Performing Transaction Synthesis through Machine Learning Models

Authors:

Justin Charron

Li Li

Yudi Wang

Shihao Xia

Faculty Advisors:

Professor Elke Rundensteiner

Assistant Professor Yanhua Li

Sponsor Organization:

ACI Worldwide, Inc.

Sponsor Advisor:

Eric Gieseke (ACI)

Performing Transaction Synthesis through Machine Learning Models

A Major Qualifying Project
Submitted to the Faculty of WORCESTER POLYTECHNIC INSTITUTE
in partial fulfilment of the requirements for the
Degree of Bachelor of Science by:

Justin Charron
Li Li
Yudi Wang
Shihao Xia

Date:
22 March 2017

Report Submitted to:

Professor Elke Rundensteiner, Professor Yanhua Li
Worcester Polytechnic Institute

Eric Gieseke
ACI Worldwide, Inc.

TABLE OF CONTENTS

TABLE OF FIGURES

ABSTRACT

ACI Worldwide is a payment processing company that uses fraud detection solutions to process the massive amount of transactions that go through the company every day. The goal of this MQP project was to address privacy concerns in using real transaction data to test fraud detection software. We worked with our advisors at WPI and ACI to develop a product that can be used by third party companies to test their fraud detection solutions. Our team looked at different machine learning and statistical methods to build working models from the large quantities of transactional data and then use those models to synthesize artificial data that follow the same patterns and behaviors. Our team also developed a test suite to measure the accuracy of and validate the generated data.

EXECUTIVE SUMMARY

ACI Worldwide and MQP Projects

ACI Worldwide, the "Universal Payments" company, is a payment processing company that services more than 5,100 customers worldwide with more than 1,000 of them being some of the world's largest financial institutions. Processing more than $14 trillion in payments daily, having an effective suite of fraud detection software is crucial for ACI to operate their business effectively. Through the sponsoring of a series of Major Qualifying Projects (MQP) at Worcester Polytechnic Institute, ACI has a history of experimenting with the latest and most powerful rising open-source technologies available. These projects allow ACI to make their fraud detection and payment processing systems as fast and up to date as they can to handle the increasing amounts of raw data that they process daily.

All of the MQP projects at WPI that ACI has sponsored have built on top of each other and mainly focus on ACI's fraud detection suite. The first such project took place in 2013 and created what is now ACI's Complex Event Processing (CEP) system. This system was built using the Esper engine with the goal of replacing previously slow-running SQL queries. The following project in 2014 expanded on the CEP system by using distributed computing platforms, namely Kafka and Storm, to make the system horizontally scalable. The 2015 MQP sponsored by ACI built a graph database using technologies such as Titan and Cassandra that was capable of computing extra attributes on each node and retrieving nodes quickly. That project also built a visualization engine to use the graph database using Vis.JS. This made the job of data analysts at ACI easier by providing a tool with which outlier detection could be performed.

Model Generation

The goal of this MQP project was to address privacy concerns in using real transaction data to test fraud detection software. Our team's objective was to create a model that accurately represented the behaviors and patterns that existed in the transaction data that ACI processes, and then to use this model to generate new synthesized data that does not contain any private information yet still exhibits the same behaviors and patterns of the original data. To do this, we started by looking at a number of existing machine learning methods and libraries that could be used to generate this model.

Our team decided to first implement a precursory approach to model generation that used statistical methods to build a Multivariate Gaussian distribution on the data. A simple sampling of this distribution was used to generate the new, fake data. Columns that did not contain continuous data, such as string data, were fitted to a normal curve based on the frequency of each unique value and then translated into continuous data.

Technologies Used

Our team used open-source libraries throughout our entire project, many of them from Apache. The initial model generation was done using an Apache Derby server embedded into our Java program, and the calculations and curve fitting were done using Apache Math. We then created a Hadoop server on which we hosted an HBase database to hold the transaction data, using Apache Phoenix to read and write to this database. Our group then refactored the model generation code to use Spark and its built-in HBase support. We moved to Spark to make use of its RDD-based calculations, which are already optimized to be horizontally scalable to be run in a Hadoop server using MapReduce.

Data Validation

Once new data was synthesized, it was necessary to do some validation to make sure that the synthesized data simulated the original data successfully without revealing any private information. Our process was simple: calculate the covariance matrix of the actual data first, and then calculate the covariance matrix of the synthesized data, then compare these two covariance matrices to get the similarity percentage. This percentage approaches 1 the closer the simulated data is to the original dataset.

Future work

During the course of this project, several areas were found that the project could be expanded upon. One of these such areas is an entirely new use case for the models generated for the data. It would be possible that if there were a model generated on a specific customer's transactions that fraud detection could be performed by comparing new incoming transactions to the model generated for that customer. A possible implementation of this could look at how many standard deviations away from the mean each column was located. There is also still room in our project for further horizontal scaling of the model generation and curve fitting parts by running them on larger Hadoop servers.

ACKNOWLEDGEMENTS

Our team would like to thank Professor Elke Rundensteiner and Professor Yanhua Li from WPI and our mentor Eric Gieseke from ACI Worldwide for their constant support and guidance throughout the duration of this project. Also, we would like to thank ACI Worldwide for sponsoring this project and for providing many of the resources necessary to complete this project. Without their help, this project could not have been possible.

CHAPTER 1: INTRODUCTION

ACI Worldwide, the "Universal Payments" company, is a payment processing company that services more than 5,100 customers worldwide with more than 1,000 of them being some of the world's largest financial institutions. Processing more than $14 trillion in payments daily, having an effective suite of fraud detection software is crucial for ACI to effectively operate their business. One important aspect of an effective fraud detection suite is the data being consumed.

As denoted by the FTC, under the Safeguards Rule in the United States "financial institutions" must protect their customers' sensitive information and ACI satisfies the conditions put forth by the FTC of being a "financial institution" (Federal Trade Commision, 2006). Due to the need to adhere to FTC guidelines, ACI requires the approval of the customer to use their data in any sort of analysis which is time consuming to collect. Therefore, a method of assuring that the customers' data is secure and protected whilst allowing for a relatively small downtime in redacting any sensitive information by generating data from models is highly sought after not just by ACI but by other companies worldwide as well. To accomplish this, ACI continued its history of teaming up with WPI to sponsor this concept as a major qualifying project.

ACI has a history of sponsoring major qualifying projects from WPI on a variety of subjects concerning their transaction systems. All of the projects have been designed to build on top of each other to further expand and improve ACI's extensive fraud detection suite. The first project took place in 2013 and focused on creating ACI's Complex Event Processing (CEP) system that used the Esper engine to replace previously slow-running SQL queries. The next project in 2014 expanded upon the previous one to use the distributed computing systems Kafka and Storm to make it horizontally scalable, increasing the speed of data ingestion and feature computation with more machines added to the cluster.

The 2015 WPI project expanded upon the previous two by using a distributed graph database and visualization engine to make data viewing easier. The students did this with the use of Titan, a distributed graph database with Cassandra as the backend storage system. The Titan database was able to quickly retrieve nodes in the graph and compute any extra attributes as well. For visualization, the team used Vis.JS to make a fully interactive graphical solution that worked cross-browser. Below is a sample of this solution.

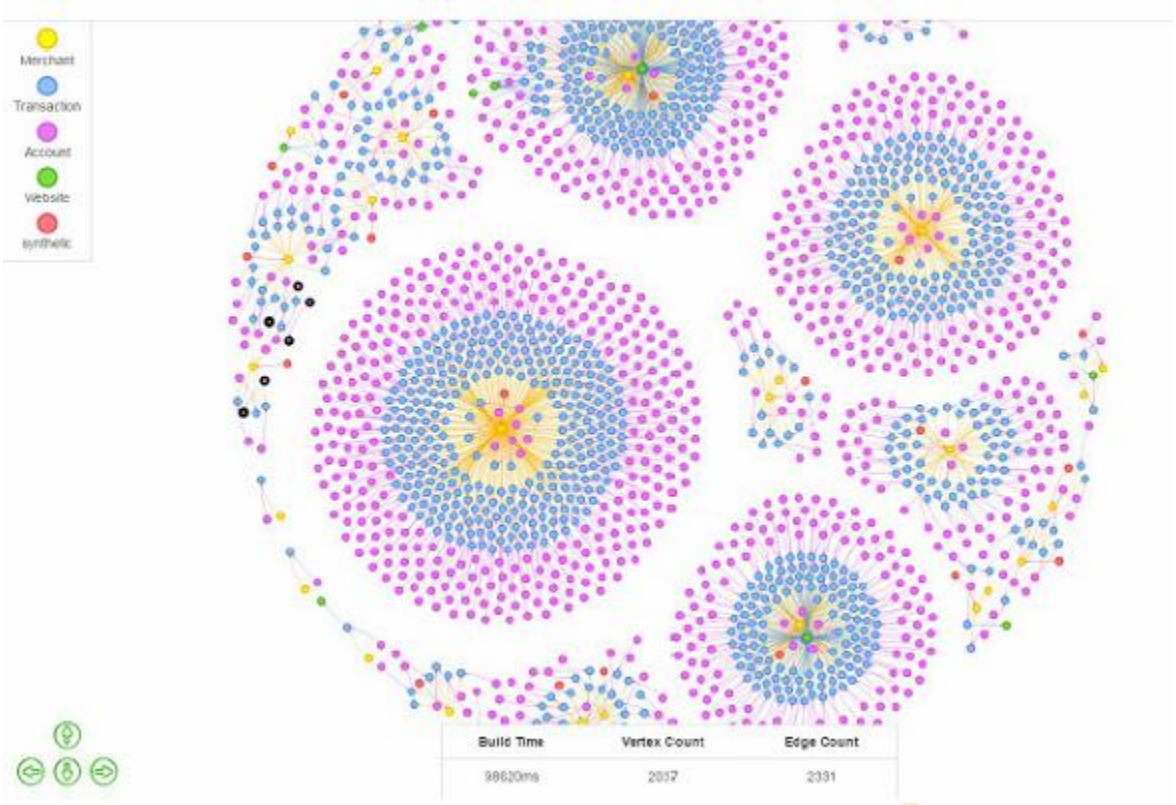


Figure 1.1. Sample graph from 2015 project. (Baia, et al., 2015)

Each of the colored dots in the graph is a node which represents a specific customer, transaction, retailer, account, or website. Each line in the graph links these nodes based on the input transactions. Rather than trying to find anomalies in the raw transaction data, Data Analysts can now use this tool to visually analyze the patterns that exist in the data and search for outliers to find fraudulent data and examine behavior.

As a follow-up to the 2015 project, our team's primary objective was to develop a pipeline that would be capable of anonymizing transaction data that can then be safely and quickly used by ACI and any third-party companies as an input in applications like fraud detection without fear of breaking any law. Statistical analysis and machine learning were the main methods we researched in our attempt to solve the problem at hand. Eventually, after much deliberation the team decided to focus on statistical methods as our main approach in solving this problem.

CHAPTER 2: BACKGROUND

This section introduces research papers, machine learning libraries and methods, and several useful statistical distributions and methods we examined during the course of the project.

2.1: Literature Review

Our team started off by researching the current privacy protection methods that exist for sensitive information. We found many methods that focused on anonymizing existing data as opposed to generating new unidentifiable data, which was our goal. Our goal can still be accomplished through the use of existing anonymization or clustering methods, however these introduce a level of error and inaccuracy in the data and some data loss would be expected. Below are a few of the academic papers that we looked at.

2.1.1 CASTLE

Most of the existing privacy preserving techniques, such as k-anonymity methods, are designed for static data sets only, which cannot be applied to continuous, transient, and usually unbounded streaming data. Moreover, in streaming applications, there is a need to offer strong guarantees, on the maximum allowed delay between an incoming data and its anonymized output. *CASTLE* (Continuously Anonymizing STreaming data via adaptive cLustEring) (Cao, 2008) is a cluster-based scheme that anonymizes data streams and, at the same time, ensures the freshness of the anonymized data by satisfying specified delay constraints. It can produce k_s-anonymized data streams and avoid security flaws. *CASTLE* can also output anonymized data progressively, and offers better output quality than existing methods.

2.1.2 PCTA

Privacy-Constrained Clustering-Based Transaction data Anonymization, or PCTA, was developed to solve an issue that was similar to this project's main problem, where privacy concerns were preventing useful medical data from being able to be released and used to support applications such as biomedical studies. The research paper also addressed the large amount of information loss that was present in anonymization methods at the time of the study.

PCTA is a framework that performs data generalization on top of a clustering method. It is flexible in that it can support whatever clustering algorithm that is input to it. Because of this, PCTA can produce usable data sets that address a large variety of privacy issues while only allowing a minimal amount of information loss. PCTA proved to produce significantly better results than previous methods based on a single privacy model.

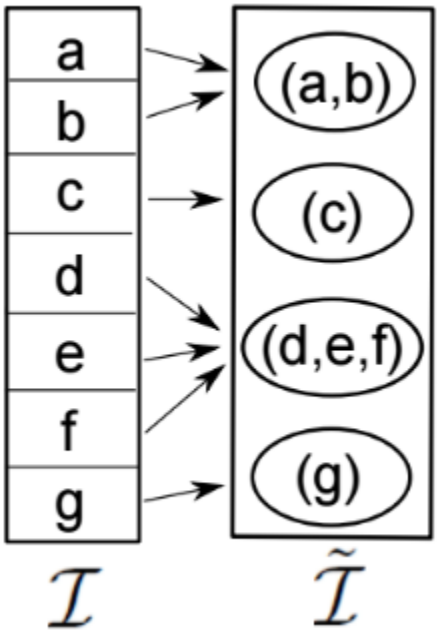


Figure 2.1. Mapping original to generalized items using global generalization. (Gkoulalas-Divanis, 2011)

2.2: Technical Background

2.2.1 Scikit-learn

Scikit-learn is a free software machine learning library with many simple and efficient tools for data mining and data analysis using the Python programming language (Pedregosa, 2011). It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Scikit-learn is largely written in Python, with some core algorithms written in Cython, a superset of Python that allows for C functions and types, that Scikit-learn utilizes to achieve better performance (Cython). Support vector machines are implemented by a Cython wrapper around LIBSVM; logistic regression and linear support vector machines by a similar wrapper around LIBLINEAR. It is open source and commercially usable with the BSD license.

2.2.2 Java-ML

Java Machine Learning Library (Java-ML) is a collection of machine learning algorithm packages such as data manipulation, clustering, feature selection, classification, and statistics. The classification package (net.sf.javaml.classification) provides several classification methods such as Bayes, meta (provides meta-classifiers), and tree (provides classification trees and derivative algorithm). Another useful package is Statistics (net.sf.javaml.utils.Statistics) which implements some of the common distributions, such as Gaussian, a variety of tests, etc. The code is mostly adapted from the CERN Jet Java libraries: Copyright 2001 University of Waikato Copyright 1999 CERN-European Organization for Nuclear Research. Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is therefore

granted without any fees.

2.3: Machine Learning Methods

As one of the potential routes that we could of taken the project, the team researched several different machine learning techniques. The three main methods chosen for research were clustering, naive bayes classifier, and utilizing the apriori algorithm.

2.3.1 Clustering

Clustering, or cluster analysis, is the process of dividing a dataset into groups, or clusters, that contain data that have one or more properties similar with one another. In respect to machine learning, clustering would be defined as an unsupervised learning algorithm as it derives patterns or similarities from datasets with no labeled responses (MathWorks). The similarities are defined through the use of Euclidean or probabilistic distance. There are a variety of clustering algorithms each with their own advantages and disadvantages. Regardless of the type of clustering algorithm, there are four standards that need to be kept in mind when using a clustering algorithm (Jain, 2010).

First, the data representation, or the features that you select to cluster around, must be the best representation of the dataset that is being grouped. A bad feature selection would not create clusters that represent the entire dataset. Secondly, the reason behind grouping a specific cluster together must be in line with how the groupings are selected as having different weights on the different features would lead to different cluster representations of the data. Next, the number of clusters needs to be decided to ensure that the proper depth of the dataset is being shown through the clusters. Finally, the validity of the clusters need to be verified. In other words, a cluster can only occur if a particular cluster can be made around a specified feature (Jain, 2010).

2.3.2 Naive Bayes Classifier

The Naive Bayes Classifier is a supervised learning family of algorithms that applies Bayes' theorem on a dataset and assumes the features of the data being classified are independent of one another. Naive Bayes Models can be easily built from large datasets due to the simplicity of the algorithms themselves. Naive Bayes Classifiers also perform really well on datasets with dependent features due in part to optimality not being necessarily dependent on whether or not the assumptions on the independence between the features are appropriate. A classifier is optimal if the actual and estimated distributions agree on the same most-probable class (Rish, 2001).

2.3.3 Apriori Algorithm

The Apriori Algorithm searches through a dataset for data points that appear frequently throughout and creates boolean association rules with those data points using two criteria: the minimum confidence and the minimum support. Association rules are statements that show relationships between different data in a database. One of the criteria, minimum confidence, is a specific percentage that the association rules generated must surpass in order to be accepted as a strong association rule. The other criteria, minimum support, is the minimum amount a data point has to appear in a dataset in order to be considered as part of the set used to create association rules. (Wasilewska)

2.4: Statistical Distributions

In order to model large volumes of transaction data, we needed to research several different statistical distributions to determine the most optimal approach. For the purposes of the project, the team focused on five main statistical distributions which were the following: Gaussian, Poisson, Beta, Conditional, and Marginal Distributions.

2.4.1 Gaussian Distributions

Gaussian distributions, commonly known as the normal distribution, organize data sets into a binomial distribution that centers around a mean value. This distribution is commonly shown to be in the shape of a "bell curve" since the majority of the data falls within one standard deviation from the mean which naturally creates a bell like curve. The gaussian distribution is the most common type of distribution and can be found to represent many things in our daily lives. Some common examples would be the heights of people, IQ scores, blood pressure and the like (StatisticsHowTo, n.d.). The equation used to model a gaussian curve is shown in Figure 2.2. The notation is as follows: sigma (σ) is the standard deviation of the data being modeled, μ is the mean value derived from the data, and x is a normal random variable (StatisticsHowTo, n.d.).

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\sigma^2\pi}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Figure 2.2. Gaussian curve formula.

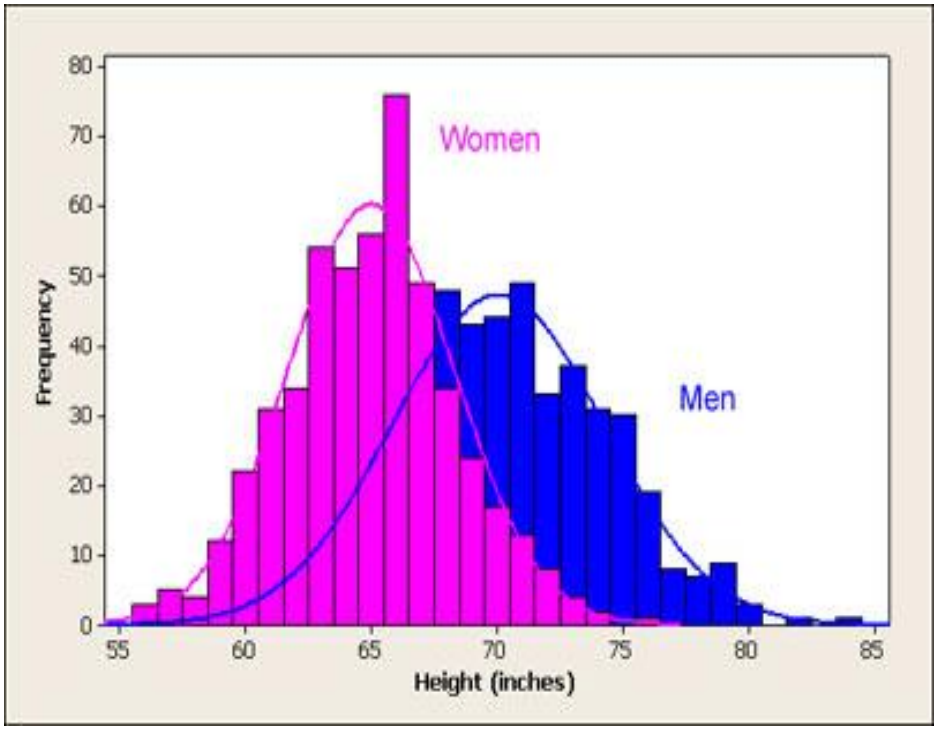


Figure 2.3. Gaussian graph of heights from men and women (Sauro, n.d.)

2.4.2 Multivariate Gaussian Distribution

As the number of columns in a dataset is always greater than one, it's indispensable to implement multivariate normal distribution dealing with multi-dimension case. Multivariate normal distribution or multivariate Gaussian distribution, is a generalization of the one-dimensional (univariate) normal distribution to higher dimensions. One possible definition is that a random vector is said to be k-variate normally distributed if every linear combination of its k components has a univariate normal distribution. It is often used to describe, at least approximately, any set of (possibly) correlated real-valued random variables each of which cluster around a mean value.

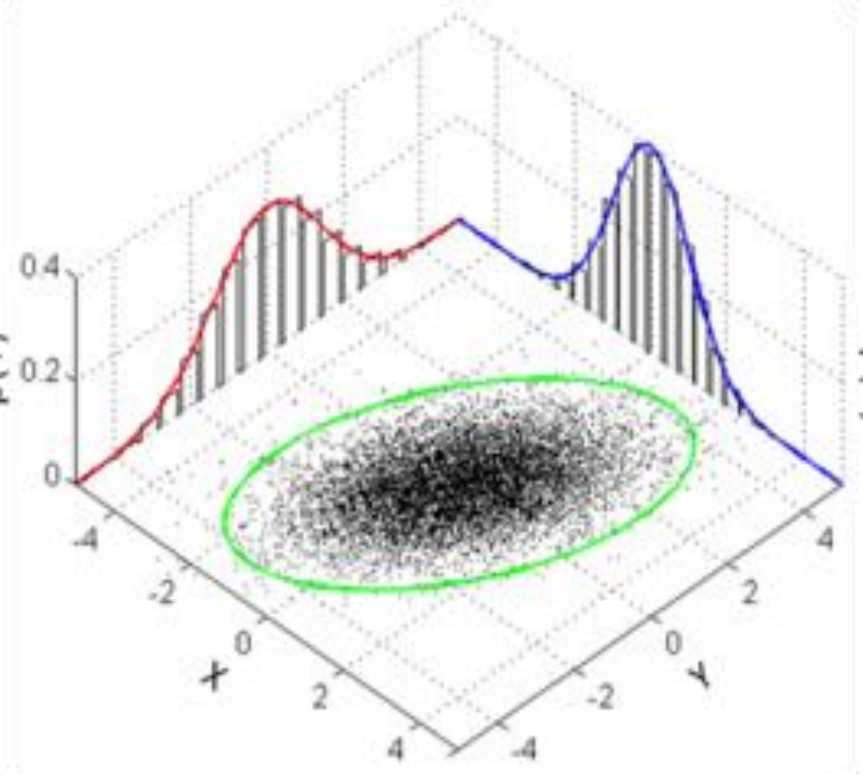


Figure 2.4. Multivariate Gaussian distribution with the 3-sigma ellipse, the two marginal distributions, and the two histograms.

2.4.3 Poisson Distributions

Poisson distributions are typically used to model datasets that are non-normal. The poisson distribution gives the probability of a particular event occurring based on past information (StatisticsHowTo, n.d.). Some clear examples of how a poisson distribution could be utilized would be to forecast future profits for businesses and to determine the most optimal time to avoid rush hour traffic. The poisson function is $P(x; \lambda) = (e^{-\lambda}) * (\lambda^x)\ x!$. The notation is as follows: λ is the expected number of occurrences within the interval and x is the actual number of occurrences. Some special properties of the poisson distribution is that the mean and variance are equal to λ.

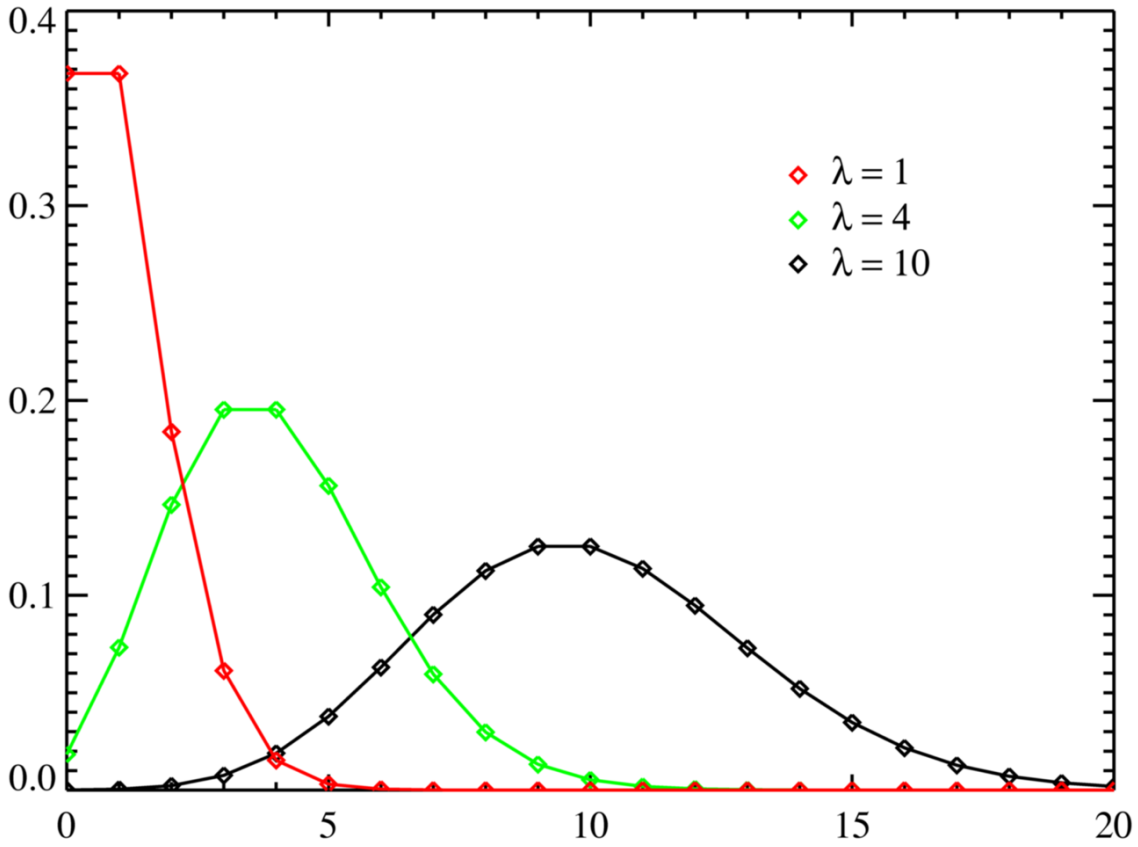


Figure 2.5. Example poisson graph (StatisticsHowTo, n.d.).

2.4.4 Beta Distributions

A Beta distribution is normally used to represent the outcomes of percentages or proportions. In other words, the beta distribution shows a probability distribution of probabilities for situations where the probability is unknown but can be guessed. For example, a beta distribution is a good predictor of a baseball player's batting average. A player's batting average obviously changes across the season and with previous expectations to fall within the average batting record, an initial beta distribution can be drawn and then updated as new information is available (Robinson, David). One form of the probability density function for beta distributions is $x^{\alpha-1}(1-x)^{\beta-1}B(\alpha,\beta)$ (StatisticsHowTo, n.d.).

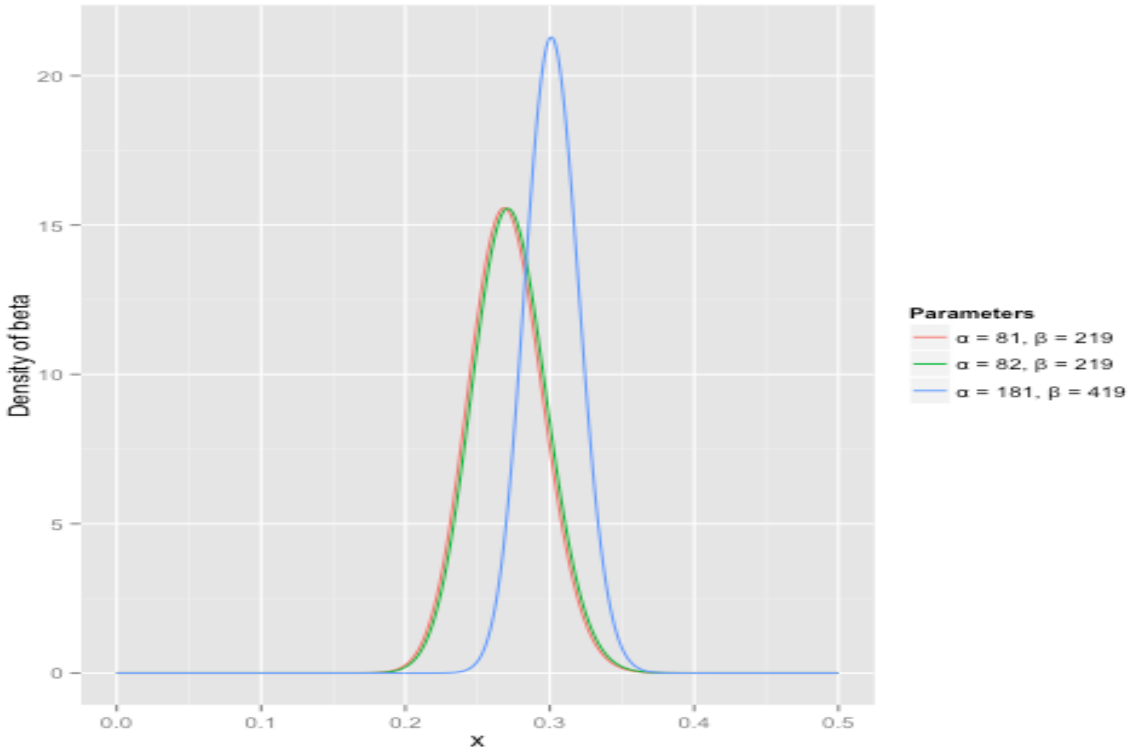


Figure 2.6. Example beta distribution graph. (Robinson, David)

2.4.5 Conditional Distributions

Conditional probability distribution models the probability of an event occurring in regards to one or more other events. Due to how conditional probability can support more in depth probability statements, multi-event predictions can be made. As seen in the example from Figure 2.6, the probabilities of a man and woman, of an unknown sample size, owning or not owning a pet are displayed. With the knowledge from the table, anyone can calculate complicated predictions like if a person does not own a pet what is the probability that they are female. The function for conditional probability is P(B|A) = P(A and B) / P(A) (StatisticsHowTo, n.d.).

	Have pets	Do not have pets	Total
Male	0.41	0.08	0.49
Female	0.45	0.06	0.51
Total	0.86	0.14	1

Figure 2.7. Probability of either gender having a pet (StatisticsHowTo, n.d.)

2.4.6 Marginal Distributions

Marginal probability distributions model the independent probability distributions between two random variables. Differing from conditional distributions, marginal distributions only display the sum probabilities of one random variable at a time. For example, in Figure 2.7 the different pet preferences can be turned into probabilities by individually dividing the different pet totals over the total amount of pets. One common way to show a marginal probability distribution is through the use of a frequency distribution table (StatisticsHowTo, n.d.). The other way to calculate marginal probability is to use the marginal probability distribution functions which are $g(x) = \Sigma y\ f(x,y)$ and $h(y) = \Sigma x\ f(x,y)$.

	Cats	Fish	Dogs	
Men	2	4	6	12
Women	5	3	2	10
	7	7	8	22

Figure 2.8. The frequency of different pets between men and women.

2.5: Hortonwork's Ambari Distribution

Apache Ambari is an open-source management platform for monitoring Apache Hadoop clusters. The use of Ambari simplifies any manual effort in installing and maintaining a Hadoop Cluster. The list of software that the Ambari distribution offers will be discussed in more detail

in the methodology section.

2.5.1 Hadoop Cluster

Apache Hadoop is an open-source software that allows for distributed and scalable processing of large datasets. Hadoop itself contains the following modules: the Hadoop Common, Hadoop Distributed File System, Hadoop MapReduce, and Hadoop YARN. Hadoop Common contains the utilities that the other modules all need to function. The HDFS module is used for its speed in accessing the application data when performing operations. Hadoop YARN is a module that handles the job scheduling and the resource management for the cluster. Finally, Hadoop MapReduce is the module that performs parallel processing of large datasets (Hadoop, 2016).

2.5.2 Spark

Apache Spark originated from a 2009 research project from the University of California Berkeley AMPLab under the name of Mesos. The Mesos project was a cluster management framework that aimed to support a variety of cluster computing systems (Phatak, 2015). Apache Spark, originally just called Spark, was built off of the Mesos project to include an emphasis on "interactive iterative computations" like machine learning to aid in cluster computing. The team behind Spark later donated the project, in early 2010, to the Apache Software Foundation who open sourced it (Phatak, 2015).

Apache Spark offers various advantages in processing big data over its competitors. The engine provides a unified framework to handle big data through offering support for a large variety of datasets, such as text data and graph data, and for working with different sources of

data, such as data batches or real-time streaming data. Spark also cleanly integrates with other cluster processing engines like Hadoop and can also process data obtained from a variety of different databases like Cassandra and HDFS. Spark has also been shown to run applications 100 times faster in memory on Hadoop clusters and ten times faster when running the applications on disk. Spark's other features include the support for SQL queries, graph data processing, and machine learning (Phatak, 2015).

2.5.3 Apache Phoenix and HBase

Apache HBase is an open source project under the care of the Apache Software foundation. HBase itself is a NoSQL database designed to store large amounts of data. The database is primarily paired with software platforms that perform calculations on these vast data sets, like Apache Hadoop. Due to the fact that HBase is a NoSQL database, the structure of the database table and the data stored within differ from the conventional SQL type databases (George).

Although Apache HBase does utilize the column-oriented format when storing data on disk, HBase is not a column-oriented database in the traditional relational database management system sense. A traditional relational database provides real-time analytical access to the stored data while HBase focuses on providing access to specific cells of data instead (George).

Paired nicely with HBase, Apache Phoenix is a SQL wrapper that enables the use of many familiar SQL commands to interact with the Apache HBase database on a Hadoop cluster. Phoenix also naturally integrates with any system that utilizes Hadoop technology like Apache Spark. Due to the integration of native map-reduce support, scans of large datasets from HBase would be reduced to mere seconds using phoenix (Apache Software Foundation).

2.5.4 Apache Commons Math

Apache Commons Math is a useful library under the Apache Software Foundation that provides a variety of mathematical tools for usage in a number of different scenarios. The main draw of this particular library is the abundance of statistical and mathematical operations it contains. The classes of particular interest in this library were specifically the Multivariate Normal Distribution, RealMatrix and RealVector as they provided the necessary functions and structures for transaction synthesis pipeline. In addition to its vast repertoire of classes, the Apache Commons Math package was also designed to be easily integratable with no little to no external dependencies needed which allows for easy usage (Apache Software Foundation, 2016). Since this library contained a number of convenient statistical structures and functions, the team decided to heavily utilize this library. The below are the classes and the functions we utilized for our pipeline.

3 METHODOLOGY

3.1 Goals and Objectives

In order to achieve our goal of addressing privacy concerns in using real data and transactions to test fraud detection software, our team divided the project into stages. These stages include the Ingestion Engine, Preprocessing, Distribution generation, Data Synthesis, and Validation. After ingesting the transaction data and performing preprocessing to prepare the data for learning we created models to represent the patterns that exist in the data. These models allowed us to generate new synthesized data that behaves and looks the same as the real data, and therefore is suitable to use for testing purposes. This model could also possibly be used to predict, with a fair degree of accuracy, the purchasing behavior of a particular customer and check if a new transaction fits their usual behavior pattern. Simplicity was a focus the team decided on and thus the following methodology would reflect this philosophy.

3.2 Project Architecture

In the backend of our project, we created a Hadoop cluster using the ACI resources allowed to us using three Linux server machines. We also implemented a distributed database using HBase, which was included in the Hadoop installation.

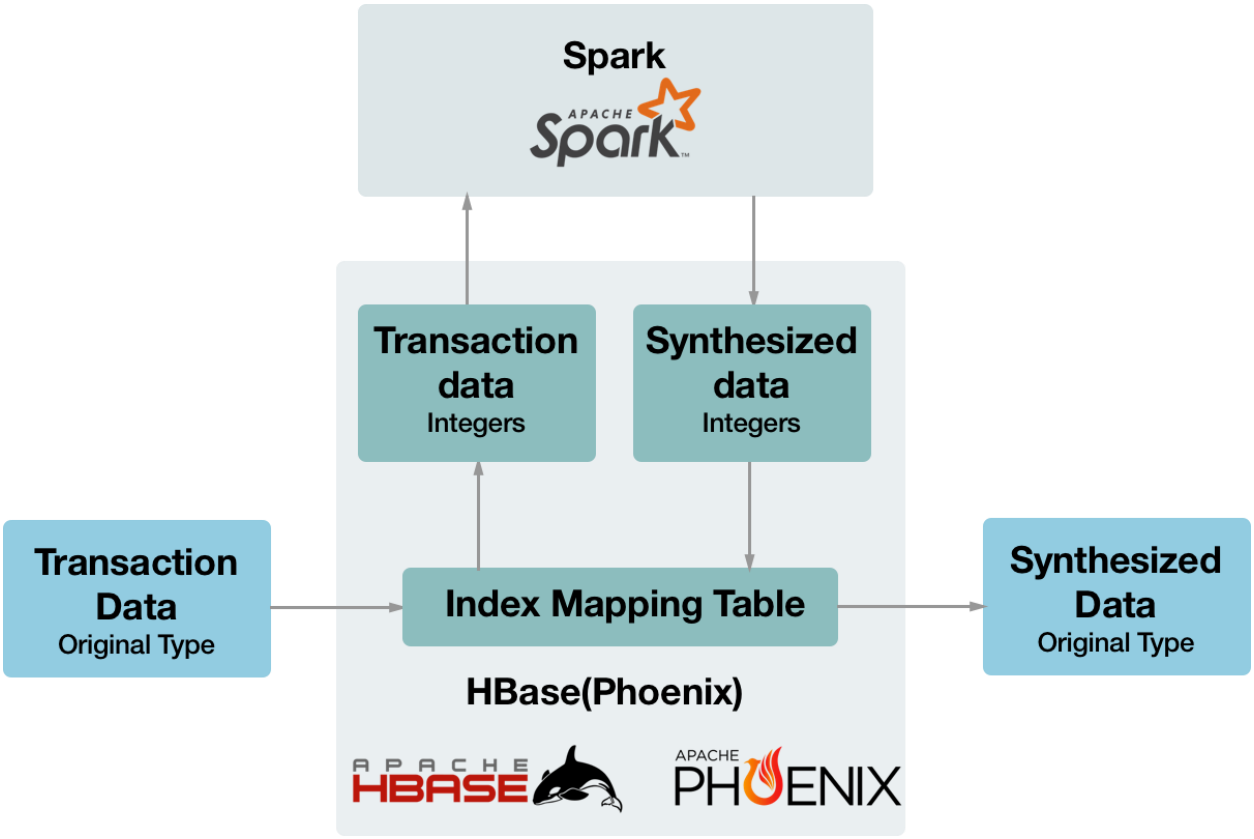


Figure 3.1. Project architecture.

Hadoop Cluster

The system is hosted on a Hadoop cluster that contains one master node and two different slave nodes. These nodes are on three separate RedHat Linux Servers. Each individual machine contains separate components. This diagram shows an overview of the different components.

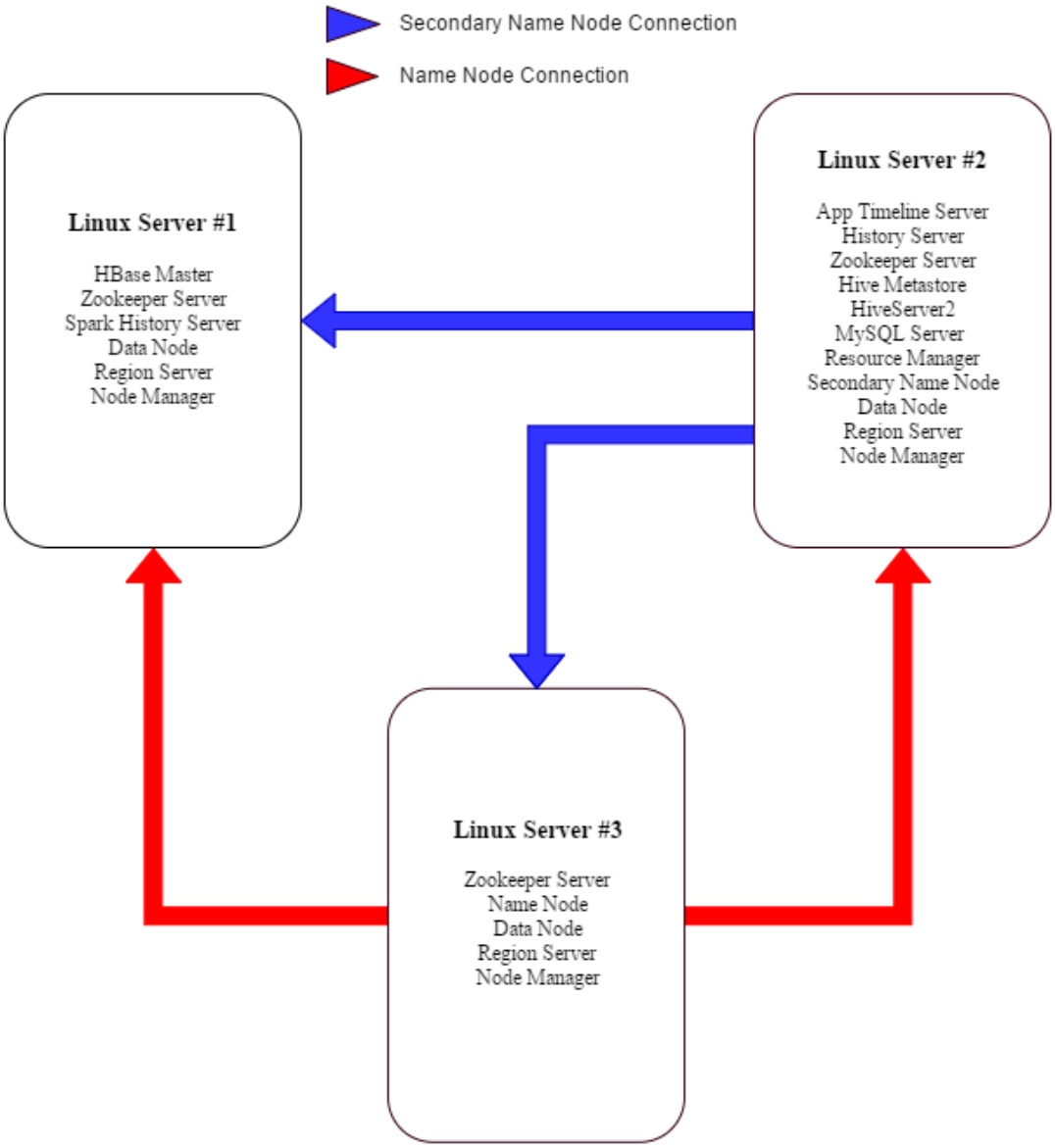


Figure 3.2. Hadoop cluster structure.

HBase Master Server - This server stores all of the HBase database information and allows operations on the data.

    - **Region Server** - The slave servers to the HBase Master

- **Namenode** - The master server where commands are issued to the datanodes.

    - **Datanode** - The slave machines that Hadoop uses in its distributed calculations.
    - **SNamenode** - The secondary namenode that would become active if the main namenode became unusable.
- **NodeManager** - Monitors and acts as the gatekeeper for the traffic that each server machine handles.
- **ResourceManager** - Manages the queue that allocates resources to the jobs sent into the cluster.
- **Hive** - Data storage infrastructure that is built over Hadoop. Allows for an SQL-like interface to query any stored data.
    - **Hive Metastore** - Storage for Hive metadata.
- **History Server** - Contain the results of the jobs ran on the Hadoop cluster.
- **Zookeeper Server** - Centralized service for providing distributed synchronization, maintaining configuration information, naming, and providing general group services.

3.2.1 Design

The overall outline of our project is as follows:

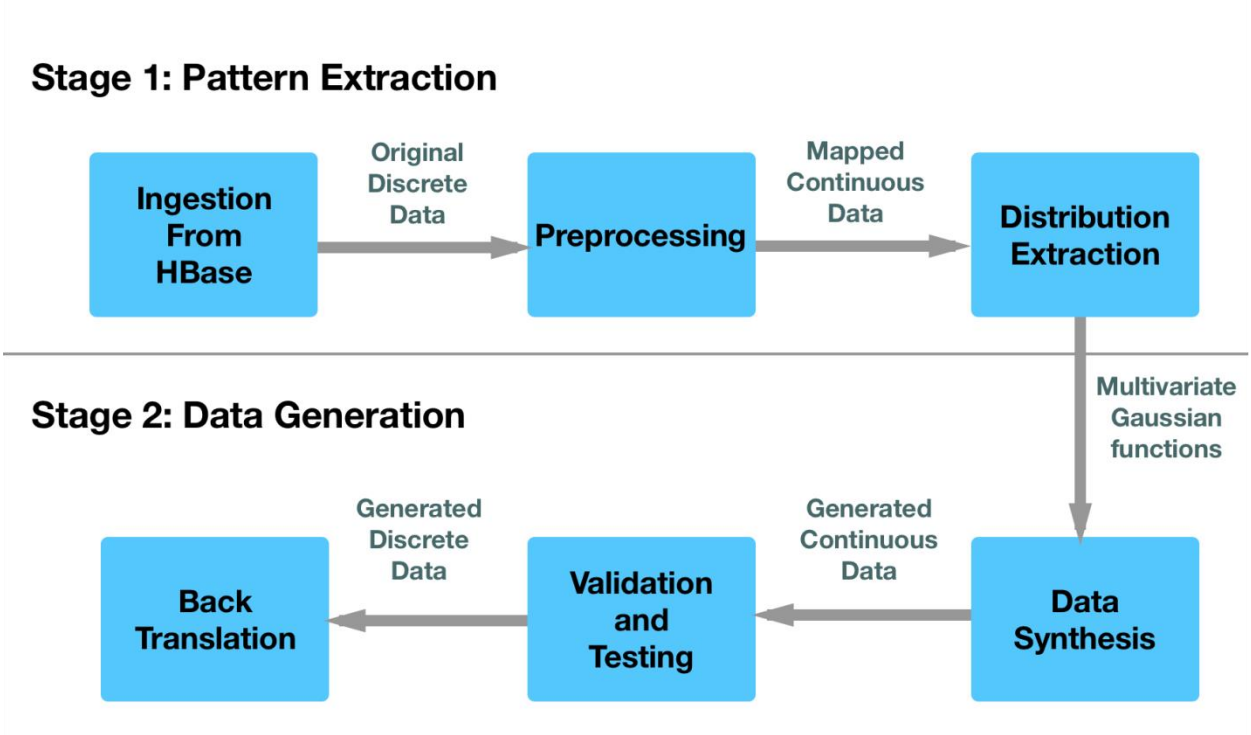


Figure 3.3. Project outline.

3.3 Ingestion Engine

The Ingestion Engine is a program written in Java that reads in given test data and uploads it to the HBase database with the use of Apache Phoenix. The data we worked with throughout the project was already "scrubbed", and did not contain any real personal or private information. However, the project can easily be expanded to use real data by simply uploading any arbitrary amount of information to the HBase server.

3.3.1 What does the data look like

The data being processed were transaction records. Each transaction had around 200

columns associated with it, which covered everything from the transaction amount and customer information, all the way down to the specific terminal and location the transaction was performed at. The large data set was unformatted and in a text file. Since it was not feasible for us to use every attribute of a transaction, we worked with our advisors to pick a list of about 30 columns that seemed the most useful. Each column is represented as an object in a JSON file that we adapted from the previous year's project, and is described by a line offset and a length (each transaction being on one line ~1100 characters long). Our program took in this JSON and parsed only the data in those specified columns to upload into a table in the database, reducing the amount of storage space required on the HBase server.

3.4 Data Preprocessing

Data pre-processing is an important step in the data mining process. The phrase "garbage in, garbage out" is particularly applicable to data mining and machine learning projects. Data gathering methods are often loosely controlled, resulting in out-of-range values (e.g., Income: -100), impossible data combinations (e.g., Sex: Male, Pregnant: Yes), missing values, etc. Analyzing data that has not been carefully screened for such problems can produce misleading results. Thus, the representation and quality of data is the most urgent concern before running any sort of analysis. (Pyle, 1999)

3.5 Index Mapping Table

3.5.1 Index Mapping Table Overview

Most of the columns in datasets obtained from ACI Worldwide are discrete such as zip code, fraud index, transaction date, first name and last name of customers, so it's impossible to

analyze these columns straightforwardly. In order to learn the frequency distributions and correlativity on not only continuous data columns such as transaction amount but also on these discrete columns, it's expedient to build an index mapping table to store those discrete columns of data, and translate them to continuous data. With the support of Apache Phoenix, which takes SQL queries, compiles them into a series of HBase scans, and orchestrates the running of those scans to produce regular JDBC result sets, it works perfectly that all SQL queries can be implemented in HBase, which is a type of NoSQL database. The primary reason for selecting HBase to store the data ingested by the Ingestion Engine was the easy integration into a Hadoop Cluster.

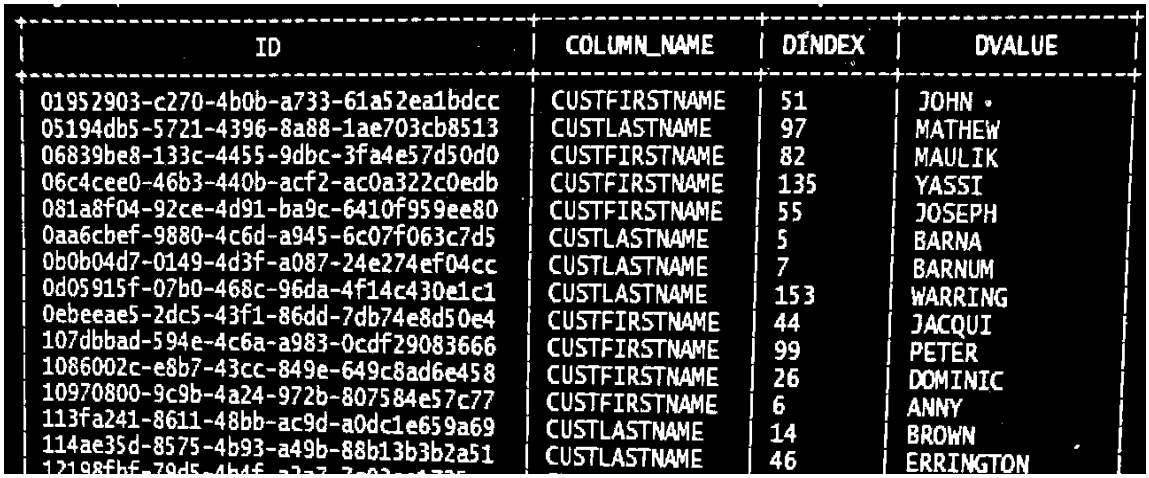
3.5.2 Index Mapping Table Features

The index mapping table contains two main features: storing original data values as indices, which can be used efficiently in analyzing the original data distribution and translating indices back to original data values.

3.5.2.1 Storing Original Data Values As Indices

In order to analyze discrete columns in the transaction dataset, the first step is to translate them into continuous columns. The way we approach this process is using indices to represent discrete values.

Figure 3.6 shows how the index mapping table is used for storing original discrete data values as indices. It has four columns in total: The first column is the unique ID which Hbase requires as an identifier for each independent row, the second column stores the column name that the data belongs to, the third column stores the index number of the data from 1 to the last number, the last column stores the original value of this data.



ID	COLUMN_NAME	DINDEX	DVALUE
01952903-c270-4b0b-a733-61a52ea1bdcc	CUSTFIRSTNAME	51	JOHN •
05194db5-5721-4396-8a88-1ae703cb8513	CUSTLASTNAME	97	MATHEW
06839be8-133c-4455-9dbc-3fa4e57d50d0	CUSTFIRSTNAME	82	MAULIK
06c4cee0-46b3-440b-acf2-ac0a322c0edb	CUSTFIRSTNAME	135	YASSI
081a8f04-92ce-4d91-ba9c-6410f959ee80	CUSTFIRSTNAME	55	JOSEPH
0aa6cbef-9880-4c6d-a945-6c07f063c7d5	CUSTLASTNAME	5	BARNA
0b0b04d7-0149-4d3f-a087-24e274ef04cc	CUSTLASTNAME	7	BARNUM
0d05915f-07b0-468c-96da-4f14c430e1c1	CUSTLASTNAME	153	WARRING
0ebeeae5-2dc5-43f1-86dd-7db74e8d50e4	CUSTFIRSTNAME	44	JACQUI
107dbbad-594e-4c6a-a983-0cdf29083666	CUSTFIRSTNAME	99	PETER
1086002c-e8b7-43cc-849e-649c8ad6e458	CUSTFIRSTNAME	26	DOMINIC
10970800-9c9b-4a24-972b-807584e57c77	CUSTFIRSTNAME	6	ANNY
113fa241-8611-48bb-ac9d-a0dc1e659a69	CUSTLASTNAME	14	BROWN
114ae35d-8575-4b93-a49b-88b13b3b2a51	CUSTLASTNAME	46	ERRINGTON

Figure 3.4. Index mapping table working in HBase.

In order to optimize the storage space index mapping table required in HBase, our team finally remove the unique ID column and add one more TRX_ID (Transaction ID) column as a more semantic identifier for each row of data.

3.5.2.2 Translating Indices Back To Original Value

Because the newly synthesized data which is generated based on the typical distribution can only stay in an index version, in order to understand what the actual value is behind the corresponding index, it is vital to translate the index back into the original data value.

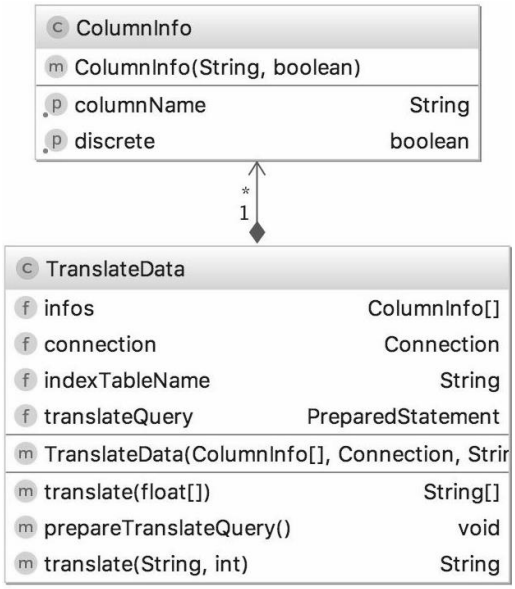


Figure 3.5. Translating index back to original data diagram.

Transaction data has two types of columns, discrete and continuous. As mentioned before, a majority of those columns are discrete, such as “CARD_BIN”, “ACCTNUM” (Account Number), and “CARD_TYPE”. But there are still columns with continuous data type such as “TRX_AMOUNT” (Transaction Amount).When translating generated data values which are in index form back to original data values, it is essential to treat continuous columns and discrete columns as two separate situations: For discrete columns, the index mapping table will straightforwardly map them back to their original values and original data types. For continuous columns, the index mapping table will keep them in the same data type and value as generated.

3.6 Column Analysis

When beginning to create models for the data, our group first implemented the use of simple statistical methods. We started by looking at discrete columns that were represented as a string or integer (Card type, merchant country, MCC code, etc.) and performed a frequency count of the unique values in each of these columns. We then tried to fit the data to a curve, one

such Gaussian fit attempt is shown below for the MCC code (the x-axis values are not representative of the actual MCC codes).

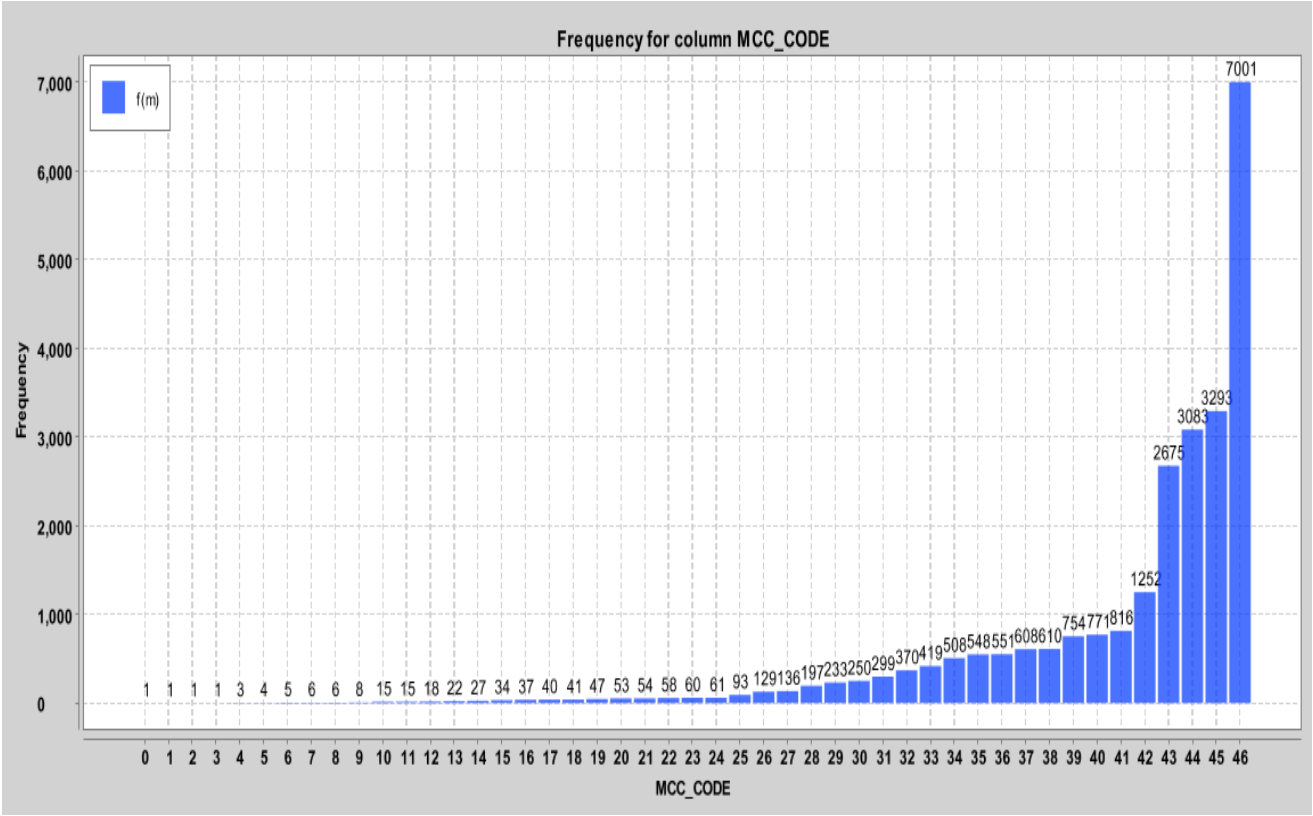


Figure 3.6. Frequency counts of each unique MCC code in the test data.

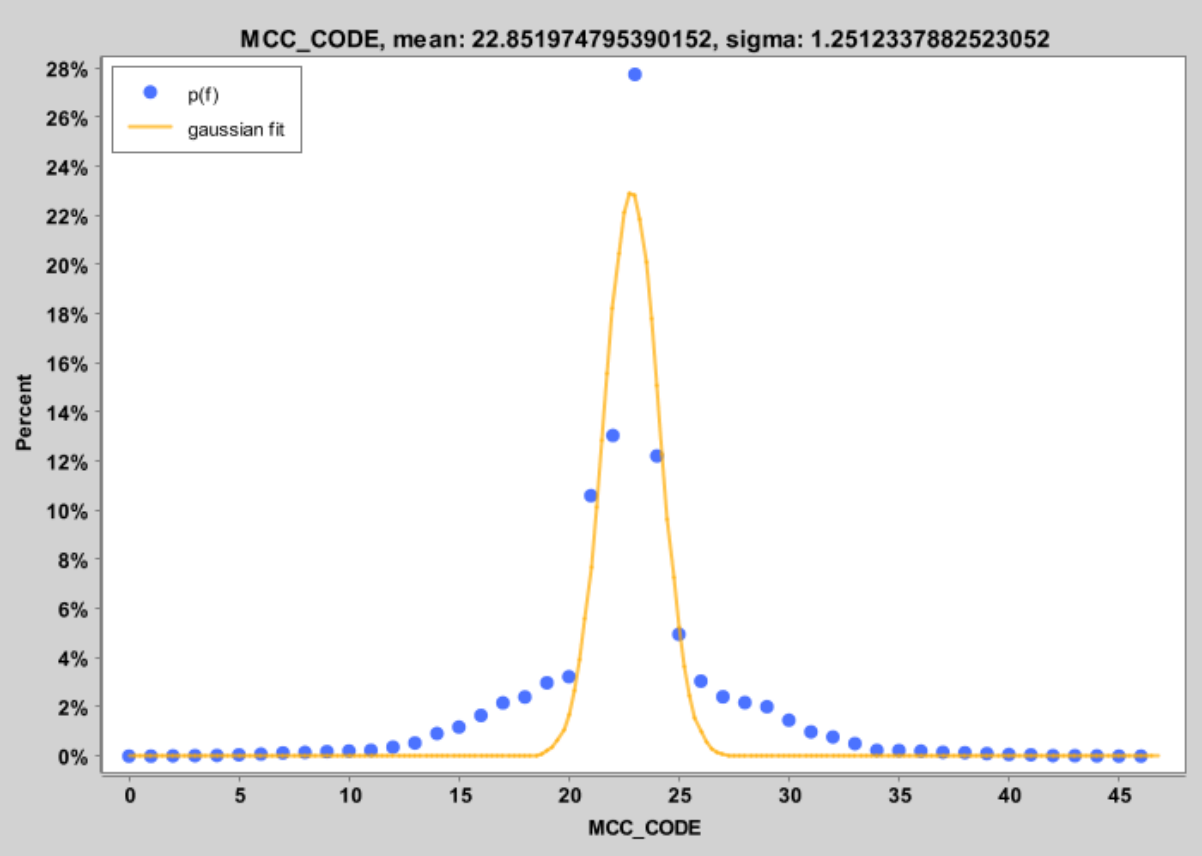


Figure 3.7. Attempting to fit MCC code frequencies to a Gaussian curve.

While these methods were not the most accurate representations of the real data, they did help our group to understand what the data we were working with looked like and how it behaved. This method also assumed that the columns being used were independent of each other, which was most likely not the case. For an in-depth look into the columns the team analyzed, refer to 4 COLUMN ANALYSIS.

Then, we moved on to modeling the data using multivariate Gaussian distributions. While this does take multiple columns into account, it again falls short in that it forces the data to a best-fit Gaussian curve which may, in many cases, not accurately represent the data. Our group used functions packaged into Apache's Math library to calculate the covariance and mean matrices of the columns we chose. We then created a new multivariate Gaussian distribution

using these matrices as the parameters and then generated a number of random samples using that distribution.

After creating the new multivariate Gaussian distribution, we used them to synthesize new data. Due to the issues with utilizing Apache Math's random sample method, the process we used was as follows. Cholesky decomposition was performed on the covariance matrix of the new multivariate Gaussian distribution and then the lower triangular matrix was calculated with the help of the existing Java library, JAMA. Using the Apache Math vector and matrix libraries, the lower triangular matrix was transformed by being multiplied by a vector of random normal vectors with the same length as there are of columns in the covariance matrix. Finally, the resultant vector is multiplied by the mean matrix, now converted into a matrix, to return a vector that made up one row of data. This procedure is repeated until the number of rows matched the number of rows in the original dataset (Seydel, 2012). The data we produced was in the indexed continuous format which was not understandable, so we reused the index mapping table to translate all of the data back to their original types.

Performing the covariance and mean matrix calculations on this newly generated data, we compared them to the original to see how accurate this synthesized data was. A sample output of this is shown in Figure 3.9 below with the covariance matrix from the original data on top and the covariance matrix of the synthesized data on the bottom for comparison. The two mean matrices are also shown in the figure at the very bottom.

```
Generated 1000 lines of fake data.
Covariance matrices:
50326.75 -798.22 1181.31 1039.22 31419.22
-798.22 2917431.05 -22420.47 -26620.21 -1394390.12
1181.31 -22420.47 318684.93 33367.64 180107.74
1039.22 -26620.21 33367.64 161800.66 140712.21
31419.22 -1394390.12 180107.74 140712.21 7122762.90

51333.83 -3078.56 373.27 5995.69 9177.49
-3078.56 3120022.61 -928.05 -36858.00 -1374168.05
373.27 -928.05 333104.57 40743.80 86485.57
5995.69 -36858.00 40743.80 157510.30 160038.34
9177.49 -1374168.05 86485.57 160038.34 6940139.55
Mean matrices:
128.87 5165.89 9967.03 9982.79 9227.07
133.91 5192.95 9977.45 9982.17 9305.47
```

Figure 3.8. Sample output of multivariate Gaussian distribution calculation.

To test the validity of these Multivariate Gaussian models, we created the validation suite in the section below.

3.7 Validation Methods

3.7.1 KL Divergence

In probability and information theory, the Kullback-Leibler divergence, (Kullback, 1951) (Kullback, 1959) also called discrimination information, information divergence, information gain, KL divergence, is a measure of the difference between two probability distributions P and Q. In applications, P typically represents the "true" distribution of data, observations, or a precisely calculated theoretical distribution, while Q typically represents a theory, model, description, or approximation of P. During current implementation, P here represents the distribution of original data, while Q represents the distribution of synthesized data.

Specifically, the Kullback-Leibler divergence from Q to P, denoted $D_{KL}(P \parallel Q)$, is a

measure of the information gained when one revises one's beliefs from the prior probability distribution Q to the posterior probability distribution P. In other words, it is the amount of information lost when Q is used to approximate P. (Burnham, 2002) KL divergence has different definitions on P and Q over different cases such as discrete probability distributions and continuous random variables. But the only situation here need to be considered is discrete probability distributions. In this case, the Kullback–Leibler divergence from *Q* to *P* is defined to be (MacKay, 2003):

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i)\, \log \frac{P(i)}{Q(i)}.$$

Figure 3.9. Discrete probability distribution for KL divergence.

In words, it is the expectation of the logarithmic difference between the probabilities of P and Q, where the expectation is taken using the probabilities of P. The KL divergence is defined only if $Q(i)=0$ implies $P(i)=0$, for all i (absolute continuity). Whenever $P(i)$ is zero the contribution of the i-th term is interpreted as zero.

3.7.2 Covariance Matrix

In probability theory and statistics, a covariance matrix (also known as dispersion matrix) is a matrix whose element in the (i, j) position is the covariance between the i^{th} and j^{th} elements of a random vector. A random vector is a random variable with multiple dimensions. Each element of the vector is a scalar random variable. Each element has either a finite number of observed empirical values or a finite or infinite number of potential values. The potential values are specified by a theoretical joint probability distribution.

Intuitively, the covariance matrix generalizes the notion of variance to multiple

dimensions. As an example, the variation in a collection of random points in two-dimensional space cannot be characterized fully by a single number, nor would the variances in the x and y directions contain all of the necessary information; a 2x2 matrix would be necessary to fully characterize the two-dimensional variation.

In order to compare the behavior of synthesized data and original data, it's crucial to compare the covariance matrix.

3.8 Run Time Optimization

In building the model generation portion of our project, we first started out using Apache Math's built in distribution framework and Multivariate Normal Distribution for data sampling. While these methods were easy to use, and worked effectively, they were only run on a single local virtual machine on ACI's network. These methods were not built for the vast amounts of data or columns that ACI possesses and therefore did not scale well when the amount of test data was increased.

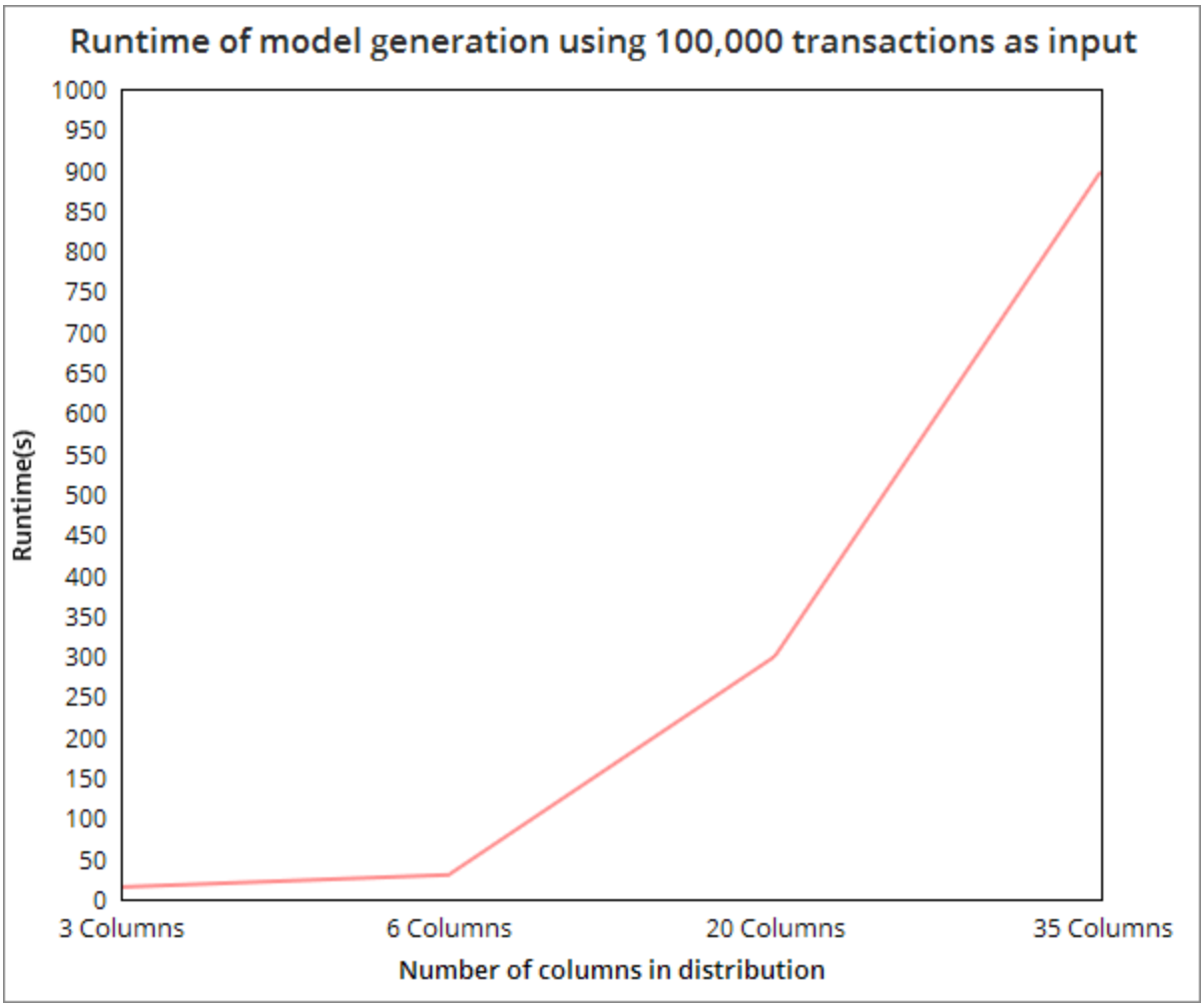


Figure 3.10. Runtime without optimization.

The runtime of our project when using 100,000 transactions as input and changing the number of columns in the model generated is shown above in Figure 3.10. Our group therefore searched for ways to improve runtime and the scalability of the calculations performed. Because we already had a Hadoop server setup, we wanted to look into its MapReduce capabilities in order to be able to submit a job and have it run distributed on the Hadoop machines. In the end, the code was refactored to utilize Apache Spark's existing MLlib functions in calculating the distributions. By refactoring our code towards a job to be submitted on a Hadoop machine, our project would be horizontally scalable depending on the number of machines that are in the

server.

3.8.1 Data Ingestion

Up until this point, our group had been using Apache's Phoenix plugin for Java that provides a SQL interface on top of Hadoop. Because the local virtual machine and the server machine that our HBase database was run on were located on separate machines, this led to increasingly long read times from the server as the amount of test data was increased. Our group already knew that Spark was optimized to make use of MapReduce methods, but as we looked more into Spark libraries we found that it not only had built-in HBase support but it also used Spark's data structure called an RDD. A Spark RDD, or Resilient Distributed Dataset, behaves in the front end as a data structure similar to an array, but in the back end it uses a distributed framework that has multiple "regions" split across the different machines in the server.

Because of this distributed nature, working with the HBase data using Spark's RDDs not only makes the process scalable, once the job is run on the same Hadoop server that the HBase database lies on, there is virtually no read time as Spark RDDs work with the entire dataset.

3.8.2 Model Generation

Our group also looked into optimizing the time it took for fitting the Gaussian curves and for generating the models. Again, turning to Spark, we refactored what methods we could to reduce our dependency on libraries such as Apache Math. We were able to convert the covariance and mean matrix calculations to use Spark methods. However, switching to the methods provided by Spark the team encountered issues with incorporating Apache Math's random sampling method and turned towards using Cholesky decomposition to generate the random datasets.

4 COLUMN ANALYSIS

This chapter looks at the different columns that our team ran an initial analysis over. We looked at several individual columns in the data that were chosen as being columns that we want in the distributions. The methods we used to analyze these columns were the following. First, we performed a frequency count of each unique string in the column. Then, we attempted to fit that column's values to look like a Gaussian distribution by sorting the highest frequency in the middle and the next highest alternating to the left and right of the center. The frequencies were normalized to percentages, and then a best fit Gaussian distribution was found using Apache Math's curve fitting implementation.

An analysis of the results from each column looked at is below. It is important to note that the data our team was given to use as test data was already "scrubbed." This means that all private information has been removed and this data may not accurately represent the real transactional data that ACI has access to.

4.1 MCC Code

One of the first columns from the dataset we reviewed was the Merchant Category Code (MCC). The MCC code shows the type of merchant the transaction belonged to. For example, the MCC code for a merchant that is a clothing store would differ from a restaurant's merchant code. The MCC code distribution of the dataset we reviewed is displayed in the figure below.

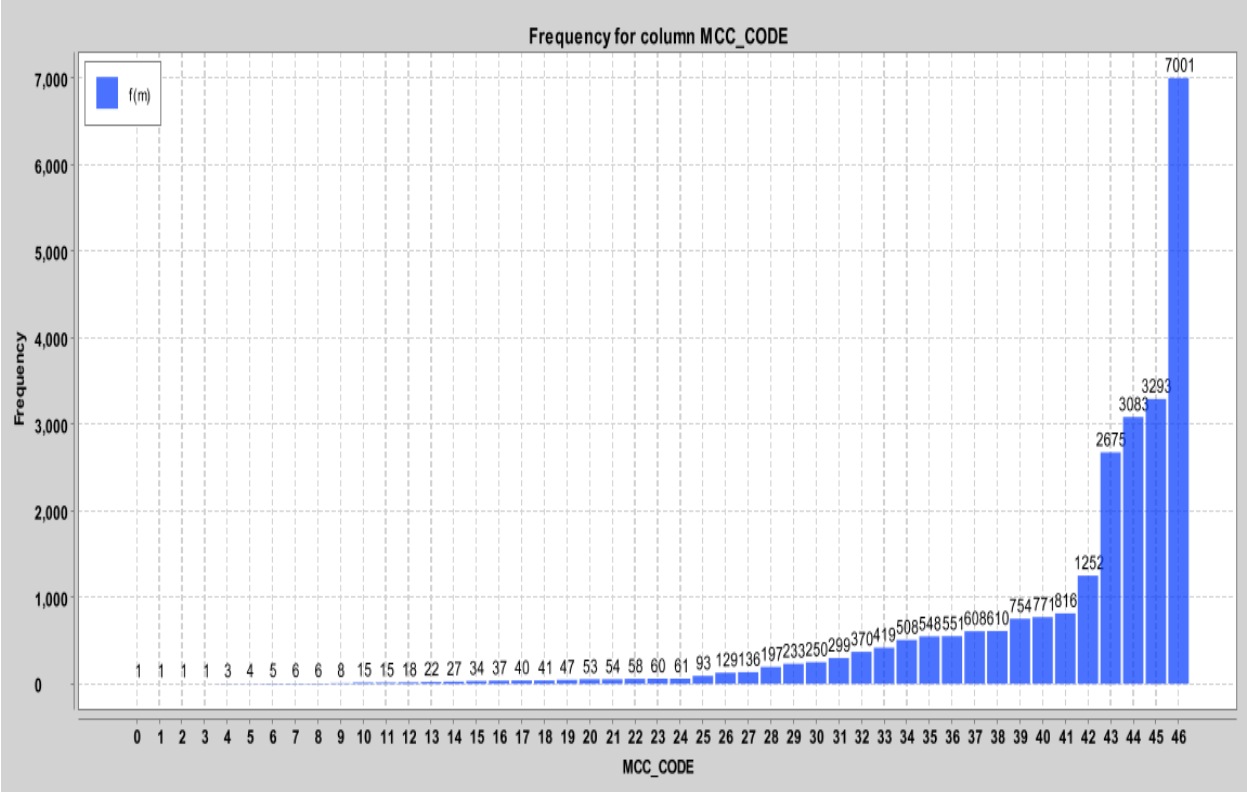


Figure 4.1. Frequency counts of MCC code column.

The above graph shows the frequency distribution of each unique MCC code in the test dataset of 100,000 lines of transaction data. This chart has been sorted, but the order does not make a difference. In this chart, the merchant at the X value 46 had the highest number of occurrences while the lowest number of occurrences is at X value 0. As you can see, a large amount of the transactions (~68%) took place at the top 5 merchants, with the bottom 25 combined adding up to less than 1%. After performing a frequency count of each unique string these values were sorted to look like a Gaussian curve and then normalized to percentages, which can be seen in the figure below. Then, a best fit Gaussian curve was found for these data points using Apache Commons Math. These mappings from the Gaussian curve are the same mappings that are saved to the Index Mapping Table where each X value maps to a unique string in the column, in this case an MCC code.

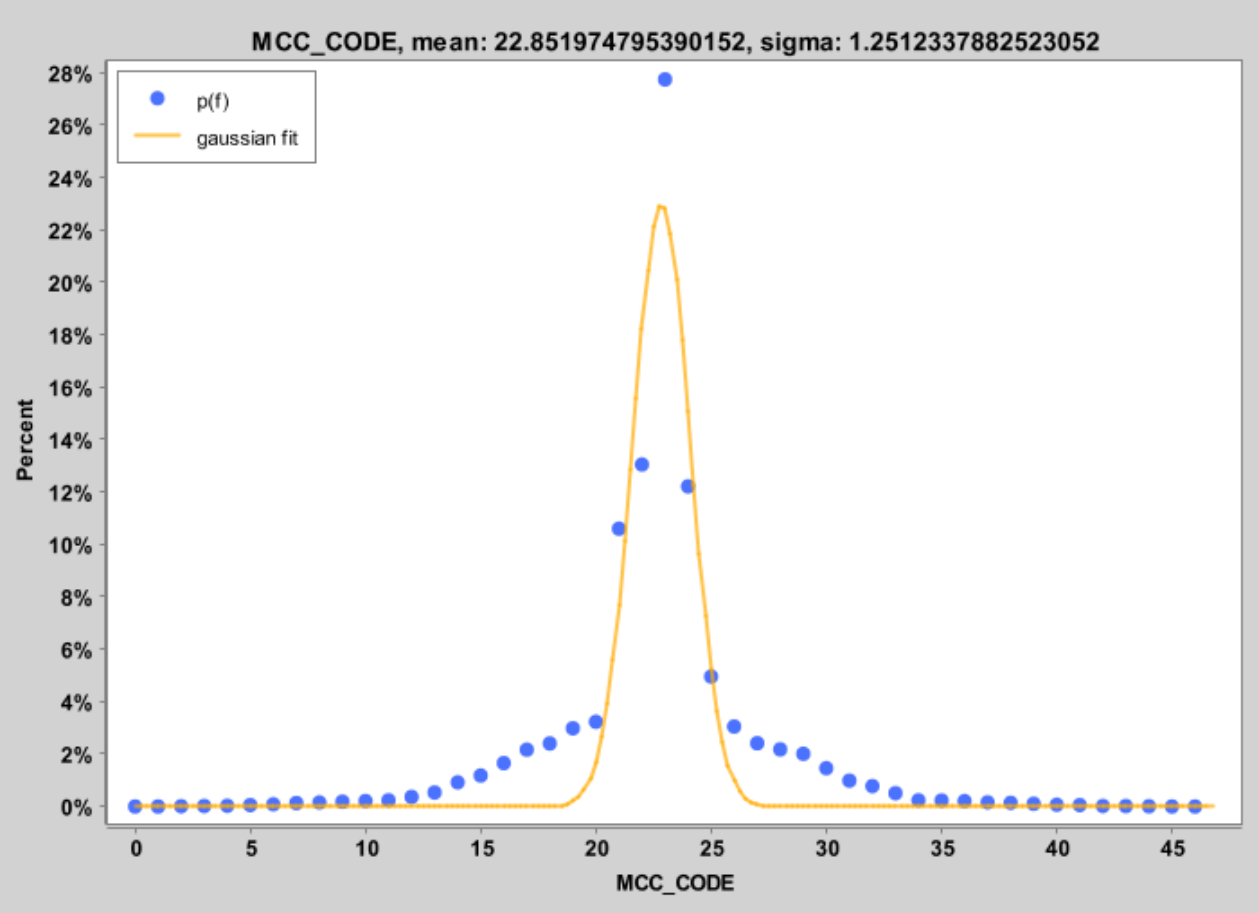


Figure 4.2. Gaussian fit of MCC Code column.

The results of the Gaussian fit on the MCC column, seen in the figure above, turned out well. The merchants that had a large amount of transactions seem to fit the curve well, as do the columns on the lower end. The points in between still fit with a low (~2%) margin of error.

4.2 User Country

In this transaction dataset, the User Country is the country in which the person who made the transaction resides. The results of the analysis performed on the User Country column can be seen in figures 5.3 and 5.4 below. For this column, we can see that 95% of the transactions in the test data were performed by customers from the top two countries, with the remaining 5% being spread out over the bottom 90 countries in the data. This sharp distribution was modeled

clearly with the calculated Gaussian distribution, as seen in Figure 5.4.

We can infer from this column distribution that the large majority of our consumers come from one same country, which should be the United States here, but we still have a broad range of consumers who comes from 92 countries which shown in the x-axis.

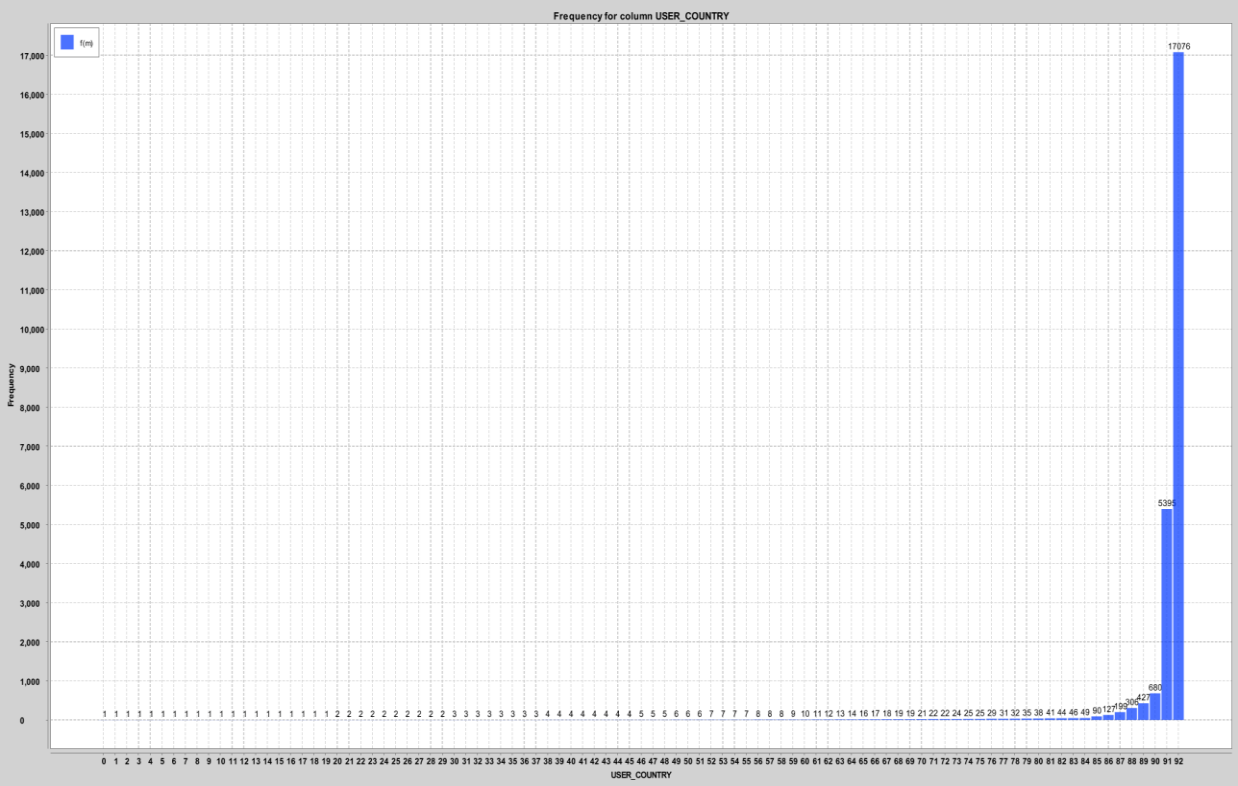


Figure 4.3. Frequency count of User Country column.

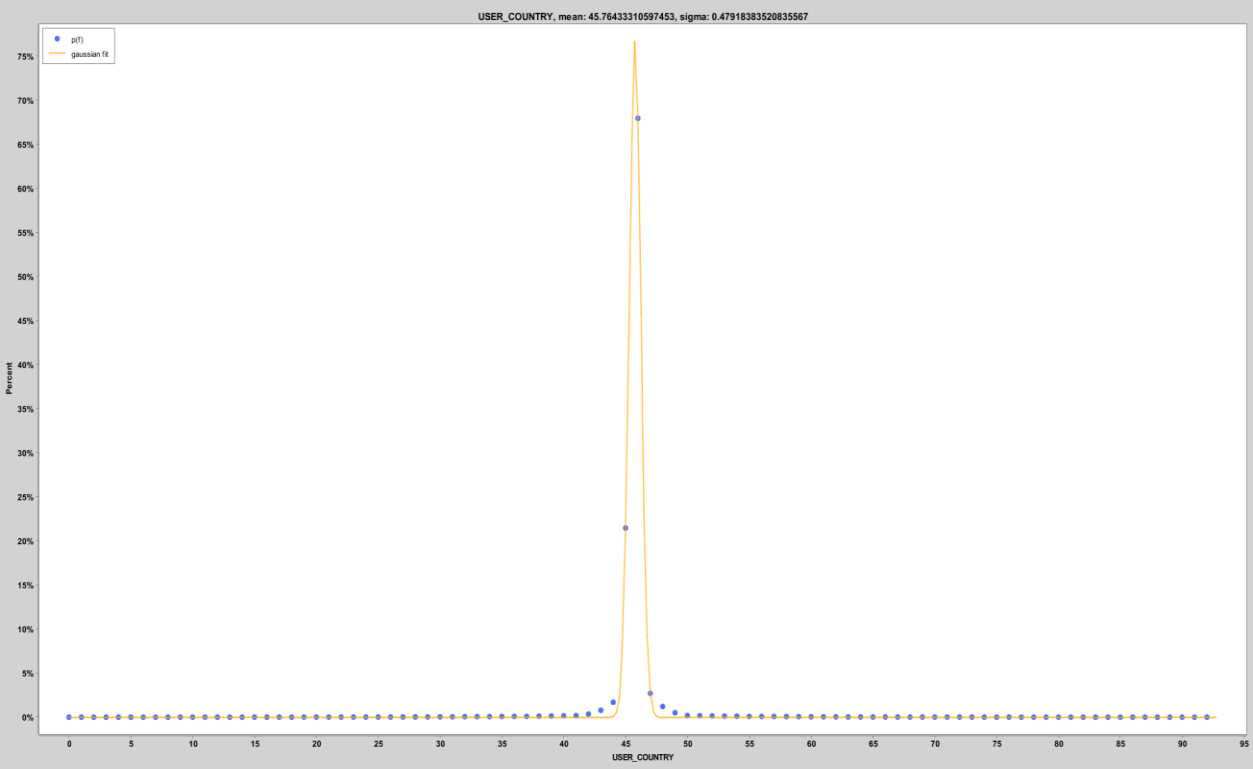


Figure 4.4. Gaussian fit of User Country column.

4.3 Merchant Country

The Merchant Country in this data is the country where the merchant took place at resides. The results of the Merchant Country column can be seen in figures 5.5 and 5.6. As can be seen in Figure 5.5, about 80% of the transactions took place in the highest two countries, with there being 15 countries overall that transactions took place at. The Gaussian fit of this data worked quite well, with all of the points closely matching the curve.

We can infer from this column distribution that:

- Most of the customers in the test dataset come from two countries because transactions that are more local to the customer are more likely to occur than transactions that occur in a foreign country.
- There are still, however, a great number of countries that do not have a large

number of merchant transactions, which may represent the fact that there are still customers that like to travel or the fact that the original dataset did not contain a large number of transactions from these other countries and the frequency count reflects that.

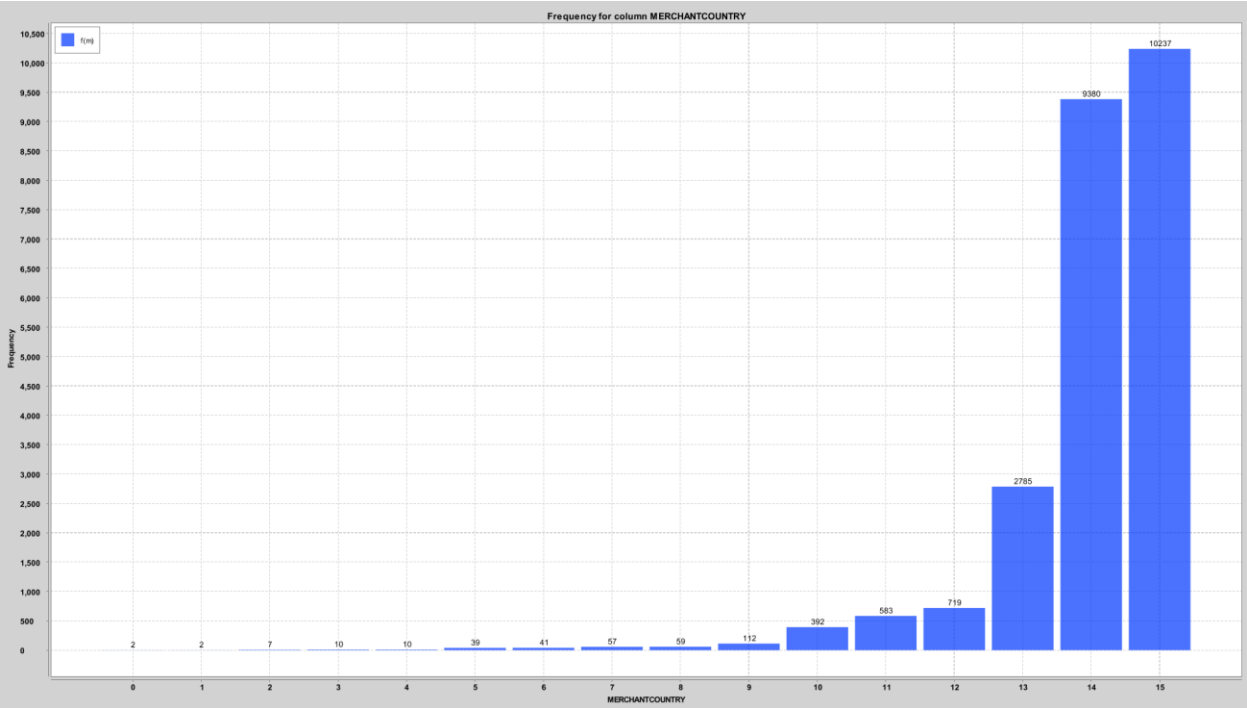


Figure 4.5. Frequency count of Merchant Country column.

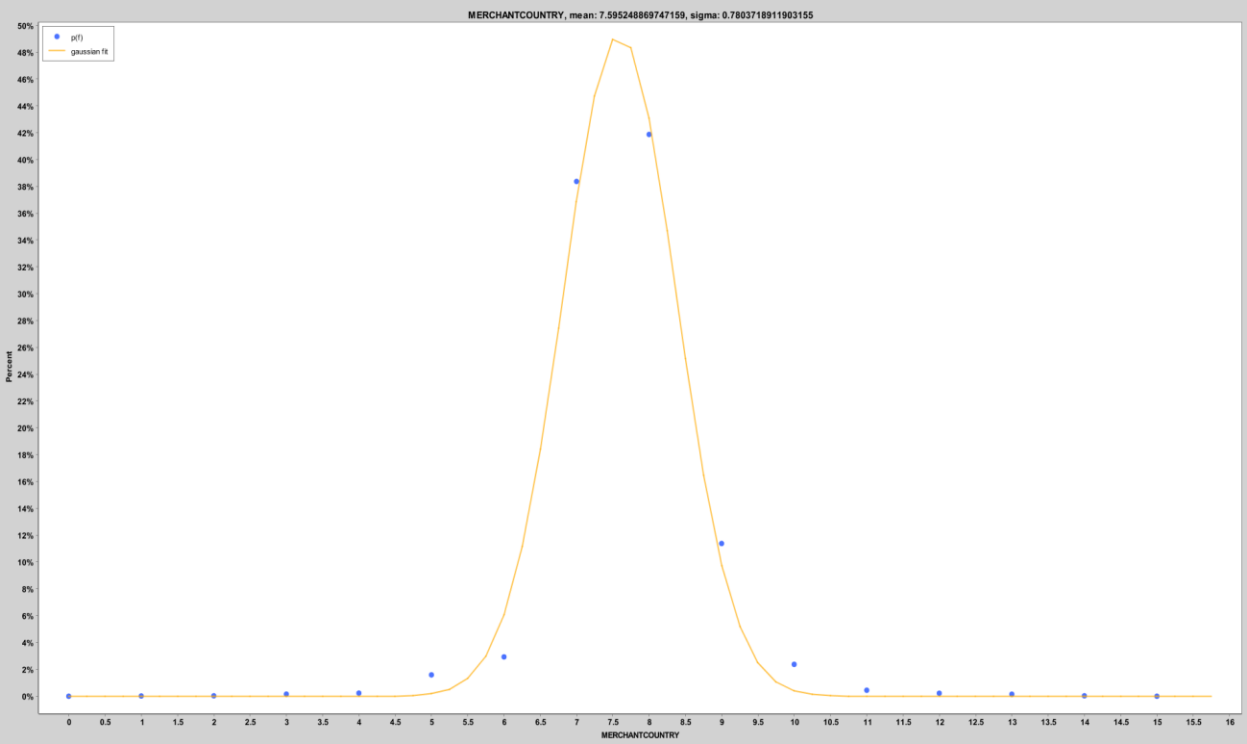


Figure 4.6. Gaussian fit of Merchant Country column.

4.4 Card Type

The results from the Card Type column can be seen in figures 5.7 and 5.8. The card type of a transaction is simply the company the card that was used in the transaction belongs to (Visa, American Express, etc.). The curve made by this data naturally fit the Gaussian curve very well, with almost 90% of the transactions falling within the top three card types.

We can infer from this column analysis that:

- Although consumers may have various types of credit or debit card such as VI (Visa), AX (American Express), BC (BC Card), CA (Master Card), DS (Discover Card), E (Visa Electron), JC (Japan Credit Bureau), and TO (Maestro), there are only three card types that dominate usage in daily transactions. Since these three card types are the most used, it is safe to assume that they are the most popular.

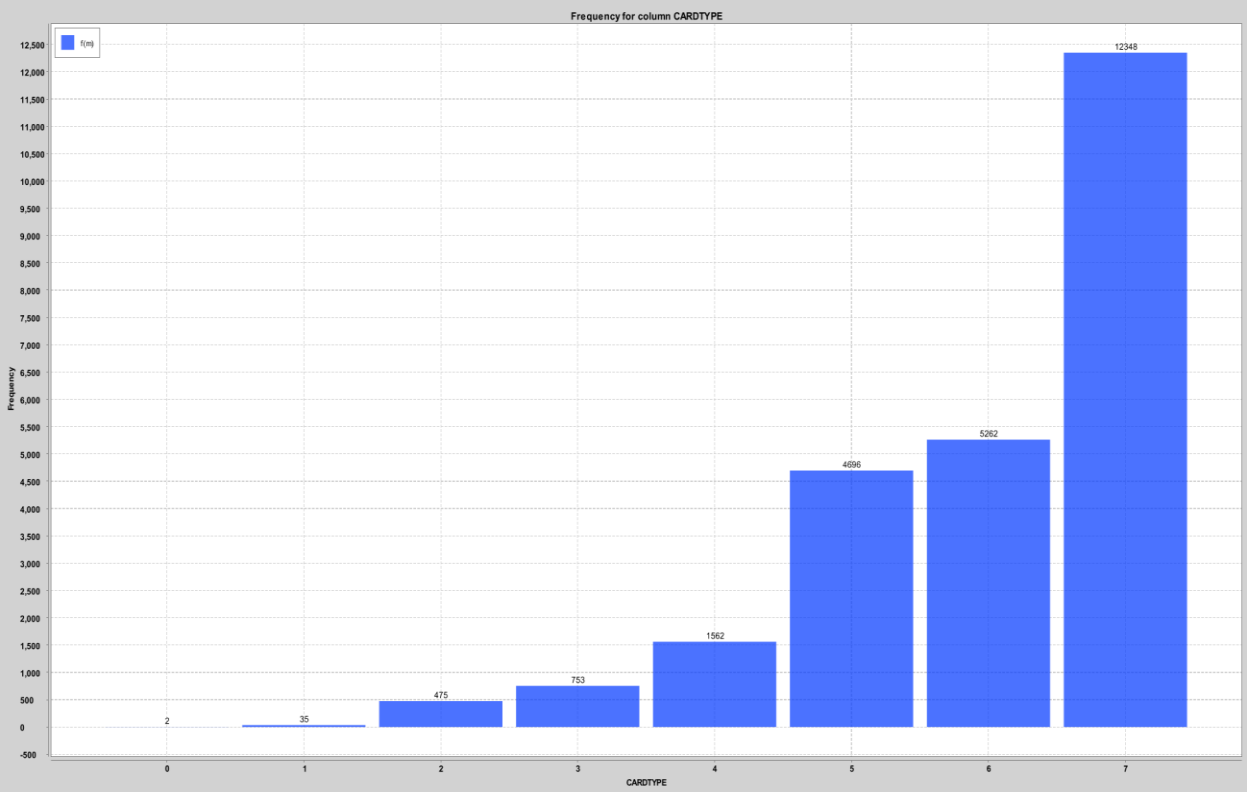


Figure 4.7. Frequency count of Card Type column.

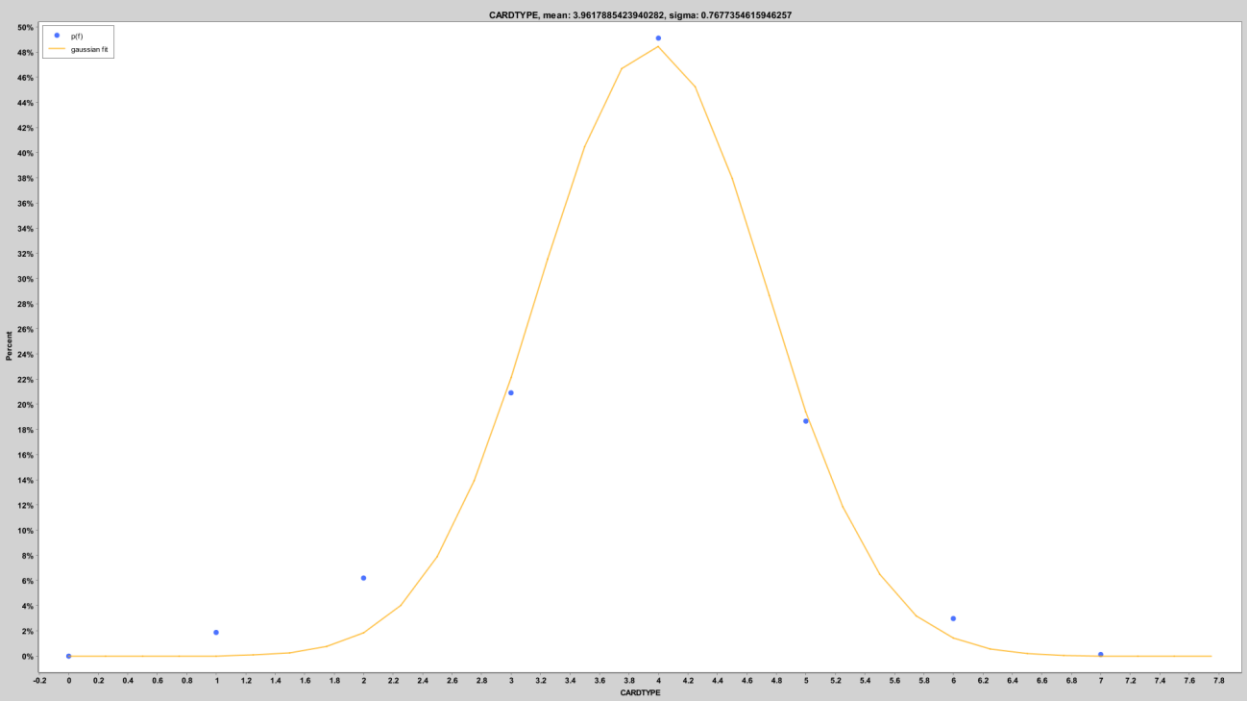


Figure 4.8. Gaussian fit of Card Type column.

4.5 Merchant State

The Merchant State column indicates the state the merchant that the transaction took place at resides. Since there are many countries in the data, the list of states is several hundred entries long. This data did not fit the Gaussian fit curve that well. One possible reason for this could simply be the amount of entries and the way they are spread. This data seems to be too spread out in the lower regions for the Gaussian to fit well. Another explanation could be that we did not account for multiple strings representing the same state - for example, the state Massachusetts could be represented by the strings “MA”, “Mass”, or “Massachusetts”. In order to take something like this into account in our program, there would have to be some sort of dictionary built for this column (and possibly other columns where this may make sense) that could be loaded in at runtime to alter how the mapping function works.

Further improvements could be made:

- In order to analyze the columns more accurately, the vocabularies which utilize the same semantics should be grouped into only one column.
- A country-based filter should be added to make the merchant state analysis more specific.

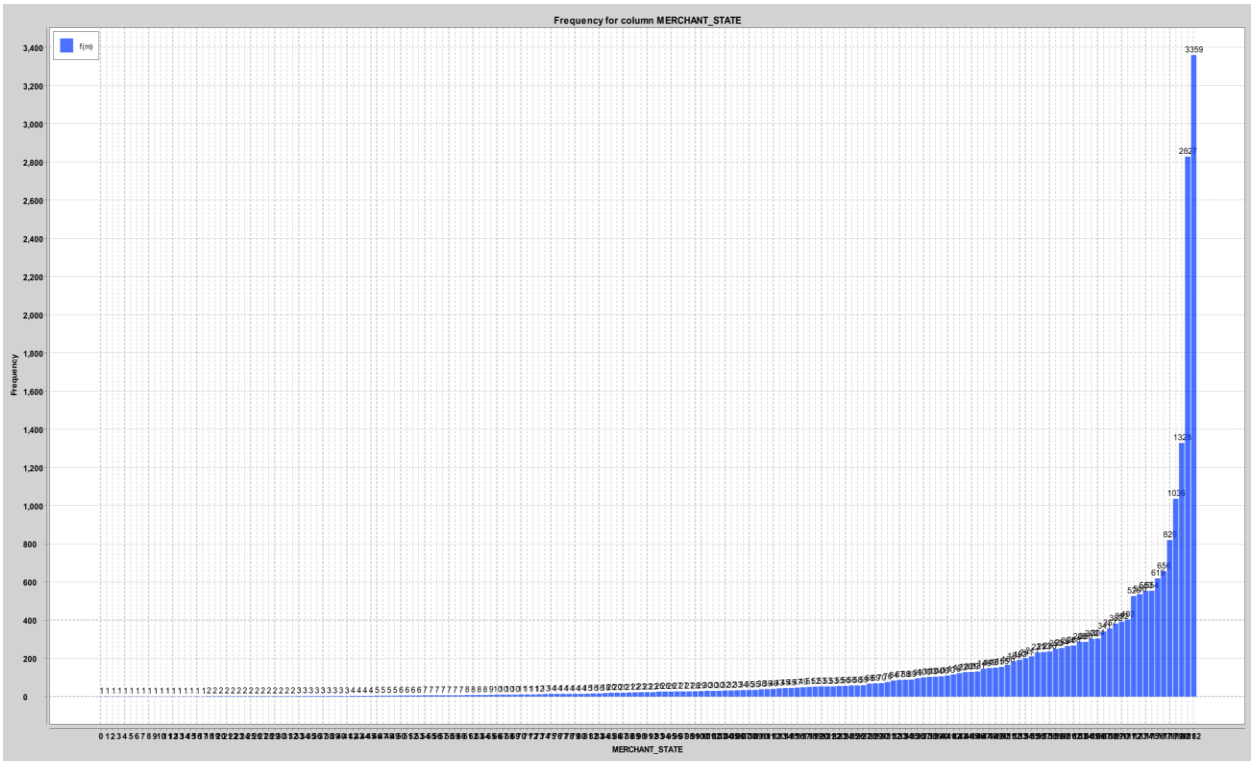


Figure 4.9. Frequency count of Merchant State column.

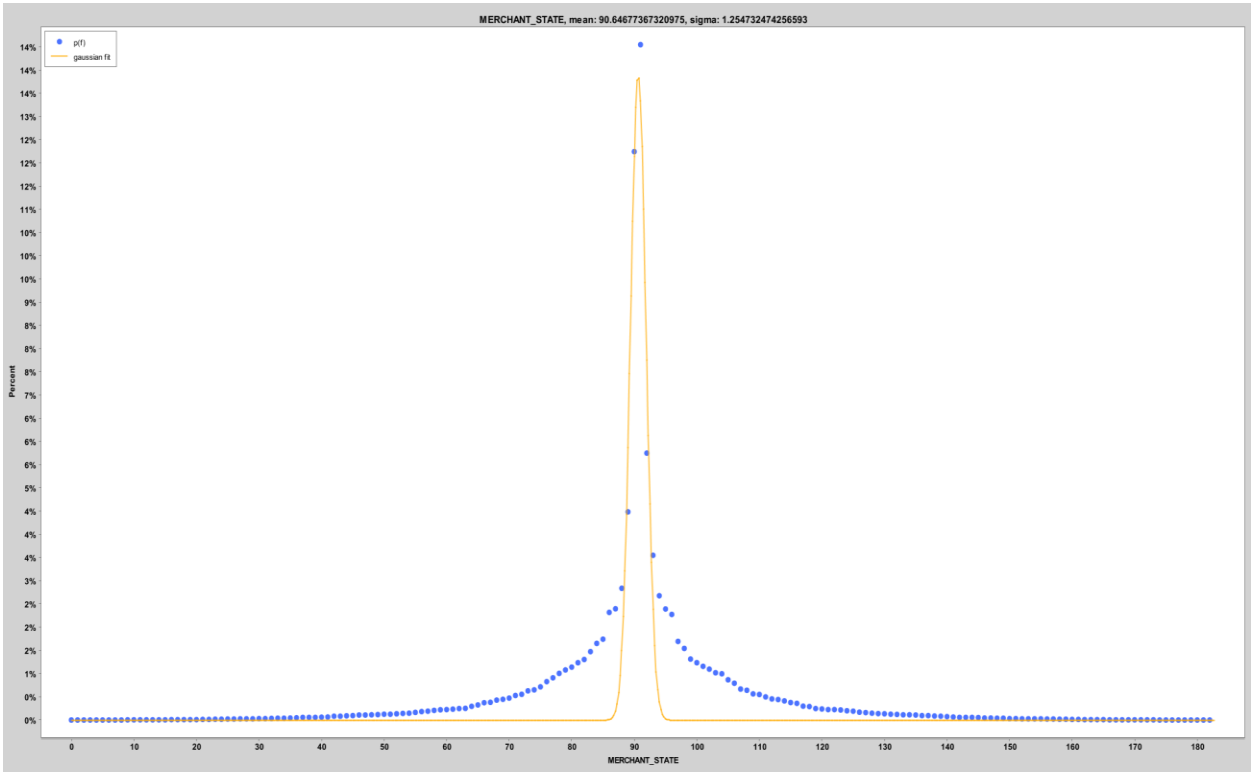


Figure 4.10. Gaussian fit of Merchant State column.

5 RESULTS

5.1 Data Synthesis

5.1.1 Synthesis Implementation

With the use of Cholesky decomposition and Spark's multivariate Gaussian distribution methods, the team successfully generated 1,000 lines of continuous data from the 1000 lines read from the original dataset with ten different columns and attempted to translate them back to their more readable original state. As the first try, the team only attempted to translate fifteen rows of continuous data with the index mapping table back to the original values. The translation took two minutes when run locally and successfully produced readable data as seen in Figure 5.1. Unfortunately, further results were no longer able to be obtained due to persistent technical issues at the end of our project.

TRX_AMOUNT	MCC_CODE	CARD_BIN	ACCTNUM	CARDTYPE	CARDEXPDT	MERCHANTID	USER_COUNTRY	MERCHANT_STATE	ZIPCODE
252.44	9632	054123	d1ea61afba35735c41d1	E	1125	6edac305750a3a9afa3b	417	fl	33815
145.22	3887	054123	d1ea61afba35735c41d1	E	1125	6edac305750a3a9afa3b	417	fl	33815
0.08	4448	054123	d1ea61afba35735c41d1	E	1125	6edac305750a3a9afa3b	417	fl	33815
-152	4297	054123	d1ea61afba35735c41d1	E	1125	6edac305750a3a9afa3b	417	fl	33815
107	4255	054123	d1ea61afba35735c41d1	E	1125	6edac305750a3a9afa3b	417	fl	33815
190	5481	054123	d1ea61afba35735c41d1	E	1125	6edac305750a3a9afa3b	417	fl	33815
319	6027	054123	d1ea61afba35735c41d1	E	1125	6edac305750a3a9afa3b	417	fl	33815
38	1488	054123	d1ea61afba35735c41d1	E	1125	6edac305750a3a9afa3b	417	fl	33815
144	7331	054123	d1ea61afba35735c41d1	E	1125	6edac305750a3a9afa3b	417	fl	33815
241	1700	054123	d1ea61afba35735c41d1	E	1125	6edac305750a3a9afa3b	417	fl	33815
397	947	054123	d1ea61afba35735c41d1	E	1125	6edac305750a3a9afa3b	417	fl	33815
97	4483	054123	d1ea61afba35735c41d1	E	1125	6edac305750a3a9afa3b	417	fl	33815
378	2628	054123	d1ea61afba35735c41d1	E	1125	6edac305750a3a9afa3b	417	fl	33815
-521	2333	054123	d1ea61afba35735c41d1	E	1125	6edac305750a3a9afa3b	417	fl	33815
209	5334	054123	d1ea61afba35735c41d1	E	1125	6edac305750a3a9afa3b	417	fl	33815

Figure 5.1. Synthesized data.

5.1.2 Synthesized Data Analysis

Clearly, due to small size of our results it is difficult to reach any sort of significant conclusion. Technical issues involving the status of the server machines impeded any further progress; nonetheless, it is still possible to analyze our current results to determine whether or not the synthesized dataset made sense. The following inferences were obtained after some analysis on our generated dataset (the data does not contain any sensitive information and is randomly generated from patterns gleaned from a test dataset that was scrubbed of sensitive data as well).

- There is a customer from Liechtenstein (User Country Code 417) that visited Lakeland (Zip code 33815), Florida (Merchant State FL) and made some transactions there.
- The customer has a credit card with card BIN (Bank Identification Number) 054123, card type E (Visa Electron), and with a card expiration date of 11/25.
- The amount in this customer's transactions varies roughly from $0 to $400, which leads to the conclusion that if there is a new transaction of large amount such as $5,000 that appears in this customer's transaction record, it may be possible to detect this new transaction as a fraudulent transaction.
- There are two negative transaction amounts that appeared in the transaction list which are an unfortunate byproduct of random generation. Further improvement could obviously be made to the data and model generation.

Obviously, the synthesized data could be interpreted to represent that one specific

customer; however, further tests and analysis on the full 1000 lines of generated data would produce a more concrete result and analysis set.

5.2 Validation Testing

After synthesizing and translating data back to its original state, the team needed to verify whether the generated data had a similar behavior to the original transaction data. Unfortunately, once again due to technical issues concrete validation cannot be performed. Nonetheless, a "theoretical validation" was performed that underwent the steps that an actual validation would have taken.

5.2.1 Validation Process

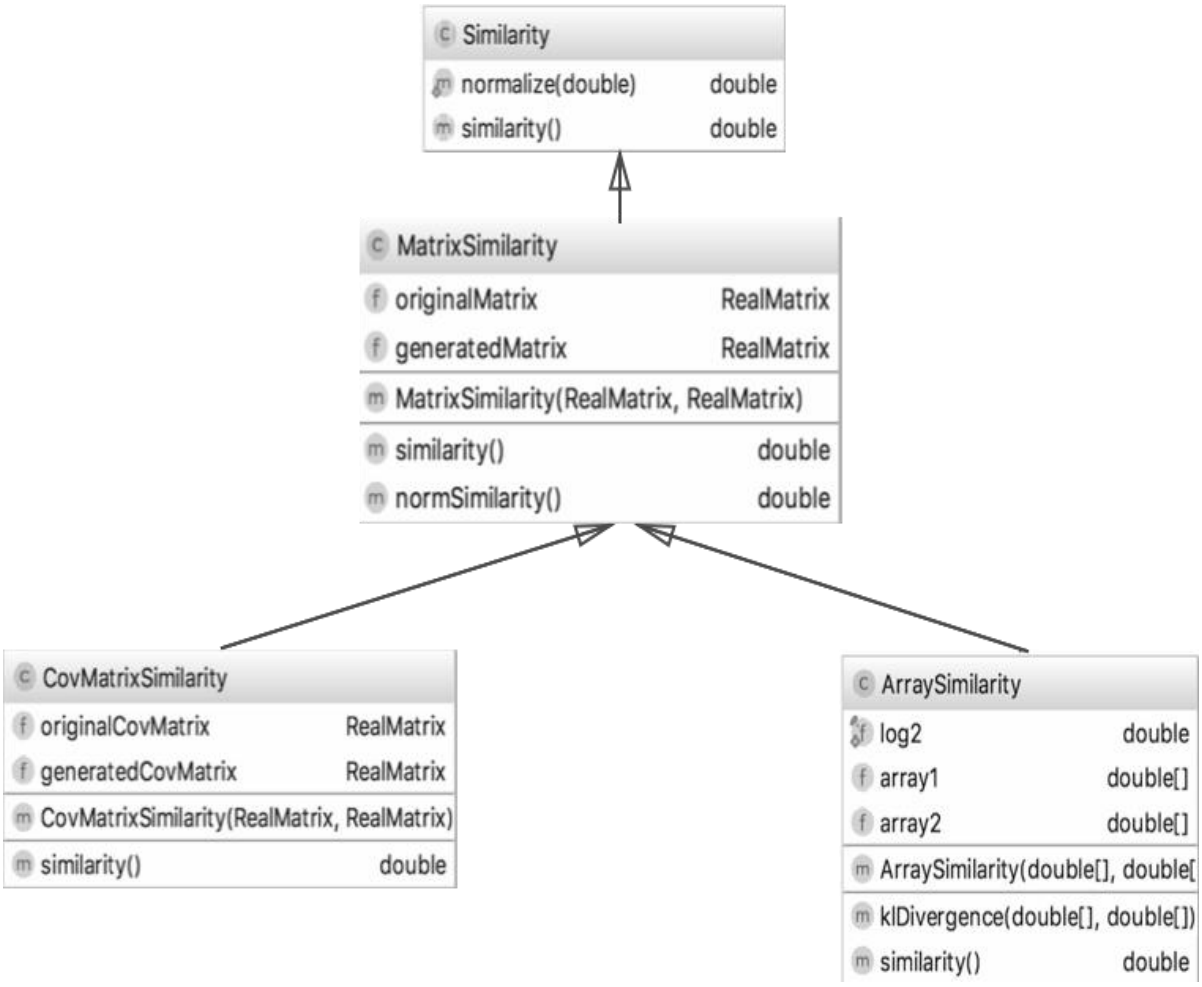


Figure 5.2. Validation diagram.

All of the classes in validation inherit the same interface, Similarity. The public method, similarity, returns an integer which will be in range of 0 to 1 that indicates the similarity between two comparable objects. The MatrixSimilarity class uses the CovMatrixSimilarity class and ArraySimilarity to calculate similarity between two matrixes. A similarity value between two matrices is composed of the similarity of two covariance matrices that come from themselves and a total similarity which results from comparing each column in both matrixes. The similarity of two covariance matrices is calculated by the sign of the value in the covariance matrix. If two

values in the same position in two matrices have the same sign, the function will treat them with the same relevance and vice versa. The similarity of two arrays comes from KL divergence, which is a famous algorithm used to test the divergence of two different arrays. MatrixSimilarity will combine these two similarities to create a final similarity.

```
public void simlarityTest(){
    RealMatrix m1 = new Array2DRowRealMatrix(new double[][]{
            {1,2,3},
            {4,5,6},
            {7,8,9}
    });

    RealMatrix m2 = new Array2DRowRealMatrix(new double[][]{
            {1,2,3},
            {4,5,6},
            {7,8,9}
    });

    double similarity = new MatrixSimilarity(m1, m2).similarity();
    System.out.print(similarity);
}
```

Result: 100%

```
public void simlarityTest(){
    RealMatrix m1 = new Array2DRowRealMatrix(new double[][]{
            {10,2,3},
            {40,5,6},
            {7,8,9}
    });

    RealMatrix m2 = new Array2DRowRealMatrix(new double[][]{
            {1,2,3},
            {4,5,6},
            {7,8,9}
    });

    double similarity = new MatrixSimilarity(m1, m2).similarity();
    System.out.print(similarity);
}
```

Result: 78 %

Figure 5.3. Validation testing sample.

The figure above shows how the validation process works. The MatrixSimilarity class receives two matrices that contain the original dataset and the synthesized data in continuous format to compare. An instance of the similarity method can return a value in the range of 0 to 1 that indicates how similar the two matrices are. If the return value is close to 1, it implies that these two matrices are quite similar and vice versa. The left side part of the figure above indicates that if the MatrixSimilarity class instance receives two matrices that are exactly the same, the similarity function will return the integer “1” which shows that the generated dataset and the original dataset are 100% same, which is too ideal to happen. After changing any one of the matrices, which is shown in the right side of the figure above, the similarity function will

return an integer that is between 0 and 1 depending on how close the generated matrix is to the original one.

When actually performing the validation, these two matrices will be replaced by the original and synthesized data in continuous format. The reason for abstracting transaction data to mathematical matrices is that there are many existing, well tested algorithms, frameworks or libraries that could be used in a similarity test. In the MatrixSimilarity class, covariance matrix related functions from Apache Common Math to calculate covariance matrices were used. The covariance matrix is a mathematical way to measure the joint variability of two matrices. A value of one variable in one matrix will correspond with the value of the variable in the same position in the other matrix and the correlation value in a covariance matrix is positive and vice versa. These compose the core algorithm behind the similarity function.

6 CONCLUSIONS

6.1 **Conclusions**

The goal of this project was to detect distribution patterns in data, build statistical models, and generate transaction data that simulated the actual data's behavior without revealing private information. The index mapping table, distribution model and validation method created during this project provided an effective pipeline to meet our project's main criteria.

The index mapping table was capable of performing data preprocessing and clean up. The two major features of the table were: converting the original discrete transaction data to continuous indices to prepare for future data analysis, and to translate generated continuous data back to their original discrete state. The index mapping table provided an efficient way to approach the actual data and also helped in creating a much more readable version of the synthesized data.

Subsequently, the distribution model is capable of generating data based on the original transaction data. After analyzing the most typical five columns from transaction dataset separately, we concluded that every column presented behaviors that tended toward a Gaussian distribution although not every column had a perfect fit. As a result, multivariate Gaussian distribution is an ideal distribution type while dealing with all the columns together. Accordingly, our distribution model successfully generated through the use of Spark's multivariate Gaussian distribution functions with the help of Cholesky decomposition generated several rows of correlated transaction data that was also successfully translated back to readable data through the index mapping table.

The validation method is capable of evaluating how close the synthesized data is to the original data. The major tool for comparison are the two covariance matrices. The validation

method is capable of using the two covariance matrices (the original continuous transaction data and the generated continuous transaction data) as input and will straightforwardly output a percentage representing how similar these two datasets are to each other. Unfortunately, this portion of the pipeline was never officially tested with the generated data due to technical issues nearing the end of the project duration.

In conclusion, the team has designed and built a working pipeline that could handle data ingestion, model generation, data synthesis, and a theoretical method of validating the synthesized data, which satisfies all of the project requirements.

6.2 Reflections

Throughout the course of this project, the team encountered many trials and tribulations that were a roadblock on the path to project completion. Although not unscathed, the team still managed to break through to create a working pipeline. We hope to impart the experiences we learned to any future groups that would attempt to expand upon this project. Firstly, do not underestimate the scope of the project. Even though at first glance, the objective is simple, data generation using patterns drawn from a test dataset, the scope of the project encompasses much more than that. The project approach, the different technologies available for use, and the equipment needed to use for the project are just some of the considerations that need to be made before the project can even begin.

Next, communication is key. If the project does not have an extremely communicative leader, then progress becomes muddy and slow. Proper communication between the different team members would greatly increase the speed and quality of the work being completed as each team member could aid one another if one part of the pipeline was causing an issue. Also, time

management is an important skill to have when undergoing any sort of lengthy project and if managed by a conscientious leader should bring any group far.

As for the development process, the main issues the team encountered were mainly related to technical issues. An important lesson for future groups is to secure a sufficiently large group of servers with a large storage space that can be utilized for extremely long periods of time to avoid the same issues our group underwent. In addition to having better technical planning, a more in-depth study of the material the project is about would have also greatly helped with any design decisions that needed to be made.

Finally, a huge roadblock the team encountered was the integration of various different technologies into our system. Although there are many impressive technologies available, choosing the right ones, that not only integrated with our system but also performed the necessary actions, was one of the more difficult and time-consuming processes during the project experience.

7 FUTURE WORKS

7.1 Improvement on Current Project

One obvious expansion is to discover and fix the issue regarding the data generation and translation. As mentioned in the Data Synthesis section in Results, there are a few negative data values in the Transaction Amount column, which are supposed to contain all positive values. Although more of an issue with using randomness to generate data, in a future work investigation into a more accurate method of data generation and synthesis is recommended. Alternatively, the team recommends investigating the usage of machine learning instead of statistical methods in an attempt at solving this problem.

Connected to the data generation, a recommendation to any future teams is to implement a different method of data generation compared to the one currently utilized within this pipeline. The reasoning behind using a custom data generation as opposed to an existing method like the "sampling" method in Apache Commons Math is due to a "Matrix is Singular" error. Future teams may want to look into solving this error which would allow the usage of the numerous sampling methods that are currently available.

As explained in the Data Synthesis section, the team only managed to generate and back translate fifteen rows of data into the readable format that is shown in that section. In other words, another recommendation for any future projects built upon this one is to test the data synthesis and back translation on the full 100,000 lines to ensure that the generation and validation are performing properly.

7.2 Other Use Cases

A second interesting use case can be looked at is involving the usage of generated distributions to implement a method of fraud detection that examined how well a new incoming transaction fitted a generated model. If, for example, the new transaction was at a new retailer or country that the customer has never been to before then that transaction would give a much worse fit to the distribution than a transaction performed at a retailer that the customer has been to many times before. A possible implementation of this could look at how many standard deviations away from the mean each column was located.

Another possible expansion would be to implement the ability for the program to analyze a dataset and then compute the distribution that best fits that particular dataset. Using existing machine learning libraries, the best distribution to fit a column or a set of columns as well as the correlation between the different columns could be found and a dynamic model could be created. Using these methods could also expand upon our team's model and data generation engine to improve model and synthesized data accuracy. Finally, the current validation suite's execution speed is rather slow on larger datasets and converting the suite to use distributed calculations would vastly improve the speed.

BIBLIOGRAPHY

Apache Software Foundation. (2017). *Overview*. Retrieved 3/23/2017 from https://phoenix.apache.org/

Apache Software Foundation. (2016). *Commons Math: The Apache Commons Mathematics Library. General Format.* Retrieved from http://commons.apache.org/proper/commons-math/index.html

Baia, J, et al. (2015). *Scalable multi dimensional threat analysis for financial risk analysis*. Major Qualifying Project, Worcester Polytechnic Institute.

Banovic, N, et al. (2016). *Modeling and Understanding Human Routine Behavior*. Human-Computer Interaction Institute.

Burnham, K.P., Anderson, D.R. (2002). *Model Selection and Multi-Model Inference* (Springer). (2nd edition), p.51

Cython. (n. d.). Retrieved 1/15/2017 from http://cython.org/

Cao, J., Carminati, B., Ferrari, E., & Tan, K. L. (2008). CASTLE: A delay-constrained scheme for ks-anonymizing data streams. *2008 IEEE 24th International Conference on Data Engineering*. doi:10.1109/icde.2008.4497561

Federal Trade Commission. (2006, April). Financial Institutions and Customer Information: Complying with the Safeguards Rule. Retreived from https://www.ftc.gov/tips-advice/business-center/guidance/financial-institutions-customer-information-complying

George, Lars. (2011). *HBase The Definitive Guide.* Sebastopol, CA: O'Reilly Media, Inc.

Gkoulalas-Divanis, A., & Loukides, G. (2011). *PCTA: privacy-constrained clustering-based transaction data anonymization*. Proceedings of the 4th International Workshop on Privacy and Anonymity in the Information Society (PAIS '11). http://dx.doi.org.ezproxy.wpi.edu/10.1145/1971690.1971695.

Jain, A.K. (2010). Data clustering: 50 years beyond K-means. Pattern Recognition Letters. Volume 31, Issue 8. Pages 651-666. http://dx.doi.org/10.1016/j.patrec.2009.09.011

Kullback, S., Leibler, R.A. (1951). "On information and sufficiency". *Annals of Mathematical Statistics*. **22** (1): 79–86. doi:10.1214/aoms/1177729694. MR 39968.

Kullback S. (1959). *Information Theory and Statistics.* John Wiley & Sons.

Hadoop. (2016, October 11). *What is Apache Hadoop?* Retrieved from

http://hadoop.apache.org/

MacKay, David J.C. (2003). *Information Theory, Inference, and Learning Algorithms* (First ed.). Cambridge University Press. p. 34.

MathWorks. (n. d.) *Unsupervised Learning*. Retrieved from https://www.mathworks.com/discovery/unsupervised-learning.html

Pedregosa, F., et al. (2011). *Scikit-learn: machine learning in python*. Journal of Machine Learning Research. 12: 2825–2830.

Penchikala, S. (2015). *Big Data Processing with Apache Spark*. Retrieved from https://www.infoq.com/articles/apache-spark-introduction

Pentland, A, Liu, A. (1999). *Modeling and Prediction of Human Behavior*. Neural Computation. 11, 229–242.

Phatak, M. (2015). *History of Apache Spark : Journey from Academia to Industry*. Retrieved from http://blog.madhukaraphatak.com/history-of-spark/

Putze, F. (2012). *Human Behavior Modeling*. Retrieved 10/16/2016 from https://csl.anthropomatik.kit.edu/downloads/vorlesungsinhalte/KM2012-V12-HumanBehaviorModeling.pdf

Pyle, D., 1999. *Data Preparation for Data Mining*. Morgan Kaufmann Publishers, Los Altos, CA.

Rish, I. (2001). An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence* (Vol. 3, No. 22, pp. 41-46). IBM New York. Chicago

Robinson, David. (2014, December 20). *Understanding the beta distribution (using baseball statistics).* Varianceexplained. Retrieved from http://varianceexplained.org/statistics/beta_distribution_and_baseball/

Sauro, Jeff. (n. d.). [A frequency graph of the heights of men and women]. Usablestats. Retrieved from http://www.usablestats.com/lessons/normal

Seydel, R. (2012). *Tools for Computational Finance.* [PDF]. Retrieved from http://www.springer.com/cda/content/document/cda_downloaddocument/9781447129929-c2.pdf?SGWID=0-0-45-1314437-p174313273

StatisticsHowTo. (n. d.). Retrieved 10/16/2016 from http://www.statisticshowto.com/

Wasilewska, A. APRIORI Algorithm [PDF document]. Retrieved from http://www3.cs.stonybrook.edu/~cse634/lecture_notes/07apriori.pdf

APPENDIX A

APPENDIX A: User Guide

System Requirements:

- JDK 1.7 and above
- Apache Maven 3.0 and above
- HBase (with Phoenix)
- Spark

Project Logic Structure:

Project code is divided into three parts: Index Mapping, Ingestion Engine, and Validation. Each part has its own pom.xml for dependency management.
Goal of Index Mapping: to index the continuous data to discrete data and have the ability to translate them back.
Goal of Ingestion Engine: to generate synthesized data using spark engine
Goal Validation: to validate how well the synthesized and original data match

Install & Run:

This project used Maven to construct dependencies. Make sure pom.xml is in all project folders and using [mvn install] under each project folder maven will download dependencies and build the project automatically.

Ingestion Engine

The Ingestion_Engine_2016 project contains both the data ingestion and distribution generation parts. Running both of these parts happens in the IngestData class in the engine package.

Uploading Data

To run the ingestion engine to upload the data to the Hadoop server, you may use the ParseData class. This only needs to be done once and uses a file that contains the transaction data in a continuous, non-delimited format with the line indexes and lengths specified in the JSON file input into the IngestData class.

Building Distributions

Distribution building depends on a Hadoop server with an HBase database running on it. The name of the table our group used was “TRX4” but this may be changed in the FittedMultivariateGaussian class in the distribution package.

There are two ways to build a distribution:

1. Distribution on the entire data: no arguments are given to the constructor
2. Distribution on a subset of data: the constructor takes in the column being filtered and the string to filter by

There are two methods of generating a distribution currently. The first method is using the generate() function, which uses Phoenix to read the data and local Apache Math calculation methods to generate the distribution. The sparkGen() function was written to use spark’s built in HBase reading and RDD calculations, which are horizontally scalable when run on a Hadoop cluster. However, the sparkGen() function will only generate a distribution on the entire data, not filtered.

Distributions are saved both on the HBase server and locally in a file in the Intestion_Engine_2016/distributions folder.

Index Mapping:

Index mapping is a tool, like dictionary, that used inside the ingestion engine to index continues data so that they can be treated as discrete data. Discrete data is like name, country name and etc.. IndexData will create a table ‘INDEXTABLE’ in HBase as dictionary. It has ability translate back and forward between continuous data and discrete data. To translate generated data from continuous type back to original type, you need to translate whole matrix which represents the generated data. If the column is in continuous data type, the index mapping table will jump over and keep this column in the same form.

Convert discrete data to continuous data

IndexData class is built for translating continuous data to discrete data. Before the translation, we need to create a new IndexData instance with a parameter, a connection object connected to HBase with Apache Phoenix:

IndexData index = new IndexData(connection)

Then we can use instance method mapIndexByArray to translate continuous data. The function takes two parameter, first is the name of the column that holds continuous data, second is thedata itself as a double array:

Index.mapIndexByArray(columnName, continuousDataArray)

Convert continuous data to discrete data

The TranslateData class is built for translating continuous data to discrete data. Before using it, we need to create a new TranslateData instance.

TranslateData dict = new TranslateData(connection)

Next we must set the name of the lookup table for the TranslateData instance, otherwise it doesn't know which table it should use to translate data:

dict.setTableName("INDEXTABLE")

Now we can use the instance function, translate, to translate the data. The function takes three parameters: first is the continuous data matrix, which is our generated data, second is the offset of the continuous data columns in a matrix(we suppose all the columns that holds continuous data are in front of all the discrete data columns), the third parameter is a string array that holds all of the names of the discrete columns:

dict.translate(dataMatrix, continuousColOffset, discreteColNameArray)

Validation

Validation is a way to verify the similarity of the synthesized transaction data and original transaction data. We abstract these two dataset into two matrices with the same dimension and

compare the covariance matrix of the datasets to check whether they have similar behavior or not.

To validate synthesized data

If we want to know how similar the generated data and original data are, we can use the class MatrixSimilarity. First let us create two example matrices(In actual validation, they will be replaced by two matrices of real data):

```
RealMatrix m1 = new Array2DRowRealMatrix(new double[][]{
{1,2,3},
{4,5,6},
{7,8,9}
});
RealMatrix m2 = new Array2DRowRealMatrix(new double[][]{
{1,2,3},
{4,5,6},
{7,8,9}
});
```

Then create a MatrixSimilarity instance with these two matrices. Now we can get a similarity value by using an instance of the similarity method:

```
double similarity = new MatrixSimilarity(m1,m2).similarity();
```

Detailed Implementation Example:

1. Before embedding IngestionEngine to project, make sure you have HBase database address and set up Phoenix properly on HBase
2. Insert into your main function these lines of code:

```
Connecton connection = DriverManager.getConnection(“jdbc:phoenic:xxxx”); [1]
FittedMultivariateGaussian fit = new FittedMultivariateGaussian();
String[] discreteCols = {“name1”,”name2”}; [2]
String[] continuousCols = {“name3”,”name4”}; [2]
fit.sparkGen(connection, discreteCols, continuesCols); [3]
```

Comment:

1. xxxx is your hbase database address

2. name1, name2, name3, name4 are column names in -the transaction dataset, such as “MCC_CODE”, and “CARD_BIN”. If the column is discrete, it belongs to the discreteCols variable; If the column is continuous, it belongs to the continuousCols variable.

3. Call sparkGen() with the Phoenix connection to Hbase, an array of the names of the discrete columns, and an array of the names of the continuous columns.

APPENDIX B: Detailed Project Pipeline

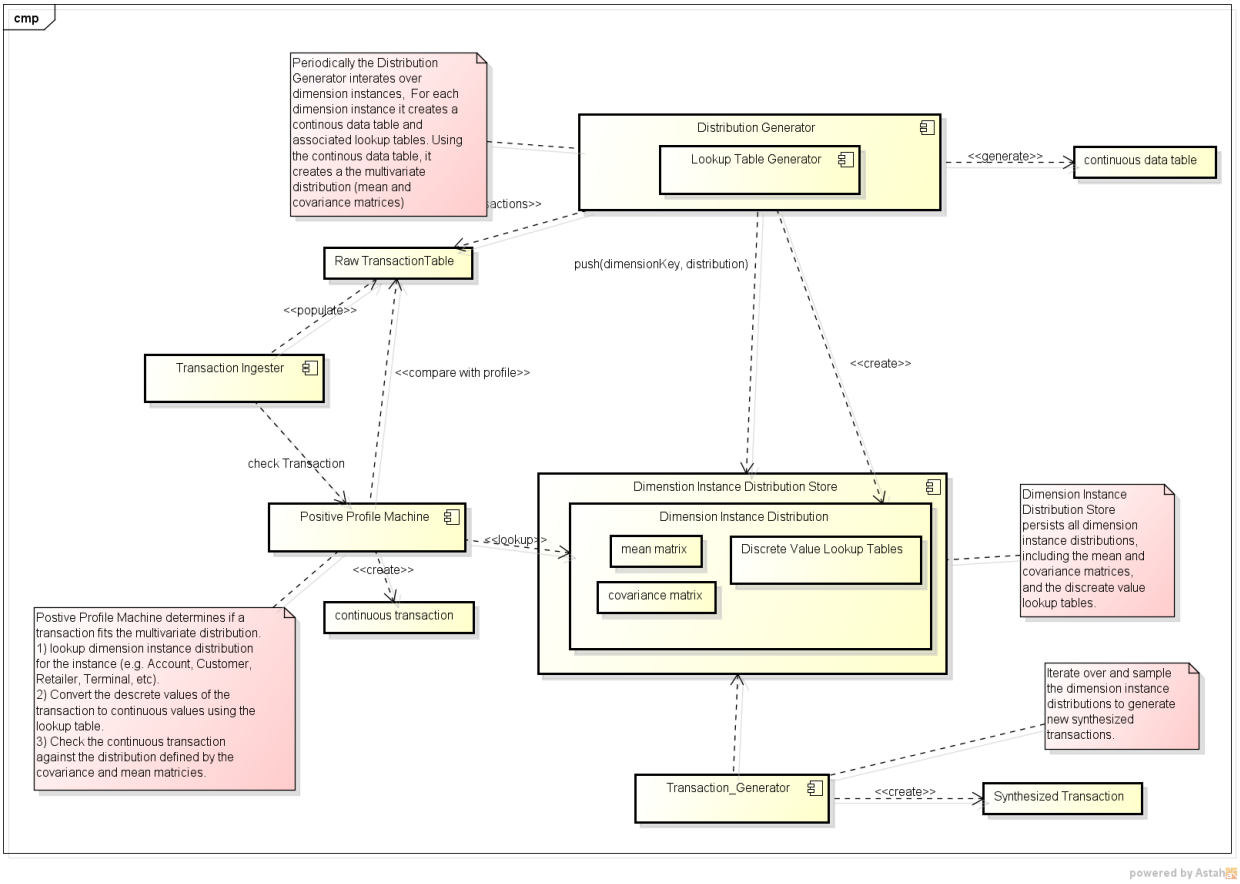


Figure 3.3. Complete Project outline.