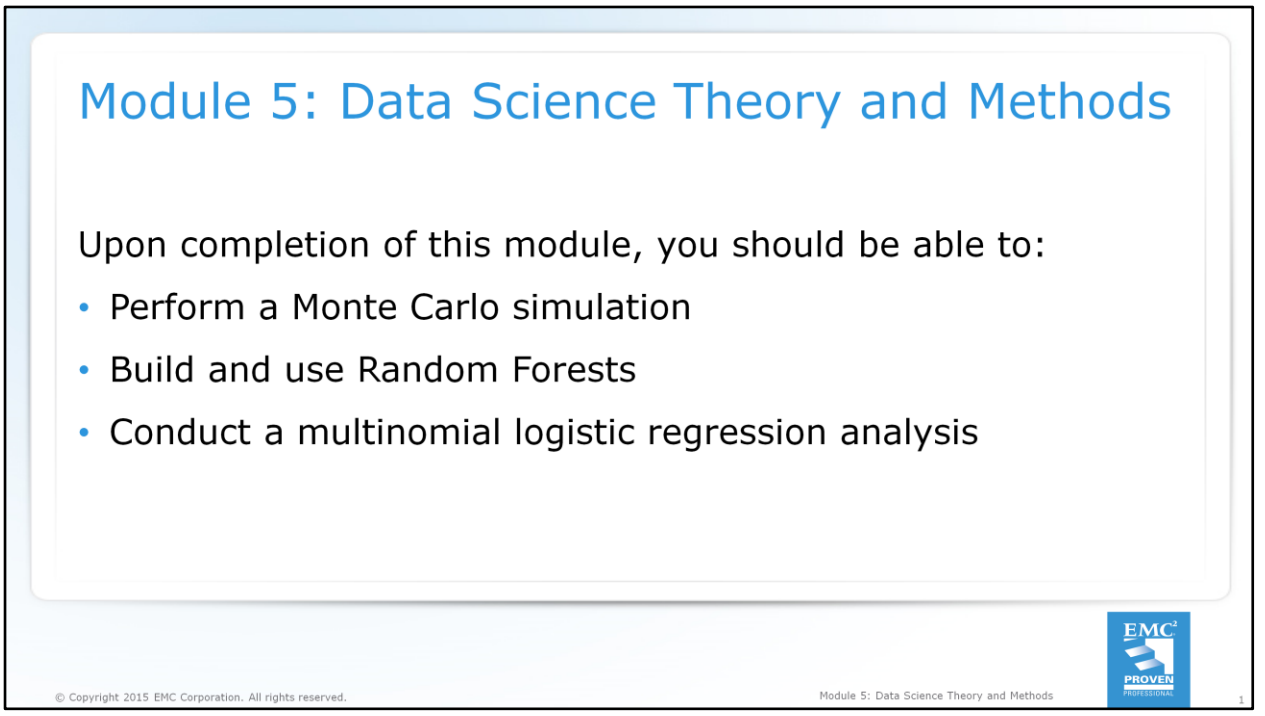
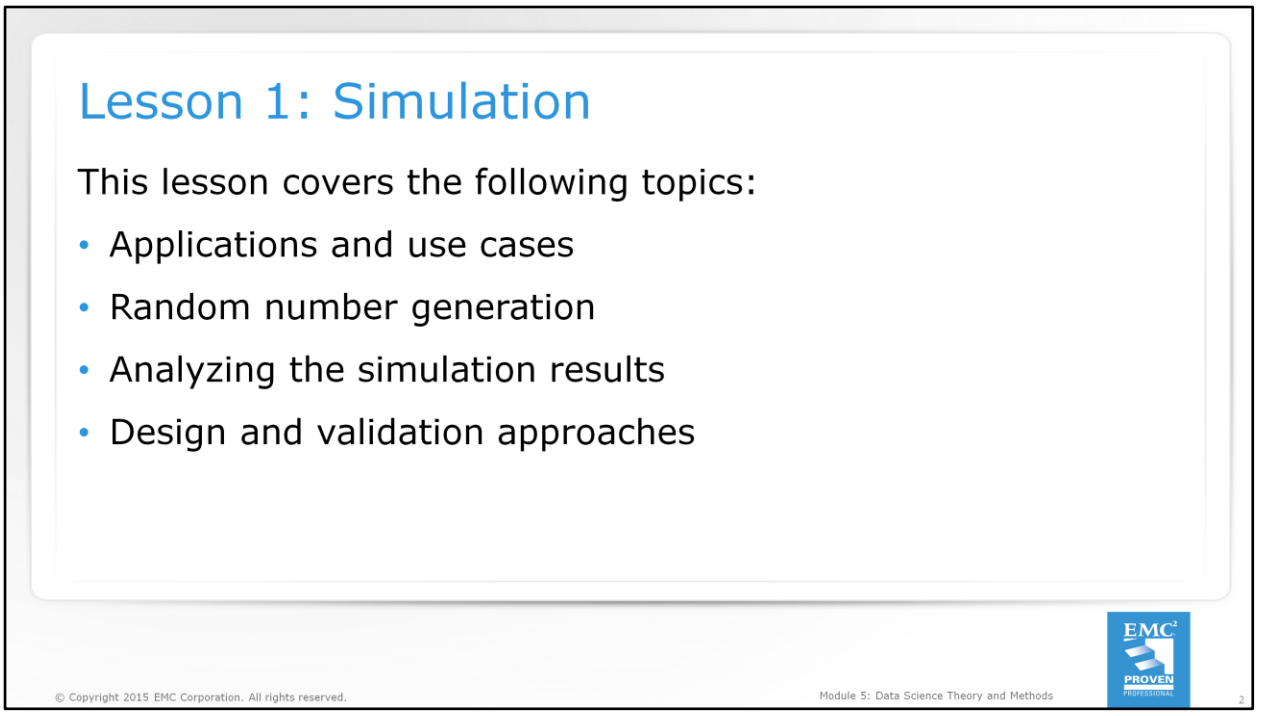


This module focuses on three analytical techniques: Monte Carlo simulation, Random Forests, and multinomial logistic regression. Monte Carlo simulation solves complex probabilistic problems that would be difficult to answer by algebraic calculations. The Random Forests technique is an ensemble method, that combines the results of multiple decision trees. Building on binary logistic regression, multinomial logistic regression addresses classification problems with more than two class labels. Decision trees and logistic regression, with two outcomes, are discussed in the data science and big data analytics associate level course.



This lesson covers various aspects of using simulation to solve complex problems that are often too difficult to answer analytically. After a few examples to illustrate how simulation works, this lesson will examine the yield management problem of optimizing airline ticket sales to maximize overall net revenue.

Overview of Simulation

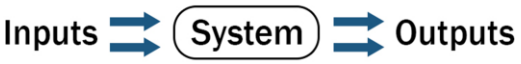
- Mimics reality
- Provides a cost effective and practical alternative
- Typical use cases
  - Scientific
  - Engineering
  - Education
  - Business

In general, simulation is a technique that, by mimicking reality, attempts to provide some insight and experience into how a real life situation may play out. For example, a football team may prepare for a game in an opponent's stadium by playing recorded crowd noise through a series of speakers to mimic the actual crowd noise during a game. Ideally, such a simulation prepares the players to overcome the noise-related difficulties that will be encountered during the actual game.

There are scientific and engineering use cases for simulation. Simulation can be used to predict chemical reactions under stated environmental conditions. Electrical circuits can be modeled to ensure that the design will properly handle in-rush currents and transients.

For training purposes, flight simulators are a cost effective way to train new pilots or to maintain a pilot's skills. For business applications, simulation can be used for optimization purposes to ensure adequate resources and personnel are available for manufacturing and service industries. For example, in a supermarket, there may be multiple cashiers to service multiple customers with varying number of items to be purchased. Determining the question of optimal staffing and answering "what if" scenarios are ideal use cases for simulation.

Monte Carlo Method



- Approach
  - Build a mathematical model to represent the system of interest
  - Generate random inputs to the model
  - Record the model outputs
  - Repeat many times
- Benefits
  - Answers questions about highly complex systems
  - Accounts for the variability in the input variables
  - Provides the distribution of possible outcomes
- Named after the Monte Carlo casino

This lesson will look at an analytical technique known as the Monte Carlo method. By modeling how various inputs to a system will interact and repeatedly running this model with randomly generated input values, it is possible to examine the behavior of the system and obtain not only the expected outcome, but also the distribution of the possible outcomes.

The Monte Carlo method is named after a famous European casino. By observing enough card hands or rolls of the die, the probabilities of the possible outcomes can be approximated with reasonable accuracy. The following simulation example will examine the outcomes of tossing a coin 1,000 times.

Coin Toss Simulation

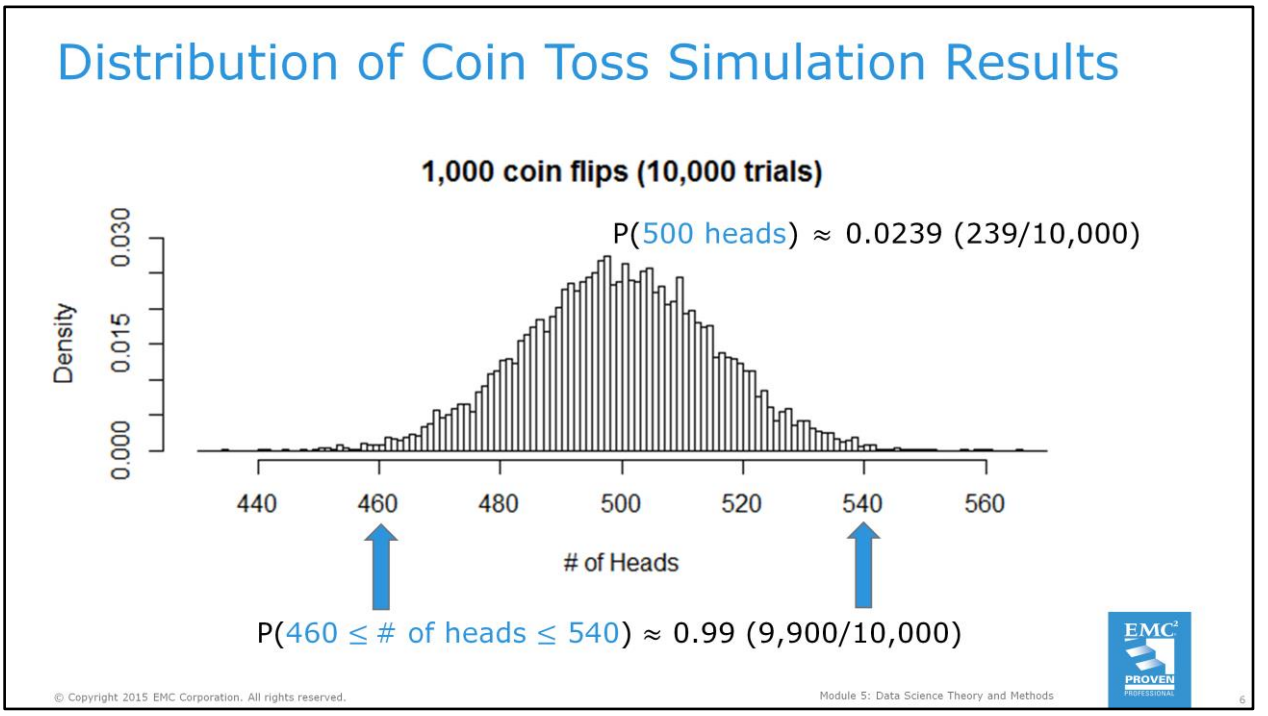
- Flip a coin
  - Possible outcomes are: {head, tail}
  - P(head) = P(tail) = 0.5
- Flip a coin 1,000 times
  - Expect 500 heads and 500 tails
  - How likely is it to get exactly 500 heads?
- Simulate 1,000 coin flips (one trial)
  - Randomly generate a number, r, between 0 and 1
  - If r < 0.5, then the outcome is a head,
    else the outcome is a tail
  - Repeat 1000 times

Run 10,000 trials

The flipping of a fair coin once will result in a head or a tail each with probability of 0.5.

With a little arithmetic, the flipping of a fair coin twice will result in exactly one head and one tail with probability of 0.5.

Intuitively, one might expect that the flipping of fair coin 1,000 times will result in 500 heads and 500 tails, but how likely is that specific outcome? A simulation, as outlined, can be used to approximate an answer.



In the 10,000 trials that were simulated, 500 heads occurred 239 times. So, 0.0239 (239/10,000) is the approximate probability of observing exactly 500 heads and 500 tails when a coin is flipped 1,000 times. The histogram plot shows that the majority of the observed number of heads are between 460 and 540 heads. Based on the performed simulation, there is a 99% chance of observing somewhere between 460 and 540 heads, inclusive, out of a 1,000 coin flips.

It should be noted that for this simple example, it is not necessary to use simulation to obtain the approximate distribution of the number of heads. The binomial distribution, shown below, provides a closed form solution to the distribution of the number of heads observed in a 1,000 coin flips.

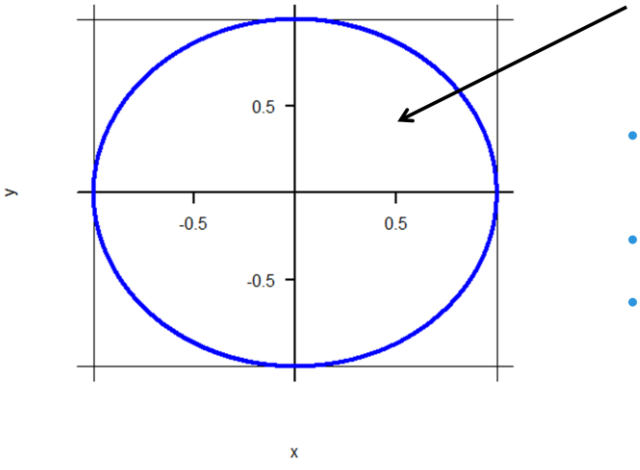
$$P(x \text{ outcomes out of } n \text{ attempts}) = \binom{n}{x} p^x (1-p)^{n-x} \quad \text{where } p \text{ is the probability of a single outcome}$$

In this coin flip example, p = 0.5 and the probability of observing 500 heads out of 1,000 coin flips is 0.0252.

$$\text{P(500 heads out of 1,000 attempts)} = \binom{1000}{500} .5^{500} (1-.5)^{1000-500} = 0.0252.$$

Calculating Pi Using Simulation

Circle with radius = 1 has Area = π

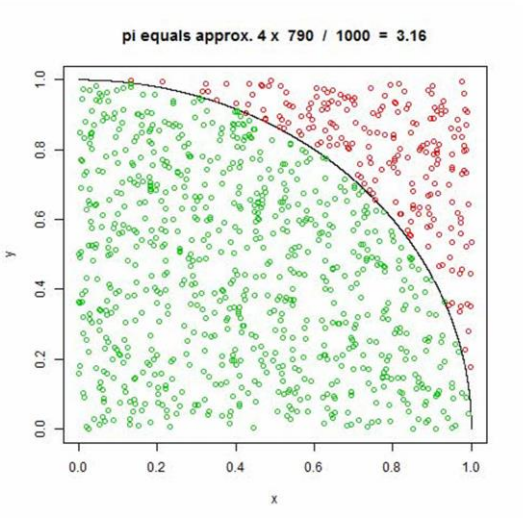


- Randomly select (x,y)
  - 0 < x < 1
  - 0 < y < 1
- Determine if point is within circle
- Repeat n times
- Fraction of points within quarter circle ≈ $\pi/4$

Simulation can be used to evaluate definite integrals (i.e. calculate area under curves). The general approach is to bound the area of interest inside a region such as a rectangle, repeatedly generate random points (x,y), and determine *p*, the proportion of random points that fall in the area of interest. If the rectangle has known area *A*, then the simulation will estimate the area of interest to be, *A*p*.

Since the area of a circle with a radius = 1 has an area of π , the area of a quarter of the circle will be $\pi/4$. This quarter of a circle is bounded by a square with an area of 1. Thus, by randomly generating (x,y) points in the square, an approximate value of $\pi/4$ can be obtained. It is then a simple matter to approximate the value of π.

Simulating Pi



The provided simulation plots the randomly generated points and periodically updates the approximation for π. The green dots denote the simulated points within the quarter circle; the red dots denote the simulated points outside the quarter circle.

A common need in many simulations is to generate a random number between 0 and 1. The next part of this lesson will examine how to generate a sequence of random numbers, and how a random number between 0 and 1 can be used to generate random values that follow a given distribution, such as the discrete uniform or exponential distributions.

Generating a Uniform Random Number in [0,1]

- General algorithm
  1. Obtain a random integer, x
     - From 0 to MaxInt (2^{32} - 1)
  2. Divide x by MaxInt to obtain u in [0,1]
  3. Address the possibility of generating a 0 or 1

$$f(u) = \begin{cases} 1, & u \in [0,1] \\ 0, & otherwise \end{cases}$$

- Algorithm requirements
  - Ability to recreate the same sequence of random numbers
  - Pass statistical randomness tests
  - A long period before the sequence repeats

Since by their nature, computers and associated software are deterministic in nature, computer-generated random numbers are not purely random and thus are referred to as **pseudo-random numbers**. However, the deterministic nature of pseudo-random number generators enable the user to generate the same sequence of random variables by saving the state that generated the sequence. This state is known as the **seed**. For a given seed, the same random numbers can be generated, which may be useful for comparing two different models or implementations. Without the same sequence of the random numbers, the difference in the two simulations may be due to random chance and not the difference in the models or inputs. Also, the ability to reproduce the same random numbers is very useful for debugging simulations.

Although they may be deterministically generated, it is desirable for the sequence of numbers to pass numerous statistical randomness tests. Also, it is not desirable for the sequence of random numbers to quickly repeat itself. The length of a random number sequence before it repeats itself is known as its **period**.

Two pseudo-random generators are: linear congruential generator (LCG) and the Mersenne Twister. Given the ith random number of a sequence, x_i, the next random number generated by an LCG is $x_{i+1} = (ax_i + b) \bmod m$ where a and m are positive integers and b is a non-negative integer. The initial value of the random sequence, x_o, is known as the seed. The Mersenne Twister, the default random generator in R, has a seed that consists of 624 integers and a theoretical period of (2^19937)-1.

For the details on the Mersenne Twister, see the following 1998 paper:
http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/ARTICLES/mt.pdf

Generating a Uniform Random Number in [a,b]

- Algorithm
  1. Generate u' in [0,1]
  2. Calculate u = a+ u'(b-a)

$$f(u) = \begin{cases} \frac{1}{b-a}, & u \in [a,b] \\ 0, & otherwise \end{cases}$$

- An application of the probability integral transformation theorem

  $Let\ X\ have\ a\ continuous\ cdf, F_X(x).$
  $For\ Y = F_X(X), \text{Y is uniformly distributed on } (0,1)$

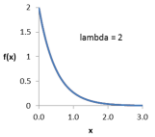
  - Solve $y = F_x(x)$ for x
  - Then for y in (0,1), $x = F_X^{-1}(y)$ has pdf f(x)

Suppose a uniform random number between a and b needs to be generated. For example, suppose the time to perform an oil change on an automobile is expected to take 15 minutes, but has been seen to take anywhere from 10 to 20 minutes with a fairly uniform distribution. How does one generate such a random number between 10 and 20? In this case, the probability integral transformation theorem can be applied. Since the range of a continuous cumulative density function (cdf) is [0,1], applying the inverse cdf to a randomly generated uniform random number from (0,1) will generate a uniform random number in [a,b].

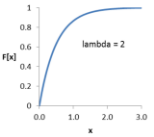
In fact, this transformation can be applied to other distributions with a continuous cdf, such as the exponential distribution.

Exponentially Distributed Random Numbers

- Exponential pdf: $$f(x) = \begin{cases} 0, & x \in (-\infty, 0) \\ \lambda e^{-\lambda x}, & x \in [0, \infty), \lambda > 0 \end{cases}$$



- Exponential cdf: $$y = F[x] = \begin{cases} 0, & x \in (-\infty, 0) \\ 1 - e^{-\lambda x}, & x \in [0, \infty) \end{cases}$$



- Inverse cdf: $$x = F^{-1}[y] = \begin{cases} 0, & y = 0 \\ -\frac{1}{\lambda}\ln(1-y), & y \in (0,1) \end{cases}$$

- For randomly generated uniform $y \in (0,1)$,
  $x = -\frac{1}{\lambda}\ln(1-y)$ is exponentially distributed

As a second example of using the probability integral transformation to generate random numbers based on randomly generated uniform random numbers from (0,1), consider the exponential probability density function (pdf). Its continuous cdf can be easily solved for its inverse function. Thus, by generating random uniform numbers y_1, y_2, … y_n in (0,1), exponentially distributed random variables x_1, x_2, … x_n can be generated.

Generating Discrete Uniform Random Numbers

Example: simulate the roll of a die

- Possible outcomes S = {1,2,3,4,5,6}
- For s ϵ S, P(s) = 1/6

$$F(s) = \begin{cases} 0, & -\infty < s < 1 \\ 1/6, & 1 \le s < 2 \\ 2/6, & 2 \le s < 3 \\ 3/6, & 3 \le s < 4 \\ 4/6, & 4 \le s < 5 \\ 5/6, & 5 \le s < 6 \\ 1, & 6 \le s \le \infty \end{cases} \quad \text{for } y \in (0,1) \Rightarrow \quad s = \begin{cases} 1, & 0 < y < 1/6 \\ 2, & 1/6 \le y < 2/6 \\ 3, & 2/6 \le y < 3/6 \\ 4, & 3/6 \le y < 4/6 \\ 5, & 4/6 \le y < 5/6 \\ 6, & 5/6 \le y < 1 \end{cases}$$

So, far the examples have covered the case where random numbers are generated from a continuous distribution. However, there is often the need to generate random numbers from a binomial, Poisson, or other discrete distribution. In this example, the generation of discrete uniform random numbers to simulate the rolling of a die is examined. The basic approach is to map intervals of the continuous uniform distribution on (0,1) with equal probability of obtaining a particular discrete value.

To simulate the rolling of a die, a continuous uniform random number, y, on (0,1) is generated. For simplicity, the interval (0,1) is divided into 6 subintervals of equal length (1/6) corresponding to the possible rolls of the die {1,2,3,4,5,6}. The roll of the die is determined by which subinterval the value of y belongs.

Now, that the basic mechanics of generating random numbers have been examined, a more complex simulation example will be examined.

Scenario: Airline Yield Management

- An airline sells two types of tickets for a 120 seat flight

Ticket Type	Price	Rebooking Fee	Prob. Of a No Show	Ticket Demand
Full Fare	$400	$0	0.15	$\mu = 50$, $\sigma = 15$
Discount Fare	$150	$75	0.05	$\mu = 100$, $\sigma = 30$

- More tickets are sold than there are seats on a plane
  - To accommodate any no-shows
  - Any bumped customer receives $125
- Objective: maximize net revenue
- Decision: Need to identify the optimal upper bound on:
  - # of discount fare tickets to sell
  - # of tickets (discount and full fare) to sell

Yield management refers to a tiered pricing strategy typically applied to the selling of a limited number of items, such as the number of seats on an airline flight. In this scenario, two types of tickets are sold. Sold within 5 days of the flight's departure, a full fare ticket sells for $400 and allows the customer to choose another flight without any additional rebooking fees and guarantees the customer a seat. These full fare tickets are intended for business travelers who often can not plan trips far in advance.

Sold more than 5 days in advance of a flight's departure, discount fare tickets sell for $150, but the ticket holder can be bumped to another flight at the airline's discretion. For the lower cost, the customer will have to pay a $75 rebooking fee to choose a different flight. Based on this pricing structure, the airline has observed an approximately normal distributed demand for full fare tickets with a mean demand of 50 tickets and a standard deviation of 15 tickets. For the discount fare tickets, the demand is also modeled by a normal distribution with a mean of 100 tickets and standard deviation of 30 tickets.

The airline needs to determine an upper bound on the number of discount tickets and total tickets to sell that will maximize the airline's net revenue after accounting for items such as the $125 penalty for bumping a passenger.

Although this is a somewhat simple scenario to state, an analytical solution is not readily apparent. The random demand as well as the constraint, that no more than 120 seats can be filled on a given flight, make simulation an excellent choice for evaluating such a dynamic system.

This scenario is based on the problem posed in The Science of Decision Making, by Eric V. Denardo. For more detail on the application of yield management in the airline industry see The Evolution of the US Airline Industry: Theory, Strategy and Policy by Eldad Ben-Yose.

Yield Management Simulation

1. Specify upper bounds to evaluate
2. Simulate discount fare ticket demand
3. Simulate full fare ticket demand
4. Adjust demand by upper bounds
5. Simulate # of no shows
6. Determine # of customers to bump
7. Compute revenue
8. Repeat steps 2 through 7 many times (say 1,000,000 times)
9. Compute average net revenue
10. Repeat steps 1 through 9 for other possible upper bounds

These are the basic steps for executing a yield management simulation. The next slide provides a detailed example of a single pass through steps 1 through 7.

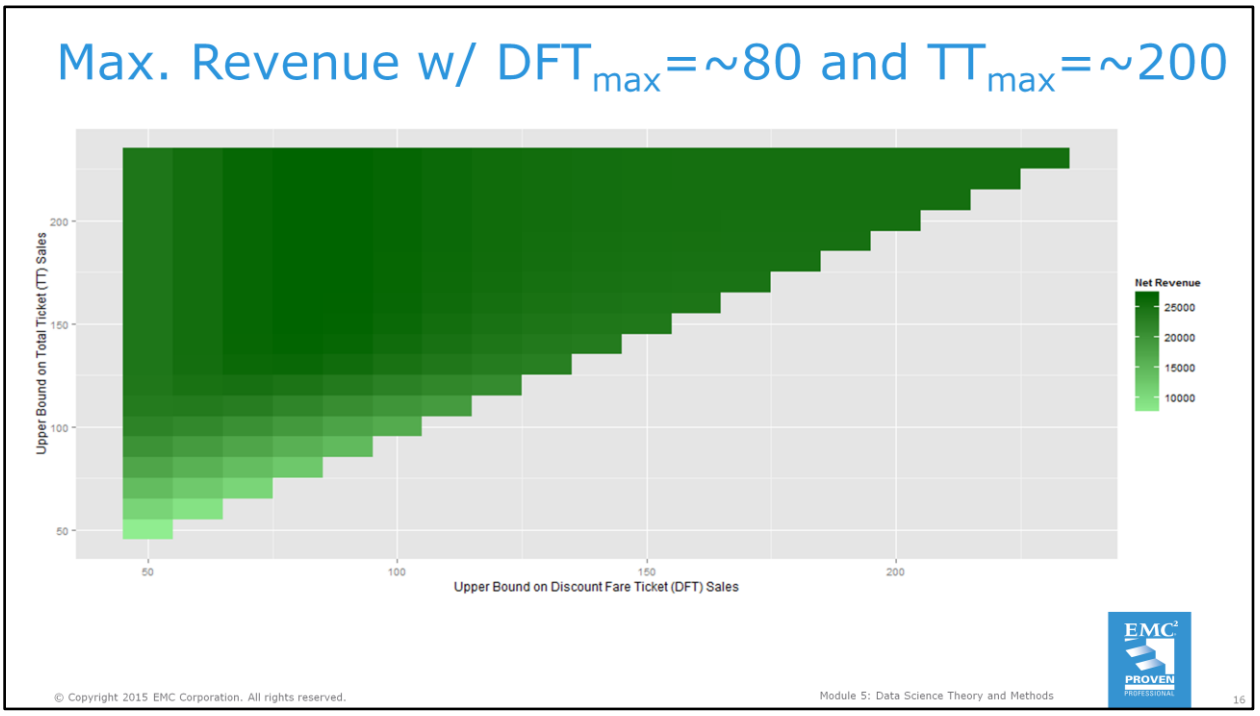
Yield Management Simulation (Single Pass)

1. Specify upper bounds to evaluate $DFT_{max} = 150$ $TT_{max} = 200$
2. Simulate discount fare ticket demand $DFT_{demand} = 170$
3. Simulate full fare ticket demand $FFT_{demand} = 60$
4. Adjust demand by upper bounds $DFT_{sales} = 150$ $FFT_{sales} = 50$
5. Simulate # of no shows $DFT_{ns} = 7$ $FFT_{ns} = 3$
6. Determine # of customers to bump $Cust_{bump} = 143 + 47 - 120 = 70$
7. Compute revenue $= 150*73 + 400*47 + 75*7 - 125*70 = \$21,525$
8. Repeat steps 2 through 7 many times (say 1,000,000 times)
9. Compute average net revenue
10. Repeat steps 1 through 9 for other possible upper bounds

In step 1 of a single pass through the simulation model, the upper bounds on the number of discount fare tickets (DFT_{max}) and the total tickets (TT_{max}) were set to 150 and 200 tickets, respectively. From steps 2 and 3, using the specified normal distributions, the simulated discount fare (DFT_{demand}) and full fare ticket (FFT_{demand}) demand is 170 and 60, respectively.

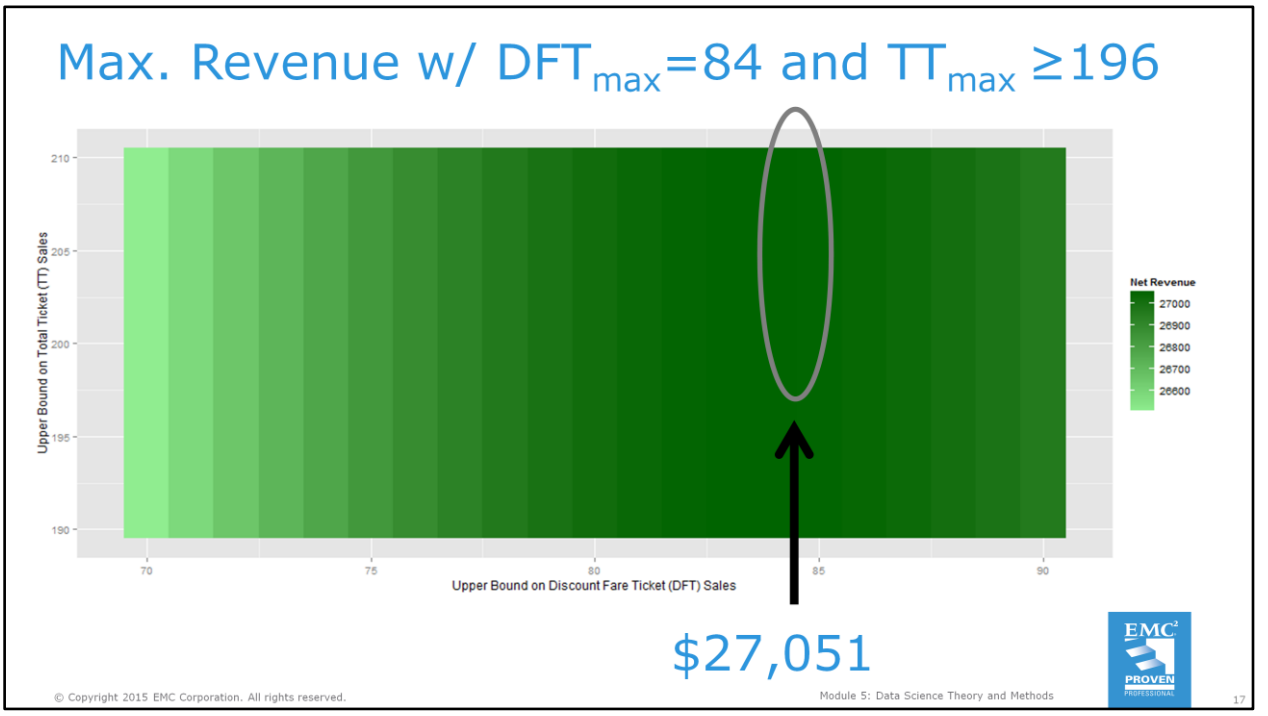
In step 4, the simulated ticket sales are determined by comparing the simulated demand numbers against the upper bounds specified in step 1. So, in this pass, the number of discount fare (DFT_{sold}) and full fare (FFT_{sold}) tickets sold are 150 and 50, respectively. In step 5, the number of no shows are simulated using the binomial distribution. In this pass, the number of discount fare (DFT_{ns}) and full fare (FFT_{ns}) no shows are 7 and 3, respectively.

In step 6, based on the 190 ticket holders who did show for the 120 seat flight, the number of customers to bump is determined to be 70 customers. In step 7, the revenue for this simulated flight is based on the 73 discount fare and 47 full fare ticket holders seated, the 7 discount fare ticket holders who will pay a rebooking fee of $75, and the 70 bumped customers who will receive $125 credit.



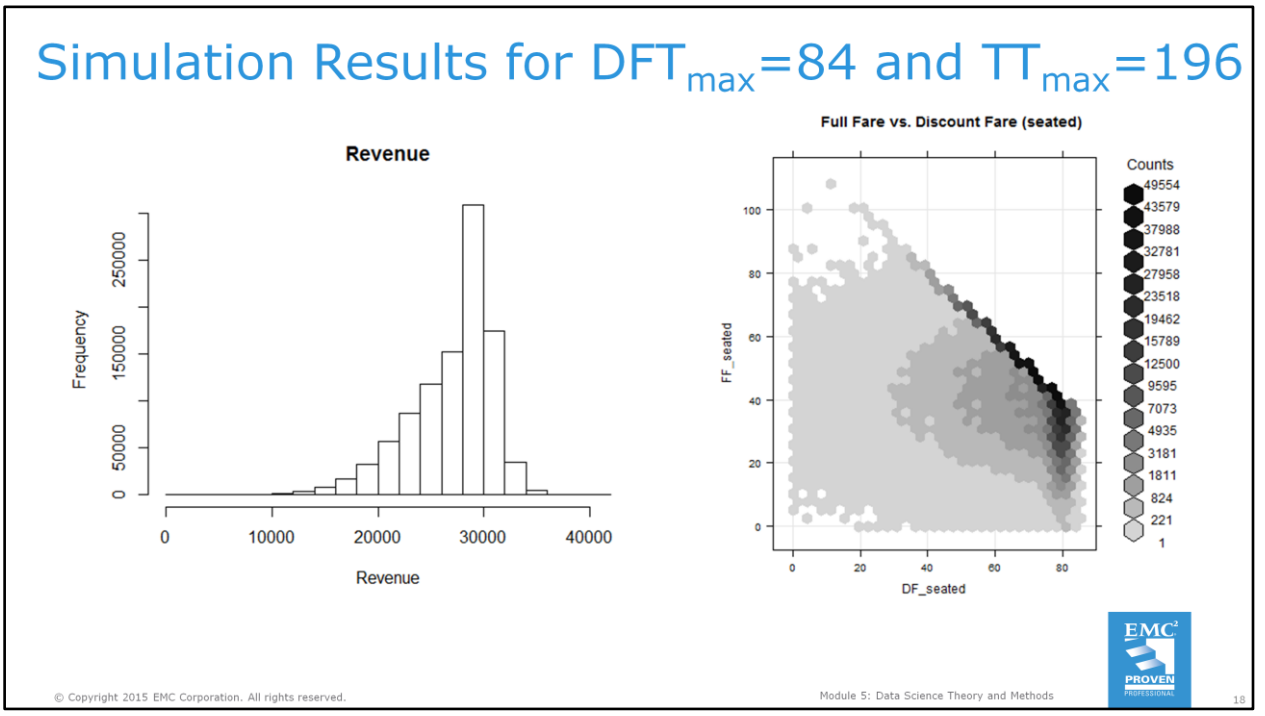
This heat map shows the average net revenue that resulted from the simulation of several scenarios. In this case, possible values of each upper bound was considered from 50 to 230 in steps of 10 tickets. To save computation time, the impractical cases of selling more discount fare tickets than total tickets were ignored. Thus, the upper diagonal of the heat map is the only region of any importance.

The darker green sections of the heat plot indicates that the maximum overall revenue will be obtained by setting the upper bounds for the discount fare and total tickets sold in the neighborhood of 80 and 200, respectively. Since the possible upper bound values assume integer values, the next step is to rerun the simulation in this neighborhood for a step size of 1.



A series of simulations was generated to evaluate the following combinations: upper bound of discount fare tickets from 70 to 90, inclusive and the upper bound of total ticket sales from 190 to 210, inclusive. The simulation results indicate that upper bounds of 84 on the discount fare ticket sales and 196 or greater on the full fare ticket sales will achieve the greatest average net revenue of $27,051.

Additionally, the results reveal that the upper bound on the total tickets sold is somewhat flexible. The reason for this flexibility is that the full fare ticket demand and associated seats filled never reaches a large enough value to have any impact on the net revenue. So, this may present an opportunity to introduce some new constraints to the simulation model in order to reduce the customer satisfaction impact of bumping customers. Also, the new constraint may help identify more locally optimal upper bounds. As the heat map indicates, the nearly maximal revenue is obtained with DFT_{max} between 83 and 86.

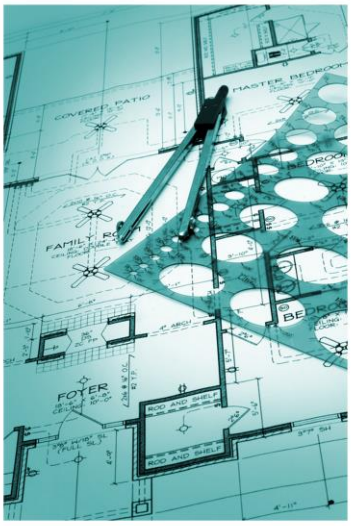


A simulation can often provide additional insights behind the optimal solution. Although the expected net revenue per flight is $27,051, a simple histogram can provide the distribution of the simulated revenues. Similar to the coin flip example, the actual net revenue for any particular flight is likely to be anywhere from $10,000 to $34,000. With a skewed distribution, an analyst may be tempted to use the median rather than the mean as the metric to optimize. However, using the median may not be the best choice for the following reason: With the mean of $27,051 over 1,000 flights, the expected net revenue will be $27.051 million dollars, but the median, in this case $28,255, can not be used in the same way to calculate the total revenue. After all, the mean net income is not really the value of interest, it is just a proxy for maximizing total net revenue over all future flights.

A hexbin plot provides the concentrations for combinations of seated full fare and discount ticket holders. Since the flight is limited to 120 seats, there is a high frequency of combinations along the line that fully fills the plane. Also, there is a high concentration of flights with 80 seats filled with discount fare ticketholders. This may seem to be a problem with the simulation where $DFT_{max}=84$, but is actually expected when the 5% no shows is taken into account. This example helps to identify the biggest issue with building and executing a simulation model: how to tell if the model is performing as intended? The remainder of this lesson will examine the such questions as well as simulation modeling considerations in general.

Simulation Design Considerations

- Clearly define the business problem
  - Identify the decision points
  - State the underlying assumptions
  - Validate with an initial simulation model
- Ensure the model inputs align with reality
- Apply the principle of parsimony
- Reuse simulated inputs and/or seeds
- Rare event modeling
  - Understand the random number generator
  - Run enough trials



As is the case with most analyses, it is essential to clearly understand the business problem or situation that the intended simulation will address. Often times, a simple question will be asked about whether something will work or not, such as: Is having two bank tellers working during the lunch hour sufficient to handle the customer volume? However, the real question may be how many bank tellers should be on the counter during the lunch hour and what are the tradeoffs. It is important to understand the possible decisions or choices that can be made to affect the outcomes. Running an initial simulation model and comparing the results with the business owners can help refine the problem statement as well as flush out the required data to build the model inputs.

The principle of parsimony should be applied to the simulation model design; a simpler model is better than an complex model as long as both models address the same reality. With a simpler model, it will be easier to develop, debug, validate, and explain to business users.

A common practice is to reuse the simulated inputs and/or seeds used to generate the random numbers. In addition to saving some processing time in running multiple simulations, the reuse of the simulated inputs ensure that direct comparisons between the alternatives can be made. Otherwise, in the airplane yield management simulation, for example, marginal differences in the average net revenue could be observed, not due to the change in choice of the upper bounds, but due to different random numbers that were generated in each case. Since the choice of the upper bounds had no effect on the ticket demand, the full fare and discount ticket demand for a million flights was generated and used for each combination of the two upper bounds.

If the purpose of the simulation is to model the effect of rare events which occur with very small probabilities, it is necessary to ensure that the utilized uniform random number generator, for example, will provide numbers close enough to 0 or 1 to generate the occurrence of such rare events. For example, a disk drive may have a nonrecoverable read error specification of 1 per 10E14 bits read. However, a 32 bit system will only result in about 4.3x10E9 possible random numbers from 0 to 2^32. Determining the number of trials to run will be discussed later in this lesson.

Debugging and Validation

- Test pieces of the simulation
- Test for the invalid situations
- Run with fixed input values
  - Typical input values
  - Extreme input values
  - Compare results to hand calculations

When validating the simulation, it is useful to test functionally independent pieces of the implemented model. Once the user is sure one part of the model is working as intended, the next piece of functionality or complexity can be introduced or tested. Also, specific test cases should be established to validate the model.

The model should account for any invalid situations or inputs such as an airplane flight holding more than 120 passengers or the ticket demand being less than zero. The latter is possible when a normal distribution is use to model the possible input value.

For a set of fixed input values, the outcomes should be hand calculated and then compared to the result of the simulation model run against those inputs. In testing the airplane yield management simulation, the two standard deviations for the demand normal distributions were set to zero in order to obtain the expected demand numbers. Also, testing the model against the extremes is useful to ensure the outcome moves in the direction that the model is suppose to predict. For example, if the time to service a customer in a bank teller simulation was fixed at a large number, say 10 years, then the model should show that queue will simply get longer and longer while each teller attends to a single customer.

How Many Simulation Trials to Run?

- Estimate the mean outcome within a specified error, ε
  - Consider the $(1-\alpha)100\%$ confidence interval for μ

$$\bar{x} \pm z_{\alpha/2} * \sigma/\sqrt{n}$$

  - Need to determine n, the sample size, such that $z_{\alpha/2} * \sigma/\sqrt{n}$ equals ε

$$\mathrm{n} = \left(\frac{z_{\alpha/2} * \sigma}{\varepsilon}\right)^2$$

- Example: Estimating the average revenue per flight
  - For $\varepsilon = \$10$, $\sigma = \$5000$, and $z_{\alpha/2} = 1.96$ (95% CI),

$$\mathrm{n} = 960{,}400 \text{ flights}$$

  - For $\varepsilon = \$1$, what would n need to be?

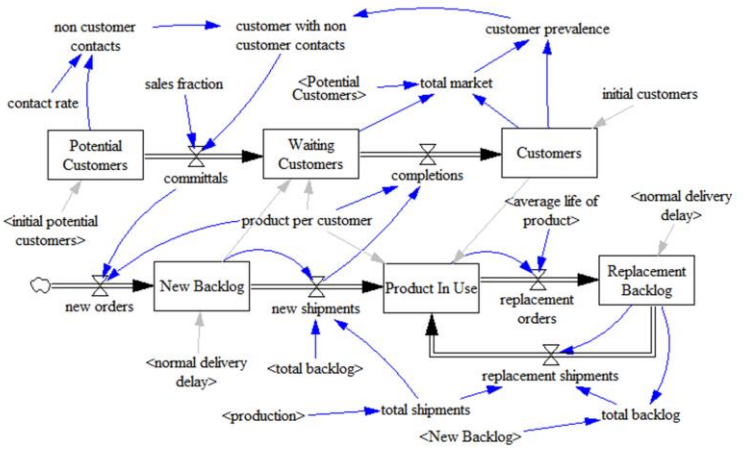
Determining the number of simulation trials to run is a similar problem to determining the appropriate sample size for a hypothesis test or a confidence interval. Suppose the objective of a simulation is to estimate the expected (mean) outcome within a specified error, ε. Thus, using the confidence interval for μ, the sample size is the value of n such that $z_{\alpha/2} * \sigma/\sqrt{n}$ equals ε.

As in the earlier airline yield management problem, suppose the average revenue per flight is to be estimated within +/- $10 with 95% confidence. With an estimate for the revenue standard deviation of $5,000, based on previous flight data, the number of trials is determined to be 960,400 flights.

Of course, increasing the number of trials only addresses the sampling error introduced by the simulation. Any significant deviation from the assumed distributions for the input variables will result in considerable difference between the simulation results and what is actually observed.

Simulation Tools

- Code your own model
  - R
  - SimPy w/ Python
- GUI front ends
  - Vensim
  - SAS
- Recommendations:
  - Understand the built-in random generators
  - For models with many interactions and dependencies, use a GUI version



When a simulation model is to be built and run, the analyst has to make the decision between building the simulation via code or using a simulation tool with a graphical user interface (GUI) to model the system of interest. Building the simulation via code often allows the user greater control and flexibility, but also requires more work from the analyst. R is ideal for simulations with minimal interaction between the simulated observations.

If there is a high degree of interaction between the modeled processes and simulated discrete events, a tool, such as SimPy, provides the framework and functionality to handle the various dependencies. For a more complex model, it is usually beneficial to utilize a simulation tool with a GUI, such as Vensim or SAS Simulation Studio. The provided Vensim illustration models the dynamic system related to a new product introduction, and the interactions and transitions between customers, the ordering process, and the fulfillment process.

Regardless of the chosen tool, ensure that the possible random number generators are well understood. Also, for complex simulation models, a simulation tool with a GUI will save development time as well as debugging time.

Reasons to Choose and Cautions

Reasons to Choose(+)

- A closed-form solution is difficult to obtain
- The input variables are numerous and random
- Quick validation of solutions to simpler problems
- Evaluate "What if" scenarios

Cautions(-)

- Model validation is difficult
- The inputs need to be well-understood
- Reuse seeds/random numbers when performing comparisons

As described earlier, simulation is an excellent choice for solving complex problems with multiple random inputs, for which a closed-form solution may not exist or would be difficult to obtain. Even when a closed-form solution can be obtained, simulation can also be used to validate the proposed solution. Also, applying simulation modeling techniques is an excellent way to perform "what if" analyses on changes to the system behavior as well as changes to the inputs.

Often times, the simulation modeling validation can be difficult. The analyst should attempt to evaluate pieces of the implemented simulation model to ensure the model behaves as intended. In addition to the model being properly defined, it is important for the distributions of the random inputs to be well-defined. Like many things, the quality of the output will only be as good as the quality of the inputs.

When simulation is intended to compare different operational models, it is important to properly reuse the seeds for the random number generators and/or reuse the same randomly generated inputs to the model. Otherwise, observed differences may be due to random chance.

Lab Introduction: Discrete Event Simulation

This lab covers:

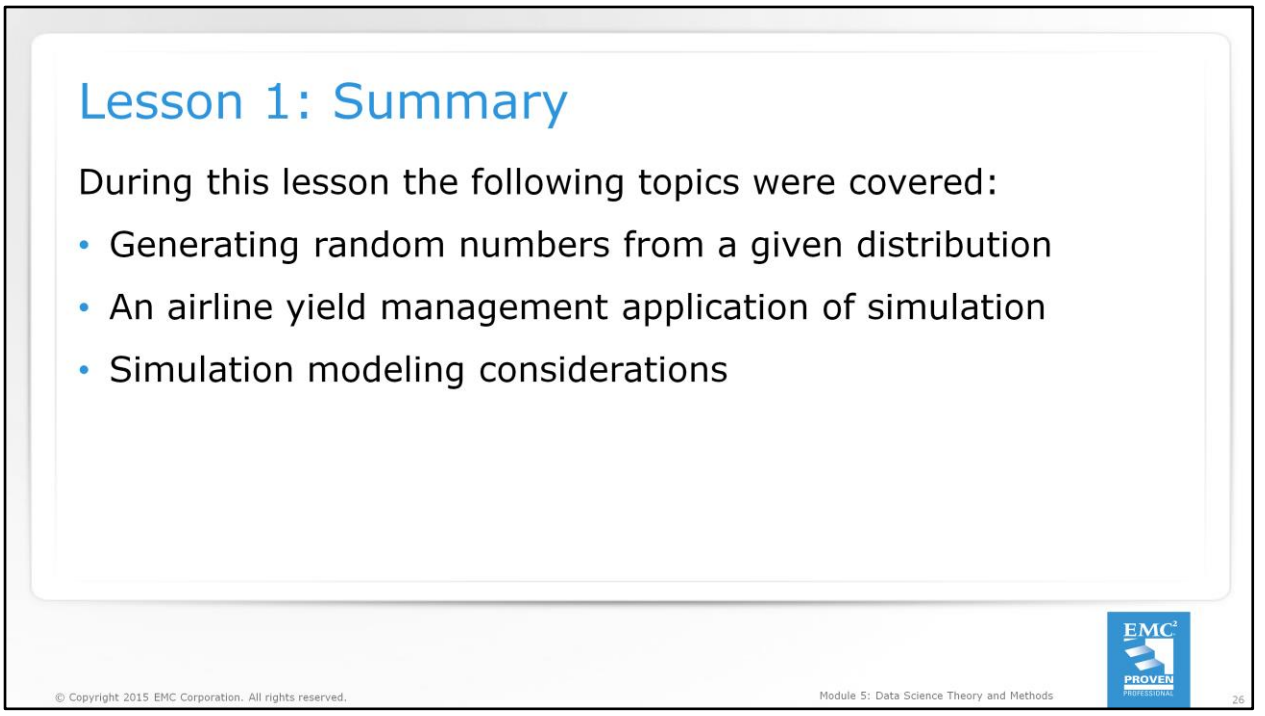
- Key R functionality for random number generation
- Simulation of a single server system with a queue
  - Exponentially distributed time between arrivals
  - Uniformly distributed time to service
  - Explores various system attributes:
    - Queue length
    - Wait times
    - Distribution of the customers time in the system

After addressing some of the basic simulation functionality in R, this lab guides the student in building a simulation model to address the discrete events involved in a single server system with a queue such as may be experienced with a bank's ATM.

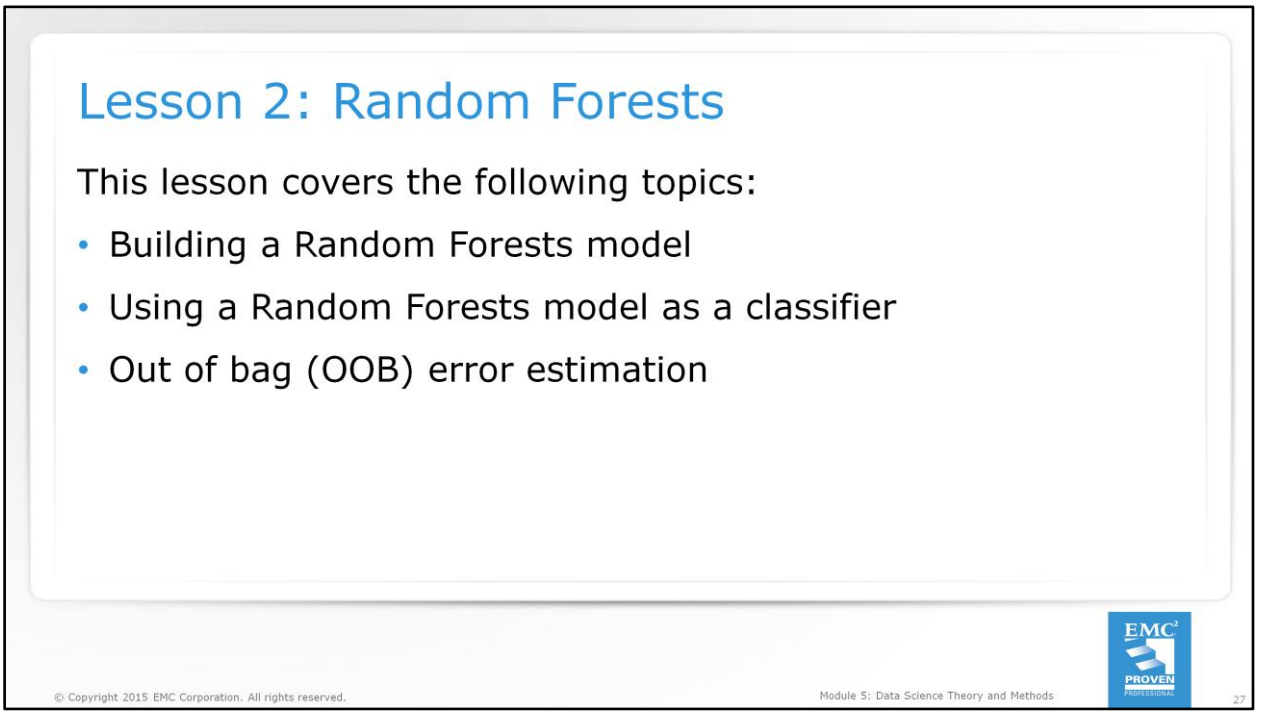
Check Your Knowledge

- Name some advantages of simulation.
- What are key aspects of Monte Carlo simulation?
- Explain the probability integral transformation theorem.
- Name several simulation design considerations.

Write your answers here.

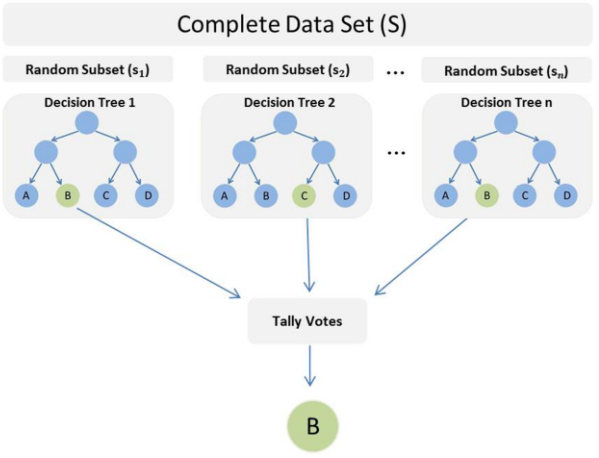


This lesson covered techniques to generate random numbers from a given distribution such as the uniform and exponential distributions. After presenting a few basic examples of the application of simulation, a simulation solution was presented for an airline yield management optimization problem. These examples, as well as the lab exercise, illustrated several simulation implementation techniques such as initializing the seed and reusing the random numbers across various scenarios.



Decision trees were introduced in the associate level Data Science and Big Data Analytics course. A Random Forests model uses randomization techniques to create an ensemble learner consisting of multiple decision trees. This approach is highly accurate and computationally efficient. This lesson describes the process to build and use a Random Forests model as well as how to evaluate the accuracy of the model. In the lab exercise, both R and Apache Mahout are used to train and test a Random Forests Model.

Overview of Random Forests

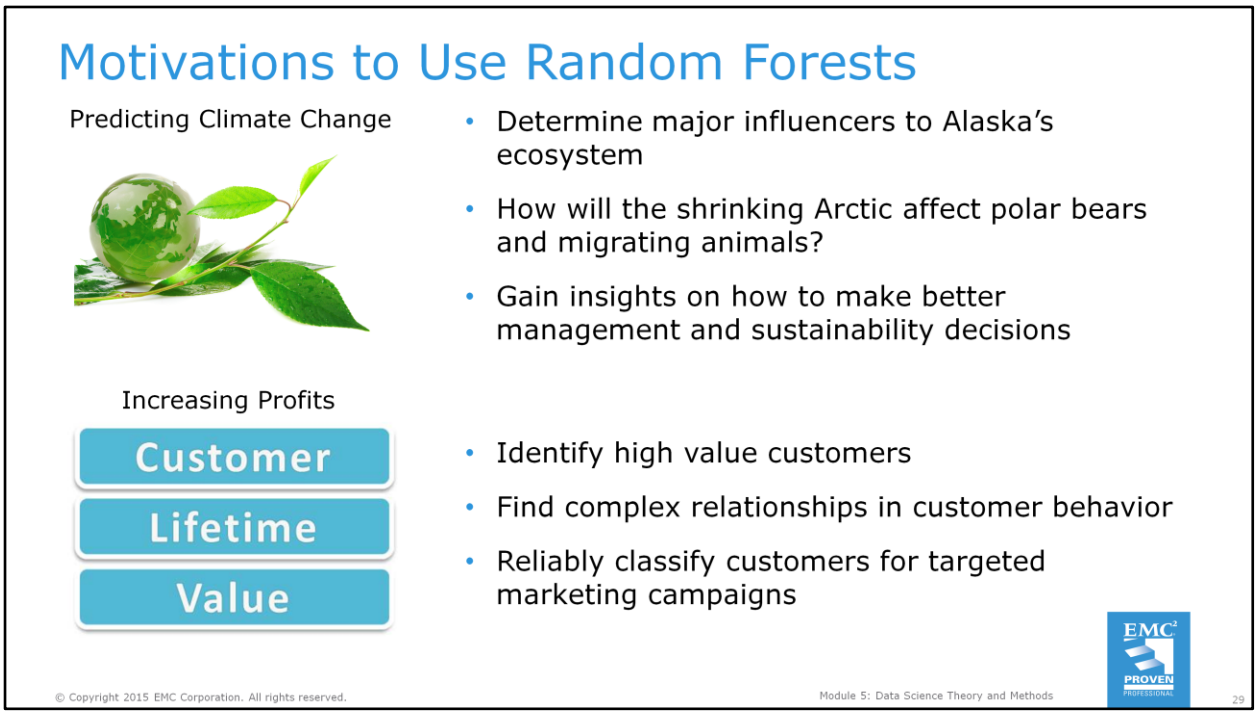


- Consists of n decision trees
  - Each built on a random subset of the data
  - Each subset of the data is obtained by sampling with replacement
- For any given input vector of features, each tree assigns a class label
- Final class label is determined by a majority vote

Random Forests are ensembles of decision trees that leverage the power of a randomized strategy that results in a strong learner. The term "Random Forests" is trademarked by its inventors Leo Breiman and Adele Cutler.

For a given dataset with N observations, each decision tree in the forest is built or trained using a random sample with replacement of size N. This process is referred to as bootstrap aggregation or **bagging**, for short. Any observations not included in an individual sample is called **out of bag** (**OOB**) data. For fairly large N, one would expect about 37% (e^{-1}) of the observations to be OOB data. In general, a Random Forests model does not require cross-validation or a separate testing data set to determine an estimate of error, this functionality is part of the Random Forests algorithm.

Further randomization is introduced in the building of each decision tree by choosing only a small subset of the available features at each split in the tree. In the building of the forest, each tree is grown to its fullest extent, no pruning is necessary. The result is a forest of individual weak learners that, when combined, creates a strong learner.



The U.S. Fish and Wildlife Services sponsored research to investigate and analyze data on more than 400 species of animals, thousands of plant species, and diverse landscape biomes (arctic tundra, coastal tundra plains, mountain and alpine areas), deciduous forests, arboreal forests, coastal rainforests, and the interior. The research group was tasked with forecasting how climate change, human activities, and cataclysmic events might effect Alaska's ecosystem over the next 100 years. The study's principal investigators chose Random Forests because:

"Random Forests is extremely well-suited to handle data based on GIS, spatial and temporal data. It delivers the high degree of accuracy and generalization required for our study, something other solutions couldn't achieve because of the immense size of the database and the statistical interactions involved. It achieves this with amazing speed."

In other situations, businesses seek to identify and retain those customers that are of high value. This activity can be challenging due to complex interactions of predictors or features that are used in the analysis. Random Forests provides an approach that is resilient to complex non-linear interactions. Businesses leverage Random Forests to help classify customers into groups such as buying, defection, or profitability and then act on those group appropriately.

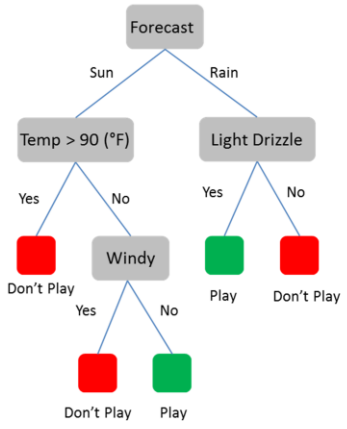
References:

Murphy K., Huettmann F., Fresco N. and Morton J. (2010). Connecting Alaska Landscapes into the Future - results from an interagency climate modeling, land management and conservation project. *SNAP connectivity Project, Final Report.* August 2010.

Predicting Climate Change: https://www.snap.uaf.edu/projects/landscape-connectivity

A Quick Review of Decision Trees

Based on the weather forecast should I play golf today?



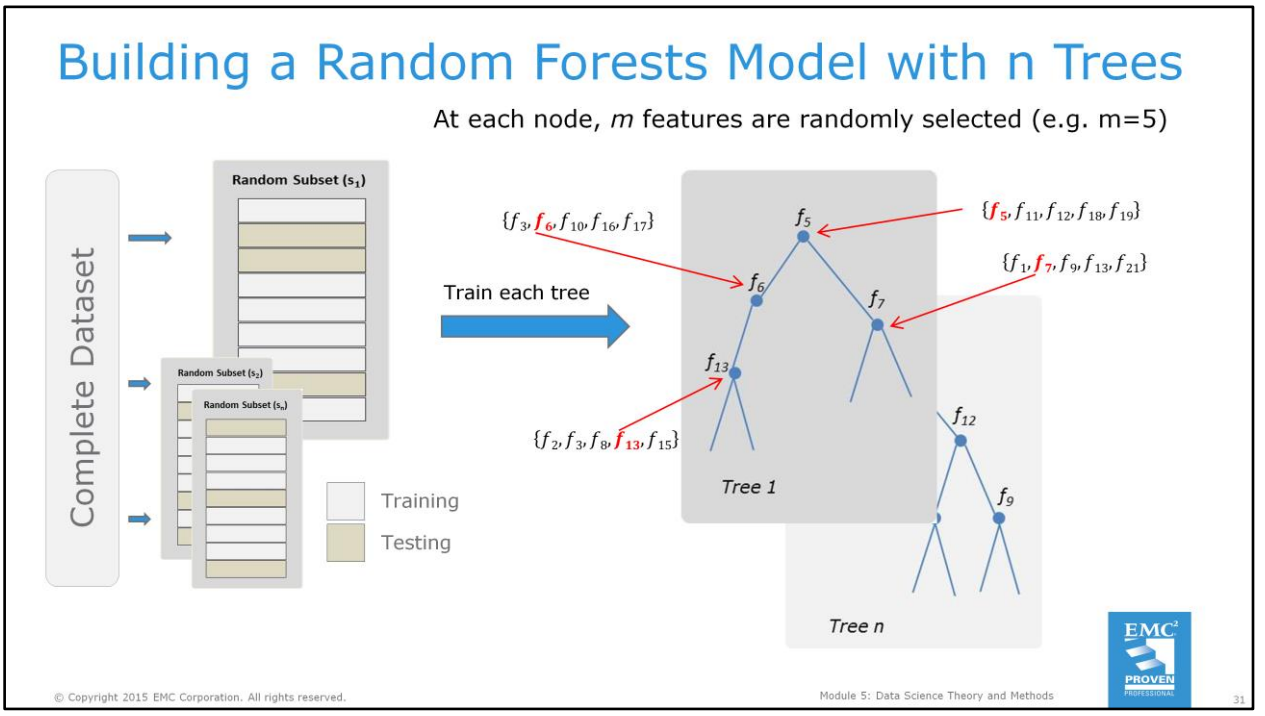
- Handles continuous and discrete data
- Implements feature selection
  - Takes most informative feature
  - Continues to build the tree, until adding more features yields minimal improvement

A decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while, at the same time, an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Forecast in the example) has two or more branches (e.g., Sun, Rain). A leaf node (e.g., Play in the example) represents a classification or decision. The topmost decision node in a tree that corresponds to the most influential predictor is called the root node. Decision trees can handle both categorical and numerical data.

The depicted tree predicts two scenarios where someone would play golf:

Scenario 1: Forecast (Sunny) –> Temp ≤ 90 (°F) –> Not Windy

Scenario 2: Forecast (Rain) –> Light Drizzle



Random Forests modeling involves two randomization approaches to help reduce the variance or the uncertainty in the model's output.

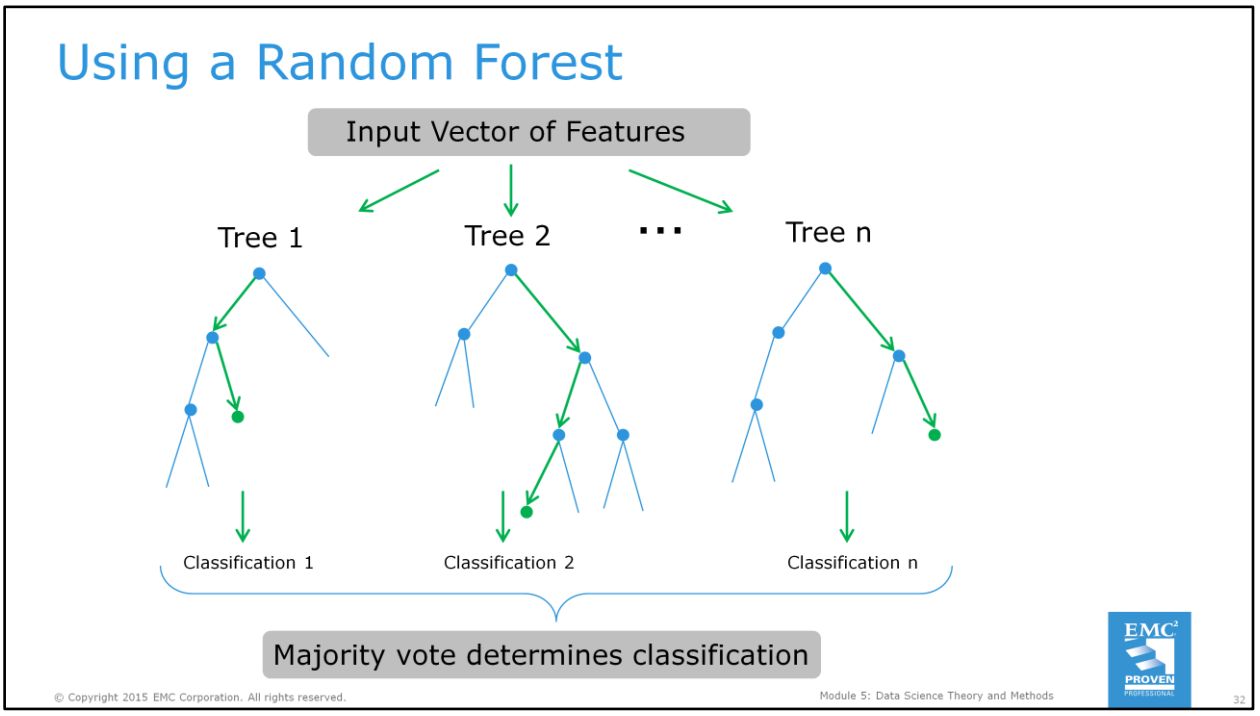
- Bootstrap Aggregation (bagging): Each tree is grown with a random sample (with replacement) of data from the complete data set. The predicted outcome from each individual tree may have a high variance, but averaging or pooling the results of many trees will reduce the variance.

- Random Subspaces of the Features - At each node in the decision tree, only a random subset of features are used to determine the next split. In bagging, the variance will be proportional to the pair-wise correlation between the decision trees. By using only a subset of the available features for each split of a tree, the correlation between the trees will be reduced. Thus, a further reduction in the variance of the predicted outcome is achieved.

The Random Forests algorithm to build n trees is as follows:

1. Draw n bootstrap samples from the original data.

2. For each of the bootstrap samples, grow a decision tree (unpruned) making the following modification: at each split, rather than choosing the best split among all predictors, randomly select m of the predictors and choose the best split from among those variables.



When operationalized, a Random Forests model will process an input vector of features through each tree in the forest. So, for n trees in the forest, there will be n outputs for a given input vector. For classification trees, the final classification will be based on the class label that was most frequently assigned by the trees.

In the case of regression trees, where the output of each individual tree is a continuous value, an arithmetic average of the n outputs is computed for the final output from the model.

Modeling Automobile Buyer Satisfaction

Data Profile

Number of observations (N = 1,728)

Features	Domain
Buying Price	vhigh/high/med/low
Maintenance cost	vhigh/high/med/low
Doors	2/3/4/5-more
Persons	2/4/more
Trunk Space	small/med/big
Safety	low/med/high

Class	Class Labels
Customer Satisfaction	unacc/acc/good/vgood

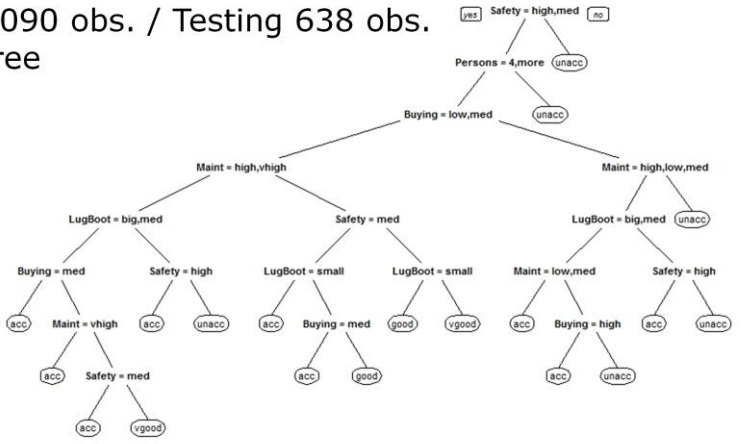
Two modeling approaches:

- Decision Tree
- Random Forests

In the example of modeling buyer satisfaction, overall customer satisfaction is evaluated against key features that describe an automobile. The goal is to use these features to predict customer satisfaction.

Decision Tree (Automobile Buyer Satisfaction)

- Split dataset: Training 1090 obs. / Testing 638 obs.
- Train a single decision tree
- Evaluate with test data



Confusion Matrix

	acc	good	unacc	vgood
acc	127	8	4	3
good	8	14	0	3
unacc	18	0	429	0
vgood	0	0	0	24

Misclassification Rate = 6.2%

1 – (127+14+429+24)/(638) = .062

The complete dataset is split into training and test sets. The training set is used to build the decision tree and the test set is used to evaluate the accuracy of the resulting model. Below is the R code used to perform this analysis:

```
library(caret)
library(rpart)
library(rpart.plot)
library(tree)
library(Rmixmod)

data(car)
df <- car
set.seed(12460)
trainIndex <- createDataPartition(df$acceptability, p = .63,
                                  list = FALSE,
                                  times = 1)
train <- df[trainIndex,]
test <- df[-trainIndex,]

fit <- rpart(acceptability ~ ., data=train, method="class")
pred <- predict(fit, newdata=test, type = "class")
confusion <- table(test$acceptability,pred)
print(confusion)

# plot tree
prp(fit, faclen=5)
```

Random Forests (Automobile Buyer Satisfaction)

- Train a forest of n = 100 trees
- Choose m = 2 random features at each node

Confusion Matrix

```
                Number of trees: 100
No. of variables tried at each split: 2

        OOB estimate of  error rate: 3.47%
Confusion matrix:
      acc good unacc vgood class.error
acc   374    5     3     2  0.02604167
good   10   53     0     6  0.23188406
unacc  23    0  1187     0  0.01900826
vgood  11    0     0    54  0.16923077
```

Misclassification Rate = 3.47%

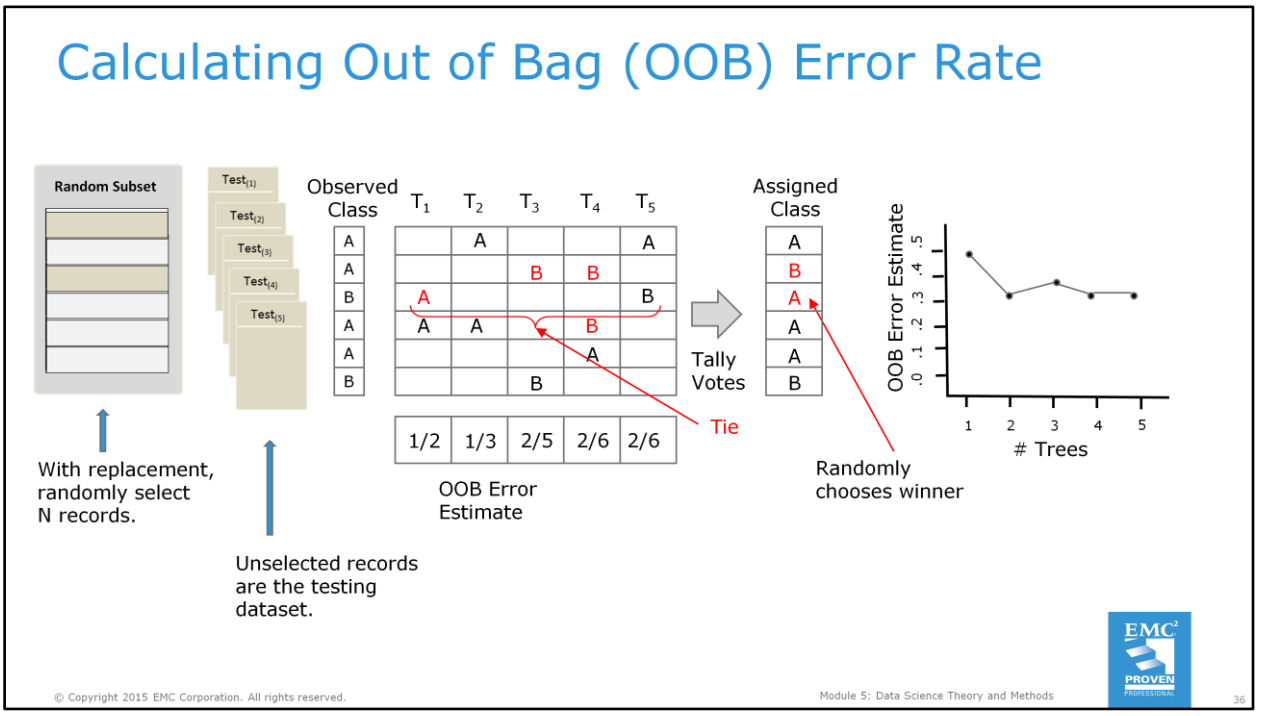
1 – (374+53+1187+54)/(1728) = .0347

Building a Random Forest in R

```
library(randomForest)
library(Rmixmod)

data(car)
df <- car

set.seed(12460)

fit <- randomForest(acceptability ~ .,
                    data=df,
                    importance=TRUE,
                    mtry=2,
                    ntree=100)

print(fit)
```


In R, the complete data set is analyzed using Random Forests. Because Random Forests "hold out" approximately 1/3 of the data for test there is no need to break the data into training and testing datasets. Random Forests will use the hold-out data from each tree to test the performance of the tree it just trained.

In this example, ntree=100 decision trees are trained using mtry=2 randomly selected features at each possible split in the tree.



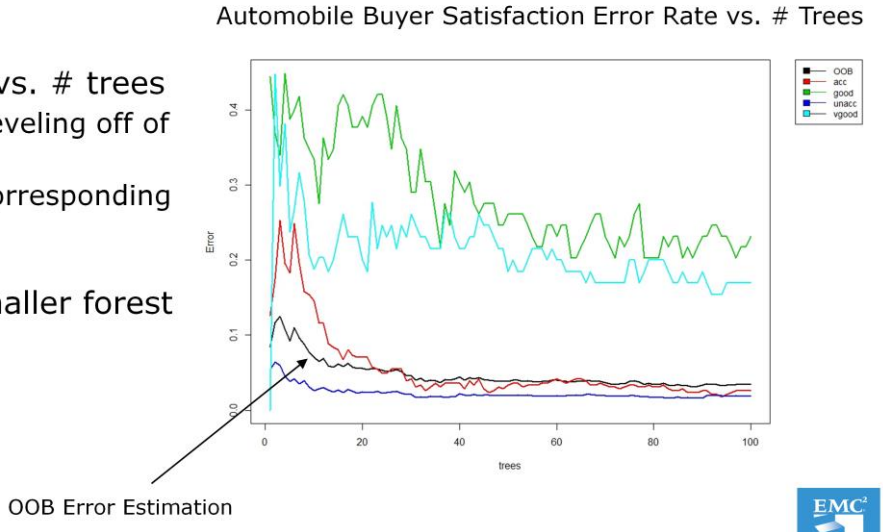
The out of bag (OOB) error estimate is calculated using the "hold out" data from each random subset. It provides an unbiased estimate of the error.

After each tree is constructed, the "hold out" data is pushed through and the classification results are tracked. By aggregating the results of the individual trees, the number of misclassifications are tallied, the result is the out of bag estimate of error.

This example illustrates how the OOB error estimate would be obtained on a dataset of N=6 observations after building five decision trees for the forest. In general, many trees will be required until the OOB error rate stabilizes. Such a plot of the OOB error against the number of trees can be used to determine a reasonable number of trees to include in the implementation of a random forests model in a production environment.

Optimizing the # of Trees in Random Forests

- Plot error rates vs. # trees
  - Identify the leveling off of error rates
  - Choose the corresponding # of trees
- Implement a smaller forest



The number of trees necessary for good performance grows with the number of predictors. When the subsets work as well as the full forest, you have enough trees. Random Forests modeling does not overfit the data; having an excess of trees in the forest will not have a negative effect on model accuracy. However, it will consume extra CPU cycles without any benefit to the model.

Considerations and Cautions

Reasons to Choose(+)

- Simple to build
- Computationally fast
- Robust to correlated variables
- Ideal for many variables
- Used for feature selection

Cautions(-)

- Difficult to interpret
- To avoid ties, use odd number of trees in forest
- Infrequently occurring class labels
  - Use stratified sampling
  - Oversample a class label of particular interest

Random Forests is an ensemble method that is computationally fast and straight-forward to implement. The Random Forests model gets its strength from the randomization which allows the model to handle the input of many variables, particularly when some of these variables are highly correlated with each other. By examining how often each feature is used in a tree, Random Forests can be used for exploratory analytics as well as feature selection. Feature importance is examined in the lab exercise.

When compared to a single decision tree, Random Forests are difficult to interpret and explain to a broad audience. One relatively small issue that may cause some concern with some individuals is the possibility of ties and essentially flipping a coin to break the tie. In the case of exactly two class labels, the possibility of a tie can be avoided with the implementation of an odd number of trees in the forest.

As is the case with many techniques, modelling infrequent or rare events is always difficult. In Random Forests, if one class label appears fairly infrequently, it is possible that the infrequent class label will be under represented in many of the random subsets used to train the individual decision trees. Stratified random sampling can be used to ensure that the observed proportion of the class labels is maintained in each random training dataset.

If it is decided that a relatively infrequent class label is of particular interest, oversampling that class label will improve the identification of such infrequent events. The penalty of oversampling is that many events may be falsely classified as the rare event. However, this penalty may be acceptable if the cost of not identifying a rare event is greatly exceeded by the cost of falsely classifying an event. For example, a company wants to build a Random Forest model that is to predict whether or not a customer may churn (no longer do business with the company). It may be decided that it is acceptable to falsely identify 25% of customers as likely to churn as long as 95% of the actual churners are identified.

Lab Introduction: Random Forests with R and Mahout

This lab covers:

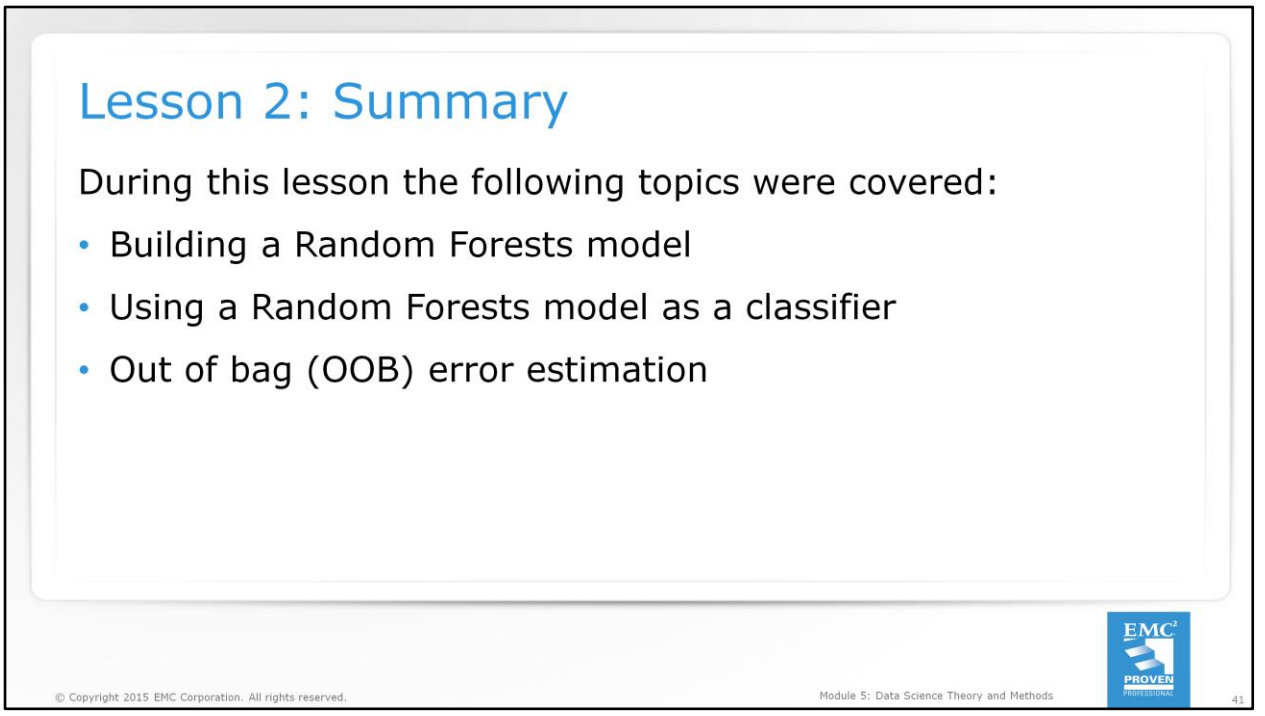
- Using R to build a Random Forests model
  - Exploring modeling options and considerations
  - Evaluating the resulting model
- Using Mahout to build a Random Forests model

In this lab, nursery school application data is used to build a Random Forests model to predict the approval assessment assigned to an application. The models are built using R and Apache Mahout.

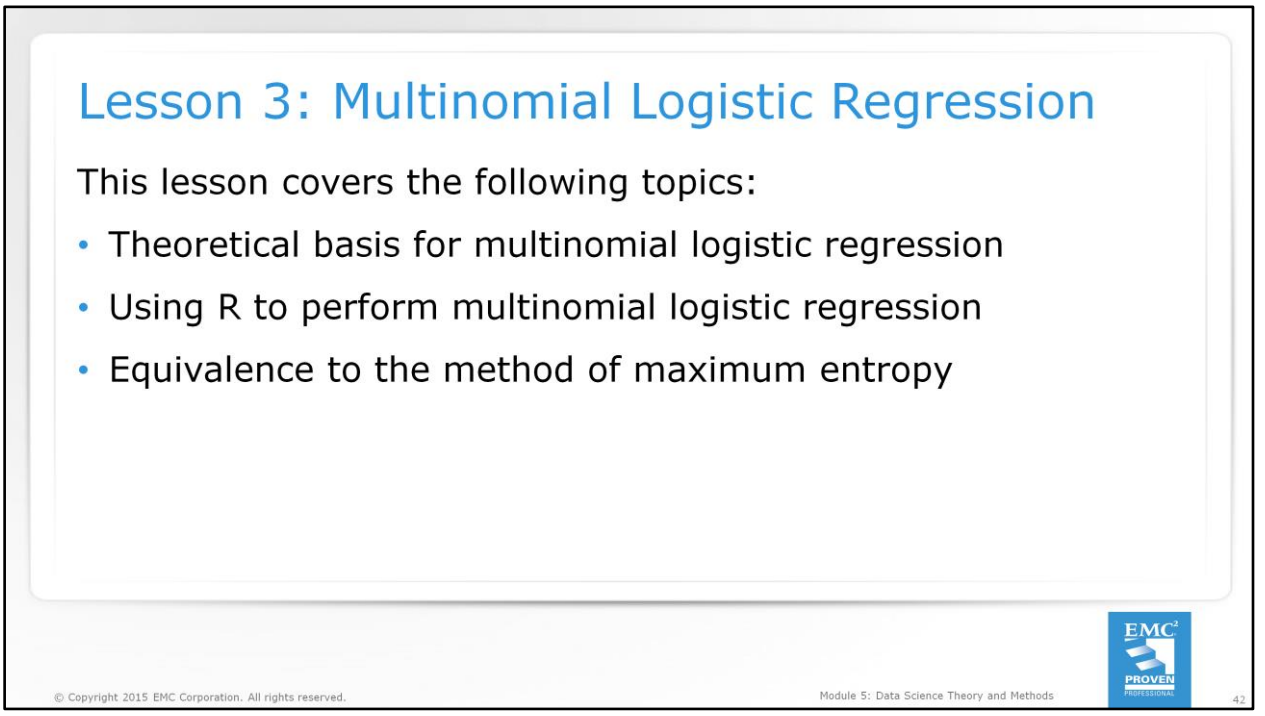
Check Your Knowledge

- Describe the steps of building Random Forests.
- What are the two main input options to Random Forests?
- How do Random Forests calculate OOB error rate?
- How are ties handled in the tallying of the votes?
- What is an approach to prevent ties?

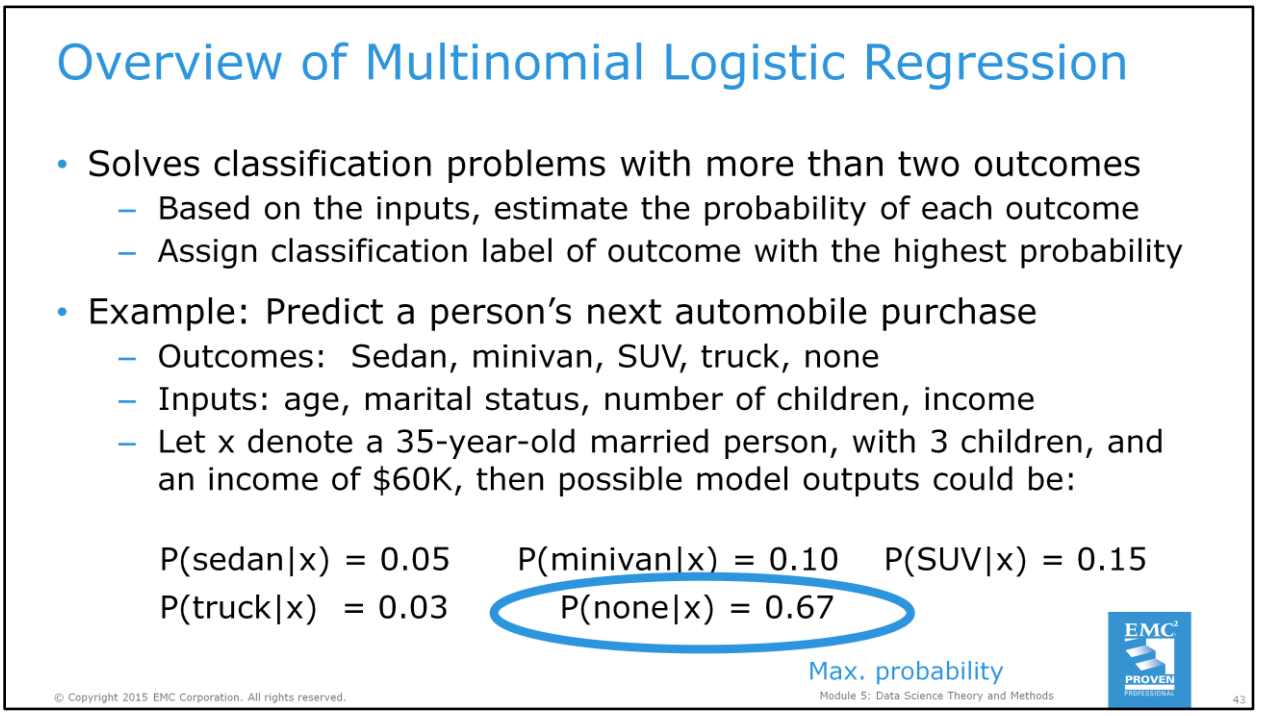
Write your answers here.



This lesson covered the Random Forests ensemble learning method. Random Forests can be used for several purposes including regression, classification, feature selection, and variable importance. Random Forests gets its strength by building a multitude of uncorrelated decision trees and then averaging (regression) or voting (classification) across the forest. Random Forests reduces the possibility of overfitting the data. Random Forests uses the Out of Bag (OOB) Estimate of Error to measure the models' accuracy.



The associate level data science course covers logistic regression where the outcome is binary, such as pass/fail or true/false. Multinomial logistic regression is a classifier that generalizes the binary case to two or more possible outcomes. This lesson describes the theoretical underpinnings of fitting a multinomial logistic regression model and how such a model can be fitted using R. Also, the equivalence of multinomial logistic regression to the method of maximum entropy is presented.



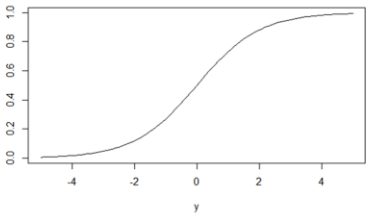
Based on categorical and numerical input variables, multinomial logistic regression estimates the probability of various outcomes occurring. Typically, these probabilities are used to assign a class label to the most likely outcome.

Suppose a multinomial logistic regression model is to be built to predict the most likely type of automobile (sedan, minivan, sport utility vehicle (SUV), truck, or none) that a person is likely to purchase in the next year. From a sample of people and their vehicle purchases over the last year, characteristics such as age, marital status, number of children, and income are obtained. Once the model is built, the automobile purchase probabilities can be calculated for a given individual's characteristics. For example, suppose the probabilities are to be determined for a 35-year-old married person, with 3 children, and an income of $60K. The model estimates the probability that such an individual not purchasing an automobile in the next year to be 0.67. Thus, this person may not be a useful candidate to receive automobile marketing materials. However, if it was decided to provide such materials, this person appears most likely to purchase an SUV.

Binary Logistic Regression Model

- Suppose there are two possible outcomes: {*A, not A*}
- The probability of *A, P(A),* is expressed as:

$$P(A) = f(y) = \frac{e^y}{1 + e^y} \quad \text{for} - \infty < y < \infty$$



where: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_{p-1} x_{p-1}$

$x_1, x_2, \ldots, x_{p-1}$ are the input variables

$\beta_0, \beta_1, \ldots, \beta_{p-1}$ are the unknown model parameters

- The probability of the second outcome is: *P(not A)* $= 1 - f(y)$
- Example automobile purchase outcomes: {auto, no auto}

Consider the case where there are only two possible outcomes, A and not A. Based on the earlier example, the outcomes were reduced to an automobile purchased in the next year and no automobile purchased. In such a case, the problem simplifies to a binary logistic regression model that was presented in the data science associate level course. P(A) is expressed by the logistic function, $f(y) = e^y/(1 + e^y)$ where *y* is a linear combination of the input variables. As y➔∞, f(y)➔1. As y➔−∞, f(y)➔0. Furthermore, at y=0, f(y) = ½. So, the use of the logistic function appears to be reasonable way to model probabilities.

Estimating Binary Logistic Regression Parameters

- Given a dataset of $i = 1,2,\ldots,n$ observations, let:
  - $z_i = 1$ denote outcome *A* was observed
  - $z_i = 0$ denote outcome *A* was not observed
  - $y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_{p-1} x_{i,p-1}$
- The likelihood, L, of observing such a set of observations is:

$$L(\boldsymbol{\beta}|\boldsymbol{z},\boldsymbol{x}) \propto \prod_{i=1}^{n} f(y_i)^{z_i}(1-f(y_i))^{(1-z_i)} \quad where\ f(\,)\ is\ the\ logistic\ function$$

- Maximum Likelihood Estimation (MLE)
  - Choose the values of $\boldsymbol{\beta}$ that maximize L
  - Typically, maximize $\ln(L)$

Given a dataset of i=1,2,…n observations, which include the values of the input variables and whether or not outcome A was observed, let $z_i = 1$ denote outcome A was observed. Otherwise, let $z_i = 0$ denote outcome A was not observed. From this representation of the data, it is desired to find the estimates for parameters, $\beta_0, \beta_1, \ldots, \beta_{p-1}$. For a given outcome, z_i, and a known value of y_i, $f(y_i)^{z_i}(1-f(y_i))^{(1-z_i)}$ provides the probability of the outcome occurring; this probability is based on the Bernoulli distribution. For n observations, the probability is proportional to the product of these probabilities. So, let's choose the estimates of parameters $\beta_0, \beta_1, \ldots, \beta_{p-1}$ to be the values that maximize the product of the probabilities. In order words, choose the parameter values that are most likely to provided the observed dataset. This estimation approach is known as Maximum Likelihood Estimation.

Fitting a Multinomial Logistic Regression Model

- For $K \geq 2$ possible outcomes $\{A_1, A_2, \ldots, A_K\}$,
  - If outcome A_j occurred for observation i, then $z_{ij} = 1$, else $z_{ij} = 0$
  - Let $y_{ij} = \beta_{0j} + \beta_{1j}x_{i1} + \beta_{2j}x_{i2} + \cdots + \beta_{p-1,j}x_{i,p-1}$
  - Maximize:

$$L(\boldsymbol{\beta}|\boldsymbol{z}, \boldsymbol{x}) \propto \prod_{i=1}^{n} \prod_{j=1}^{K} f(y_{ij})^{z_{ij}} \quad where\ f(\,)\ is\ the\ logistic\ function$$

    under the constraint that for a given **x**, $\sum_{j=1}^{K} f(y_{ij}) = 1$

- Since $P(A_k) = 1 - [P(A_1) + P(A_2) + \cdots + P(A_{K-1})]$ for a given ***x***,

  $p \times (K-1)$ parameters must be estimated!

- The necessary numerical analysis routines are available in:
  - MADlib
  - R

Next, the binary logistic model is generalized to handle two or more possible outcomes, $\{A_1, A_2, \ldots, A_K\}$. Since there is a set of coefficients for each possible outcome, fitting a multinomial logistic regression model can become quite complex from a numerical analysis perspective. The notation is somewhat similar to the binary logistic regression case, but multiple z_{ij} indicator values are required to identify each possible outcome in the observations, and multiple y_{ij} expressions are required to account for the outcome specific model coefficients to be estimated.

The effort is somewhat reduced by using the constraint that for any given vector, **x**, of input variables, the sum of the probabilities of the possible outcomes must equal one. Thus, once $K-1$ sets of parameters are determined, the Kth set of parameters is not necessary to calculate $P(A_k)$. So, the numerical analysis problem is slightly reduced to only needing to estimate $p \times (K-1)$ paremater values.

In practice, the provided kernel of the likelihood function is not the expression that is maximized. The constraint of the sum of the probabilities is incorporated through several pages of algebra as provided by: http://czep.net/stat/mlelr.pdf.

Fortunately, the estimation process is provided in some familiar software tools such a MADlib, http://doc.madlib.net/latest/group__grp__multinom.html, and R, http://cran.r-project.org/web/packages/mlogit/index.html. The number of parameter estimates, $p \times (K-1)$, is illustrated with the following R example.

Using the mlogit Package in R

Includes:

- The mlogit() function
- The mlogit.data() function to structure the required data.frame
- The dataset called Fishing

```
library(mlogit)
data(Fishing)
Fish <- mlogit.data(Fishing, choice="mode", shape="wide",
                    varying=c(2:9) )
head(Fish,4)
```

	mode	income	alt	price	catch	chid
1.beach	FALSE	7083.332	beach	157.93	0.0678	1
1.boat	FALSE	7083.332	boat	157.93	0.2601	1
1.charter	TRUE	7083.332	charter	182.93	0.5391	1
1.pier	FALSE	7083.332	pier	157.93	0.0503	1

The mlogit package for R provides a mlogit() function for performing multinomial logistic regression analyses as well as a function, mlogit.data(), for structuring the dataset for analysis. The mlogit package also includes a dataset, Fishing, which contains the preferred mode of saltwater fishing for 1,182 individuals.

The possible modes are fishing from: a beach, boat, charter boat, or pier. The provided input variables include the individuals monthly income which does not depend on the person's preferred mode of fishing. However, the values of the remaining two variables do depend on the alternative modes of fishing. As illustrated for the first individual in the dataset, the price of fishing on a charter boat is more expensive for this individual than the other modes are. The individual's catch rate, a measure of fishing success, varies greatly depending on the mode of fishing selected.

Building a Multinomial Logistic Regression Model

Using variables price and catch

- Estimate coefficients for each alternative
- The reference level is the first alternative, "beach"

```
m1 <- mlogit(mode ~ 0 | price + catch , data = Fish)
summary(m1)
Call:
mlogit(formula = mode ~ 0 | price + catch, data = Fish, method = "nr", …)

Frequencies of alternatives:
  beach    boat charter    pier
0.11337 0.35364 0.38240 0.15059
```


The provided R code fits a multinomial regression model to the restructured Fishing dataset. The generic summary() function is used to display the model results on this page and the next page.

Parameter Estimates – Fishing Dataset

```
Coefficients :
                      Estimate Std. Error t-value  Pr(>|t|)
boat:(intercept)     1.6859962  0.1504298 11.2079 < 2.2e-16 ***
charter:(intercept)  1.9029825  0.1770235 10.7499 < 2.2e-16 ***
pier:(intercept)     1.6554857  0.2049730  8.0766 6.661e-16 ***
boat:price          -0.0123271  0.0015301 -8.0562 8.882e-16 ***
charter:price       -0.0102204  0.0013822 -7.3946 1.419e-13 ***
pier:price          -0.0255988  0.0026692 -9.5905 < 2.2e-16 ***
boat:catch           1.3578497  0.4253244  3.1925 0.0014105 **
charter:catch        0.4695766  0.1248353  3.7616 0.0001689 ***
pier:catch           1.0632619  0.6520553  1.6306 0.1029681
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Log-Likelihood: -1328.1
```


Since two variables are included in the model, $p = 3$ to account for the intercept parameter, β_{0j} for each outcome. Since there are $K = 4$ outcomes and beach is used as the reference level, there are a total of $p \times (K - 1) = 9$ parameters that must be estimated. The output includes the standard *t*-test results on the significance of the coefficients being not equal to zero.

Maximum Entropy (MaxEnt) Classifier

- Popular classification technique in text analysis
  - Train the model based on how frequently terms (words) appear
  - Origins from information theory
  - For a set X of terms, entropy is measured as:

$$H(X) = \sum_{x_i \in X} P(x_i) \cdot I(x_i) = \sum_{x_i \in X} P(x_i) \cdot log_2\left(\frac{1}{P(x_i)}\right) = -\sum_{x_i \in X} P(x_i) \cdot log_2 P(x_i)$$

  - Infrequently appearing words provide the most information, $I()$
- Equivalent to multinomial logistic regression
  - Under certain conditions
  - Although the approaches are different, the two terms are used interchangeably

In an earlier lesson, the maximum entropy (MaxEnt) classifier is identified as one of the text analytic methods to perform named entity recognition and sentiment analysis. Named entity recognition classifies proper nouns as either people names, organization names, or location names such as cities or streets. Sentiment analysis classifies text feedback as either favorable or unfavorable. With its origins in information theory, the maximum entropy classifier utilizes the least frequently appearing words to provide the most information about the class label to assign. In the data science associate level course, the concept of entropy and information gain are discussed in the decision tree lesson.

Although the processing techniques are quite different, under certain conditions, the maximum entropy method is equivalent to multinomial logistic regression. Thus, the two terms for these methods are often interchangeable. Details on the equivalence, can be found in the following reference: http://www.win-vector.com/blog/2011/09/the-equivalence-of-logistic-regression-and-maximum-entropy-models/.

Using Maximum Entropy in R

Scenario: Build a classifier

- To assign a Topic Code to an article based on the terms in the title
- 27 different Topic Codes
- 1,000s of terms

```
library(maxent)              # also loads the tm (text mining) package

data <- read.csv(system.file("data/NYTimes.csv.gz",package="maxent"))
data[c(1:2),c(3,5)]

                                                          Title  Topic.Code
1 Nation's Smaller Jails Struggle To Cope With Surge in Inmates         12
2               FEDERAL IMPASSE SADDLING STATES WITH INDECISION         20
```


The maxent R package implements the algorithm to use the maximum entropy approach to building a multinomial logistic regression model. When maxent is loaded into the R workspace, the tm (text mining) package is automatically loaded at the same time. The maxent package includes a file of NY Times newspaper article titles and the assigned Topic code used to categorize the articles. Selected fields from the first two records are provided for illustrative purposes. The task is to build a classifier that will assign Topic Codes to a future set of article titles.

Prepare the Dataset for Modeling

- The tm package includes:
  - Corpus()
  - DocumentTermMatrix()
- Most of the terms do not appear in most of the titles
  - The document term matrix will be large, but many entries will be 0
  - Use as.compressed.matrix() to build a sparse matrix

```
corpus <- Corpus(VectorSource(data$Title))
matrix <- DocumentTermMatrix(corpus)
sparse_matrix <- as.compressed.matrix(matrix)
```


In this step, the dataset is prepared for the maximum entropy modeling by utilizing the Corpus() and DocumentTermMatrix() function provided in the tm (text mining) package. Since the article titles only contain a few words, most of the term frequency entries in the matrix will be a zero. The inspect() function can be used view the contents of the resulting matrix. The code below provides 21 columns for the first document.

```
inspect(matrix[1,500:520])

<<DocumentTermMatrix (documents: 1, terms: 21)>>
Non-/sparse entries: 0/21
Sparsity           : 100%
Maximal term length: 11
Weighting          : term frequency (tf)

    Terms
Docs antitrust anxieties anxiety anxiety: anxious any aol apart apartment
   1         0         0       0        0       0   0    0     0         0
    Terms
Docs apartments apocalyptic apologizes appeal appeal, appeal; appeals appear
   1          0           0          0      0       0       0       0      0
    Terms
Docs appearance appears appetite apple
   1          0       0        0     0
```

To reduce the memory requirements, the maxent package provides as.compressed.matrix() to build a sparse matrix. This matrix representation eliminates the need to store all of the zeroes.

Train and Test a MaxEnt Model

```
model <- maxent(sparse_matrix[1:3000,],as.factor(data$Topic.Code))
results <- predict(model,sparse_matrix[3001:3104])
results[1,]
```

```
               labels                     19                     15                      2
                 "19"   "0.980473738445362"  "0.00601048830660638"  "0.0001691610568210 9"
                    6                     29                     14                     24
 "0.0001691610568210 9"  "0.0001691610568210 9"  "0.0001691610568210 9"  "0.0001691610568210 9"
                   28                     12                     10                      5
 "0.0001691610568210 9"  "0.0001691610568210 9"  "0.0066256335574606 1"  "0.0001691610568210 9"
                   16                      8
"0.0008722283674783 49"  "0.0001691610568210 9"  ……
```

```
sum(data[3001:3104,]$Topic.Code == results[,1]
55
```

What could be done to improve the accuracy?

Using the first 3,000 documents in the sparse matrix, a maximum entropy model is trained. The predict() function is then used assign a Topic Code to the remaining 104 documents. Based on this testing set, the maximum entropy model successfully assigned the proper Topic Code 53% (55/105) of the time.

Choosing between the Two Approaches

Mult. Logistic Regression

- Include continuous predictor variables
- Explicitly state the linear function of the predictor variables
- Desire a more explanatory model

Maximum Entropy

- Useful for dealing with many discrete variables
- Sparse datasets
- Make minimal assumptions

Multinomial logistic regression easily handles continuous input variables and incorporates all input variables as a linear function. Thus, various transformations of the variables as well as interactions of the input variables can be explicitly incorporated into the model. Based on the estimated coefficients, the resulting multinomial logistic regression model provides some insight into how changes in the input variables affect the probabilities of the various outcomes occurring.

Maximum entropy should be considered when dealing with many discrete variables as is often the case in text analysis when thousands of unique words may appear. Since there are so many possible words, the observed datasets are often considered sparse, which means that most of the possible words appear zero times in a document. Maximum entropy simplifies the modeling because a linear function of the input variables does not need to be built and validated, as is the case in multinomial logistic regression.

Reasons to Choose and Cautions

Reasons to Choose(+)

- Interested in obtaining the outcome probabilities
- Avoid the naïve independence assumption in Naïve Bayes

Cautions(-)

- Number of parameters to be estimated can be large
- Text Analysis
  - Introduction of new words
  - A model for a set of documents in one domain, may not be useful in another

As mentioned previously, multinomial logistic regression provides estimates of the probabilities of the outcomes occurring based on the provided inputs. As seen in the associate level course, classifiers such as naïve Bayes do not provide the true probabilities. Additionally, the naïve independence assumption is not necessary. In some cases, multinomial logistic regression is more forgiving for highly correlated input variables than naïve Bayes classifiers are.

Cautions include the need to estimate many coefficients and the fact that in text analysis using the maximum entropy approach may provide better results.

In terms of text analysis, a maximum entropy model may be useful for a set of documents at a given time, but as new words and terms, for products, individuals, and organizations are introduced, the models will need to be updated in order to use these new terms. Finally, a model built on a set of documents from one domain may not be useful to address another subject area. For example, an analysis conducted on reviews about movies and television shows would not be useful to help classify documents pertaining to the medical profession. Such items are discussed in more detail in the module on Natural Language Processing.

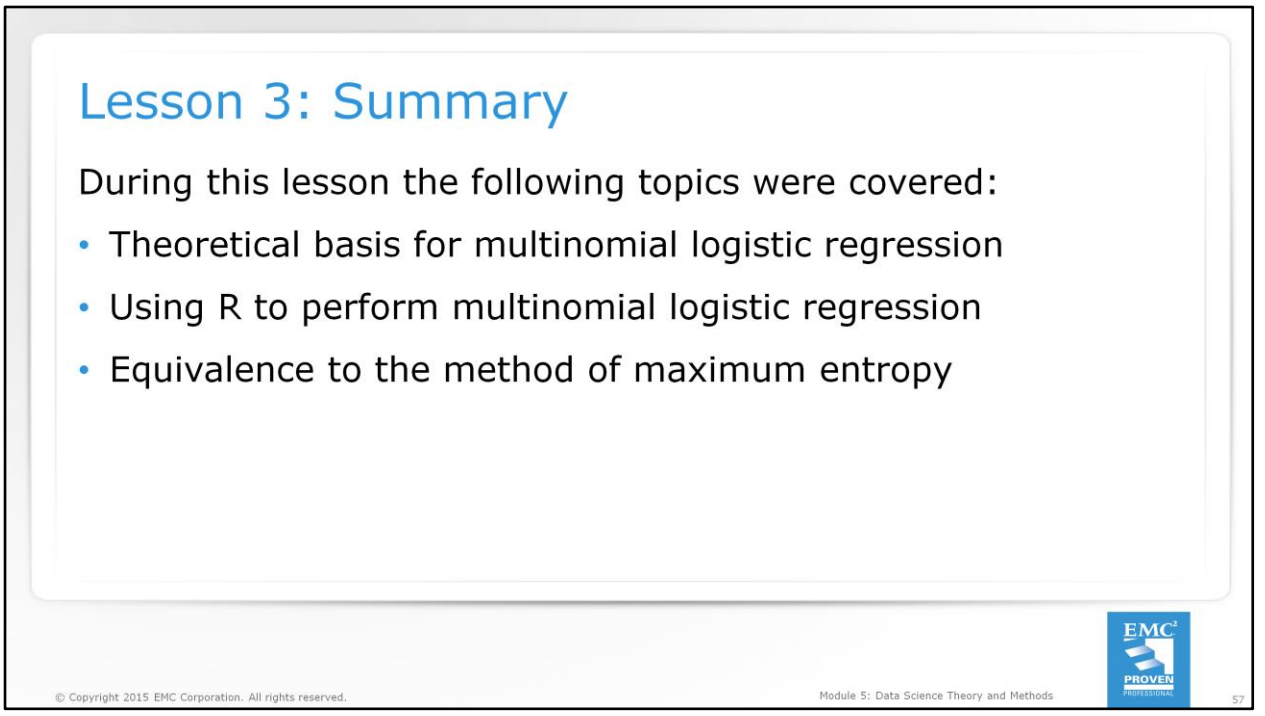
Check Your Knowledge

- Why is the logistic function ideal for modeling probabilities?
- In binary logistic regression, what does the $(1 - f(y_i))$ term represent in:

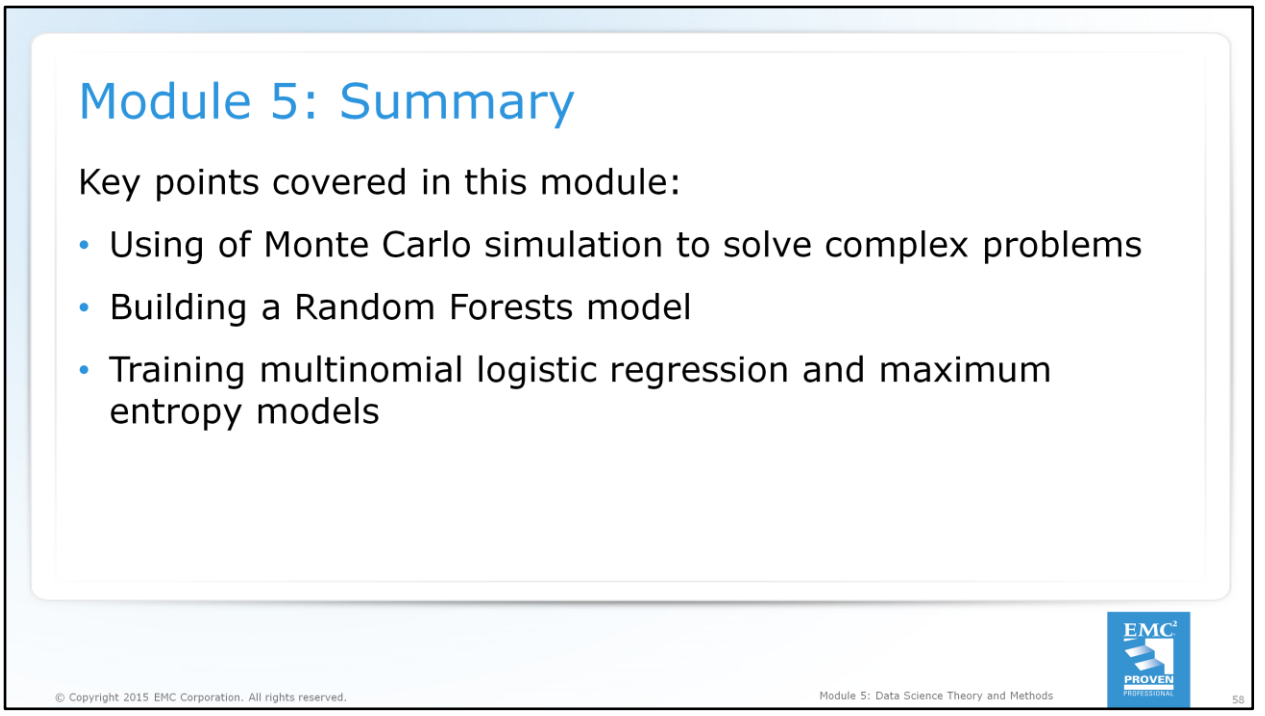
$$L(\boldsymbol{\beta}|\boldsymbol{z}, \boldsymbol{x}) \propto \prod_{i=1}^{n} f(y_i)^{z_i} (1 - f(y_i))^{(1-z_i)}$$

- Based on the fishing R example, for 9 input variables and 5 possible outcomes, how many coefficients would need to be estimated in multinomial logistic regression?
- What classifier method is often associated with text analytics?

Write your answers here.



Building on an understanding of binary logistic regression, this lesson presented the theoretical foundations for multinomial logistic regression as well as the use of maximum likelihood. Using the Fishing dataset in the mlogit R package, an analysis was conducted to model the probability of an individual choosing a mode of fishing based on the cost of each mode as well as the effectiveness of each mode. Finally, this lesson introduced the method of maximum entropy as an equivalent classification method useful in text analytics.



This module and the labs covered the use of random number generators to simulate and analyze complex systems. Building on the associate level data science course, two new classifiers were introduced: Random Forests and multinomial logistic regression. In the area of text analytics, the application of multinomial logistic regression models is often referred to as maximum entropy modeling.