Introduction to Machine Learning and Data Mining Lecture-8: Mid-term Review

Prof. Eugene Chang

Python

Python in one slide

- Interpreted language
- Comments: # and ’’’
- Variables are assigned, not declared
- Use indentation to determine code block boundary (scope)
- Data types: numbers, strings, lists, tuples, dictionary, array
- Control statements: if elif else, for, while
- Expressions
- Functions: built-in
  - Functions: def function_name(*arguments*):
  - Anonymous function: lamda *arguments*: *simple_expression*
- Modules: import

Review String Slicing

```
s = "Good Afternoon"

s[0]  evaluates to "G"

s[5:10] selects "After"      # string slicing

s[:10] selects "Good After"
s[5:]  selects "Afternoon"

s[-4:] selects "noon"     # last 4 characters
```

Review Lists

Ordered sequence of items

Can be floats, ints, strings, Lists

a = [16, 25.3, “hello”, 45]

a[0] contains 16

a[-1] contains 45

a[0:2] is a list containing [16, 25.3]

Create a List

```
days = [ ]
days.append(“Monday”)
days.append(“Tuesday”)

years = range(2000, 2014)
```

List Methods

List is a Class with data & subroutines:

d.insert( )
d.remove( )
d.sort( )

Can concatenate lists with +

Tuple

Designated by ( ) parenthesis

A List that can not be changed. Immutable.
No append.

Good for returning multiple values from a subroutine function.

Can extract slices.

Review math module

```
import math
dir(math)

math.sqrt(x)
math.sin(x)
math.cos(x)
```

```
from math import *
dir()

sqrt(x)
```

```
from math import pi
dir()

print pi
```

import a module

```
import math              # knows where to find it
_____________________________________________
import sys
sys.path.append(“C:\Users\Yuhlin\Documents\Python Scripts”)
import  mypython.py      # import your own code
_____________________________________________
if task == 3:
    import math          # imports can be anywhere
```

Review Defining a Function

Block of code separate from main.

Define the function before calling it.

```
def myAdd(a, b):          # define before calling
     return a + b

p = 25                    # main section of code
q = 30

r = myAdd(p, q)
```

Keyword Arguments

Provide default values for optional arguments.

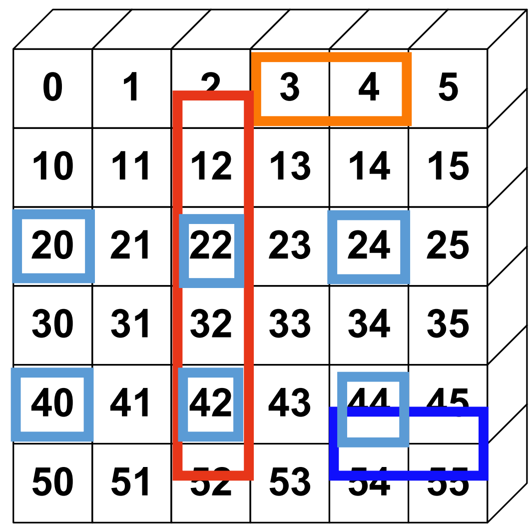
```
def setLineAttributes(color="black",
    style="solid", thickness=1):
    ...
```

Call function from main program

```
setLineAttributes(style="dotted")
setLineAttributes("red", thickness=2)
```

Numpy

- N-d array: a = array([[0, 1, 2, 3], [10,11,12,13]])
- Indexing
  - Start at 0, -1: last element, -2: last element - 1
- Shape functions
  - a.shape -> (row, column) = (2, 4)
  - Resize, swapaxes, flatten
- Slicing
  - a[2::2,::2] -> array([[20, 22, 24], [40, 42, 44]])
  - a[:,2] -> array([2,12,22,32,42,52])
  - a[0,3:5] -> array([3, 4])
  - a[4:,4:] -> array([[44, 45], [54, 55]])
- Array methods
  - Calculation
  - Statistics



Scikit-learn

- Cover many standard machine learning techniques
- Training
  - estimator.fit(X_train, y_train)
- Classification, regression, clustering
  - y_pred = estimator.predict(X_test)
- Predictive models, density estimation
  - test_score = estimator.score(X_test)
- Filters, dimension reduction, latent variables
  - X_new = estimator.transform(X_test)
- Code sample

```
>>> from sklearn import linear_model
>>> regr= linear_model.LinearRegression()
>>> regr.fit(X_train, y_train)
>>> y_pred = regr.predict(X_test)
```

Array Calculation Methods

SUM FUNCTION

```
>>> a = array([[1,2,3], [4,5,6]], float)

# Sum defaults to summing all
# *all* array values.
>>> sum(a)
21.

# supply the keyword axis to
# sum along the 0th axis.
>>> sum(a, axis=0)
array([5., 7., 9.])

# supply the keyword axis to
# sum along the last axis.
>>> sum(a, axis=-1)
array([6., 15.])
```

SUM ARRAY METHOD

```
# The a.sum() defaults to
# summing *all* array values
>>> a.sum()
21.

# Supply an axis argument to
# sum along a specific axis.
>>> a.sum(axis=0)
array([5., 7., 9.])
```

PRODUCT

```
# product along columns.
>>> a.prod(axis=0)
array([ 4., 10., 18.])

# functional form.
>>> prod(a, axis=0)
array([ 4., 10., 18.])
```

Min/Max

MIN

```
>>> a = array([2.,3.,0.,1.]) >>> a.min(axis=0)
0.
# use Numpy's amin() instead
# of Python's builtin min()
# for speed operations on
# multi-dimensional arrays.
>>> amin(a, axis=0)
0.
```

MAX

```
>>> a = array([2.,1.,0.,3.]) >>> a.max(axis=0)
3.
# functional form
>>> amax(a, axis=0)
3.
```

ARGMIN

```
# Find index of minimum value.
>>> a.argmin(axis=0)
2
# functional form
>>> argmin(a, axis=0)
2
```

ARGMAX

```
# Find index of maximum value.
>>> a.argmax(axis=0)
1
# functional form
>>> argmax(a, axis=0)
1
```

Statistics Array Methods

MEAN

```
>>> a = array([[1,2,3],
               [4,5,6]], float)

# mean value of each column
>>> a.mean(axis=0)
array([ 2.5,  3.5,  4.5])
>>> mean(a, axis=0)
array([ 2.5,  3.5,  4.5])
>>> average(a, axis=0)
array([ 2.5,  3.5,  4.5])

# average can also calculate
# a weighted average
>>> average(a, weights=[1,2],
...         axis=0)
array([ 3., 4., 5.])
```

STANDARD DEV./VARIANCE

```
# Standard Deviation
>>> a.std(axis=0)
array([ 1.5,  1.5,  1.5])

# Variance
>>> a.var(axis=0)
array([2.25, 2.25, 2.25])
>>> var(a, axis=0)
array([2.25, 2.25, 2.25])
```

Other Array Methods

CLIP

```
# Limit values to a range

>>> a = array([[1,2,3],
               [4,5,6]], float)

# Set values < 3 equal to 3.
# Set values > 5 equal to 5.
>>> a.clip(3,5)
>>> a
array([[ 3.,  3.,  3.],
       [ 4.,  5.,  5.]])
```

ROUND

```
# Round values in an array.
# Numpy rounds to even, so
# 1.5 and 2.5 both round to 2.
>>> a = array([1.35, 2.5, 1.5])
>>> a.round()
array([ 1.,  2.,  2.])

# Round to first decimal place.
>>> a.round(decimals=1)
array([ 1.4,  2.5,  1.5])
```

POINT TO POINT

```
# Calculate max – min for
# array along columns
>>> a.ptp(axis=0)
array([ 3.0,  3.0,  3.0])
# max – min for entire array.
>>> a.ptp(axis=None)
5.0
```

Linear Regression

Definition

- In **linear regression**, we assume that the model that generates the data involved **only** a linear combination of input variables.

$$y(\vec{x}, \vec{w}) = w_0 + w_1 x_1 + \ldots + w_D x_D$$

$$y(\vec{x}, \vec{w}) = w_0 + \sum_{j=1}^{D} w_j x_j$$

Where **w** is a vector of weights which define the D parameters of the model

Evaluation

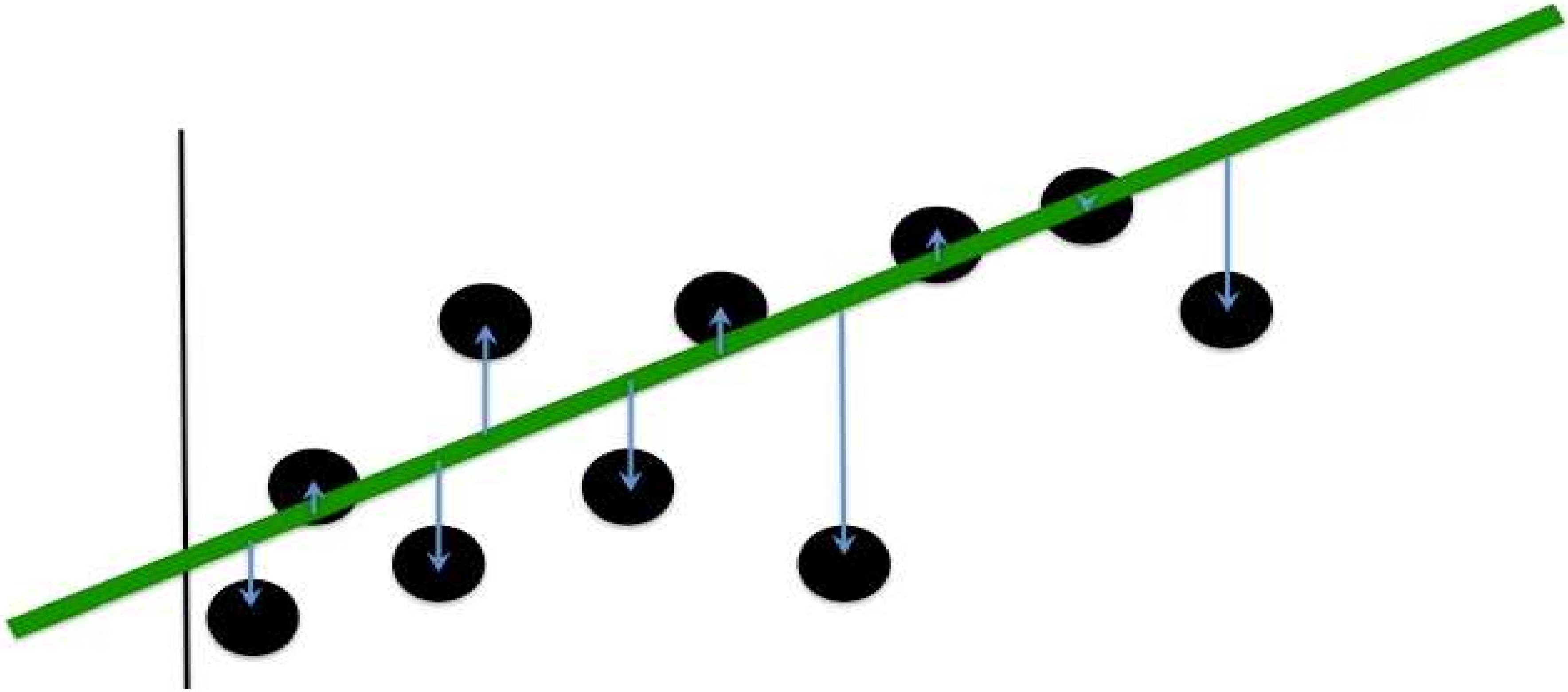
- How can we evaluate the performance of a regression solution?
- Error Functions (or Loss functions)
  - Squared Error

$$E(t_i, y(\vec{x}_i, \vec{w})) = \frac{1}{2}(t_i - y(\vec{x}_i, \vec{w}))^2$$

  - Linear Error

$$E(t_i, y(\vec{x}_i, \vec{w})) = |t_i - y(\vec{x}_i, \vec{w})|$$

Regression Error



How do we "learn" parameters

- For the 2-*d* problem (line) there are coefficients for the bias and the independent variable (*y*-intercept and slope)

$$Y = w_0 + w_1 X$$

- To find the values for the coefficients which minimize the objective function we take the partial derivates of the objective function (SSE) with respect to the coefficients. Set these to 0, and solve.

$$w_1 = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - \left(\sum x\right)^2} \qquad w_0 = \frac{\sum y - w_1 \sum x}{n}$$

Least Square Solution Examples

Y	3	5	7
X	1	2	3

$$Y = 1 + 2X$$

Y	3	6	6
X	1	2	3

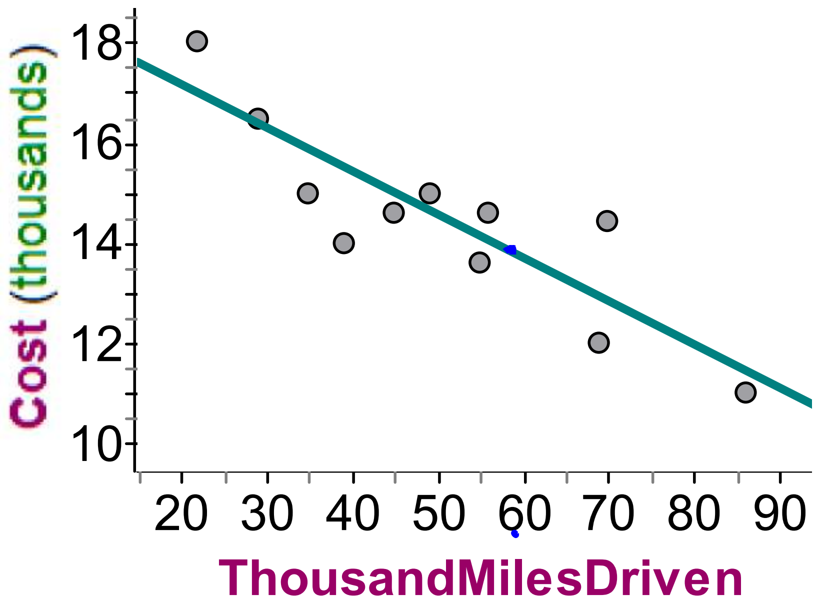
$$Y = 2 + 1.5X$$

$$w_1 = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - \left(\sum x\right)^2}$$

$$w_0 = \frac{\sum y - w_1 \sum x}{n}$$

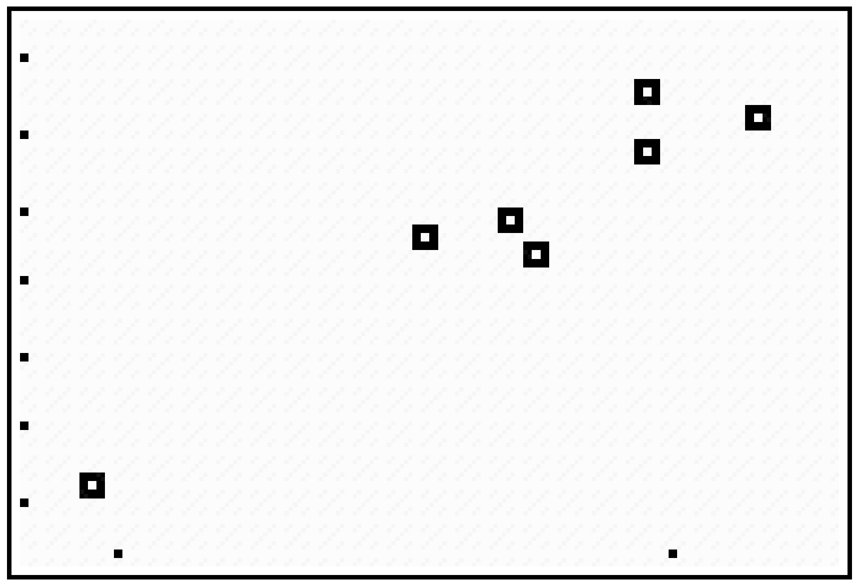
The following data shows the number of miles driven and advertised price for 11 used Honda CR-Vs from the 2002-2006 model years (prices found at www.carmax.com). The scatterplot below shows a strong, negative linear association between number of miles and advertised cost. The correlation is -0.874. The line on the plot is the regression line for predicting advertised price based on number of miles.

Thousand Miles Driven	Cost (dollars)
22	17998
29	16450
35	14998
39	13998
45	14599
49	14988
55	13599
56	14599
69	11998
70	14450
86	10998



Fat vs Calories in Burgers

Fat (g)	Calories
19	410
31	580
34	590
35	570
39	640
39	680
43	660



Decision Tree

Definition of a Decision Tree

- A **Tree** data structure
- Each **internal node** corresponds to a feature
- **Leaves** are associated with target values.
- Nodes with **nominal features** have *N* children, where *N* is the number of nominal values
- Nodes with **continuous features** have two children for values less than and greater than or equal to a **break point**.

Example Data Set

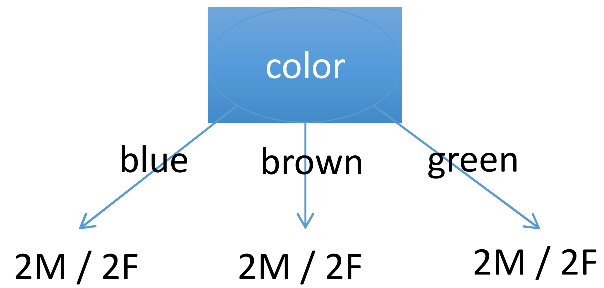
Height	Weight	Eye Color	Gender
66	170	Blue	Male
73	210	Brown	Male
72	165	Green	Male
70	180	Blue	Male
74	185	Brown	Male
68	155	Green	Male
65	150	Blue	Female
64	120	Brown	Female
63	125	Green	Female
67	140	Blue	Female
68	165	Brown	Female
66	130	Green	Female

Baseline Classification Accuracy

- Select the majority class.
  - Here 6/12 Male, 6/12 Female.
  - Baseline Accuracy: 50%
- How good is each branch?
  - The improvement to classification accuracy

Training Example

- Possible branches



50% Accuracy before Branch

50% Accuracy after Branch

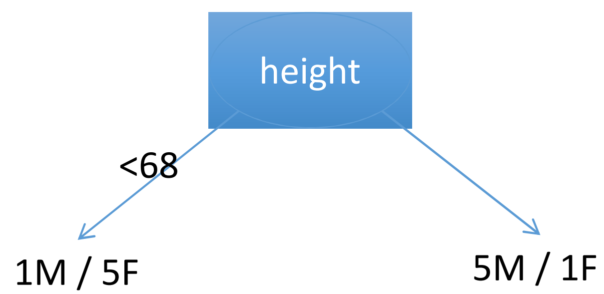
0% Accuracy Improvement

Example Data Set

Height	Weight	Eye Color	Gender
63	125	Green	Female
64	120	Brown	Female
65	150	Blue	Female
66	170	Blue	Male
66	130	Green	Female
67	140	Blue	Female
68	145	Brown	Female
68	155	Green	Male
70	180	Blue	Male
72	165	Green	Male
73	210	Brown	Male
74	185	Brown	Male

Training Example

- Possible branches



50% Accuracy before Branch

83.3% Accuracy after Branch

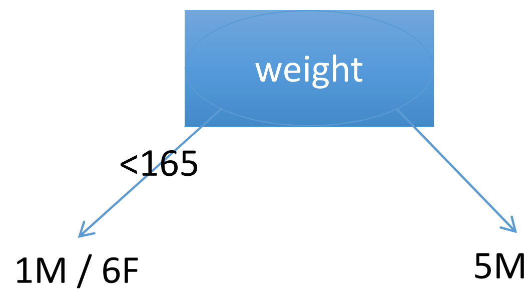
33.3% Accuracy Improvement

Example Data Set

Height	Weight	Eye Color	Gender
64	120	Brown	Female
63	125	Green	Female
66	130	Green	Female
67	140	Blue	Female
68	145	Brown	Female
65	150	Blue	Female
68	155	Green	Male
72	165	Green	Male
66	170	Blue	Male
70	180	Blue	Male
74	185	Brown	Male
73	210	Brown	Male

Training Example

- Possible branches



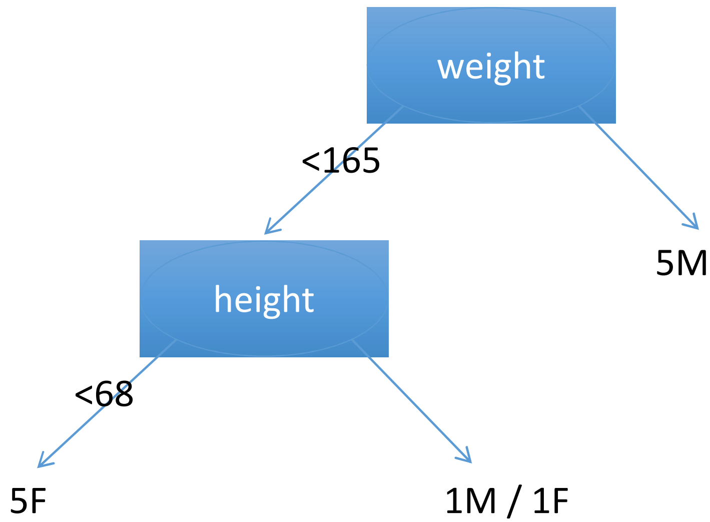
50% Accuracy before Branch

91.7% Accuracy after Branch

41.7% Accuracy Improvement

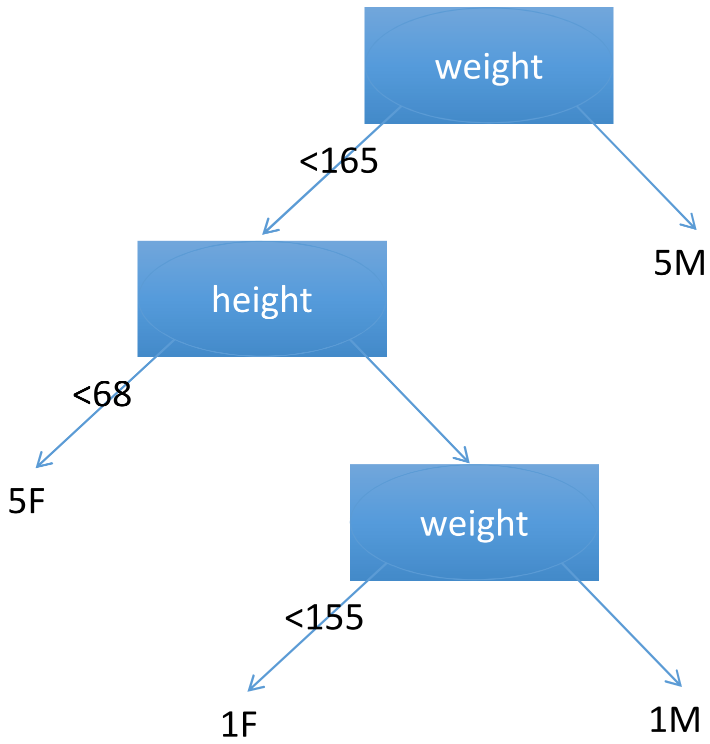
Training Example

- Recursively train child nodes.



Training Example

- Finished Tree

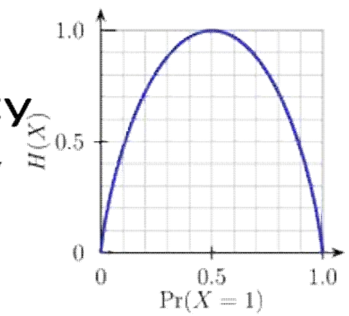


Decision Tree Induction Algorithm

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in top-down recursive divide-and-conquer
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

Brief Review of Entropy

- Entropy (Information Theory)
  - A measure of uncertainty associated with a random variable
  - Calculation: For a discrete random variable Y taking m distinct values $\{y_1, \dots, y_m\}$,
    - $H(Y) = -\sum_{i=1}^{m} p_i \log(p_i)$, where $p_i = P(Y = y_i)$
  - Interpretation:
    - Higher entropy => higher uncertainty
    - Lower entropy => lower uncertainty
- Conditional Entropy
  - $H(Y|X) = \sum_x p(x) H(Y|X = x)$



m = 2

Attribute Selection Measure: Information Gain

- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i, D}|/|D|$
- Expected information (entropy) needed to classify a tuple in D:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

- Information needed (after using A to split D into v partitions) to classify D:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

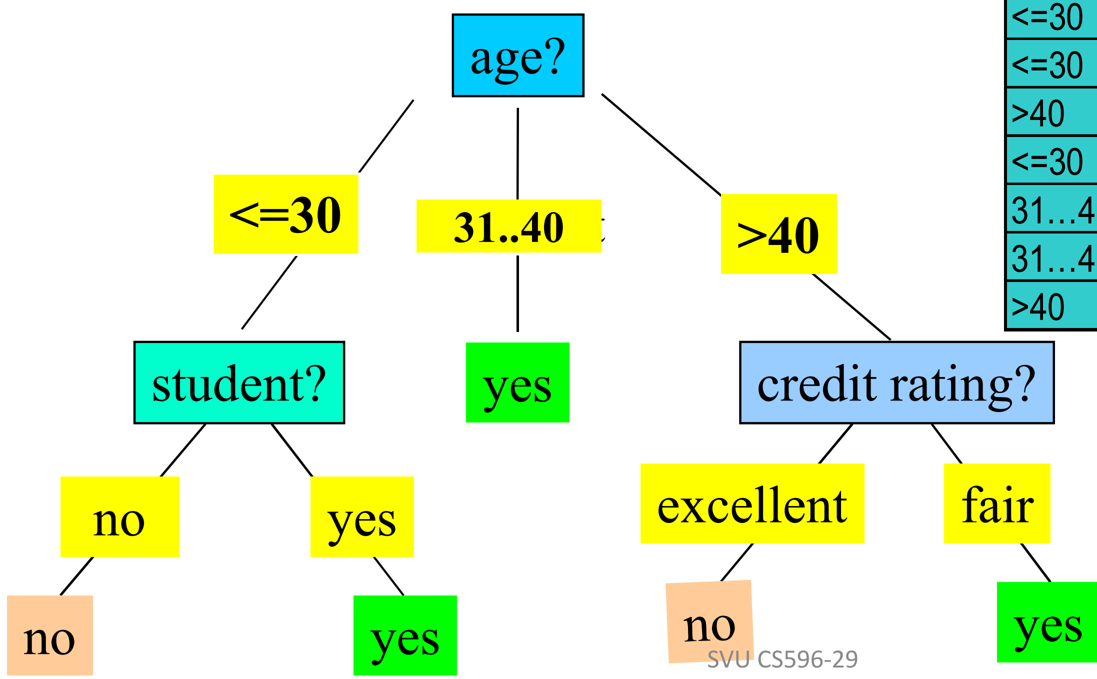
- Information gained by branching on attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

- Select the attribute with the highest information gain

Decision Tree Results

- Training data set: Buys_computer
- The data set follows an example of Quinlan's ID3 (Playing Tennis)
- Resulting tree:



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

Attribute Selection: Information Gain

- Class P: buys_computer = “yes”
- Class N: buys_computer = “no”

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940$$

$$Info_{age}(D) = \frac{5}{14}I(2,3) + \frac{4}{14}I(4,0) + \frac{5}{14}I(3,2) = 0.694$$

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

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

$\frac{5}{14}I(2,3)$ means “age <=30” has 5 out of 14 samples, with 2 yes’es and 3 no’s. Hence

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

Similarly,

$$Gain(income) = 0.029$$

$$Gain(student) = 0.151$$

$$Gain(credit\_rating) = 0.048$$

Bayes and Naïve Bayes

Bayes Probability

- Example: drawing a fruit from 2 color boxes
- Here the Box is the class (source), and the fruit is a feature, or observation.

	Orange	Apple	
Blue box	1	3	4
Red box	6	2	8
	7	5	12

Interpretation of Bayes Rule

Likelihood

Prior

Posterior

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}$$

- **Prior**: Information we have before observation.
- **Posterior**: The distribution of Y after observing X
- **Likelihood:** The likelihood of observing X given Y
- $P(X) = \sum_i P(X|Y_i)P(Y_i)$
- Example
  - P(Y_0) red box Prior probability = 1/3, P(Y_1) blue box Prior probability = 2/3
  - $P(X_0|Y_0)$ orange given red box = ¼, $P(X_1|Y_0)$ apple given red box = 3/4
  - $P(X_0|Y_1)$ orange given blue box = 3/4, $P(X_1|Y_1)$ apple given blue box = 1/4

The Correct Posterior Probability Calculation

$$P(X_0) = \sum_i P(X_0|Y_i)P(Y_i) = \frac{1}{4} * \frac{2}{3} + \frac{3}{4} * \frac{1}{3} = \frac{5}{12}$$

- After drawing an orange (X_0), the probability that it comes from red box (Y_0)

$$P(Y_0|X_0) = \frac{P(X_0|Y_0)P(Y_0)}{P(X_0)} = \frac{\frac{1}{4} * \frac{2}{3}}{\frac{5}{12}} = \frac{2}{5}.$$

- After drawing an orange (X_0), the probability that it comes from blue box (Y_1)

$$P(Y_1|X_0) = \frac{P(X_0|Y_1)P(Y_1)}{P(X_0)} = \frac{\frac{3}{4} * \frac{1}{3}}{\frac{5}{12}} = \frac{3}{5}.$$

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
    - E.g., Given that **X** will buy computer, the prob. that X is 31..40, medium income

Prediction Based on Bayes' Theorem

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

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

- Informally, this can be viewed as

  posteriori = likelihood x prior/evidence

- Predicts **X** belongs to C_i iff the probability $P(C_i|\mathbf{X})$ 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 Is to Derive the Maximum Posteriori

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

and $P(x_k|C_i)$ is $$g(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

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

Naïve Bayesian Categorization

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

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

- Therefore, we then only need to know $P(X_i \mid C_i)$ for each possible pair of a feature-value and a category.
- If Y and all X_i and binary, this requires specifying only $2n$ parameters:
  - $P(X_i\text{=true} \mid Y\text{=true})$ and $P(X_i\text{=true} \mid Y\text{=false})$ for each X_i
  - $P(X_i\text{=false} \mid Y) = 1 - P(X_i\text{=true} \mid Y)$
- Compared to specifying 2^n parameters without any independence assumptions.

Naïve Bayes Example

- Y (binary class): positive or negative
- X (features)
  - Size: large, medium, small
  - Color: red, green, blue
  - Shape: square, triangle, circle
- Independence among features

Naïve Bayes Example

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

Test Instance:
<medium, red, circle>

Naïve Bayes Example

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

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

Naïve Bayes Example

$P(\text{positive} \mid X) = P(\text{positive}) * P(\text{medium} \mid \text{positive}) * P(\text{red} \mid \text{positive}) * P(\text{circle} \mid \text{positive}) \, / \, P(X)$

$0.5 \quad * \quad 0.1 \quad * \quad 0.9 \quad * \quad 0.9$

$= 0.0405 \, / \, P(X) \quad = 0.0405 \, / \, 0.0495 = 0.8181$

$P(\text{negative} \mid X) = P(\text{negative}) * P(\text{medium} \mid \text{negative}) * P(\text{red} \mid \text{negative}) * P(\text{circle} \mid \text{negative}) \, / \, P(X)$

$0.5 \quad * \quad 0.2 \quad * \quad 0.3 \quad * \quad 0.3$

$= 0.009 \, / \, P(X) \quad = 0.009 \, / \, 0.0495 = 0.1818$

$P(\text{positive} \mid X) + P(\text{negative} \mid X) = 0.0405 \, / \, P(X) + 0.009 \, / \, P(X) = 1$

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

Naïve Bayes Classifier: Training Dataset

Class:

C1:buys_computer = 'yes'

C2:buys_computer = 'no'

Data to be classified:

X = (age <=30,

Income = medium,

Student = yes

Credit_rating = Fair)

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

Naïve Bayes Classifier Example

- $P(C_i)$: 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

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

age	income	student	credit_rating	com
<=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

$P(X|C_i)$: 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")

Estimating Probabilities

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

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

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

Probability Estimation Example

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

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

P(positive | *X*) = 0.5 * 0.0 * 1.0 * 1.0 / P(X) = 0

P(negative | *X*) = 0.5 * 0.0 * 0.5 * 0.5 / P(X) = 0

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

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.

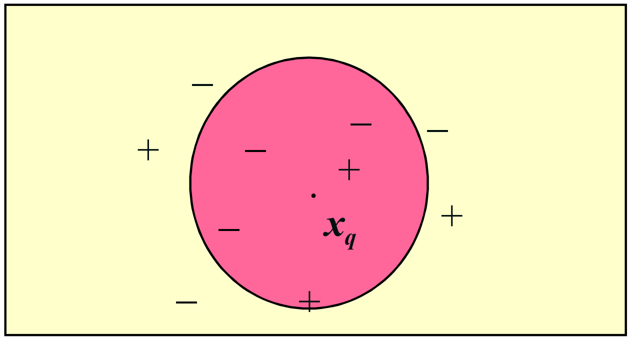
Laplace Smoothing Example

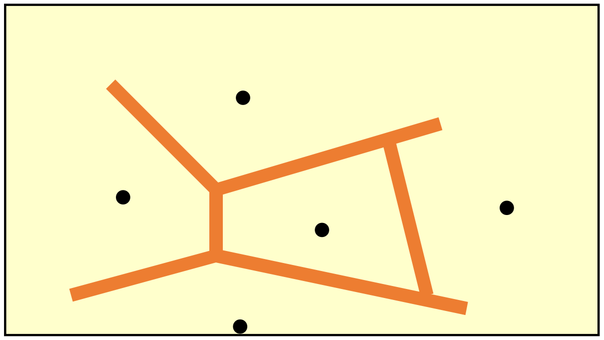
- Assume training set contains 10 positive examples:
  - 4: small
  - 0: medium
  - 6: large
- Estimate parameters as follows (if m=1, p=1/3)
  - P(small | positive) = (4 + 1/3) / (10 + 1) = 0.394
  - P(medium | positive) = (0 + 1/3) / (10 + 1) = 0.03
  - P(large | positive) = (6 + 1/3) / (10 + 1) =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

K Nearest Neighbors

k-Nearest Neighbor (kNN) Classification

- All instances correspond to points in the n-D space
- The nearest neighbor are defined in terms of Euclidean distance, dist(**X_1**, **X_2**)
- Target function could be discrete- or real- valued
- For discrete-valued, *k*-NN returns the most common value among the *k* training examples nearest to x_q
- Vonoroi diagram: the decision surface induced by 1-NN for a typical set of training examples





k-Nearest Neighbor (kNN) Classification

- Unlike all the other supervised learning methods, kNN does not build model from the training data.
- To classify a test instance *d*, define *k*-neighborhood *P* as *k* nearest neighbors of *d*
- Count number *n* of training instances in *P* that belong to class c_j
- Estimate P(c_j|*d*) as *n*/*k*
- No training is needed. Classification time is linear in training set size for each test case.

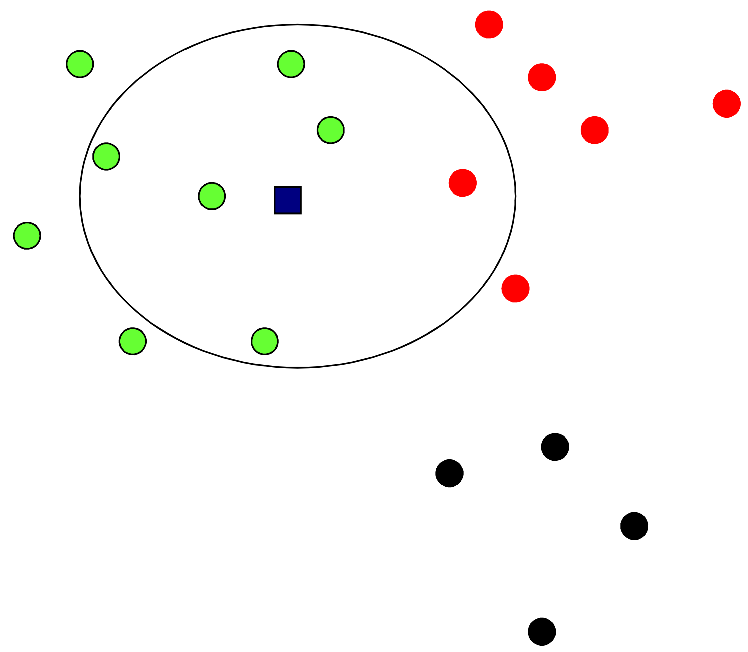
kNN Algorithm

Algorithm kNN(D, d, k)

1 Compute the distance between d and every example in D;
2 Choose the k examples in D that are nearest to d, denote the set by P ($\subseteq D$);
3 Assign d the class that is the most frequent class in P (or the majority class);

- *k* is usually chosen empirically via a validation set or cross-validation by trying a range of *k* values.
- Distance function is crucial, but depends on applications.

Example: k=6 (6NN)



Government

Science

Arts

A new point ■

P(science| ■)?

Clustering

Clustering

- Clustering is an **unsupervised** learning task.
  - There is no target value to shoot for.
- Identify groups of “similar” data points, and “dissimilar” from others
- **Partition** the data into groups (clusters) that satisfy these constraints
  1. Points in the same cluster should be **similar.**
  2. Points in different clusters should be **dissimilar.**
- **2 approaches**
  - Partitional
    - Divide the space into a fixed number of regions and position their boundaries appropriately
  - Hierarchical
    - Either merge or split clusters.

K-Means

- K-Means clustering is a **partitional** clustering algorithm.
  - Identify different partitions of the space for a fixed number of clusters
  - Input: a value for K – the number of clusters
  - Output: K cluster centroids.

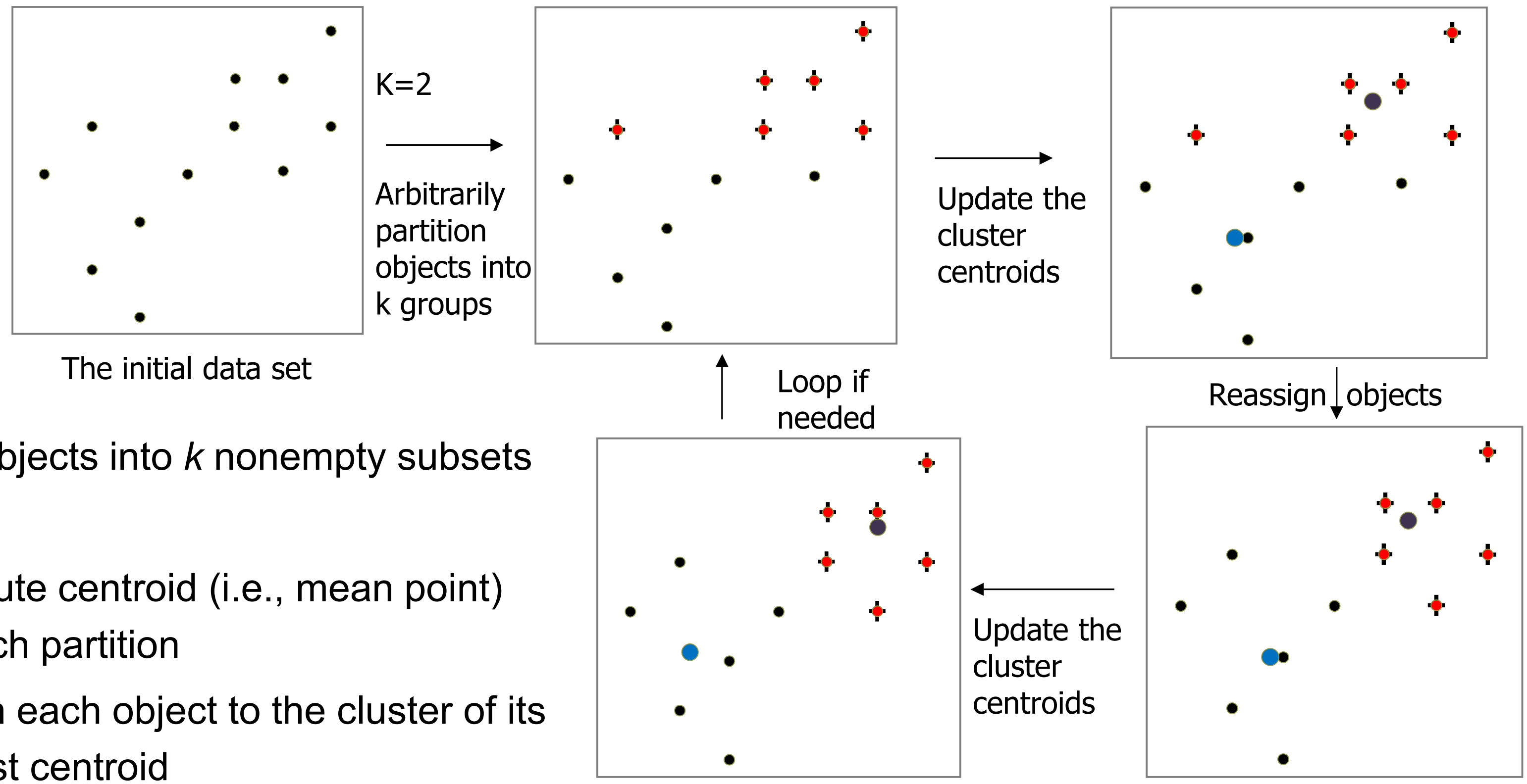
K-Means Algorithm

- Given an integer K specifying the number of clusters
- Initialize K cluster centroids
  - Select K points from the data set at random
  - Select K points from the space at random
- For each point in the data set, assign it to the cluster center it is closest to $\underset{C_i}{\operatorname{argmin}}\, d(\vec{x}, C_i)$
- Update each centroid based on the points that are assigned to it

$$C_i = \frac{1}{|C_i|} \sum_{\vec{x} \in C_i} \vec{x}$$

- If any data point has changed clusters, repeat

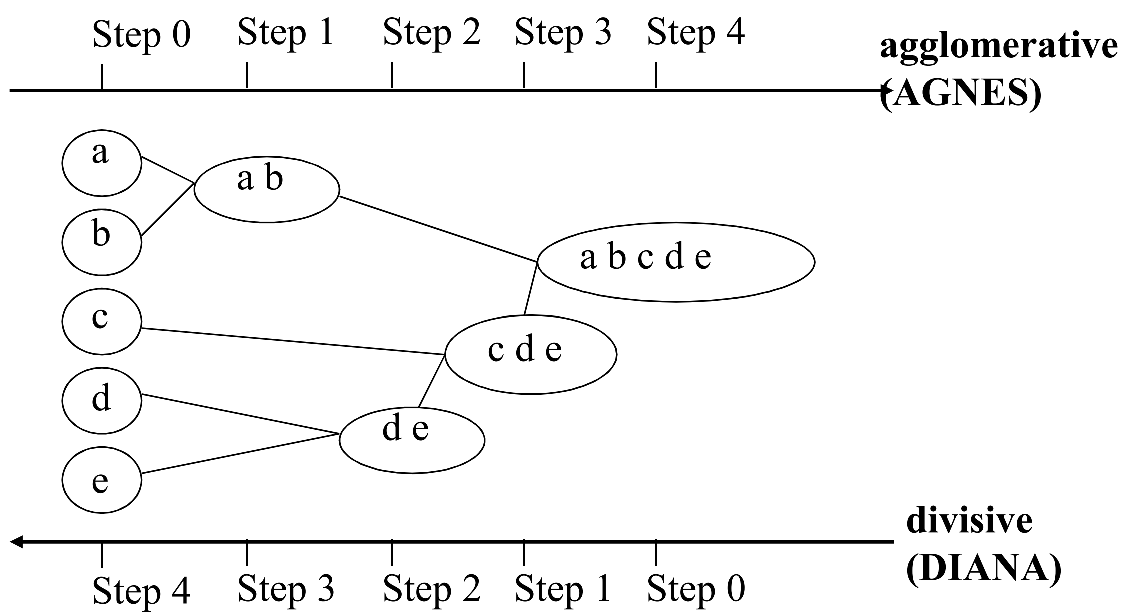
An Example of *K-Means* Clustering



- Partition objects into *k* nonempty subsets
- Repeat
  - Compute centroid (i.e., mean point) for each partition
  - Assign each object to the cluster of its nearest centroid
- Until no change

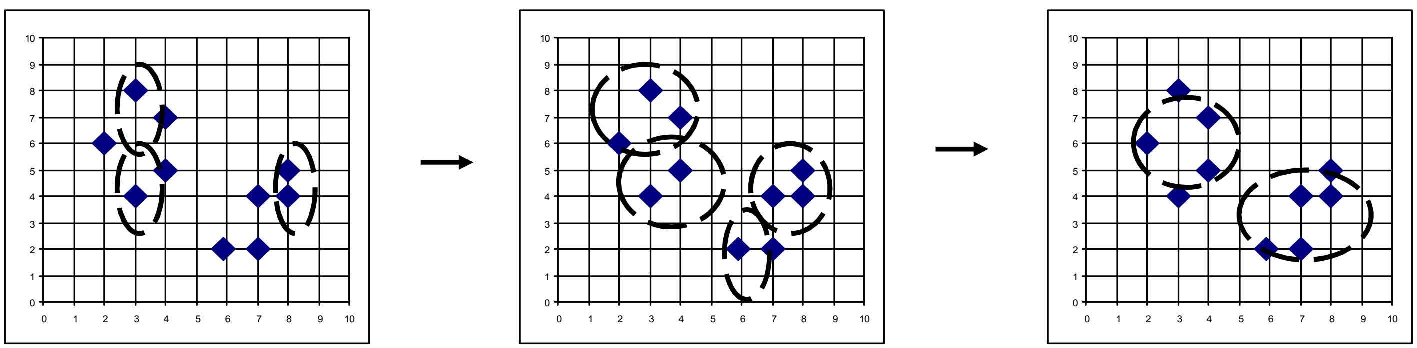
Hierarchical Clustering

- Use distance matrix as clustering criteria. This method does not require the number of clusters ***k*** as an input, but needs a termination condition



AGNES (Agglomerative Nesting)

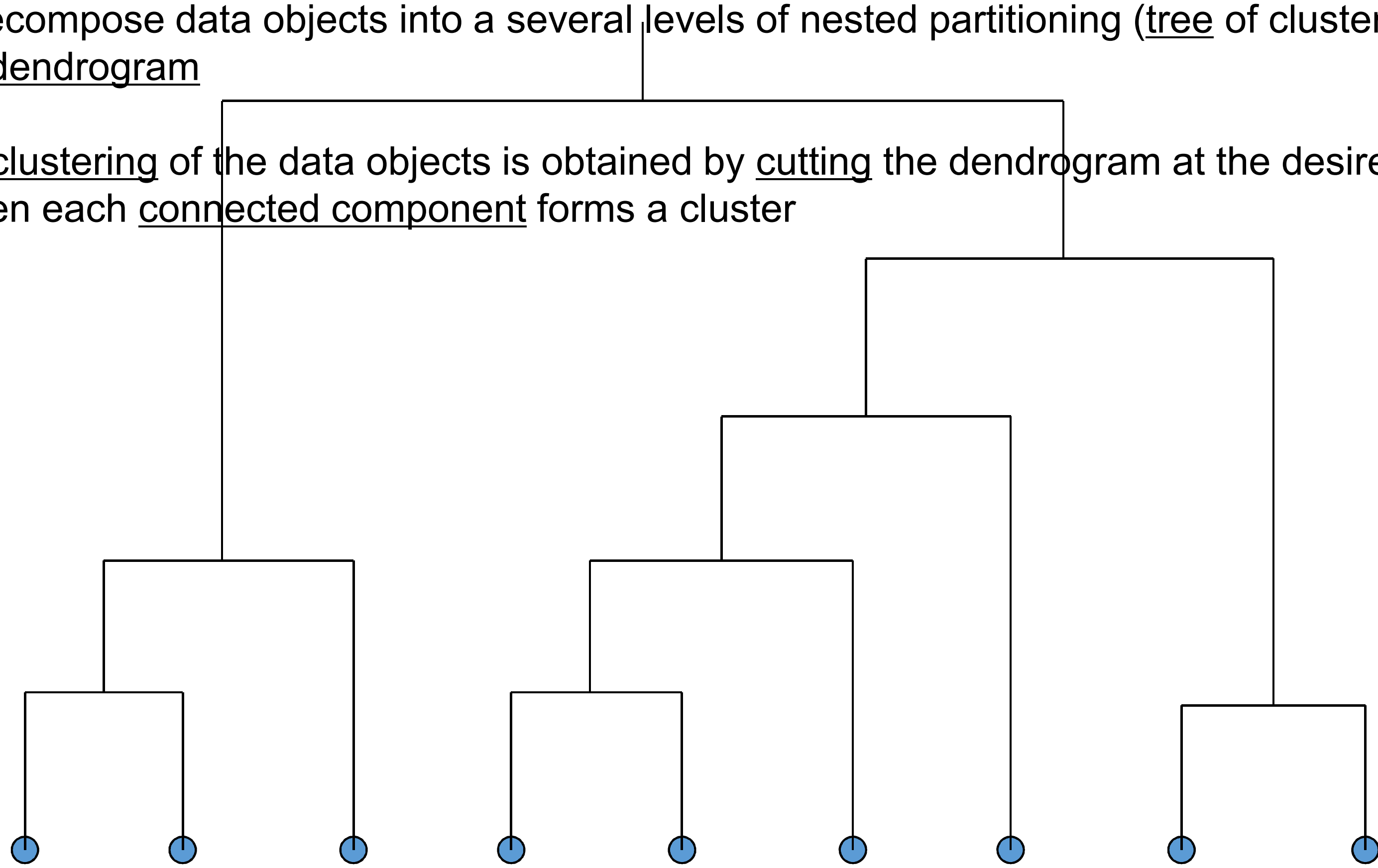
- Introduced in Kaufmann and Rousseeuw (1990)
- Implemented in statistical packages, e.g., Splus
- Use the **single-link** method and the dissimilarity matrix
- Merge nodes that have the least dissimilarity
- Go on in a non-descending fashion
- Eventually all nodes belong to the same cluster



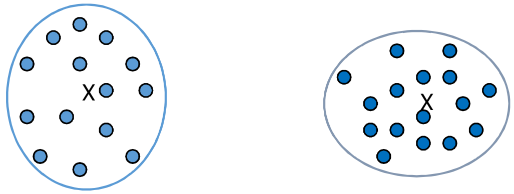
Dendrogram: Shows How Clusters are Merged

Decompose data objects into a several levels of nested partitioning (tree of clusters), called a dendrogram

A clustering of the data objects is obtained by cutting the dendrogram at the desired level, then each connected component forms a cluster



Distance between Clusters



- **Single link:** smallest distance between an element in one cluster and an element in the other, i.e., $dist(K_i, K_j) = min(t_{ip}, t_{jq})$
- **Complete link:** largest distance between an element in one cluster and an element in the other, i.e., $dist(K_i, K_j) = max(t_{ip}, t_{jq})$
- **Average:** avg distance between an element in one cluster and an element in the other, i.e., $dist(K_i, K_j) = avg(t_{ip}, t_{jq})$
- **Centroid:** distance between the centroids of two clusters, i.e., $dist(K_i, K_j) = dist(C_i, C_j)$
- **Medoid:** distance between the medoids of two clusters, i.e., $dist(K_i, K_j) = dist(M_i, M_j)$
  - Medoid: a chosen, centrally located object in the cluster

Validation and Performance Metrics

Holdout & Cross-Validation Methods

- **Holdout method**
  - Given data is randomly partitioned into two independent sets
    - Training set (e.g., 2/3) for model construction
    - Test set (e.g., 1/3) for accuracy estimation
  - Random sampling: a variation of holdout
    - Repeat holdout k times, accuracy = avg. of the accuracies obtained
- **Cross-validation** (*k*-fold, where k = 10 is most popular)
  - Randomly partition the data into *k mutually exclusive* subsets, each approximately equal size
  - At *i*-th iteration, use D_i as test set and others as training set
  - Leave-one-out: *k* folds where *k* = # of tuples, for small sized data
  - ***Stratified cross-validation***: folds are stratified so that class dist. in each fold is approx. the same as that in the initial data

Types of Errors or Metrics

- False Positives
  - The system predicted **TRUE** but the value was **FALSE**
  - aka "False Alarms" or Type I error

- False Negatives
  - The system predicted **FALSE** but the value was **TRUE**
  - aka "Misses" or Type II error

Simplest Measure: Accuracy

- Easily the most common and intuitive measure of classification performance.

$$Accuracy = \frac{\#correct}{N}$$

Problems with Accuracy

- Precision: how many hypothesized events were true events
- Recall: how many of the true events were identified
- F1-Measure: Harmonic mean of precision and recall

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN}$$

$$F = \frac{2PR}{P + R}$$

		True Values	
		Positive	Negative
Hyp Values	Positive	0	0
	Negative	10	90

F-Measure

		True Values	
		Positive	Negative
Hyp Values	Positive	1	0
	Negative	9	90

$$F_\beta = \frac{(1+\beta^2)PR}{(\beta^2 P) + R}$$

$$\begin{aligned} P &= 1 \\ R &= \frac{1}{10} \\ F_1 &= .18 \end{aligned}$$

F-Measure

		True Values	
		Positive	Negative
Hyp Values	Positive	10	50
	Negative	0	40

$$F_\beta = \frac{(1+\beta^2)PR}{(\beta^2 P) + R}$$

$$\begin{aligned} P &= \frac{10}{60} \\ R &= 1 \\ F_1 &= .29 \end{aligned}$$

F-Measure

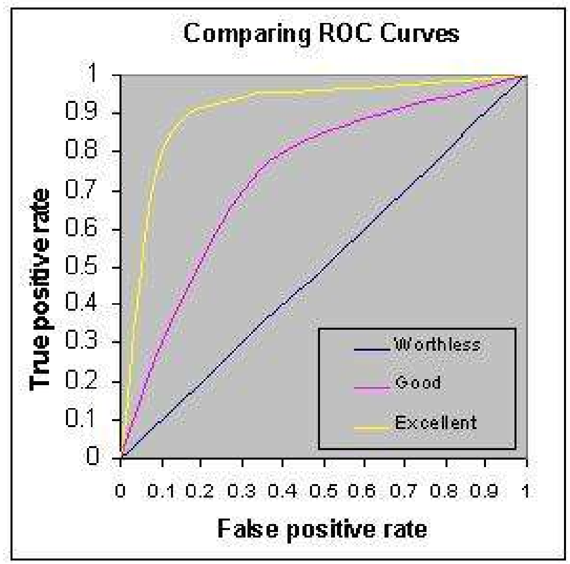
		True Values	
		Positive	**Negative**
Hyp Values	**Positive**	9	1
	Negative	1	89

$$F_\beta = \frac{(1+\beta^2)PR}{(\beta^2 P) + R}$$

$$\begin{aligned} P &= .9 \\ R &= .9 \\ F_1 &= .9 \end{aligned}$$

ROC Curves

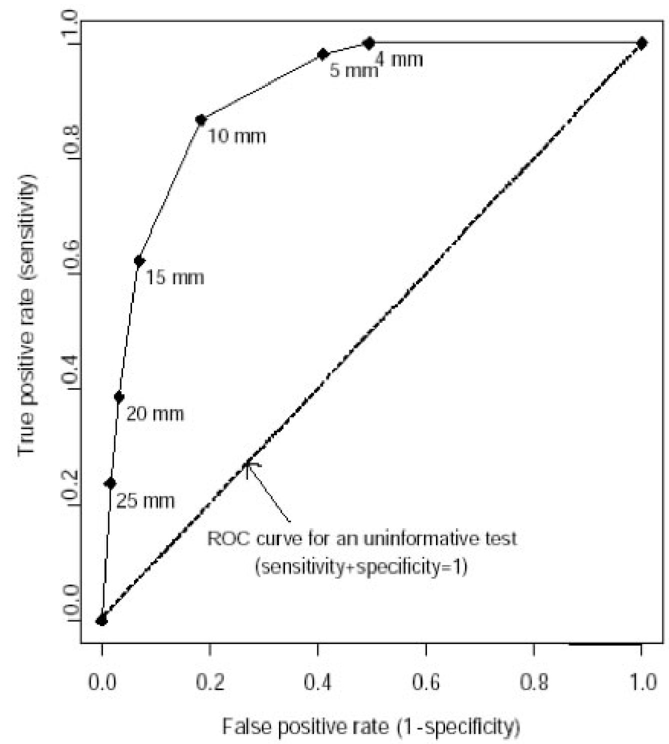
- It is common to plot classifier performance at a variety of settings or thresholds
- Receiver Operating Characteristic (ROC) curves plot true positives against false positives.
- The overall performance is calculated by the Area Under the Curve (AUC)



ROC Curve Example: Endometrial ultrasound

Ultrasound can be used to detect thickening in the lining of the uterus, which may be an early sign of cancer. If abnormal, a biopsy or minor surgical procedure is needed. This is painful and invasive, and has some risk. So our goal is to maximize the number of true positives (correctly diagnosed cancers) with an acceptable number of false positives (false alarms requiring a biopsy).

Cutoff for abnormal wall thickness	Sensitivity (%)	Specificity (%)	1 - Specificity (%)
> 4 mm	99	50	50
> 5 mm	97	61	39
> 10 mm	83	80	20
> 15 mm	60	90	10
> 20 mm	40	95	5
> 25 mm	20	98	2



Classifier Evaluation Metrics: Confusion Matrix

Confusion Matrix:

Actual class\Predicted class	C_1	$\neg C_1$
C_1	**True Positives (TP)**	**False Negatives (FN)**
$\neg C_1$	**False Positives (FP)**	**True Negatives (TN)**

Example of Confusion Matrix:

Actual class\Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**	7000
buy_computer = no	**412**	**2588**	3000
Total	7366	2634	10000

- Given *m* classes, an entry, $CM_{i,j}$ in a **confusion matrix** indicates # of tuples in class *i* that were labeled by the classifier as class *j*
- May have extra rows/columns to provide totals

Classifier Evaluation Metrics: Accuracy, Error Rate, Sensitivity and Specificity

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

- **Classifier Accuracy,** or recognition rate: percentage of test set tuples that are correctly classified

  Accuracy = (TP + TN)/All

- **Error rate:** *1 – accuracy*, or

  Error rate = (FP + FN)/All

- **Class Imbalance Problem**:
  - One class may be *rare*, e.g. fraud, or HIV-positive
  - Significant *majority of the negative class* and minority of the positive class
  - **Sensitivity**: True Positive recognition rate
    - **Sensitivity = TP/P**
  - **Specificity**: True Negative recognition rate
    - **Specificity = TN/N**

Summary: Precision and Recall, and F-measures

- **Precision**: exactness – what % of tuples that the classifier labeled as positive are actually positive

$$precision = \frac{TP}{TP + FP}$$

- **Recall:** completeness – what % of positive tuples did the classifier label as positive?
- Perfect score is 1.0

$$recall = \frac{TP}{TP + FN}$$

- Inverse relationship between precision & recall
- ***F* measure (*F_1* or *F*-score)**: harmonic mean of precision and recall,

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

- ***F_β*:** weighted measure of precision and recall
  - assigns ß times as much weight to recall as to precision

$$F_\beta = \frac{(1 + \beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$

Classifier Evaluation Metrics: Example

Actual class/Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**	7000
buy_computer = no	**412**	**2588**	3000
Total	7366	2634	10000

- *Accuracy = (***6954+2588)** / 10000 = 95.42%
- *Precision* = **6954**/7366 = 94.4%
- *Recall* = *Sensitivity* = **6954**/7000 = 99.34%
- Specificity = **2588** /3000 = 86.27%
- F1 = P*R / 2(P+R) = 48.4%