Methods of Anomaly Detection: An Applied Framework and Application to Health Economics

---

A Thesis

Presented To

Eastern Washington University

Cheney, Washington

In Partial Fulfillment of the Requirements

for the Degree

Master of Science

By

Nathaniel Eric Islip

Spring 2022

THESIS OF NATHANIEL ERIC ISLIP APPROVED BY

_____ DATE:_____

DR. DALE GARRAWAY GRADUATE STUDY COMMITTEE

_____ DATE:_____

DR. VIKTORIA TAROUDAKI , GRADUATE STUDY COMMITTEE

_____ DATE:_____

DR. GERMÁN IZÓN, GRADUATE STUDY COMMITTEE

ACKNOWLEDGMENTS

First, I would like to thank my mentor Dr. Germán Izón of the Economics department, for always taking the time to listen and provide crucial feedback on my work and research. Both as an undergraduate and graduate student. Additionally, I want to thank my committee members, Dr. Viktoria Taroudaki and Dr. Dale Garraway, for reviewing my work and being excellent teachers. Lastly, I thank my fiancé, Julianna Olde, for sticking by my side and supporting my career and academic aspirations through thick and thin.

Table of Contents:

List of Figures

List of Tables

1 Anomaly Detection

1.1 Introduction to Anomaly Detection

An anomaly or outlier is a value demonstrating behavior substantially different from what is considered normal. Typically, anomalies arise in the context of disparaging results. For instance, an incorrect measurement, invalid data, or numerical codes; However, this is not always the case. Anomalies can provide important information and lead to discoveries like the element, Argon, discovered by Lord Rayleigh in 1894. Figure 1 demonstrates anomalies in the context of a 2-dimensional dataset. Clusters 1 and 2 demonstrate the normal population, and Objects 1 and 2 are anomalies. Additionally, Cluster 3 shows a potential grouping of abnormal instances.

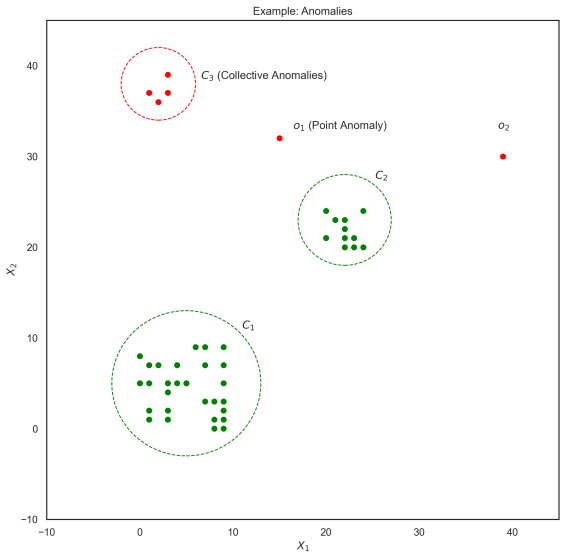


Figure 1: Example of anomalies

The normal or nominal population is first defined to identify an anomaly in a data set. Quantitative criteria test whether a data point or instance lies within or outside the average population. Standard tests include the three-sigma rule, which states a point must lie within three standard deviations of the mean. Alternatively, one could use outlier resistant or statistically robust procedures such as M-, R-, and L- Estimators. The M-Estimator replaces the fitting criterion for ordinary least squares with a term that penalizes significant prediction errors. R-estimators rank the degree of association among features. An example of an R-estimator is the Wilcoxon rank-sum statistic which compares the difference between two means. Other methods are distribution-based, with a standard distribution such as Poisson or Normal used to test fitting the data. Provided these methods, they pose limitations.

- Many methods are univariate (e.g., function on a single feature)
- Multivariate methods are unaware of the underlying distribution
- Distribution-based methods are cost-ineffective and generally relay poor results

Anomaly (Outlier) detection (AD) is a research area of machine learning (ML) that works to identify anomalies in a multi-dimensional dataset and address the above limitations. Novelty detection is closely related to AD; however, there is a crucial distinction between the methods. Novelty detection uses a model built from normality to test unseen data instances and determine if they are anomalies or not. AD has gained traction in numerous fields. Fields such as cyber security utilize AD to identify attacks on large servers or faulty virtual machines. The financial sector identifies fraudulent activity in a bank or card activity. Additionally, there has been a significant push in recent years to utilize

AD in medical imaging to aid in identifying rare instances such as brain infractions or tumors as seen in figure 2 [18].

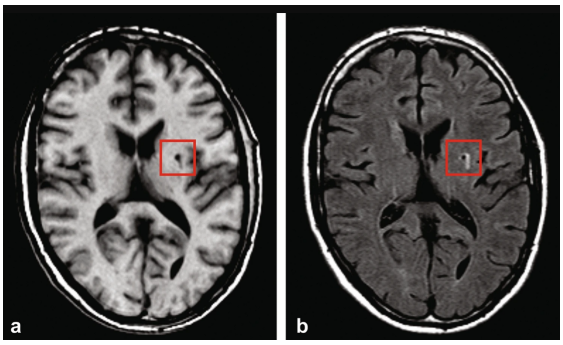


Figure 2: Identification of chronic brain infracts from an MRI

Single instances anomalous to the nominal population are point anomalies, the most common ones. Intuitively, clusters of instances could be considered anomalous to the normal population, and clusters or groups of anomalous instances are group anomalies. For instance, figure 1 shows a single cluster of points labeled as anomalies, C_3, and two point anomalies o_1 and o_2. In the context of fraud, scammers using similar strategies would be considered group anomalies.

Similar to the traditional methods described above, AD has its complications. The first and most notable are class imbalances and unknowns. A class can be defined as a category. In the context of AD, observations are considered as one of two classes: Anomalous or normal. Therefore, a class imbalance refers to the observations classified as anomalous being significantly disproportionate to the class of normal observations. For instance, a bank may struggle to collect large amounts of labeled anomalous instances, thus yielding a disproportionate ratio of labeled anomalies and normal instances. These limitations lead to another challenge of AD, the misclassification of anomalies. There are two types of misclassification in AD: Swamping and Masking.

Swamping is identifying a normal instance as an anomaly. Conversely, masking is identifying an anomaly as a normal instance. Misclassifying an instance as an anomaly or nominal can have dangerous repercussions. For example, an AD algorithm can identify an MRI scan of a brain as nominal when it is truly abnormal, thus containing a growth.

Besides swamping and masking, there are other potential challenges to consider in AD. Another challenge is mitigating the effects of dimensionality, or the "curse of dimensionality." A high-dimensional data set exhibits the following properties: high-volume, high-velocity, and high-variety.

High-volume refers to the sheer size measured in the volume of rows and columns. For instance, financial institutions contain millions, if not billions, of records on customers to model customer behavior and optimize pricing tactics. High-Velocity refers to the rate of new data being consumed and analyzed in real-time. A perfect example is in healthcare, where devices monitor an individual's vitals every second while simultaneously analyzing the data. Lastly, high-variety refers to the diversity of instances for a given data set. As the volume and velocity increase, the structure of data changes in retrospect to the diversity.

In a low-dimensional space, anomalies exhibit apparent behavior. However, the data becomes more non-linear and heterogeneous in a high-dimensional space. As the dimensionality grows, data becomes difficult to generalize accurately and more sparse due to irrelevant features. Sparsity within data can lead to the effect of swamping or masking. The "curse of dimensionality" is relevant to AD as a whole.

Given the complexity of identifying anomalies through AD, choosing a suitable model for the situation is imperative. For instance, some models are less susceptible to the curse of dimensionality than others. There are two types of output in AD algorithms: Scoring and classes. Models producing a score demonstrate the degree to which an observation is outlying. Other models wish to label observations as either anomalies or normal. The following section discusses two popular methods of AD: Local Outlier Factor (LOF) and Isolation Forest (iforest). Further, we apply the iforest algorithm to health economics by exploring anomalies in health care expenditure, utilization, and access to care data within a subset of Medicaid beneficiaries.

1.2 Local Outlier Factor (LOF)

The following section highlights the Local Outlier Factor (LOF) method. The purpose of the LOF method is to compute a score representative of the degree to which a point is outlying. Throughout this section we use a synthetic data set to calculate the components of the LOF score for a point in $\mathbb{R}^2$. Throughout section 1.2 the following notation listed in table 2 are used to define the local outlier factor method.

Notation	Definition
D	Data set of instances
$x_i \in X$	Objects or instances in a data set
C	Cluster (group of similar objects)
M	Parameter for the minimum number of instances
$d(x_1, x_2)$	Distance between the instances x_1 and x_2
$d(x_1, C)$	Distance between x_1 and an instance x_i where $x_i \in C$
$K_d(x_i)$	k-distance of an instance x_i
$N_k(x_i)$	k-distance neighborhood of an instance x_i
$R_k(x_i, x_j)$	Reachability distance of an instance x_i with respect to x_j
$\ell_k(x_i)$	Local reachability distance of an object x_i
$F_K(x_i)$	Local outlier Factor (LOF) of an object x_i

Table 1: Common notation used throughout the LOF Section

The first algorithm in this section is LOF. Two fundamental properties of the LOF method are (1) it is used in multiple dimensions and (2) it reflects an outlier as a degree representing how outlying it is. These properties address the issues of using traditional statistical methods for outlier detection [2]. The LOF algorithm uses distance and nearest neighbor to compute the degree to which a point may be outlying. Let us denote the distance between two points as $d(\cdot, \cdot)$. Distance can be measured using a number of methods. The first, and most popular is Euclidean distance. For two data points in a n-dimensional space euclidean distance is defined as

$$d_e(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{j=1}^{d} (x_j - y_j)^2} \tag{1}$$

where x_j and y_j are the coordinates of the jth attribute. Other distances can be used in place of Euclidean distance. Later on we will discuss the use of different measures of distance in the context of LOF. To begin the LOF method we must define the kth-distance of a point, p_i, in our data set, D. The hyper-parameter k where $k \in \mathbb{Z}^+$ represents the k neighbor of a point. For instance, let us use table 2 with 10 data points, and 2 features to find the k neighbor of each point.

Point	X	Y
A	9	13
B	4	9
C	18	8
D	24	4
E	3	24
F	13	13
G	2	14
H	4	4
I	19	9
J	5	6

Table 2: Data set of points in $\mathbb{R}^2$

First, let us calculate all the distances among every point using euclidean measure. For example, the euclidean distance between AB is computed as $d(A, B) = \sqrt{(9-4)^2 + (13-9)^2} = 6.40$. This is done in the table below for the points A and B. Next, we want to find the kth distance of each object. Denote the kth-distance of an object p_i as $K_d(p_i)$. First, we will define our hyper-parameter k as $k = 3$.

$d_e(\cdot, \cdot)$	Point
6.403124	AB
10.295630	AC
17.492856	AD
12.529964	AE
4.000000	AF
7.071068	AG
10.295630	AH
10.770330	AI
8.062258	AJ

(a) Computed distances of point A

$d_e(\cdot, \cdot)$	Point
6.403124	BA
14.035669	BC
20.615528	BD
15.033296	BE
9.848858	BF
5.385165	BG
5.000000	BH
15.000000	BI
3.162278	BJ

(b) Computed distances of point B

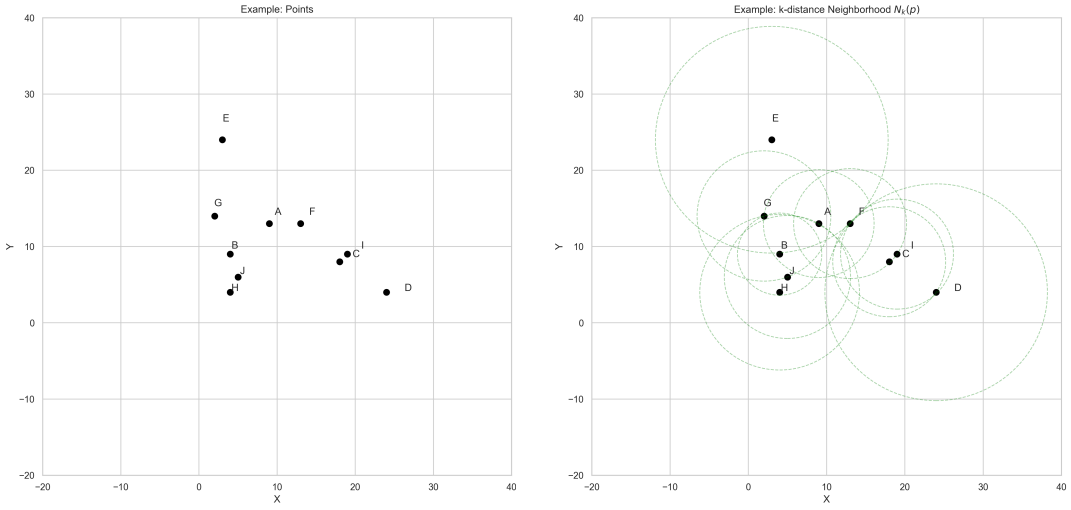
Table 3: Example of computing the kth-distance of a point

By setting $k = 3$, the point 3rd closest to A is G with a distance of 7.071068. Similarly, the point 3rd closest to B is also G with a distance of 5.385165. The

hyper-parameter k dictates the k-distance neighborhood of an object p_i. The neighborhood of a point, denoted as $N_k(p_i)$, is the set of all points whose distance is not greater than the k-distance of the object and is formally defined as

Definition 1.1 (k-Neighborhood of an object x) *The k-Neighborhood of an object* $\mathbf{x}$ *is denoted as* $N_k(\mathbf{x})$ *where D is the data set and $d(\cdot, \cdot)$ is a distance metric.*

$$N_k(\mathbf{x}) = \{\mathbf{y} \in D : d(\mathbf{x}, \mathbf{y}) \leq k\} \tag{2}$$



(a) Plotted points from 2 (b) k-distance neighborhoods

Figure 3: Computing k-distance neighborhoods of all points in table 2

Figure 3a is a plot of all points from table 2. One can visualize the k-distance neighborhood of a point by centering a circle about p_i and setting the radius equal to $K_d(p_i)$. For instance, we verified previously that $K_d(A)$ for $k = 3$ is $K_d(A) = 7.071068$. Therefore, the circle containing all points whose distance from A is not greater than $K_d(A)$ is of the form $(x-9)^2 + (y-13)^2 = 7.071068^2$. Figure 3b shows the k-distance neighborhood of the object A, denoted as $N_k(A)$, is $N_{k=3}(A) = \{F, B, G\}$ and $N_{k=3}(B) = \{J, H, G\}$.

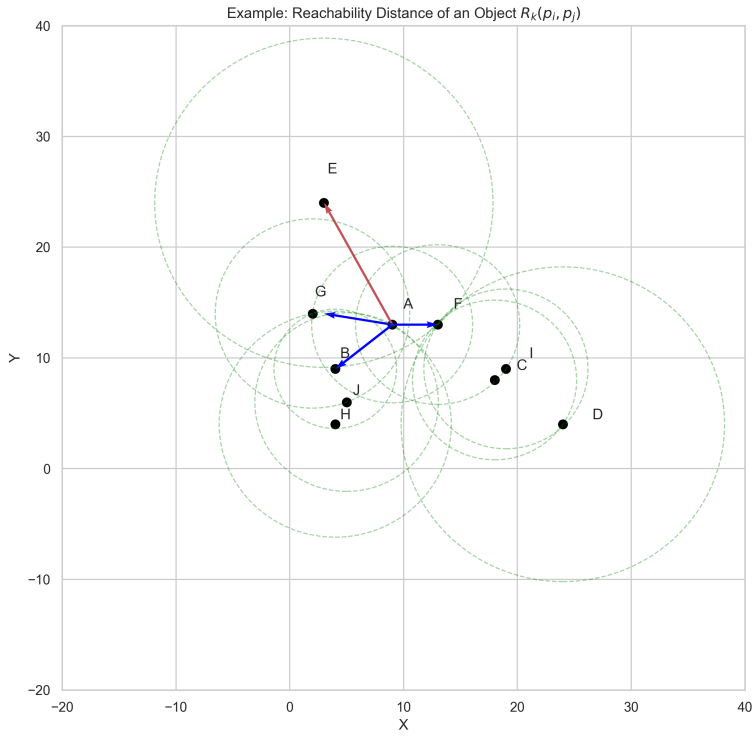


Figure 4: Reachability distance of an object p_i with respect to another p_j

In figure 4 the blue vectors represent the distance between A and its neighbors, $N_k(A) = \{G, F, B\}$. Additionally, the red vector shows a point, E, outside the neighborhood of A. For a point sufficiently close to A, points in the $N_k(A)$, the reachability distance of A with respect to a point in $N_k(A)$ is defined as the $K_d(A)$. Alternatively, a point outside $N_k(A)$ such as E is defined as $d(p_i, p_j)$.

Definition 1.2 (Reachability Distance of a point p_i and p_j) *Let $k \in \mathbb{N}$. The reachabiltiy distance of an object p_i with respect to p_j is defined as*

$$R_k(p_i, p_j) = \max\{K_d(p_i),\ d(p_i, p_j)\} \tag{3}$$

After defining reachability distance, k-neighborhood and k-distance we must derive an equation to identify the density of an object using the reachability distance of every neighbor. The notion of density is akin to comparing the density of an object p_i with respect to its neighbors. To find the density of an object p_i we first calculate the average reachability distance of the object p_i, defined as $\bar{R}_k(p_i)$ with respect to all neighbors of p_i. Next, we must convert the average reachability distance into a density by inverting $\bar{R}_k(p_i)$. The local reachability density of an object p_i is defined below.

Definition 1.3 (Local Reachability density of an object p_i) *The local reachability of a point p_i is defined as*

$$\ell_k(p_i) = 1 \Bigg/ \left(\frac{\sum_{p_j \in N_k(p_i)} R_k(p_i, p_j)}{|N_k(p_i)|} \right) \tag{4}$$

Let us refer to our previous example. Consider the point A and its euclidean distance to all other points in column 3 and $K_d(A)$ in column 4 of table 4. By definition 1.2 we calculate the $R_{k=3}(p_1, p_i)$ in column 5.

	p_1	p_j	$d_{euc}(\cdot,\cdot)$	$K_d(p_j)$	$R_{k=3}(p_1, p_{j,j\neq 1})$
1	A	B	6.403124	5.385165	6.403124
2	A	C	10.295630	7.211103	10.295630
3	A	D	17.492856	14.21267	17.492856
4	A	E	12.529964	14.86607	14.86607
5	A	F	4.000000	7.071068	7.071068
6	A	G	7.071068	18.544004	18.544004
7	A	H	10.295630	10.19804	10.295630
8	A	I	10.770330	7.211103	10.770330
9	A	J	8.062258	8.062258	8.062258

Table 4: Example: Computed reachability distances

Using column 5 of table 4 we can calculate $\ell_{k=3}(A)$ by first computing the average reachability distance.

$$\begin{aligned}\frac{1}{|N_k(A)|}\sum_{p_j\in N_K(A)}\bar{R}_k(p_j) &= \frac{1}{3}\Big(R_{k=3}(A,B)+R_{k=3}(A,F)+R_{k=3}(A,G)\Big)\\ &= \frac{1}{3}\Big(6.403124+7.071068+18.544004\Big)\\ &= 10.672732\end{aligned}$$

To convert the average reachability distance from above to a density we will take its inverse by inverting it, $\ell_K(A) = \frac{1}{10.672732} = 0.093696722$

Using definition 1.3, the outlier factor for an object p_i can be computed by dividing the summing the proportion of local reachability distances of an object p_j with respect to p_i.

Definition 1.4 (Local outlier Factor of an object p_i) *The local outlier factor, or degree to which a point p_i is outlying is*

$$F_k(p_i) = \frac{\sum_{p_j \in N_k(p_i)} \frac{\ell_k(p_j)}{\ell_k(p_i)}}{|N_k(p_i)|} \tag{5}$$

The value $F_k(p_i)$ shows the degree to which a data point is outlying. A low value for LOF indicates the object p_j has a low density, therefore it is considered an anomalous instance. Conversely, a high local outlier factor score indicates an observation is an inlier. As one may gather, the LOF algorithm is robust in identifying anomalies, however, it is susceptible to a number of limitations. The first being computational complexity. Calculating the distance between two points, let alone a large dataset is extremely expensive. Therefore LOF is not a desirable method for big data. Also, the structure of the data may pose significant risk to misclassifying an anomaly versus a normal data point. In the next section the algorithm iforest is discussed at length. The iforest algorithm helps mitigate the limitations constraining the LOF method

1.3 Isolation Forest

The iforest algorithm is a powerful method used in anomaly detection. Unlike LOF which uses a distance based measure to calculate an anomaly score, the iforest uses the properties of binary search trees (BST), more specifically a forest of BST to fully isolate a data point. First, to understand the algorithm one must be familiar with graphs, trees and BST. Throughout the following section we will use [4] to define components of iforest with our notation defined in table 5.

Notation	Definition
$G = \langle V, E \rangle$	Graph
$P = \langle V, E \rangle$	Path
$V(G)$	Set of vertices or nodes
$E(G)$	Set of edges
T	Tree
T'	Sub-tree where $T' \subseteq T$
T_l	Left-child node of a BT or BST
T_r	Right-child node of a BT or BST
r	root
I	Total internal nodes in T
L	Total external nodes in T
N	Total nodes in T
$\bar{d}$	Average depth
$\|A\|$	Cardinality of the set A
$H(N-1)$	$(n-1)^{\text{th}}$ harmonic number
$E(n)$	External path length of unsuccessful BST
$c(n)$	Average depth of an external node

Table 5: Common notation used throughout Isolation Forest Section

1.3.1 Graphs and Trees

A graph is a set $G = \langle V,\ E\rangle$ satisfying $E \subseteq [V]^2$ where the elements of V are the vertices or nodes and the elements of E are the edges [4]. Figure 5 shows a graph G with vertices $V = \{1, 2, 3, 4, 5, 6\}$ and edges $E = \{\{1, 2\}, \{2, 6\}, \{6, 1\}, \{3, 4\}, \{4, 5\}\}$. The set of all vertices is denoted as $V(G)$ and the set of all edges is denoted as $E(G)$. We can represent a graph visually using a point or ball as a node and a line joining the two nodes will represent an edge. For example, the edge $\{1, 2\}$ connects the nodes 1 and 2 in figure 5.

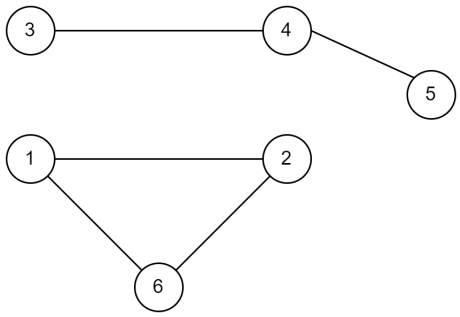


Figure 5: A simple graph $G = \langle V,\ E\rangle$

A walk from a to b is a sequence of vertices $a_1, a_2, \ldots, a_k$ such that $a_1 = a$ and $a_k = b$ and $\{a_i, a_{i+1}\} \in E$ for each i. For instance, in figure 5 the walk from node 1 to 6 could be $1, 6$ or $1, 2, 6$. A path in a walk with no repeated vertices. Furthermore, the length of a path is the number of edges traversed. A graph G is cyclic if there is a path from a vertex onto itself. For instance, the graph G in figure 5 is cyclic since there is a path from 1 onto itself as $P = \{\{1, 2\}, \{2, 6\}, \{6, 1\}\}$. Using the definitions of path and walk, a graph G is connected if any two of its nodes, a and b, are linked by a path from a to b.

A tree T is a connected graph with no cycles or acyclic. In some cases, a tree will contain a unique node called the root. A tree with a root imposes direction on all edges via all paths that start at the root. In figure 6, the root $r = a_0$

of our tree T is the least element of $V(T) = \{a_0, a_1, \ldots, a_8\}$. A node of a tree T is considered external if it is connected to only one edge. If $\{a, b\}$ is an edge in a rooted tree then a is the parent of b and b is the child of a. Moreover, an internal nodes is any node with a child. The set of internal nodes is denoted as I. Note, the root node is considered an internal node. For example, in figure 6 the node a_2 is internal since it contains a parent in $r = a_0$, and children (a_5, a_6). Let a be a node in a tree, then the depth of a is the path length from the root to a. Furthermore, the height of a tree T is the maximum depth value of a node. Consider figure 6, the depth of the internal node a_4 is 2, and the height of our tree T is 3. For a rooted tree a sub rooted tree is a truncated branch of the tree (i.e. Pick an arbitrary node and the tree generated from all the children is the sub rooted tree. For instance, in 6 the nodes $\{a_4, a_7, a_8\}$ form a sub-tree T' where $T' \subseteq T$.

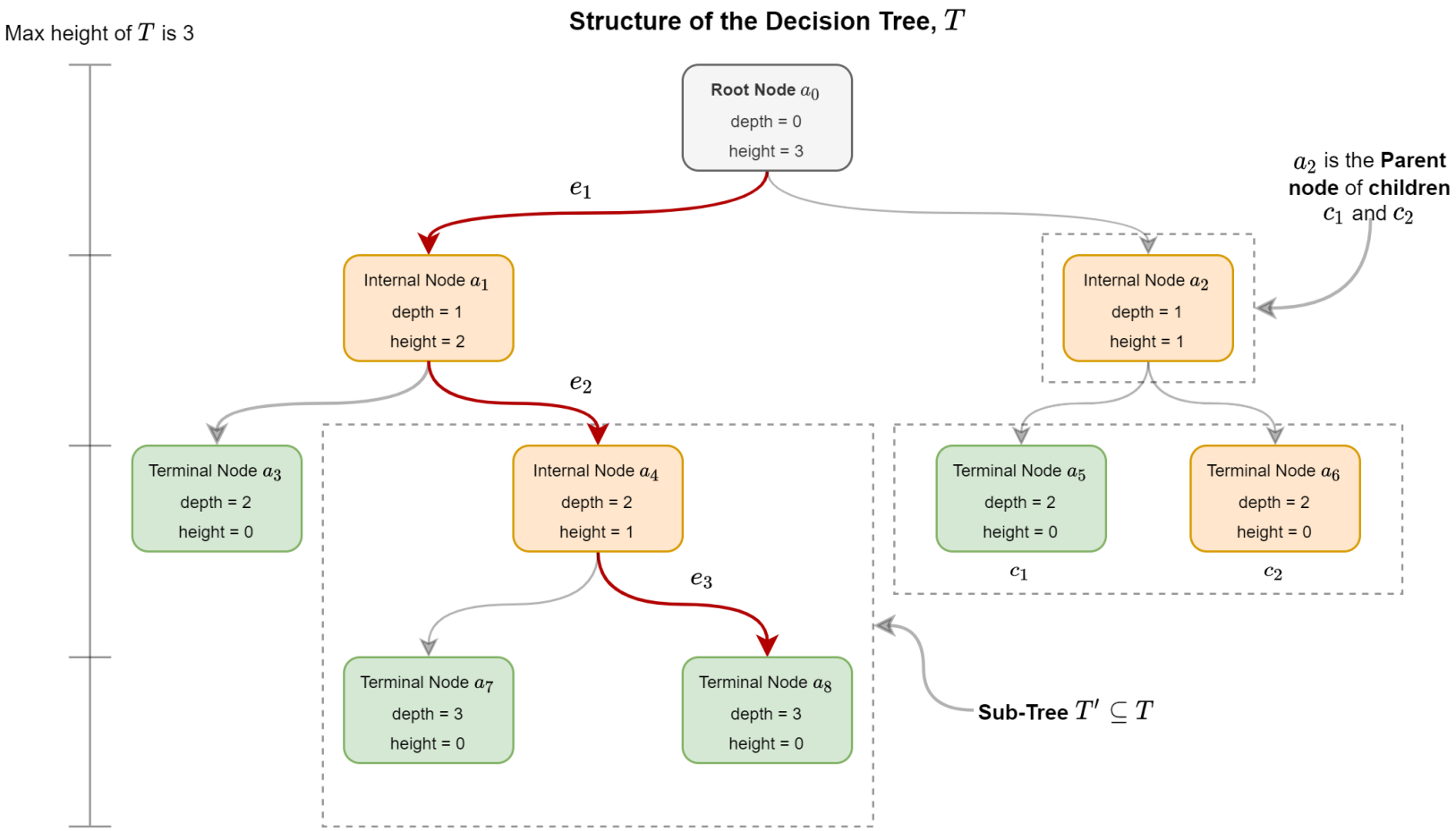


Figure 6: Schematic of a binary decision tree T

1.3.2 Binary Trees (BT)

An M-ary tree limits the number of children in a tree. A special case of M-ary Trees are Binary Trees (BT) where each node possess at most $M = 2$ child nodes. For an M-Ary tree the following properties hold.

Theorem 1.1 (N-Ary Trees) *Let T be an N-Ary tree with $n \geq 0$ internal nodes and height $h \geq 0$, then*

- *The tree contains $(N-1)n+1$ external nodes*
- *The maximum number of internal nodes in T is given by*

$$\frac{N^{h+1}-1}{N-1}$$

and the maximum number of external nodes is given by N^h

There are many different types of BT, however, we will focus our attention on a special type, a proper or full BT. The proper BT is a tree where every node has either exactly two children or zero. For a proper BT, the following properties hold [13].

Theorem 1.2 (Properties of Proper Binary Tree) *Let T be a non-empty, proper binary tree then the following hold*

(i) If T has internal nodes I then number of leaves is $L = I + 1$

(ii) If T has internal nodes I then total number of nodes is $N = 2I + 1$

(iii) If T has a total of N nodes then number of internal nodes is $I = (N - 1)/2$

(iv) If T has a total of N nodes then number of external nodes is $L = (N + 1)/2$

(v) If T has L external then total number of nodes is $N = 2L - 1$

(vi) If T has L external nodes then number of internal nodes is $I = L - 1$

Consider the following example. By Theorem 1.2 the proper BT T in figure 7 has

(i) $I = 3$ internal nodes and $L = I + 1 = 3 + 1 = 4$ external nodes

(ii) $I = 3$ internal nodes and total number of $N = 2I + 1 = 2(3) + 1 = 7$ nodes

(iii) $N = 7$ nodes and $I = (N - 1)/2 = (7 - 1)/2 = 3$ internal nodes

(iv) $N = 7$ nodes and $L = (N + 1)/2 = (7 + 1)/2 = 4$ external nodes

(v) $L = 4$ external nodes and $N = 2(4) - 1 = 7$ nodes

(vi) $L = 4$ external nodes and $I = 4 - 1$ internal nodes

For a full or proper BT, Theorem 1.2 works nicely such that if N, L, or I is known you can find the other two.

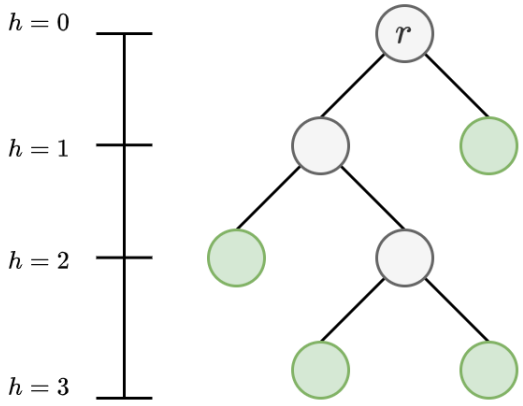


Figure 7: Proper Binary Tree

1.3.3 Binary Search Trees (BST)

In computer science a key is defined as a value within an object. In some cases, keys are values associated with the node of a tree. A BST is a special type of BT consisting of a finite set of unique keys. These trees are often used for performing efficient search operations on keys. Moreover, when a BST splits, the node to the left must contain a smaller number than the root node. Alternatively, the node to the right must contain a larger number than the root node. For instance, we may want to determine if a unique key inside a BST is present or not. There are two main types of searches performed in BST analysis: Successful search, and unsuccessful search. During a successful search $d + 1$ internal nodes are visited where d is the depth of our key being searched for. For instance, if we are searching for the key 7 in our BT in figure 8, then 4 nodes are visited. Since we are unaware of which node will contain the correct key, it is assumed the node will be present in any node of the BT. Therefore, we take the average number of nodes visited

during a successful search is $\bar{d}+1$ where $\bar{d}$ is the average depth of the nodes for a tree [13]. For a BST with $n > 0$ nodes the average depth is given by

$$\bar{d} = \frac{1}{n}\sum_{i=1}^{n} d_i \tag{6}$$

where d_i is the depth of the ith node. The average depth is referred to as the internal path length. However, the internal path lengths for different types of BST are not equal.

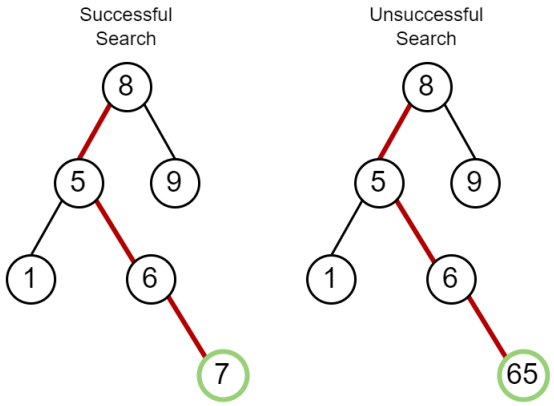


Figure 8: Example of successful and unsuccessful search of a BST with keys

Average case analysis pertains to measuring the average performance of an algorithm and its computational complexity. For a BST, average case analysis is performed with respect an unsuccessful and successful search by computing the internal and external path lengths. The average external path length of an unsuccessful search of a BST with $2n$ internal nodes is defined as $E(n) \approx 2(n+1)H-2n$ where H is the harmonic number. The average depth of an external node of a BST is defined as $c(n) \approx 2H - 2n/(n+1)$. The value of $c(n)$ demonstrates the average number of nodes visited when traversing a BST.

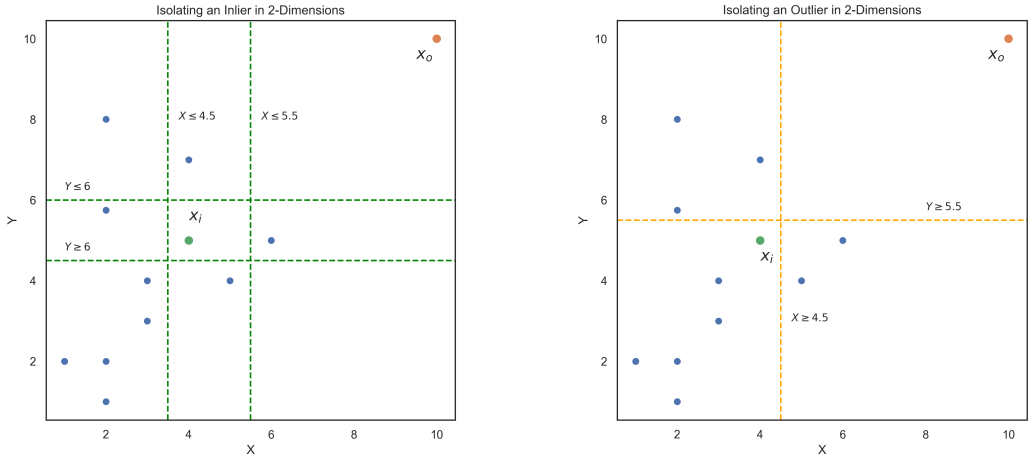
1.3.4 Introduction to Iforest

Up to now, we have defined the key terminology to begin discussing the intricacies of the isolation forest algorithm. Next, we will define new terms as they pertain to isolation forests and derive important values used in the calculation of an anomaly score. The iforest algorithm builds a forest of proper or full BT. Every proper BT within the forest is called an isolation tree (itree). Ultimately, the algorithm recursively partitions the data randomly using isolation trees until a data point is fully isolated. Note, many definitions in the following pages come directly from the original Isolation Forest paper [9].

To demonstrate the concept of recursive partitioning consider the following example. Figure 9 shows a set of data with two features x_1, and x_2. The data point x_i is a normal instance, and x_o is an anomaly. To isolate these data points from all others, conditions that filter the data are applied until x_i and x_o are isolated. The dashed vertical, and horizontal lines in figure 9 indicate a partition of the data and associated filtering condition. Table 6 shows the order or partitioning in column 1, the condition applied to the data in column 2, and column 3 displays the number of data points, N, after partitioning the data in column 2. The point x_i is isolated in 4 partitions of the data. Whereas the point x_o is isolated only in 2 partitions.

Isolating x_i			Isolating x_o		
Order	Split Condition	N	Order	Split Condition	N
1	$x \leq 4.5$	4	1	$x \geq 4.5$	3
2	$y \leq 6$	3	2	$y \geq 5.5$	1
3	$y \geq 6$	2			
4	$x \leq 5.5$	1			

Table 6: Table of partitions from figure 9



(a) Isolation of the instance x_i (b) Isolation of the instance x_o

Figure 9: Example of isolating the instances x_i and x_o with data partitions

This example demonstrates the fundamental component of the iforest algorithm: Anomalies are more likely to be isolated in fewer partitions of the data. The number of partitions required to isolate a point is represented by the path length from the root node to an external node. As demonstrated in the previous example, the model assumes the following with regards to normal and abnormal data points.

1. An *anomalous* data point will be isolated closer to the root node of a tree (i.e., shorter path length)

2. A *normal* data point will be isolated further away from the root node of a tree (i.e., long path length)

1.3.5 Isolation Trees

As mentioned in section 1.3.4, an isolation forest is made of an ensemble of itrees. Each itree within the ensemble is created with a unique set of partitions. These partitions are randomly generated by also randomly selecting an attribute q from our dataset and split value p between the minimum and maximum values of q [9]. For instance, the algorithm begins with the sample data $X = \{x_1, \ldots, x_n\}$ with n observations. The sample data is represented by the root node of the itree in figure 10.

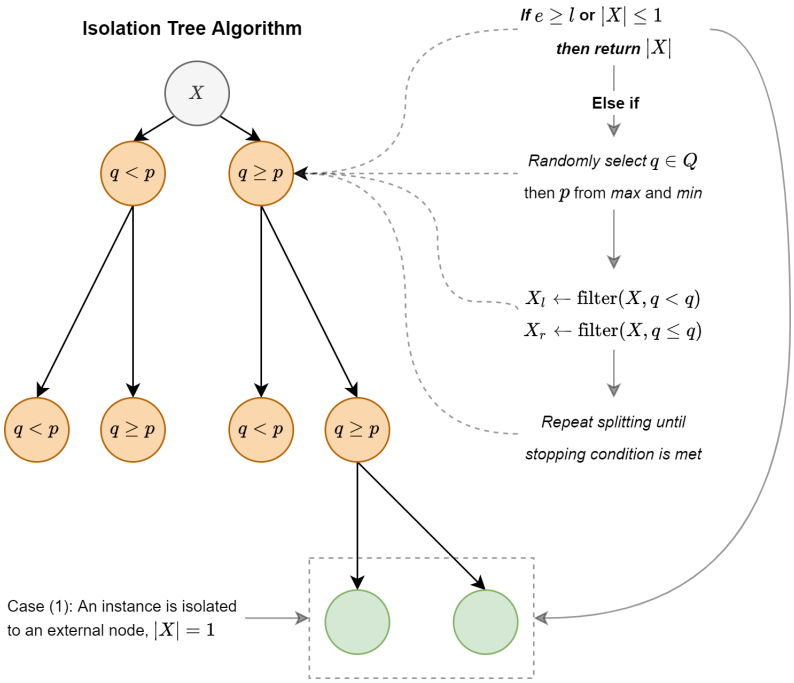


Figure 10: Schematic of an isolation tree

The dataset X is then recursively partitioned into sub-samples of the data. For instance, in figure 10 the root node is split into two additional nodes by partitioning the data using the criteria $q < p$ and $q \geq p$. The left and right nodes both contain a sub-sample of the data, denoted as X_l and X_r which are disjoint.

The algorithm terminates recursive partitioning until one of three conditions are met.

1. The tree reaches a height limit ℓ
2. An instance is completely isolated to an external node where $|X| = 1$
3. Or all data inside X contain the same value

Using the structure of a fully built itree, an anomaly score is derived using the path length, $h(x)$, of a data point x_n passed through the itree. The path length of the point x_n is measured by the number of edges traversed from the root node to an external node. Using the properties of proper BT and BST (defined in 1.3.2 and 1.3.3) the average path length $h(x)$ is estimated using the path length of unsuccessful search of BST. The average path length of an unsuccessful search in BST is defined as

$$c(n) = 2H(n-1) - (2(n-1)/n) \tag{7}$$

In equation 7 $H(N-1)$ is the harmonic number where $N-1$ is the $(N-1)^{\text{th}}$ harmonic number. The harmonic number or series is the summation,

$$H_n = \sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \ldots + \frac{1}{i}$$

This value can be estimated by the most common approximation of $H_n \approx \ln n + \gamma$ where γ is Euler's constant. Euler's constant is approximated by
$\gamma = \lim_{n\to\infty}(H(N-1) - \ln n) \approx 0.577215$.

For an observation n the anomaly score is given by,

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{8}$$

where $E(h(x))$ is the average of $h(x)$ from a collection of itrees. For an instance x, the greater $E(h(x))$ is in proportion to $c(n)$ the more anomalous an instance becomes. For instance, if the average path length across all itrees for an observation x is $E(h(x)) = 5$ and the average path length of an unsuccessful search in BST is $c(n) = 2$ then $s(x, n) = 0.75$. Conversely, if $E(h(x)) = 2$ and $c(n) = 5$ then $s(x, n) = 0.17$. A score of $s(x, n) = 0.75$ is representative of an observation with a *shorter* path length than a score of $s(x, n) = 0.17$ with a *longer* path length. Therefore,

- Instances close to $s = 1$ are considered anomalies
- Instances smaller than $s = 0.5$ are considered normal
- If all instances are $s \approx 0.5$ there are no definitive anomalies

This unique method has a number of beneficial characteristics. For instance, unlike distance-based or classification-based methods which construct a profile of normality, the iforest algorithms identifies anomalies. Additionally, iforest has low computational complexity unlike LOF or any other distance-based method. Distance-based methods contain complicated calculations such as euclidean distance and k nearest neighbor. Additionally, iforest has the ability to handle high dimensionality data.

2 Identifying Individuals with Abnormal Health Care Expenditures and Utilization using Anomaly Detection

2.1 Introduction

Most current literature analyzes the effects of premiums, cost-sharing, co-payments, and un-enrollment on increasing Medicaid spending [10]. These studies focus on identifying correlations between medical services received by beneficiaries and spending; however, they do not examine the anomalous population that skews the balance of spending among various types of medical services. Anomaly detection (AD) is a powerful tool for identifying and scoring observations that fall outside what is considered "normal" spending among Medicaid beneficiaries. AD's objective is to score observations across multiple features based on how outlying they are to other observations.

The study of anomalies has a long history of discovery in physics and chemistry. For example, Lord Rayleigh discovered the element argon through a discrepancy in the data during a routine experiment calculating the density nitrogen gas [19, 15]. The use of AD is prevalent in Health Care, specifically, financial fraud. Fraud is an ongoing issue in the healthcare insurance industry, costing public and private healthcare programs between $75 - $250 billion in 2009 [8]. Studies have employed AD to identify potentially fraudulent claims in Medicaid to reduce savings and maintain quality of care standards [17]. In the private sector, identifying anomalies that represent fraudulent behavior is more accessible due to high quality and real-time data. Alternatively, Medicaid has less accurate and lagged real-time data making it difficult to catch fraud efficiently. Addressing fraud in Medicaid

is essential; however, data quality and access to labeled observations of fraud limit the potential benefits of reducing spending. Nevertheless, AD application in Medicaid beneficiary data does not have to be limited to fraud detection.

Anomalies need not be destructive and instead informative. Our objective is to use AD to discover and characterize clusters of Medicaid beneficiaries with abnormal health care expenditures and utilization. The identification and profile of these clusters have two main benefits. (1) First, understanding the population representative of abnormal spending and utilization can aid in targeting policy to mitigate these expenditures. (2) As a result of identifying the abnormal population, we define an average population potentially representative of intentional policy action

2.2 Data

2.2.1 Data Source

The data used in this analysis is provided by the Agency for Healthcare Research and Quality (AHRQ). The MEPS data source, a component of the AHRQ, provides information regarding the cost and use of health care in the United States. This study is specifically concerned with data sourced from the Medical Expenditure Panel Survey (MEPS) House Hold component (HC-209) 2018 [7]. The MEPS Household Component contains health expenditure, health insurance coverage, health status, employment, and many other characteristics on an individual and family level. Households surveyed five times within a two-year period. Moreover, information is gathered using computer assisted personal interviewing (CAPI) software. Data is gathered from every household, and the survey builds upon this information every interview.

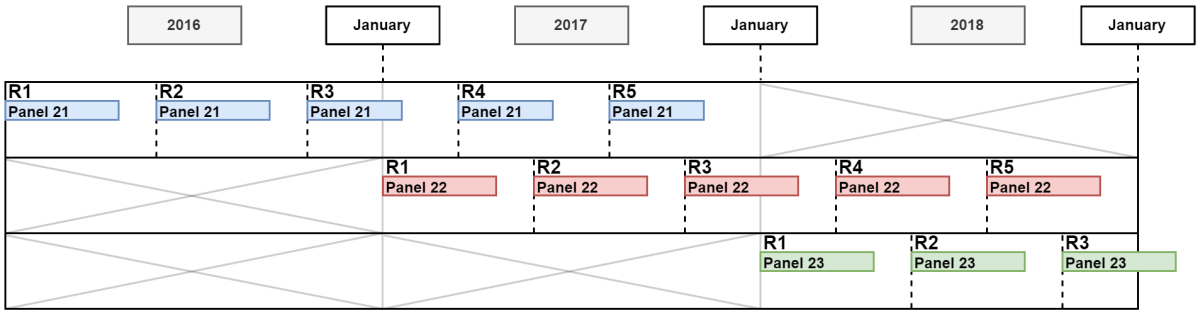


Figure 11: Outline of MEPS Panel Survey structure

The structure of the MEPS Household Component is as follows. The data is collected from the National Health Interview Survey (NHIS). Throughout the surveying process, interviews are conducted over a 2 year period with a household member.This household member is the age of the legal majority. Information is then provided by the household member on all household occupants (this condition is consistent with both single, and multi-family households). For example,

in figure 11 a set of participants are represented by the panel number. Over a span of 2 years, households are interviews in a total of 5 rounds. In figure 11 the rounds are indicated by R1, R2,...,R5.

2.2.2 Data Cleaning

The household component MEPS 2018 data set contains 30,461 observations and 1501 features. These features are related to several categories: health care utilization, expenditures, insurance, access to care, health status, demographics, and geographics. For this study, we will be focusing our attention on features about health care expenditures (see table 12), utilization (see table 13), and access to care 14. Administrative features (see table 11), such as person ID, and dwelling unity ID are utilized to consolidate and clean the data. The data is on the individual level, with each individual belonging to a unique code, DUID, indicating the dwelling unit a person belongs to. Furthermore, each DUID has a reference person indicated by the binary feature FAMIDYR where .

To obtain a family-level data set, we first filter the data by the family weight variable, FAMWT18F, and select all observations greater than zero. According to the MEPS documentation, we select individuals with a positive FAMWT18F for family-level estimates and analysis. Secondly, we concatenate the features DUID and FAMIDYR to create a new variable, DUIDFAMY. This feature allows us to aggregate the data by dwelling unit using the reference person.

For instance, table 8 contains data for a single dwelling unit. Each record represents a different family member, and the column FAMIDYR indicates which family member is the reference person. After sorting the data to one record per DUIDFAMY value and retaining the reference person, we finish with an aggre-

gated, family-level data set.

We use different functions to obtain family-level estimates for numeric and categorical features during aggregation. For instance, all expenditure and utilization variables are continuous, and therefore, we take the sum of all members for a unique dwelling unit. Additionally, all access to care features are categorical, and therefore, we take the maximum of all members for a unique dwelling unit. Some features contained extraneous values. For instance, a family unit may respond to a survey question with a value of -1. This value has a unique definition. If a respondent answers a survey question with any value in column 1 of table 7, then the row should be removed. Most observations were complete, meaning they did not contain these codes.

Code	Definition
-1	Question was not asked due to skip pattern
-2	Determined in a previous round
-7	Question was asked and respondent refused to answer question
-8	Respondent did not know the answer or information could not be retrieved

Table 7: Unique reserve codes from MEPS HC-209 data

For example, in tables 8 and 9, the DUID = 229002 retains the value for the categorical variable POVLEV18, poverty level as defined by the census. POVLEV18 is not used in the model, however, is a feature in the original data set. Since the access to care features represent the family's response, family members have no varying values. Lastly, we are solely interested in a population only enrolled in Medicaid. Therefore, the feature MCDHMO31, covered by Medicaid/SCHIP health maintenance organization is utilized to filter the population to families enrolled in Medicaid. These conditions ensure that all observations were enrolled in Medicaid throughout the entire survey period. Finally, after dropping features not used in modeling, we are left with a dataset of $n = 1,541$ observations and $q = 31$ features.

Index	DUID	PID	DUIDFAMY	FAMIDYR	POVLEV18
2	2290002	101	2290002A	1	163.92
3	2290002	102	2290002A	0	163.92
4	2290002	103	2290002A	0	163.92
5	2290002	104	2290002A	0	163.92
6	2290002	105	2290002A	0	163.92
7	2290002	106	2290002A	0	163.92

Table 8: Records from a single family unit or unique DUID

Index	DUIDFAMY	FAMIDYR	POVLEV18	RXMCD18
0	229001A	1	190.31	1103
2	2290002A	1	163.92	0
8	2290003A	1	1013.48	0
13	2290005A	1	203.18	0

Table 9: Records from multiple family units after aggregating data

2.3 Methodology

2.3.1 Model Pipeline

The methodology used in this study is a multi-stage process. Below in figure 12 is a process diagram of the methodology. In the first stage iforest scores each observation (i.e., family unit) in our input data X. The scores are transformed into a binary dependent variable $y_i = \{1, 0\}$ using a decision function where an observation labeled as $y_i = 1$ is considered an *anomaly* and an observation labeled as $y_i = 0$ is considered a *normal instance*. The second stage consists of modeling our original data X using a Gradient Boosted Random Forest classification model with $y_i = \{1, 0\}$ as the dependent variable. In the third and final stage, methods of model interpretation such as Partial Dependence Plots (PDP) and Feature Importance (FI) charts help indicate the features, and values driving the population of anomalous and normal observations.

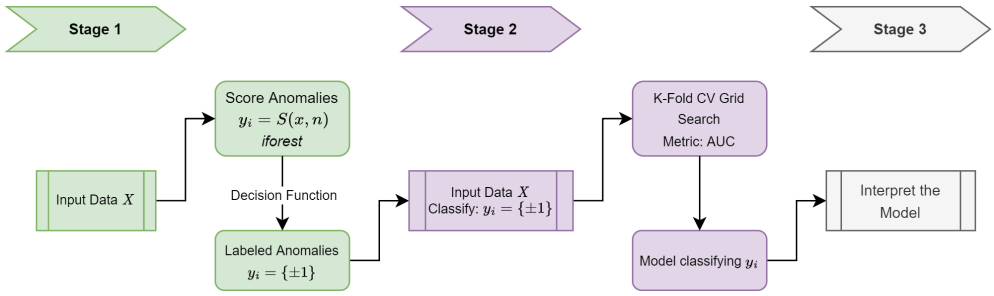


Figure 12: Model Pipeline

2.3.2 Anomaly Detection

In an AD algorithm, the objective is to identify observations that significantly deviate from other observations within the data. To quantify the level at which an observation deviates from the rest is to create an anomaly score. An anomaly score is a quantitative measure of how normal or abnormal an observation may be to other instances in a multidimensional dataset [11]. The computation of an anomaly score is dependent upon the algorithm. Algorithms such as LOF use the relative density of observations defined by an instance's neighborhood density as the anomaly score. Rule-based techniques utilize learning algorithms such as decision trees and random forests to isolate the normal and abnormal observations. Other algorithms, such as One Class Support Vector Machines (OC-SVM) use linear separators or hyper-planes to partition the data and distinguish an anomalous observation from a normal observation [3].

AD is a toolbox of methods, and the choice of algorithm depends on numerous factors relevant to the structure of the data. The first factor is the dimensionality, or the number of features and observations, in the dataset. As dimensionality increases, the potential for extraneous features concealing the identity of true anomalies is known as the "curse of dimensionality." To this effect, we define a normal instance identified as an anomaly as swamping. Conversely, any anomaly identified as a normal instance is known as masking [11, 16].

For example, we restrict the data to three feature categories: healthcare expenditures, utilization, and access to care. Constraining the data to these categories reduces the likelihood of an extraneous or redundant feature biasing the model's scoring methodology. For instance, adding a redundant feature to the model could change an observation labeled as an anomaly, $y_i = 1$, to a normal observa-

tion $y_i = 0$.

A second factor is the types of "potential" anomalies encountered throughout the data. Anomalies completely isolated with low-density neighborhoods are "point anomalies." Alternatively, we define a group of anomalous observations as clusters or group anomalies. For example, the distributions of expenditures and utilization are highly skewed and dense (i.e., close to zero). Consequently a majority of labeled outliers are assumed to be point anomalies or heterogeneous.

The final factor in choosing an AD algorithm is to consider the learning paradigm. There are two main types of learning problems: Supervised and unsupervised. Supervised learning uses labeled training and test data with normal and abnormally labeled observations. Unsupervised learning focuses strictly on patterns within the data to draw inferences. In the context of AD, the algorithm will make its best guess as to who is considered *normal* or *abnormal*. Most research in AD focuses on unsupervised learning since most anomalies in data are undefined.

Considering the above factors, our study implements an AD algorithm called iforest. Iforest is a rule-based algorithm which isolates an observation by partitioning the data recursively using binary decision trees. The model assumes an anomaly will take fewer partitions of the data to isolate than a normal instance [9]. Moreover, this study implements an unsupervised iforest since we do not know who is considered abnormal within the population. This allows the model to draw inferences as to what combination of variables and values from the three feature categories define an observation as normal or abnormal.

2.3.3 Scoring Anomalies

As mentioned previously, the iforest algorithm recursively partitions the data X to completely isolate an observation from all others. The process of recursive partitioning is performed using a ensemble of proper binary trees. A single binary tree is considered an itree and an ensemble of binary trees are considered an iforest. To produce an anomaly score for an observation, the algorithm equates the number of partitions required to isolate an observation for a given itree to the path length from the root node to an external node. The model is built upon two main assumptions.

1. There are significantly fewer anomalies than normal instances
2. Anomalies contain features with significantly different values than normal instances

Based off the above assumptions, it will take fewer partitions of the data to isolate an anomaly than a normal instance. This suggests an anomaly will be isolated with a shorter path length than a normal instance. In contrast, a normal instance will be isolated with a longer path length.

To calculate the anomaly scores for this study we use the python package Sklearn and class `sklearn.ensemble.IsolationForest` [12]. The class specifies a number of hyper-parameters. A hyper-parameter is a parameter which controls the learning process of a model. For instance, in the case of iforest one can control the number of estimators, isolation trees, used throughout the model with `n_estimators`. A total of `n_estimators = 150` are used in the final model and the hyper-parameter `bootstrap` is used to control if the `n_estimators` are fit on random subsets of the data. Bootstrapping is a re-sampling method which uses random sampling with

replacement. The `bootstrap` hyper-parameter is set to `bootstrap = True` meaning the estimators in the forest are fit on random subsets of the data with replacement.

2.3.4 Modeling Anomalies

While scores help identify which instances are considered anomalies or normal *iforest* does not indicate which features are driving an observation to be isolated from the rest. To identify potential features as drivers in the anomaly scores, generated in stage 1 in figure 12, all scores are transformed from a continuous value to a discrete one using the following criteria in stage 2 of figure 12.

- If $s \geq 0$ then the observation x_i is considered *normal* and labeled $y_i = 0$
- If $s \leq 0$ then the observation x_i is considered *abnormal* and labeled $y_i = 1$

This results in a new column indicating whether an observation is anomalous or normal. The criteria above is a product of the computation of the score in the original paper on iforest [9].

The method of K fold cross validation (CV) is used to test the prediction error on separate partitions of the data. For instance, the model uses $K = 5$ folds meaning the data is split into $K = 5$ partitions and the model is fit to $K - 1$ parts. The kth part is then set aside for validation. To find an optimal model, Grid Search CV is applied to find the optimal algorithm between Logistic Regression, Random Forest, and Gradient Boosted Random Forests. Applying Grid Search CV with $K = 5$ yielded the optimal model being a Gradient boosted Random Forest classifier.

2.4 Results

2.4.1 Feature Importance

Figure 13 is a visual representation of the anomalies created using *iforest*. Each data point represents an instance or family and their respective score. If an instance has a negative score, then they are an anomaly. Conversely, if an instance has a positive score, then they are considered normal. Furthermore, the more negative an instance, the more anomalous they are with respect to the population.

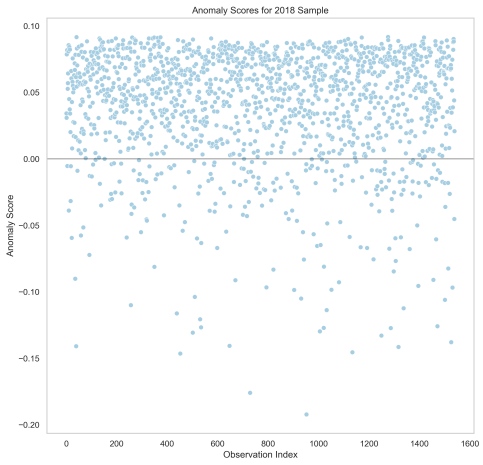


Figure 13: Scatter plot of anomalies produced by iforest

To interpret the result of the classification model split-improvement Feature importance is used as an indication of how influential a feature is on the model predictions. These values are calculated using the default split improvement feature importance in Sklearn [12]. Split-improvement aggregates the improvement of each split of a node in a decision tree or random forest. To calculate the split-improvement feature importance, first let $Q_m(T)$ be the impurity function, deviance, for a node m of a tree T.

$$Q_m(T) = -\sum_{k=1}^{K} \hat{p}_{mk} \log_2 \hat{p}_{mk} \tag{9}$$

Where p_{mk} is the proportion of class k in the node m, n_m is the number of observations in node m and $\mathbb{I}(y_i = K)$ is the indicator function for a class K. Note, node impurity is only used on tree-based models such as gradient boosted random forests, and classification.

$$p_{mk} = \frac{1}{n_{mk}} \sum_{x_i \in m} \mathbb{I}(y_i = K) \tag{10}$$

For example, consider the following dataset in table 10 with two features x_1, x_2 and dependent variable $y_i = \{1,\ 0\}$.

x_1	x_2	y_i
1	1	1
1	1	1
1	1	0
1	0	1
1	0	0
0	1	1
0	1	0
0	0	0

Table 10: Example dataset for classification

The goal is to classify whether an observation is $y_i = \{1, 0\}$ based off its attributes x_1 and x_2. To do so, let us build a decision tree. To begin, let the root node contain all samples in the data set. From here, we will split the root node into a left, l, and right, r node based off its value for x_1. For instance, figure 14 shows the root node containing all observations of class $y_i = \{1, 0\}$. For ease of interpretation, consider $y_i = 1$ as the green dot and $y_i = 0$ as the plus sign. To calculate the feature importance we first need to calculate the node impurity

using equation 9.

For example, the deviance impurity for the root node, $m = 1$ is

$$Q_{m=1}(T) = -\frac{5}{8}\log_2\frac{5}{8} - \frac{3}{8}\log_2\frac{3}{8} = 0.954$$

This process is then performed across all nodes of the decision tree. For instance, figure 14 shows the impurity calculations for all nodes in the decision tree T.

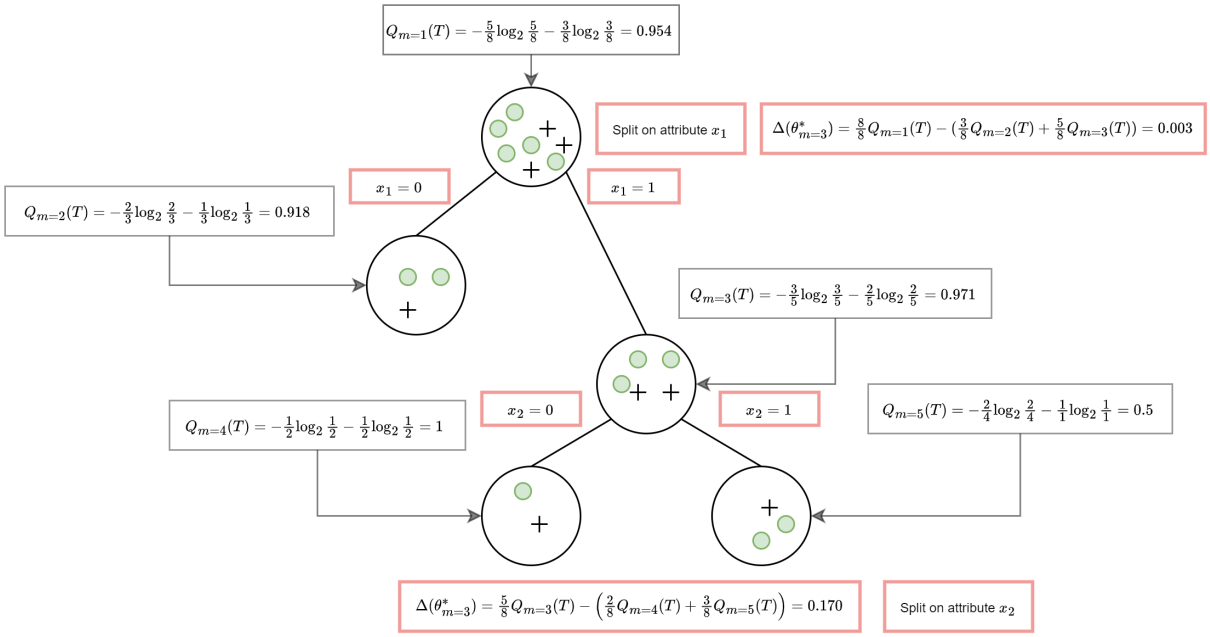


Figure 14: Calculating split-improvement feature importance for a decision tree

Let the best split at m be θ^*_m by splitting the jth variable resulting in two child nodes l,r. The decrease in impurity for θ^* is defined as

$$\Delta(\theta^*_m) = \omega_m Q_m(T) - (\omega_\ell Q_\ell(T) + \omega_r Q_r(T)) \tag{11}$$

where ω is the proportion of observations falling into each node are $\omega_m = \frac{n_m}{n}$, $\omega_\ell = \frac{n_\ell}{n}$ and $\omega_r = \frac{n_r}{n}$. To calculate the feature importance for the jth feature we add all $\Delta(\theta_m^*)$ where the split occurs at the $j$the variable. Therefore, the feature importance for the jth feature is given by

$$VI_j^T = \sum_{m, j \in \theta_m^*} \Delta(\theta_m^*) \tag{12}$$

Continuing the previous example in figure 14, to calculate the feature importance of x_1 and x_2 we simply take the decrease in impurity at split $m = 1$ and $m = 3$ since there are no other splits. This results in the following feature importance.

$$VI_{j=1}^T = \frac{8}{8} Q_{m=1}(T) - \left(\frac{3}{8} Q_{m=2}(T) + \frac{5}{8} Q_{m=3}(T) \right) = 0.003$$
$$VI_{j=2}^T = \frac{5}{8} Q_{m=3}(T) - \left(\frac{2}{8} Q_{m=4}(T) + \frac{3}{8} Q_{m=5}(T) \right) = 0.170$$

The split feature importance for an ensemble of decision trees (i.e., Random Forest) is given by VI_j^{RF} [20, 1].

$$VI_j^{\text{RF}} = \frac{1}{B} \sum_{b=1} \sum_{m, j \in \theta_m^*} \Delta_b \left(\theta_m^* \right) \tag{13}$$

where B is the base learner.

The feature importance is on a scale between 0 and 1, a score closer to 1 indicates the feature is highly influential on the model predictions. Whereas a score close to 0 indicates very little influence on model predictions. Figure 15 shows the feature importance for all variables in the model. The feature OBTOTV18, total office-based visits throughout 2018, is identified as the most influential feature with a score of 0.10.

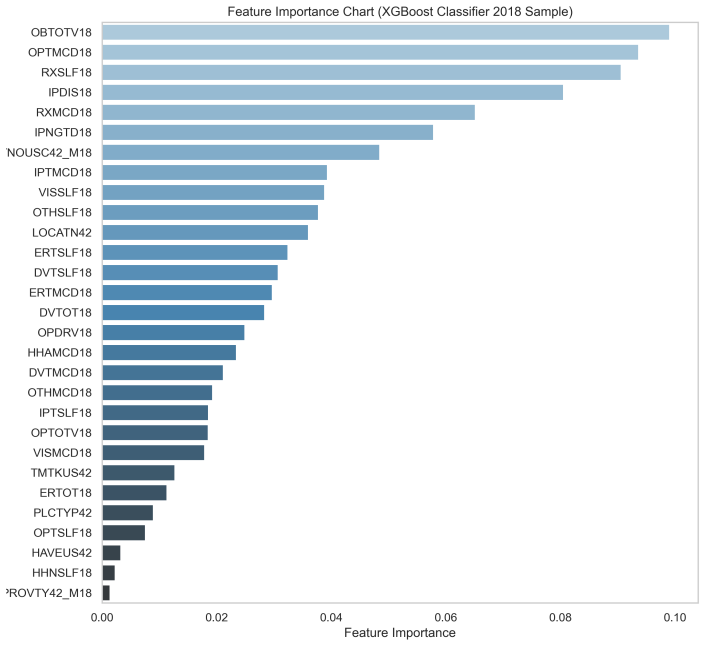


Figure 15: Feature importance from a gradient boosted random forest classifier

Additionally, the following features contribute heavily to the identification of anomalies or normal instances in the iforest model.

- OPTMCD, sum of facility and SBD expenses
- RXSLF, total amount paid for prescription medicines
- IPDIS18, number of hospital discharges during 2018
- RXMCD18, total amount paid for prescription medicines

Alternatively, the features below show little to no contribution to the model. One hypothesis could be that almost all families in the sample contained the same values for these features. Therefore, the model is unable to differentiate anomalies and normal instance from partitioning this feature.

- PROVTY42_M18, provider type
- HHNSLF18, paid independent providers home health care
- HAVEUS42, does a person have usual source care provider

have little to no effect on the model predictions. The results from figure 11 indicate which features are driving the isolation of instances in the iforest model.

2.4.2 Partial Dependence Plots (PDP)

Feature Importance is a helpful metric to identify which features contributed the most to the predictions. However, feature importance does not identify what values of the dependent variables influence the outcome of the prediction. An additional method for model interpretation that provides a more granular level of analysis than feature importance is the partial dependence plot (PDP) [5, 6]. The PDP depicts the marginal effect of a feature on the target prediction and conveys the relationship between a feature and target. Let there be k separate models $f_k(x)$ for k-class classification, each containing a sum of trees.

$$f_k(x) = \sum_{m=1}^{M} T_{km}(x) \tag{14}$$

Then each model k is related to the respective probabilities through the following equation

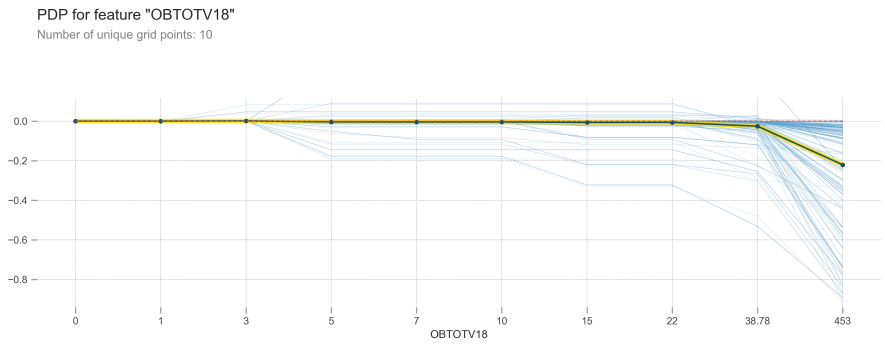
$$f_k(X) = \log p_k(X) - \frac{1}{K} \sum_{l=1}^{K} \log p_l(X) \tag{15}$$

The PDP for every $f_k(X)$ reveals how a class depends on a prospective feature via the probabilities.

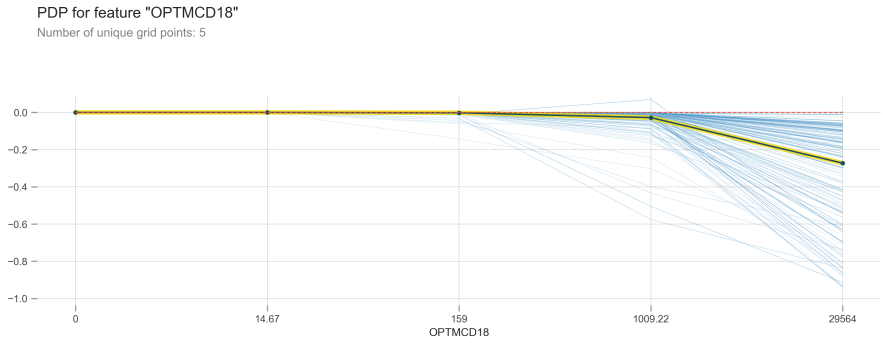
Below in figures 16 and 17 are the PDP for the top 4 features. These plots were generated using the Python package PDPbox [14]. The vertical axis indicates either an increase or decrease in the probability of being an anomaly. The horizontal axis indicates unique grid points for a specific feature. The yellow line indicates the average effect, while blue lines indicate individual effects on the feature. Note, the horizontal axis indicates the critical points of the line. Only when

there is a significant change in the average does the average react.

For instance, figure 16a shows the probability of being labeled an anomaly for OBTOTV18 as increasing between the grid points 22 and 453. Said differently, the probability of being considered normal decreases between the grid points 22 and 453. Additionally, figure 16b indicates the probability of being labeled an anomaly increasing between the grid points 1009 and 29564.

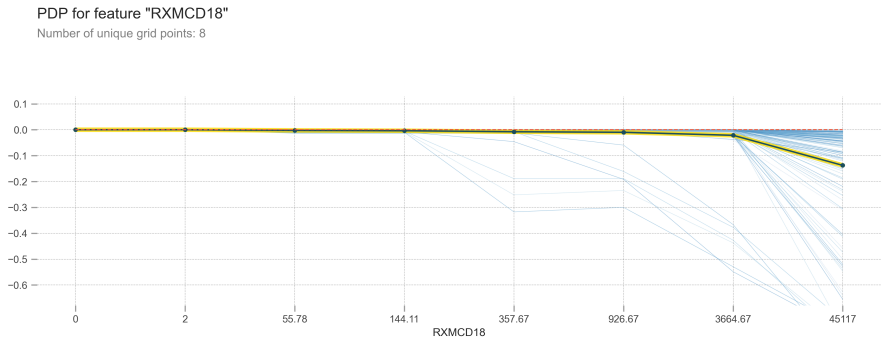


(a) OBTOTV18, total office-based visits throughout 2018

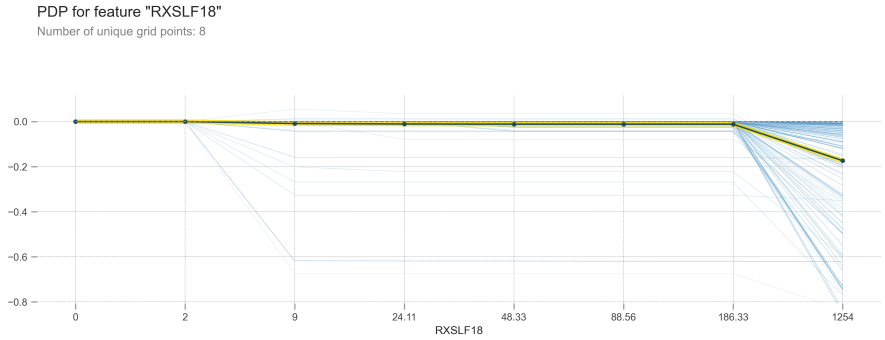


(b) OPTMCD18, sum of facility and SBD expenses

Figure 16: Partial Dependence Plots for OBTOTV18 and OPTMCD18



(a) RXMCD18, total amount paid by Medicaid for prescription drugs



(b) RXSLF18, total amount paid out-of-pocket for perscription drugs

Figure 17: Partial Dependence Plots for RXMCD18 and RXSLF18

Figure 17b is the PDP plot of RXMCD18, the total amount paid by Medicaid for prescription drugs. The probability of an individual being labeled as normal decreases by a probability of roughly 0.1. This means that a family who spends more than roughly $3000 is considered more anomalous with respect to the population. Figure 17a shows the PDP plot for RXSLF18, total amount paid out-of-pocket for prescription drugs. The probability of an instance being labeled as normal decreases by about 0.2 between the values of 186 and 1254. Since we are classifying whether or not an observation is anomalous, it follows that the average is constant (i.e. a majority of observations are considered normal) up until a critical point where the probability shifts drastically.

2.5 Conclusion:

2.5.1 Discussion

This method successfully identified anomalies across an array of features related to health care expenditures, utilization, and access to care for a sample of Medicaid beneficiaries. Furthermore, modeling the anomalies provides additional information regarding which features are driving the iforest. The features OBTOTV18, OPTMCD18, RXMCD18, IPDIS18, and RXSLF18 are identified as significant drivers in the model using the model interpretation methods of feature importance and PDP.

The limitations of this study are as follows. First, the effect of Medicaid on the State level. Medicaid functions at the state level, meaning States have different cost-sharing standards, services, and eligibility requirements. Due to no variable indicating the State in which an individual resides the model is comparing families across different states. Making the study not one-to-one.

Second, most features pertaining to access to health care are label encoded. Including these features were thought to potentially find relationships among families with poor access to care, and high medical expenses. However, due to the generality of most features within the category, many families had the same answers with few exceptions.

2.5.2 Future Research

This new methodology used the anomaly detection algorithm iforest to isolate data based on a portfolio of features related to healthcare expenditures, utilization, and access to care. The methodology aims to demonstrate a new way of analyzing Medicaid beneficiary data to understand populations identified as anomalous and why they incur abnormal costs, utilization, or poor access to care.

The scope of this study was minimal due to the small n and lack of additional information regarding beneficiaries. Given more information concerning a beneficiaries location, such as State rather than region, would allow us to control for state-level economic policy, family dynamics (i.e. divorced parents and students) and geographic location (i.e. zip codes). Access to the State is an essential factor to consider since some beneficiaries may not have the same benefits as others. Furthermore, having access to the city a beneficiary resides in could provide additional detail regarding the cost of living and proximity to health care and non-healthcare resources such as education.

There are many avenues in which the methodology can further future research. One avenue would be to cluster the collection of anomalies for a given state to identify any similarities among the anomalies. This method could help not only identify the features driving anomalous instances and instead identify who the anomalous instances are. Characterizing who the anomalous population is can provide more information to policy-makers of where and how to reallocate resources to reduce Medicaid spending, and maintain or increase the quality of care.

References

[1] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[2] Markus M Breunig et al. "LOF: identifying density-based local outliers". In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pp. 93–104.

[3] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey". In: *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–58.

[4] Reinhard Diestel. "The basics". In: *Graph Theory*. Springer, 2017, pp. 1–34.

[5] Alex Goldstein et al. "Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation". In: *journal of Computational and Graphical Statistics* 24.1 (2015), pp. 44–65.

[6] Trevor Hastie et al. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.

[7] Agency for Healthcare Research and Quality. *MEPS HC-209: 2018 Full Year Consolidated Data File*. URL: https://meps.ahrq.gov/mepsweb/data_stats/download_data_files_detail.jsp?cboPufNumber=HC-209.

[8] Robert Kelley. "Where can $700 billion in waste be cut annually from the US healthcare system". In: *Ann Arbor, MI: Thomson Reuters* 24 (2009), pp. 1–30.

[9] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation forest". In: *2008 eighth ieee international conference on data mining*. IEEE. 2008, pp. 413–422.

[10] Willard G Manning et al. “Health insurance and the demand for medical care: evidence from a randomized experiment”. In: *The American economic review* (1987), pp. 251–277.

[11] Guansong Pang et al. “Deep learning for anomaly detection: A review”. In: *ACM Computing Surveys (CSUR)* 54.2 (2021), pp. 1–38.

[12] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[13] Bruno R.. Preiss. *Data Structure and Algorithms: With Object-oriented Design Patterns in Java*. John Wiley & Sons, 1999.

[14] Jiangchun L SauceCat. *PDPbox*. 2020.

[15] Aris Spanos. “The discovery of argon: A case for learning from data?” In: *Philosophy of Science* 77.3 (2010), pp. 359–380.

[16] Srikanth Thudumu et al. “A comprehensive survey of anomaly detection techniques for high dimensional big data”. In: *Journal of Big Data* 7.1 (2020), pp. 1–30.

[17] Peter Travaille et al. “Electronic fraud detection in the US medicaid healthcare program: lessons learned from other industries”. In: (2011).

[18] Maximilian E Tschuchnig and Michael Gadermayr. “Anomaly Detection in Medical Imaging–A Mini Review”. In: *arXiv preprint arXiv:2108.11986* (2021).

[19] John W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[20] Zhengze Zhou and Giles Hooker. “Unbiased measurement of feature importance in tree-based methods”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 15.2 (2021), pp. 1–21.

A Variable Codes

A.1 Survey Administration Variables

Survey Administration Variables		
Code	Constructed	Description
DUID		Panel # + Encrypted DU Identifier
PID		Person number
PANEL		Panel Number (22 or 23)
FAMWT18F		MEPS families that consisted of more than one CPS-like family back together and assigning the MEPS family level weight based on the CPS family weight of the MEPS family reference person
FAMIDYR		Annual family identifier
DUIDFAMY	✓	Concatenated DUID, FAMIDY to create family-level file
RESPXY		1st Respondant indicator for RXY
REGIONXY		Census Region for RXY
MCDHMOXY		Covered by medicaid/SCHIP HMO anytime during XY

Table 11: A comprehensive list of administrative features. Engineered features are identified with a ✓ in the *constructed column*.

A.2 Health Expenditure Variables

Health Expenditure Variables		
Code	Constructed	Description
OBV***18		Total office based visits (physician + non-physician + unkown)
OPT***18		Sum of facility and SBD Expenses
ERT***18		Emergency room visits, sum of facility and SBD Expenses
IPT***18		Inpatient hospital stays (including zero night stays) sum of facility and SBD Expenses
RX***18		Total amount paid for prescription medicines
DVT***18		Total amount paid for dental visits
HHA***18		Agency sponsored home health care
HHN***18		Paid independent providers home health care
VIS***18		Vision aids home health care
OTH***18		Other medical supplies and equipement home health care
Health Expenditure Source of Payment Code Replacements (***)		
EXP		sum of all sources
SLF		Out of pocket
MCD		Medicaid
OTH		Other federal, state and local, private, public or unclassified sources

Table 12: A comprehensive list of all features pertaining to healthcare expenditures

A.3 Health Care Utilization Variables

Health Care Utilization		
Code	Constructed	Description
OBTOTV18		Number of office based provider visits during 2018
OPTOTV18		Number of outpatient department provider visits during 2018
OPDRV18		Number of Outpatient department physician visits during 2018
ERTOTV18		Number of Emergency Room (ER) visits during 2018
DVTOT18		Number of dental care visits during 2018
IPDIS18		Number of hospital discharges during 2018
IPNGTC18		Number of nights in hospital for discharges during 2018

Table 13: A comprehensive list of all features pertaining to healthcare utilization

A.4 Access to Care Variables

Access to Care		
Code	Constructed	Description
TYPEPE42		Usual source care provider
HAVEUS42		Does person have usual source care provider for rounds 4 and 2?
YNOUSC42_M18		Main Reason person does not have usual source care provider for rounds 4 and 2
PROVTY42_M18		Provider type rounds 4 and 2
PLCTYP42		Usual source care provider type of place
TMTKYS42		How long does it take to get to usual source care provider?
LOCATN42		Usual source care provider location for round 4 and 2

Table 14: A comprehensive list of all features pertaining to access to healthcare

B Code

B.1 Cleaning Data

```
1
2 def clean(df = None):
3
4     # Filter down to the reference person ONLY summing
    characteristics
5
6     df = df[(df['FAMWT18F'] > 0)] # Family weight > 0
7     df['DUIDFAMY'] = (df['DUID'].astype('str')) + df['FAMIDYR'] #
      concatenate to create subset of families
8     df = df[df['ACCELI42'] != -1] # Indicates whether or not
    someone is eligible to answer Access to care variables
9
10    # Data frame to merge
11    HACC = df[['DUID',     # Dwelling unit ID
12    'DUIDFAMY', # Dwellinging Family
13    'FAMIDYR',  # Family ID letter
14    'FAMRFPYR', # reference person indicator
15    'INSCOP18', # Whether an individualinscope
16    'PANEL',    # Panel Number
17    'REGION31', # region panel 31
18    'REGION42', # region panel 42
19    'REGION53', # region panel 53
20    'POVLEV18', # POVLEV18 = CPS Family IncoPoverty Line
21    'MCDHMO31', # MDC Indicator
22    'MCDHMO42', # MDC Indicator
23    'MCDHMO18', # MCD Indicator
24    'FAMSZEYR', # Family size
25    'HAVEUS42',
```

```
    'YNOUSC42_M18',
    'PROVTY42_M18',
    'PLCTYP42',
    'TMTKUS42',
    'TYPEPE42',
    'LOCATN42'
                        ]]


    # Replacing codes with another value (2 = No) or Removing
    them from the sample (-7)
    # (-8) is Do not Know
    HACC = HACC.replace({'PROBPY42' : -8
                         }, 2)

    HACC = HACC[HACC.isin([-7]) == False] # drop any rows
    containing -7 (refused)

    # for variables where it is an individual, sum all the Yes's
    and No's then subtract family size
    access_to_care = HACC.\
    groupby('DUIDFAMY').\
    agg({
        'HAVEUS42' : 'max',
        'YNOUSC42_M18' : 'max',
        'PROVTY42_M18' : 'max',
        'PLCTYP42' : 'max',
        'TMTKUS42' : 'max',
        'LOCATN42' : 'max',
        'REGION31' : 'max',
        'REGION42' : 'max',
        'REGION53' : 'max'
        }).reset_index()
```

```
56
57    # access_to_care[categorical] = access_to_care[categorical].\
58    astype('category')
59
60    HACC = HACC.drop(columns = [
61        'HAVEUS42',
62        'YNOUSC42_M18',
63        'PROVTY42_M18',
64        'PLCTYP42',
65        'TMTKUS42',
66        'LOCATN42',
67        'REGION31',
68        'REGION42',
69        'REGION53'
70                                ])
71
72    # Use the reference person FAMRFPYR to characterize the
      family unit
73    # For numeric variables we are taking the sum of all family
      members
74
75    expenditures = df.groupby('DUIDFAMY')\
76    .agg({'RXMCD18' : 'sum',
77    'RXSLF18' : 'sum',
78    'OPTMCD18' : 'sum',
79    'OPTSLF18' : 'sum',
80    'ERTMCD18' : 'sum',
81    'ERTSLF18' : 'sum',
82    'IPTMCD18' : 'sum',
83    'IPTSLF18' : 'sum',
84    'DVTMCD18' : 'sum',
85    'DVTSLF18' : 'sum',
86    'HHAMCD18' : 'sum',
```

```
    'HHNSLF18' : 'sum',
    'VISMCD18' : 'sum',
    'VISSLF18' : 'sum',
    'OTHMCD18' : 'sum',
    'OTHSLF18' : 'sum',
    'OBTOTV18' : 'sum', # TEST
    'OPTOTV18' : 'sum',
    'OPDRV18' : 'sum',
    'ERTOT18' : 'sum',
    'DVTOT18' : 'sum',
    'IPDIS18' : 'sum',
    'IPNGTD18' : 'sum'
    }).reset_index()

    join = pd.merge(pd.merge(HACC,
    expenditures,
    how = "left",
    on = 'DUIDFAMY',
    suffixes = ("_ECON", "_EXP")
            ),
    access_to_care,
    on = 'DUIDFAMY'
            )

    # Filter data set down to reference person
    join = join[(join['FAMRFPYR'] == 1)]
    join = join[((join['MCDHMO31'] == 1) |
    (join['MCDHMO31'] == 2)) &
    ((join['MCDHMO42'] == 1) |
    (join['MCDHMO42'] == 2)) &
    ((join['MCDHMO18'] == 1) |
    (join['MCDHMO18'] == 2))
    ]
```

```
120
121     holdout = join[['DUIDFAMY',
122     'REGION31',
123     'REGION42',
124     'REGION53']
125     ]\
126     .reset_index(drop = True)
127
128     join = join.drop(columns=['MCDHMO31',
129     'MCDHMO42',
130     'MCDHMO18',
131     'DUID',
132     'INSCOP18',
133     'FAMIDYR',
134     'PANEL',
135     'FAMRFPYR',
136     'FAMSZEYR',
137     'REGION31',
138     'REGION42',
139     'REGION53'
140     ]
141     ).reset_index(drop=True)
142
143     expenditures = join[['RXMCD18',
144     'RXSLF18',
145     'OPTMCD18',
146     'OPTSLF18',
147     'ERTMCD18',
148     'ERTSLF18',
149     'IPTMCD18',
150     'IPTSLF18',
151     'DVTMCD18',
152     'DVTSLF18',
```

```
    'HHNSLF18',
    'HHAMCD18',
    'VISMCD18',
    'VISSLF18',
    'OTHMCD18',
    'OTHSLF18',
    'OBTOTV18',
    'OPTOTV18',
    'OPDRV18',
    'ERTOT18',
    'DVTOT18',
    'IPDIS18',
    'IPNGTD18'
                ]]
    access_to_care = join[['HAVEUS42',
    'YNOUSC42_M18',
    'PROVTY42_M18',
    'PLCTYP42',
    'TMTKUS42',
    'LOCATN42'
    ]]

    return expenditures, access_to_care, join, holdout
```

Listing 1: Cleaning MEPS H-209 data

B.2 Isolation Forest

```
1
2  def isolation_forest(data = None,
3      estimators = 150,
4      contaminate = 'auto'):
5
6      iforest = IsolationForest(n_estimators = 150,
7              contamination = 0.15,
8              bootstrap = False,
9              n_jobs = None,
10             random_state = None,
11             verbose = 1,
12             warm_start = False).fit(data)
13
14     scores = iforest.decision_function(data) # anomaly scores
15     labels = iforest.fit_predict(data) # anomaly labels
16
17     return scores, labels
```

Listing 2: Generating anomaly scores using iforest

B.3 Training, Test Data

```
1
2  def training_test(data = None, learning_type = None):
3
4      if learning_type == 'classification':
5
6          data = data.replace({'AMLY_LABEL' : -1}, 0)
7          X = data.drop(columns=['AMLY_LABEL'])
8          Y = data['AMLY_LABEL']
9
10     elif learning_type == 'regression':
11
12         X = data.drop(columns=['AMLY_SCORE'])
13         Y = data['AMLY_LABEL']
14
15     X_train,X_test, Y_train, Y_test = train_test_split(X,
16     Y,
17     shuffle = True,
18          = 0.40)
19
20     return X_train, X_test, Y_train, Y_test
```

Listing 3: Creating training and test data sets

B.4 Grid Search Cross Validation

```
1
2  class_models_new = {'gradientboosted' :
3  {'model' : GradientBoostingClassifier(),
4      'parameters' : [
5                          {
6      "loss" : ['deviance'],
7      "n_estimators" : [50,
8          100,
9          150,
10         200,
11         250,
12         300],
13     "validation_fraction" : [0.1,
14         0.2,
15         0.3,
16         0.4,
17         0.5],
18     "learning_rate" : [0.001,
19         0.01,
20         0.1,
21         0.2,
22         0.3,
23         0.4],
24     "tol" : [0.001,
25         0.01,
26         0.02,
27         0.03]}
28             ]
29             },
30     'RandomForest' :
31     {'model' : RandomForestClassifier(),
```

```
        'parameters' : [
        {
         "n_estimators" : [100,
         150,
         200,
         250,
         300],
         "criterion" : ['gini',
                'entropy']}
                    ]
    },
    'LogisticRgr' :
    {'model' : LogisticRegression(),
        'parameters' : [
        {
        "penalty" : ['l2',
                     'l1'],
        "C" : [0.25,
            0.5,
            0.75,
            1]}
        ]
        }
        }

def gridsearchCV(model_parameters=None,
                 X_train=None,
                 Y_train=None,
                 scoring=None,
                 k=None):

    best_models = {}

```

```
    keys = list(model_parameters.keys())

    for model in model_parameters.keys():

        clf = GridSearchCV(model_parameters[model]['model'],
            model_parameters[model]['parameters'],
            scoring = scoring,
            return_train_score = True,
            cv = k)

        clf.fit(X_train, Y_train)
        model_obj = clf.best_estimator_
        detailed_output = pd.DataFrame(clf.cv_results_)
        best_models[model] = {} # model_obj
        best_models[model]['model'] = model_obj

    return best_models, detailed_output
```

Listing 4: Grid-search cross validation

B.5 Model Evaluation

```
1
2
3  def classification_evaluation(class_models = None,
4                                X_train = None,
5                                y_train_class = None,
6                                X_test = None,
7                                y_test_class = Y_test):
8
9      for model_name in class_models.keys():
10
11         fitted_model = class_models[model_name]['model'].fit(
       X_train, Y_train)
12         y_train_pred = fitted_model.predict(X_train.values)
13
14         if model_name == 'ridge':
15             y_test_pred = fitted_model.predict(X_train.values)
16         else:
17             y_test_prob = fitted_model.predict_proba(X_test.
       values)[:,1]
18             y_test_pred = np.where(y_test_prob > 0.5, 1, 0)
19
20         class_models[model_name]['fitted'] = fitted_model
21         class_models[model_name]['probs'] = y_test_prob
22         class_models[model_name]['preds'] = y_test_pred
23         class_models[model_name]['Accuracy_train'] = metrics.\
24         accuracy_score(y_train_class, y_train_pred)
25         class_models[model_name]['Accuracy_test'] = metrics.\
26         accuracy_score(y_test_class, y_test_pred)
27         class_models[model_name]['Recall_train'] = metrics.\
28         recall_score(y_train_class, y_train_pred)
29         class_models[model_name]['Recall_test'] = metrics.\
```

```
        recall_score(y_test_class, y_test_pred)

        if model_name != 'ridge':
            class_models[model_name]['ROC_AUC_test'] = metrics.\
            roc_auc_score(y_test_class, y_test_prob)
        else:
            class_models[model_name]['ROC_AUC_test'] = 0

        class_models[model_name]['F1_test'] = metrics.f1_score(
    y_test_class,
        y_test_pred)
        class_models[model_name]['MCC_test'] = metrics.\
        matthews_corrcoef(y_test_class, y_test_pred)
        class_metrics = pd.DataFrame.from_dict(class_models,\
                    'index')[['Accuracy_train',
                    'Accuracy_test',
                    'Recall_train',
                    'Recall_test',
                    'ROC_AUC_test',
                    'F1_test',
                    'MCC_test']]

        class_metrics = class_metrics.\
        sort_values(by = 'ROC_AUC_test', ascending = False).\
        style.background_gradient(cmap = 'plasma', low=0.3, high
    = 1,
        subset = ['Accuracy_train', 'Accuracy_test']).\
        background_gradient(cmap = 'viridis',
        low = 1,
        high = 0.3,
        subset = ['Recall_train', 'Recall_test',
        'ROC_AUC_test', 'F1_test', 'MCC_test'])

```

```
61        plt.tick_params(axis = 'both',
62        which = 'major',
63        labelsize = 12
64        )
65        fpr, tpr, _ = metrics.\
66        roc_curve(y_test_class, class_models[model_name]['probs'
    ])
67        plt.plot(fpr, tpr,
68        label = 'ROC Curve (area = %0.2f)' % class_models[
    model_name]['ROC_AUC_test']
69        )
70        plt.plot([0,1], [0,1], 'k-')
71        plt.ylabel('True Positive Rate', fontsize = 14)
72        plt.xlabel('False Positive Rate', fontsize = 14)
73        plt.legend(loc = "lower right")
74
75        cf_matrix = confusion_matrix(y_test_class, y_test_pred)
76
77     return class_metrics, class_models
```

Listing 5: Evaluating gradient boosted random forest classification model

VITA

Author	Nathaniel E. Islip
Place of Birth:	Saint Paul, Minnesota
Degrees Awarded:	Bachelor of Science, 2020, Eastern Washington University
Honors and Awards:	Graduate Assistantship, Mathematics Department, 2020, 2022, Eastern Washington University
	EWU SIAM Student Chapter Award, 2021-2022, SIAM
Professional experience:	Risk Modeling Analyst, Enact Mortgage Insurance, 2022
	Internship, Enact Mortgage Insurance, 2021
	Internship, City of Kirkland, 2020
	Graduate Service Assistant, Eastern Washington University, 2020
	Economic Research Assistant, Eastern Washington University, 2019