Statistical Data Analysis

N. Sairam

sairam@cse.sastra.edu

Unit-IV

Monte Carlo Simulation

- The term Monte Carlo simulation is frequently used to describe any method that involves the generation of random points as input for subsequent operations
  1. Simulations allow us to verify analytical work and to experiment with it
  2. Simulations are a way of obtaining results from models for which analytical solutions are not available

Monte Carlo Simulation

- Monte Carlo (MC) technique is a numerical method that makes use of random numbers to solve mathematical problems for which an analytical solution is not known
- The first article, "The Monte Carlo Method" by Metropolis and Ulam, has appeared for the first time in 1949, even though well before that certain statistical problems were solved using random numbers
- Since the simulation of random numbers is very time consuming, MC has became practical only with the advent of computers

Simulating π

- A simple MC simulation is the determination of π
- Suppose we have a circle with radius r = 1 inscribed within a square. Then the area ratio is:

$$\frac{A_{circle}}{A_{square}} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4} \tag{1}$$

- To determine π, we select randomly n points $p_i=(x_i,y_i)$, for i=1,2,...n in the square and approximate area ratio by

$$\frac{A_{circle}}{A_{square}} = \frac{m}{n} \tag{2}$$

Simulating π

- where m is the number of random points in the circle satisfying $x_i^2+y_i^2 \leq 1$. Hence $\pi = 4 \times \frac{m}{n}$
- This method is called hit-or-miss Monte Carlo since the estimate is computed as the actual ratio of hits to random tries. It is the least efficient MC method

Simulation of π

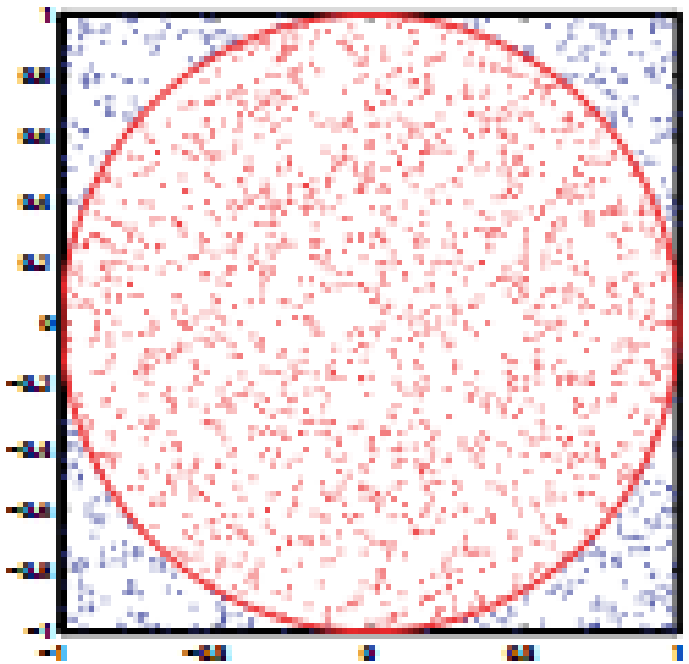


Figure: Compute π

Combinatorial Problems

- Many basic combinatorial problems can be solved exactly-but obtaining a solution is often difficult
- Even when one is able to find a solution, it is surprisingly easy to arrive at incorrect conclusions, missing factors like 1/2 or 1/n! and so on
- In contrast, simulations for typical combinatorial problems are often trivially easy to write
- Hence they are a great way to validate theoretical results, and they can be extended to explore problems that are not tractable otherwise

Some Examples

- If we place n balls into n boxes, what is the probability that no more than two boxes contain two or more balls? What if I told you that exactly m boxes are empty? What if at most m boxes are empty?
- If we try keys from a key chain containing n different keys, how many keys will we have to try before finding the one that fits the lock? How is the answer different if we try keys randomly (with replacement) as opposed to in order (without replacement)?

Envelope Example

- Suppose, we are presented with a choice of three closed envelopes. One envelope contains a prize, the other two are empty. After we have selected an envelope, it is revealed that one of the envelopes that we had not selected is empty. We are now permitted to choose again. What should we do? Stick with our initial selection? Randomly choose between the two remaining envelopes? Or pick the remaining envelope-that is, not the one that we selected initially and not the one that has been opened?
- This is a famous problem, which is sometimes known as the "Monty Hall Problem"

Python Code

```
import sys
import random as rnd
strategy = sys.argv[1]
# must be 'stick', 'choose',
 or 'switch'
wins = 0
for trial in range( 1000 ):
# The prize is always in envelope 0 ...
 but we don't know that!
envelopes = [0, 1, 2]
first_choice = rnd.choice( envelopes )
if first_choice == 0:
envelopes = [0, rnd.choice( [1,2] ) ]
# Randomly retain 1 or 2
```

Python Code

```
else:
envelopes = [0, first_choice]
# Retain winner and first choice
if strategy == 'stick':
second_choice = first_choice
elif strategy == 'choose':
second_choice = rnd.choice( envelopes )
elif strategy == 'switch':
envelopes.remove( first_choice )
second_choice = envelopes[0]
# Remember that the prize is in envelope 0
if second_choice == 0:
wins += 1
print wins
```

Obtaining Outcome Distributions

- Simulations can be helpful to verify with combinatorial problems, but the primary reason for using simulations is that they allow us to obtain results that are not available analytically
- To arrive at an analytical solution for a model, we usually have to make simplifying assumptions
- One particularly common one is to replace all random quantities with their most probable value
- This allows us to solve the model, but we lose information about the distribution of outcomes
- Simulations are a way of retaining the effects of randomness when determining the consequences of a model

Visitors Problem

- We had a visitor population making visits to a certain website. Because individual visitors can make repeat visits, the number of unique visitors grows more slowly than the number of total visitors. We found an expression for the number of unique visitors over time but had to make some approximations in order to make progress. In particular, we assumed that the number of total visitors per day would be the same every day, and be equal to the average number of visitors per day. (We also assumed that the fraction of actual repeat visitors on any given day would equal the fraction of repeat visitors in the total population.)

Mean Fit Model Strategy

- First, the number of visitors per day is no longer fixed, instead it is distributed according to a Gaussian distribution. Second, we have a notion of individual visitors (as elements of the list has visited), and on every "day" we make a random selection from this set of visitors to determine who does visit on this day and who does not

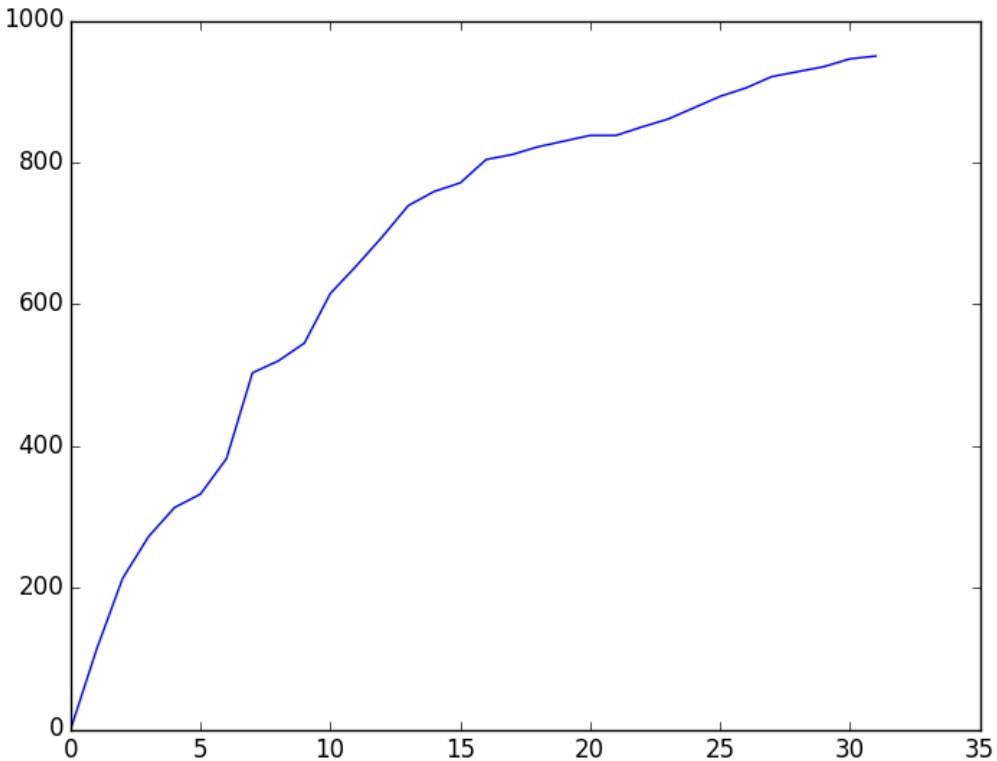
Python Code

```
import random as rnd
n = 1000
k = 100
s = 50
# total visitors
# avg visitors per day
# daily variation
def trial():
visitors_for_day = [0]
# No visitors on day 0
has_visited = [0]*n
# A flag for each visitor
for day in range( 31 ):
visitors_today = max( 0, int(rnd.gauss( k, s )) )
```

Python Code

```
# Pick the individuals who visited today and mark them
for i in rnd.sample( range( n ), visitors_today ):
has_visited[i] = 1
# Find the total number of unique visitors so far
visitors_for_day.append( sum(has_visited) )
return visitors_for_day
# Find the total number of unique visitors so far
visitors_for_day.append( sum(has_visited) )
return visitors_for_day
for t in range( 25 ):
r = trial()
for i in range( len(r) ):
print i, r[i]
print print
```

Graphical Result



Advantages and Limitations

- Advantages:
  - Easy to program
  - Fairly successful in the end i.e one can arrive at some approximate answer in the end
  - Valuable for verifying analytical work
  - Enhances the insight gained from the analytical, conceptual modeling of the the mechanisms driving a system
- Limitations:
  - Simulations produces numbers
  - Simulations cannot replace analytical modeling-simulations do not yield the kind of insight into the mechanisms driving certain developments that a good analytical model affords
  - Simulations without analysis produce only numbers, not insight

Resampling Methods

- Let us consider a sample of 'n' points from some population
- A point estimate like mean is not often powerful. We really want an interval estimate that gives us a sense of reliability of the answer
- To make statements about the reliability of an estimate only based on a set of observations, we need to make some assumptions about the way values are distributed
- Sampling distributions play vital role on this

Two important Problems to be addressed

1. Our assumptions about the shape of those distributions may not be correct, or we may not be able to formulate those distributions at all
2. Even if we know the sampling distribution, determining confidence limits from it may be tedious, opaque, and error-prone

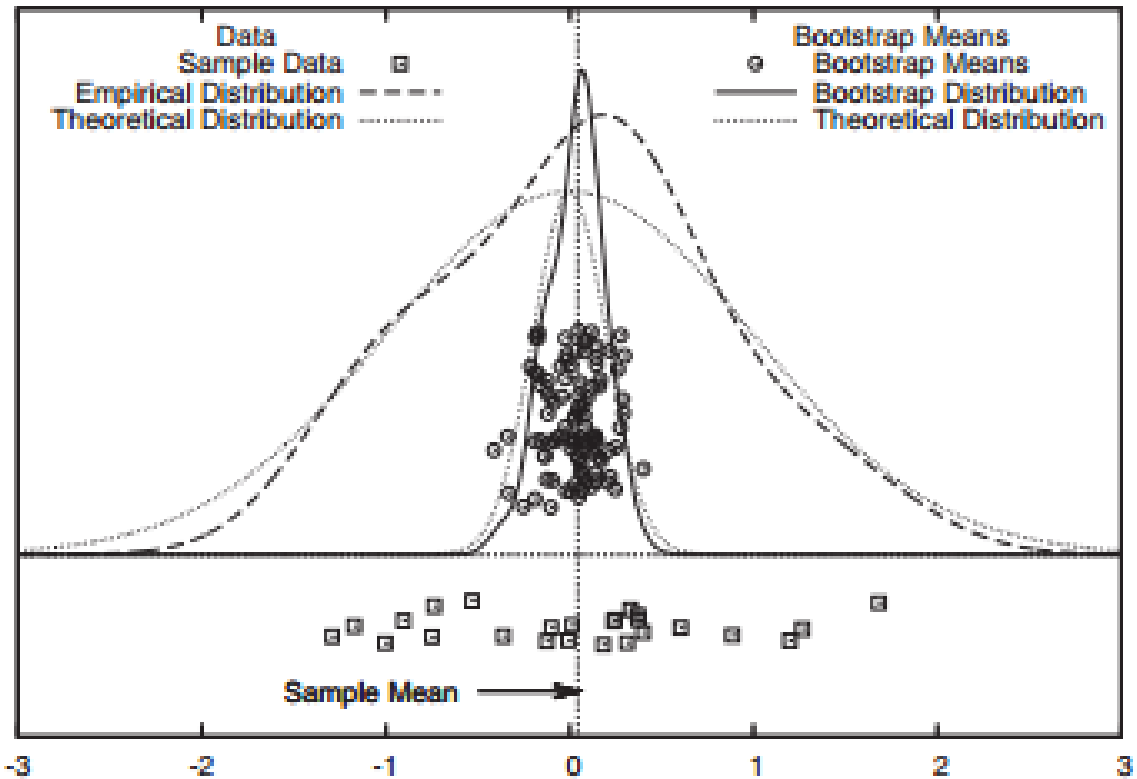
Bootstrap

- The bootstrap is an alternative approach for finding confidence intervals and similar quantities directly from the data
- Based on drawing addtional samples from the population instead of making assumptions about the distribution of values and then employing theoretical arguments
- Create additional samples of the same size with replacement
- Calculate the mean(or any other quantity) of these additional samples, then use this set of values for the mean to determine a measure of the spread of its distribution via any standard method(Ex:inter quartile range)

Example

- Analytical Estimate of the Standard Error
  - Let us draw n = 25 points from a standard Gaussian distribution
  - Then compute the observed sample mean and its standard error
  - The standard error is $\frac{\sigma}{\sqrt{n}}$ which is equal to $\frac{1}{5}$
- Bootstrap Estimate of the Standard Error
  - Let us draw 100 samples, each containing n = 25 points, from the original sample of 25 points
  - Points are drawn randomly with replacement (so that each point can be selected multiple times)
  - For each of these bootstrap samples, we calculate the mean
  - Find the spread of the distribution of these 100 bootstrap means

Example


Data
Sample Data
Empirical Distribution
Theoretical Distribution
Bootstrap Means
Bootstrap Means
Bootstrap Distribution
Theoretical Distribution
Sample Mean
-3
-2
-1
0
1
2
3

Example

- If we repeatedly took samples from the original distribution, the sample means would be distributed similarly to the bootstrap means
- Note:
  1. Bootstrapping is a method to estimate the spread of some quantity
  2. It is not a method to obtain "better" estimates of the original quantity itself - for that, it is necessary to obtain a larger sample by making additional drawings from the original population
  3. The bootstrap is not a way to give the appearance of a larger sample size by reusing point

When does bootstrapping work?

- The following conditions to be fulfilled
  1. The original sample must provide a good representation of the entire population
     - Samples should be sufficiently large and clean
     - No outliers
  2. The estimated quantity must depend "smoothly" on the data points
     - Does not work well for quantities that depend critically on only a few data points

Bootstrap Variants

- Non parametric Bootstrap: Points are drawn directly from the original sample
- Parametric Bootstrap:
  - The original population follows some particular probability distribution (such as the Gaussian)
  - The parameters(mean and SD) are estimated from the original sample
  - The bootstrap samples are then drawn from this distribution rather than from the original sample
  - Advantage:
    - The bootstrap values do not have to coincide exactly with the known data points

Jackknife

- No need to draw random samples
- Given an original sample consisting of "n" data points, calculate the "n" estimates of the quantity of interest by successively omitting one of the data points from the sample
- Use these "n" values in a similar way that we used values calculated from bootstrap samples
- It is a deterministic procedure since it does not contain any random element

Simulation

- A computer program that creates a virtual environment in order to study physical problems
- When to use simulation
  - hard to do real experiment, e.g. battle field, or banking system
- cheaper to do simulation,e.g. RTL simulation for IC design, or highway/freeway route planning
- analyzing bottleneck of current workflow
- When not to use simulation
  - more expensive to do simulation, e.g. simple harmonic motion
  - problems that can be analyzed by pencil and paper

Categories of Simulation

1. Continuous or Discrete
   - state variable is continuous, e.g. weather systems.
     Discrete-state variable is discrete, e.g. number of customers
2. Static or Dynamic
   - Static: represents a system at a particular point of time
     - representation of time is unnecessary
     - sometimes called Monte-Carlo simulation
   - Dynamic:represents systems as they change over time
     - e.g. banking system from 9:00 AM to 5:00 PM
3. Deterministic or Stochastic
   - Deterministic: contains no random variable
   - Stochastic: has one or more random variables

Discrete Event Simulation

- Discrete-Event Simulation is
  - Discrete
  - Dynamic
  - Stochastic
- Simulation for queueing in a post office is DES
- Mostly, but not limited to, queueing systems
  - factory work flow
  - freeway traffic simulation
  - network traffic simulation

Discrete Event Simulation World Views

- Activity-oriented
  - fixed increment of time
  - time-consuming
- Event Oriented
  - on each event, generate next event and put into event queue and sort
  - simulation time advances to next closest event
  - faster than activity-oriented
- Process Oriented
  - abstract one object into a process
  - easier to maintain in the end
  - SimPy belongs here

simpy 3.0.8

- SimPy is a process-based discrete-event simulation framework based on standard Python
- Its event dispatcher is based on Pythons generators and can also be used for asynchronous networking or to implement multi-agent systems (with both, simulated and real communication)
- Processes in SimPy are defined by Python generator functions and can, for example, be used to model active components like customers, vehicles or agents
- SimPy also provides various types of shared resources to model limited capacity congestion points (like servers, checkout counters and tunnels)

simpy 3.0.8

- Simulations can be performed "as fast as possible", in real time (wall clock time) or by manually stepping through the events
- SimPy is released under the MIT License

How simpy works?

- It is just an asynchronous event dispatcher
- You generate events and schedule them at a given simulation time
- Events are sorted by priority, simulation time, and an increasing event id
- An event also has a list of callbacks, which are executed when the event is triggered and processed by the event loop
- Events may also have a return value

How simpy works?

- The components involved in simpy are the Environment, events and the process functions that you write
- Process functions define the behavior of the simulation i.e. plain Python generator functions that yield instances of Event
- The environment stores these events in its event list and keeps track of the current simulation time
- If a process function yields an event, SimPy adds the process to the events callbacks and suspends the process until the event is triggered and processed
- When a process waiting for an event is resumed, it will also receive the events value

Example 1

```
import simpy

def example(env):
    event = simpy.events.Timeout(env, delay=1, value=42)
    value = yield event
    print('now=%d, value=%d' % (env.now, value))

env = simpy.Environment()
example_gen = example(env)
p = simpy.events.Process(env, example_gen)

env.run()
```

Explanation of the above Code

- The example() process function first creates a Timeout event
- It passes the environment, a delay, and a value to it
- The Timeout schedules itself at now + delay; other event types usually schedule themselves at the current simulation time
- The process function then yields the event and thus gets suspended
- It is resumed, when SimPy processes the Timeout event
- The process function also receives the events value (42)

Explanation of the above code

- Finally, the process function prints the current simulation time (that is accessible via the environments now attribute) and the Timeouts value
- If all required process functions are defined, you can instantiate all objects for your simulation
- In most cases, you start by creating an instance of Environment
- Starting a Process function involves two things:
  1. You have to call the process function to create a generator object
  2. You then create an instance of Process and pass the environment and the generator object to it. This will schedule an Initialize event at the current simulation time which starts the execution of the process function. The process instance is also an event that is triggered when the process function returns

Explanation of the above code

- Finally, you can start SimPys event loop
- By default, it will run as long as there are events in the event list, but you can also let it stop earlier by providing an until argument

Example 1

- A clock process that prints the current simulation time at each step:

```
import simpy
def clock(env, name, tick):
    while True:
        print(name, env.now)
        yield env.timeout(tick)
env = simpy.Environment()
env.process(clock(env, 'fast', 0.5))
env.process(clock(env, 'slow', 1))
env.run(until=2)
```

Example 2

- A counter with a random service time and customers who renege
- It models a bank counter and customers arriving t random times
- Each customer has a certain patience
- It waits to get to the counter until he/she is at the end of her tether
- If he/she gets to the counter, he/she uses it for a while before releasing it
- New customers are created by the source process every few time steps

Example 2 Python Code

- Scenario: A counter with a random service time and customers who renege

```
import random
import simpy
RANDOM_SEED = 42
NEW_CUSTOMERS = 5  # Total number of customers
INTERVAL_CUSTOMERS = 10.0  # Generate new customers
roughly every x seconds
MIN_PATIENCE = 1  # Min. customer patience
MAX_PATIENCE = 3  # Max. customer patience
```

Example 2 Python Code

```
def source(env, number, interval, counter):
    """Source generates customers randomly"""
    for i in range(number):
        c = customer(env, 'Customer%02d' % i,
        counter, time_in_bank=12.0)
        env.process(c)
        t = random.expovariate(1.0 / interval)
        yield env.timeout(t)
```

Example 2 Python Code

```
def customer(env, name, counter, time_in_bank):
    """Customer arrives, is served and leaves."""
    arrive = env.now
    print('%7.4f %s: Here I am' % (arrive, name))

    with counter.request() as req:
        patience = random.uniform(MIN_PATIENCE,
         MAX_PATIENCE)
        # Wait for the counter or abort at the
        end of our tether
        results = yield req | env.timeout(patience)
```

Example 2 Python Code

```
wait = env.now - arrive

        if req in results:
            # We got to the counter
            print('%7.4f %s: Waited %6.3f' %
            (env.now, name, wait))

            tib = random.expovariate(1.0 / time_in_bank)
            yield env.timeout(tib)
            print('%7.4f %s: Finished' % (env.now, name))
            else:
            # We reneged
            print('%7.4f %s: RENEGED after %6.3f'
             % (env.now, name, wait))
```

Example 2 Python Code

```
# Setup and start the simulation
print('Bank renege')
random.seed(RANDOM_SEED)
env = simpy.Environment()

# Start processes and run
counter = simpy.Resource(env, capacity=1)
env.process(source(env, NEW_CUSTOMERS,
INTERVAL_CUSTOMERS, counter))
env.run()
```

Finding Clusters

Definition (Cluster)

The term Clustering refers to the process of finding groups of points with a data set that are in some way "lumped together". It is also called unsupervised learningunsupervised because we dont know ahead of time where the clusters are located or what they look like

Distance and Similarity Measures

- A distance is any function d(x, y) that takes two points and returns a scalar value that is a measure for how different these points are: the more different, the larger the distance
- Depending on the problem domain, it may make more sense to express the same information in terms of a similarity function s(x, y), which returns a scalar that tells us how similar two points are: the more different they are, the smaller the similarity
- Any distance can be transformed into a similarity and vice versa
- For example if we know that our similarity measure s can take on values only in the range [0, 1], then we can form an equivalent distance by setting d = 1 - s

Distance and Similarity Measures

- In other situations, we might decide to use d $= 1/s$, or s $=$ e^{-d}, and so on; the choice will depend on the problem we are working on

Properties of Distance or Similarity

1. $d(x, y) \geq 0$
2. $d(x, y) = 0$ if and only if $x = y$
3. $d(x, y) = d(y, x)$
4. $d(x, y) + d(y, z) \geq d(x, z)$

- Constructing a symmetric function from asymmetric
  - $d_S(x,y)=\frac{d(x,y)+d(y,x)}{2}$ is always symmetric
- Smoothness: $d(x, y) = 0$ if $x = y$; 1 otherwise

Common Distance and Similarity Measures

- We are looking for ways to measure the distance between any two data points
- Each point is a collection of individual values: $x = \{x_1, x_2, \ldots, x_d\}$, where d is the number of dimensions (or features)
- Example: The data point (0, 1) has two dimensions and describes a point in space; whereas the tuple ['male', 'retired', 'Florida'], which describes a person, has three features

Common Distance and Similarity Measures

- For any given data set containing n elements, we can form n^2 pairs of points
- The set of all distances for all possible pairs of points can be arranged in a quadratic table known as the distance matrix
- The distance matrix embodies all information about the mutual relationships between all points in the data set. If the distance function is symmetric, as is usually the case, then the matrix is also symmetric
- Furthermore, the entries along the main diagonal typically are all 0, since $d(x, x) = 0$ for most well-behaved distance functions

Numerical Data

- If the data is numerical and also "mixable" or vector-like, then the data points bear a strong resemblance to points in space; hence we can use a metric such as the familiar Euclidean distance
- The Euclidean distance is the most commonly used from a large family of related distance measures, which also contains the so-called Manhattan (or taxicab) distance and the maximum (or supremum) distance
- All of these are in fact special cases of a more general Minkowski or p-distance

Numerical Data

Name	Definition
Manhattan	$d(x, y) = \sum_i^d \lvert x_i - y_i \rvert$
Euclidean	$d(x, y) = \sqrt{\sum_i^d (x_i - y_i)^2}$
Maximum	$d(x, y) = \max_i \lvert x_i - y_i \rvert$
Minkowski	$d(x, y) = \left(\sum_i^d \lvert x_i - y_i \rvert^p \right)^{1/p}$
Dot product	$x \cdot y = \frac{\sum_i^d x_i y_i}{\sqrt{\sum_i^d x_i^2} \sqrt{\sum_i^d y_i^2}}$
Correlation coefficient	$\text{corr}(x, y) = \frac{\sum_i^d (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i^d (x_i - \bar{x})^2} \sqrt{\sum_i^d (y_i - \bar{y})^2}}$ $\bar{x} = \frac{1}{d} \sum_i^d x_i \qquad \bar{y} = \frac{1}{d} \sum_i^d y_i$

Distance Matrix Computation in r

- Syntax: dist(x, method = "euclidean", diag = FALSE, upper = FALSE, p = 2)
- This function computes and returns the distance matrix computed by using the specified distance measure to compute the distances between the rows of a data matrix
- Arguments:
  - x - a numeric matrix, data frame or "dist" object.
  - method-the distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski". Any unambiguous substring can be given
  - diag-logical value indicating whether the diagonal of the distance matrix should be printed by print.dist.
  - upper-logical value indicating whether the upper triangle of the distance matrix should be printed by print.dist.
  - p-The power of the Minkowski distance

Categorical Data

- If the data is categorical, then we can count the number of features that do not agree in both data points (i.e., the number of mismatched features); this is the Hamming distance
- In certain data mining problems, the number of features is large, but only relatively few of them will be present for each data point. Moreover, the features may be binary: we care only whether or not they are present, but their values dont matter
- In such situations, where features are not merely categorical but binary and sparse, we may be more interested in matches between features that are On than in matches between features that are Off

Categorical Data

- The Jaccard coefficient s_J is a similarity measure; the corresponding distance function is the Jaccard distance $d_J = 1 - s_J$
  - n_{00} features that are Off in both points
  - n_{10} features that are On in the first point, and Off in the second point
  - n_{01} features that are Off in the first point, and On in the second point
  - n_{11} features that are On in both points
- $s_J = \frac{n_{11}}{(n_{10}+n_{01}+n_{11})}$
- $d_J = \frac{n_{10}+n_{01}}{(n_{10}+n_{01}+n_{11})}$
- There are many other measures of similarity or dissimilarity for categorical data, but the principles are always the same

String Data

- If the data consists of strings, then we can use a form of Hamming distance and count the number of mismatches
- If the strings in the data set are not all of equal length, we can pad the shorter string and count the number of characters added as mismatches
- If we are dealing with many strings that are rather similar to each other, then we can use a more detailed measure of the difference between them-namely the edit or Levenshtein distance
- The Levenshtein distance is the minimum number of single-character operations (insertions, deletions, and substitutions) required to transform one string into the other
- Another approach is to find the length of the longest common subsequence. This metric is often used for gene sequence analysis in computational biology

Note

- The best distance measure to use does not follow automatically from data type; rather, it depends on the semantics of the data-or, more precisely, on the semantics that you care about for your current analysis
- In some cases, a simple metric that only calculates the difference in string length may be perfectly sufficient
- In another case, you might want to use the Hamming distance
- If you really care about the details of otherwise similar strings, the Levenshtein distance is most appropriate

Special Purpose Metrics

- A more abstract measure for the similarity of two points is based on the number of neighbors that the two points have in common; this metric is known as the shared nearest neighbor (SNN) similarity
- To calculate the SNN for two points x and y, you find the k nearest neighbors (using any suitable distance function) for both x and y

Center Seekers Algorithm

- Initial set of clusters randomly chosen
- Iteratively, items are moved among sets of clusters until the desired set is reached
- High degree of similarity among elements in a cluster is obtained
- Given a cluster $K_i=\{t_{i1},t_{i2},,t_{im}\}$, the cluster mean is $m_i = (1/m)(t_{i1} + \ + t_{im})$

K Means

Input:
$D = \{t_1, t_2, ..., t_n\}$ // **Set of elements**
A // **Adjacency matrix showing distance between elements.**
k // **Number of desired clusters.**
Output:
K // **Set of clusters.**
K-Means Algorithm:
assign initial values for means $m_1, m_2, ..., m_k$ **;**
repeat
assign each item t_i **to the cluster which has the closest mean ;**
calculate new mean for each cluster;
until convergence criteria is met;

K Means Example

- Given: $\{2,4,10,12,3,20,30,11,25\}$, $k=2$
- Randomly assign means (seeds): $m_1=3, m_2=4$
- $K_1=\{2,3\}$, $K_2=\{4,10,12,20,30,11,25\}$, $m_1=2.5, m_2=16$
- $K_1=\{2,3,4\}$, $K_2=\{10,12,20,30,11,25\}$, $m_1=3, m_2=18$
- $K_1=\{2,3,4,10\}$, $K_2=\{12,20,30,11,25\}$, $m_1=4.75, m_2=19.6$
- $K_1=\{2,3,4,10,11,12\}$, $K_2=\{20,30,25\}$, $m_1=7, m_2=25$
- Stop as the clusters with these means are the same. (until the centroids do not change)

K Means Algorithm

- The k-means algorithm is surprisingly modest in its resource consumption
- On each iteration, the algorithm evaluates the distance function once for each cluster and each point
- Hence the computational complexity per iteration is O(kn), where k is the number of clusters and n is the number of points in the data set

K Means Algorithm

- The k-means algorithm is nondeterministic: a different choice of starting values may result in a different assignment of points to clusters
- For this reason, it is customary to run the k-means algorithm several times and then compare the results

Advantages and Disadvantages

- Advantages:
  - fast and relatively robust
  - Quick Convergence
  - Easy to augment or extend-To incorporate points with different weights
- Disadvantages:
  - If your data sets consists of very dense and widely separated clusters, then the k-means algorithm can get "stuck" if initially two centroids are assigned to the same cluster: moving one centroid to a different cluster would require a large move
- Variant
  - In fuzzy clustering, we dont assign each point to a single cluster; instead, for each point and each cluster, we determine the probability that the point belongs to that cluster
  - Each point therefore acquires a set of k probabilities or weights

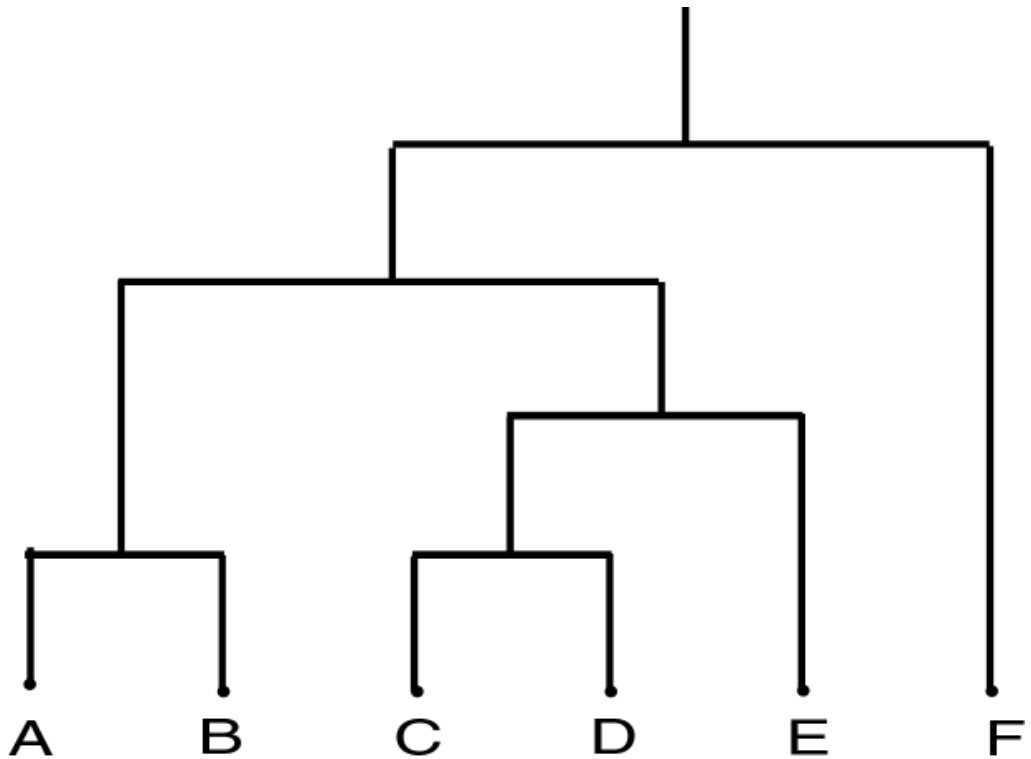
Tree Builders

- Distance between Clusters
  - Single Link: smallest distance between points
  - Complete Link: largest distance between points
  - Average Link: average distance between points
  - Centroid: distance between centroids
- Hierarchical Clustering
  - Clusters are created in levels
  - Actually creating sets of clusters at each level
  - Agglomerative
    - Initially each item in its own cluster
    - Iteratively clusters are merged together
    - Bottom Up

Dendrogram

- Dendrogram: a tree data structure which illustrates hierarchical clustering techniques
- Each level shows clusters for that level
  - Leaf - individual clusters
  - Root - one cluster
- A cluster at level i is the union of its children clusters at level i+1.

Dendrogram



Agglomerative Example

	A	B	C	D	E
A	0	1	2	2	3
B	1	0	2	4	3
C	2	2	0	1	5
D	2	4	1	0	3
E	3	3	5	3	0

Tree Builders

- The result of hierarchical clustering is not actually a set of clusters
- Instead, we obtain a treelike structure that contains the individual data points at the leaf nodes
- This structure can be represented graphically in a dendrogram

Time Complexity

- Tree builders are expensive: we need at least the full distance matrix for all pairs of points (requiring $O(n^2)$ operations to evaluate)
- Building the complete tree takes $O(n)$ iterations: there are n clusters (initially, points) to start with, and at each iteration, the number of clusters is reduced by one because two clusters are combined
- For each iteration, we need to search the distance matrix for the closest pair of clusters-naively implemented, this is an $O(n^2)$ operation that leads to a total complexity of $O(n^3)$ operations
- However, this can be reduced to $O(n^2 \log n)$ by using indexed lookup

Finding Important Attributes

- Dimensionalty Reduction Techniques
  - Different techniques are employed to select the most important variables or features from a multivariate data set in which all variables appear to arise on equal footing
  - In doing so, the dimension of the data set from the original number of variables (or features) is reduced to a smaller set, which (hopefully) captures most of the "interesting" behavior of the data
  - These methods are therefore also known as feature selection or dimensionality reduction techniques

Principal Component Analysis

- Principal component analysis (PCA) is the primary tool for dimensionality reduction in multivariate problems
- It is a foundational technique that finds applications as part of many other, more advanced procedures

Procedure

1. Get some data
2. Subtract the mean
3. Find the covariance matrix
4. Find the Eigen values and vectors
5. Choosing components and forming a feature vector
6. Deriving the new data set:Final Data = Row Feature Vector * RowDataAdjust

Biplots

- Biplots are an interesting way to visualize the results of a principal component analysis
- In a biplot, we plot the data points in a coordinate system spanned by the first two principal components (i.e., those two of the new variables corresponding to the largest eigenvalues)
- In addition, we also plot a representation of the original variables but now projected into the space of the new variables
- The data points are represented by symbols, whereas the directions of the original variables are represented by arrows

Biplots

- In a biplot, we can immediately see the distribution of points when represented through the new variables (and can also look for clusters, outliers, or other interesting features)
- Moreover, we can see how the original variables relate to the first two principal components and to each other: if any of the original variables are approximately aligned with the horizontal (or vertical) axis, then they are approximately aligned with the first (or second) principal component (because in a biplot, the horizonal and vertical axes coincide with the first and second principal components)

Example

- Lets consider a nontrivial example. For a collection of nearly 5,000 wines, almost a dozen physico-chemical properties were measured, and the results of a subjective "quality" or taste test were recorded as well. The properties are: 1 - fixed acidity, 2 - volatile acidity, 3 - citric acid, 4 - residual sugar, 5 - chlorides, 6 - free sulfur dioxide, 7 - total sulfur dioxide, 8 - density, 9 - pH, 10 - sulphates, 11 - alcohol, 12 - quality (score between 0 and 10). Can we find a way to make sense of them and possibly even find out which are most important in determining the overall quality of the wine?

R Code

```
wine <- read.csv( "winequality-white.csv", sep=';',
 header=TRUE )
#.csv file can be downloaded from the UCI data repository
pc <- prcomp( wine )
# prcomp() computes the PCA
plot( pc )
#Eigen values are printed
print(pc)
#Eigen vectors are printed
```

Summary - A Discussion

- First and the Second eigenvector are dominated by the sulphur dioxide concentration
- To understand the data, it is better to print the summary of the data
- Two sulphur-dioxide columns have values in the tens to hundreds, whereas all other columns have values between 0.01 and about 10.0
- Dominate the eigenvalue spectrum simply because they were measured in units that make the numerical values much larger than the other quantities
- If this is the case, then we need to scale the input variables before performing the PCA
- We can achieve this by passing the scale option to the prcomp() command

Summary - A Discussion

- Only grades between 3 and 9 have actually been assigned to quality
- The first quartile is 5 and the third quartile is 6, which means that at least half of all entries in the data set have a quality ranking of either 5 or 6
- In other words, the actual range of qualities is much narrower than we might have expected (given the original description of the data) and is strongly dominated by the center

R Code Contd..

```
pcx <- prcomp( wine, scale=TRUE )
table( wine$quality )
```

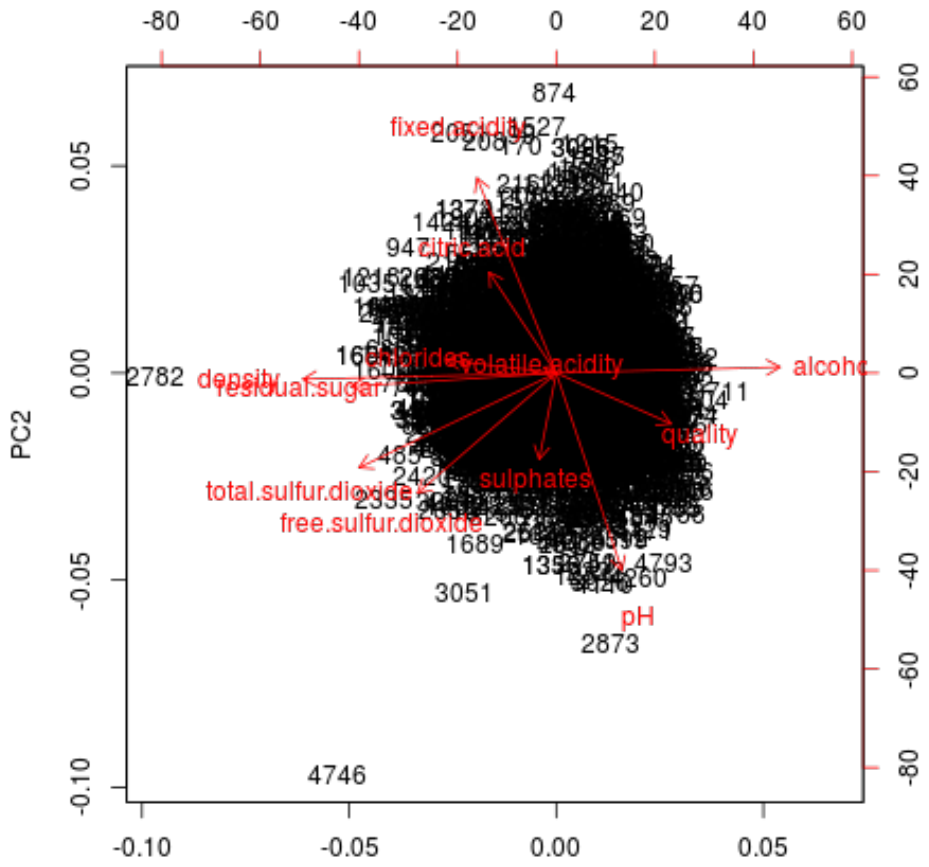
- Middling ranks totally dominate the distribution

```
summary(pcx)
# Eigen values of the scaled version
```

- No single eigenvalue dominates now, and the first 5 (out of 12) eigenvalues account for only 70 percent of the total variance

biplot()

```
biplot( pcx )
```



biplot()- A Discussion

- Three of the original variables-alcohol content, sugar content, and density- are parallel to the first principal component (the horizontal axis)
- Moreover, alcohol content is aligned in the direction opposite to the other two quantities
- Alcohol has a lower density than water, and sugar syrup has a higher density
- So the result of the PCA reminds us that density, sugar concentration, and alcohol content are not independent
- If you change one, the others will change accordingly
- And because these variables are parallel to the first principal component, we can conclude that the overall density of the wine is an important quantity

biplot()-A Discussion

- The next set of variables that we can read off are the fixed acidity, the citric acid concentration, and the pH value
- Again, this makes sense: the pH is a measure of the acidity of a solution (with higher pH values indicating less acidity)
- In other words, these three variables are also at least partially redundant
- The odd one out, then, is the overall sulphur content, which is a combination of sulphur dioxide and sulphate concentration
- Finally, it is interesting to see that the quality seems to be determined primarily by the alcohol content and the acidity
- This suggests that the more alcoholic and the less sour the wine, the more highly it is ranked