

A horizontal bar with a teal segment on the left and an orange segment on the right.

# Decision Tree

STAT5241 Section 2

Statistical Machine Learning

Xiaofei Shi



# Tasks

Input → Regressor → Predict real number

Input → Classifier → Predict category

Input → Density Estimator → Probability



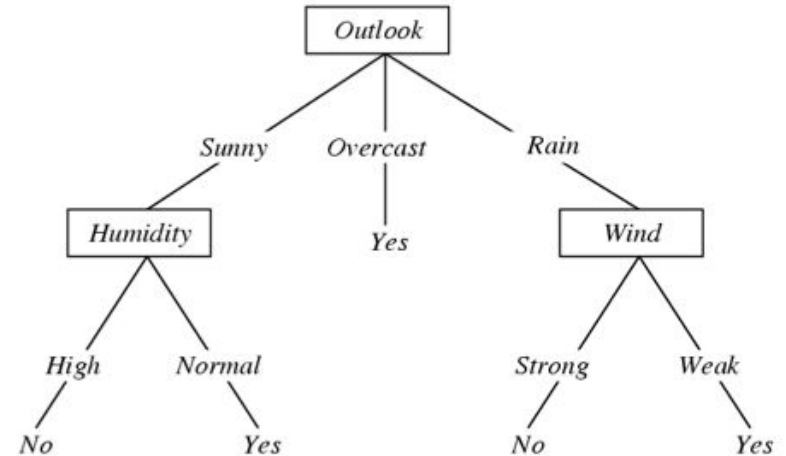


# Types of classifiers

- Discriminative classifiers:
  - Directly estimate a decision rule/boundary
  - e.g. decision tree, SVM
- Instance based classifiers:
  - Use observation directly
  - e.g. K nearest neighborhood
- Generative classifiers:
  - Build a generative statistical model
  - e.g. Bayesian Network

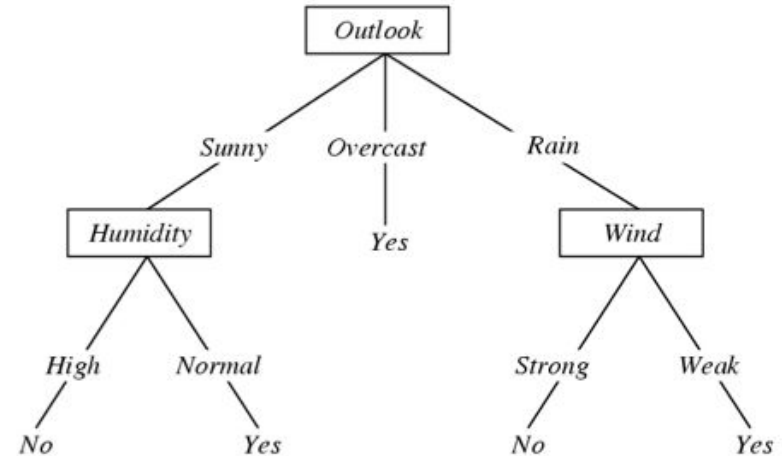
# How to make a decision...

- A decision tree for whether to play tennis
- Given a decision tree, how do we assign a label to a test point



# How to make a decision...

- A decision tree for whether to play tennis
- Each internal node: test one feature
- Each branch from a node: choose one value for the feature considered
- Each leaf node: prediction for the label





# Prediction

If given a decision tree:

- What function does a decision tree represent
- Given a decision tree, how do we assign label to a test point

Now the real problem is:

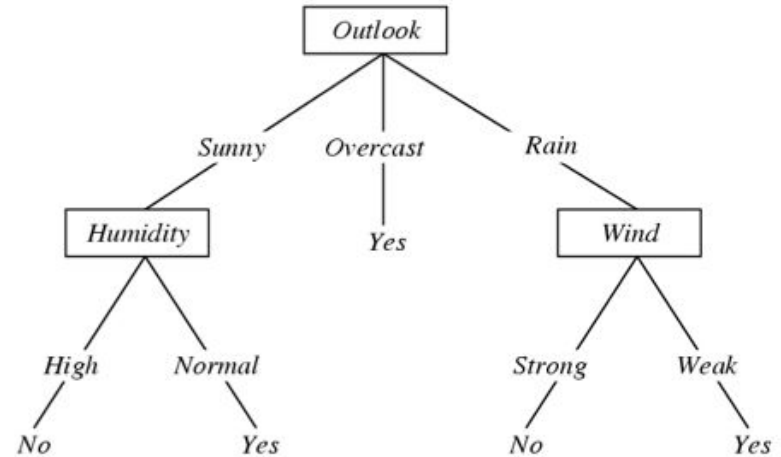
- How do we learn a decision tree from training data?

# How to learn and build a decision tree

Top-down induction: (ID3)

Main loop:

1.  $X \leftarrow$  the “best” decision feature for next *node*
2. Assign  $X$  as decision feature for *node*
3. For each value of  $X$ , create new descendant of *node* (**Discrete features**)
4. Sort training examples to leaf nodes
5. If training examples perfectly classified, Then STOP, Else iterate over new leaf nodes



After all features exhausted, assign majority label to each leaf node.

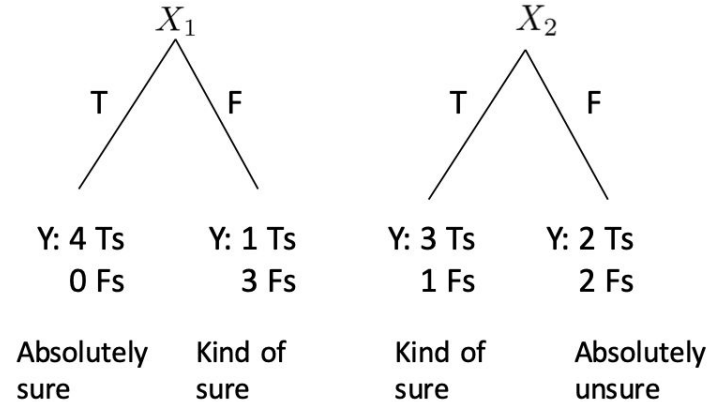
Which feature is considered as “best”

$X_1$	$X_2$	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F



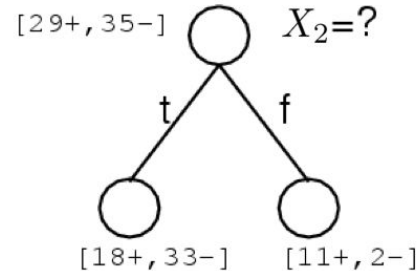
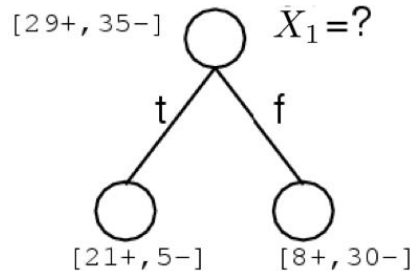
# Which feature is considered as “best”

$X_1$	$X_2$	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F



Good split if we are more certain about classification after split!

## Which feature is considered as “best”

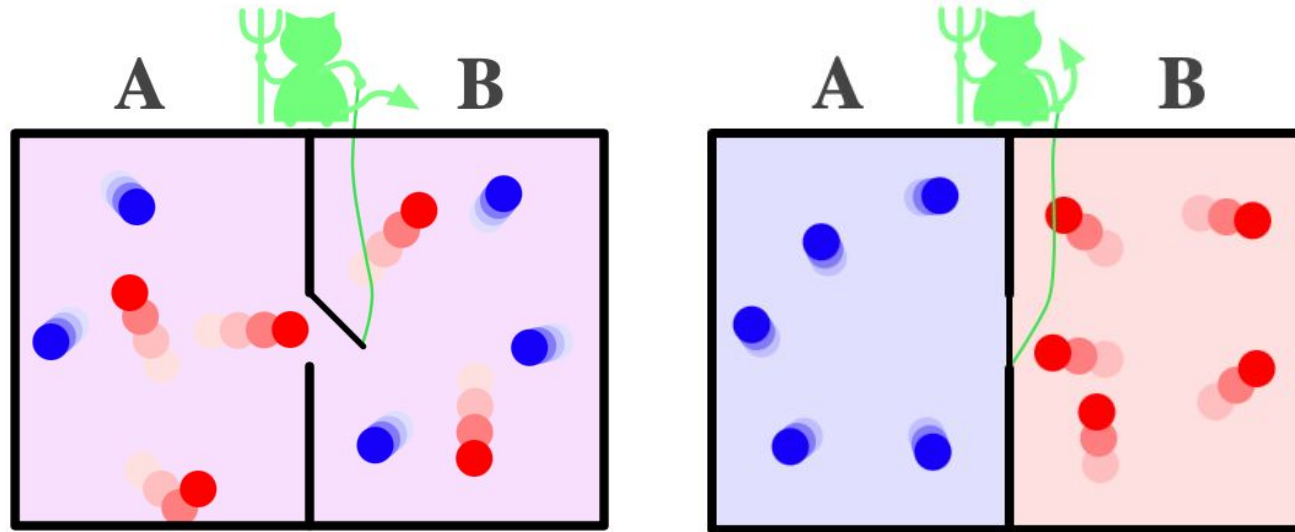


Pick the attribute/feature which yields maximum information gain:

$$\arg \max_i I(Y, X_i) = \arg \max_i [H(Y) - H(Y|X_i)]$$

$H(Y)$  – entropy of  $Y$      $H(Y|X_i)$  – conditional entropy of  $Y$

# Entropy: the level of uncertainty



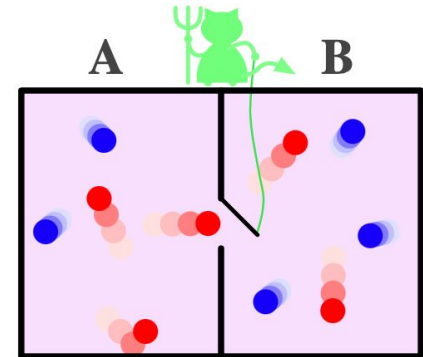
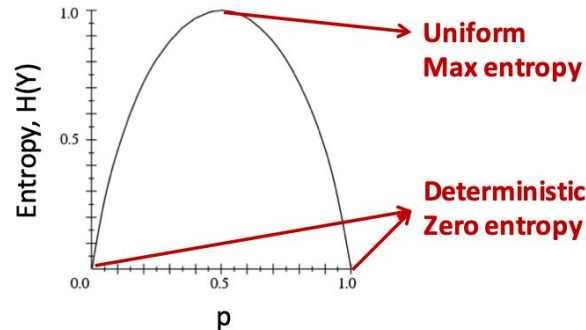
# Entropy: the level of uncertainty

- Entropy of a random variable  $Y$

$$H(Y) = - \sum_y P(Y = y) \log_2 P(Y = y)$$

***More uncertainty,  
more entropy!***

$Y \sim \text{Bernoulli}(p)$



**Information Theory interpretation:**  $H(Y)$  is the expected number of bits needed to encode a randomly drawn value of  $Y$  (under most efficient code)



# Information gain

- Advantage of attribute = decrease in uncertainty

- Entropy of Y before split

$$H(Y) = - \sum_y P(Y = y) \log_2 P(Y = y)$$

- Entropy of Y after splitting based on  $X_i$

- Weight by probability of following each branch

$$\begin{aligned} H(Y | X_i) &= \sum_x P(X_i = x) H(Y | X_i = x) \\ &= - \sum_x P(X_i = x) \sum_y P(Y = y | X_i = x) \log_2 P(Y = y | X_i = x) \end{aligned}$$

- Information gain is difference

$$I(Y, X_i) = H(Y) - H(Y | X_i)$$

**Max Information gain = min conditional entropy**





# Which feature is considered as “best”

Pick the attribute/feature which yields maximum information gain:

$$\begin{aligned}\arg \max_i I(Y, X_i) &= \arg \max_i [H(Y) - H(Y|X_i)] \\ &= \arg \min_i H(Y|X_i)\end{aligned}$$

Entropy of Y

$$H(Y) = - \sum_y P(Y = y) \log_2 P(Y = y)$$

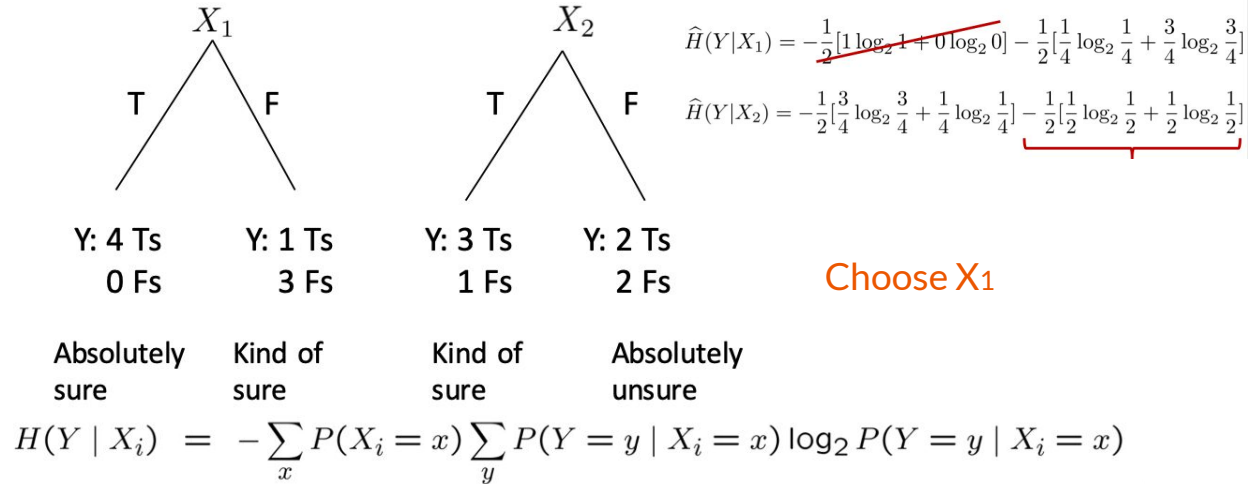
Conditional entropy of Y

$$H(Y | X_i) = \sum_x P(X_i = x) H(Y | X_i = x)$$

Feature which yields maximum reduction in entropy (uncertainty)  
provides maximum information about Y

# Which feature is considered as “best”

$X_1$	$X_2$	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F

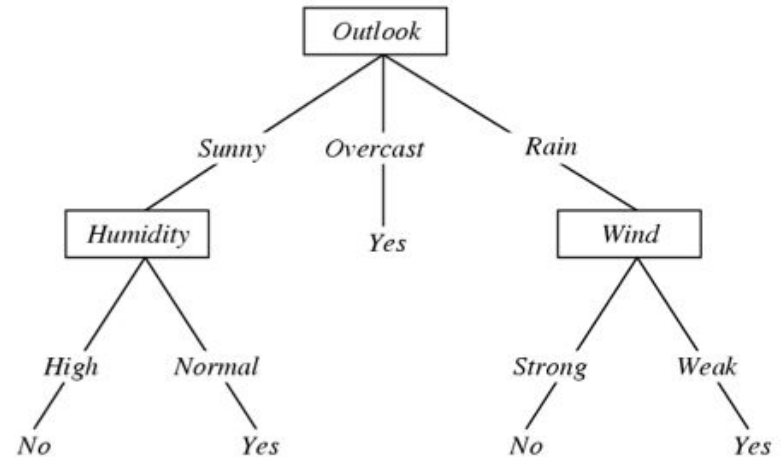


# How to learn and build a decision tree

Top-down induction: (ID3)

Main loop:

1.  $X \leftarrow$  the “best” decision feature for next *node*
2. Assign  $X$  as decision feature for *node*
3. For each value of  $X$ , create new descendant of *node* (**Discrete features**)
4. Sort training examples to leaf nodes
5. If training examples perfectly classified, Then STOP, Else iterate over new leaf nodes

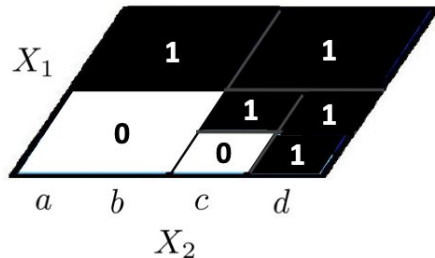
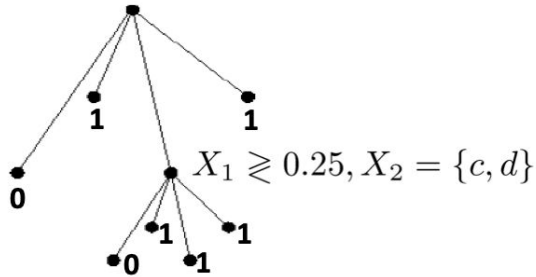


After all features exhausted, assign majority label to each leaf node.



## More generally

$$X_1 \geq 0.5, X_2 = \{a, b\} \text{ or } \{c, d\}$$

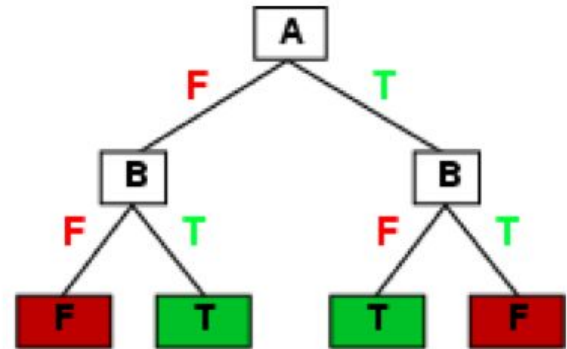


- Features can be discrete, continuous or categorical
- Each internal node: test some set of features  $\{X_i\}$
- Each branch from a node: selects a set of value for  $\{X_i\}$
- Each leaf node: prediction for  $Y$

## More generally

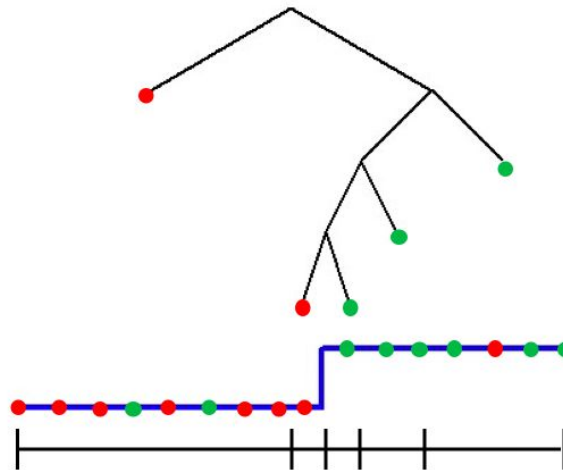
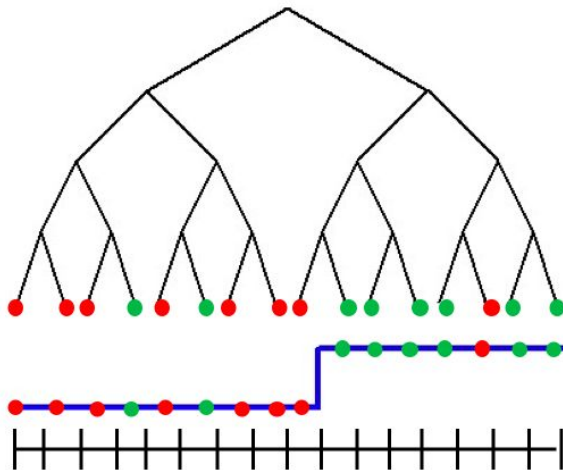
- Decision trees in general (without pruning) can express any function of the input features.
- There is a decision tree which perfectly classifies a training set with one path to leaf for each example - overfitting
- But it won't generalize well to new examples - prefer to find more compact decision trees

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F

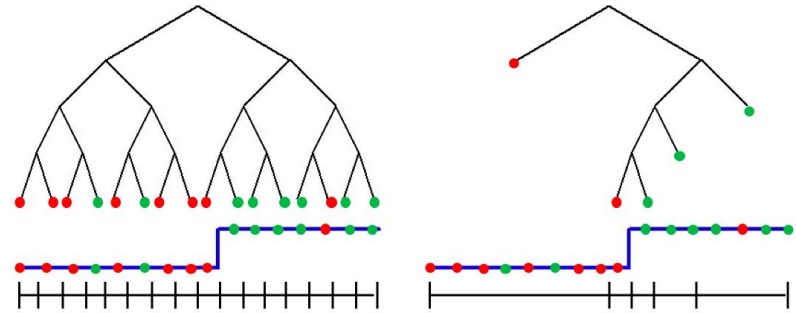


# Overfitting

One training example per leaf – overfits, need compact/pruned decision tree



# When to stop...



- Occam's razor: *the more assumptions you have to make, the more unlikely an explanation*
- People therefore look for simpler trees:
  - Pre-Pruning:
    - Fixed depth
    - Fixed number of leaves
  - Post-Pruning: Chi-square test of independence
    - Convert decision tree to a set of rules
    - Eliminate variable values in rules which are independent of label (using Chi-square test)
    - Simplify rule set by eliminating unnecessary rules
  - Complexity Penalized/MDL model selection

# Information Criteria

- Penalize complex models by introducing cost

$$\hat{f} = \arg \min_T \left\{ \underbrace{\frac{1}{n} \sum_{i=1}^n \text{loss}(\hat{f}_T(X_i), Y_i)}_{\text{log likelihood}} + \underbrace{\text{pen}(T)}_{\text{cost}} \right\}$$

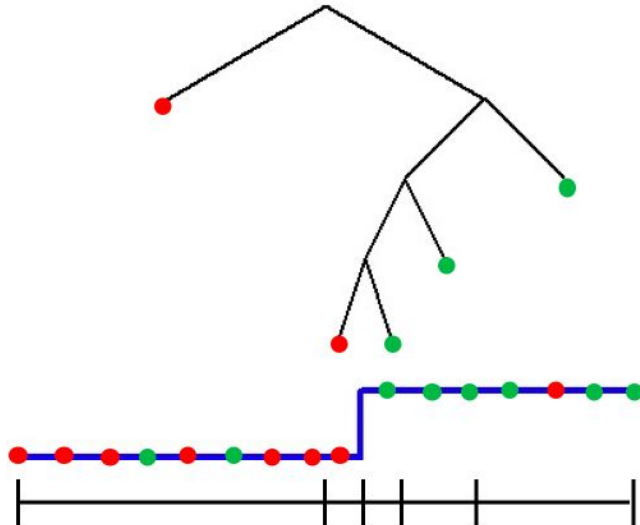
$$\begin{aligned} \text{loss}(\hat{f}_T(X_i), Y_i) &= (\hat{f}_T(X_i) - Y_i)^2 && \text{regression} \\ &= \mathbf{1}_{\hat{f}_T(X_i) \neq Y_i} && \text{classification} \end{aligned}$$

$\text{pen}(T) \propto |T|$       penalize trees with more leaves

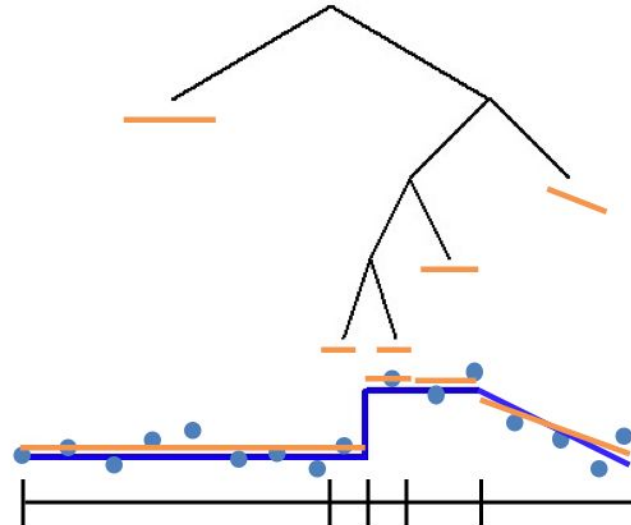
CART – optimization can be solved by dynamic programming

# How to assign values to each leaf

Classification – Majority vote



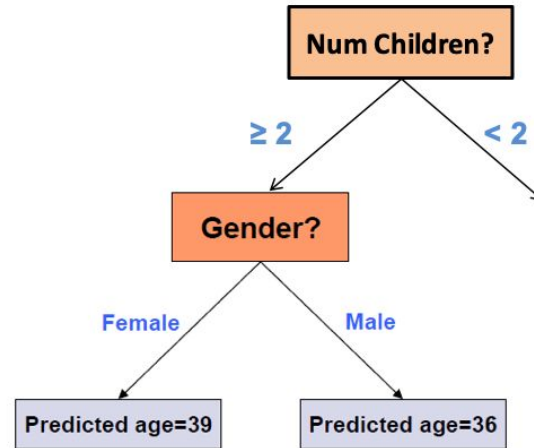
Regression – Constant/  
Linear/Poly fit



# Regression Tree

$X^{(1)}$       ....       $X^{(p)}$      $Y$

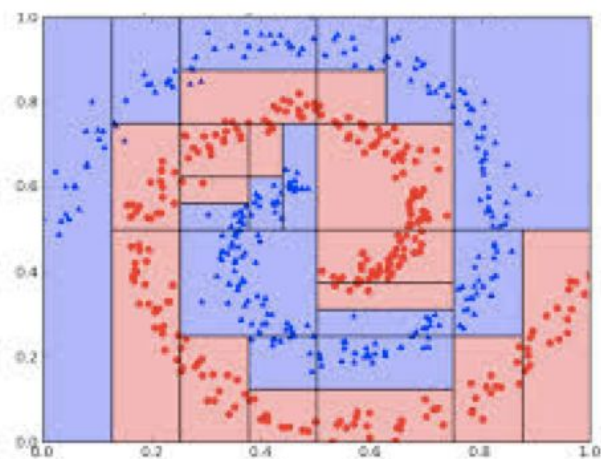
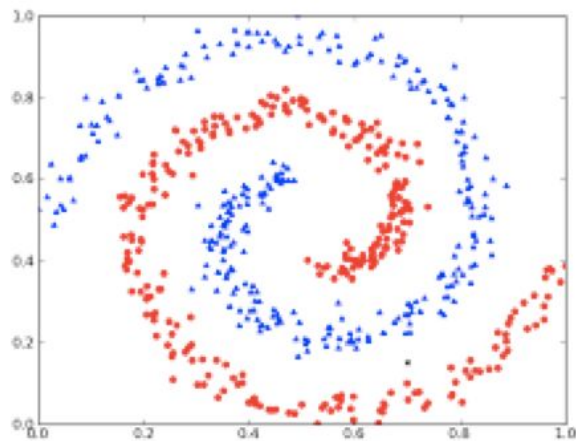
Gender	Rich?	Num. Children	# travel per yr.	Age
F	No	2	5	38
M	No	0	2	25
M	Yes	1	0	72
:	:	:	:	:



Average (fit a constant ) using  
training data at the leaves



## A 2D example







# Takeaways

- Decision trees are one of the most popular data mining tools
  - Interpretability
  - Easy to implement
  - Good performance in practice (for small dimensions!!!)
- Information gain to select attributes (ID3, C4.5, CART...)
- Can be used for classification, regression and density estimation too
- **Decision trees will overfit!!!** Must use tricks to find “simple trees”, e.g.,
  - Pre-Pruning: Fixed depth/Fixed number of leaves
  - Post-Pruning: Chi-square test of independence
  - Complexity Penalized/MDL model selection



# K Nearest Neighbors and Kernel Regression



# Loss function

Recall the loss function in previous classifiers:

- Logistic Regression: MLE of conditional (log)likelihood
- Decision Tree: maximum information gain
- What else?



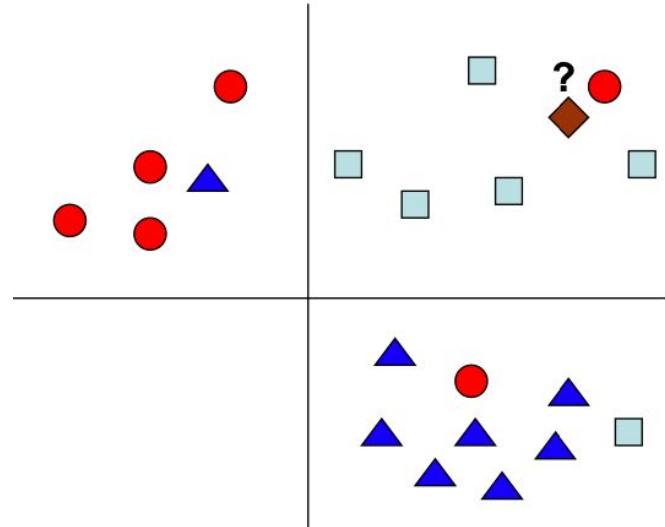
# Loss function

- The risk value we computed assumes that both errors (assigning instances of class 1 to class 0 and vice versa) are equally harmful.
- However, this is not always the case.
- Why?
- In general our goal is to minimize loss, often defined by a loss function:  $L_{0,1}(x)$  which is the penalty we pay when assigning instances of class 0 to class 1

$$E[L] = L_{0,1}p(y=0) \int_{L_1} p_0(x)dx + L_{1,0}p(y=1) \int_{L_0} p_1(x)dx$$

# K nearest neighbors (KNN)

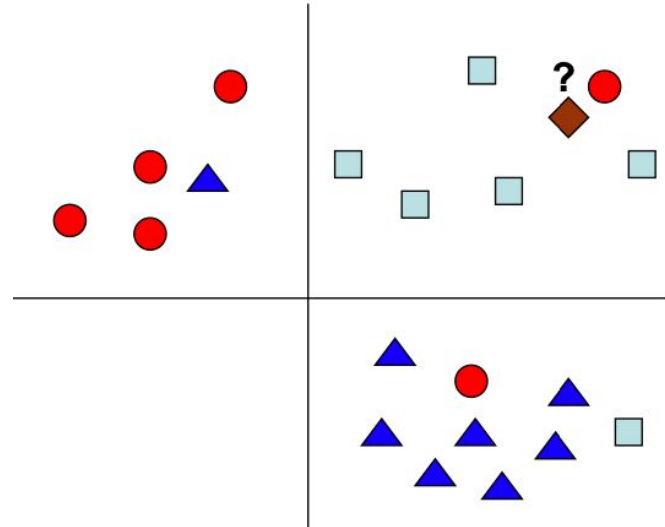
- A simple yet surprisingly efficient algorithm
- Requires the definition of a similarity measure or a distance function between sample points
- Select the class based on the majority vote among the K nearest sample points



# K nearest neighbors (KNN)

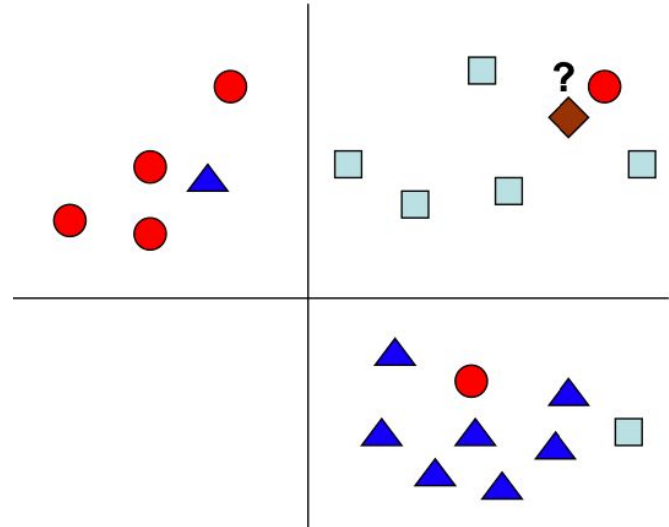
- A simple yet surprisingly efficient algorithm
- Requires the definition of a similarity measure or a distance function between sample points
- Select the class based on the majority vote among the K nearest sample points

What is the best value of K?



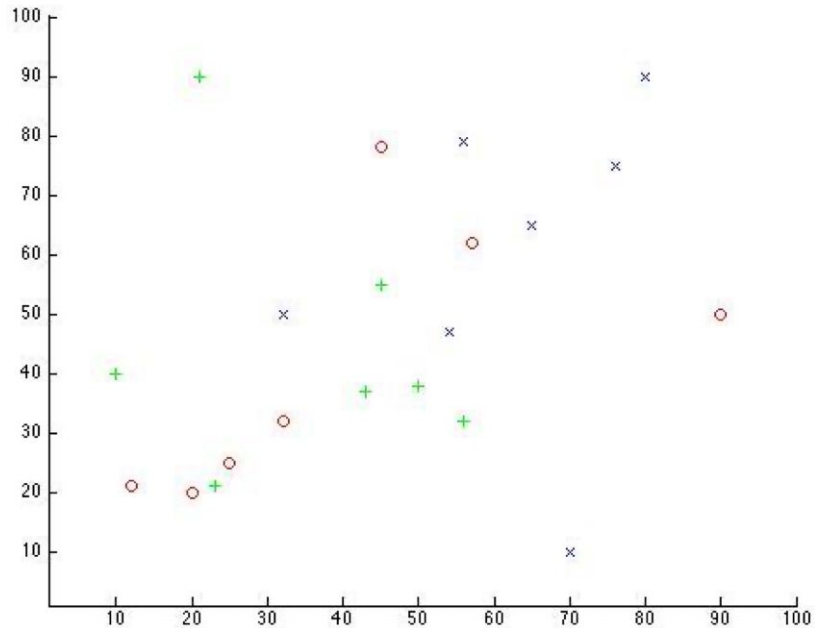
# K nearest neighbors (KNN)

- Choice of  $K$  influences the “smoothness” of the resulting classifier
- In this sense, the KNN classifier is similar to a kernel methods
- However, the smoothness of the classifier should be determined by the actual distribution of the data, i.e. the density function  $p(x)$  of the data, not any predefined parameter.



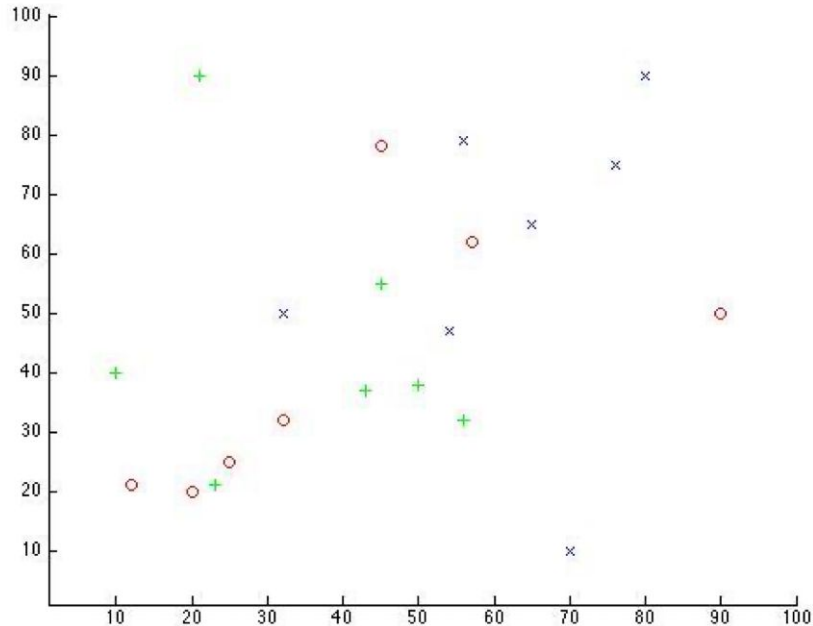


# The effect of increasing K





# The effect of increasing K

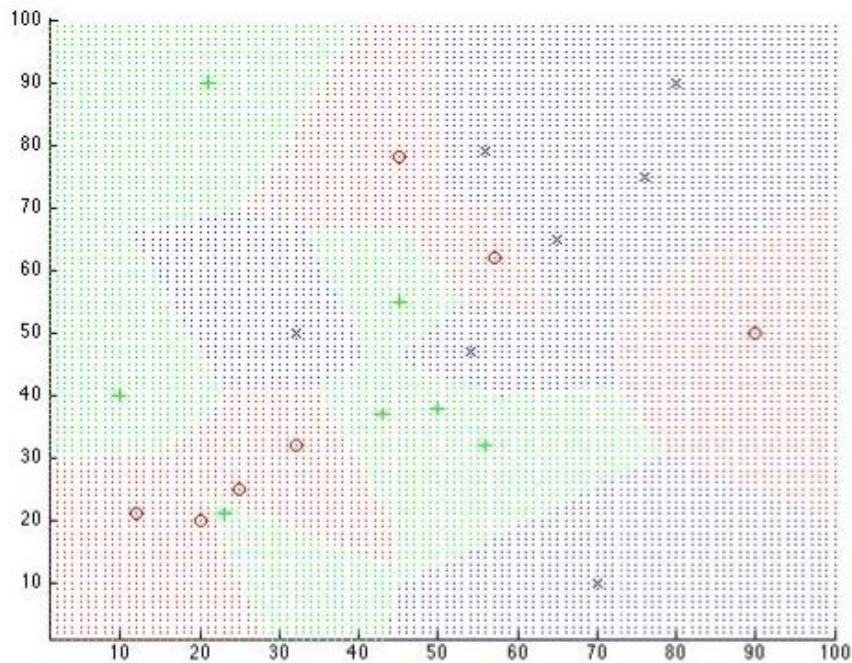


We will be using Euclidian distance to determine what are the k nearest neighbors:

$$d(x, x') = \sqrt{\sum_i (x_i - x'_i)^2}$$



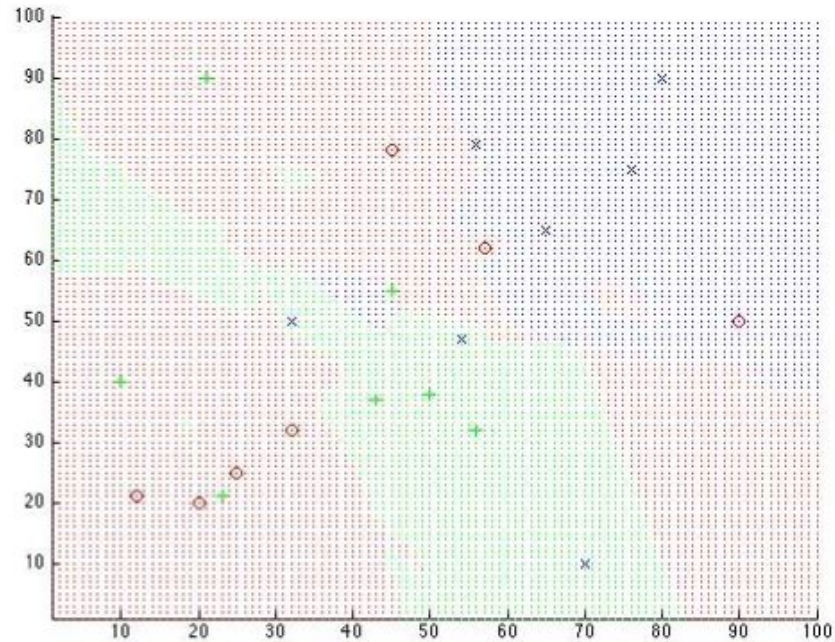
# The effect of increasing K



$K = 1$



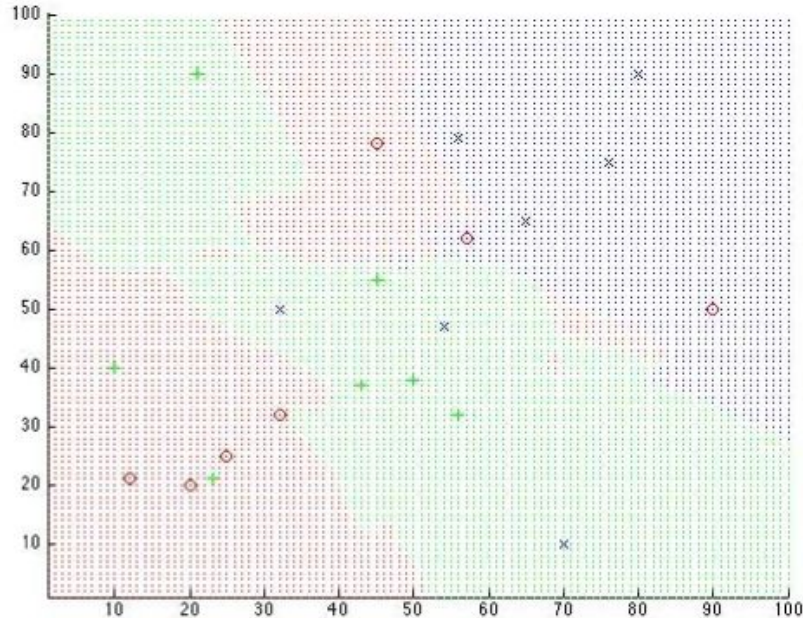
# The effect of increasing K



K = 3



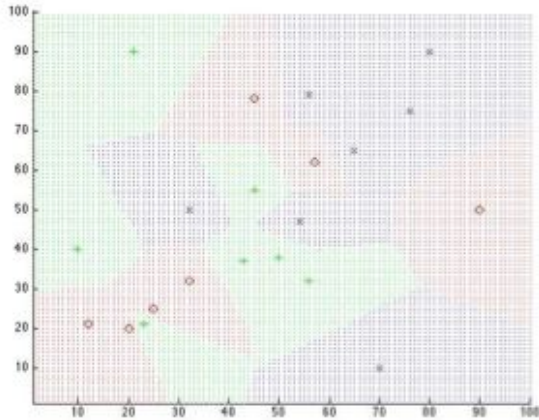
# The effect of increasing K



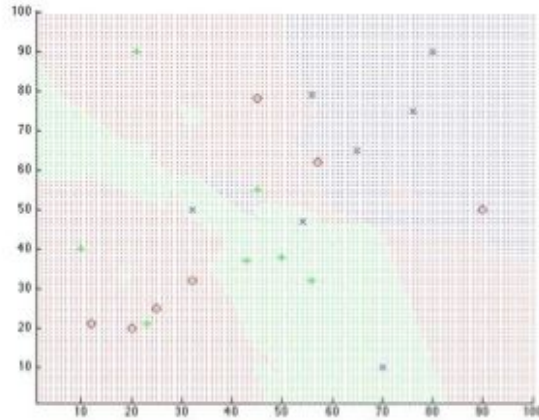
K = 5

# Comparison of different values of K

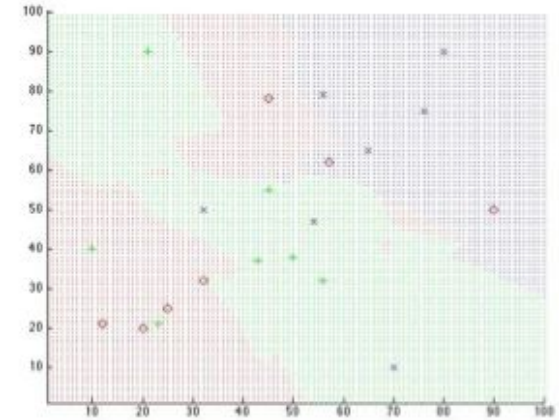
K = 1



K = 3



K = 5





# A probabilistic interpretation of KNN

- The decision rule of KNN can be viewed using a probabilistic interpretation
- What KNN is trying to do is approximate the Bayes decision rule on a subset of the data
- To do that we need to compute certain properties including the conditional probability of the data given the class ( $p(x|y)$ ), the prior probability of each class ( $p(y)$ ) and the marginal probability of the data ( $p(x)$ )
- These properties would be computed for some small region around our sample and the size of that region will be **dependent on the distribution of the test samples\***



- Let  $V$  be the volume of the  $m$  dimensional ball around  $z$  containing the  $k$  nearest neighbors for  $z$  (where  $m$  is the number of features).
- Then we can write

$$p(x)V = P = \frac{K}{N} \quad p(x) = \frac{K}{NV} \quad p(x | y = 1) = \frac{K_1}{N_1V} \quad p(y = 1) = \frac{N_1}{N}$$

- Using Bayes rule we get:

Choose the class with the highest probability

$$p(y = 1 | z) = \frac{p(z | y = 1)p(y = 1)}{p(z)} = \frac{K_1}{K}$$

$z$  – new data point to classify

$V$  - selected ball

$P$  – probability that a random point is in  $V$

$N$  - total number of samples

$K$  - number of nearest neighbors

$N_1$  - total number of samples from class 1

$K_1$  - number of samples from class 1 in  $K$



## Bayes decision rule

**Bayes Rule:**  $P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$

$$P(Y = y|X = x) = \frac{P(X = x|Y = y)P(Y = y)}{P(X = x)}$$



# Bayes decision rule

x – input feature set  
y - label

- If we know the conditional probability  $p(x | y)$  and class priors  $p(y)$  we can determine the appropriate class by using Bayes rule:

$$P(y = i | x) = \frac{P(x | y = i)P(y = i)}{P(x)} \stackrel{\text{def}}{=} q_i(x)$$

Minimizes our probability of making a mistake

- We can use  $q_i(x)$  to select the appropriate class.
- We chose class 0 if  $q_0(x) \geq q_1(x)$  and class 1 otherwise
- This is termed the ‘Bayes decision rule’ and leads to optimal classification.
- However, it is often very hard to compute ...

Note that  $p(x)$  does not affect our decision




# Bayes decision rule

$$P(y = i | x) = \frac{P(x | y = i)P(y = i)}{P(x)} \stackrel{def}{=} q_i(x)$$

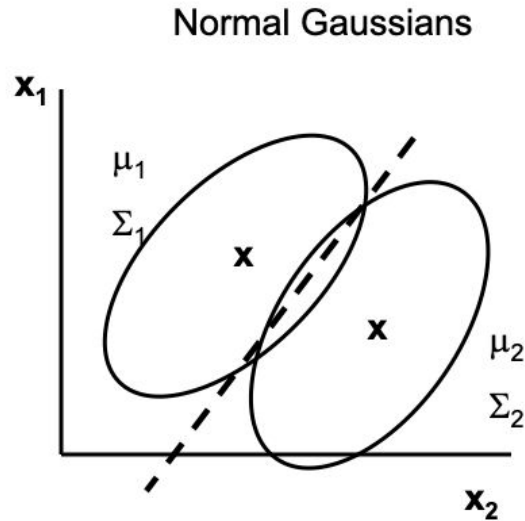
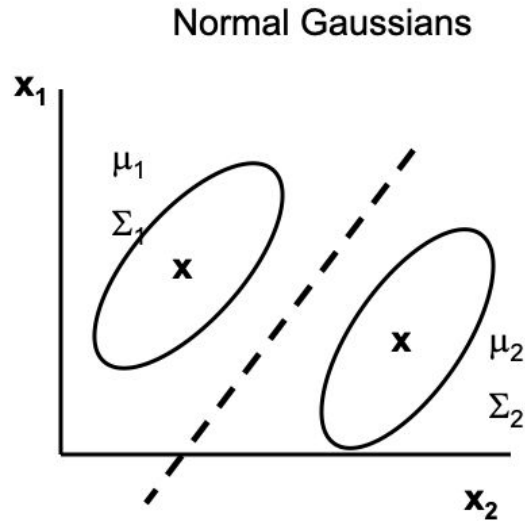
- We can also use the resulting probabilities to determine our **confidence** in the class assignment by looking at the likelihood ratio:

$$L(x) = \frac{q_0(x)}{q_1(x)}$$



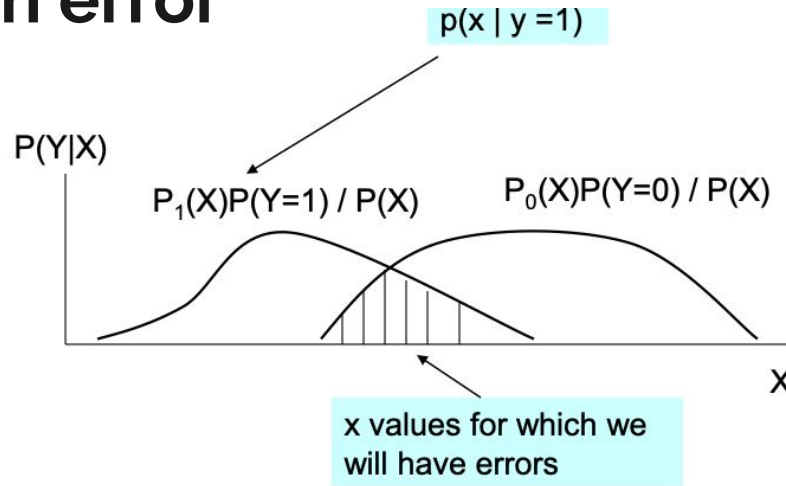
Also known as likelihood ratio, we will talk more about this later

# Binary case: separable v.s. non-separable



# Classification error

- For the Bayes decision rule we can calculate the probability of an error
- This is the probability that we assign a sample to the wrong class, also known as the **risk**



- The risk for sample  $x$  is:

$$R(x) = \min\{P_1(x)P(y=1), P_0(x)P(y=0)\} / P(x)$$

Risk can be used to determine a 'reject' region

# Bayes error

- The probability that we assign a sample to the wrong class, is known as the **risk**

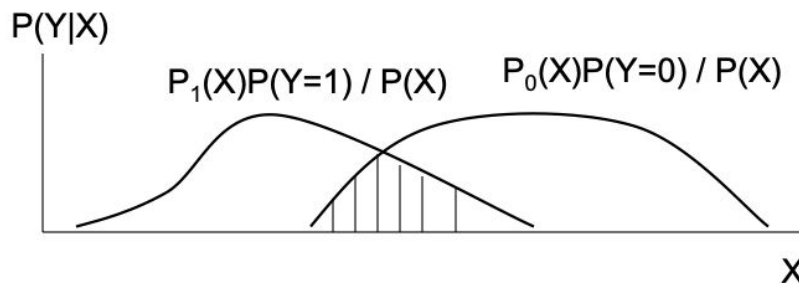
- The risk for sample  $x$  is:

$$R(x) = \min\{P_1(x)P(y=1), P_0(x)P(y=0)\} / P(x)$$

- We can also compute the expected risk (the risk for the entire range of values of  $x$ ):

$$\begin{aligned} E[r(x)] &= \int r(x)p(x)dx \\ &= \int \min\{p_1(x)p(y=1), p_0(x)p(y=0)\} dx \\ &= p(y=0) \int_{L_1} p_0(x)dx + p(y=1) \int_{L_0} p_1(x)dx \end{aligned}$$

$L_1$  is the region where we assign instances to class 1





# Takeaways

- Optimal decision using Bayes rule
- Type of classifiers
- Effect of values of  $K$  on KNN classifiers
- Probabilistic interpretation of KNN



# References

- Christopher Bishop: Pattern Recognition and Machine Learning, Chapter 14.4
- Tom Mitchell: Machine Learning, Chapter 3, 8
- Kevin Murphy: Machine Learning: A probabilistic perspective, Chapter 14
- Trevor Hastie, Robert Tibshirani, Jerome Friedman: The Elements of Statistical Learning: Data Mining, Inference and Prediction, Chapter 6, 9, 13
- Ziv Bar-Joseph, Tom Mitchell, Pradeep Ravikumar and Aarti Singh: CMU 10-701