A Case Study and Methodology for using Anomaly Detection and Supervised Classification for Incident Prediction.

Contents

I. Introduction and Overview

Quickly identifying the emergence of and source of a service impacting situation is a central goal of IT and Network Operations for Enterprises with large, distributed, computing infrastructure. The closer the service is to the source of revenue for the Enterprise, the more human resources are applied to the work of collecting, processing, and ultimately visually monitoring the large amounts of event and log data that serves as the primary source of symptomatic indicators for emerging situations. While automation can be applied at each layer of the Incident Detection process: *monitor*, *detect*, *alert*, *correlate*, *validate*, and *assign,* efforts to automate Incident Detection and Response have been limited by the sheer volume and variety of logs and events that flow into the process. The investment required to write automation rules (e.g. polling rules, correlation rules, runbooks, etc.) for each layer is significant and the rate of entropy of the collection of automation rules grows rapidly over time. Moving from fixed rules to a machine learning based approach to Incident Detection will reduce the substantial effort required to build and maintain the rules at each layer of the process, reduce the work to identify and manage incidents, and speed up the detection, and therefore resolution, of the Incidents.

Some of the challenges of attempting to apply machine learning techniques to this problem are related to the scale of the feature space, which can exceed 10 Million distinct scalar features. With the size of the input space, any approach that requires feature engineering can be intractable.

Our contribution to this problem of Incident Prediction is that of using an unsupervised anomaly detection algorithm to compute anomaly likelihood values, which de-noises and scales the input space, and applying a supervised training loop comprised of a Random Forest to classify Incident patterns in the anomaly likelihood features. The success of the Random Forest learning algorithm is then evaluated based on accuracy, precision, and recall, and is demonstrated to perform well against legacy incident detection methods based on fixed rules.

II. Methodology

We first formalize the methodology and then look at the results of our analysis.

1.1 Anomaly Detection

We first apply an anomaly detection algorithm against each metric. Formally, for a set of 'd' enumerated objects (such as a Network Switch#1, Host #1,... Etc.) we first consider a historical sample of performance data from which to begin our analysis. We consider the collection of such d objects, **Ω, where Ω** = { Ω_1 , Ω_2 , ….., Ω_d }, and each Ω_i = {M_{i1}, $M_{i2,\ldots\ldots}$ M_{in} } where each entry in the subset $\boldsymbol{\Omega}_i$ is itself a set of tuples, where M_{it} = {(timestamp$_1$,M_{it-1}), (timestamp$_2$,M_{it-2}),…….(timestamp$_p$, M_{it-p})}, each tuple containing a timestamp and a scalar metric value and where p is the total number of samples of the t_{th} metric on the object $\boldsymbol{\Omega}_i$.

The data collected for this study spans over 300 core network devices with each object containing between 200 and 5000 distinct metrics, for a total of approximately 250,000 metrics across the 300 objects.

For the purpose of this work we use Numenta's HTM algorithm to calculate the Anomaly Likelihood score for each of the p samples for each of the n metrics in the ith collection Ω_i. So we thus augment the data structure to include the anomaly likelihood score α, where

$\widehat{\Omega}$ contains the collection of each of the p triplets for each of the n Metrics on each of the d objects, where

$$\widehat{\Omega}_i = \{\widehat{M}_{i1}, \widehat{M}_{i2,\ldots\ldots} \widehat{M}_{in}\} \text{ and}$$

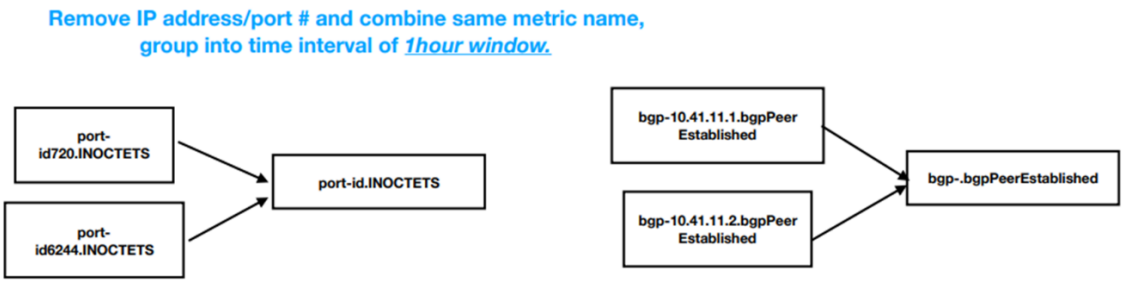
$$\widehat{M}_{it} = \{(\text{timestamp}_1, M_{it-1}, \alpha_{it-1}), (\text{timestamp}_2, M_{it-2}, \alpha_{it-2}), \ldots\ldots, (\text{timestamp}_p, M_{it-p}, \alpha_{it-p})\}$$

Though the dataset discussed so far contains historical data, we can also consider using this anomaly detection technique to notify IT Operations whenever a particular metric is observed to be in an anomalous state. The challenge with this construct lies in the nature of the behavior of scaled, redundant, distributed (SRD) systems. If we look for anomalies at the point of highest resolution, i.e. the individual metric, we are likely to observe anomalous behavior that while genuinely anomalous, is not indicative of an emerging situation. For example, if a CPU has been running at 20% utilization for 6 months, it is certainly anomalous for the CPU to spike to 40% but it is likely not indicative of an emerging situation. SRD systems behavior can be affected by attributes external to the raw metrics, such as capacity or operating parameters and built-in redundancies. Hence in order to minimize noise and maximize accuracy, we look to apply historical context to operating patterns through supervised learning.

1.2 Classification

In order to make the input space more tractable, we applied the following dimensionality reduction steps:

(1) For each $\boldsymbol{\Omega}_i$ we created a modified set of metric-tuples $\widehat{M}_{it}$ where we took the union of any such collection that is similar.
(2) We consider 2 collections $\widehat{M}_{xt}$ and $\widehat{M}_{yt}$ to be similar if the metric name is identical save having different IPAddress and/or Port ID specified in the metric name (see below).



After reducing the input space to roughly 200 base features, we generated a set of 5 statistics for the training data across each 1-hour interval from the earliest timestamp to the latest.

- Statistic 1: Mean
- Statistic 2: Max
- Statistic 3: Min
- Statistic 4: Skew
- Statistic 5: Kurtosis

Once we have the full features formalized, we label each sample with the correct classification, where 0 = “no Incident ticket” and 1 = “an Incident ticket”. In order to do this we iterate through each ticket in the Incident history that fit the following criteria: (1) the ticket has the correct datacenter label.

Final data Format
(unbalanced datasets, 0.6% positive examples)

Is a ticket created for this device at time interval t ~ t+1hour ?

Timestamp	FM1_min	FM1_max	FM1_var	...	FM100_min	FM100_max	Target
T1							0
T2							1
T3							0

After preparing the dataset we end up with 364 samples with 1078 total features.

```
X.shape
```

```
(364, 1078)
```

We then split the data using the scikit learn train_test_split package.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
X_train.shape, X_test.shape, sum(y_train), sum(y_test)
```

```
((291, 1078), (73, 1078), 56, 12)
```

Because we have a biased dataset with only 23% of the dataset representing the positive class, we evaluate several up-sampling techniques to balance the dataset and overcome the implicit negative

bias. We use both the Synthetic minority oversampling technique (SMOTE) and the Adaptive Synthetic (ADASYN) sampling technique. SMOTE creates artificial data based on the feature space similarities between existing minority class by introducing non-replicated minority class. ADASYN sampling uses a weighted distribution for different minority class examples according to their level of learning.

We alternately use SMOTE and ADASYN and evaluate the results separately.

In [31]:
```
# Upsample the training set using SMOTE
sm = SMOTE(random_state=12, ratio=1.0)
X_train_res, y_train_res = sm.fit_sample(X_train, y_train)
X_test_res, y_test_res = sm.fit_sample(X_test, y_test)

# Upsample the training set using ADASYN
ada = ADASYN()
X_train_res, y_train_res = ada.fit_sample(X_train, y_train)
X_test_res, y_test_res = ada.fit_sample(X_test, y_test)
```

We choose Random Forest Classification for our prediction function. Random Forests are an ensemble learning method for classification tasks. Random Forests work by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes.

In [32]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn import metrics
X_train = X_train_res
y_train = y_train_res
X_test = X_test_res
y_test = y_test_res


clf = RandomForestClassifier(max_depth=15, n_estimators=300)
clf.fit(X_train, y_train)
```

Out[32]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=15, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=300, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

III. Analysis of Results

First we apply SMOTE to up-sample the dataset. After applying SMOTE, we generate 51 additional positive training examples.

```
y_test
```

```
array([0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 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], dtype=int64)
```

```
len(y_test)
```

```
124
```

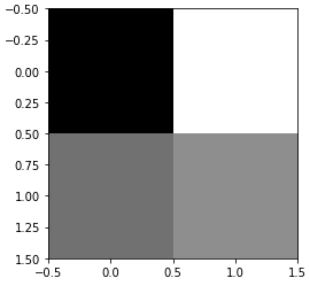
We look at both the accuracy, precision, and the recall of the model and consider the probability distribution of each class.

```
In [36]: y_pred_test = clf.predict(X_test)
         y_prob_test = clf.predict_proba(X_test)
         accuracy_score(y_test, y_pred_test), precision_score(y_test,y_pred_test),recall_score(y_test,y_pred_test)
Out[36]: (0.6935483870967742, 0.875, 0.45161290322580644)
```

We can see from the low recall score for the model that even with using SMOTE, the model is still negatively biased.

The confusion matrix provides a better visual illustrating the fact that only 28 of the 'true positive' samples in the test set have been predicted by our trained model.

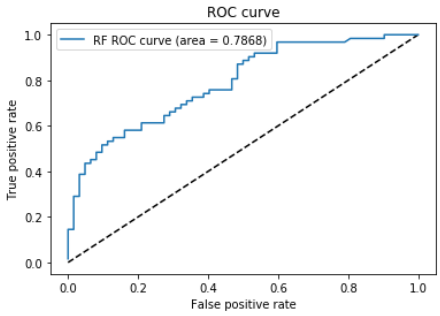
```
In [42]: cm2 = confusion_matrix(y_test, y_pred_test)
         plt.imshow(cm2, cmap='binary',interpolation='None')
         cm2, plt.show()
```



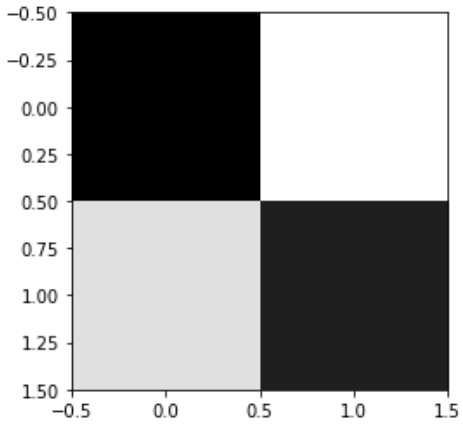

```
Out[42]: (array([[58,  4],
                 [34, 28]], dtype=int64), None)
```

The ROC curve plots the tradeoff between sensitivity and specificity where any increase in sensitivity will be accompanies by a decrease in specificity. The below ROC curve plots the true positive rate against the false positive rate for the Incident Prediction classifier.

In [44]:
```
y_test_prob = clf.predict_proba(X_test)
fpr_rf, tpr_rf, _ = metrics.roc_curve(y_test, y_test_prob[:,1])
roc_auc = metrics.auc(fpr_rf, tpr_rf)
plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rf, tpr_rf, label='RF ROC curve (area = %0.4f)' % roc_auc)
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```

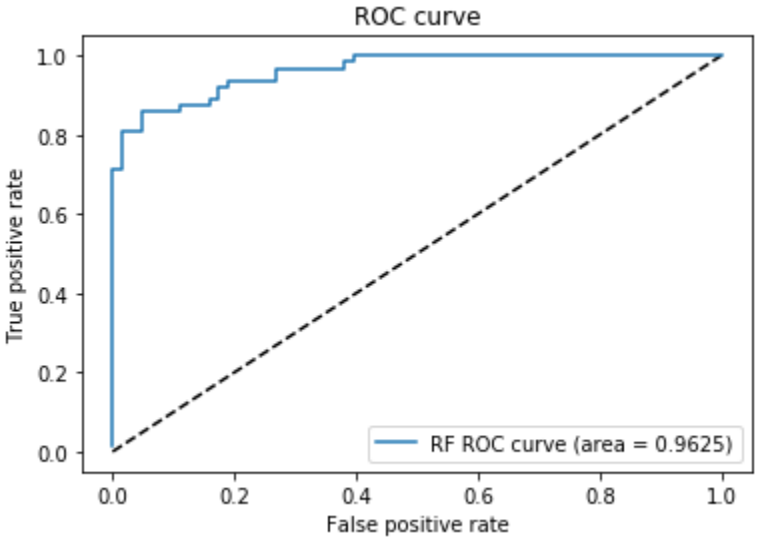


For ADASYN we get better results. In addition to moving the recall rate from 45% to 82%, we also increase the precision of the model, as is demonstrated from the confusion matrix.




```
(array([[60,  3],
        [10, 53]], dtype=int64), None)
```

The area under the ROC curve (AUC) is a measure of how well our Incident Prediction Classifier distinguishes between anomaly signatures that are indicative of emerging Incidents against those that are not so indicative.



The AUC when ADASYN is applied, and thus the implicit negative biased is counter balanced, is demonstrably better- from 78% to 96%. One explanation for the better result is likely related to the fact that ADASYN creates different synthetic samples for the minority class depending on its distribution and not just for the borderline instances.

IV. Conclusion

While anomaly detection provides a promising means of processing large volumes of IT operational data and bringing appropriate attention to emerging negative situations, the nature of IT operational data is such that the volume of metrics available and relevant for monitoring will generate an intractable number of anomalies to investigate. We have shown that by applying a supervised learning approach to filter out anomalies that are only reflective of discreet adjustments naturally occurring in the environment, we can provide a much more useful result that can perform well against traditional threshold-based event management techniques. Indeed we have shown that we are able to predict Incidents with a 90% accuracy rate with only 3 weeks of training data.

About Tribal Genius

Tribal Genius is a global service company with the most robust AIOps Framework for IT, Network and Customer Care. We believe in an AIOps future where systems respond to changes dynamically within their environment. Our experience and knowledge from past engagements have informed our investment in IP around the machine learning and automation space.

For more information on how Tribal Genius can help your business write us at Info@tribalgenius.com.

www.tribalgenius.com