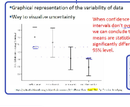
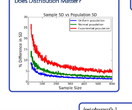
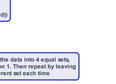
MITx 6.00.2x

- Modelling aspects of the world
  - Deterministic models, e.g., graph theoretic
    - Statistical models:
      - Simulation models: Monte Carlo simulation
      - Curve fitting (linear regression)
      - Machine learning
      - Characterizing accuracy is critical
      - Central limit theorem
      - Empirical rule
  - Models always inaccurate
    - Provide abstractions of reality
  - Random Walks
  - Games of chance
  - Making sense of data
- Deterministic Statistics
  - Organising and summarising data using numbers and graphs (not predicting or inferring)
    - Measures for Variability:
      - Range
      - Variance
        - $variance(X) = \frac{\sum_{x \in X}(x-\mu)^2}{|X|}$
        - Mu is the mean.
        - |X| is the nr of elements in X
      - Standard Deviation
        - $\sigma(X) = \sqrt[1]{\frac{1}{|X|}\sum_{x \in X}(x-\mu)^2}$
        - Standard Deviation (sigma)
          - Standard deviation simply the square root of the variance
          - Standard deviation should always be considered relative to mean
          - (Manual way to calc mean and standard deviation without numpy)
            - def getMeanAndStd(X):
            - mean = sum(X)/float(len(X))
            - tot = 0.0
            - for x in X:
            - tot += (x - mean)**2
            - std = (tot/float(len(X)))**0.5
            - return mean, std
            - sDev = numpy.std(estimates)
      - Coefficient of variation = std / mean
      - Outliers can have a big effect
    - Measures of Central Tendency:
      - Mean
      - Median
      - Mode
- Inferential Statistics (Unit3)
  - - Population: a set of examples
    - Sample: a proper subset of a population
    - Goal: Estimate some statistic about the population based on statistics about the sample
    - Key fact: Only if the sample is random, does it tend to exhibit the same properties as the population from which it is drawn
    - Gambler's Fallacy: For independent events like roulette, it doesn't matter how many successive times it's fallen on red, the probability of getting black in the next round is still 50%
      - Regression to the Mean
        - Following an extreme random event, the next random event is likely to be less extreme
          - E.g. if you had 10 reds in a row, the average of the next 10 rounds should be closer to the expected mean than the previous 10
  - Uses Probability to determine how confident we can be about the conclusion
    - Confidence Intervals
      - Confidence depends on:
        - Size of sample (100 versus 2)
        - Variance of sample (all heads versus 52 heads)
        - As the variance grows, we need larger samples to have the same degree of confidence
      - A Confidence Interval provides a range that is likely to contain the unknown value and a Confidence Level that the unknown value lays within that range
        - Increasing the confidence level will widen the confidence interval
        - Used to determine the Confidence Interval range (e.g. the probability + and - the error)
    - Margin of Error
      - Probability Distribution:
        - Relative Frequency with which a random variable takes on certain values
          - For discrete variable, simply list the probability of each value
            - Histogram (looks like a bar graph without spaces in between bars). Use pylab.hist
            - must add up to 1
          - For Continuous variable, use grouping into ranges
            - Can generate sample data with random.gauss
            - probability density functions (PDFs) See Unit1 Lec7 Exercise 5
              - Uniform (flat line)
              - Exponential
              - Normal Distribution (bell curve). Also called gaussian
                - $P(x) = \frac{1}{\sigma\sqrt{2\pi}} * e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ $\sigma = \sqrt{\sum_{i=0}^{n-1}\frac{1}{n}d_i^2}$
                - Use scipy.integrate.quad. Returns a tuple:
                - Approximation to result
                - Estimate of absolute error
      - Empirical Rule
        - ~68% of data within 1 standard deviation of mean
        - ~95% of data within 2 (1.96) standard deviations of mean
          - E.g. The expected return is 100*mean +/- 100*1.96*std with 95% confidence
        - ~99.7% of data within 3 standard deviations of mean
      - Relies on a random sample exhibiting the same properties as the population
        - E.g. Buffon-Laplace used it to estimate Pi, by dropping needles on a circle, encompassed by a square, and counting how many fell within each area
          - The simulation code does this by calling throw needles with an ever larger number of needles until the standard deviation of the results (errors?) of num trials is no larger than the desired precision divided by 1.96. (CLT and Empiricle Rule)
            - Can be used to estimate integrals, e.g. integral from 0 to Pi for sin(x)
            -
      - Prerequisites
        - Mean estimation error must be zero
        - Distribution of the errors in the estimates is normal (bell curve)
- Monte Carlo Simulations
  - Unit3 lecture 8
    - To change processes described by certain differential equations into an equivalent form, interpretable as a succession of random operations.
      - Was used during the Manhattan Project (war) to predict what would happen during nuclear fission reactions
- Central Limit Theorem (CLT)
  - CLT allows us to use the empirical rule when computing confidence intervals
    - Uses distribution of probability, not the distribution of the sample values
  - Given a sufficiently large sample:
    - 1) The means of the samples in a set of samples (the sample means) will be approximately normally distributed,
    - 2) This normal distribution will have a mean close to the mean of the population, and
    - 3) The variance of the sample means will be close to the variance of the population divided by the sample size.
- Sampling and Standard Error
  - Probability Sampling
    - Simple random sampling: each member of population has an equal chance of being chosen
    - Stratified Sampling (to make sure minority groups get fair representation)
      - Partition population into subgroups (e.g. each faculty at a uni)
        - Then take a simple random sample from each subgroup
    - Can run multiple trials, each with a random sample from a given dataset
      - And plot the Sample Means (should be a bell curve)
        - Then work out the standard deviation and the 95% confidence interval
          - Plot Error bars
            - *Graphical representation of the variability of data
            - *Way to visualize uncertainty
            - 
              - When confidence intervals don't jiggle, we can conclude that means are statistically significantly different at 95% level.
              - pylab.errorbar(xVals, sizeMeans, yerr = 1.96*pylab.array(sizeSDs), fmt = 'o', label = '95% Confidence Interval')
        - Compute standard error of the mean (SEM or SE)
          - $SE = \frac{\sigma}{\sqrt{n}}$
            - Sigma = standard error of the Population
            - n = size of the Sample
            - ...an estimate of the standard deviation of the sample (maybe he meant population?)
            - But we don't have sigma! Our best guess of the standard deviation of the population is the standard deviation of the mean.
              - Once sample reaches a reasonable size, sample standard deviation is a pretty good approximation to population standard deviation
                - Size of Population does NOT matter
                - Distribution shape matters
                - Does Distribution Matter?
                - 
                -
    - To estimate the population Mean from a Single Sample in ahead: (works great when we choose independent random samples)
      - 1) Choose sample size based on estimate of skew in population
      - 2) Chose a random sample from the population
      - 3) Compute the mean and standard deviation of that sample
      - 4) Use the standard deviation of that sample to estimate the SE
      - 5) Use the estimated SE to generate confidence intervals around the sample me
- Curve Fitting
  - E.g. use Linear Regression to fit a curve to data
    - That curve is a model of the data that can be used to predict the value associated with independent values we haven't seen (out of sample data)
      - need to define an Objective Function that provides a quantitative assessment of how well the curve fits the data.
        - finding the best fit can be formulated as finding a curve that minimizes the value of that function.
          - E.g. minimize the sum of the distances from the line to the measured points
            - 
              -
            - Why Y? ...predict the dependent y value for a given independent x.
            - Least Squares Objective Function
              - $\sum_{i=0}^{len(observed)-1}(observed[i]-predicted[i])^2$ = variance * n
            - Linear Regression
              - find the coefficients of a polynomial
                - e.g. ax*2 + bx + c
              - There are many efficient analytical solutions based upon the fact that this objective function is differentiable (similar to Newton's method)
                - pylab.polyfit - Finds coefficients of a polynomial of a specified degree, that provides a best Least Squares fit for the observed data
                - model = pylab.polyfit(xVals, yVals, 1)
                - estYVals = pylab.polyval(model, xVals) equivalent to estYVals = a*xVals + b
              - Can calc Mean Squared Error to compare Relative goodness of the fit of polynomials of different degrees
                - (Sum of all errors) / n
              - But it's better to determine Absolute goodness, using the Coefficient of Determination: R^2
                - $1 - \frac{(\text{variability of errors})^2}{(\text{variability of data})^2}$
                - By comparing the estimation errors (the numerator) with the variability of the original values (the denominator), R2 is intended to capture the proportion of variability in a data set that is accounted for by the statistical model provided by the fit.
                - Always between 0 and 1 when fit generated by a linear regression and tested on training data
                - If R2= 1, the model explains all of the variability in the data.
                - If R2= 0, there is no relationship between the values predicted by the model and the actual data.
                - If R2= 0.5, the model explains half the variability in the data.
  - Mapping from independent values (e.g. measurements to dependent values
- Understanding Experimental Data
  - R-squared used to evaluate model (See Unit 4, problem set 4)
    - $R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - f_i)^2}{\sum_{i=1}^{n}(y_i - mean)^2}$
      - def rSquared(observed, predicted):
      - error = (predicted - observed)**2).sum()
      - meanError = error/len(observed)
      - return 1 - (meanError/numpy.var(observed))
    - Higher not always "better": because of risk of over fitting
  - Choose complexity of model based on (and Reduce risk of over fitting)
    - Theory about structure of data (e.g. F = ma) already predicts linear model)
    - Simplicity
    - Cross-validation
      - If data set is small
        - use Leave-one-out Cross Validation
      - If the data set is large
        - k-fold cross validation or
          - E.g. for k=4, split the data into 4 equal sets, train on 3 and test on 1. Then repeat by leaving out a different set each time
        - repeated-random-sampling validation
          - Split dataset into two (train and test) by random sampling. Repeat (e.g. 100 times). Typically between 20% and 80% for split.
            - See lecture11-segment3.py for 50/50 split
      - E.g. Train with Dataset1 and K2 with Dataset2. The swap and compare
- Machine Learning (unit 5)
  - Traditional Programming
    - Data
    - Program
    - Computer
    - Output
  - Machine Learning
    - Data
    - Output
    - Computer
    - Program
  - All ML Methods require:
    - § Representation of the features
    - § Distance metric for feature vectors
    - § Objective function and constraints
    - § Optimization method for learning the model
    - § Evaluation method
    - Feature Engineering
      - Signal-to-Noise Ratio (SNR) maximize ratio of useful input to irrelevant input
  - Supervised learning
    - - Start with set of feature vector/value pairs
      - Goal: find a model that predicts a value for a previously unseen feature vector
      - Regression models predict a real value (e.g. linear regression)
      - Classification models predict a label (chosen from a finite set of labels)
        - Distance Between Vectors
          - *Minkowski metric
            - $dist(X1,X2,p) = (\sum_{k=1}^{len} abs(X1_k - X2_k)^p)^{1/p}$
            - p = 1: Manhattan Distance
            - P = 2: Euclidean Distance
            - Manhattan uses perpendicular distances
            - Euclidean is hypotenuse (Pythagoras)
          - KNN - K-nearest Neighbors: Using Distance Matrix to classify data, based on their closest neighbours
            - Advantages
            - Learning fast, no explicit training
            - No theory required
            - Easy to explain method and results
            - Disadvantages
            - Memory intensive and predictions can take a long time
            - Are better algorithms than brute force
            - No model to shed light on process that generated data
            - Another method, logistic regression, covered in textbook
            - Scaling different features so they will all have the same weight
              - Z-Scaling: Each feature has a mean of 0 & a standard deviation of 1
                - def zScaleFeatures(vals):
                - """Assumes vals is a sequence of floats"""
                - result = pylab.array(vals)
                - mean = float(sum(result)/len(result)
                - result = result -mean
                - return result/stdDev(result)
              - Interpolation: Map minimum value to 0, maximum value to 1, and linearly interpolate
                - def zScaleFeatures(vals):
                - """Assumes vals is a sequence of floats"""
                - minVal, maxVal= min(vals), max(vals)
                - fit = pylab.polyfit(minVal, maxVal, [0, 1], 1)
                - return pylab.polyval(fit, vals)
  - Unsupervised learning
    - - Start with a set of feature vectors (no values)
      - Goal: uncover some latent structure in the set of feature vectors
      - Clustering is the most common technique:
        - Define some metric that captures how similar one feature vector is to another
        - Group examples based on this metric
          - the answer we get is dictated by the features we choose to use, and the distance metric we use to compare the resultant feature vectors, not, as we saw with supervised learning, by some set of labels that gives us ground truth.
            - Optimization Problem
              - $variability(c) = \sum_{d \in c} distance(mean(c), d)^2$
              - $dissimilarity(C) = \sum_{c \in C} variability(c)$
              - Objective Function
              - c is one cluster
              - C is the collection of all clusters
              - We purposely do not divide by the size of the cluster because variability in large datasets is actually worse than variability in a small set
              - We want to minimize dissimilarity but only up to point (else each data point would form its own cluster on its own)
                - Use constraints, e.g.
                - Minimum distance between clusters
                - Number of clusters
            - K-means Clustering:
              - Constraint: exactly k non-empty clusters
              - Use a greedy algorithm to find an approximation to minimizing objective function
                - randomly chose k examples as initial centroids
                - while true:
                - create k clusters by assigning each example to closest centroid
                - compute k new centroids by averaging examples in each cluster
                - if centroids don't change:
                - break
                - Risk of finding local minimum that is not global minimum
                - Mitigated by running multiple K-means and choosing the one that minimizes the Objective Function
                - Or by playing around with different values of k
                - See K-Means simulation with Gaussians
                - best = kMeans(points)
                - for i in range(numTrials):
                - C = kMeans(points)
                - if dissimilarity(C) < dissimilarity(best):
                - best = C
                - return best
  - Unsupervised learning
    - Manhattan uses perpendicular distances
    - Euclidean is hypotenuse (Pythagoras)
- Cum Hoc Ergo Propter Hoc
  - Latin for "With this, therefore because of this"
  - Humans "wired" to find patterns in information, and like to think causally
    - Establishing Causation
      - Lurking variables (i.e something else affecting both the datasets, making it seem that they correlate with each other when in fact they have no causal relationship with each other)
- Statistical Abuses
  - Moral: Beware of percentages when you don't know the baseline
  - Moral: Context matters. A number means little without context.
  - assumption that by sampling a subset of a population we can infer things about the population as a whole
    - Only true if if random sampling is used
      - Not so easy in the field, where some examples are more convenient to acquire than others. E.g. "survivor bias"
        - Moral: Understand how data was collected, and whether assumptions used in the analysis are satisfied. If not, be wary.
  - Statistics about the data is not the same as the data
    - Use visualization tools to look at the data itself
      - Look carefully at the axes labels and scales
      - Ask whether the things being compared are actually comparable
      - Analysis of bad data can lead to dangerous conclusions.
- Attempt to control for all variables other than the variables of interest (Rarely possible)!
  - Randomized control studies is the gold standard
  - Start with a population
  - Randomly assign members to either
    - Control group
    - Treatment group
  - Deal with two groups identically except with respect to the one thing being evaluated (Very hard to do)
- Graph theory (Lecture 3)
  - Set of nodes (vertices), with properties
  - Set of edges (arcs), each connecting two nodes
    - Models networks based on relationships
  - Can be used to model transition between permutations
    - Undirected (graph) Directed (digraph)
      - Source (parent) and destination (child) nodes
      - Unweighted or weighted
        - DFS can easily be modified to minimize the sum of the weights of the edges, not the number of edges. BFS cannot
  - E.g. Find shortest (weighted) path
    - Uses recursive function similar to Left First, Depth first (similar to Search Tree, but with added prevention of circular paths).
      - Depth First Search (DFS)
      - Breadth-first Search (BFS)
        - Explore all paths with n hops before exploring any path with more than n hops (faster)
  - A tree is a directed graph where each pair of nodes is connected by a single path (e.g. Knapsack Search Tree)
    - A Clique is an unweighted graph where each node connects to ALL other nodes
      - DFS: O(|n~2|)
      - BFS: O(n)
- Stochastic processes (Lecture 5)
  - Predictive nondeterminism
    - An ongoing process where the next state might depend on both the previous states and some random element
      - randomness is present, and variable states are not described by unique values, but rather by probability distributions
  - Link to Probability mindmap
- Random Walks (Lecture 6)
  - E.g. modelling the stock market, diffusion etc.
    - Then use simulation with randomised inputs to calculate probable outcomes
- Optimisation (Lectures 1 and 2)
  - Many optimization problems inherently exponential
    - But dynamic programming often works
    - And memoization generally useful technique
  - Examples: knapsack problems, graph problems, curve fitting, clustering
  - Many problems can be formulated in terms of
    - Objective function (e.g. minimise travel time)
    - Set of constraints (e.g. limit on $spent and arrival time)
  - Knapsack: How many items can you include without exceeding the constraints
    - Find a V that maximizes
      - $\sum_{i=0}^{n} V[i] * I[i] : value$
      - subject to the constraint that
      - $\sum_{i=0}^{n} V[i] * I[i] : weight \leq w$
    - Brute Force Algorithm
      - 1. Enumerate all possible combinations of items (the power set). Size = 2^n
      - 2. Remove all of the combinations whose total units exceeds the allowed weight.
      - 3. From the remaining combinations choose any one whose value is the largest.
        - Not practical. O(2^n)
          - Code for Powerset generator for putting items in 3 bags O(3^n). Using bitwise operators
            - Always an Optimal solution
    - Greedy Algorithm (practical alternative)
      - Left child node: for taking next item
      - Right child node: for not taking next item
        - Left-first, depth-first
          - A Search Tree enumerates possibilities
            - Dynamic Programming
              - Memoization: Stores results during recursion to avoid calculating same thing twice (e.g. Fibonacci)
                - Only works for: Optimal Substructure and Overlapping Subproblems
                - Merge Sort has Opt Substructure but NOT Overlapping subproblems
                - Same issue for Search Tree but can work in spite of having different items in each subproblem
                - See fastMaxVal function
                - Each node = <taken, left, value, remaining-calories>
                -
                - 0,0,,,