Overview of Machine Learning and Pattern Recognition

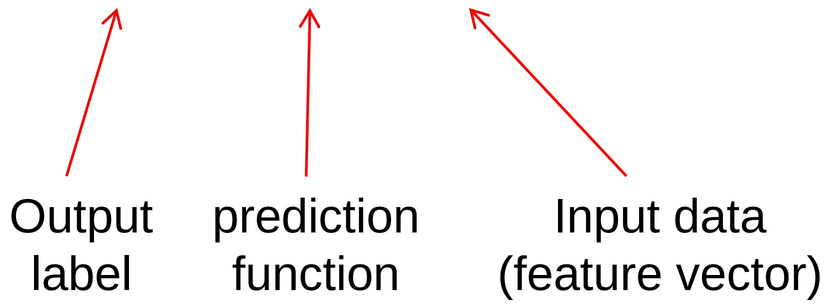
Dipartimento di Automatica e Informatica
Politecnico di Torino, Torino, ITALY

Summary of ML problems

	Supervised Learning	*Unsupervised Learning*
Discrete	classification or categorization	clustering
Continuous	regression	dimensionality reduction

The classification framework

$$y = f(\mathbf{x})$$



Output label

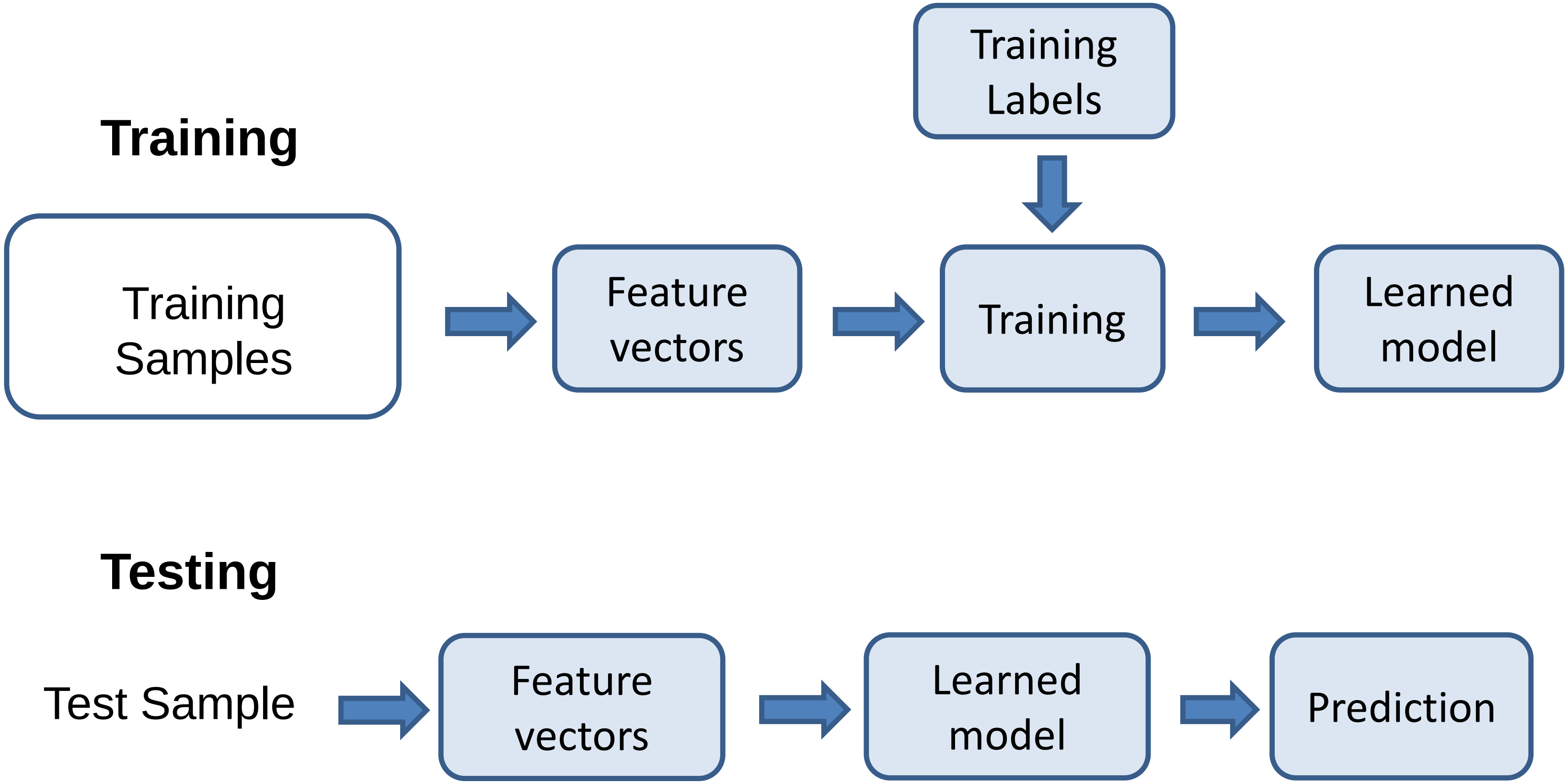
prediction function

Input data (feature vector)

- **Training:** given a *training set* of labeled examples $\{(\mathbf{x}_1,y_1), \ldots, (\mathbf{x}_N,y_N)\}$, estimate the prediction function f by minimizing the prediction error on the training set
- **Testing:** apply f to a never before seen *test example* $\mathbf{x}$ and output the predicted value $y = f(\mathbf{x})$

Slide credit: L. Lazebnik

Steps



Training

Training Samples → Feature vectors → Training → Learned model

Training Labels → Training

Testing

Test Sample → Feature vectors → Learned model → Prediction

Many classifiers to choose from

- Nearest mean
- K-nearest neighbor
- Naïve Bayes
- SVM
- Decision Trees
- Neural networks
- Randomized Forests
- Etc.

Which is the best one?

Slide credit: D. Hoiem

No Free Lunch Theorem

there is no model that works best for every problem!

Slide credit: D. Hoiem

Choosing a classifier

- **Training error**: classification error estimated on the training data. Measure of how well the model fits the training set
- **Test error**: classification error estimated on the test data, which is independent from the data the model was trained on. Measure of how well the model fits new data
- **Generalization:** how well does a learned model generalize from the data it was trained on to a completely new dataset?
  - **Bias:** how much the average model over all training sets differ from the true model?
    - Error due to inaccurate assumptions/simplifications made by the model
  - **Variance:** how much models estimated from different training sets differ from each other

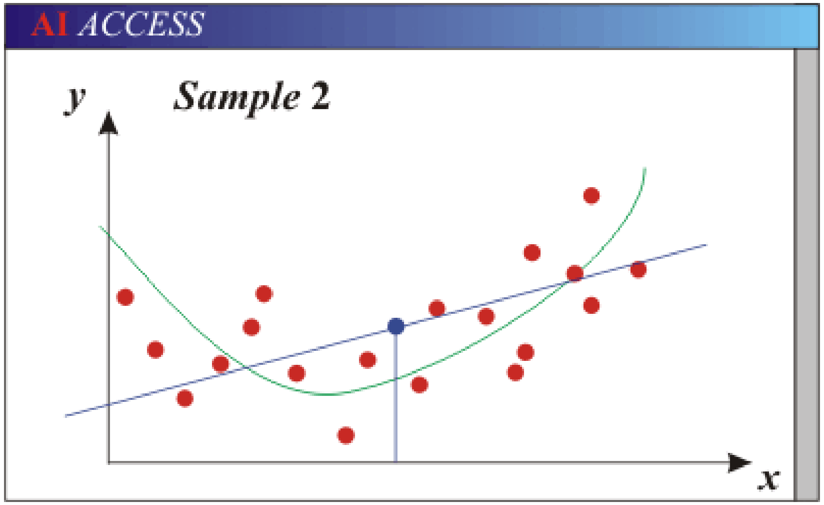
Slide credit: L. Lazebnik

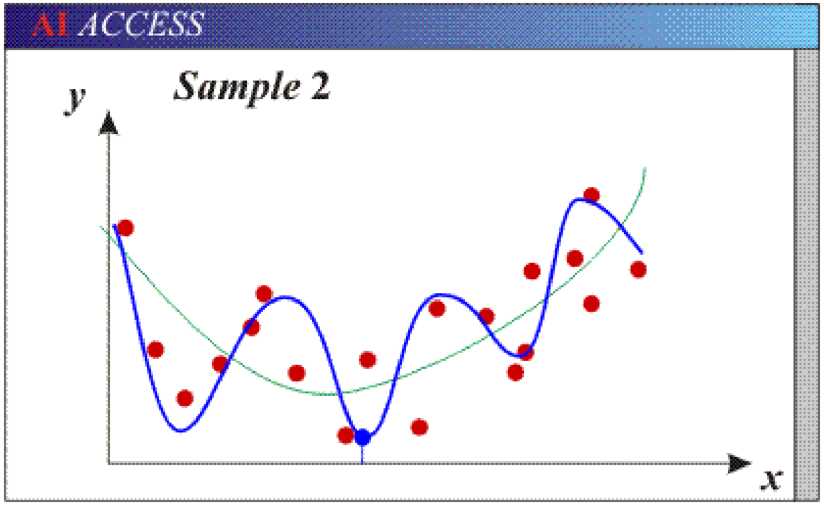
Generalization

- **Underfitting:** the learned model is too “simple” to represent all the relevant class characteristics
  - High training error and high test error
  - High bias and low variance
- **Overfitting:** model is too “complex” and fits irrelevant characteristics (noise) in the data
  - Low training error and high test error
  - Low bias and high variance

Slide credit: L. Lazebnik

Bias-Variance Trade-off





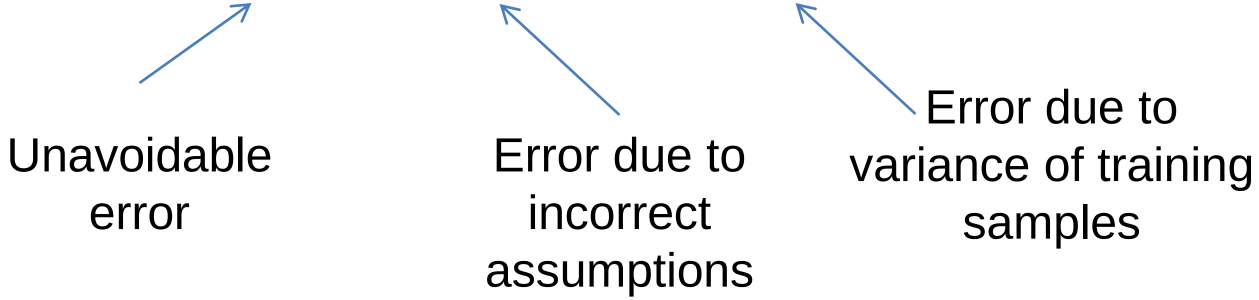
- Models with too few parameters are inaccurate because of a large bias (not enough flexibility).

- Models with too many parameters are inaccurate because of a large variance (too depenent on the training s).

Slide credit: D. Hoiem

Bias-Variance Trade-off

$$E(MSE) = noise^2 + bias^2 + variance$$



Unavoidable error

Error due to incorrect assumptions

Error due to variance of training samples

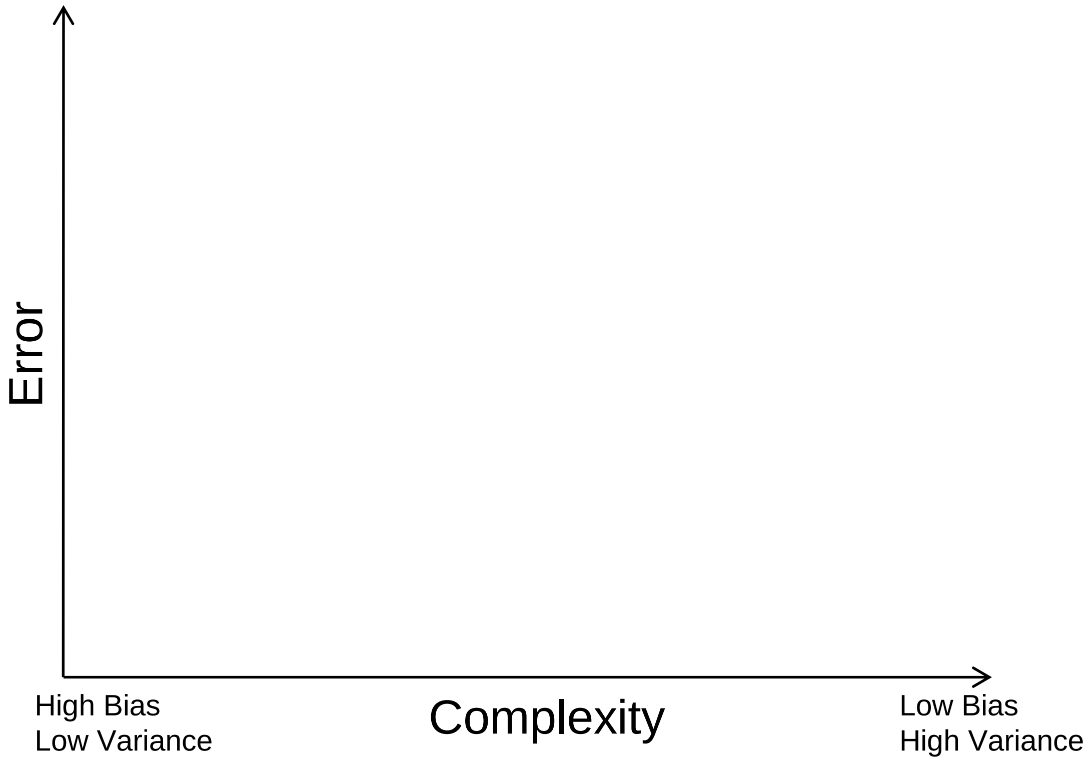
See the following for explanations of bias-variance (also Bishop's "Neural Networks" book):

- http://www.inf.ed.ac.uk/teaching/courses/mlsc/Notes/Lecture4/BiasVariance.pdf

Slide credit: D. Hoiem

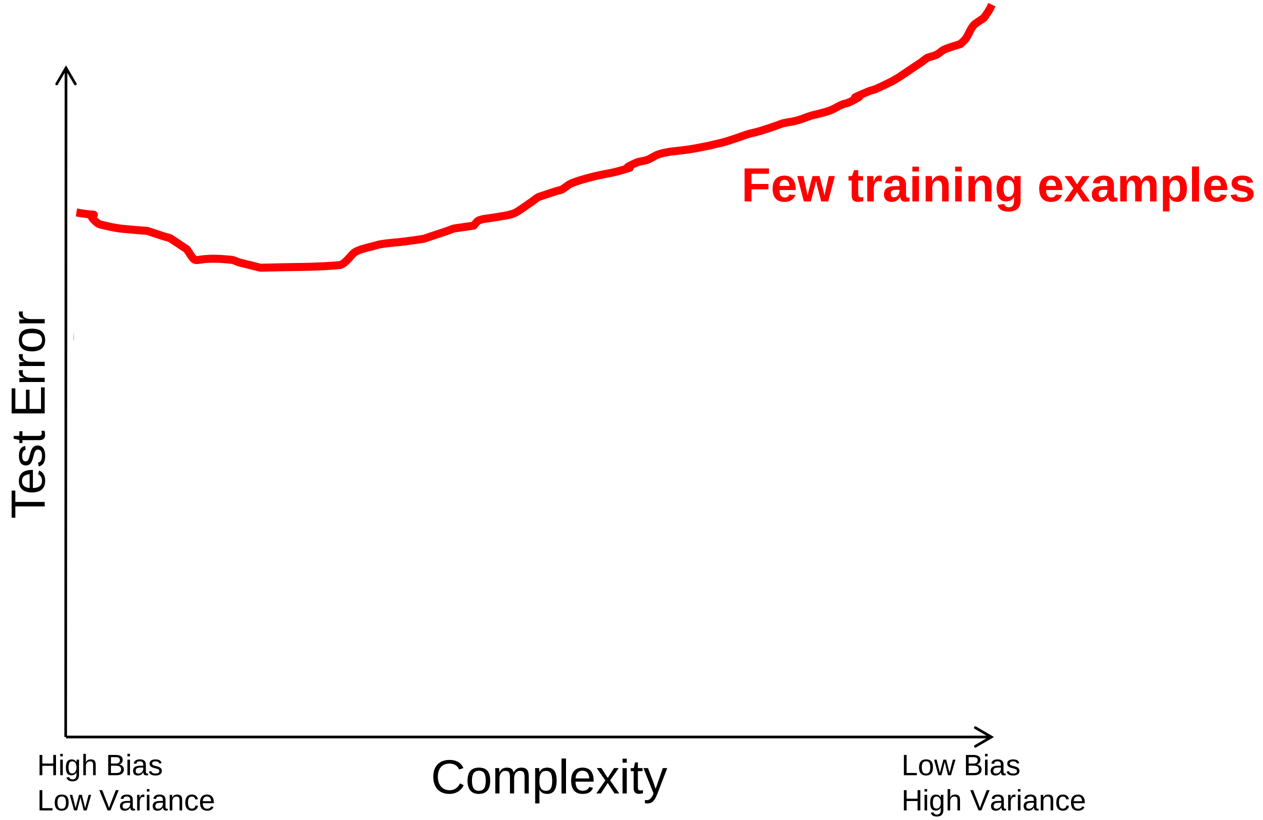
Bias-variance tradeoff

Underfitting **Overfitting**



Slide credit: D. Hoiem

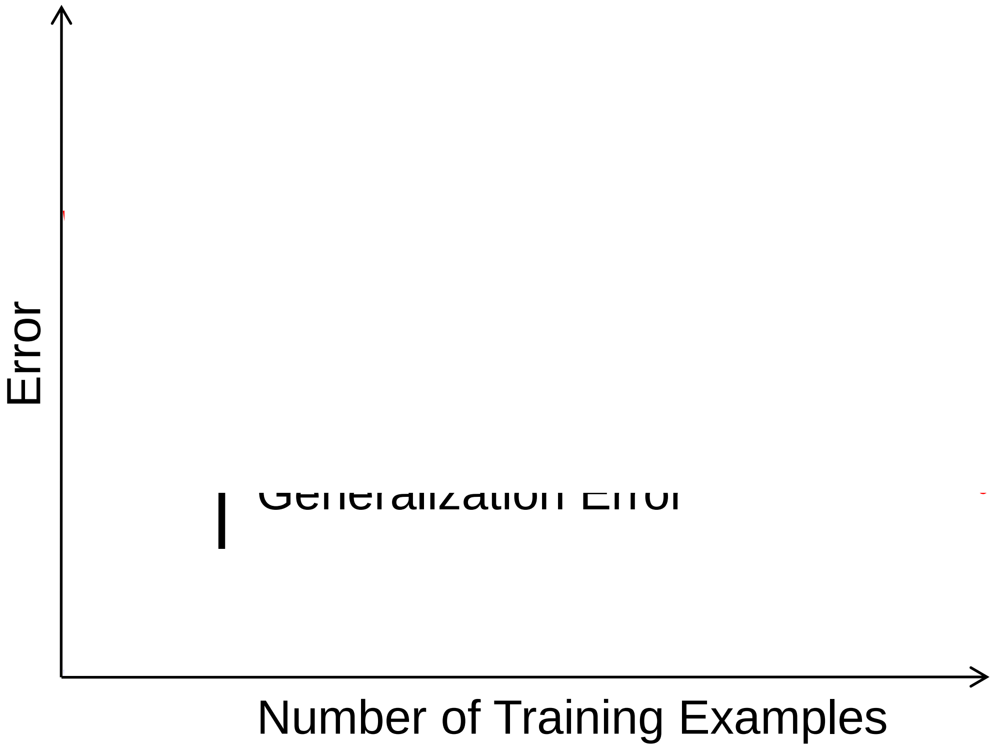
Bias-variance tradeoff



Slide credit: D. Hoiem

Effect of Training Size

Fixed prediction model



Slide credit: D. Hoiem

Remember…

- No classifier is inherently better than any other: you need to make assumptions to generalize

- Three kinds of error
  - Inherent: unavoidable
  - Bias: due to over-simplifications
  - Variance: due to inability to perfectly estimate parameters from limited data

Slide credit: D. Hoiem

How to reduce variance?

- Choose a simpler classifier
- Regularize the parameters
- Get more training data (if you can…)

Slide credit: D. Hoiem

Brief tour of some classifiers

- Nearest mean
- K-nearest neighbor (KNN)
- Naïve Bayes classifier
- Support Vector Machines (SVM)
- Decision Trees
- Randomized Forests
- Neural networks
- Deep learning
- Etc.

Generative vs. Discriminative Classifiers

Generative Models

- Represent both the data and the labels
- Often, makes use of conditional independence and priors
- Examples
  - Naïve Bayes classifier
- Models of data may apply to future prediction problems

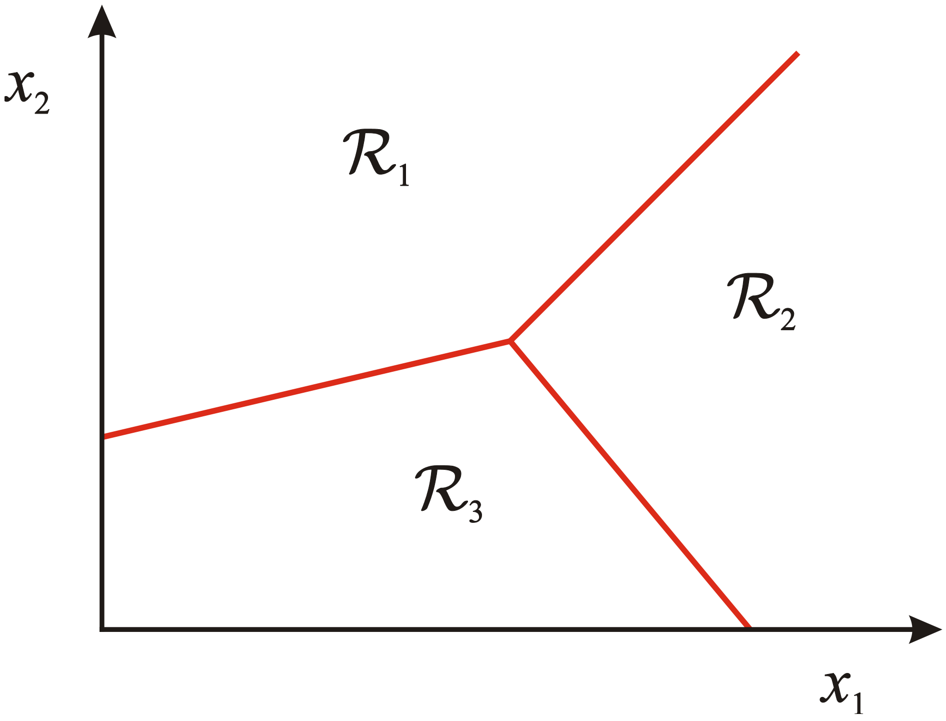
Discriminative Models

- Learn to directly predict the labels from the data
- Often, assume a simple boundary in the feature space (e.g., linear)
- Examples
  - SVM
- Often easier to predict a label from the data than to model the data

Slide credit: D. Hoiem

Classification

- Assign input vector to one of two or more classes
- Any decision rule divides input space into *decision regions* separated by *decision boundaries*

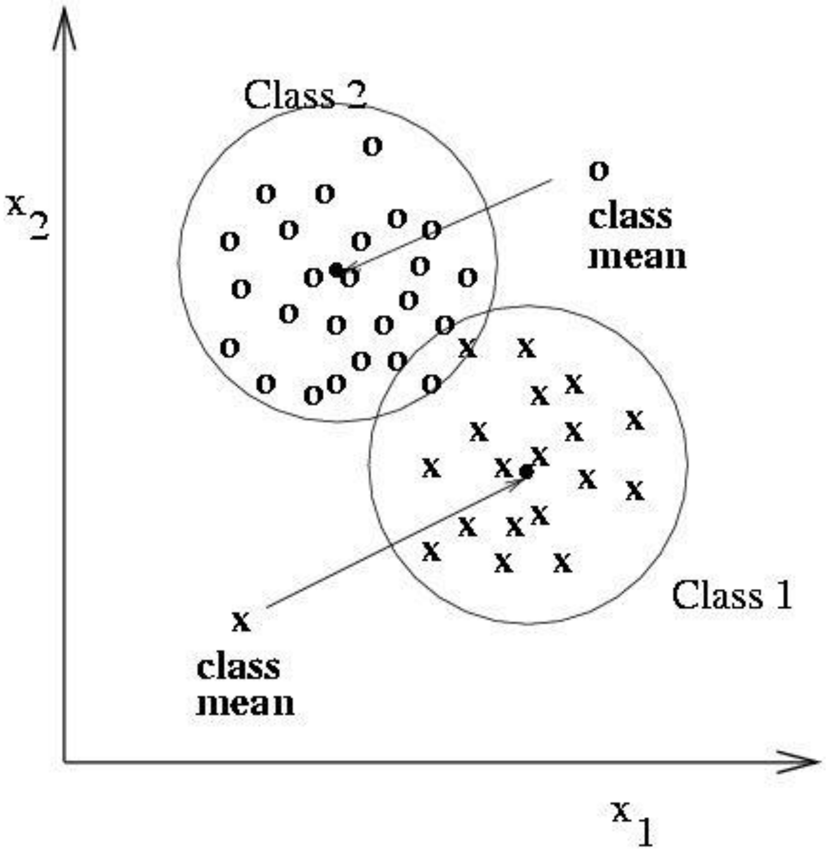


Slide credit: L. Lazebnik

Nearest mean classification

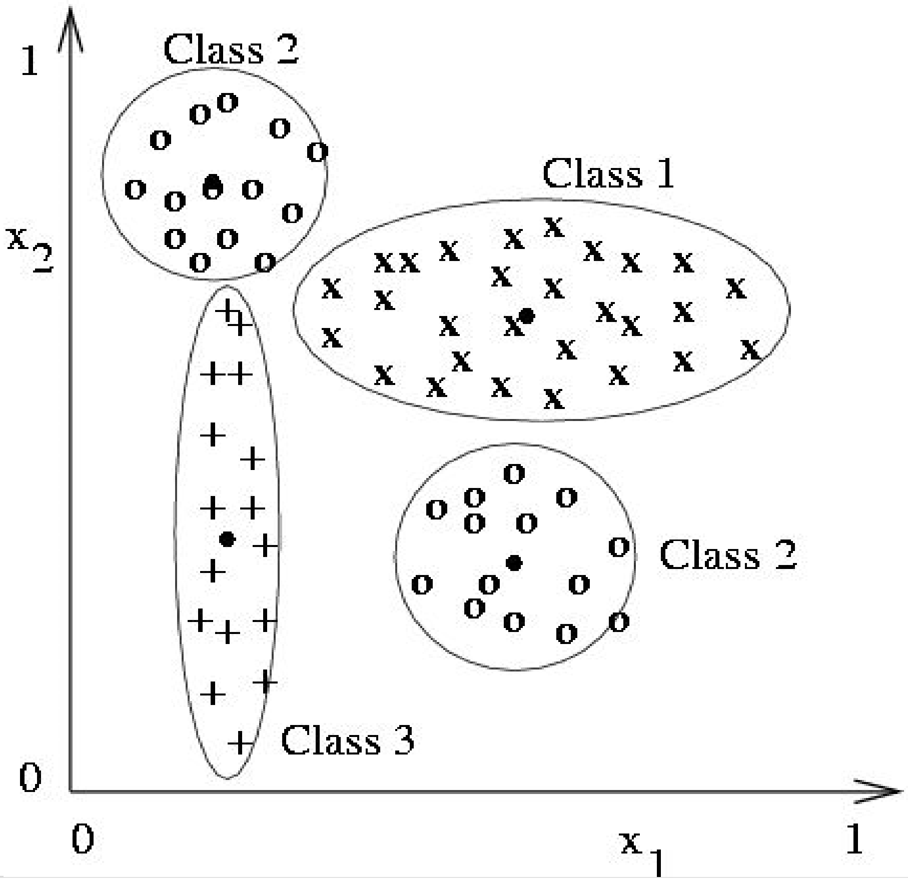
- Compute the Euclidean distance between feature vector X and the mean of each class.

- Choose the closest class, if close enough (reject otherwise)



Nearest mean: limitations

- Poor results with complex class structure



- Class 2 has two modes; where is its mean?
- But if modes were computed, two subclass mean vectors could be used…

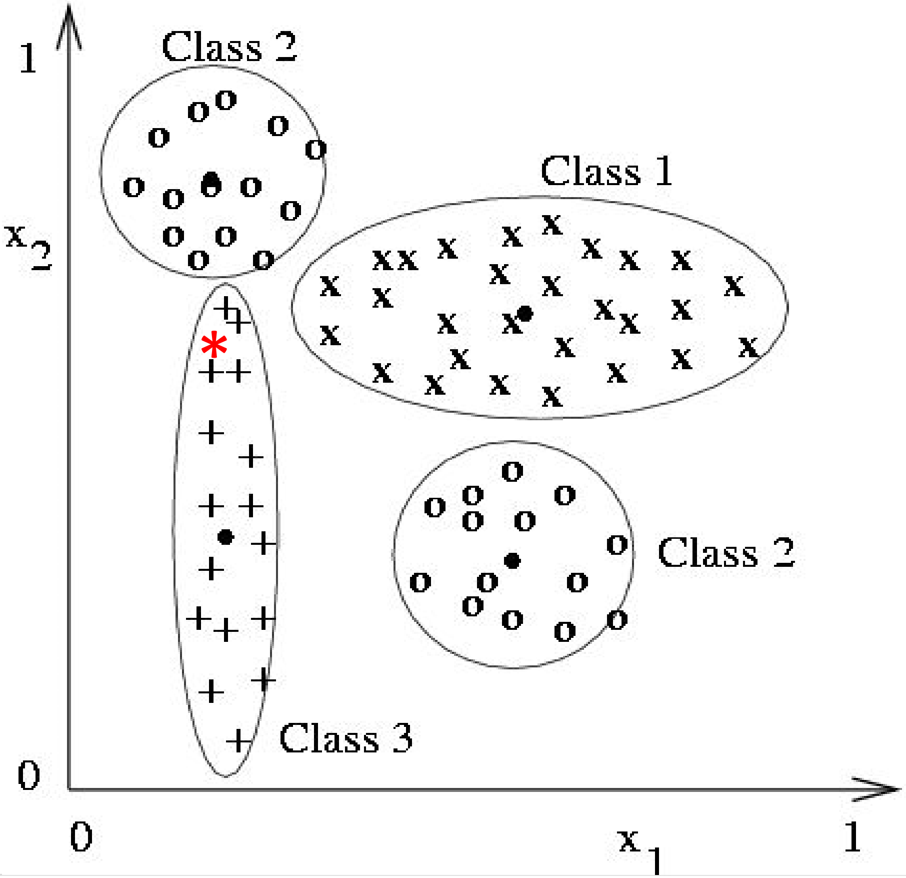
Improving nearest mean

- Modified Euclidean distance:
  - Scaling the distance from the feature vector x to the class mean x_c by the spread (or standard deviation) $\sigma_i$of class *c* along each dimension *i*

$$||x \ - x_c|| = \sqrt{\sum_{i=1}^{d} \left(\frac{x[i] - x_c[i]}{\sigma_i} \right)^2}$$

Improving nearest mean

- Improves results with complex class structure

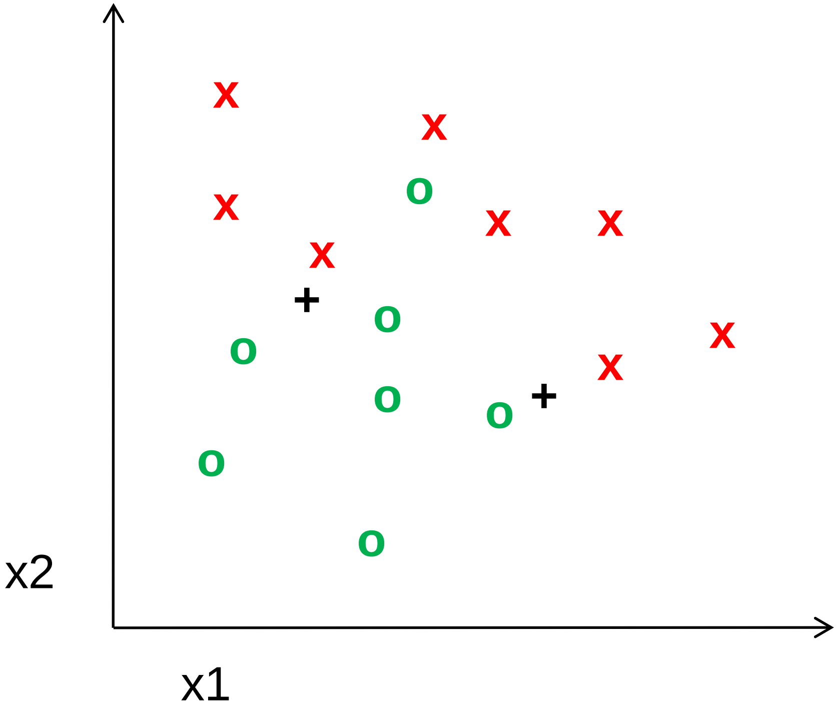


- If we use regular distance, point * will be assigned to Class 2
- If we use the modified distance, * will sclale closer to the mean of Class 3.

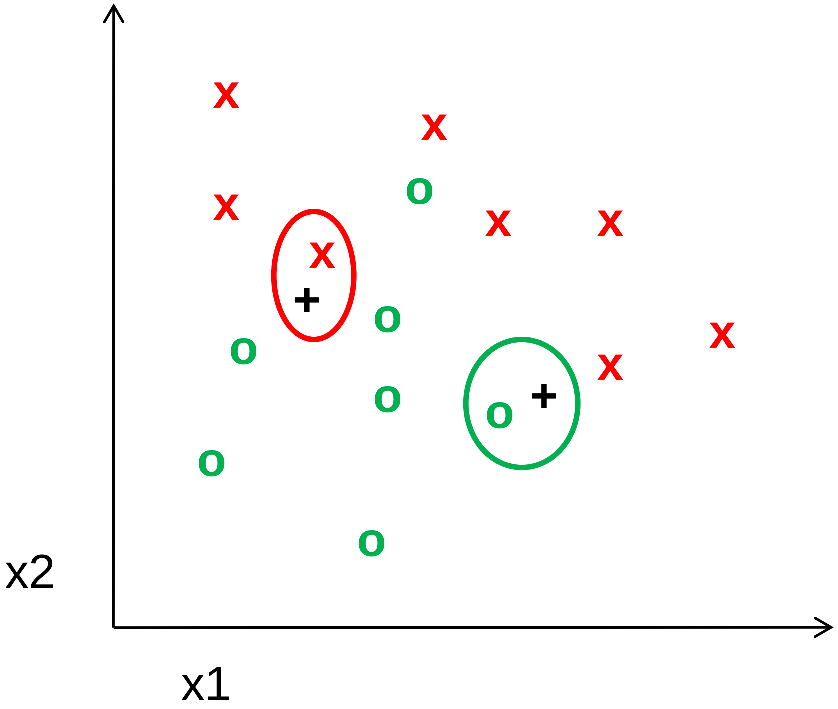
Nearest Neighbor Classifier

- Keep all the training samples in some efficient look-up structure.
- Find the nearest neighbor of the feature vector to be classified and assign the class of the neighbor.
- Can be extended to K-nearest neighbors.

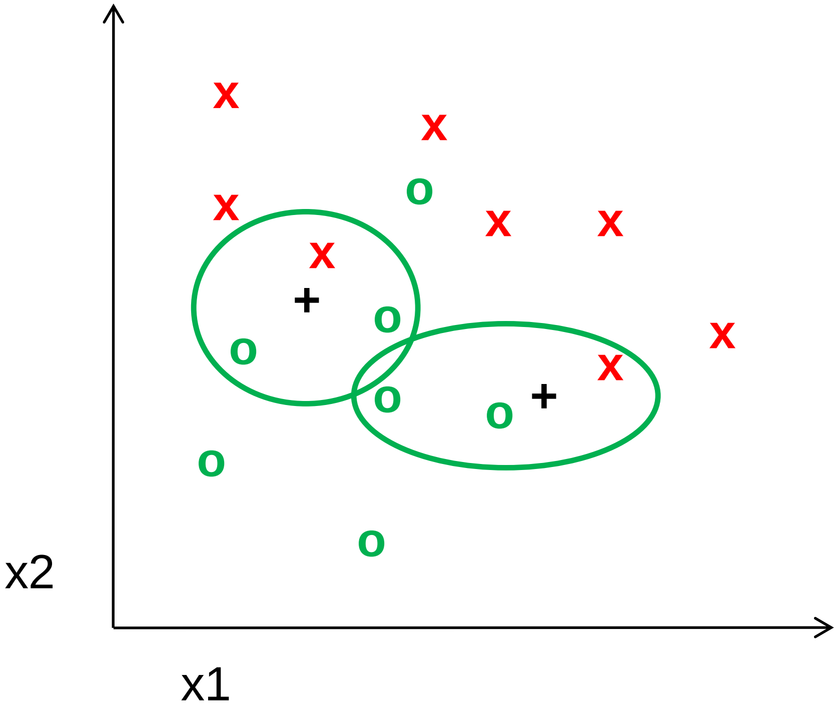
K-nearest neighbor



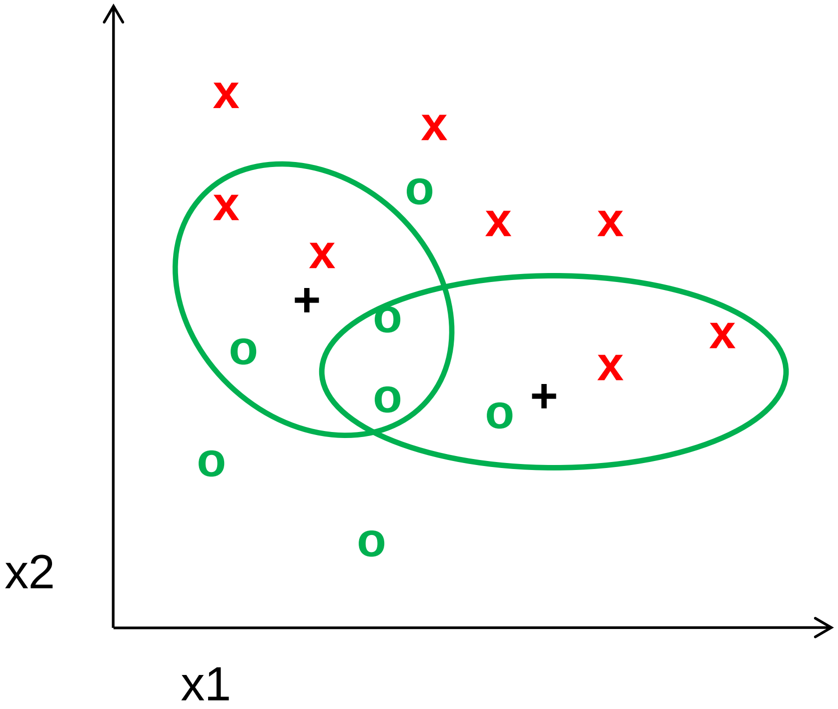
1-nearest neighbor



3-nearest neighbor



5-nearest neighbor

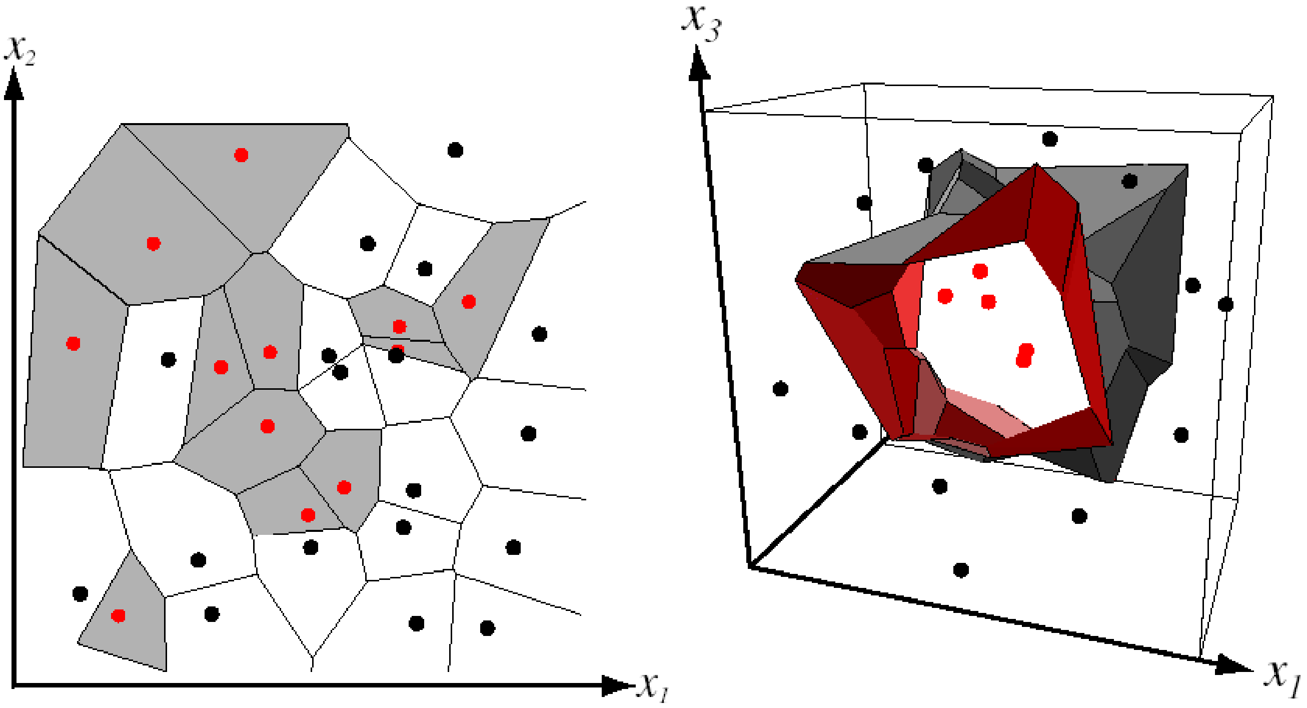


Using K-NN

- Simple algorithm, a good one to try first
- Learning is simple, but classification is time-consuming and computationally intensive
- Choosing K might be tricky

Voronoi diagrams

- Feature space boundaries established by K-NN



Bayesian Classification: Why?

- A statistical classifier: performs *probabilistic prediction, i.e.,* predicts class membership probabilities
- Foundation: Based on Bayes' Theorem.
- Incremental: Each training example can incrementally increase/decrease the probability that a hypothesis is correct — prior knowledge can be combined with observed data
- Standard: Even when Bayesian methods are computationally intractable, they can provide a standard of optimal decision making against which other methods can be measured

Bayes' Theorem: Basics

- Total probability Theorem:

$$P(B) = \sum_{i=1}^{M} P(B|A_i)P(A_i)$$

- Bayes' Theorem:

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$

  - Let **X** be a data sample ("*evidence*"): class label is unknown
  - Let H be a *hypothesis* that X belongs to class C
  - Classification is to determine P(H|**X**), (i.e., *posteriori probability):* the probability that the hypothesis holds given the observed data sample **X**
  - P(H) (*prior probability*): the initial probability
    - E.g., **X** will buy computer, regardless of age, income, …
  - P(**X**): probability that sample data is observed
  - P(**X**|H) (likelihood): the probability of observing the sample **X**, given that the hypothesis holds

Prediction Based on Bayes' Theorem

- Given training data **X**, *posteriori probability of a hypothesis* H, P(H|**X**), follows the Bayes' theorem

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$

- Informally, this can be viewed as

  posteriori = likelihood x prior/evidence

- Predicts **X** belongs to C_i iff the probability P(C_i|**X**) is the highest among all the P(C_k|X) for all the *k* classes
- Practical difficulty: It requires initial knowledge of many probabilities, involving significant computational cost

Classification: deriving max Poteriori

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n-D attribute vector **X** = (x_1, x_2, …, x_n)
- Suppose there are *m* classes C_1, C_2, …, C_m.
- Classification is to derive the maximum posteriori, i.e., the maximal P(C_i|**X**)
- This can be derived from Bayes' theorem

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$$

- Since P(X) is constant for all classes, only

$$P(C_i|\mathbf{X}) = P(\mathbf{X}|C_i)P(C_i)$$

needs to be maximized

Naïve Bayes Classifier

- A simplified assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

- This greatly reduces the computation cost: Only counts the class distribution
- If A_k is categorical, $P(x_k|C_i)$ is the # of tuples in C_i having value x_k for A_k divided by $|C_{i,D}|$ (# of tuples of C_i in D)
- If A_k is continous-valued, $P(x_k|C_i)$ is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

$$g(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

and $P(x_k|C_i)$ is

$$P(\mathbf{X}|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$$

Example 1

- *A patient takes a lab test and the result comes back positive (+).*
- *The test returns a correct positive result (+) in only 98% of the cases in which the disease is actually present, and a correct negative result (-) in only 97% of the cases in which the disease is not present*
- *Furthermore, 0.008 of the entire population have this cancer*

Does the patient have cancer or not?

Example 1 (cont.)

What we know from the data is:

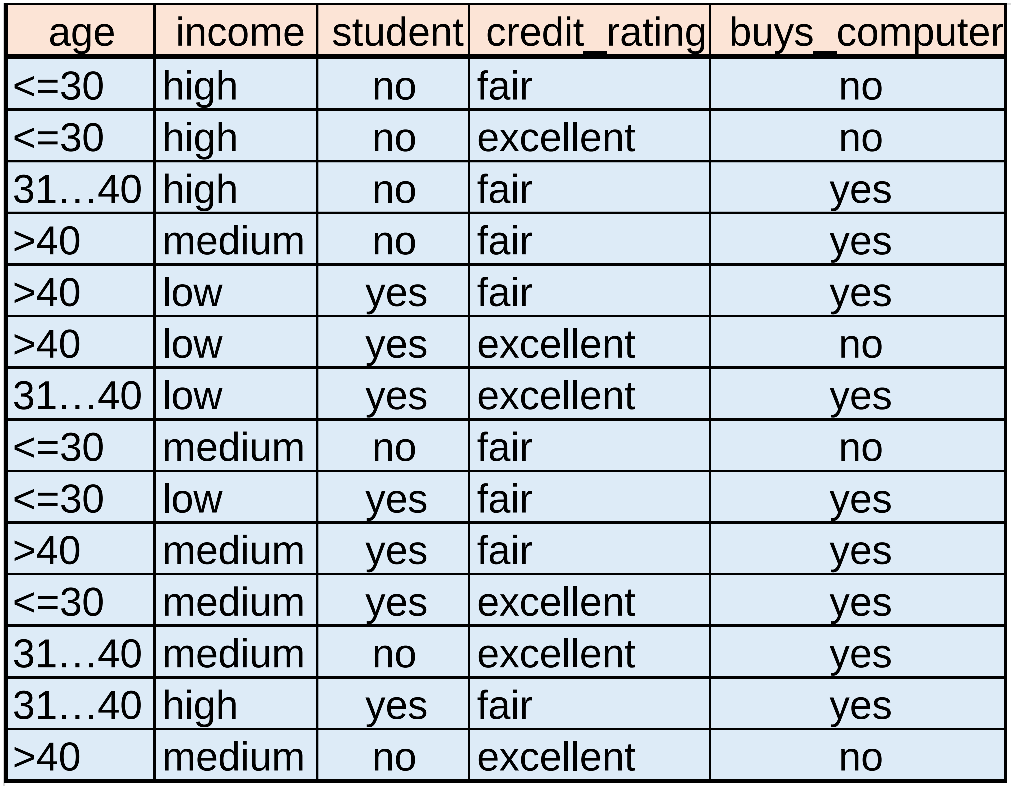
- P(cancer) = 0.008 → P(not_cancer) = 1-0.008 = 0.992
- P(+/cancer) = 0.98 → P(-/cancer) = 1-0.98 = 0.02
- P(-/not_cancer) = 0.97 → P(+/not_cancer) = 1-0.97 = 0.03

We apply Bayes rule:

Prediction = argmax (P(cancer/+) , P(not_cancer/+)) =
argmax(P(+/cancer)*P(cancer) , P(+/not_cancer)*P(not_cancer)) =
argmax(0.98*0.008, 0.03*0.992) = argmax(0.0078 , 0.0298)

→ The Maximum a posteriori Probability is associated to the hypothesis that THE PATIENT DOES NOT HAVE CANCER

Example 2



age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Class:

C1:buys_computer = ‘yes’

C2:buys_computer = ‘no’

Data to be classified:

X = (age <=30,

Income = medium,

Student = yes

Credit_rating = Fair)

Example 2

P(C_i): P(buys_computer = “yes”) = 9/14 = 0.643

P(buys_computer = “no”) = 5/14= 0.357

Compute P(X|C_i) for each class

- P(age = “<=30” | buys_computer = “yes”) = 2/9 = 0.222
- P(age = “<= 30” | buys_computer = “no”) = 3/5 = 0.6
- P(income = “medium” | buys_computer = “yes”) = 4/9 = 0.444
- P(income = “medium” | buys_computer = “no”) = 2/5 = 0.4
- P(student = “yes” | buys_computer = “yes) = 6/9 = 0.667
- P(student = “yes” | buys_computer = “no”) = 1/5 = 0.2
- P(credit_rating = “fair” | buys_computer = “yes”) = 6/9 = 0.667
- P(credit_rating = “fair” | buys_computer = “no”) = 2/5 = 0.4

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

X = (age <= 30 , income = medium, student = yes, credit_rating = fair)

P(X|C_i) : P(X|buys_computer = “yes”) = 0.222 x 0.444 x 0.667 x 0.667 = 0.044

P(X|buys_computer = “no”) = 0.6 x 0.4 x 0.2 x 0.4 = 0.019

P(X|C_i)*P(C_i) : P(X|buys_computer = “yes”) * P(buys_computer = “yes”) = 0.028

P(X|buys_computer = “no”) * P(buys_computer = “no”) = 0.007

Therefore, X belongs to class (“buys_computer = yes”)

Avoiding the Zero-Probability Problem

- Naïve Bayesian prediction requires each conditional probability be **non-zero**. Otherwise, the predicted probability will be zero!

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i)$$

- Ex. Suppose a dataset with 1000 tuples, x_i =low (0), x_i = medium (990), and i x_i = high (10)
- Use **Laplacian correction** (or Laplacian estimator)
  - *Adding 1 to each case*

    Prob(x_i = low) = 1/1003

    Prob(x_i = medium) = 991/1003

    Prob(x_i = high) = 11/1003
  - The "corrected" prob. estimates are close to their "uncorrected" counterparts, hence classification accuracy is not affected much by the correction

Naïve Bayes Classifier: Comments

- Advantages
  - Easy to implement, especially for categorical data
  - Very fast to train/test

- Disadvantages
  - Assumption: class conditional independence, therefore loss of accuracy
  - Practically, dependencies exist among variables, which cannot be modeled by the classifier
    - E.g., hospitals: patients: Profile: age, family history, etc.
      Symptoms: fever, cough etc., Disease: lung cancer, diabetes, etc.

Support Vector Machine (SVM)

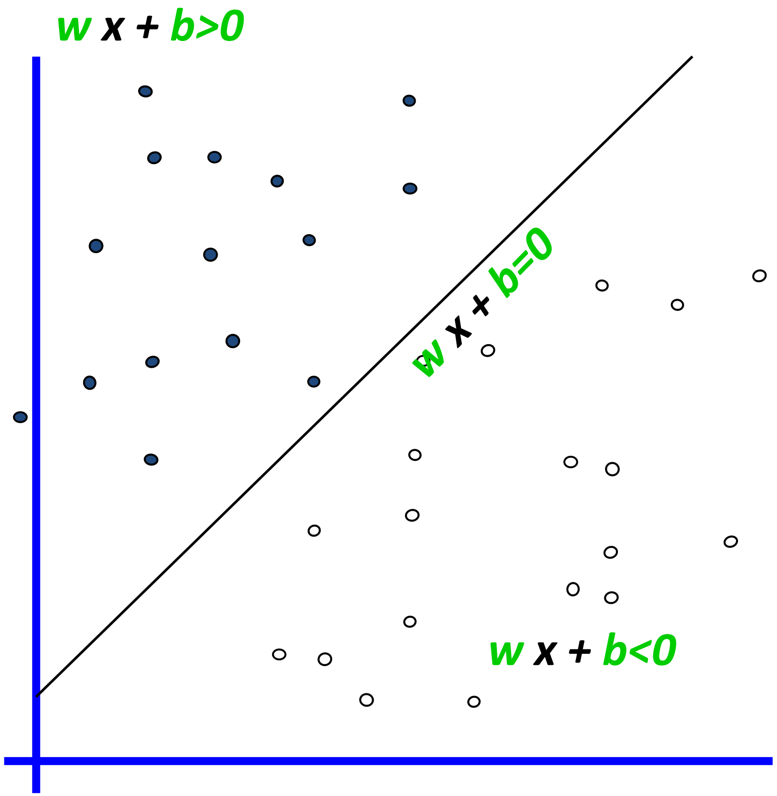
- non-probabilistic binary linear classifier
- constructs a hyperplane or set of hyperplanes in a high-dimensional space
- can be used for either classification or regression

Linear classifier

$$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\,\mathbf{x} + b)$$

● : +1

○ : -1



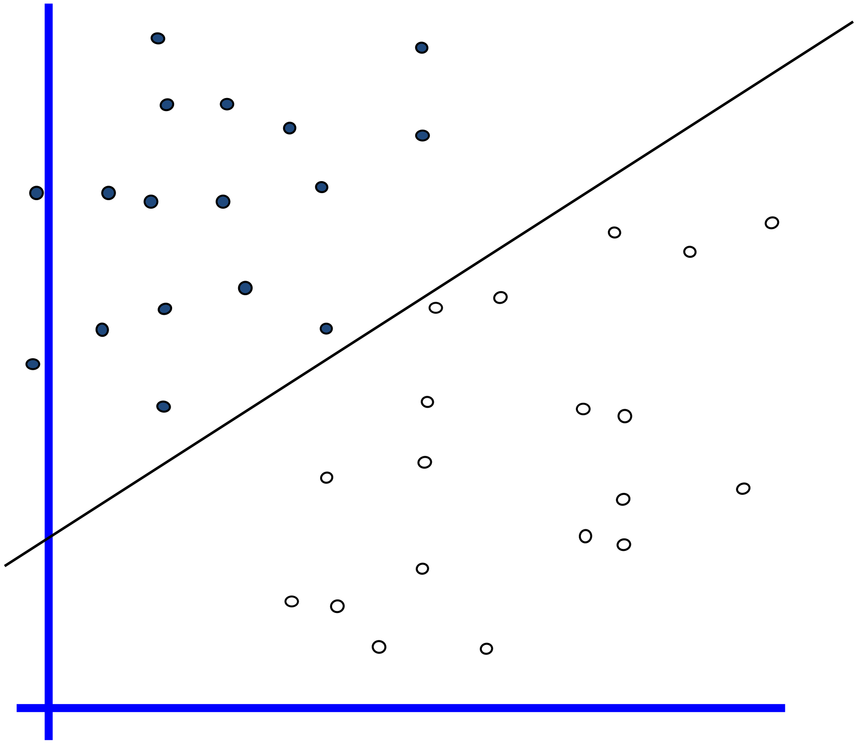
How would you classify this data?

Linear classifier

$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\,\mathbf{x} + b)$

● : +1

○ : -1



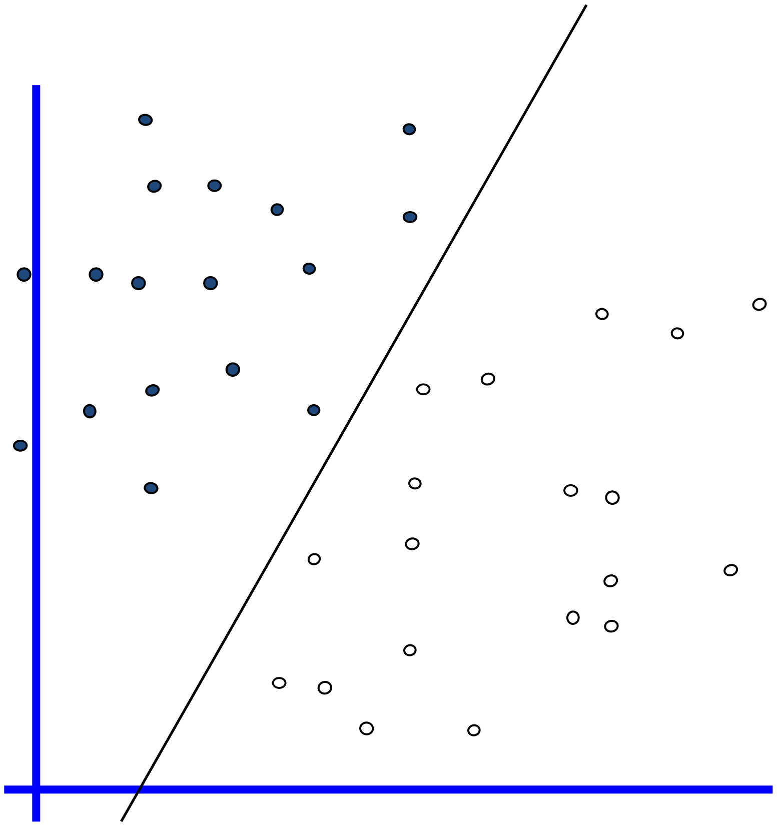
How would you classify this data?

Linear classifier

$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\,\mathbf{x} + b)$

● : +1

○ : -1



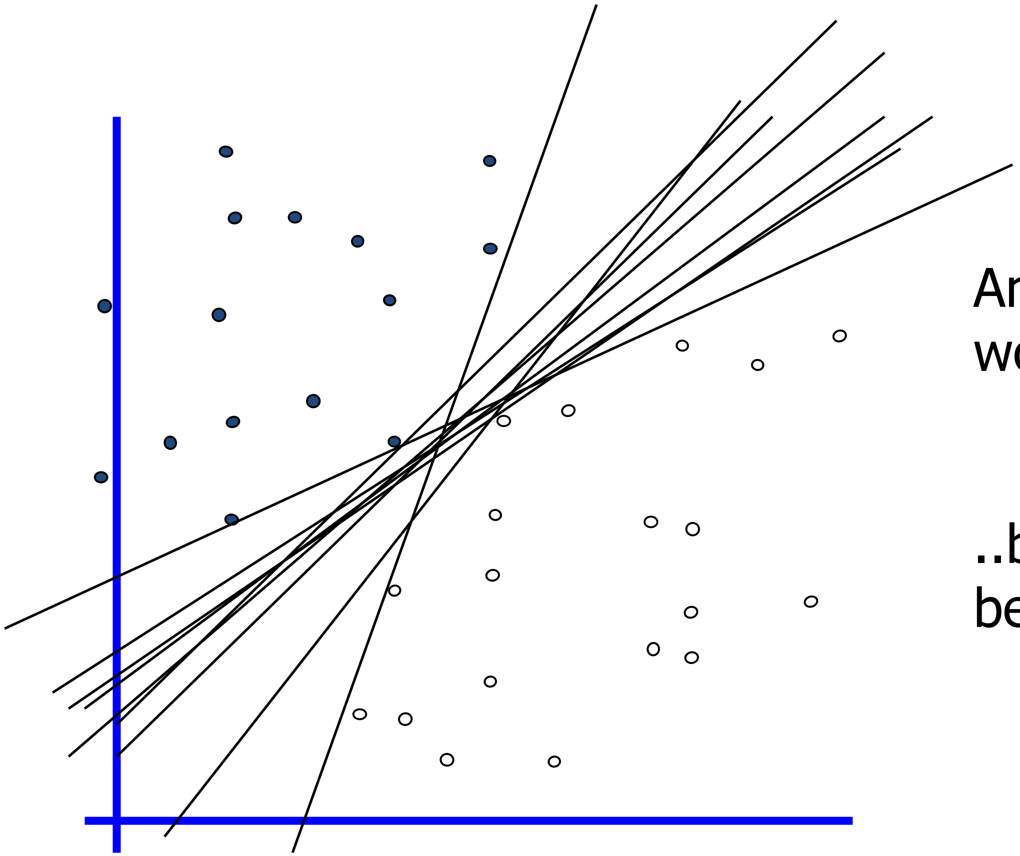
How would you classify this data?

Linear classifier

$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\,\mathbf{x} + b)$

● : +1

○ : -1



Any of these would be fine..

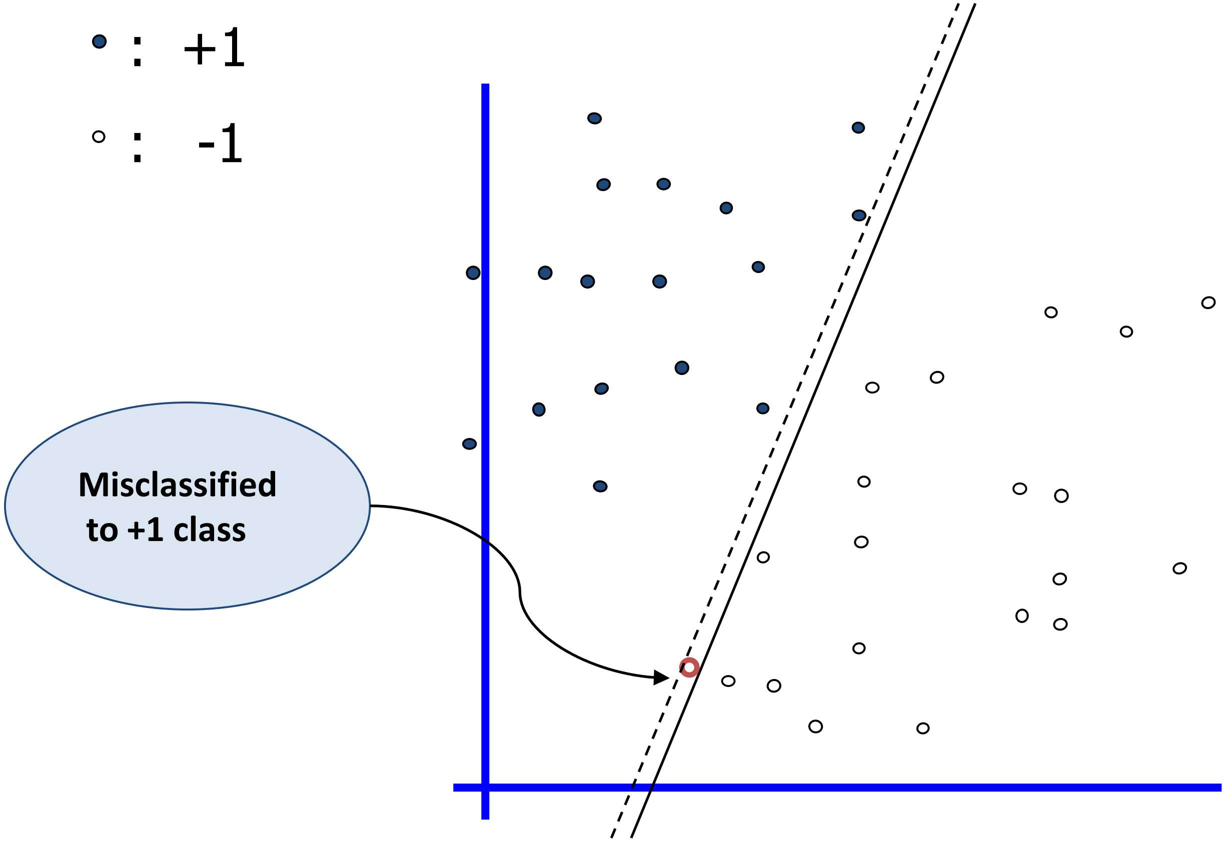
..but which is best?

Linear classifier

$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}\,\mathbf{x} + b)$

● : +1

○ : -1



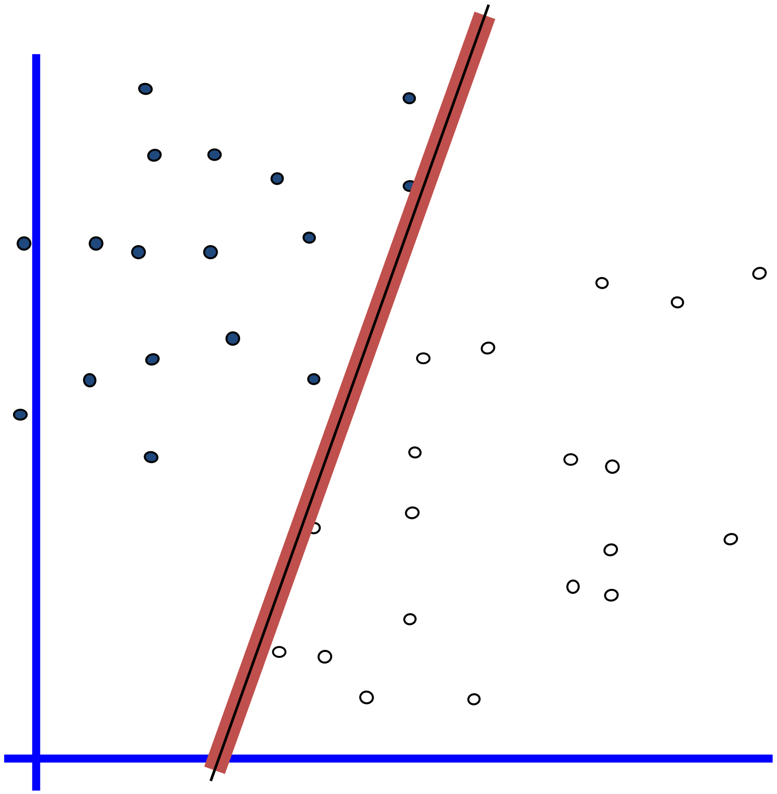
How would you classify this data?

Linear classifier

$$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\,\mathbf{x} + b)$$

● : +1

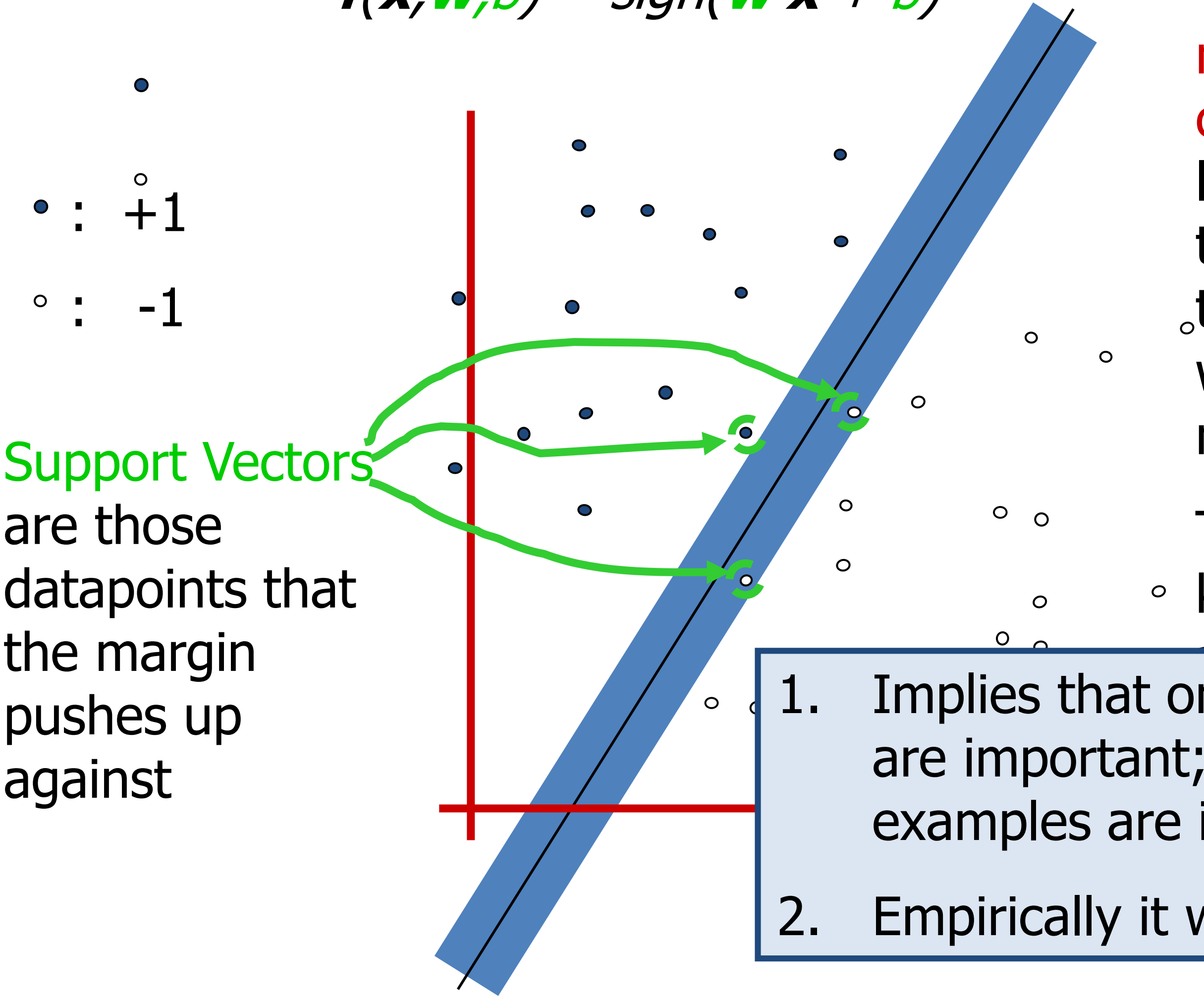
○ : -1



Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Maximum margin

$$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\,\mathbf{x} + b)$$



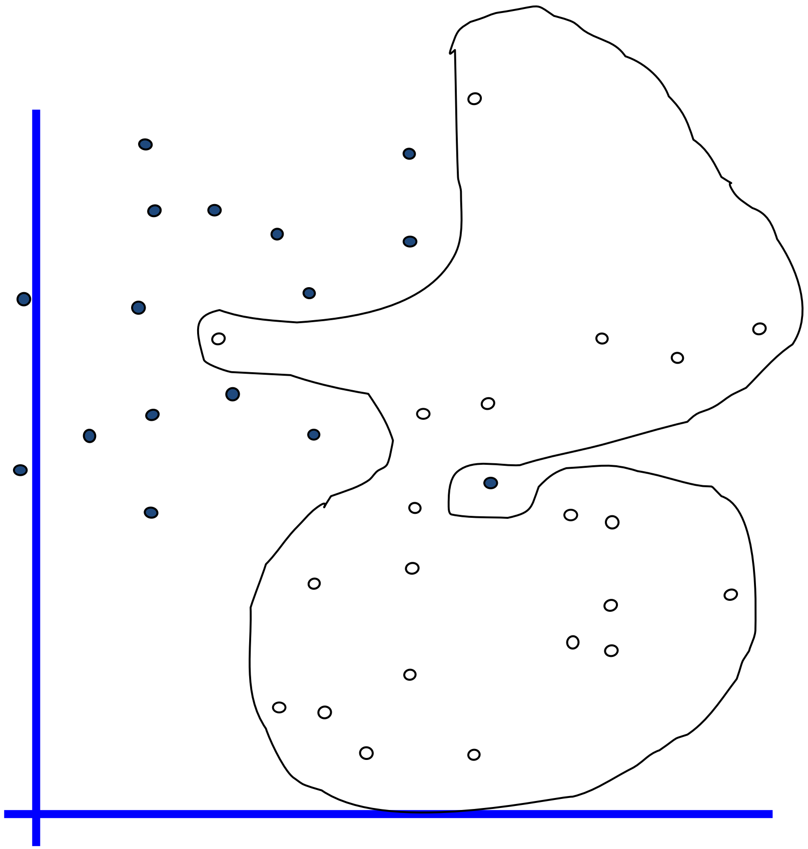
Support Vectors are those datapoints that the margin pushes up against

The maximum margin linear classifier is the linear classifier able to separate the training data points with the maximum margin.

This is the simplest kind of SVM (Called an LSVM)

1. Implies that only support vectors are important; other training examples are ignorable.
2. Empirically it works very very well.

Hard Margin classification



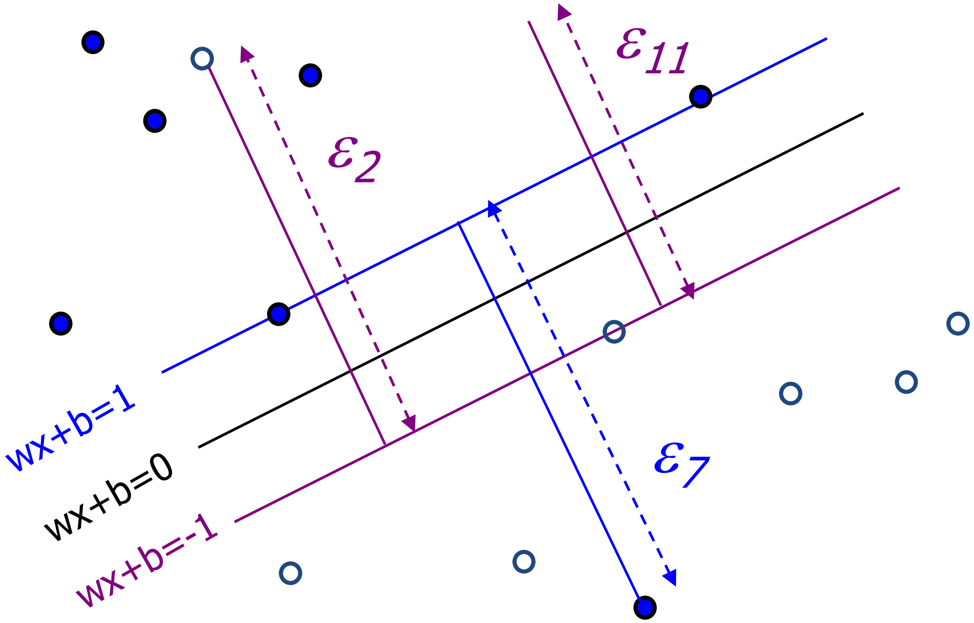
- **Hard Margin:** Requires all the training data points to be classified correctly

  - No training error

- **What if the training set is noisy?**

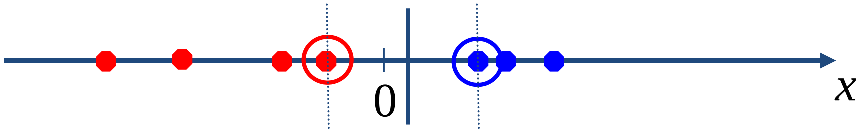
Soft Margin Classification

Slack variables *ξi* can be added to allow the misclassification of difficult or noisy examples (decreased overfitting).

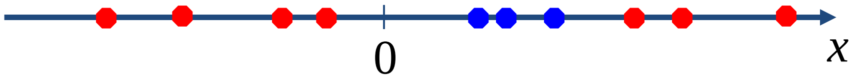


Nonlinear SVMs

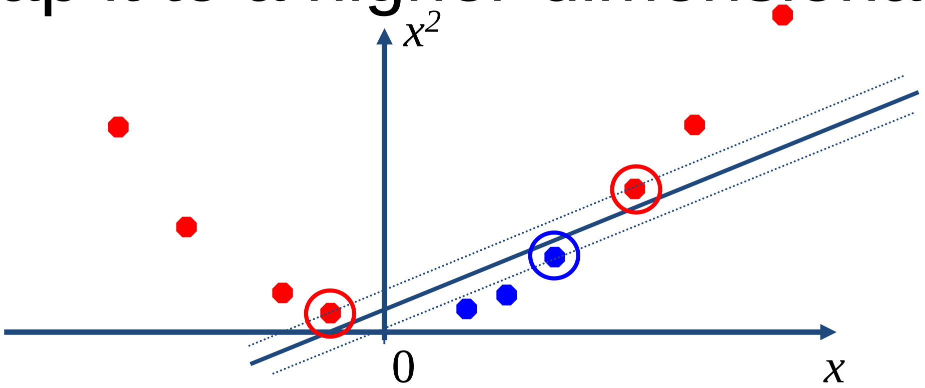
- Datasets that are linearly separable work out great:



- But what if the dataset is just too hard?

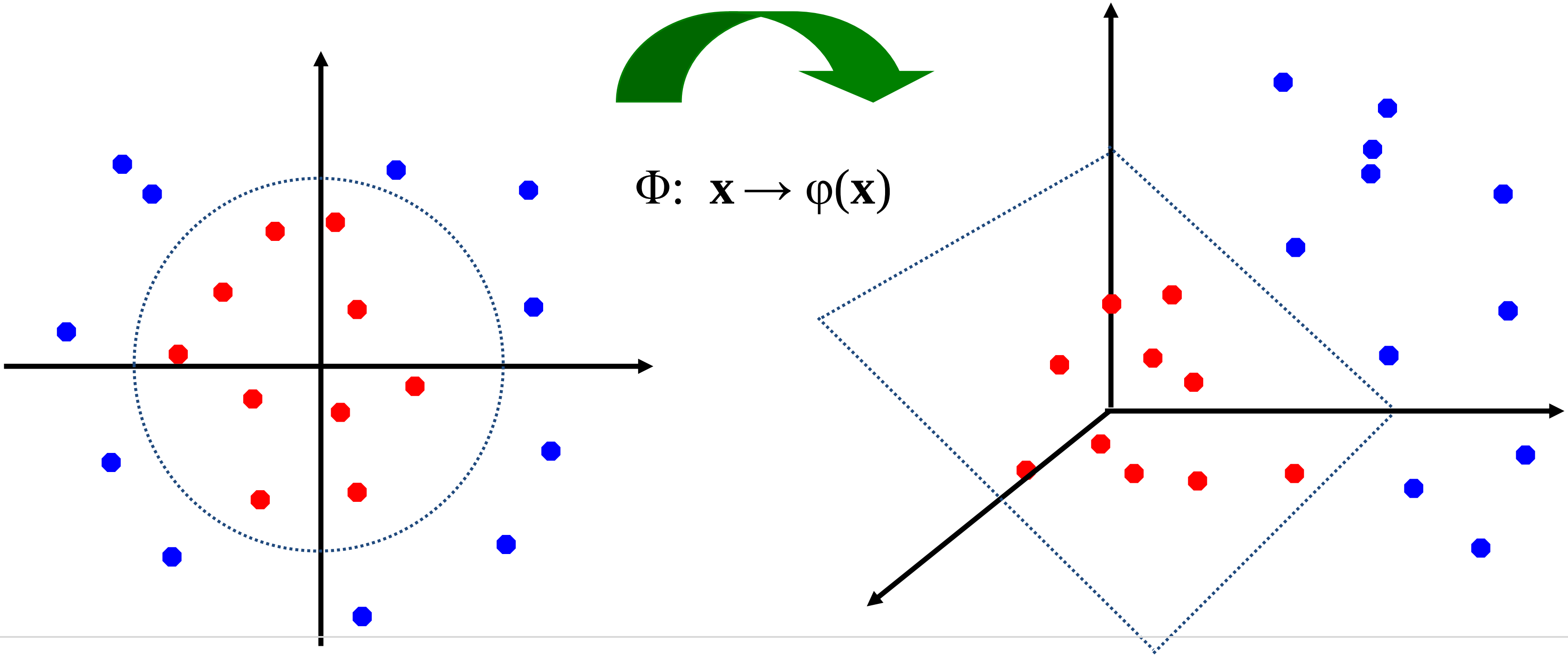


- We can map it to a higher-dimensional space:



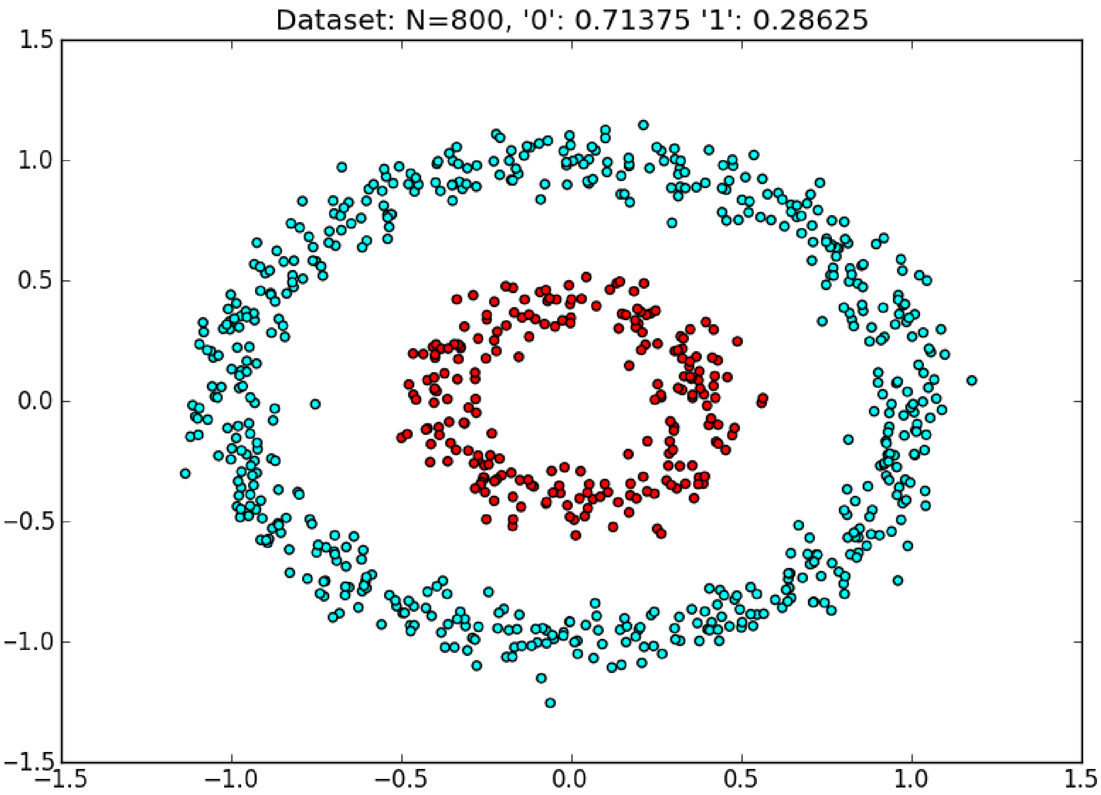
Nonlinear SVMs

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



Slide credit: Andrew Moore

Nonlinear SVMs - example



A two-class dataset that is not linearly separable. The outer ring (cyan) is class '0', while the inner ring (red) is class '1'

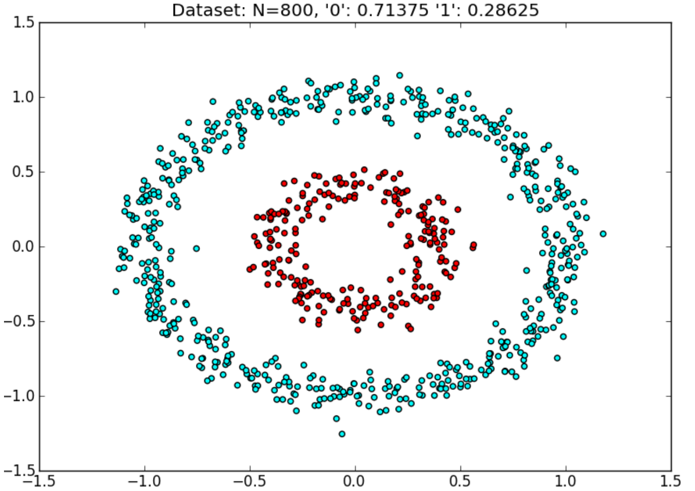
Nonlinear SVMs - example

If we try to apply a linear SVM… 44.5% accuracy

```
==== Evaluating SVM (kernel='linear'), 2-fold cross validation
    Parameters to be chosen through cross validation:
        C: [1.0, 10.0, 100.0, 1000.0, 10000.0]
== Best Params: {'kernel': 'linear', 'C': 1.0}
== Best Score: 0.476666666667
== Accuracy: 0.445
             precision    recall  f1-score  support
          0       0.78      0.37      0.50      151
          1       0.26      0.67      0.37       49

avg / total       0.65      0.45      0.47      200

==== Finished Linear SVM (1.290 s)
```

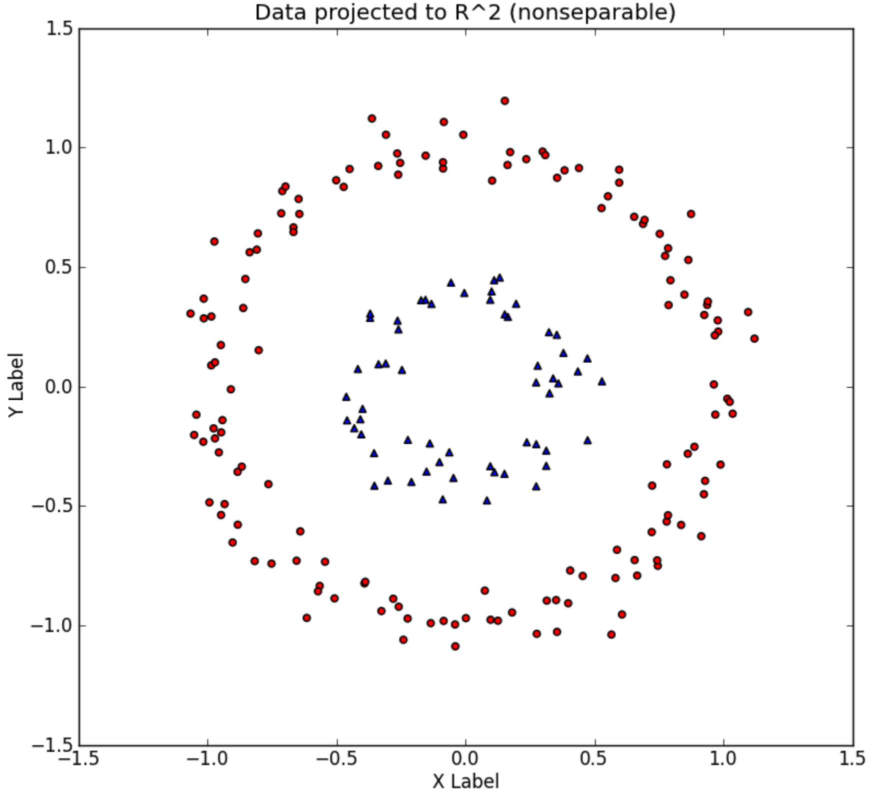


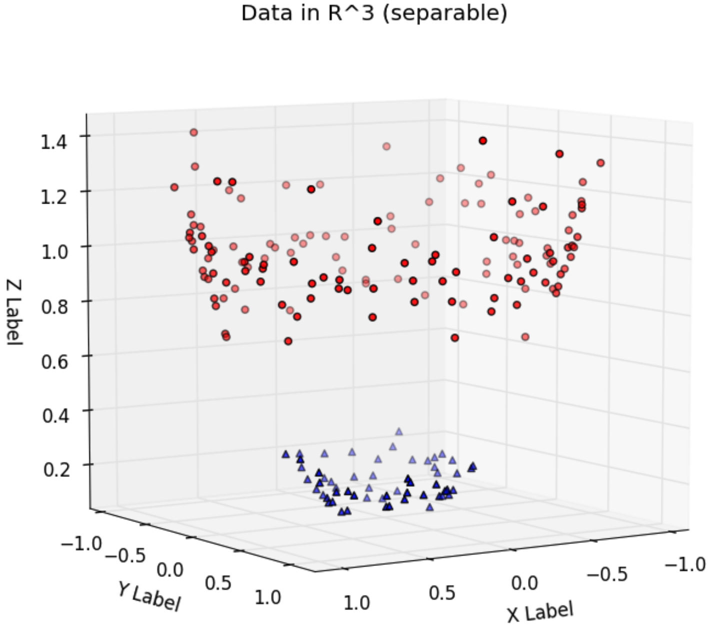
… less than a Random Classifier!

```
==== Evaluating Random Classifier
== Accuracy: 0.45
             precision    recall  f1-score support
          0       0.74      0.42      0.53     151
                  1       0.23      0.55      0.33      49
avg / total       0.62      0.45      0.48     200

==== Finished Random Classifier (0.000 s)
```

Nonlinear SVMs - example

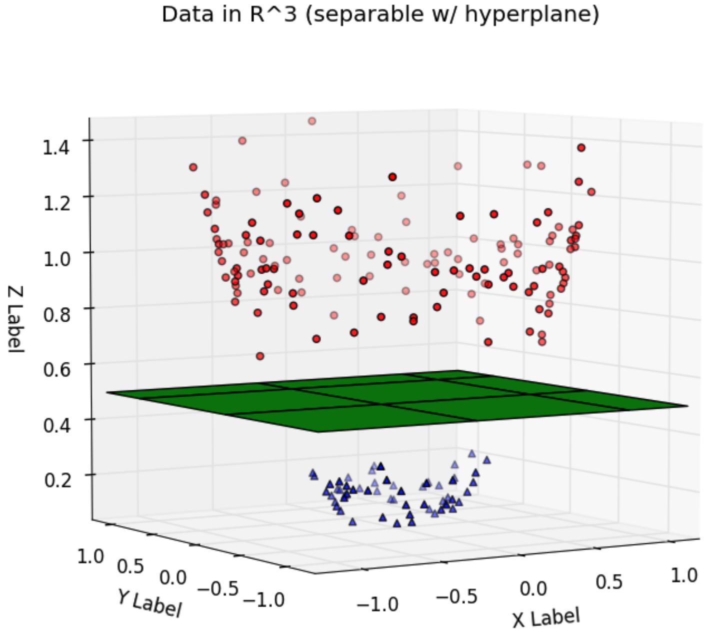


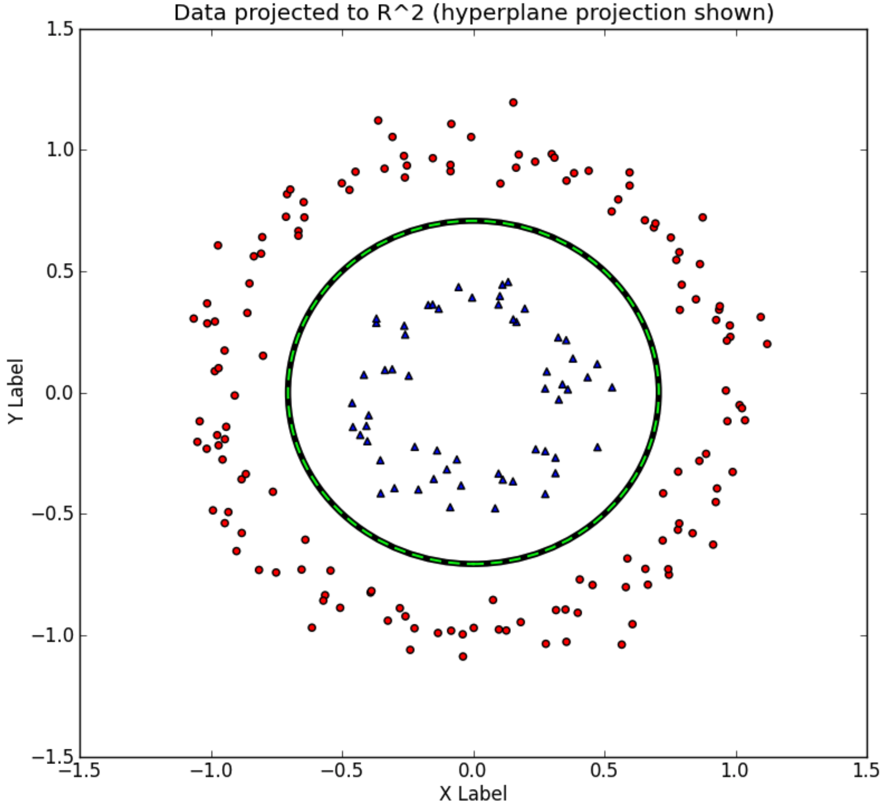


The same dataset transformed by the transformation:

$[x_1,x_2] = [x_1, x_2, x_1^2 + x_2^2]$ becomes linearly separable

Nonlinear SVMs - example





A dataset that is not linearly separable in $\mathbb{R}^N$ may be linearly separable in a higher-dimensional space $\mathbb{R}^M$ (where M > N). Thus, if we have a transformation φ that lifts the dataset *D* to a higher-dimensional *D'* such that *D'* is linearly separable, then we can train a linear SVM on *D'*. Projecting the decision boundary found in $\mathbb{R}^M$ back to the original space $\mathbb{R}^N$ will yield a nonlinear decision boundary.

Nonlinear SVMs

- The scheme described so far is attractive due to its theoretical simplicity.
- However, consider the computational and memory consequences of increasing the dimensionality from N to M!
- … fortunately, SVM has no need to explicitly work in the higher-dimensional space at training or testing time!

Nonlinear SVMs

- It can be shown that during the training, the optimization problem only uses the training examples to compute pair-wise **dot products** $\langle \vec{\boldsymbol{x}}_{\boldsymbol{i}} , \vec{\boldsymbol{x}}_{\boldsymbol{j}} \rangle$**,** where $\vec{\boldsymbol{x}}_{\boldsymbol{i}} , \vec{\boldsymbol{x}}_{\boldsymbol{j}} \in \mathbb{R}^N$

- There exist functions that, given two vectors v and w in $\mathbb{R}^N$, implicitly compute the dot product between v and w in a **higher-dimensional** $\mathbb{R}^M$ **without explicitly transforming v and w to** $\mathbb{R}^M$. Such functions are called **kernel** functions, $\mathrm{K}(\vec{\boldsymbol{v}} , \vec{\boldsymbol{w}})$

- In symbols:

For $\vec{\boldsymbol{x}}_{\boldsymbol{i}} , \vec{\boldsymbol{x}}_{\boldsymbol{j}} \in \mathbb{R}^N$, $\mathrm{K}(\vec{\boldsymbol{x}}_{\boldsymbol{i}} , \vec{\boldsymbol{x}}_{\boldsymbol{j}}) = \langle \boldsymbol{\phi}(\vec{\boldsymbol{x}}_{\boldsymbol{i}}), \boldsymbol{\phi}(\vec{\boldsymbol{x}}_{\boldsymbol{j}}) \rangle_M$

wehere $\langle . , .) \rangle_M$ is an inner product in $\mathbb{R}^M$ and $\boldsymbol{\phi}(\vec{\boldsymbol{x}}_{\boldsymbol{i}})$ is the transformation function that maps $\vec{\boldsymbol{x}}_{\boldsymbol{i}}$ *from* $\mathbb{R}^N$ to $\mathbb{R}^M$

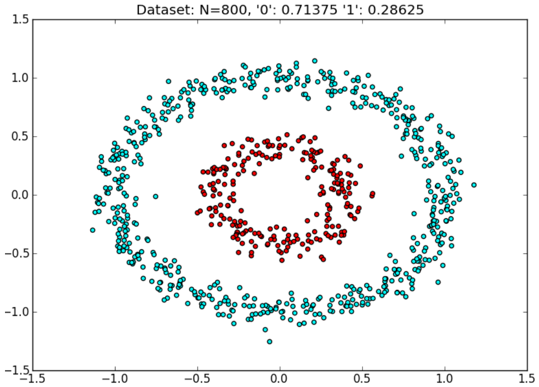
The kernel trick

1. By using a kernel function $\mathrm{K}(\overrightarrow{\boldsymbol{x}_i}, \overrightarrow{\boldsymbol{x}_j})$ we can **implicitly** transform datasets to a higher-dimensional space using no extra memory, and with a minimal effect on computation time
2. The only effect on computation is the extra time required to compute $\mathrm{K}(\overrightarrow{\boldsymbol{x}_i}, \overrightarrow{\boldsymbol{x}_j})$
3. By virtue of (1), we can efficiently learn nonlinear decision boundaries for SVMs simply by **replacing all dot products** $\langle\overrightarrow{\boldsymbol{x}_i}, \overrightarrow{\boldsymbol{x}_j}\rangle$ **in the optimization problem with** $\mathrm{K}(\overrightarrow{\boldsymbol{x}_i}, \overrightarrow{\boldsymbol{x}_j})$

The kernel trick

- Unfortunately, choosing the 'correct' kernel is a nontrivial task, and may depend on the specific task at hand: Polynomial kernel, Radial Basis Function (RBF) kernel, Sigmoid kernel, …
- No matter which kernel it is chosen, it is necessary to **tune the kernel parameters** to get good performance (K-Fold Cross Validation, grid-search)
- For example, for RBF kernel (most standard kernel function)
  - **Gamma**: defines how far the influence of a single training example reaches
  - **C**: trades off misclassification of training examples against simplicity of the decision surface.
- Best practice:
  - Define a grid of (C_{min} : C_{step}: C_{max}, $gamma_{min}$: $gamma_{step}$: $gamma_{max}$) pairs, with default C,gamma in the center and large step values
  - Compute crossvalidation accuracy per each pair of parameters values, chose the best ones
  - Define a finer grid around the optimal pair
  - Go on until convergence

Back to the example…



```
==== Evaluating SVM (kernel='rbf'), 2-fold cross validation
   Parameters to be chosen through cross validation:
      C: [1.0, 10.0, 100.0, 1000.0, 10000.0]
      gamma: [0.0001, 0.001, 0.01, 0.1]
== Best Params: {'kernel': 'rbf', 'C': 10.0, 'gamma': 0.1}
== Best Score: 1.0
== Accuracy: 1.0
          precision    recall  f1-score   support

        0      1.00      1.00      1.00       151
        1      1.00      1.00      1.00        49

avg / total      1.00      1.00      1.00       200

==== Finished RBF Kernel (0.555 s)
```

What about multi-class SVMs?

- Unfortunately, there is no “definitive” multi-class SVM formulation
- In practice, we have to obtain a multi-class SVM by combining multiple two-class SVMs
- One vs. others
  - Traning: learn an SVM for each class vs. the others
  - Testing: apply each SVM to test example and assign to it the class of the SVM that returns the highest decision value
- One vs. one
  - Training: learn an SVM for each pair of classes
  - Testing: each learned SVM “votes” for a class to assign to the test example

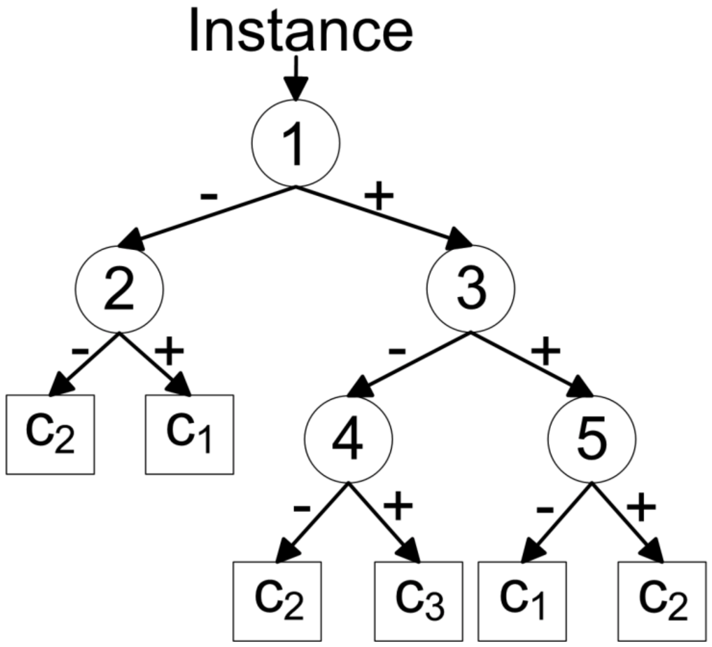
Slide credit: L. Lazebnik

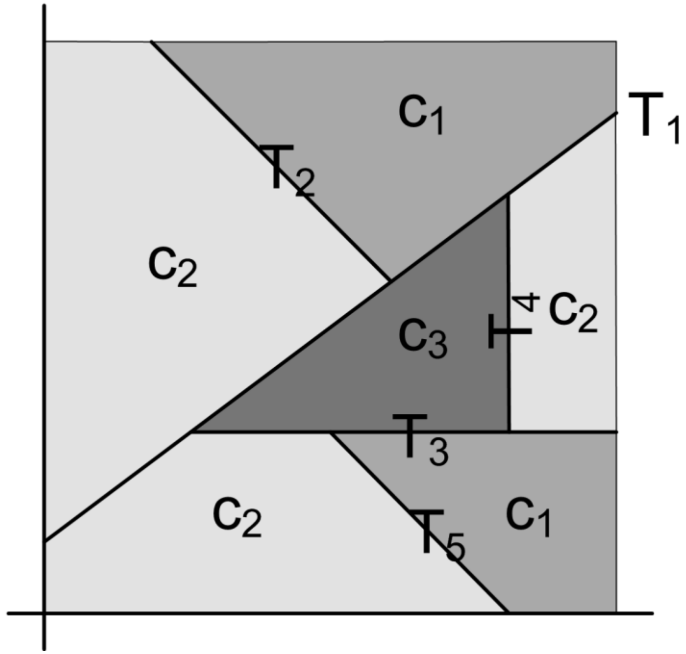
SVMs: Pros and cons

- Pros
  - Many publicly available SVM packages: http://www.kernel-machines.org/software (e.g. LibSVM)
  - Kernel-based framework is very powerful, flexible
  - SVMs work very well in practice, even with very small training sample sizes
- Cons
  - No “direct” multi-class SVM, must combine two-class SVMs
  - Computation, memory
    - Learning can take a very long time for large-scale problems
  - Tuning kernel’s parameters might be tricky

Decision trees

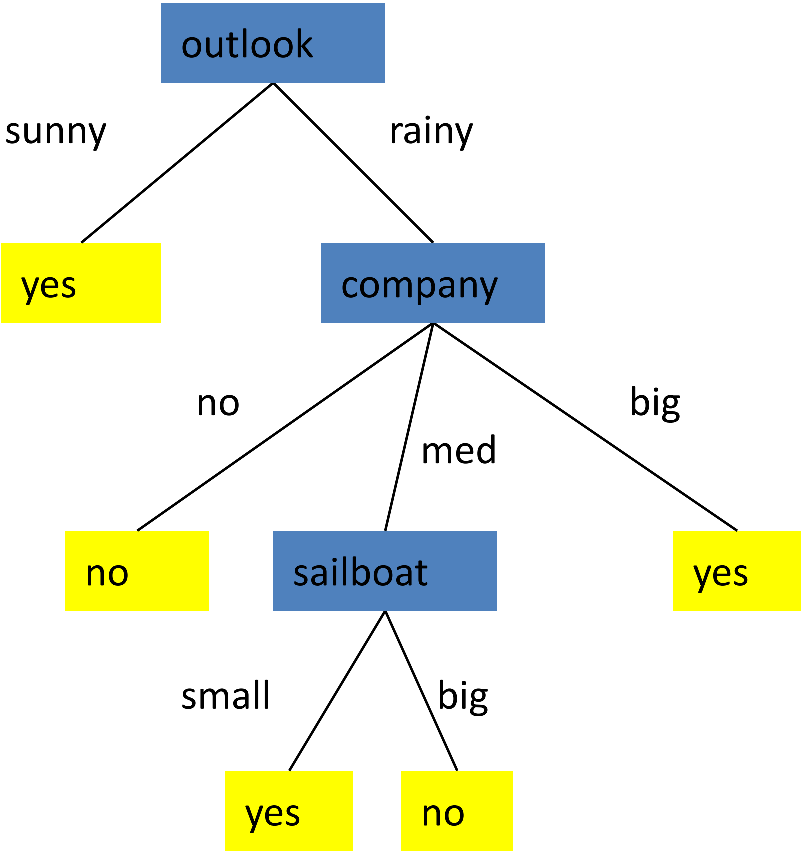
- The classifier is represented by a decision tree of finite depth. Every **node** of the tree specifies a test involving an **attribute** (i.e. one element of the feature vector), and every **branch** descending from a node matches one of the possible outcomes of the test (i.e. one of the possible valies of the corresponding attribute).
- Classifying an instance means performing a sequence of tests, starting with the root node and terminating with a leaf node.





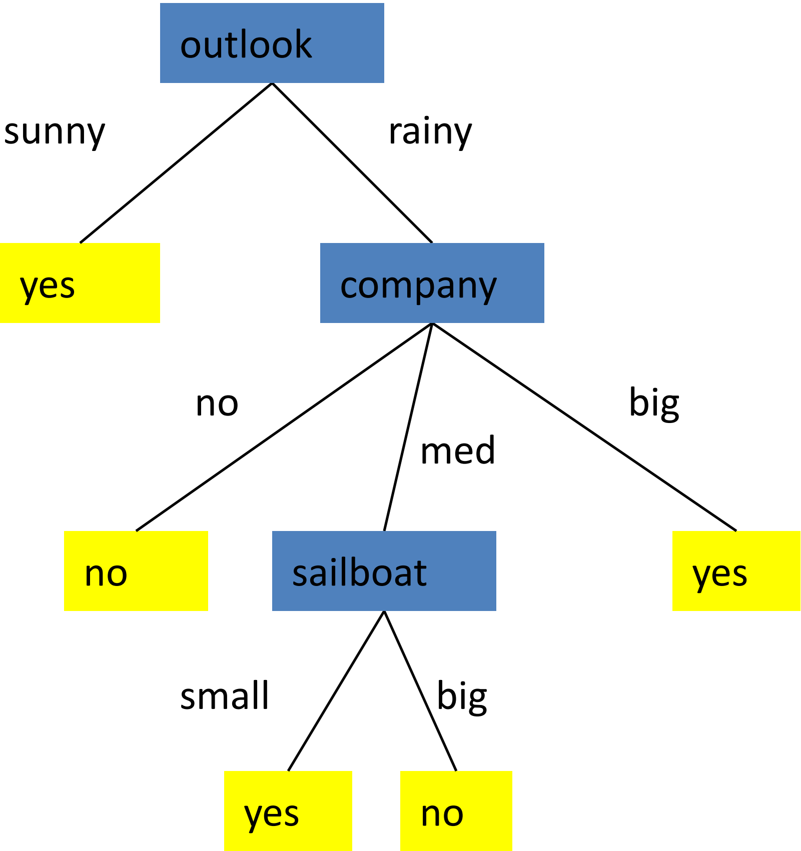
Example – Training set

#	*Attribute*			*Class*
	Outlook	Company	Sailboat	Sail?
1	sunny	big	small	yes
2	sunny	med	small	yes
3	sunny	med	big	yes
4	sunny	no	small	yes
5	sunny	big	big	yes
6	rainy	no	small	no
7	rainy	med	small	yes
8	rainy	big	big	yes
9	rainy	no	big	no
10	rainy	med	big	no



Example - Classification

#	*Attribute*			*Class*
	Outlook	Company	Sailboat	Sail?
1	sunny	no	big	?
2	rainy	big	small	?



Induction of Decision Trees

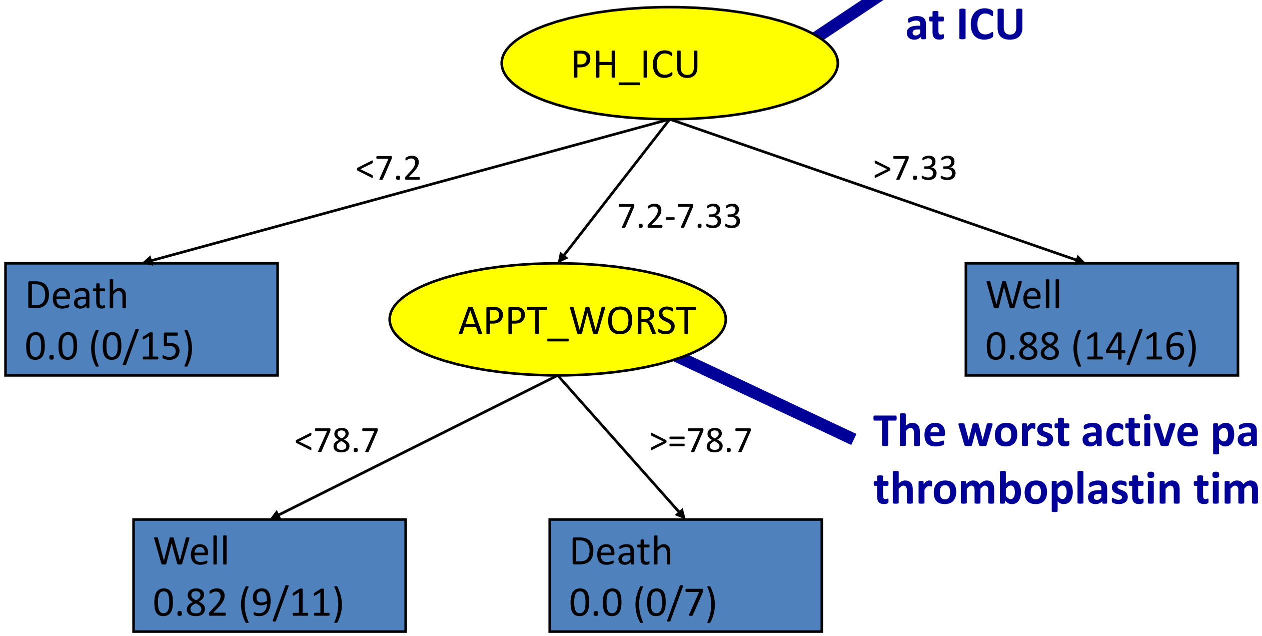
- Data Set (Training Set)
  - Each instance of the training = Attributes + Class
- Induced description = Decision tree

- TDIDT
  - Top Down Induction of Decision Trees

Some TDIDT Systems

- ID3 (Quinlan 79)
- CART (Brieman et al. 84)
- Assistant (Cestnik et al. 87)
- C4.5 (Quinlan 93)
- See5 (Quinlan 97)
- ...
- Orange (Demšar, Zupan 98-03)

Analysis of Severe Trauma Patients

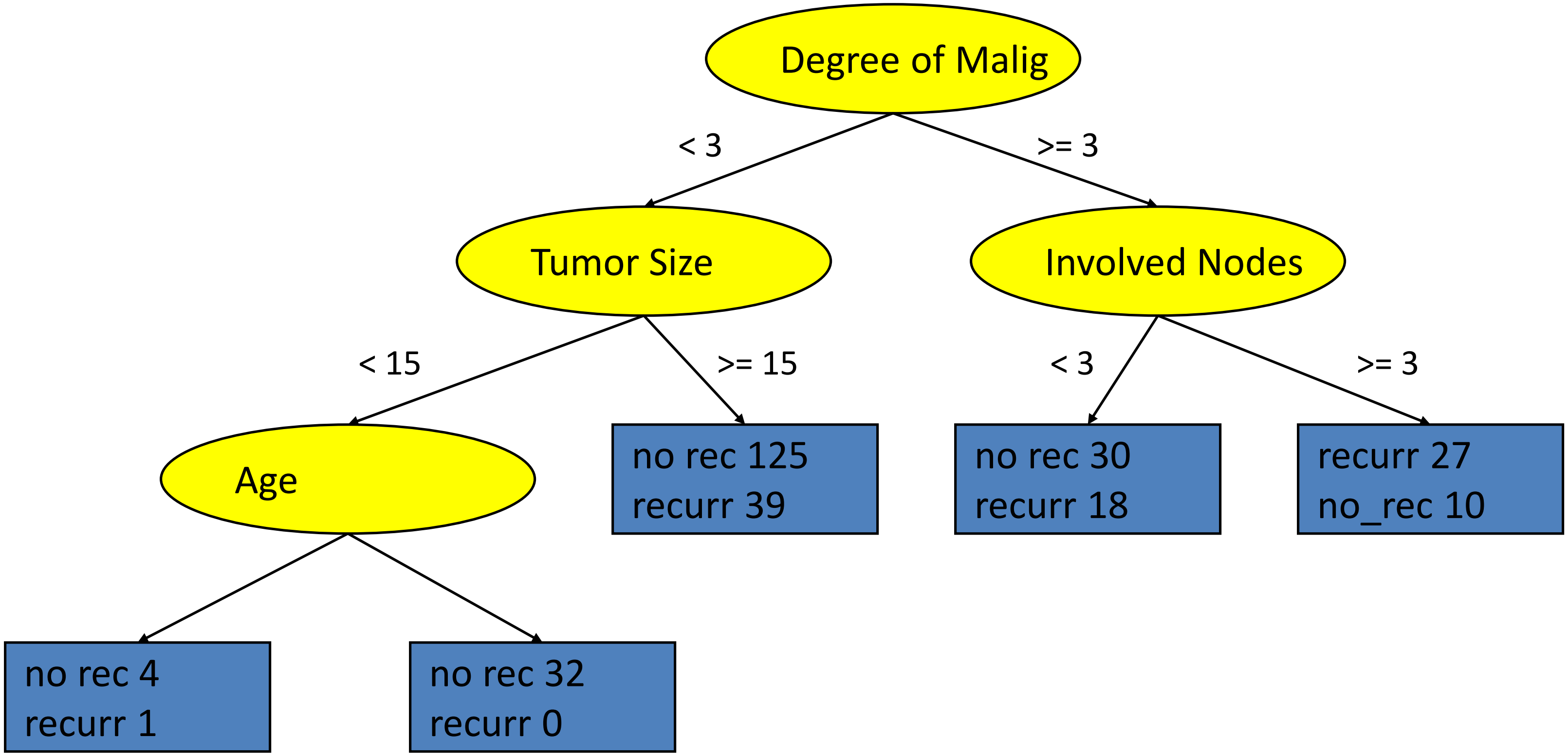


The worst pH value at ICU

The worst active partial thromboplastin time

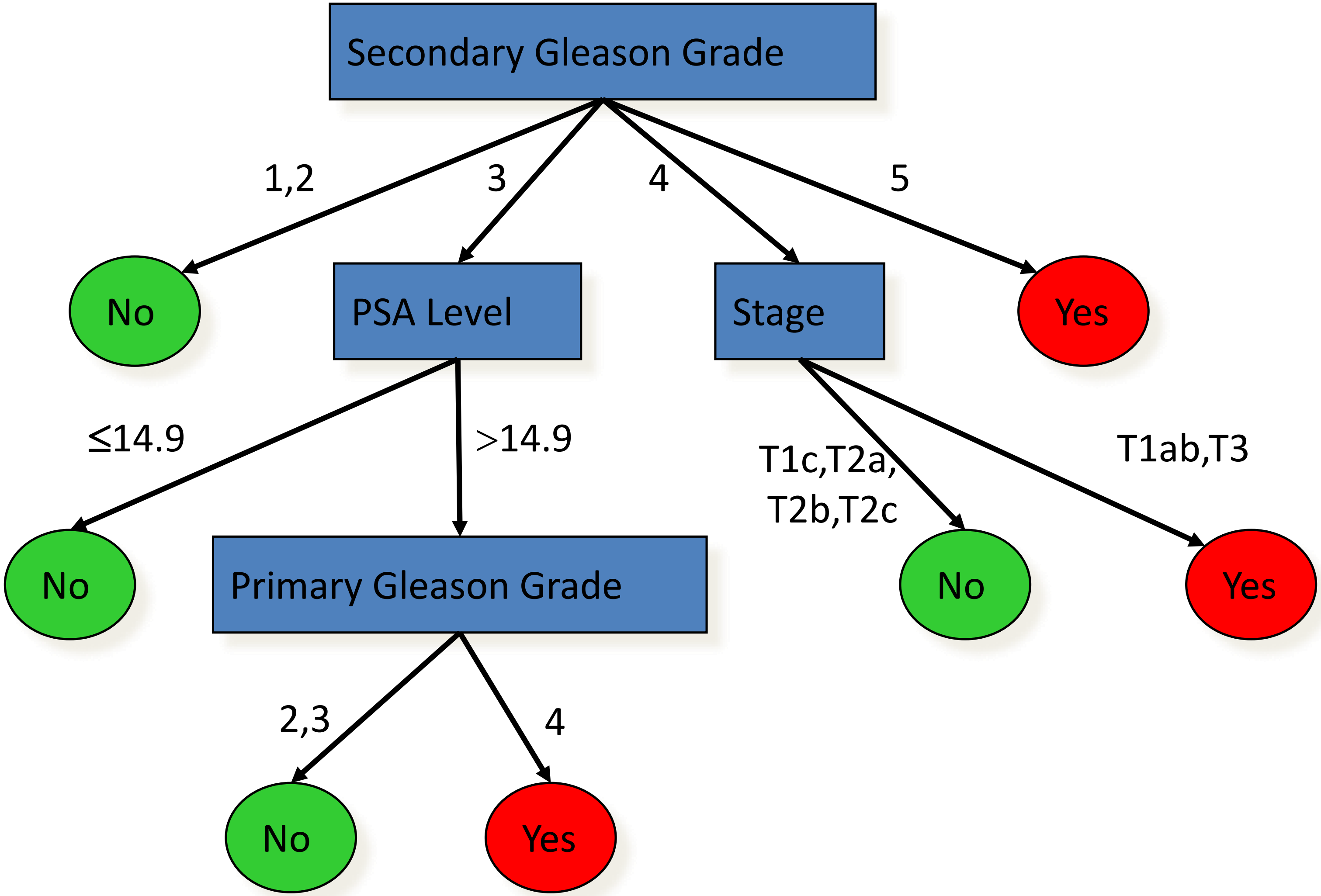
PH_ICU and APPT_WORST are exactly the two factors (theoretically) advocated to be the most important ones in the study by Rotondo et al., 1997.

Breast Cancer Recurrence



Interesting: Accuracy of this tree compared to medical specialists

Prostate cancer recurrence

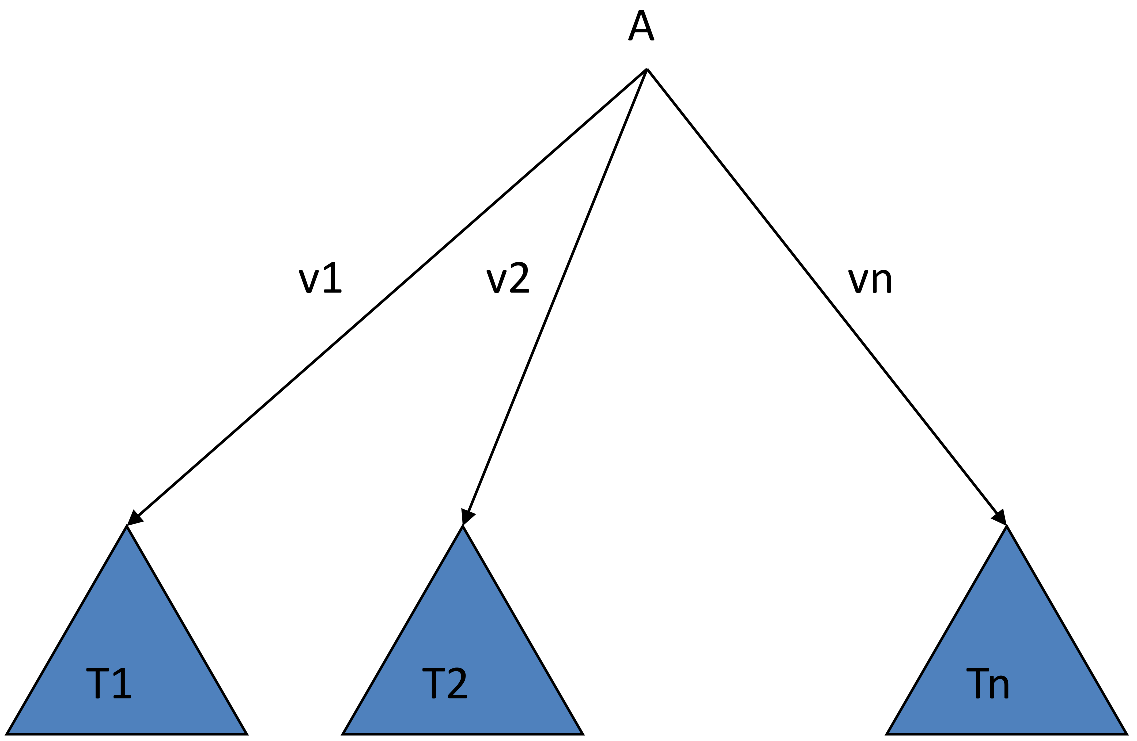


TDIDT Algorithm

- Also known as ID3 (Quinlan)
- To construct decision tree T from learning set S:
  - **If** all examples in S belong to some class C **Then** make leaf labeled C
  - **Otherwise**
    - select the *“most informative”* attribute A
    - partition S according to A’s values
    - recursively construct subtrees T1, T2, ..., for the subsets of S (one per each value of A)

TDIDT Algorithm

- Resulting tree T is:



Attribute A

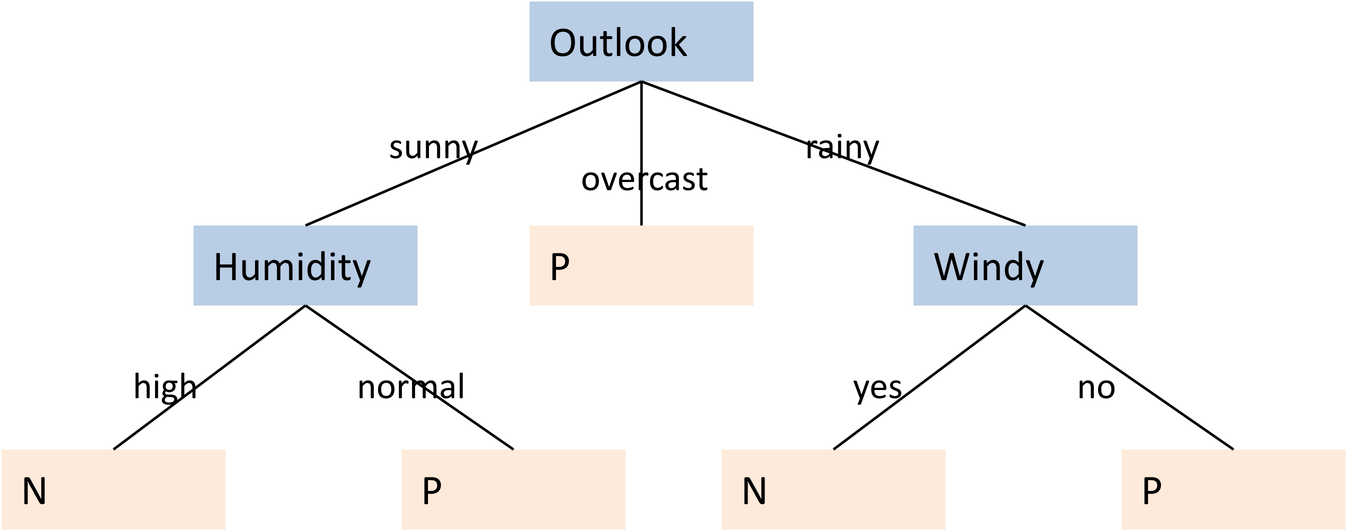
A's values

Subtrees

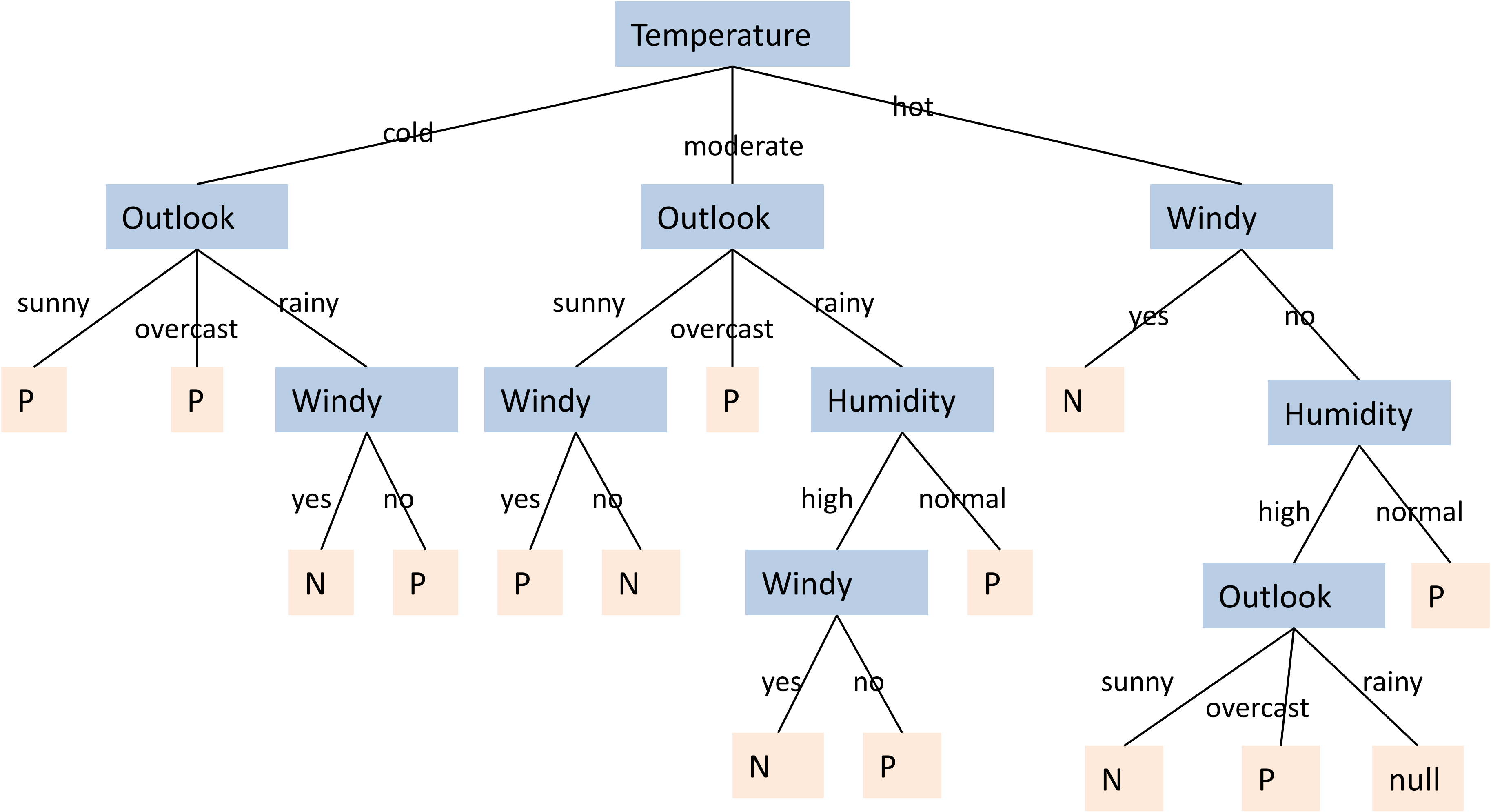
Another Example

#	Attribute				Class
	Outlook	Temperature	Humidity	Windy	Play
1	sunny	hot	high	no	N
2	sunny	hot	high	yes	N
3	overcast	hot	high	no	P
4	rainy	moderate	high	no	P
5	rainy	cold	normal	no	P
6	rainy	cold	normal	yes	N
7	overcast	cold	normal	yes	P
8	sunny	moderate	high	no	N
9	sunny	cold	normal	no	P
10	rainy	moderate	normal	no	P
11	sunny	moderate	normal	yes	P
12	overcast	moderate	high	yes	P
13	overcast	hot	normal	no	P
14	rainy	moderate	high	yes	N

Simple Tree


Outlook
sunny
overcast
rainy
Humidity
P
Windy
high
normal
yes
no
N
P
N
P

Complicated Tree



Attribute Selection Criteria

- Main principle
  - Select attribute which partitions the learning set into subsets as “pure” as possible (i.e. homogeneous in terms of class label)
- Various measures of purity
  - Information-theoretic
  - Gini index
  - X^2
  - ReliefF
  - ...
- Various improvements
  - probability estimates
  - normalization
  - binarization, subsetting

Information-Theoretic Approach

- To classify an object, a certain information is needed
  - I, information
- After we have learned the value of attribute A, we only need some remaining amount of information to classify the object
  - Ires, residual information
- Gain
  - Gain(A) = I – Ires(A)
- The most ‘informative’ attribute is the one that minimizes Ires, *i.e.*, maximizes Gain

Entropy

- The average amount of information *I* needed to classify an object is given by the **entropy** measure

$$I = -\sum_{c} p(c) \log_2 p(c)$$

Where p(c) is the proportion of examples of class c

- If all the examples belong to the same category, the entropy is 0.

- If the examples are equally mixed (1/c examples of each class), the entropy is maximum (1.0)

e.g. for c=2, -.5 $\log_2$ .5 - .5 $\log_2$ .5 = -.5(-1) -.5(-1) = 1

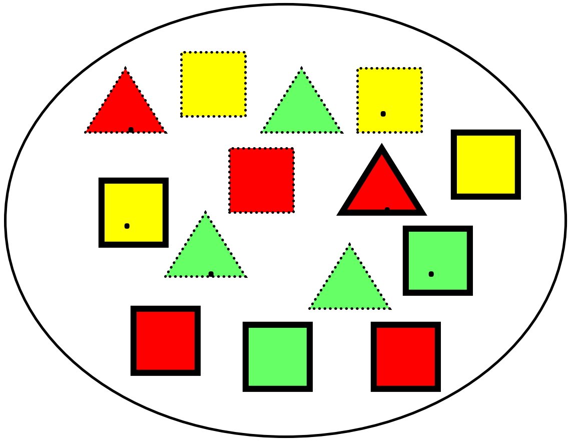
Residual Information

- After applying attribute *A*, *S* is partitioned into subsets according to values *v* of *A*
- *Ires* is equal to weighted sum of the amounts of information for the subsets

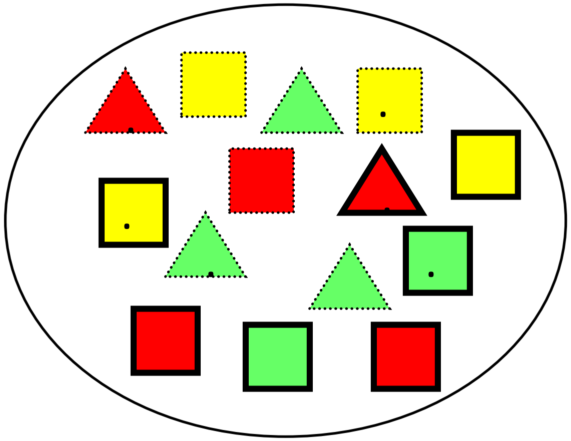
$$I_{res} = -\sum_{v} p(v) \sum_{c} p(c|v) \log_2 p(c|v)$$

Example: classify triangles/square

#	Attribute			Shape
	Color	Outline	Dot	
1	green	dashed	no	triange
2	green	dashed	yes	triange
3	yellow	dashed	no	square
4	red	dashed	no	square
5	red	solid	no	square
6	red	solid	yes	triange
7	green	solid	no	square
8	green	dashed	no	triange
9	yellow	solid	yes	square
10	red	solid	no	square
11	green	solid	yes	square
12	yellow	dashed	yes	square
13	yellow	solid	no	square
14	red	dashed	yes	triange



Entropy



- 5 triangles
- 9 squares
- class probabilities

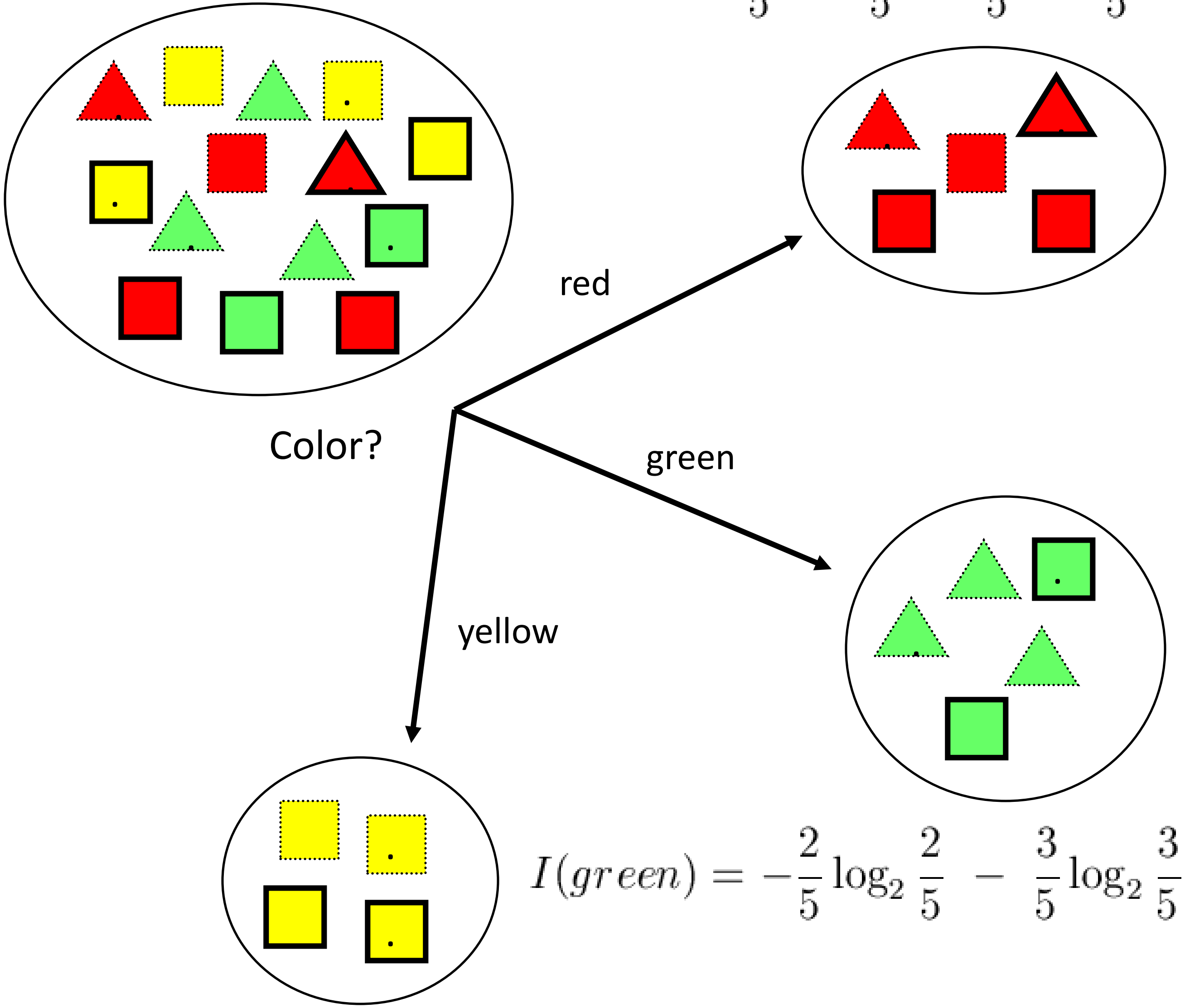
$$p(\square) = \frac{9}{14}$$

$$p(\triangle) = \frac{5}{14}$$

- entropy

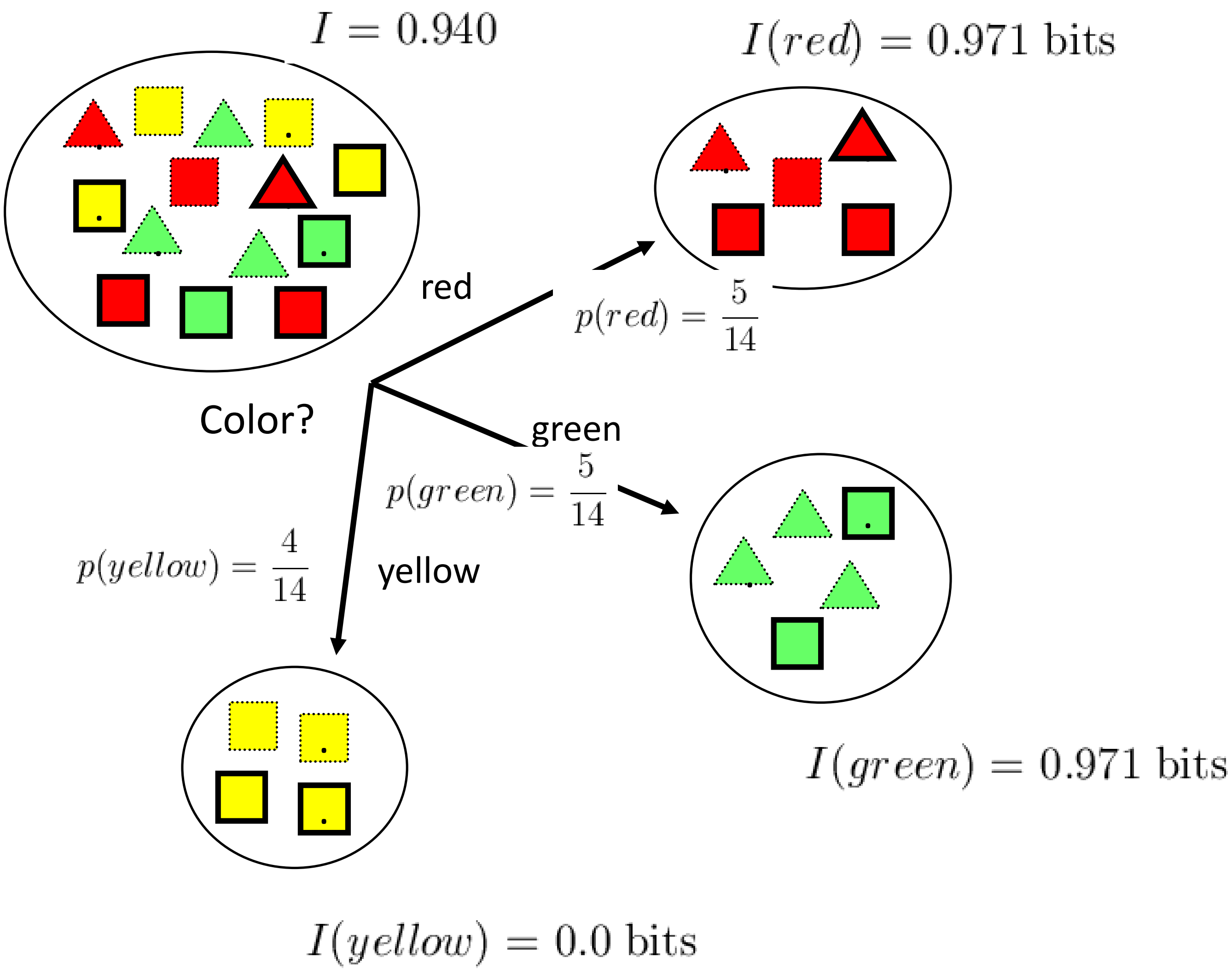
$$I = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940 \text{ bits}$$

$$I(red) = -\frac{3}{5}\log_2\frac{3}{5} \;-\; \frac{2}{5}\log_2\frac{2}{5} = 0.971 \text{ bits}$$



$$I(green) = -\frac{2}{5}\log_2\frac{2}{5} \;-\; \frac{3}{5}\log_2\frac{3}{5} = 0.971 \text{ bits}$$

$$I(yellow) = -\frac{4}{4}\log_2\frac{4}{4} \;-\; \frac{0}{4}\log_2\frac{0}{4} = 0.0 \text{ bits}$$



$I = 0.940$

$I(red) = 0.971$ bits

red

$p(red) = \frac{5}{14}$

Color?

green

$p(green) = \frac{5}{14}$

$p(yellow) = \frac{4}{14}$

yellow

$I(green) = 0.971$ bits

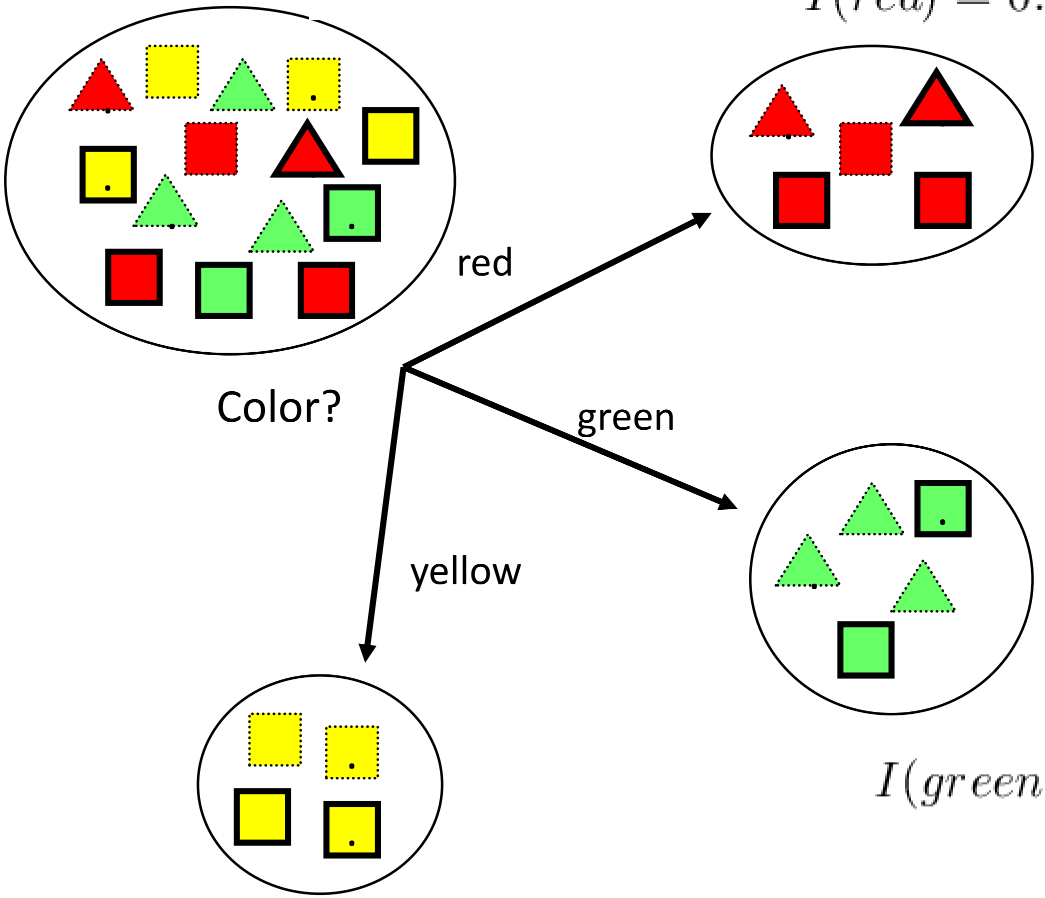
$I(yellow) = 0.0$ bits

$$I_{res}(\text{Color}) = \sum p(v) I(v) = \frac{5}{14}0.971 + \frac{5}{14}0.971 + \frac{4}{14}0.0 = 0.694\ bits$$

Information Gain

$I = 0.940$

$I(red) = 0.971$ bits



Color?

red

green

yellow

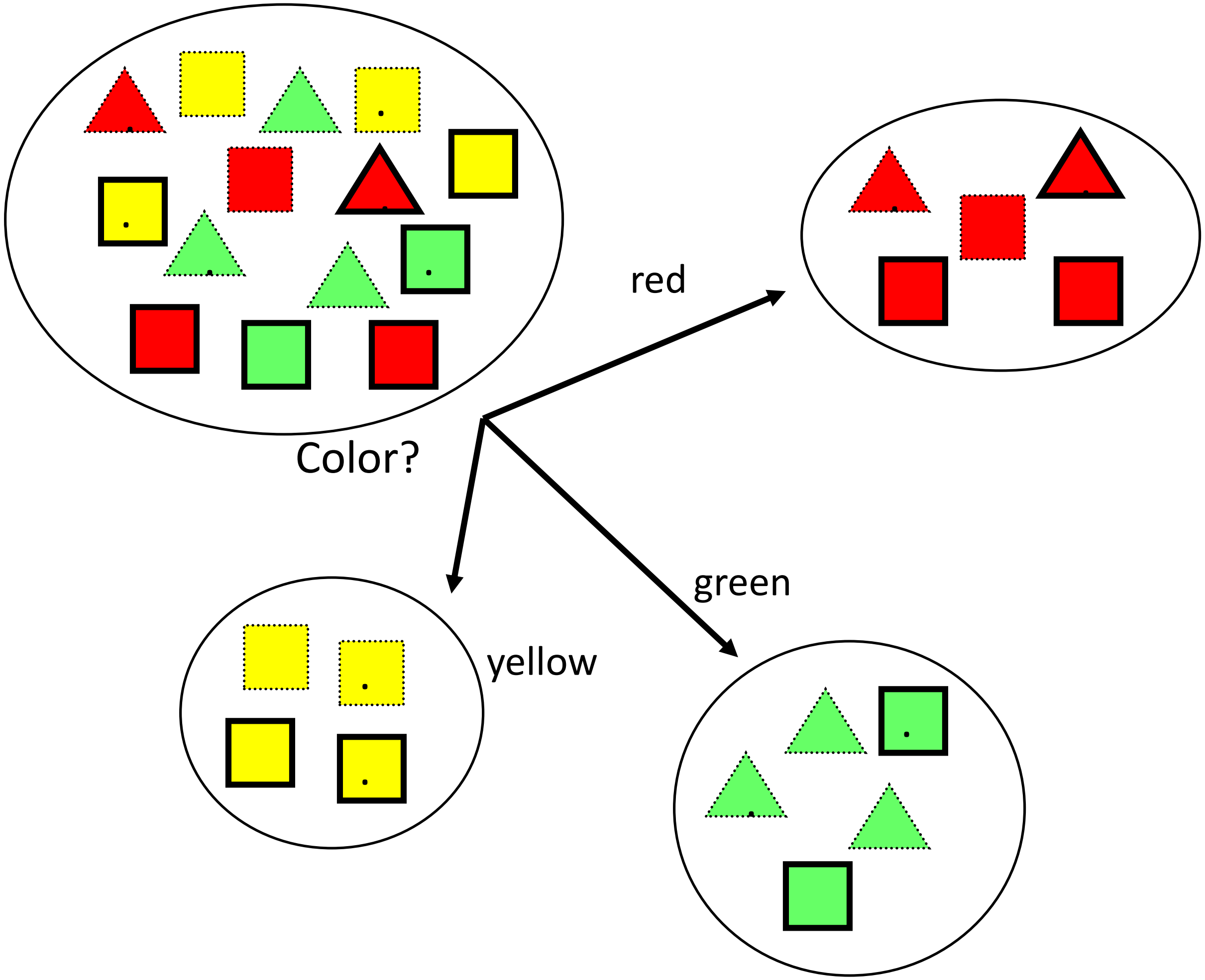
$I(green) = 0.971$ bits

$I(yellow) = 0.0$ bits

$$\boxed{Gain(\text{Color}) = I - I_{res}(\text{Color}) = 0.940 - 0.694 = 0.246\ bits}$$

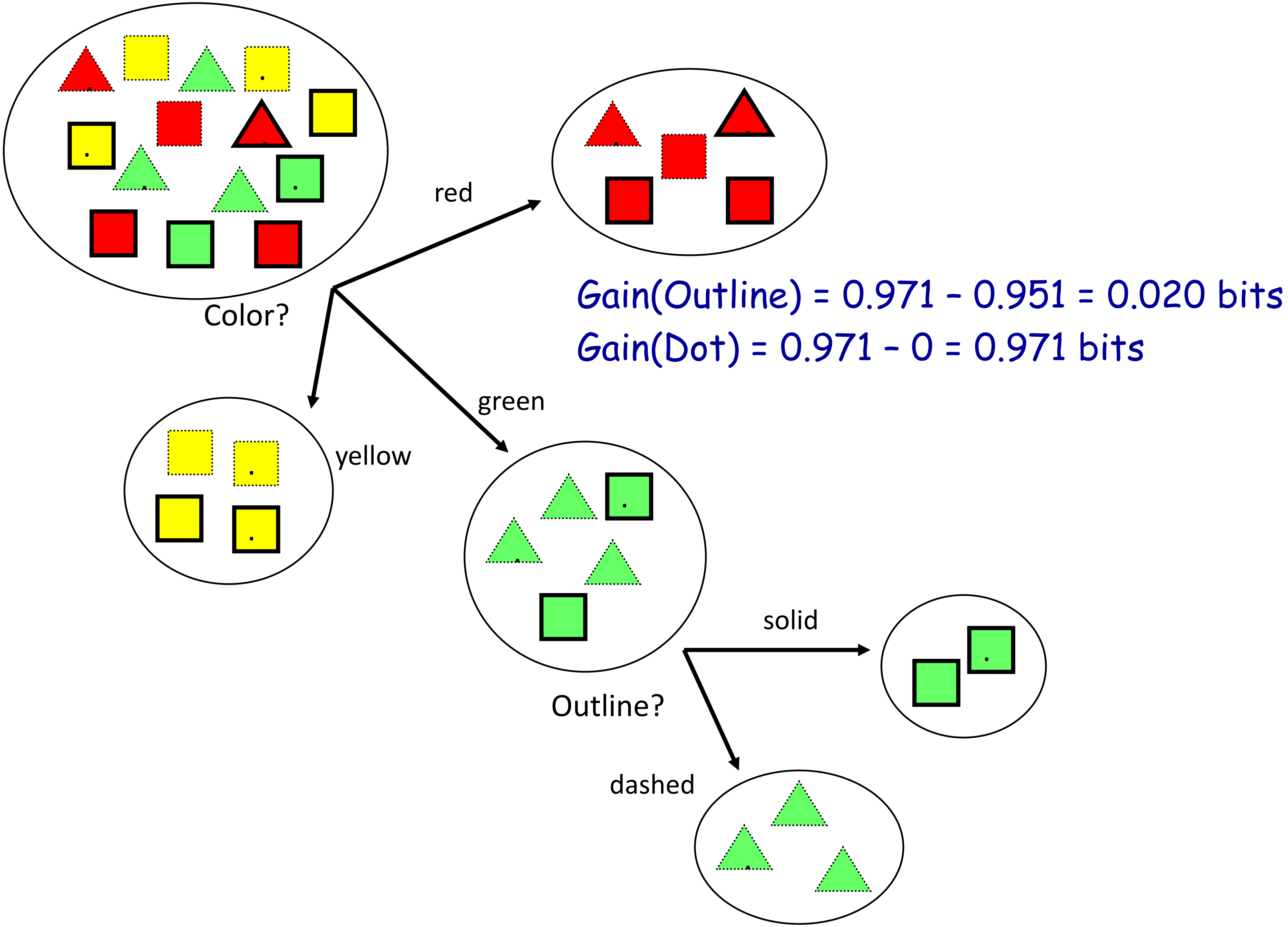
Information Gain of The Attribute

- Attributes
  - Gain(Color) = 0.246
  - Gain(Outline) = 0.151
  - Gain(Dot) = 0.048
- Heuristics: attribute with the highest gain is chosen
- This heuristics is local (local minimization of impurity)



Gain(Outline) = 0.971 – 0 = 0.971 bits

Gain(Dot) = 0.971 – 0.951 = 0.020 bits



Color?

red

yellow

green

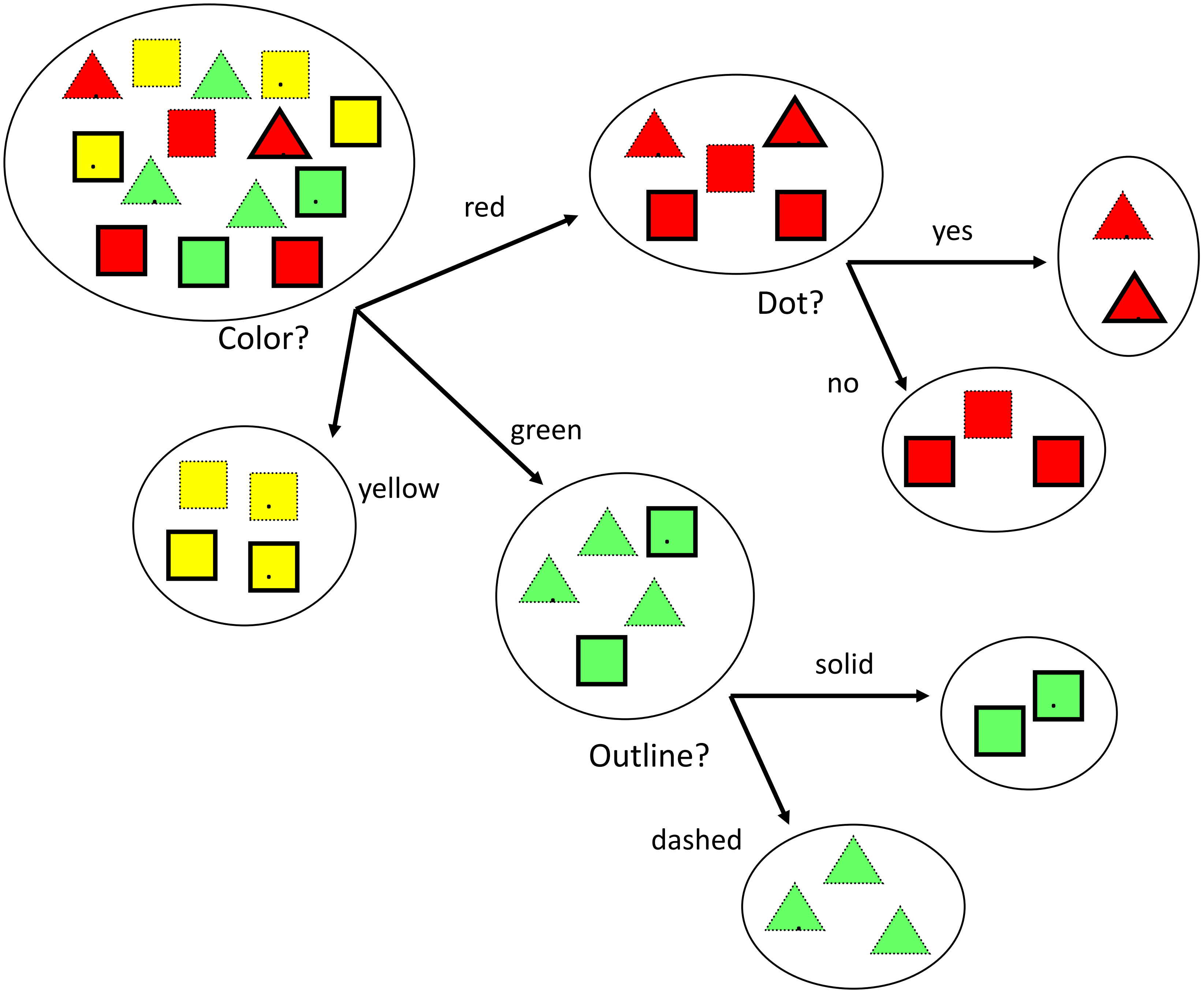
Outline?

solid

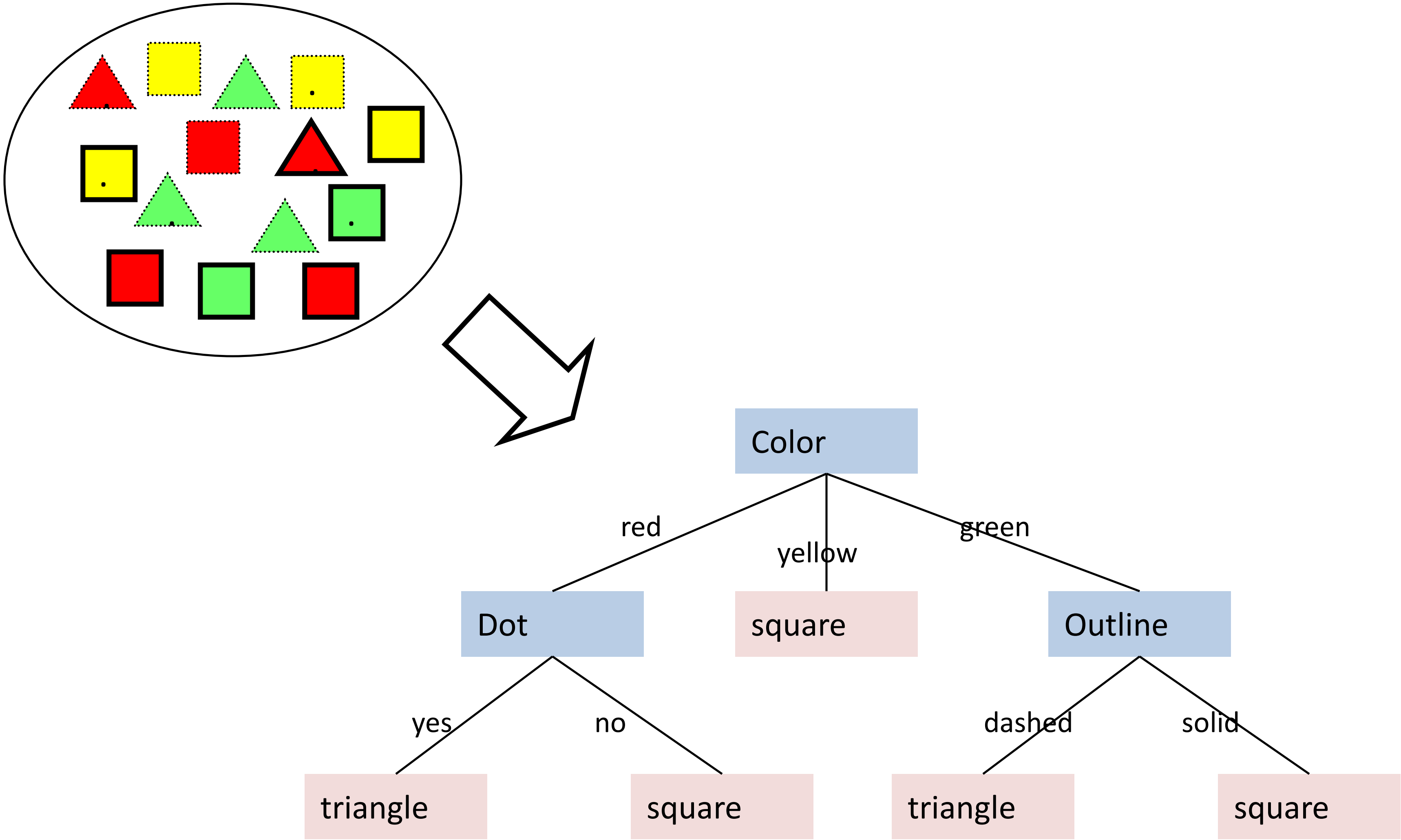
dashed

Gain(Outline) = 0.971 – 0.951 = 0.020 bits

Gain(Dot) = 0.971 – 0 = 0.971 bits


red
Color?
yellow
green
Dot?
yes
no
Outline?
solid
dashed

Decision tree



Color

red

yellow

green

Dot

square

Outline

yes

no

dashed

solid

triangle

square

triangle

square

Limitation

- Attributes with a lot of values will split S into a very high number of subsets, and if these are small, they will tend to be pure anyway
- → *Ires* favors attributes with many values
- One way to rectify this is through a corrected measure of **information gain ratio**.

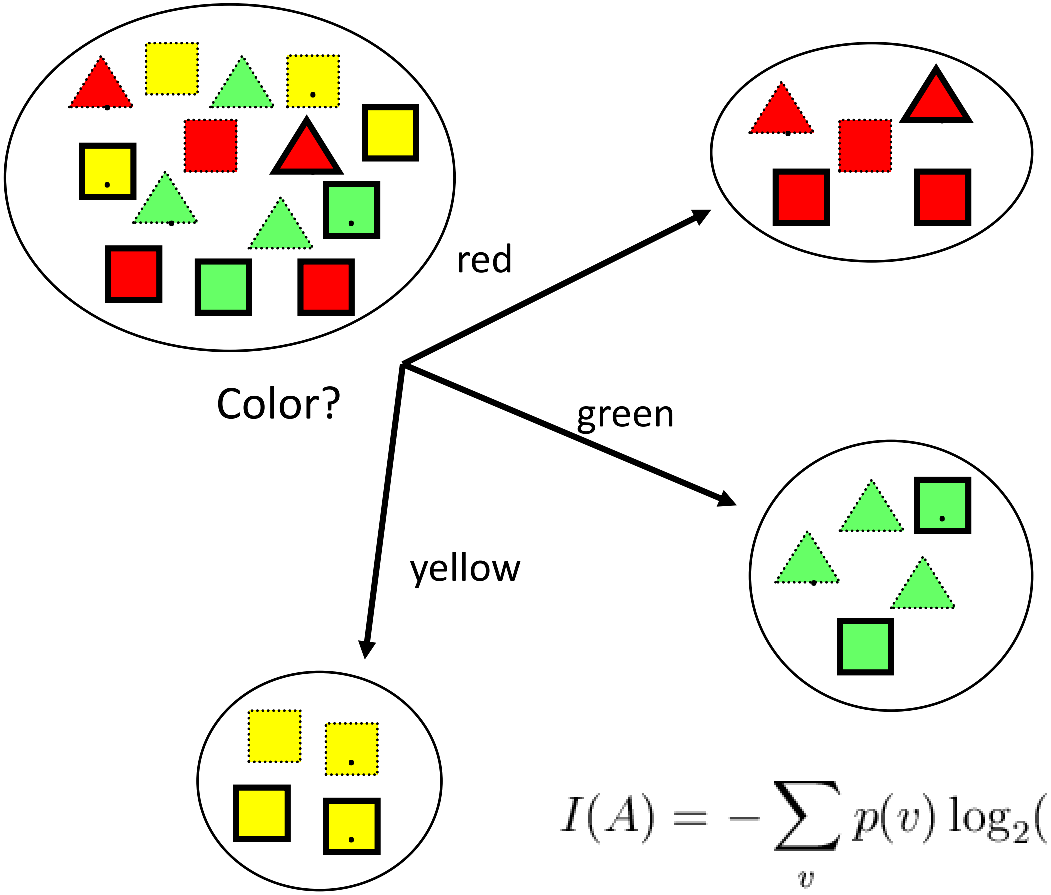
Information Gain Ratio

- I(A) is amount of information needed to determine the value of an attribute A

$$I(A) = -\sum_v p(v)\log_2(p(v))$$

- Information gain ratio

$$GainRatio(A) = \frac{Gain(A)}{I(A)} = \frac{I - I_{res}(A)}{I(A)}$$



$$I(A) = -\sum_{v} p(v)\log_2(p(v))$$

$$I(\text{Color}) = -\frac{5}{14}\log_2\frac{5}{14} - \frac{5}{14}\log_2\frac{5}{14} - \frac{4}{14}\log_2\frac{4}{14} = 1.58 \text{ bits}$$

$$\mathit{GainRatio}(\text{Color}) = \frac{\mathit{Gain}(\text{Color})}{I(\text{Color})} = \frac{0.940 - 0.694}{1.58} = 0.156$$

Gini Index

- Another sensible measure of impurity (i and j are classes)

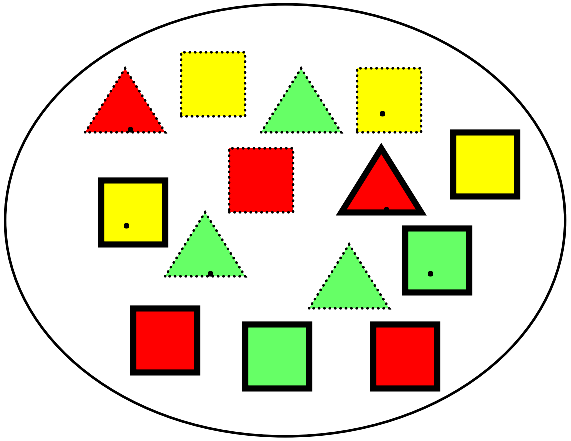
$$Gini = \sum_{i \neq j} p(i)p(j)$$

- After applying attribute A, the resulting Gini index is

$$Gini(A) = \sum_{v} p(v) \sum_{i \neq j} p(i|v)p(j|v)$$

- Gini can be interpreted as expected **error rate**

Gini Index



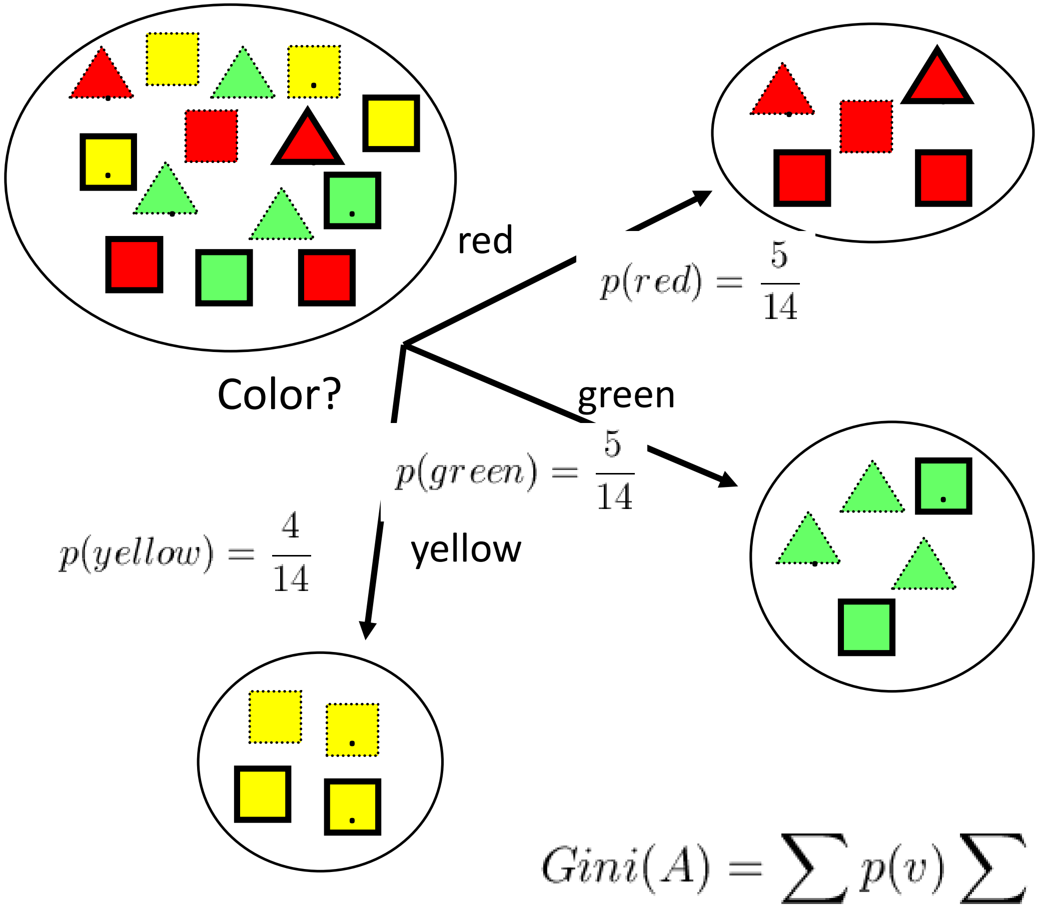
$$p(\square) = \frac{9}{14}$$

$$p(\triangle) = \frac{5}{14}$$

$$Gini = \sum_{i \neq j} p(i)p(j)$$

$$Gini = \frac{9}{14} \times \frac{5}{14} = 0.230$$

Gini Index



red

$p(red) = \frac{5}{14}$

Color?

green

$p(green) = \frac{5}{14}$

$p(yellow) = \frac{4}{14}$

yellow

$$Gini(A) = \sum p(v) \sum p(i|v)p(j|v)$$

$$Gini(\text{Color}) = \frac{5}{14} \times (\frac{3}{5} \times \frac{2}{5}) + \frac{5}{14} \times (\frac{2}{5} \times \frac{3}{5}) + \frac{4}{14} \times (\frac{4}{4} \times \frac{0}{4}) = 0.171$$

Gain of Gini index

$$Gini = \frac{9}{14} \times \frac{5}{14} = 0.230$$

$$Gini(\text{Color}) = \frac{5}{14} \times (\frac{3}{5} \times \frac{2}{5}) + \frac{5}{14} \times (\frac{2}{5} \times \frac{3}{5}) + \frac{4}{14} \times (\frac{4}{4} \times \frac{0}{4}) = 0.171$$

$$\boxed{GiniGain(\text{Color}) = 0.230 - 0.171 = 0.058}$$

Three measures of impurity

A	Gain(A)	GainRatio(A)	GiniGain(A)
Color	0.247	0.156	0.058
Outline	0.152	0.152	0.046
Dot	0.048	0.049	0.015

Decision trees

- **Training**
  - Entropy-based Methods
- **Strengths**
  - Easy to Understand
  - Easy to Implement
- **Weaknesses**
  - Straightforward for categorical data… data discretization needed
  - Tendency to overtraining

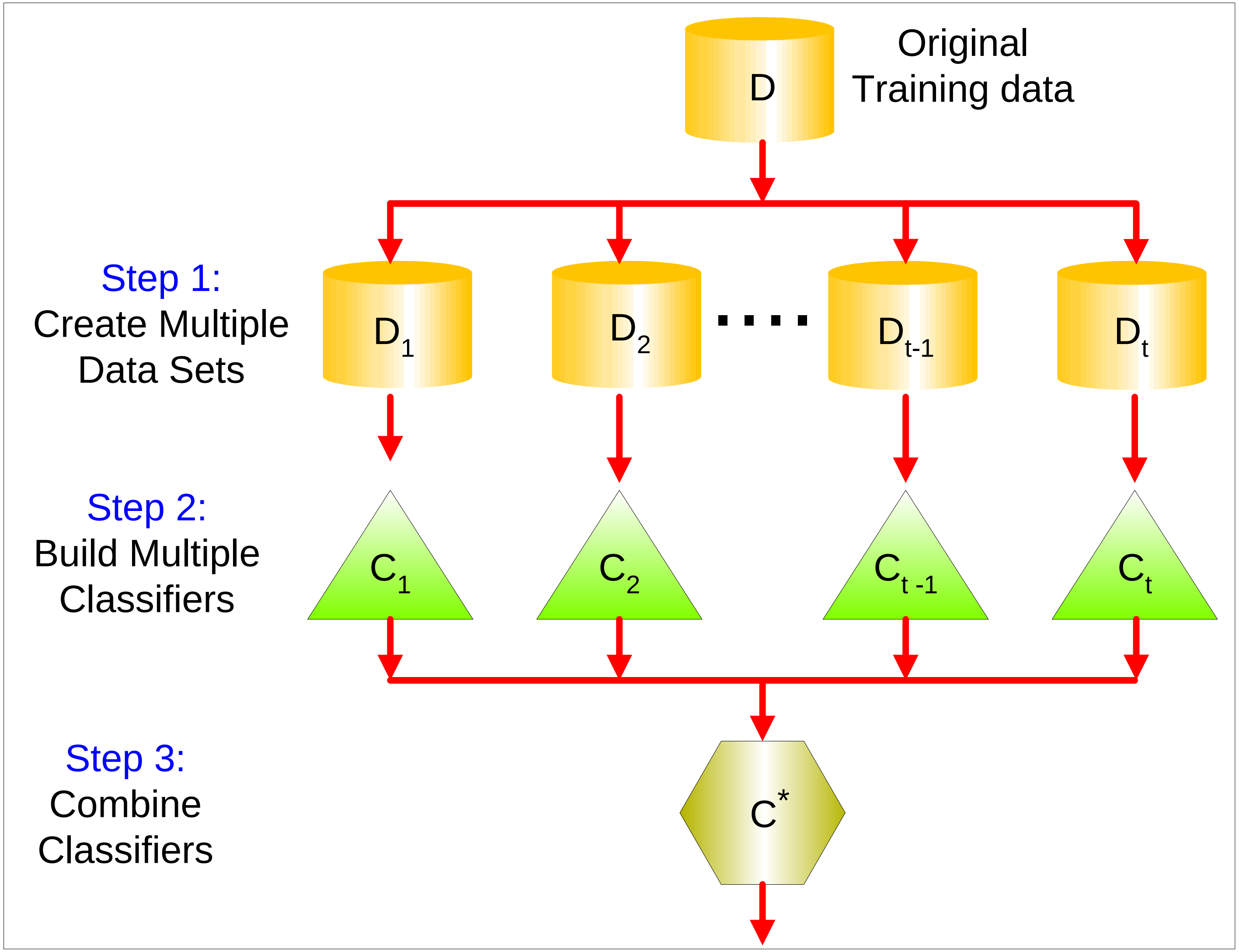
Ensemble classifiers

- *No Free Lunch Theorem*: There is no algorithm that is always the most accurate
- Generate a group of **base learners** and combine them to achieve higher accuracy (*In WEKA, these are called* ***meta-learners***).
- Examples: Human ensembles are demonstrably better
  - How many jelly beans in the jar?: Individual estimates vs. group average.
  - Who Wants to be a Millionaire: Audience vote.

Ensemble classifiers

- Each algorithm makes assumptions which might be or not be valid for the problem at hand or not.
- Each algorithm may have a low accuracy (at least higher than random guessing!)
- Different learners may use different
  - Algorithms
  - Parameters
  - Representations
  - Training sets
  - Subproblems
- Combine the decision of the individual learners by **majority voting**, or by **weighted voting**

General Idea



Step 1: Create Multiple Data Sets

Step 2: Build Multiple Classifiers

Step 3: Combine Classifiers

Why does it work?

- **Majority voting:** the final decision corresponds to the decision of the majority of the base learners
- Suppose we have 5 completely independent classifiers…
  - If accuracy is 70% for each
    - $(.7^5)+5(.7^4)(.3)+10(.7^3)(.3^2)$
    - **83.7% majority vote accuracy**

Note - Binomial Distribution: The probability of observing *k* heads in a sample of *n* independent coin tosses, where in each toss the probability of heads is *p*, is

$$P(X=k)=\binom{n}{k}p^k(1-p)^{n-k}$$

- Suppose there are 101 of such classifiers: **99.9% accuracy**

Main challenge: promote diversity

- The main challenge is not to obtain highly accurate base models, but rather to obtain base models which make **different kinds of errors**.
- For example, if ensembles are used for classification, high accuracies can be accomplished if different base models **misclassify different training examples**, even if the base classifier accuracy is low.
- Independence between two base classifiers can be assessed by measuring the degree of **overlap in training examples they misclassify** ($|A \cap B|/|A \cup B|$)—more overlap means less independence between two models.

Ensemble methods

- Some popular ensemble approaches:
  - Bootstrap aggregating (bagging)
  - Boosting
    - Example: Adaboost
  - Random Forests

Bagging

- Create different training sets by “***bootstrap aggregation***”, i.e., repeatedly randomly resampling the original training data (Brieman, 1996).

- Base learners are trained on the bootstrapped training sets

- Decreases error especially of ***unstable learners*** (e.g. decision trees) whose output can change dramatically when the training data is slightly changed.

Bagging

- Given a standard training set *D* of size *n*
- For i = 1 .. M
  - Draw a sample of size $n^*<n$ from *D* uniformly and with replacement
  - Learn classifier C_i
- Final classifier is a vote of $C_1 .. C_M$
- Increases classifier stability/reduces variance

Boosting

- Rationale: build a **strong learner** by combining several **weak learners**

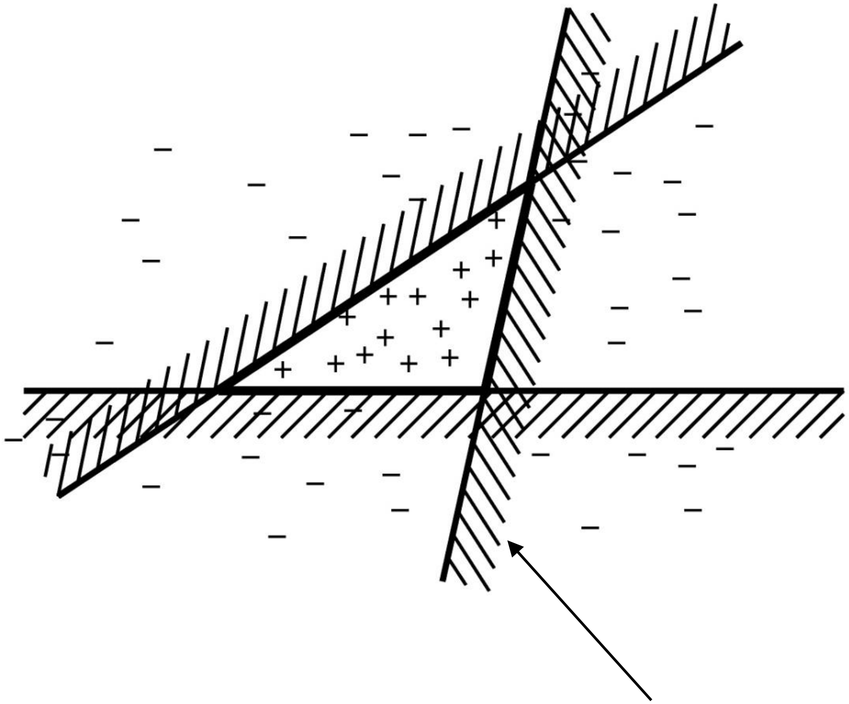
- Strong Learner → Objective of machine learning
  - Take labeled data for training
  - Produce a classifier which can be *arbitrarily accurate*
  - Difficult to construct

- Weak Learner
  - Take labeled data for training
  - Produce a classifier which is just more accurate than random guessing
  - Easy to construct

Boosting

- Most popular algorithm: AdaBoost
- Instead of resampling (as in bagging), re-weigh examples!
- Training samples are given **weights**. At each iteration, a new weak learner is trained, and the training samples are reweighted to focus the system on examples that the most recently learned classifier got wrong (i.e. training samples that were misclassified are given a higher waight than the ones correctly classified)
- Final classification based on **weighted voting** of weak classifiers
- When used with DT algorithms as base learners, considerably decreases overfitting. On the other hand, it is sensitive to noise and outliers

Example



This line is one simple classifier saying that everything to the left + and everything to the right is -

AdaBoost

```
$D_1$= initial dataset with equal weights (sum of weights is 1)

FOR i=1 to k DO
    Learn new classifier Ci;
    Update example weights;
        (Increase weight of misclassified examples)
        (Decrease weight of correctly classified )
    Compute $\alpha_i$ (classifier's importance);
    Create new training set Di+1 (using weighted sampling)
        (Probability of sampling is proportional to weight)
END FOR
```

Construct Ensemble combining Ci classifiers weighted by $\alpha\iota$ (i=1,k):

$$C_{ensemble}(x) = \sum_{i=1}^{k} \alpha_i C_i(x)$$

AdaBoost

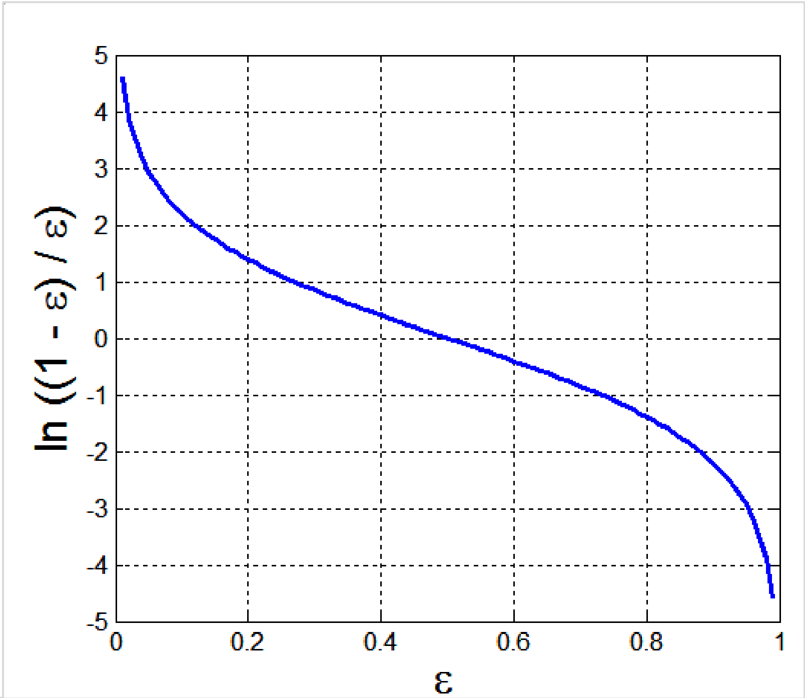
- Base classifiers: C$_i$ (i=1 to k)

- Error rate i (weights add up to 1):

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

- Importance of a classifier i:

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$



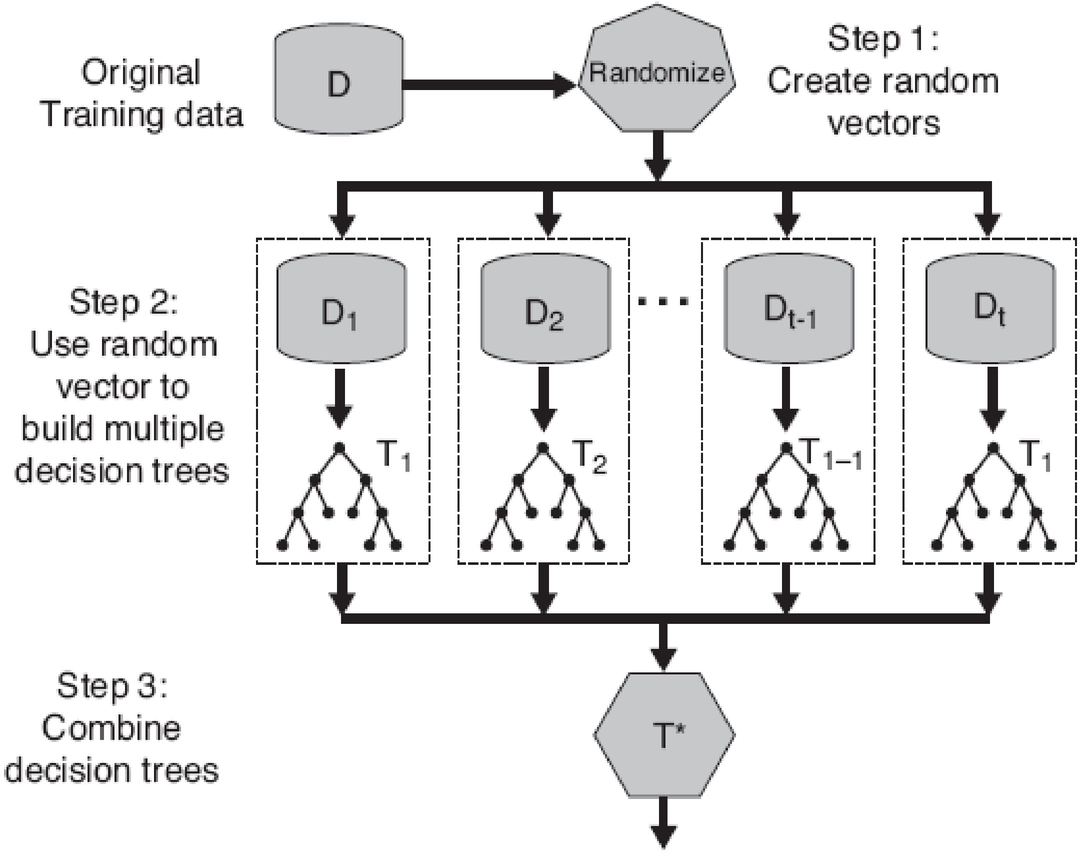
Random Forest

- Ensemble method specifically designed for decision tree classifiers
- Random Forests grows many trees
  - Ensemble of unpruned decision trees
  - Each base classifier classifies a “new” vector of attributes from the original data
  - Final result on classifying a new instance: voting over all the trees in the forest

Random Forest

- Ensemble method specifically designed for DT classifiers
- Random Forests grows a forest of many trees:
  - Ensemble of unpruned decision trees
  - Final result on classifying a new instance: majority voting over all the trees in the forest
- Introduce two sources of randomness:
  - **Bagging method**: each base tree is grown using a bootstrap sample of the original training data
  - **Random vector method**: at each node of a tree, the best split is chosen not from all the attributes, but from a random sample of *m* attributes

Random Forest



Original Training data

Step 1: Create random vectors

Step 2: Use random vector to build multiple decision trees

Step 3: Combine decision trees

Figure 5.40. Random forests.

Methods for Growing the Trees

- Fix a $m < M$ (number of attributes).
- At each node
  - Method 1:
    - select m attributes randomly.
    - choose the attribute with the largest information gain to split.
  - Method 2:
    - Compute the information gain of all M attributes.
    - Select the top m attributes by information gain.
    - Randomly select one of the m attributes as the splitting node.
  - … many other variants