Assignment 1

James Marquez
Lewis University

Inquiry 1

The first theme we learned this week was how to analyze the run-time and complexity of an algorithm using big-Theta, big-O, and big-Omega notation. However, we were not interested in calculating the exact run-time of an algorithm due to the many factors that are involved, such as the programing language used, speed of the computer, and the compiler that translates the program. Instead, we are looking at the complexity of an algorithm and how long it will take to run in regard to its input size. More specifically, we are analyzing the rate at which the run-time grows in relation to the growth in input size, which is called the rate of growth of the running time. The first step in the analysis is to count the number of machine instructions that are performed in the algorithm. For example, a simple loop would take a few instructions to setup the loop regardless of the input size n. If the input size n equals 50 or 1,000, it would still always require the same constant number of instructions to setup the loop. When analyzing the complexity of an algorithm, it's important to only count the sets of instructions that have the greatest impact on the overall speed of the algorithm. For example, say we have 3 instructions to setup the loop and our n equals 100. This would equate to 3 + n, or 3 + 100. We can drop the constant 3 and only keep n because the size of n is what affects the speed of the algorithm the most. Now, say we have a loop and then another loop within a loop. This would equate to 3 + n + 3 + n^2. We can drop the two constants of 3 and also the first n because this time n^2 has the greatest impact on the algorithm's speed. With big-O notation, we're only concerned with the portion of the algorithm that has the greatest impact on the speed of the algorithm in relation to the size of the input size. We use big-O notation to denote the upper bound of the algorithm's complexity. Big-Theta is used to denote the exact complexity of an algorithm while big-Omega is the lower bound of the algorithm's complexity (Cormen & Belkcom, 2017).

The second theme we studied was about formal verification of mathematical algorithms. The topic of verification is very important because safety-critical systems could cause loss of life if a failure occurs that is caused by a bug in the code. Bugs can also cause financial loss if products, such as microprocessors and automobiles, must be recalled or replaced due to faulty algorithms. Current trends have companies creating much more complex designs year after year leading to an increase in the likeliness of bugs. One way to find bugs is to perform exhaustive tests, which are slow and impossible to create a nearly infinite number of possible test cases. One method that is used instead of extensive testing is formal verification, which is to mathematically prove the correctness of a design with respect to a mathematical formal specification. However, formal verification is also difficult to perform in real-world software and hardware scenarios because the intended behavior must be formally specified, hidden assumptions must be made explicit, and they require long detailed proofs that are difficult to review. One possible solution is to have a program either generate a proof or check a proof automatically. Some benefits of automation include a reduction in the risk of mistakes and the capability to automate portions of the proofs. However, proofs currently cannot be fully automated, so human intervention is still required. Formal verification has three primary approaches: symbolic simulation, temporal logic model checking, and general theorem proving (Harrison, 2002).

The third theme we were instructed on was distributed algorithms. A distributed system is one in which the process of an autonomous computational entity communicates with the process of another autonomous entity. Each processor in the system does not share memory or a clock, and does not have to know the total number of processors in the system. A distributed algorithm is one that runs on a distributed system, and does not have a central coordinator processor. Distributed systems include applications such as domain name system (DNS), peer-to-peer file sharing, and cloud computing. The largest distributed system is the Internet while some of the smallest are multi-core processors. The primary definition of a distributed system is one in which many entities may be active in the system and have a level of freedom from the other entities in the system. Each member shares resources and information to solve a problem that affects some or all of the members in the system. A distributed algorithm coordinates the communication and computation between the members of the system to solve a given problem or task. One important aspect of large distributed systems is that each member does not have a global view of all the members in the system at any given time. Each member has only a view of the members within its vicinity, and must contribute to the solution of the problem with only the local information available to it. Some tasks can be solved with only the local information each member has available, such as the maximum temperature in a group of sensors, but not the entire system. However, some problems are global, such as finding the maximum temperature in the whole system instead of only a local view. The distributed algorithm's job is to coordinate the transfer of information across the whole system (Lenzen & Wattenhofer).

Inquiry 2

Working as a Healthcare Data Analyst for the U.S. Veterans Health Agency (VHA) provides the opportunity to connect to many nationwide databases housed within a large data center in Austin Texas. Analysts connect to their desired database via Microsoft SQL Server and use SQL to query and retrieve data. An enormous amount of patient health related data, patient appointment history, and visit history is stored to provide analysts, statisticians, and researchers with the capability to perform many types of analyses. Retrieving data from these databases and turning it into actionable intelligence is vital to the mission and goals of VHA's national leadership, as well as each facility's executive leadership and mid-level managers. However, at the time of this writing, only structured data is extracted and loaded to the data warehouse. This current system excludes enormous amounts of unstructured free-text patient health data rendering them useless for large data mining projects. The current data architecture is not capable of extracting, transforming, and loading (ETL) information from the many digital notes that each medical provider writes when seeing a patient. The only information currently available to retrieve from the database regarding provider notes is note title, title description, date created, created by, etc. Perhaps a research-based approach should be taken to understand the necessary changes to the architecture required to ETL large amounts of free-text data.

Another area of computer science that VHA has not fully embraced yet is the technique and use of data mining and machine learning to improve clinic management practices. Machine learning is being used in the bio-informatics and medicine fields to diagnose diseases such as cancer, but has not fully been embraced by leadership to improve clinic management practices. One area that can be improved by learning algorithms is in reducing the number of patients that do not show up for their appointment, also called missed opportunities. Each patient that no-shows decreases patient access by wasting a clinic slot that otherwise could have been used by another patient, while also disrupting the daily workflow of a provider leading to decreased productivity. It also reduces the number of billable encounters a medical facility has in a given period. My facility has roughly a 191,000+ veteran patient population with an average 17.5% no-show rate out of roughly 450,000 annual appointments. In an attempt to reduce the number of no-shows at my facility, I used data mining to gather many years' worth of patient appointment historical data due to learning algorithms benefiting from large amounts of data. For this project, I selected is a gradient boosting tree learning algorithm. Also, I selected the R statistical programming language environment to perform all ETL, data transformations, and tests in.

The first step in the machine learning project was to use embed SQL in the R environment to query VHA's corporate data warehouse (CDW). After acquiring the data, the next step was to create features that the algorithm can learn from, also called explanatory or independent variables. I created 30 features: patient historical no-show rate, patient historical cancelation rate, patient's historical no-show rate by each clinic, patient cancelation rate by each clinic, boolean value whether patient has been seen in the same clinic within 24 months, patient's historical no-show and cancelation rates by each department, boolean value whether patient has been seen in the same department within 24 months, number of days patient is waiting for their appointment, number of days since patient's previous appointment, number of most recent consecutive no-shows patient has accrued, number of most recent consecutive no-shows patient has accrued by clinic and department, sum of appointments patient has on the same day of the predicted appointment, patient's historical sum of appointments in the past 12 months, some of patient's historical consults in the past 12 months, name of appointment's department, appointment's duration length in minutes, appointment's month (Jan, Feb, etc.), weekday (Mon, Tues, etc.), appointment's hour (8am, 9am, etc.), number of days elapsed since consult was requested, and patient's gender, age, marital status, race, address, and boolean values for if the patient is homeless, a substance abuser, or has low income. I also tried gathering each patient's transit times from Google's mapping API, but had too many patients and exceeded Google's maximum daily request limit. Most of these features are not stored directly in the database and must be created during the phase called feature engineering. After centering and scaling the data, one-hot encoding, and splitting the data into train and test sets, I trained the algorithm on the training set and performed a 10-fold cross-validation on the test set. My final conclusion was a model that when setting the probability threshold to 0.22 resulted in a true positive rate of 0.7283 and a true negative rate of 0.8617.

References

Cormen, T., & Belkcom, D. (2017). Big-O notation. Retrieved from Khan Academy: https://www.khanacademy.org/computing/computer-science/algorithms/asymptotic-notation/a/big-o-notation

Harrison, J. (2002, 06). Formal Verification of Mathematical Algorithms. Retrieved from Intel Corporation: https://lewisuniversity.blackboard.com/bbcswebdav/pid-2980604-dt-content-rid-16259947_1/courses/FA17-CPSC-59700-005/Formal%20Verification.pdf

Lenzen, C., & Wattenhofer, R. (n.d.). Distributed Algorithms for Sensor Networks. Retrieved from Hebrew University of Jerusalem: https://lewisuniversity.blackboard.com/bbcswebdav/pid-2980604-dt-content-rid-16259946_1/courses/FA17-CPSC-59700-005/Distributed%20Algorithms.pdf