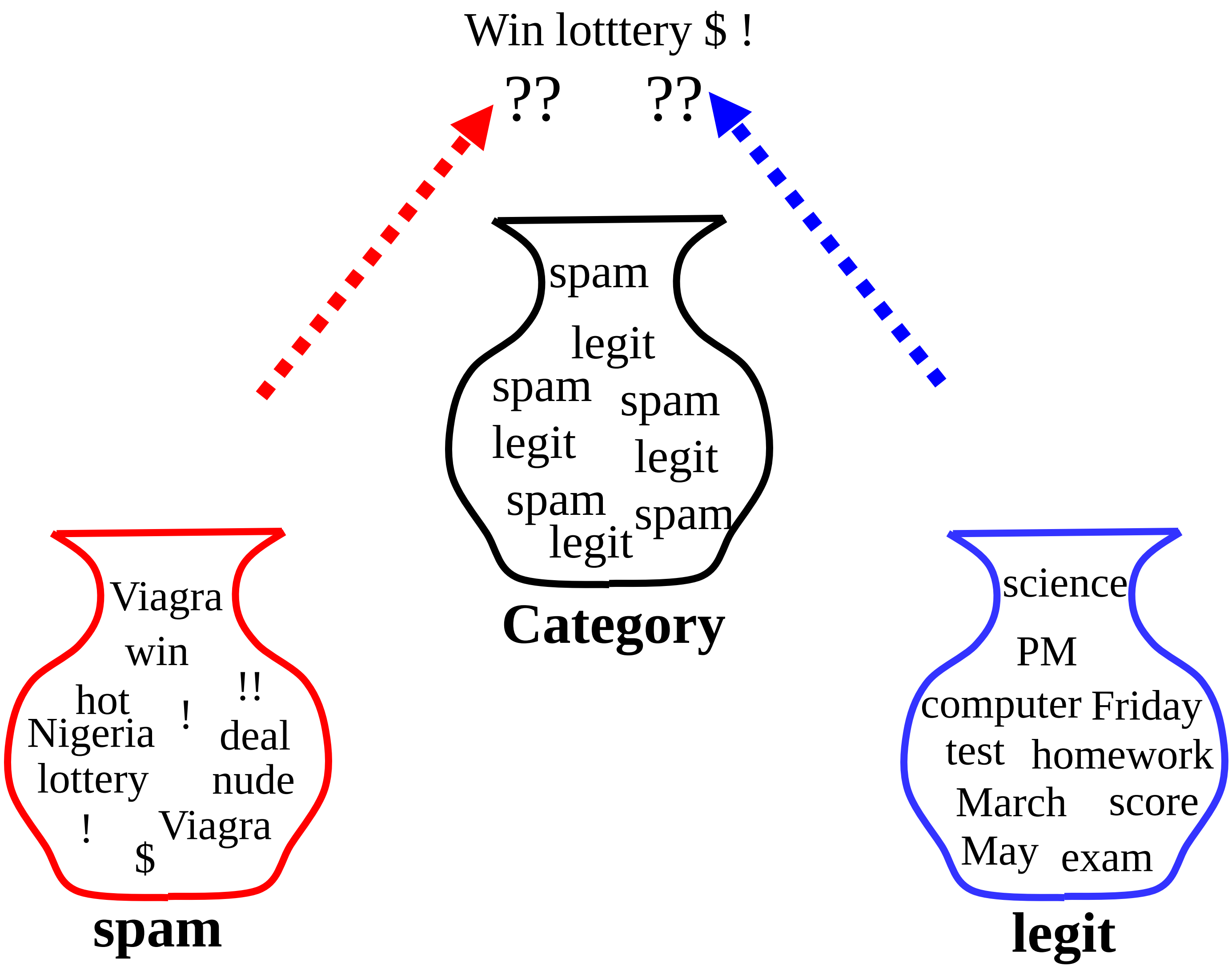
# Naïve Bayes for Text

- Modeled as generating a bag of words for a document in a given category 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.

# Bayes Training Example



# Naïve Bayes Classification



# 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 $P(w_j \mid c_i) = (n_{ij} + 1) / (n_i + |V|)$

# 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 *i*th 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.

# Evaluating Accuracy of Classification

- 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, $N = 10$.

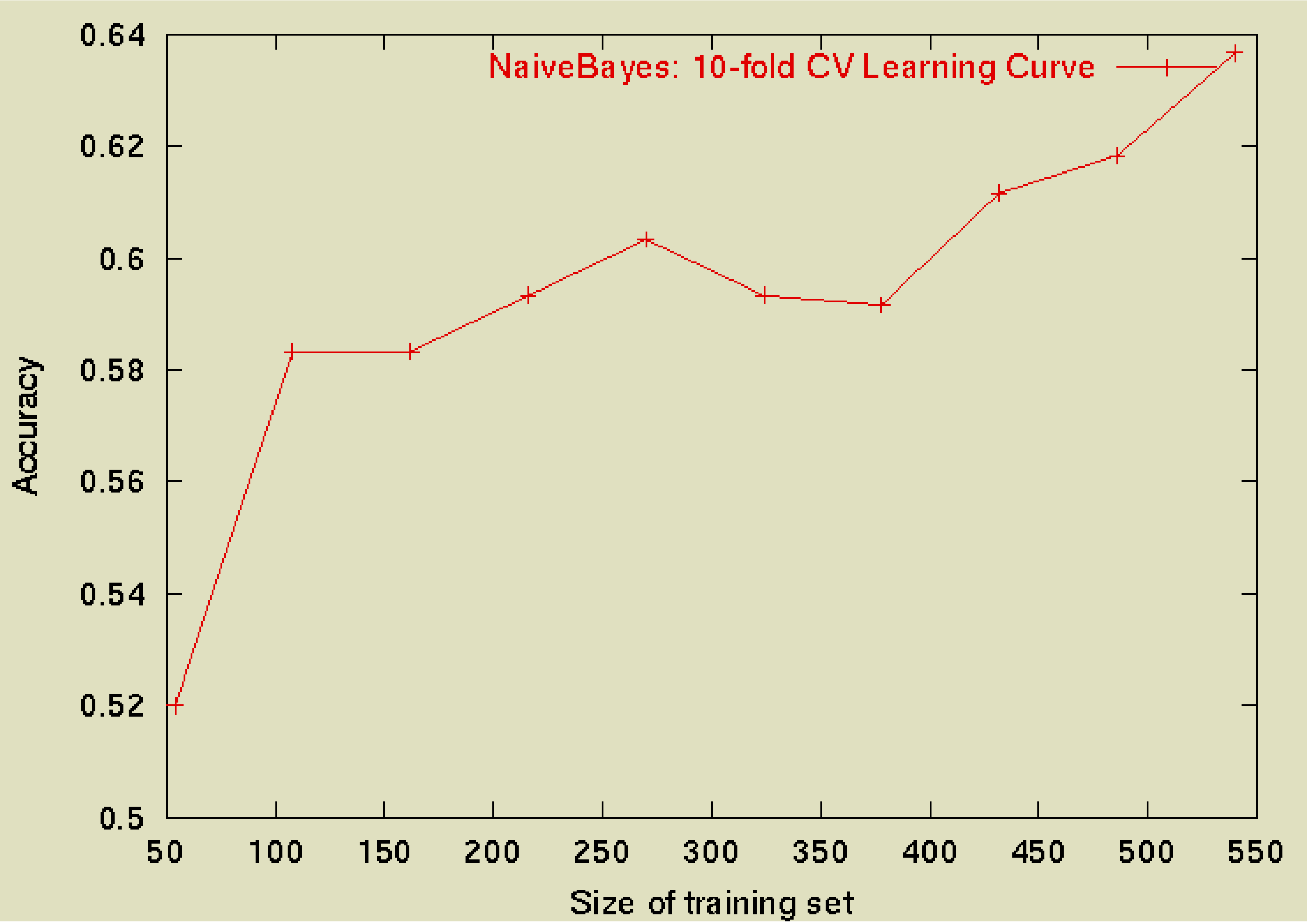
# Learning Curves

- 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.

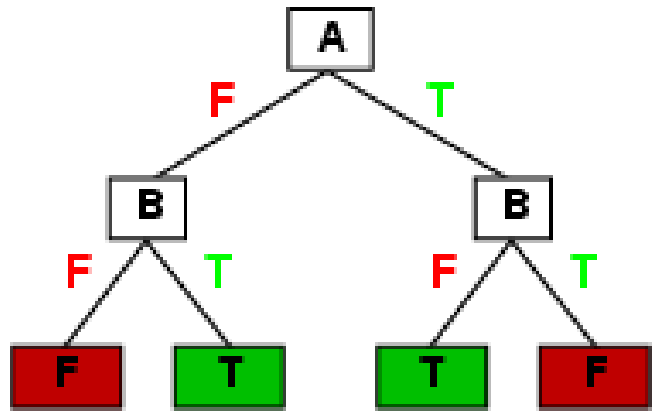
# Sample Learning Curve (Yahoo Science Data)



# Decision Trees

- Decision trees can express any function of the input attributes.
- E.g., for Boolean functions, truth table row → path to leaf:

| A | B | A xor B |
|---|---|---|
| F | F | F |
| F | T | T |
| T | F | T |
| T | T | F |




- Trivially, there is a consistent decision tree for any training set with one path to leaf for each example (unless *f* nondeterministic in *x*) but it probably won't generalize to new examples

- Prefer to find more compact decision trees: we don’t want to memorize the data, we want to find structure in the data!

# Decision Trees: Application Example

Problem: decide whether to wait for a table at a restaurant, based on the following attributes:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

# Training data: Restaurant example

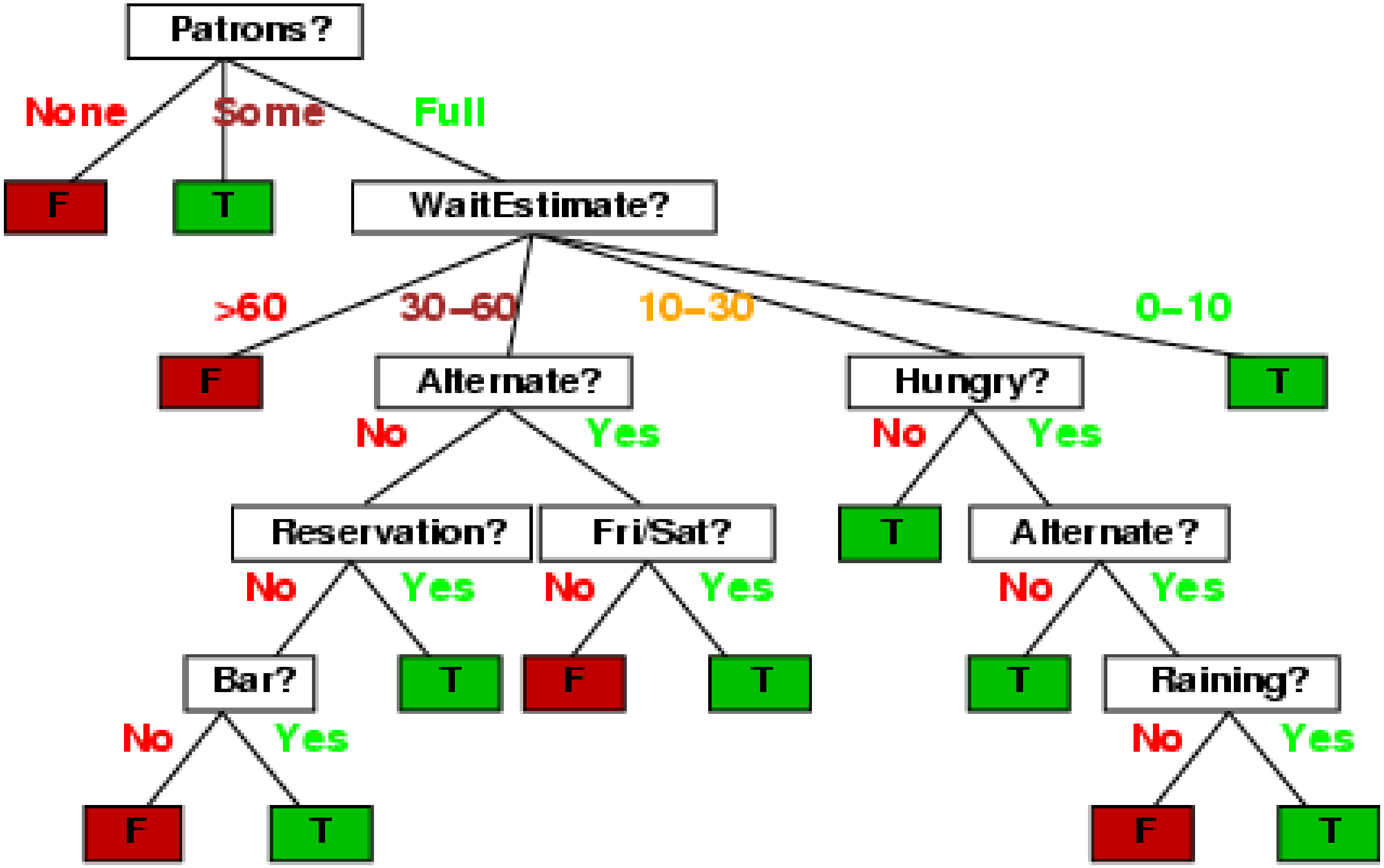
- Examples described by attribute values (Boolean, discrete, continuous)
- E.g., situations where I will/won't wait for a table:

| Example | Attributes | | | | | | | | | | Target |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Alt* | *Bar* | *Fri* | *Hun* | *Pat* | *Price* | *Rain* | *Res* | *Type* | *Est* | *Wait* |
| $X_1$ | T | F | F | T | Some | \$\$\$ | F | T | French | 0–10 | T |
| $X_2$ | T | F | F | T | Full | \$ | F | F | Thai | 30–60 | F |
| $X_3$ | F | T | F | F | Some | \$ | F | F | Burger | 0–10 | T |
| $X_4$ | T | F | T | T | Full | \$ | F | F | Thai | 10–30 | T |
| $X_5$ | T | F | T | F | Full | \$\$\$ | F | T | French | >60 | F |
| $X_6$ | F | T | F | T | Some | \$\$ | T | T | Italian | 0–10 | T |
| $X_7$ | F | T | F | F | None | \$ | T | F | Burger | 0–10 | F |
| $X_8$ | F | F | F | T | Some | \$\$ | T | T | Thai | 0–10 | T |
| $X_9$ | F | T | T | F | Full | \$ | T | F | Burger | >60 | F |
| $X_{10}$ | T | T | T | T | Full | \$\$\$ | F | T | Italian | 10–30 | F |
| $X_{11}$ | F | F | F | F | None | \$ | F | F | Thai | 0–10 | F |
| $X_{12}$ | T | T | T | T | Full | \$ | F | F | Burger | 30–60 | T |

- Classification of examples is positive (T) or negative (F)

# A decision tree to decide whether to wait

- imagine someone talking a sequence of decisions.

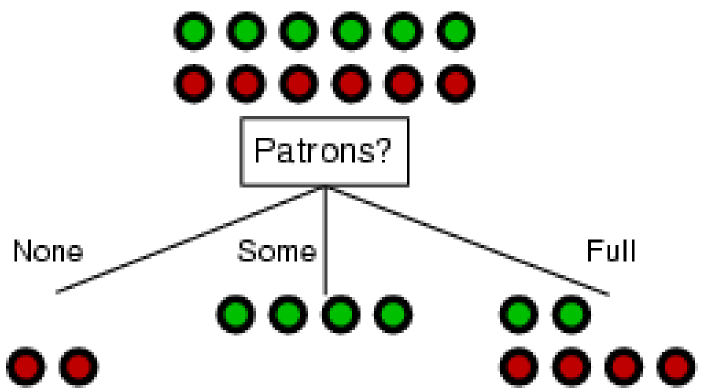


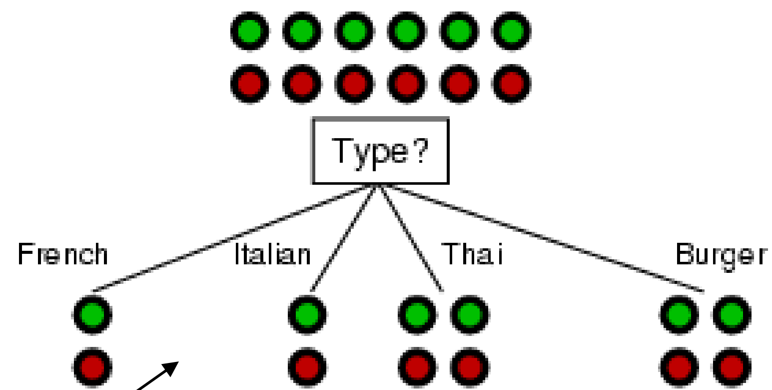
# Decision tree learning

- If there are so many possible trees, can we actually search this space? (solution: greedy search).
- Aim: find a small tree consistent with the training examples
- Idea: (recursively) choose "most significant" attribute as root of (sub)tree.

# Choosing an attribute for making a decision

- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"




To wait or not to wait is still at 50%.

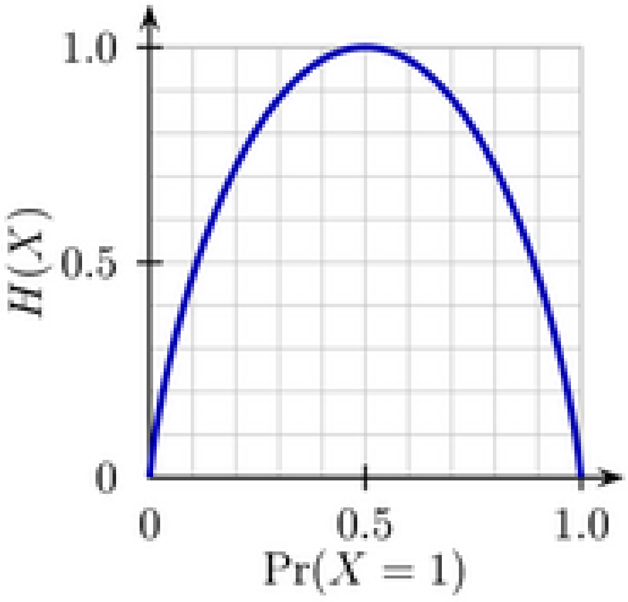
# Information theory background: Entropy

- Entropy measures uncertainty

  $-p \log (p) - (1-p) \log (1-p)$

Consider tossing a biased coin.
If you toss the coin VERY often,
the frequency of heads is, say, p,
and hence the frequency of tails is
1-p.

Uncertainty (entropy) is zero if p=0 or 1
and maximal if we have p=0.5.



# Using information theory for binary decisions

- Imagine we have p examples which are true (positive) and n examples which are false (negative).

- Our best estimate of true or false is given by:

$$P(true) \approx p / p + n$$

$$p(false) \approx n / p + n$$

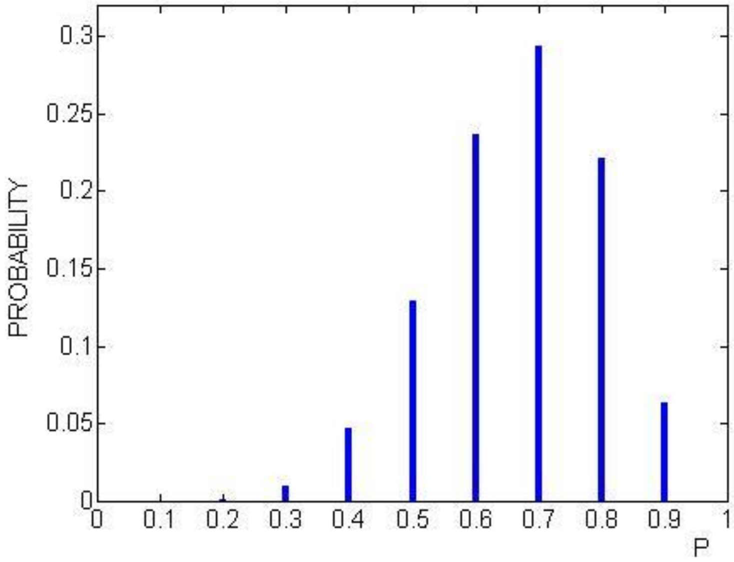
- Hence the entropy is given by:

$$Entropy(\frac{p}{p+n}, \frac{n}{p+n}) \approx -\frac{p}{p+n}\log\frac{p}{p+n} - \frac{n}{p+n}\log\frac{n}{p+n}$$

# Using information theory for more than 2 states

- If there are more than two states s=1,2,..n we have (e.g. a die):

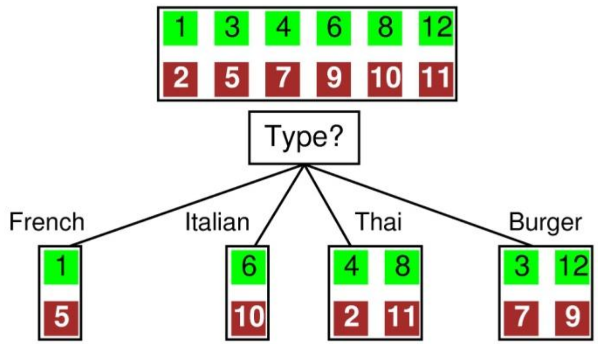
$$
\begin{aligned}
Entropy(p) = &-p(s=1)\log[p(s=1)] \\
&-p(s=2)\log[p(s=2)] \\
&\ldots \\
&-p(s=n)\log[p(s=n)]
\end{aligned}
$$

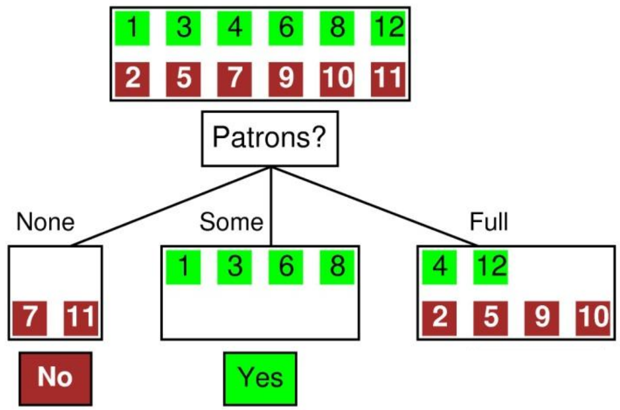



$$\sum_{s=1}^{n} p(s) = 1$$

# ID3 Algorithm: Using Information Theory to Choose an Attribute

- How much information do we gain if we disclose the value of some attribute?
- ID3 algorithm by Ross Quinlan uses information gained measured by maximum entropy reduction:
  - IG(A) = uncertainty before – uncertainty after
  - Choose an attribute with the maximum IA




Before: Entropy = - ½ log(1/2) – ½ log(1/2)=log(2) = 1 bit:
There is "1 bit of information to be discovered".

After: for Type: If we go into branch "French" we have 1 bit, similarly for the others.

French: 1bit
Italian: 1 bit
Thai: 1 bit
Burger: 1bit

On average: 1 bit and gained nothing!

After: for Patrons: In branch "None" and "Some" entropy = 0!,
In "Full" entropy = -1/3log(1/3)-2/3log(2/3)=0.92

So Patrons gains more information!

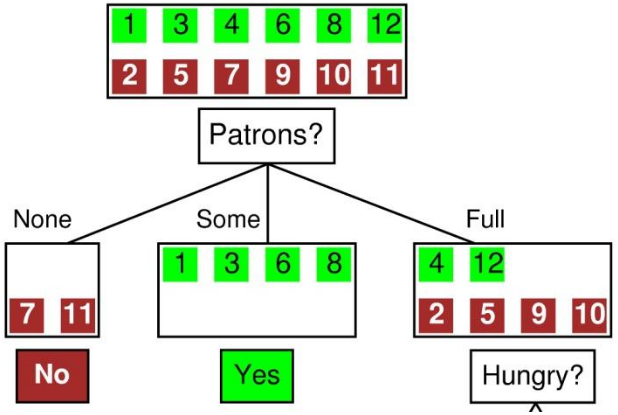
# Information Gain: How to combine branches

- 1/6 of the time we enter “None”, so we weight“None” with 1/6. Similarly: “Some” has weight: 1/3 and “Full” has weight ½.

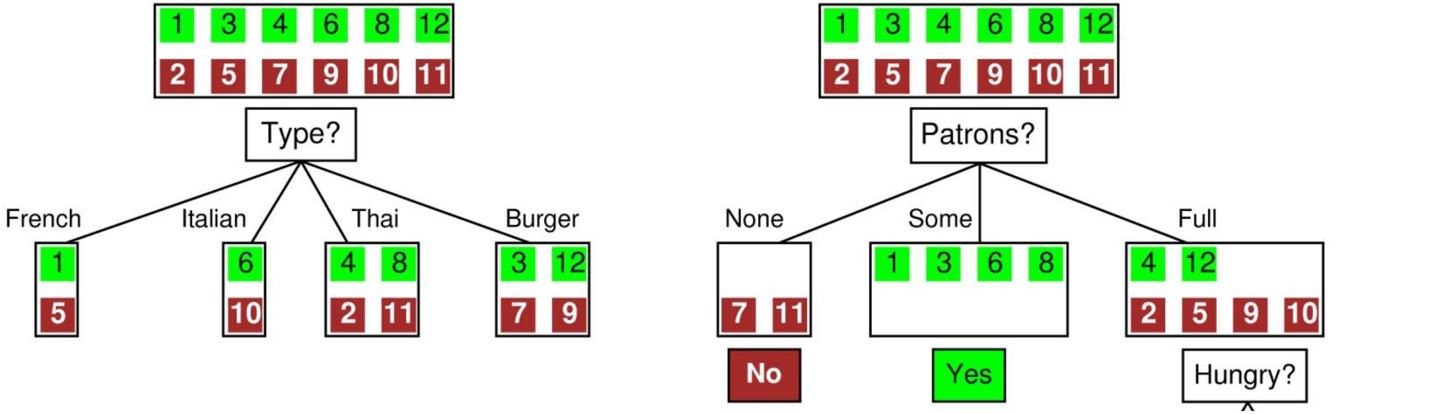
$$Entropy(A) = \sum_{i=1}^{n} \frac{p_i + n_i}{p+n} Entropy\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)$$

entropy for each branch.

weight for each branch



# Choose an attribute: Restaurant Example




For the training set, $p = n = 6$, $I(6/12, 6/12) = 1$ bit

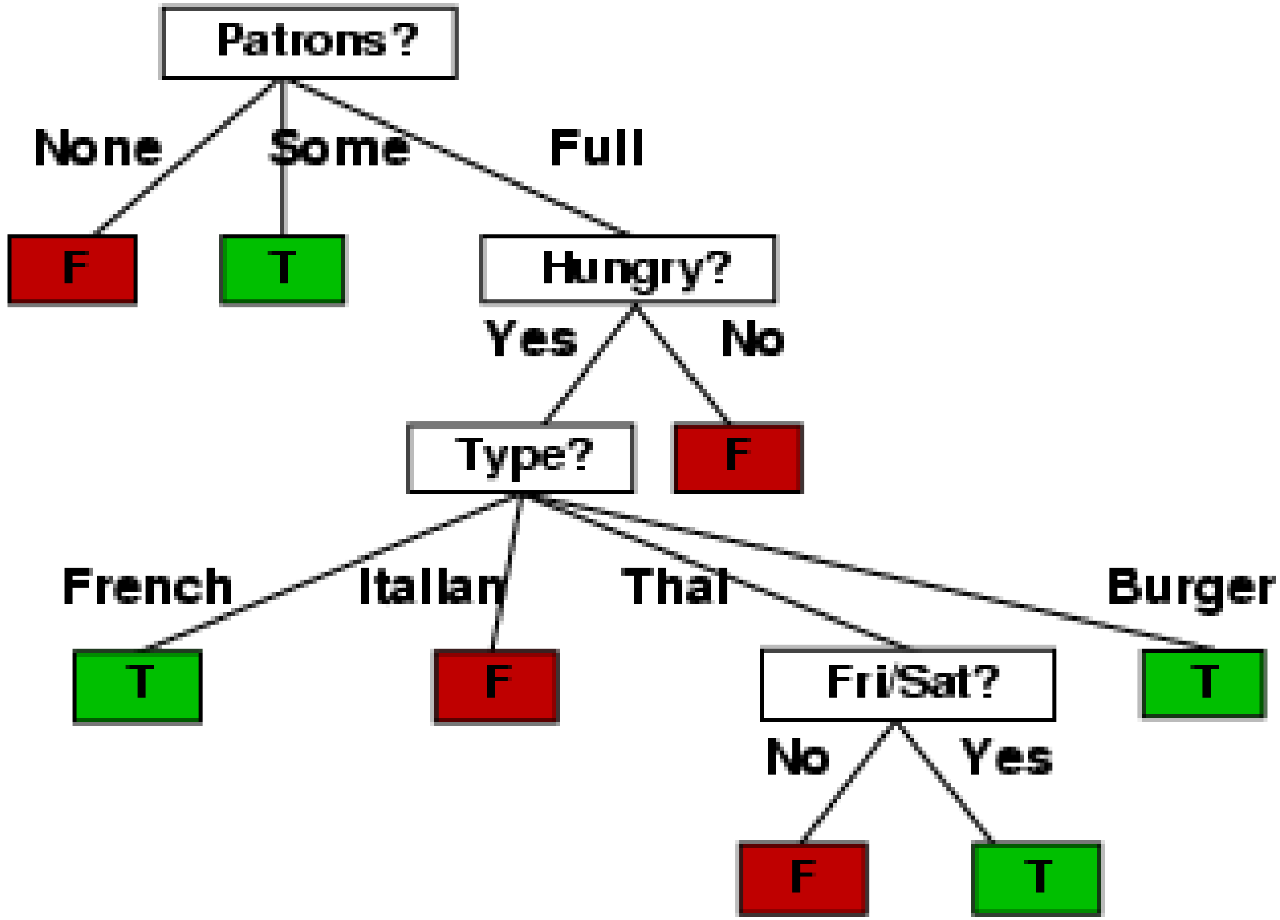
$$IG(Patrons) = 1 - [\frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I(\frac{2}{6}, \frac{4}{6})] = .0541 \text{ bits}$$

$$IG(Type) = 1 - [\frac{2}{12} I(\frac{1}{2}, \frac{1}{2}) + \frac{2}{12} I(\frac{1}{2}, \frac{1}{2}) + \frac{4}{12} I(\frac{2}{4}, \frac{2}{4}) + \frac{4}{12} I(\frac{2}{4}, \frac{2}{4})] = 0 \text{ bits}$$

*Patrons* has the highest IG of all attributes and so is chosen by the DTL algorithm as the root

# Example: Decision tree learned

- Decision tree learned from the 12 examples:



# Issues

- When there are no attributes left:
  - Stop growing and use majority vote.
- Avoid over-fitting data
  - Stop growing a tree earlier
  - Grow first, and prune later.
- Deal with continuous-valued attributes
  - Dynamically select thresholds/intervals.
- Handle missing attribute values
  - Make up with common values
- Control  tree size
  - pruning