NETWORK VERIFICATION PROJECT

Network Verification with Outlier Analyzer

Calculating Outliers using Statistical, ML-based Techniques & Signature-based Approach.

WINGS LAB
STONY BROOK UNIVERSITY
Department of Computer Science

Contents

Chapter 1

Introduction

1.1 Problem Statement

Outliers are considered as deviation among set of values of the distributions from their most popular entries. We explore outliers in the context of enterprise networks and IoT ecosystems. In this research we currently explore about the enterprise networks. We explore two different types of outliers (*i*) control plane outliers and (*ii*) data plane outliers.

i. *Control Plane Outliers*: With respect to enterprise network, the network hosts or devices that has their configurations deviant from the majority of their devices in the is considered as *control plane outliers*.

2. *Data Plane Outliers*: While the *data plane outliers* are considered as deviation in the traffic patterns of the among the hosts and network nodes compared to the normal or predicted traffic characteristics of these devices.

Note: We consider derived outliers as a bug with some level of *confidence* and specific level of *severity*. The data on which the outliers are computed (i.e., configurations, states, and data traffic patterns) from the enterprise network devices and hosts is categorical[1].

[1]In statistics, a categorical variable can take on one of a limited, and usually fixed number of possible values that are not just limited to numerical values [1].

Chapter 2

Setup Details of Batfish & Pybatfish

2.1 Batfish

Batfish is a network validation tool that provides correctness guarantees for security, reliability, and compliance by analyzing the configuration of network devices. It builds complete models of network behavior from device configurations and finds violations of network policies (built-in, user-defined, and best-practices).

A primary use case for Batfish is to validate configuration changes before deployment (though it can be used to validate deployed configurations as well). Predeployment validation is a critical gap in existing network automation workflows. By Batfish in automation workflows, network engineers can close this gap and ensure that only correct changes are deployed.

Batfish does NOT require direct access to network devices. The core analysis requires only the configuration of network devices. This analysis may be enhanced using additional information from the network such as:

- BGP routes received from external peers
- Topology information represented by LLDP/CDP

Note**: Above mentioned batfish and pybatfish descriptions are copied from Batfish & Pybatfish github pages.

Install Batfish:

`https://github.com/batfish/batfish`

2.2 Pybatfish

Pybatfish is a Python client for Batfish.

Install Pybatfish:

`https://github.com/batfish/pybatfish`

2.3 Understanding Batfish (video repository)

`https://www.youtube.com/channel/UCA-OUW_3IOt9U_s60KvmJYA/videos`

2.4 Quick Install Steps

1. Set up jdk manually by downloading it.

   `https://www.oracle.com/technetwork/java/javase/downloads/index.html`

   Note: I think batfish requires version 8 of java, please check. If so mention link to specific version.
2. To install maven directly use: brew install maven
3. Go to the batfish folder and firstly give the shell access to batfish functions by:

   `source tools/batfish_functions.sh`

   » Once, the shell is given access to batfish functions, we want to run our own python client run the following command:

   `allinone -runclient false`

   » Now, because we want to load questions as well we will be running:

   `allinone -runclient false -coordinatorargs "-templatedirs questions"`

Chapter 3

Outlier Analysis Tool

3.1 Outlier Analyzer

The outlier analyzer tool that we built is python-based which supports following three types of outlier detection techiques for control plane analysis.

3.2 Techniques

Broadly, we performed outliers calculation study with three different types of outliers calculation:

- Statistical approach
- ML-based technique
- Signature-based technique

Future Task: The data plane outliers calculation is left as part of next task.

3.3 Setup, Run & Install Steps

3.3.1 environment setup

Include this command in your .bashrc file:

```
source tools/batfish_functions.sh
```

Or run it manually each time.

3.4 Run Outlier Analyzer Tool

3.4.1 Run using flat-file dataset

The outlier analyzer tool could be run independently using the flat-file dataset (see below steps).

1. **Install outliers analyzer (git)**:

```
git clone git@github.com:vasu018/outlier-analyzers.git
```

2. **Setup Python Version 3**:

```
https://help.dreamhost.com/hc/en-us/articles/115000695551-Installing-and-using-virtu

virtualenv . -p /Library/Frameworks/Python.framework/Versions/3.4/bin/python3
source  bin/activate "Or"
source .env3/bin/activate
```

3. Run Outlier Analyzer Code:

```
python3 outlierAnalyzer.py -i datasets/flat-sample/outputAnonymizedFile.json > outpu
```

3.4.2 Run using batfish

Note: Steps may be bit incomplete.

If the outlier analyzer tool need to be run with the batfish then install the batfish as discussed in Section 2.

1. **Install outliers analyzer (git)**:

```
git clone git@github.com:vasu018/outlier-analyzers.git
```

2. **Setup Python Version 3**:

```
https://help.dreamhost.com/hc/en-us/articles/115000695551-Installing-and-using-virtu

virtualenv . -p /Library/Frameworks/Python.framework/Versions/3.4/bin/python3
source  bin/activate "Or"
source .env3/bin/activate
```

3. **Install batfish/Pybatfish**

   Note: If you wanted to run outliers directly on the flat input data set.

   Install Batfish:

   ```
   https://github.com/batfish/batfish
   ```

   Install Pybatfish (Pybatfish is a Python client for Batfish)

   ```
   https://github.com/batfish/pybatfish
   ```

4. Run Outlier Analyzer Code:

```
python3 outlierAnalyzer.py -i datasets/flat-sample/outputAnonymizedFile.json > output.txt
```

Chapter 4

ML-based Techniques

4.1 K-Means Clustering

K-means is the technique that tries to minimize the root mean square error once the partition into various clusters is done. The value of K has to be specified by us in advance before running K-means. K-Means is sometimes vulnerable to outliers. This is because large values drastically shift the mean.

Application of K-means Clustering on Network data: Network data has the following format:

As this is categorical data, we try to encode this data using MultiLabelBinarizer.

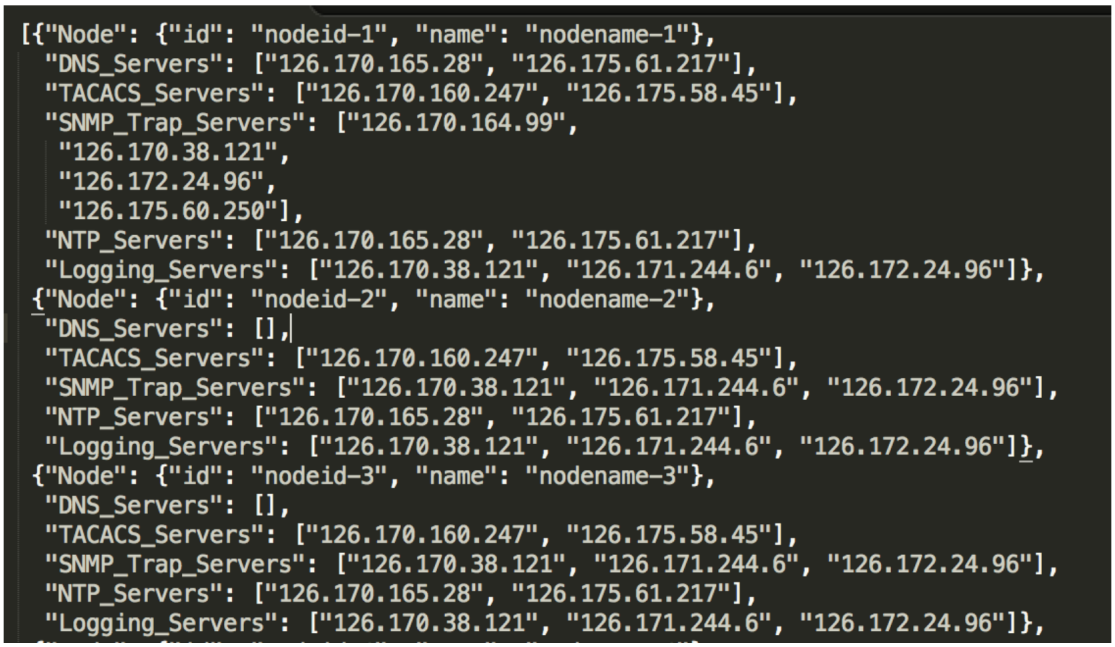

Figure 4.1: Network data

Each of the features like DNS-Servers, TACACS-Servers,SNMP-Trap-Servers, NTP-Servers, Logging-Servers are been encoded. Consider an example of how MultiLabelBinarizer works:

```
multiclass_feature_Xi = [("1.1.1.1", "2.2.2.2"),
                         ("1.1.1.1", "3.3.3.3"),
                         ("3.3.3.3", ""),
                         ("1.1.1.1", ""),
                         ("1.1.1.1", "2.2.2.2"),
                         ("1.1.1.1", "2.2.2.2"),
                         ("1.1.1.1", "2.2.2.2"),
                         ("1.1.1.1", "2.2.2.2"),
                         ("1.1.1.1", "3.3.3.3"),
                         ("1.1.1.1", "3.3.3.3"),
                         ("1.1.1.1", ""),
                         ("1.1.1.1", "3.3.3.3"),
                         (),
                         ("", ""),
                         ("1.1.1.1", ""),
                         ("2.2.2.2", "")]
```

Figure 4.2: Multi class feature

After encoding this feature, we get the following output:

```
[[0 1 1 0]
 [0 1 0 1]
 [1 0 0 1]
 [1 1 0 0]
 [0 1 1 0]
 [0 1 1 0]
 [0 1 1 0]
 [0 1 1 0]
 [0 1 0 1]
 [0 1 0 1]
 [1 1 0 0]
 [0 1 0 1]
 [0 0 0 0]
 [1 0 0 0]
 [1 1 0 0]
 [1 0 1 0]]
```

Figure 4.3: Encoded data

Four columns are present in the encoded list as there are 4 distinct values possible for the variable. Presence of 1 indicates that a particular value is present for the device. If 1 is not present then we conclude that the value is absent for the device. Thus, for every device we calculate the density list by adding all the values a particular categorical variable appears. And then, if 1 is present we add this value or ignore the value if 0 is present. In this example the density value for the 0th row will be 18(12+6) as the first column has 1 appearing 12 time while second column has 1 appearing 6 times. In this way for all the features of a device we calculate density list. Then, we pass these density lists as the distinct features

for the K-Means algorithm. We can specify the number of clusters. Then, we apply the inter-cluster and intra-cluster criteria for outlier detection once the clustering is done.

4.2 Inter-cluster outlier detection criteria

This technique tries to find the distance between the cluster centers and compares this distance to the distance of each point from its corresponding cluster center. If the ratio of the minimum distance between cluster centers and the distance of a point from it's respective cluster center falls below a certain threshold, then the point is termed as an outlier. This threshold has to be adjusted in order to get an appropriate number of outliers. We had to alter the value of threshold while experimenting with number of clusters, this is because more the number of clusters less will be the distance of the points from their respective centroids. This criterion was applied on the network data present in datasets/flat-sample/outputAnonymizedFile.json. The outliers detected by this criterion (for number of clusters as 2 and weight as 0.1) were:

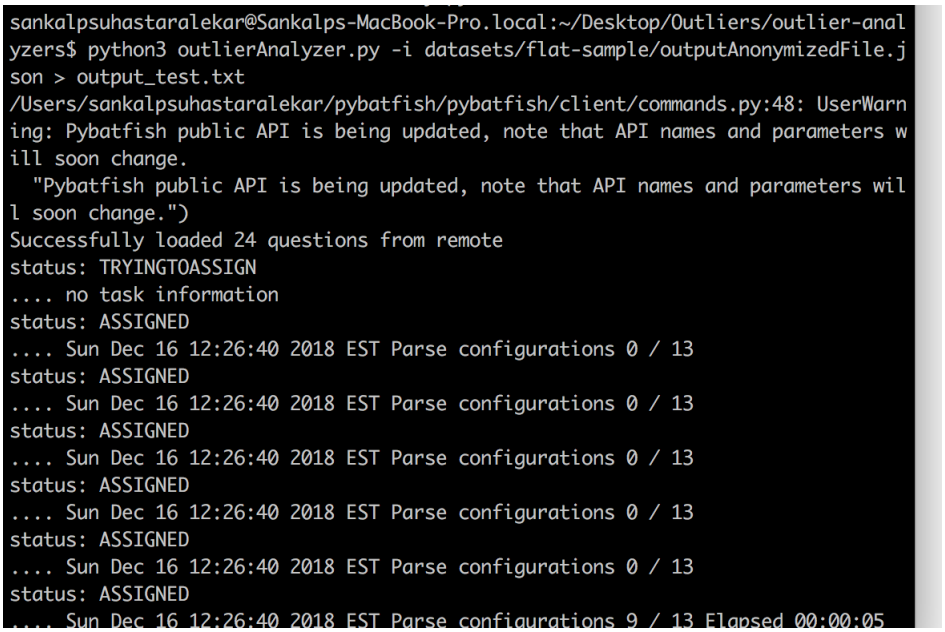

Figure 4.4: The output is directed to output-test file

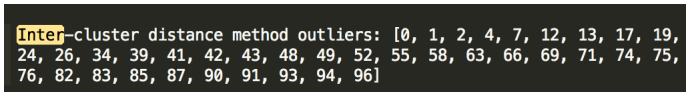

Figure 4.5: Outliers given by inter-cluster criteria

4.3 Intra-cluster outlier detection criterion

This technique compares the distance of a point from it's cluster center and the distance of the point that is closest to the corresponding cluster center. If the ratio of the distance of a point from it's cluster center and the point that is closest to the corresponding cluster center exceeds a certain threshold, then the point is termed as an outlier. This criterion was applied on the network data present in datasets/flat-sample/outputAnonymizedFile.json. The outliers detected by this criterion (for number of clusters as 2 and weight as 0.1) were:

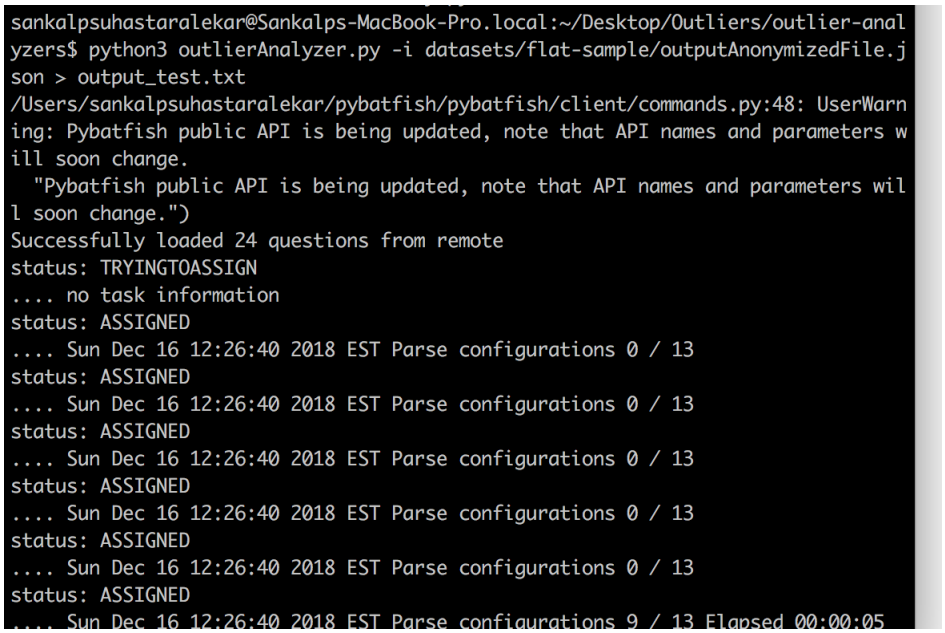

Figure 4.6: The output is directed to output-test file

Figure 4.7: Outliers given by intra-cluster criteria

4.4 Gaussian Mixture Model

The clustering techniques implemented above are hard-clustering techniques that classify samples in a particular class. Thus, we used Gaussian Mixture Model which is a soft clustering technique that assigns probability for each sample belonging to a particular cluster. This, technique gave us the likelihood of a sample belonging to a particular class.

Application of G.M.M on network data present in:

`datasets/flat-sample/outputAnonymizedFile.json`

As G.M.M gives the likelihood of a device belonging to a particular Gaussian, we passed the encoded data of features to G.M.M. We concatenated the multiple features for all the network devices and then fed this concatenated list to G.M.M. Gaussian will give us the likelihood for a particular device that it is a part of the Gaussian. The number of components was set to 2. The negative likelihoods given for all the devices is shown below:

```
sankalpsuhastaralekar@Sankalps-MacBook-Pro.local:~/Desktop/Outliers/outlier-anal
yzers$ python3 outlierAnalyzer.py -i datasets/flat-sample/outputAnonymizedFile.j
son > output_test.txt
/Users/sankalpsuhastaralekar/pybatfish/pybatfish/client/commands.py:48: UserWarn
ing: Pybatfish public API is being updated, note that API names and parameters w
ill soon change.
  "Pybatfish public API is being updated, note that API names and parameters wil
l soon change.")
Successfully loaded 24 questions from remote
status: TRYINGTOASSIGN
.... no task information
status: ASSIGNED
.... Sun Dec 16 12:26:40 2018 EST Parse configurations 0 / 13
status: ASSIGNED
.... Sun Dec 16 12:26:40 2018 EST Parse configurations 0 / 13
status: ASSIGNED
.... Sun Dec 16 12:26:40 2018 EST Parse configurations 0 / 13
status: ASSIGNED
.... Sun Dec 16 12:26:40 2018 EST Parse configurations 0 / 13
status: ASSIGNED
.... Sun Dec 16 12:26:40 2018 EST Parse configurations 0 / 13
status: ASSIGNED
.... Sun Dec 16 12:26:40 2018 EST Parse configurations 9 / 13 Elapsed 00:00:05
```

Figure 4.8: The output is directed to output-test file

4.5 Logistic Regression

This is a supervised classification technique used to classify data. This technique tries to find a line that separates 2 classes by minimizing the probability of misclassification. It uses logit function internally to calculate the probabilities. The

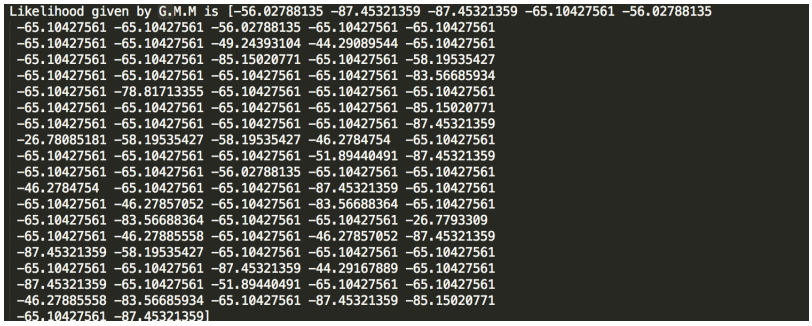

Figure 4.9: Negative likelihood given by G.M.M

following results were obtained when we ran this classifier on data present in datasets/flat-sample/outputAnonymizedFile.json:

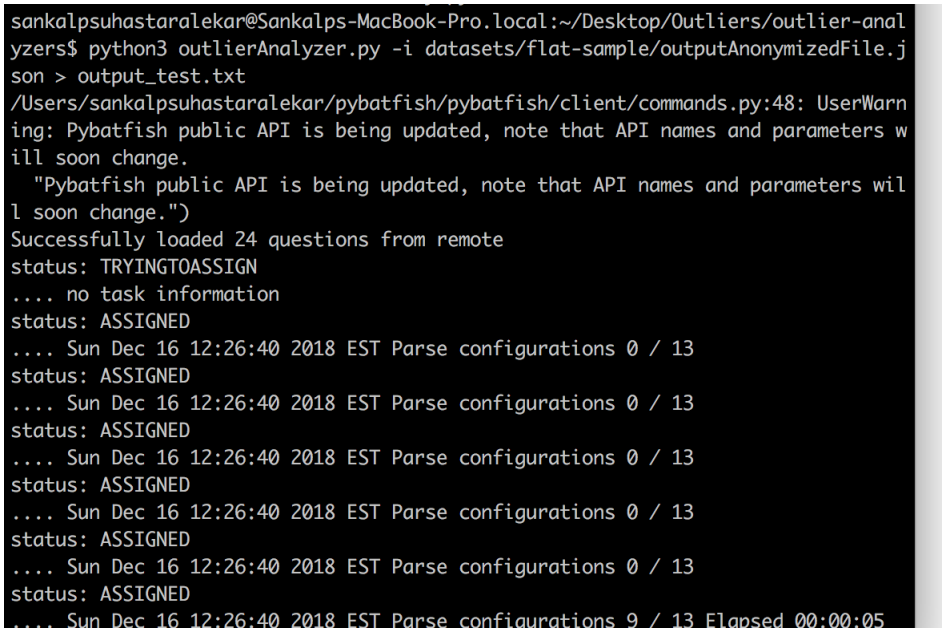

Figure 4.10: The output is directed to output-test file

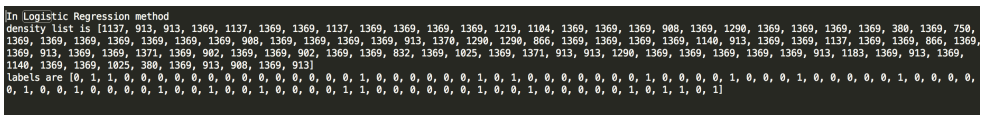

Figure 4.11: Labeled data after running Z-score threshold

One major drawback of this classification is that if the first outlier technique used to label the data performs badly, then the next classifier will also perform badly as the labels get propagated.

Figure 4.12: Classification done by Logistic Regression Classifier

4.6 Random Forests

This supervised learning technique is based on decision tree classification. In this technique, bootstrap dataset is formed by randomly selecting subset of features from some of the samples. Decision tree is formed based on these selected features. Now, when a new sample has to be classified, voting is done based on these existing trees. The class with the majority number of votes is selected as the final class. Out of bag samples are been used for the validation to ensure that there is no overfitting and classifier performs well on the new data.
Application on Network Data: As this is a supervised learning technique, we needed labeled data. The network data that we have is unlabeled data. So, we used an existing outlier detection to label the data. If the data point was detected as an outlier the label added was the 1, else the label was set to 0.The outlier detection technique used to label the data before using Random Forests Classifier was Z-score threshold. Then, we used train-test-split to split the data into training and testing sets. 67 percent of the data was used for training, while remaining 67 percent of the data was used for testing. The following results were obtained when we ran this classifier on the data present in datasets/flat-sample/outputAnonymizedFile.json:

One major drawback of this classification is that if the first outlier technique used to label the data performs badly, then the next classifier will also perform badly as the labels get propagated.

4.7 Isolation Forests

This technique can be used on unlabeled data and is used to detect outliers. Detecting various anomalies serves to be the main purpose of Isolation Forests. This anomaly detection is done by storing the rare value features closer to the root of the decision tree. Anomalies get detected based on length of the path in the decision tree classifier. Decision tree analysis gets used to average the values once anomalies are removed.

The results obtained when this classifier was run on data present in datasets/flat-sample/outputAnonymizedFile.json are as follows:

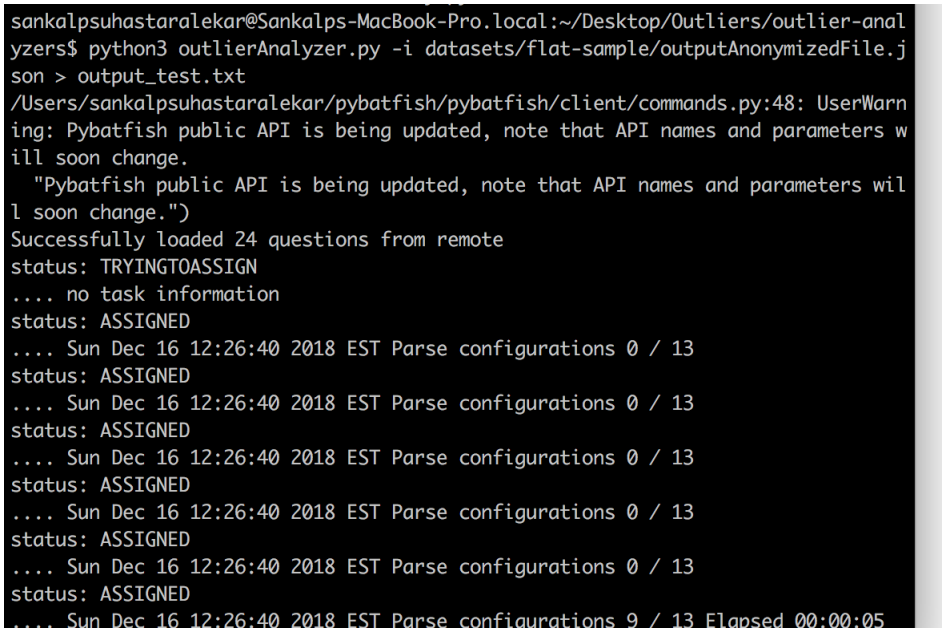

Figure 4.13: The output is directed to output-test file

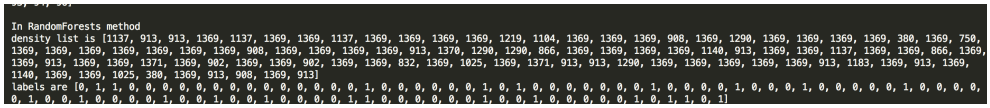

Figure 4.14: Labeled data after running Z-score threshold

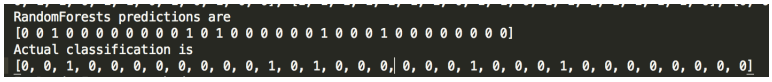

Figure 4.15: Classification done by Random Forest Classifier

4.8 Naive Bayes

This supervised learning technique uses conditional probability distribution to calculate which class a new sample belongs given the input vector for that sample. In the conditional probability table, the probabilities for the input vector given the class are stored. Assumption is made that all the features of the input vector are independent and thus we can multiply all the conditional probabilities. This is one of the major loopholes in this technique as all the features are not necessarily independent.

Application on Network Data: As this is a supervised learning technique, we needed labeled data. The network data that we have is unlabeled data. So, we used an existing outlier detection to label the data. If the data point was detected as an outlier the label added was the 1, else the label was set to 0.The outlier detection

```
sankalpsuhastaralekar@Sankalps-MacBook-Pro.local:~/Desktop/Outliers/outlier-anal
yzers$ python3 outlierAnalyzer.py -i datasets/flat-sample/outputAnonymizedFile.j
son > output_test.txt
/Users/sankalpsuhastaralekar/pybatfish/pybatfish/client/commands.py:48: UserWarn
ing: Pybatfish public API is being updated, note that API names and parameters w
ill soon change.
  "Pybatfish public API is being updated, note that API names and parameters wil
l soon change.")
Successfully loaded 24 questions from remote
status: TRYINGTOASSIGN
.... no task information
status: ASSIGNED
.... Sun Dec 16 12:26:40 2018 EST Parse configurations 0 / 13
status: ASSIGNED
.... Sun Dec 16 12:26:40 2018 EST Parse configurations 0 / 13
status: ASSIGNED
.... Sun Dec 16 12:26:40 2018 EST Parse configurations 0 / 13
status: ASSIGNED
.... Sun Dec 16 12:26:40 2018 EST Parse configurations 0 / 13
status: ASSIGNED
.... Sun Dec 16 12:26:40 2018 EST Parse configurations 0 / 13
status: ASSIGNED
.... Sun Dec 16 12:26:40 2018 EST Parse configurations 9 / 13 Elapsed 00:00:05
```

Figure 4.16: The output is directed to output-test file

```
isolationForests predictions on train data are
[ 1  1  1 -1  1  1  1  1 -1  1  1  1  1  1 -1  1  1  1  1  1  1  1
  1  1  1  1  1 -1  1  1  1  1  1  1  1  1  1  1  1 -1  1  1 -1  1
  1  1  1  1  1  1  1  1  1  1  1  1 -1  1  1  1]
isolationForests predictions on test data are
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

Figure 4.17: Results given by Isolation Forest Classifier

technique used to label the data before using Naive Bayes Classifier was Z-score threshold. Then, we used train-test-split to split the data into training and testing sets. 67 percent of the data was used for training, while remaining 67 percent of the data was used for testing. The following results were obtained when we ran this classifier on data present in datasets/flat-sample/outputAnonymizedFile.json:

One major drawback of this classification is that if the first outlier technique used to label the data performs badly, then the next classifier will also perform badly as the labels get propagated.

4.9 Pearson correlation

We used the Pearson correlation coefficient to calculate the severity value. The value of Pearson correlation coefficient was used to compare how closely two features of a particular device are associated with each other. Severity between two features is calculated using density lists of features. When we ran the Severity function on the features present in datasets/flat-sample/outputAnonymizedFile.json,

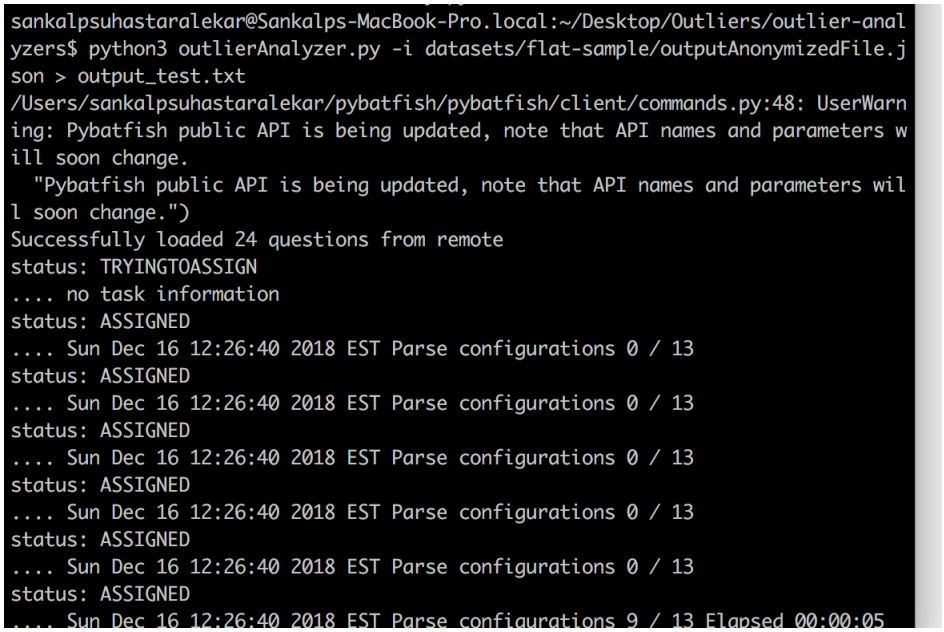

Figure 4.18: The output is directed to output-test file

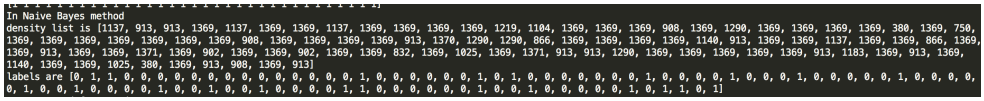

Figure 4.19: Labeled data after running Z-score threshold

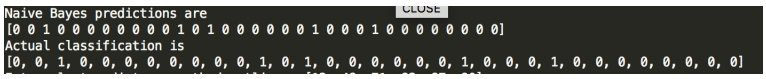

Figure 4.20: Classification done by Naive Bayes Classifier

the following results were observed:

4.10 K-Nearest neighbors

This supervised learning technique tries to classify a particular point depending upon the K closest neighbors to that point. We experimented with various values of K. The value of K was chosen to be odd to minimize ties for the closest neighbors. The key advantage of this method is that it has zero training time. Application on network data present in datasets/flat-sample/outputAnonymizedFile.json:

We used this technique to calculate the similarity between 2 devices in a network. In the nearest-neighbor classifier, the n-neighbors parameter was cho-

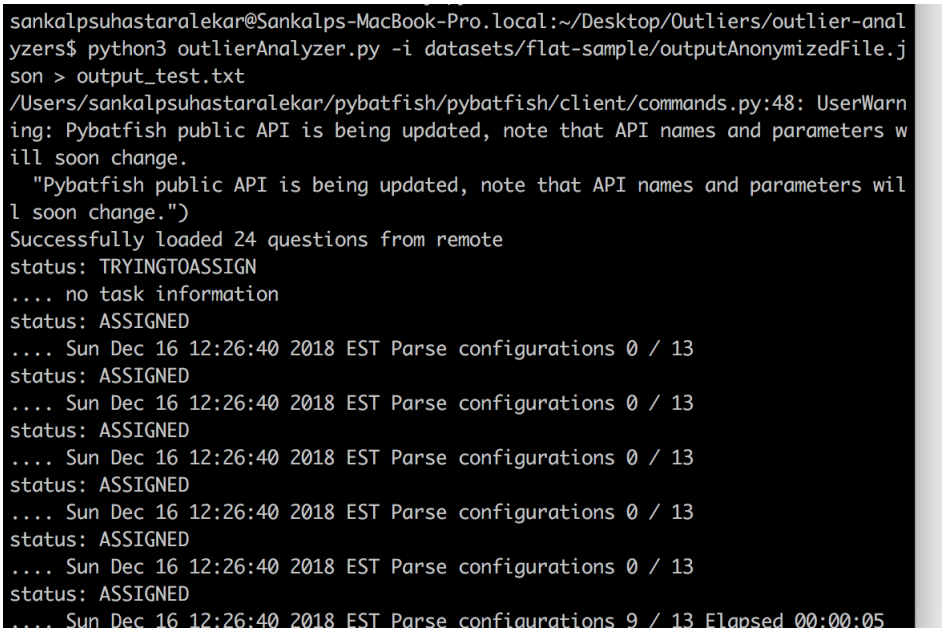

Figure 4.21: The output is directed to output-test file

sen to be 3. We passed the concatenated list of features to the classifier to obtain the 3 nearest neighbors for every device. The nearest neighbor classifier had the following results when ran on the data present in datasets/flat-sample/outputAnonymizedFile.json:

Distances given by the nearest neighbor classifier were as follows:

```
In severity function:
correlation between features indexed 0
and
1
-0.1123231385565642
correlation between features indexed 0
and
2
0.5730971769067262
correlation between features indexed 0
and
3
0.26789985988840953
correlation between features indexed 0
and
4
0.6951044469127048
correlation between features indexed 1
and
2
0.09893592640606788
correlation between features indexed 1
and
3
-0.030091353081529985
correlation between features indexed 1
and
4
-0.08612467324536466
correlation between features indexed 2
and
3
0.3783718770272406
correlation between features indexed 2
and
4
0.7649667745515929
correlation between features indexed 3
and
4
0.5061184161736787
```

Figure 4.22: Pearson correlation coefficient results

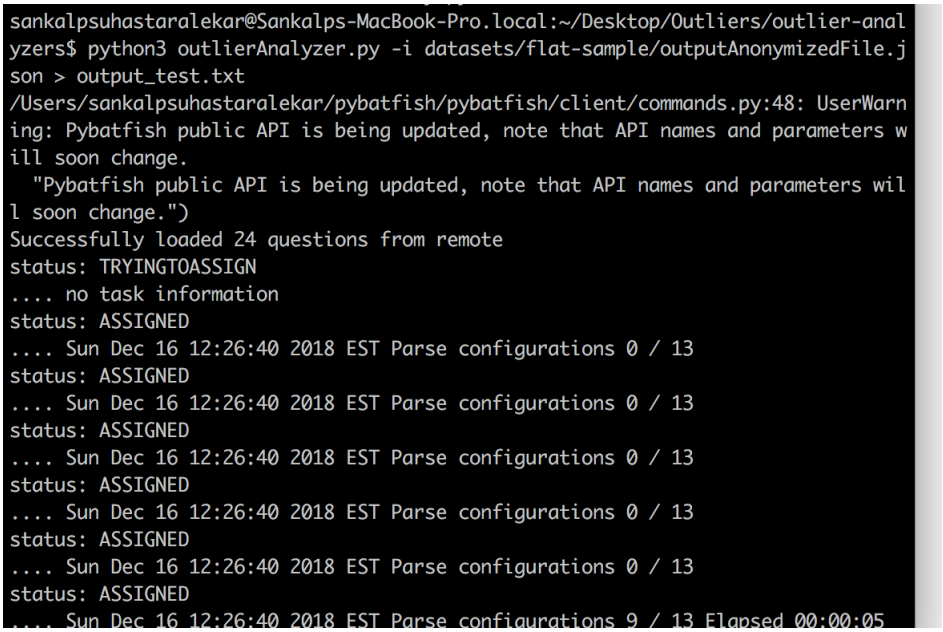

Figure 4.23: The output is directed to output-test file

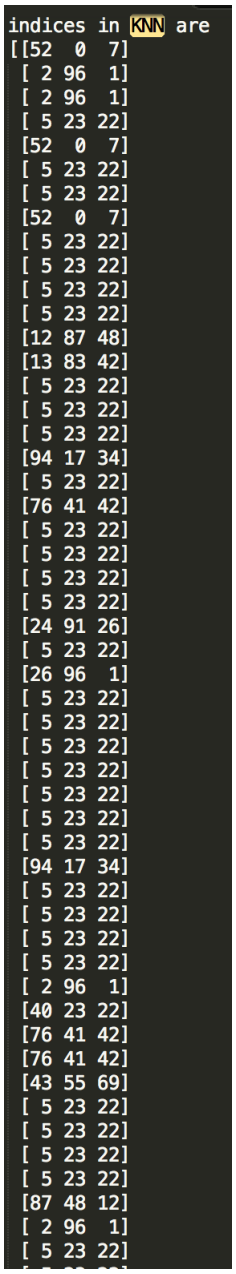

Figure 4.24: Index of Nearest Neighbors given by KNN

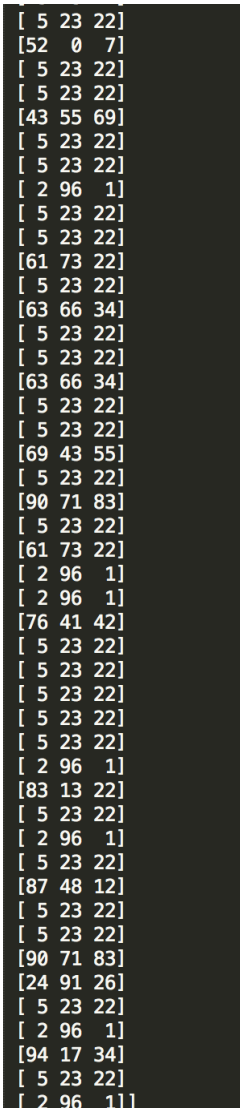

Figure 4.25: Index of Nearest Neighbors given by KNN

```
distances in KNN are
[[0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          1.          1.        ]
 [0.          1.          1.41421356]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          2.23606798]
 [0.          0.          0.        ]
 [0.          1.41421356 1.41421356]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          1.          1.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          2.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
 [0.          0.          1.        ]
 [0.          0.          0.        ]
 [0.          0.          0.        ]
```

Figure 4.26: Distance of nearest neighbors given by KNN

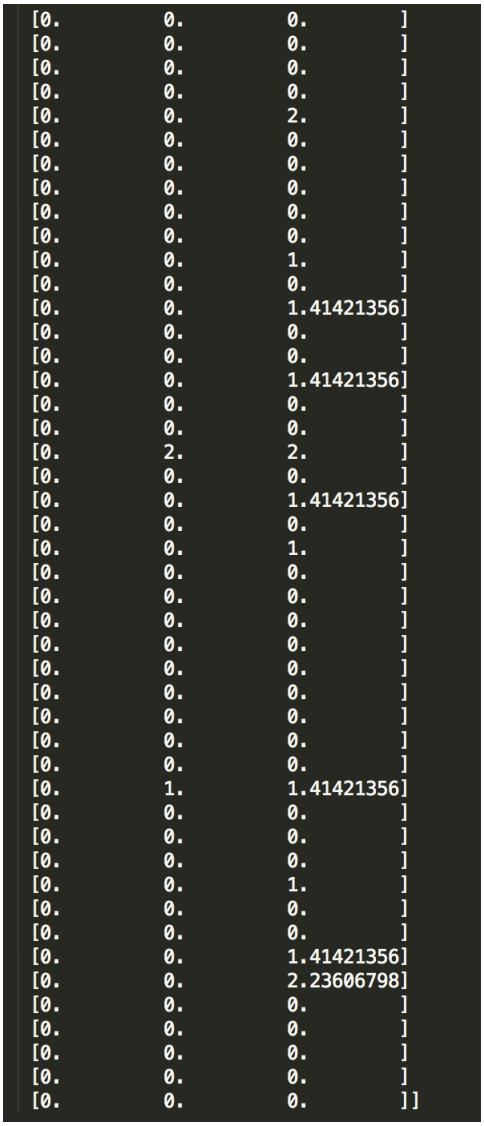

Figure 4.27: Distance of nearest neighbors given by KNN

Chapter 5

Statistical Approaches

Running from pybatfish for all techniques:

```
python3 outlierAnalyzer.py -t "nodeProperties()" -p "NTP_Servers | MTU"

# Running from pybatfish for specific technique:
python3 outlierAnalyzer.py -t "nodeProperties()" -p "NTP_Servers | MTU" -m 3

# Running definition flat files for all techniques:
python3 outlierAnalyzer.py -i <filename>

# Running definition flat files for specific technique:
python3 outlierAnalyzer.py -i <filename> -m 2

# Running server flat file for all techniques:
python3 outlierAnalyzer.py -ij <filename>

# Running server flat file for specific technique:
python3 outlierAnalyzer.py -ij <filename> -m 4

## Method codes:

    Tukey's method : 0
    Z-score method: 1
    Modified Z-score method: 2
    Cook's distance method: 3
    Intercluster distance method: 4
    Intracluster distance method: 5
    Mahalanobis distance method: 6
```

5.1 Z-Score

Another commonly used statistical technique is Z-score calculation. The most concise way to describe the Z-score is that it is the measure of how many standard deviations above or below the mean a value is. It is calculated by subtracting the value from the mean and dividing by the standard deviation. Standard deviation in turn is calculated by finding the square root of the variance (the squared vertical distance from the mean) of every point in the dataset. A Z-score of 0 means that the value is precisely the mean, while a value of -1 means that the value is one standard deviation below the mean. It is a dimensionless quantity (because both the numerator and denominator of the formula carry the same unit of measurement) which is useful because it is robust and unaffected by changing, for example, from miles to kilometers or Fahrenheit to Celsius. Outliers are detected using Z-score by defining a threshold (that can either be empirically determined or automatically calculated depending on the problem domain and nature of the data) and checking whether the Z-score for each point falls beyond the threshold or not. Like Tukey, this is a form of extreme value analysis that assumes a one-dimensional dataset. The advantage over Tukey's method, however, is that by utilizing the standard deviation it is sensitive to the "shape" of a distribution and does not just blindly consider the magnitude of potential outliers. However, it still requires an approximately normal distribution to provide meaningful output.When we ran this method on the network data present in datasets/flat-sample/outputAnonymizedFile.json, the following results were obtained:

```
Sample command using Batfish:
    python outlierAnalyzer.py -t "nodeProperties()" -p "NTP_Servers" -m 1
Output:
    Z-Score method outliers: [1, 4, 7, 12, 14]
    github.com/vasu018/outlier-analyzers/blob/master
    /Results/Pybatfish/z-score.txt

Sample command using flat file:
    python outlierAnalyzer.py -i datasets/flat-sample
    /namedStructureProperties_ip-accesslist.json -m 1
Output:
    Z-Score method outliers: [0, 1, 2, 4, 5, 6, 8, 9, 10, 13, 14,
    21, 35, 37, 41, 44, 45, 52, 53, 54, 55, 60, 70, 73, 74, 81, 83,
    84, 86, 87, 88, 95]
    github.com/vasu018/outlier-analyzers/blob/master/
    Results/Flat_ip_accesslist/zscore.txt
```

5.2 Modified Z-Score

For distributions that are skewed in some way, a tweak to the z-score outlier detection method can be made that provides more accurate results. Instead of using the mean and standard deviation to calculate the z-score, the technique can use the median and "median absolute deviation" instead. It is the same fundamental idea as the original z-score method, but because of its use of median statistics rather than mean statistics, it is more robust to data that is not evenly balanced either because of an uneven distribution or because some unusually high or low values have skewed the mean. By using the median, it is more immune to these issues and can still detect a reasonable set of outliers. When we ran this method on the network data present in datasets/flat-sample/outputAnonymizedFile.json, the following results were obtained:

```
Sample command using Batfish:
    python outlierAnalyzer.py -t "nodeProperties()" -p "NTP_Servers" -m 2
Output:
    Modified Z-Score method outliers: [1, 4, 7, 12, 14]
    github.com/vasu018/outlier-analyzers/blob/master/
    Results/Pybatfish/modified-z-score.txt

Sample command using flat file:
    python outlierAnalyzer.py -i datasets/flat-sample
    /namedStructureProperties_ip-accesslist.json -m 2
Output:
    Modified Z-Score method outliers: [7, 11, 12, 15, 16, 17, 19, 20,
    22, 23, 24, 25, 27, 28, 29, 30, 31, 33, 36, 38, 39, 40, 42, 43, 46,
    50, 51, 56, 57, 58, 62, 63, 64, 65, 67, 69, 71, 72, 78, 79, 80, 85,
    90, 91, 92, 94, 96]
    github.com/vasu018/outlier-analyzers/blob/master/
    Results/Flat_ip_accesslist/modifiedzscore.txt
```

5.3 Tukey's Method

One of the most basic outlier detection techniques is Tukey's method. It is a type outlier detection which assumes that the dataset is one-dimensional, the observations are independent among groups, and that the data is approximately normally distributed. It calculates the interquartile range (IQR), which is the difference between the first quartile (25th percentile) and the third quartile (75th percentile). Values whose magnitude lies 1.5x the IQR below the first quartile, and values

whose magnitude lies 1.5x the IQR above the third quartile are classified as outliers. It is considered a type of extreme value analysis, as the outlier detection mechanism detects values that are at each extreme (low and high). The advantage of this approach is that it is simple to understand and apply, and can work well for datasets in which extreme value analysis is an appropriate way to classify anomalies. On the other hand, the limitation of the approach is that in some datasets even values near the middle may be anomalous for various reasons, and that many datasets are two-dimensional or higher. When we ran this method on the network data present in datasets/flat-sample/outputAnonymizedFile.json, the following results were obtained:

```
Sample command using Batfish:
    python outlierAnalyzer.py -t "nodeProperties()" -p "NTP_Servers" -m 0
Output:
    Tukey's method outliers: []
    github.com/vasu018/outlier-analyzers/blob/master/
    Results/Pybatfish/tukey.txt

Sample command using flat file:
    python outlierAnalyzer.py -i datasets/flat-sample
    /namedStructureProperties_ip-accesslist.json -m 0
Output:
    Tukey's method outliers: []
    github.com/vasu018/outlier-analyzers/blob/master
    /Results/Flat_ip_accesslist/tukey.txt
```

5.4 Definition-based Outliers

Definition-based outlier detection is a method of applying the original outlier detection methods to Access Control Lists (and other network structures such as route maps). Signature-based outlier detection constructs a prototypical network structure that is then compared to each real entry for similarity. In contrast, for definition-based outlier detection we treat each key-value pair in the structure as a unique element, and run outlier detection methods such as Z-score, Mahalanobis, and Gaussian Mixture Model to detect outliers in the structure.

5.5 Cook's Distance Method

A more sophisticated outlier detection method is Cook's Distance, which involves first calculating a line that most closely fits the data points using linear regression.

The premise of the technique is that the data points that have the greatest influence on calculating the linear regression line are outliers. This is determined by deleting each point in turn and recalculating the linear regression line and seeing how much the new and old lines differ. Outliers are classified by dividing the point's Cook's distance by the mean of all the Cook's distances. A typical threshold value is 3. When we ran this method on the network data present in datasets/flat-sample/outputAnonymizedFile.json, the following results were obtained:

```
Sample command using Batfish:
    python outlierAnalyzer.py -t "nodeProperties()" -p "NTP_Servers" -m 3
Output:
    Cook's distance method outliers: [0, 1, 4, 11, 12, 13, 14]
    github.com/vasu018/outlier-analyzers/blob/master/Results/
    Pybatfish/cooks.txt

Sample command using flat file:
    python outlierAnalyzer.py -i datasets/flat-sample/
    namedStructureProperties_ip-accesslist.json -m 3
Output:
    Cook's distance method outliers: [0, 1, 2, 4, 5, 6, 88, 95]
    github.com/vasu018/outlier-analyzers/blob/master/Results/
    Flat_ip_accesslist/cooks.txt
```

5.6 Mahalanobis distance

For data that is multi-dimensional, an approach known as Mahalanobis distance can be used. It can be viewed as a generalization of the Z-score method in that it measures how many standard deviations away from the mean a given data point is. If a point is equivalent to the mean, its Mahalanobis distance is 0. As it moves away from the mean along any of the dimensional axes, the Mahalanobis distance grows. If each axis is considered to be unit length, then the Mahalanobis distance is equivalent to the Euclidean distance. It makes use of linear algebra and matrices representing the multidimensional points and the covariance matrix.

```
Sample command using Batfish:
    python outlierAnalyzer.py -t "nodeProperties()" -p "NTP_Servers" -m 6
Output:
    Malanobis distance method outliers: []
    github.com/vasu018/outlier-analyzers/blob/master/Results/
    Pybatfish/mahalanobis.txt
```

```
Sample command using flat file:
    python outlierAnalyzer.py -i datasets/flat-sample/
    namedStructureProperties_ip-accesslist.json -m 6
Output:
    Malanobis distance method outliers: [0, 1, 2, 4, 5, 6, 8, 9, 13, 14, 21,
    37, 41, 44, 45, 52, 54, 60, 70, 73, 74, 81, 83, 84, 88, 95]

    github.com/vasu018/outlier-analyzers/blob/master/Results/
    Flat_ip_accesslist/mahalanobis.txt
```

Chapter 6

Experiments: Signature-based Approach

6.1 Signature-based Outliers Detection

In addition to definition-based outlier detection, signature-based outlier detection is another method that can be used to find outliers for access control lists and similar network structures. Rather than analyzing the data directly and looking for data points that are anomalous relative to each other, the signature-based method constructs a prototypical signature that approximately represents the most common characteristics of the set of access control lists or other structures, and then compares each actual structure to the prototype and calculates a similarity score. This similarity score is used as a threshold to determine which structures are outliers.

We ran the signature detection on large real-world datasets, including definitions of IP Access Lists and Routing Policies. One advantage we found is that instead of a binary classification of "outlier" or "non-outlier", the signature-based method assigned every entry a similarity score based on how closely the entry's fields matched the prototype's fields. The number of outliers identified varied depending on how high a threshold we set for the similarity score. With a very high threshold, the signature outlier detection technique identified fewer outliers than techniques such as modified Z-score or cook's distance. With a lower threshold, it identified more outliers than any statistical-based technique. We found a middle ground where the signature-based technique found a comparable number of outliers as the previous techniques, but had the advantage of providing additional fine-grained information about each entry.

```
## Sample command using IP ACL flat file:
        python signatureOutlier.py datasets/flat-sample/
```

```
        namedStructureProperties_ip-accesslist.json
## Output file:
        github.com/vasu018/outlier-analyzers/blob/master/Results/
        Signature/signature_ip_acl.txt

## Sample command using route policies flat file:
        python signatureOutlier.py datasets/flat-sample/
        namedStructureProperties_routepolicies.json
## Output:
        https://github.com/vasu018/outlier-analyzers/blob/master/Results/
        Signature/signature_routing.txt
```

Chapter 7

Some Useful Commands

Note: Be careful some of the commands might have been out dated.

7.1 Install virtual environment

https://help.dreamhost.com/hc/en-us/articles/115000695551-Installing-and-using-virtualenv-with-Python-3

```
virtualenv . -p /Library/Frameworks/Python.framework/Versions/3.4/bin/python3

source  bin/activate \enquote{or}
source .env3/bin/activate
```

7.2 Git Commands list

http://guides.beanstalkapp.com/version-control/common-git-commands.html

7.3 IntelliJ IDE Settings

https://github.com/batfish/batfish/wiki/IntelliJ
https://www.jetbrains.com/help/idea/docker.html

7.4 Run batfish

```
source tools/batfish_functions.sh
* batfish_build_all
* mvn -f projects/pom.xml -pl batfish generate-sources
* .travis/fix_java_format.sh
```

```
* .travis/build.sh
* "Or"
*  mvn -f projects/ clean install
allinone -runclient false -coordinatorargs "-templatedirs ../pybatfish/questions''
```

7.5 Pybatfish Commands

```
pip install pybatfish/ -upgrade
init-testrig ../test-rigs/example2
```

7.6 Web UI for Batfish

```
npm install
npm run start
```

7.7 Batfish Java Client Commands

```
get roles
get outliers
get outliers hypothesis=‘sameDefinition'
get outliers hypothesis='sameName'

init-testrig test_rigs/example roles-example
init-testrig test_rigs/Visa/ini-20180522/

Hypothesis-based outliers:
get outliers hypothesis=‘sameDefinition'
get outliers hypothesis=‘sameName'
get outliers hypothesis='sameServers'

get inferRoles
Roles and RoleDimensions commands:
get perRoleOutliers hypothesis='sameDefinition', roleDimension=‘auto1'
get perRoleOutliers hypothesis='sameServers', roleDimension=‘auto1'
get perRoleOutliers hypothesis='sameName', roleDimension='auto1'
```

7.8 Git Commands

```
## Code submission and review:
# Create a new local branch
git checkout -b my-branch-name
# Queue changes
git add what_i_changed.py
# Commit changes locally
git commit -m ‘‘My comments here’’
# Create new remote branch with current committed changes
git push --set-upstream origin my-branch-name
# If you commit more local changes after that, then you can just push them with
git push

# Be careful with this command.
git reset --hard origin/master

## Adding the diff unit test cases:
diff -u unit-tests-nodes.ref unit-tests-nodes.ref.testout  > diff_nodes
  554  patch < diff_nodes

## When some code from master branch sneaked into your branch:

git fetch --all
git merge origin master

git branch
git log origin/master..
git diff origin/master --stat
git commit --allow-empty -m "noop"
git push -f
git status

git rebase -i origin/master
git push -f
```

7.9 batfish commands

```
* source tools/batfish_functions.sh
*  batfish_build_all
```

```
allinone -runclient false -coordinatorargs "-templatedirs ../pybatfish/questions''

## Run tests:
sudo python setup.py teste2e
```

Run java Apis:
java -jar projects/intentionet-all/target/intentionet-all-bundle-0.36.0.jar -runmode interactive

7.10 Test commands (Not much useful with current versions)

```
test tests/roles/roleNeighbors.ref get neighbors
neighborTypes=["ebgp","ibgp","ospf","lan"], style="role", roleDimension="default"

test tests/roles/roles.ref get roles roleDimension="default"

test tests/roles/perRole.ref get perRole question={"class":
"org.batfish.question.CompareSameNameQuestionPlugin$CompareSameNameQuestion"},
roleDimension="default"

test tests/roles/perRoleOutliers.ref get perRoleOutliers hypothesis="sameName",
roleDimension="default"

test tests/roles/inferPolicies.ref get inferPolicies roleDimension="default"

test tests/roles/roleConsistency.ref get roleConsistency
propertyName="LoggingServers", roleDimension="default"

test tests/roles/nsRoleConsistency.ref get namedStructureRoleConsistency
hypothesis="sameName", structType="CommunityList", roleDimension="default"

# example2 testrig

init-testrig test_rigs/example2 roles-example2

test -compareall tests/roles/inferRoles.ref get inferRoles

test tests/roles/roleNeighbors2.ref get neighbors
```

```
neighborTypes=["ebgp","ibgp","ospf","lan"], style="role"
```

7.11 Fix to change the run options for running Intentionet code

```
diff --git a/.idea/runConfigurations/intentionet_all.xml
b/.idea/runConfigurations/intentionet_all.xml
index 138f61c..840153d 100644
--- a/.idea/runConfigurations/intentionet_all.xml
+++ b/.idea/runConfigurations/intentionet_all.xml
@@ -2,7 +2,7 @@
   <configuration default="false" name="intentionet-all" type="Application"
   factoryName="Application" singleton="true">
     <option name="MAIN_CLASS_NAME" value="org.batfish.allinone.Main" />
     <module name="intentionet-all" />
-    <option name="PROGRAM_PARAMETERS" value="-runclient
false -coordinatorargs
&quot;-templatedirs questions/stable,questions/experimental&quot;" />
+    <option name="PROGRAM_PARAMETERS" value="-runclient false -loglevel
info -coordinatorargs &quot;-templatedirs questions/stable,questions/experimental
&quot;" />
     <option name="VM_PARAMETERS" value="-Xmx12G -XX:+UseG1GC -server
     -XX:MaxGCPauseMillis=200" />
     <option name="WORKING_DIRECTORY" value="file://$PROJECT_DIR$" />
     <method />
```

Chapter 8

Code Documentation

We are using mainly 2 files:
OutlierAnalyzer.py, signatureOutlier.py and outlierLibrary.py.

Note: 1. For statistical and ML techniques the code is in the OutlierAnalyzer.py and in the outlierLibrary.py file.
2. For signature-based outlier analyzer the code is in the signatureOutlier.py and in the outlierLibrary.py file.

8.1 ML-based Outlier Analyzer

@Vasu: Details to be added for ML-based techniques

8.2 Statistical Outlier Analyzer

We used a variety of statistical techniques for detecting outliers, including Tukey's method, Z-score, modified Z-score, Cook's distance, and Mahalanobis distance. For Tukey's method, we calculated the first and third quartiles using NumPy, and then calculated the interquartile range (IQR), defined as the difference between these two values. Then, for each data point we test whether it is below the first quartile minus 1.5 IQR and the third quartile plus 1.5 IQR. Any points that meet this criteria are classified as outliers. It takes a list of integers as input and returns a list of integers representing the indexes of the outliers.

For Z-Score method, we first calculate the mean and standard deviation using NumPy, and then calculate a Z-score for each point by subtracting the data point's value from the mean, divided by the standard deviation. We chose a Z-score threshold of 1.0, finding that to be a good balance between classifying too many points as outliers and not classifying enough. It takes a list of integers as input and returns a list of integers representing the indexes of the outliers.

The modified Z-score method is the same structure as the Z-score method, but instead of mean it uses median, and instead of standard deviation it uses an analogous metric called median absolute deviation. It takes a list of integers as input and returns a list of integers representing the indexes of the outliers.

Cook's distance takes a list of tuples as input, with each tuple being a pair of integers representing a two-dimensional point. It first calculates a regression line based on this sets of point, represented by the integers 'a' and 'b'. The integer 's' represents the mean squared error. For each point, it generates a new line using the predict() function assuming the current point is missing, and calculates the square of the difference and then divides the value by 3 * MSE, according to the Cook's distance formula. If the distance is over 0.05, it is classified as an outlier. It returns a list of integers representing the indexes of the outliers.

Mahalanobis's distance takes a list of list of integers as input, and returns a list of integers representing the indexes of the outliers. It creates a vector for each density list, then calculates an average vector representing the arithmetic mean of all the vectors. It then uses NumPy library to calculate the Mahalanobis distance according to the Mahalanobis distance formula.

8.3 Signature-based Outlier Analyzer

@Alfred: I need code design documentation for the signature-based outliers analyzer. [done]

Signature-based outlier detection is an alternate approach to detecting outliers. The previous statistical outlier detection methods worked by applying quantitative techniques directly to a dataset to identify outlier values. The signature-based method instead constructs a prototype data point and compares all the real data points to it and calculates how similar they are. In this case, a data point is a structure (such as an access control list) which is represented in Python as a nested dict.

The code first opens the file specified by the user as a command line argument and reads through it. It then reads through the file line by line. Each line in the file represents an entire nested dict corresponding to a column of the ACL (or other type such as route policy) structure. This nested dict is parsed by Python's json module and appended to a list of lists 'datas'. Only the rows specified by the user in the 'selection' list variable are saved. After this there are two utility functions, isHomogenous() and $\text{extract}_{k}eys(), whichwillbeexplainedlateron.$

The 'overall' variable is a dict. This dict is used to construct the signature. Each key of 'overall' is a field that appears in at least one of the entries. For example, for the IP Access List flat file, there exists a field called "sourceName" in most of the entries/row of the input flat file. However, because each entry is a nested dict, some fields for an entry appear in inner dicts. For those values, they are specified

fully using dot notation, such as “lines.matchCondition.headerSpace.class” (which represents a field nested 4 levels deep).

The ‘overall’ variable is constructed by iterating through each entry. For each entry, every field it contains is recursively extracted from it using the $extract_keys()function.Theextract_keys()functionsimplyrecursivelytraversesthenesteddictsandcreatesalistoffiel$

The values corresponding to those keys are themselves dicts. These dicts are mappings between strings and integers. The strings are values and the integers are counts of how many times those values appear across the entire flat file. To make things concrete, here is an example:

Row 1: “sourceName”: “NAME1”, “lines”: “action”: “PERMIT”, class: “org.batfish”, “matchCondition”: “negate” : “false”
Row 2: “sourceName”: “NAME2”, “lines”: “action”: “PERMIT”, “matchCondition”: “negate” : “false”
Row 3: “sourceName”: “NAME3”, “lines”: “action”: “DENY”, ipProtocols: “ICMP”, “matchCondition”: “negate” : “false”

Say the flat file contains these 3 rows. The code iterates through all three rows, recursively extracting every field from every row. The ‘overall’ dict now contains these keys:

“sourceName”, “lines.action”, “lines.class”, “lines.matchCondition.negate”, “lines.ipProtocols”

Each key is mapped to another dict, which itself maps values to counts:
“sourceName” -> “NAME1”: 1, “NAME2”: 1, “NAME3” : 1
“lines.action” -> “PERMIT” : 2, “DENY” : 1
“lines.class” -> “org.batfish” : 1
“lines.matchCondition.negate” -> “false” : 3
“lines.ipProtocols” -> “ICMP” : 1

The purpose of these mappings is to use this information to build a signature. By recording the count of each value across the entire dataset, we can see which is the most frequent value and include that in the signature prototype. However, we only include key-value pairs in the signature with a low variance. What this means is that, for example, a field like “sourceName” could be different for every single entry, since each entry has its own name. There is no purpose in including that field in the signature, as in our above example there are three values for three rows. We chose to only include fields where the most frequent value makes up >70% of the total number of values. The 70% threshold is arbitrary and can be tuned, and for this small example we pretend that the value is 50%.

In that case, “sourceName” is excluded because no value exceeds 33% ratio, but “lines.action” is included because “PERMIT” has a 67% ratio, and “lines.class”, “lines.matchCondition.negate”, and “lines.ipProtocols” are all included because their only values have a 100% ratio. This ratio is also used to calculate a weight.

The final signature looks like this:
“lines.action” -> “PERMIT”
“lines.class” -> “org.batfish”
“lines.matchCondition.negate” -> “false”
“lines.ipProtocols” -> “ICMP”

Note that generally speaking a prototype may not exactly match any actual row, hence why it is called a prototype.

Finally, the prototype is compared to every real entry/row and a similarity score is calculated. Entries with a similarity score below a certain threshold (which can be tuned) are classified as outliers.

References

[1] Categorical variable. 2018. https://en.wikipedia.org/wiki/Categorical_variable.