CMPT 732 Big Data Programming I

Intelligent Security System for real-time detection, classification of cyber-attacks with prescriptive analytics using Big Data and Machine Learning Techniques

Team Knights

#301436162 Gokul Mohanarangan
#301427935 Pranav Balaji
#301432876 Shivek Chhabra

Link to the GitLab project repository: https://csil-git1.cs.surrey.sfu.ca/sca320/knights-cmpt-732

Link to the project video on YouTube: https://www.youtube.com/watch?v=VqJyA-R65m4

Link to uploaded video on google drive:
https://drive.google.com/file/d/17SVqk0T6WVjX05DeaSueQhhatwICTx0l/view?usp=sharing

Problem statement and overview of challenges:

More than 60% of the world's population contributed, in some way, to internet traffic in the first quarter of 2021. With corporations swiftly embracing cloud solutions and infrastructure, cyber security becomes a core pillar that can make or break a company's business continuity. With the proper big data tools, the insights from historical network data can be a game-changer in how enterprises deal with network security. We want to combine our knowledge of big data techniques and machine learning to address this problem aptly, for the current technological landscape. The data pipeline, put together by our big data techniques, has to keep up with the varying rates of the incoming network data streams. The solution has been engineered to be easily scalable and mindful of resources available at the same time. The necessity of processing a substantial load of incoming traffic without latency is key to the solution we are aiming to achieve, as delayed attack detection will have done some damage already. By categorizing and predicting each type of attack, we also aim to provide valuable analytical insights to manage resources, loads and an up-to-date security portfolio.

We target extracting impactful analytical insights from historical and rapid incoming data, as feedback to enterprises to make informed corporate decisions, apart from detecting ongoing cyber-attacks which aids in deploying necessary responses like automated countermeasures, in time. This should also aid self-explanatory visual representations to cater to both technical and non-technical audiences. The underlying components have been carefully chosen keeping these requirements in mind. Fact-driven decisions form the core of this solution, where the techniques and the pipeline are the results of intense static and runtime performance scrutiny of our approach at all times.

Our tools, approaches, the problems we faced therein and our solutions to them are discussed in detail in the sections below. Our architecture and process flow have been visually summarized below. The various components in the flow along with the stages have also been shown.

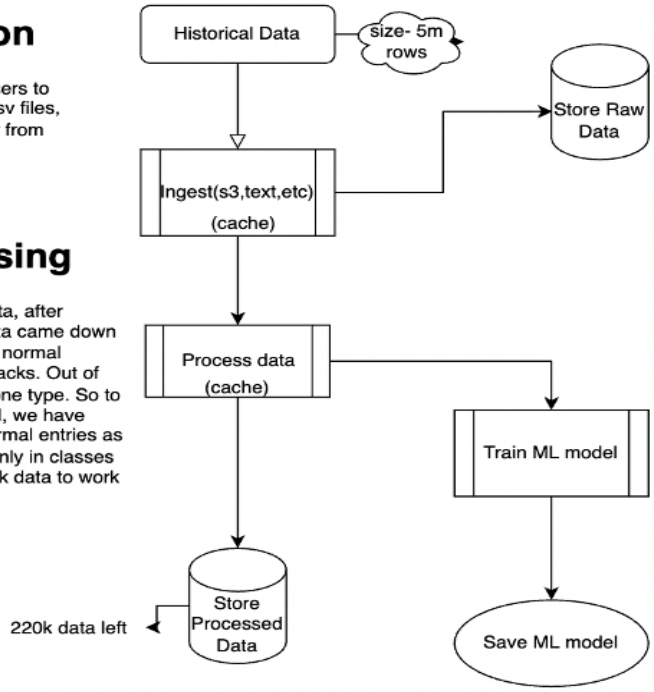
Training:

Data Ingestion

Our functions allows the users to ingest data from any text/csv files, from postgres database, or from AWS S3.

Data Processing

We started with 5m raw data, after dropping duplicates the data came down to 1m. In that we had 800k normal requests and only 200k attacks. Out of those 200k, 120k were of one type. So to make the classes balanced, we have augmented the existing normal entries as attacks and distributed evenly in classes leaving us with around 220k data to work with.



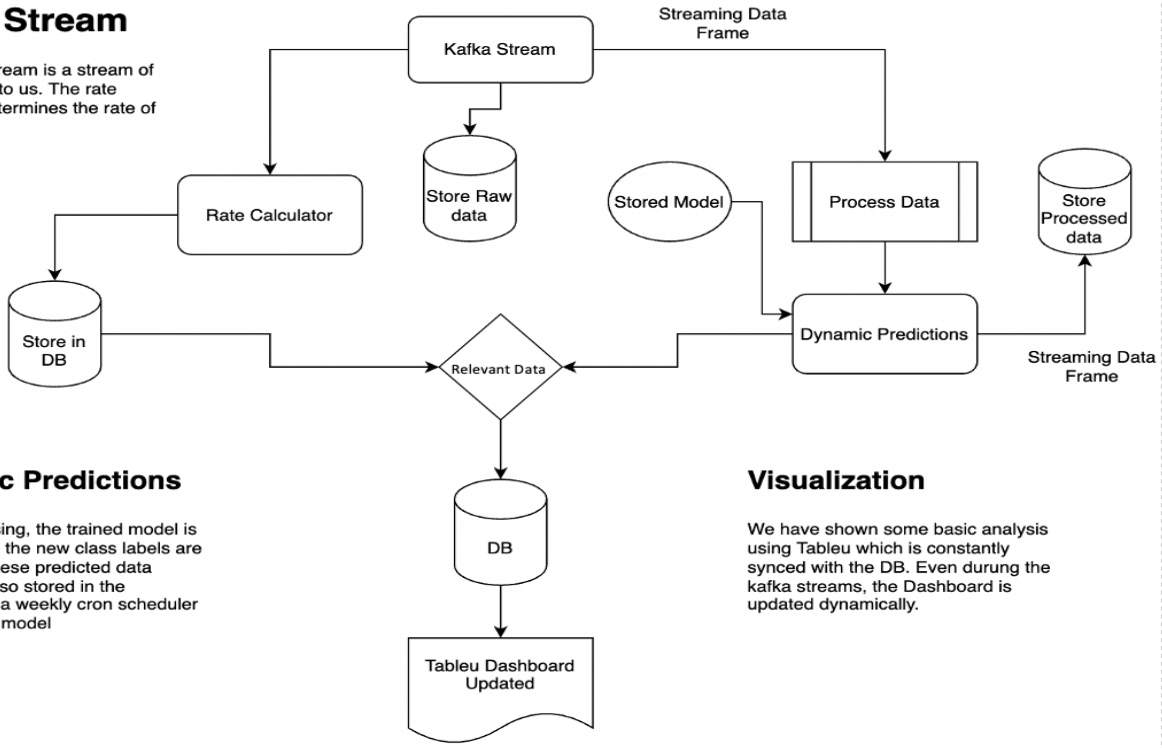
ML Model

Modeling wasn't the real goal of our project. We just wanted to show that a computationally expensive task like modeling can be performed parallely on big data.

Prediction:

Kafka Stream

The Kafka stream is a stream of data coming to us. The rate calculator determines the rate of transfer.



Dynamic Predictions

After processing, the trained model is retrieved and the new class labels are predicted. These predicted data entries are also stored in the database for a weekly cron scheduler to retrain the model

Visualization

We have shown some basic analysis using Tableu which is constantly synced with the DB. Even durung the kafka streams, the Dashboard is updated dynamically.

Methodology:

The first step in the process of creating an efficient model for cybersecurity implied extensive cleaning and processing stages, thereby enabling the generated model to make meaningful predictions without being plagued by the wide arrays of biases that occur when processing millions of data packets. We started with packet information with about 5 million data points. We wanted to retrieve this data from cloud storage and due to the low cost of storage and high availability, we chose Amazon S3. Unfortunately, due to the nature of internet traffic, data obtained within small timeframes are highly likely to be redundant. Such redundancies can adversely bias a predictive model and hence we had to make sure our data does not capture it. Through basic analysis of our raw data, we also identified a heavy skew in the kinds of cyber-attack data available in the dataset. Though our data encompassed a wide array of cyberattacks overall, it was heavily skewed to just 5-6 attack types. This meant that our resultant model will be heavily skewed by them as well. To solve this issue, we went through extensive domain analysis and figured out a broader category of attacks where each type of attack will be a sub-category. This will eliminate the inherent skew of the predictive labels and would consequently lead to better modelling results. Since we want to emulate an industrial environment, we decided to store this into a data warehouse, in our case, Postgres DB, so that subsets of this processed data can be used for machine learning and other relevant data analysis. Apart from answering questions related to cybersecurity data we deemed it important that our code performs in the most efficient way possible, to address the core of any big data project - high scalability. To realize this, we have done extensive static and runtime code analysis, in addition to rudimentary data analysis, to find the most efficient methodologies in our data processing pipeline. We pitted complex functions pre-defined in spark against manual logic built from scratch via fundamental data frame operations and compared the results. The results of that analysis were used to pick the most efficient functions which were utilized in our pipeline (inferences from code analysis are shown in the Results section). The data analysis pipeline followed the machine learning pipeline where analysis regarding the best learning algorithm was done, to get the most accurate results possible and the final model is saved for dynamic predictions.

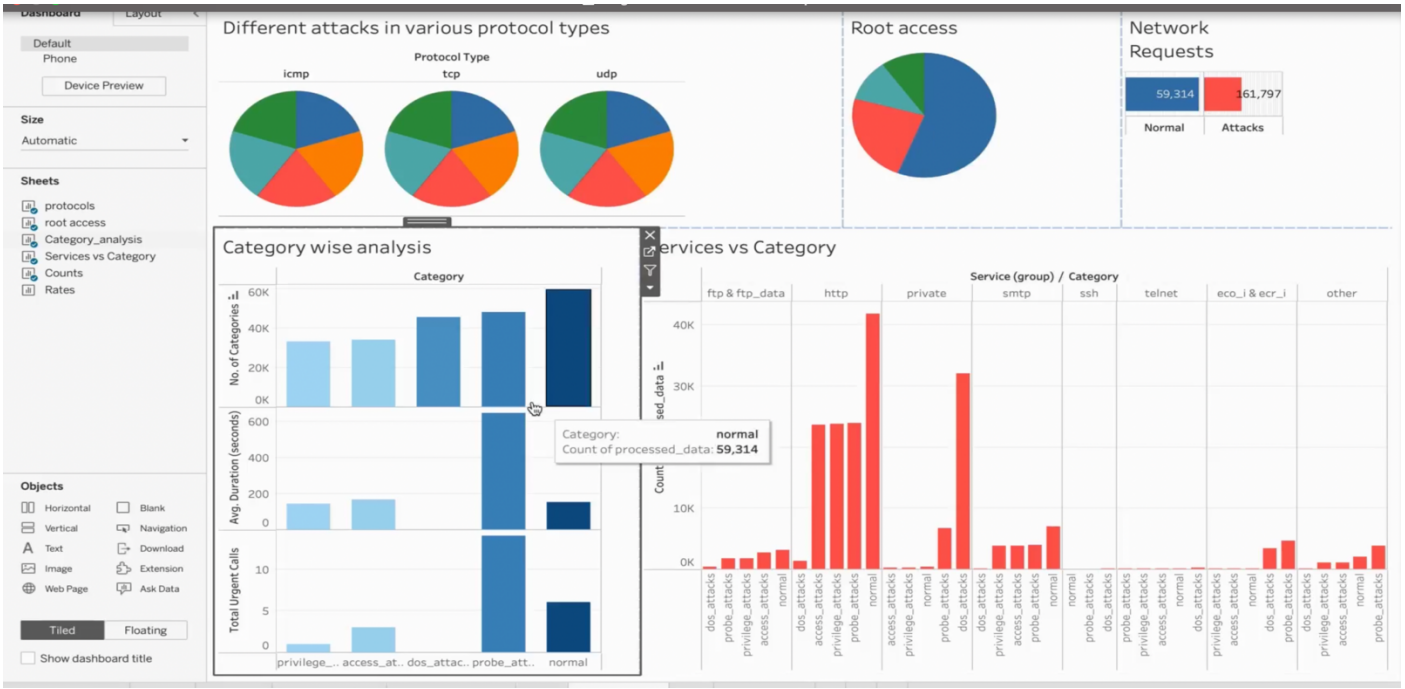
Apart from a reliable model, one of the key goals was to assert our pipeline's capability to process and predict a dynamically changing continuous stream of data to identify threats in real-time. Latency was a luxury the solution could not afford. For achieving this, Kafka was used in simulating fluctuating data traffic. This was used by our model to make real-time predictions as well as for external analysis in making informed business decisions. This simulated data traffic was pushed through a Kafka stream which is being read by two concurrent spark jobs where one dynamically produces the machine learning predictions, whereas the other further processes the streaming data to generate data points for real-time analysis and insights. Several modifications to the original pipeline were done for accommodating the unique constraints faced when working with unbounded data. To obtain the rate of traffic flow, simple time difference operations could not be done due to the very nature of distributed computing as well as how streaming data can only do aggregation or join operations on the entire table. To circumvent this caveat, we went along with a time window-based approach where the rate of traffic is calculated on a per-window basis since such a function can be naturally done via aggregation operations. The two parallel spark jobs also continuously write the results into our data warehouse. The weighted prediction results will be used for future training of the model, whereas the results from the dynamic analysis will be used for providing real-time insights. Apart from this, several kinds of data analysis, easily perceivable and intuitive visualizations that go with it, have also been crafted. This is to answer the problems posed in our problem statement, the results of which are in the results section.

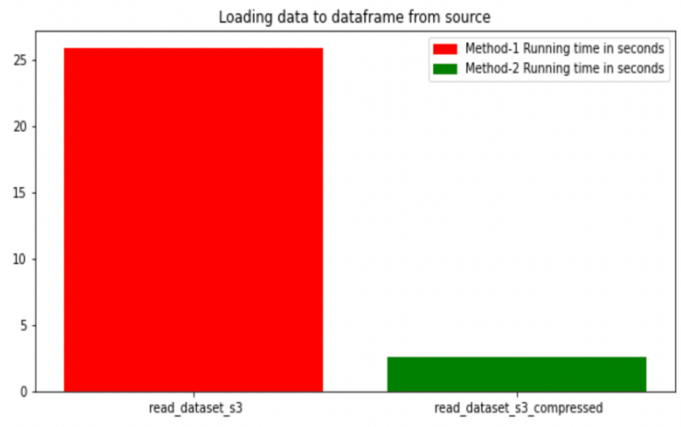
Problems faced:

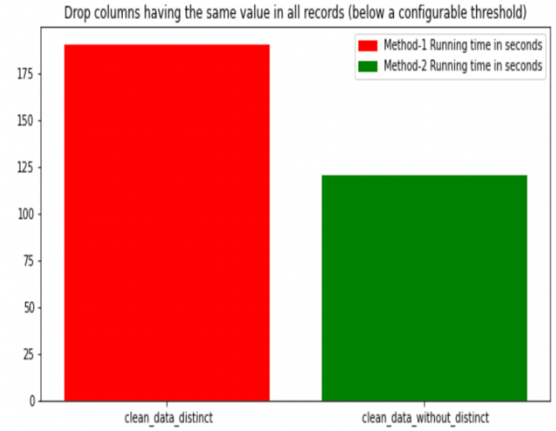
- Finding a dataset that fits our problem scenario was a bit tedious. We compared and discussed based on the rank of each of the datasets, which was decided by us on several metrics, before finding a suitable dataset.
- While trying to connect S3 to Postgres DB via pyspark functions, several connection issues occurred. Using the right connectors and ensuring compatible versions of the components solved the problem.
- Due to the highly imbalanced data, conventional data processing shrank the data from 5M rows to just over 60K rows. With a relatively small number of data points to work with, it was difficult to properly gauge the high scalability of the solution. After discussing this issue with the professors, we have augmented the data to redistribute an extensive amount of "normal" data packets available and intentionally classified them as one of the attacks. This reduced the overall data bias in addition to increasing the number of data points to better gauge the solution's scalability.
- Heuristic approaches in ETL stages consumed a lot of memory and processing time. Each method in the code has been diligently scrutinized on performance and efficiency before earning its worth in the pipeline. This required us to go one step further in doing exploratory analysis on big data processing before arriving at convincing pipelines.
- We decided to utilize spark's multi-processing capability by simultaneously running the training stage as well as storing the preprocessed data into the data warehouse. This was only computationally feasible by carefully going through the logic to figure out all possible branches and caching appropriately.
- Locally running complex codes with multiple levels of caching led to memory issues intermittently. Upon research, we found spark's garbage collector not freeing up memory fast enough. Through inspection, we manually freed memory by pre-empting the garbage collector and flushing spark caches wherever necessary, to consistently run the solution in a memory scarce environment. Unused entities consuming memory, have also been dealt with meticulously.
- Faced lots of issues while trying to connect Kafka through a confluent connecter to S3. This structure was removed due to the excessive charge which would be incurred to stream data continuously from S3.
- Structured streaming cannot handle caching well since you cannot cache an unbounded stream, unfortunately, no error directly pointed to this cause hence lots of trial and error was needed to figure out the underlying issue. This caused us to restructure where we cache for the dynamic predictions to work.
- Multiple aggregations cannot be handled by spark even though it has no theoretical limitations. Hence the time window approach had to be taken to calculate and display packet flow rate
- Initially, only one spark code was supposed to listen to the Kafka stream but since aggregation of results led us to lose data and caching is also not possible with streaming data, we had to restructure our pipeline in such a way that two spark jobs listen to the same stream and generate their respective outputs which are then related to each other in the visualizations.
- While doing a continuous runtime test initially, we quickly incurred the problem of exceeding the AWS free tier limit. We then realized that it was much more computationally efficient and also bandwidth efficient to store only compressed zip files in AWS and extract them locally through spark than loading the raw data directly from S3.

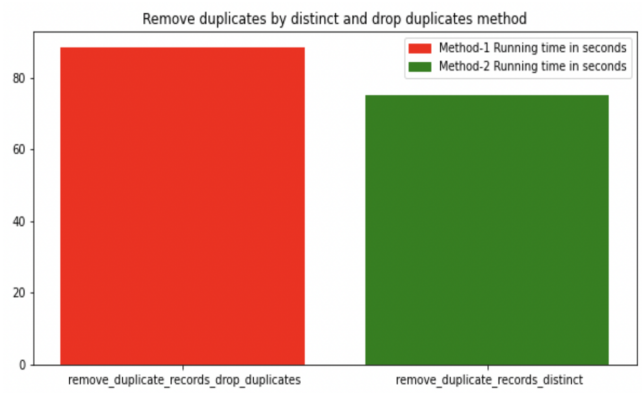
Results:

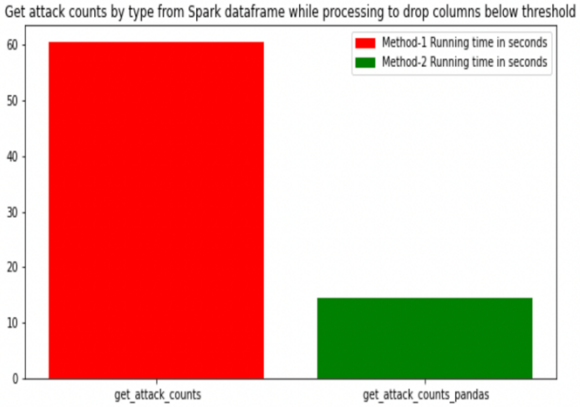
- We were able to successfully achieve our goal of ingesting, processing, and dynamically computing predictions on big data using Spark and Kafka. The performance and the capability to scale were satisfactory based on the final rubrics.
- We were able to harmonize various data and visualisation tools, some of which were relatively new, to successfully build data and ML pipelines capable of handling big data, to generate a model that works well. Using the knowledge gained through this process, we are confident of driving a big data solution from inception to completion.
- Insights regarding load scaling can also be achieved by making a comparison of what classes of packets are being predicted and network traffic. Based on this, the companies can make informed decisions.
- We created two additional google colab notebooks for visualisation of the static, run time and data analysis results, which have also been added to our project repository. The notebooks summarize insights that were both useful and interesting in our perspective.
  - Probe attacks tend to take the longest duration of connection and categorize mostly as urgent.
  - While dos and probe attacks requested Super User access the most, access attacks got them the most.
  - While most attacks came as an HTTP request, dos attacks came as private and SSH was barely used to attack the system.
- We used the results of the visualisation to answer queries that guided us throughout the project. We made informed and well-driven decisions on which approaches to use, where to focus for efficiency or accuracy, feedback of pipeline performance etc.
  - Removing duplicate entries by using distinct was faster than using spark dropDuplicates() function.
  - Using Spark DF to Pandas DF to get the counts was a faster alternative.
  - We built custom logic to pick the first value and filter the values of the column to find the uniqueness instead of using distinct.

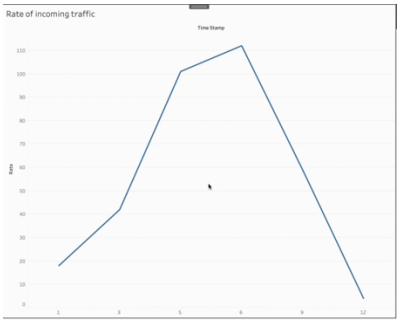












Project Summary:

- Getting the data: Acquiring/gathering/downloading. – 1
- ETL: Extract-Transform-Load work and cleaning the data set. – 3.5
- Problem: Work on defining problem itself and motivation for the analysis. - 3
- Algorithmic work: Work on the algorithms needed to work with the data, including integrating data mining and machine learning techniques. – 3
- Bigness/parallelization: Efficiency of the analysis on a cluster, and scalability to larger data sets. –5
- UI: User interface to the results, possibly including web or data exploration frontends. -0
- Visualization: Visualization of analysis results. – 2.5
- Technologies: New technologies learned as part of doing the project. -2