AnoMLy No More

An AI/ML unsupervised learning approach to detect anomalies in apps

Team Outliers

Overview

The problem at hand is to analyze consumer data to detect anomalies. At Team Outliers (ironic much?) we use **data mining and machine learning** algorithms to detect these. Anomaly detection, also referred to as outlier detection, is used to find critical incidents, such as **a technical glitch, fraud, or logistical obstacle, or potential opportunities**, like a change in consumer behavior. Companies can use our product (algorithm) to make their businesses **less vulnerable to such malicious activities.** The idea is to analyze consumer data on multiple grounds such as:

- Time Series KPI
- Hour
- Date Data – Day, Month, Day of the Week
- Events – Event_Name
- Users

And much more and find out the outliers in the system. We have tested our program on the data given in the problem statement. You can find the code along with our submission, we have also added snippets of the code in this presentation explaining the important functions. The next few slides cover the theory and details of our approach (and they includes pretty graphs!).

Logic

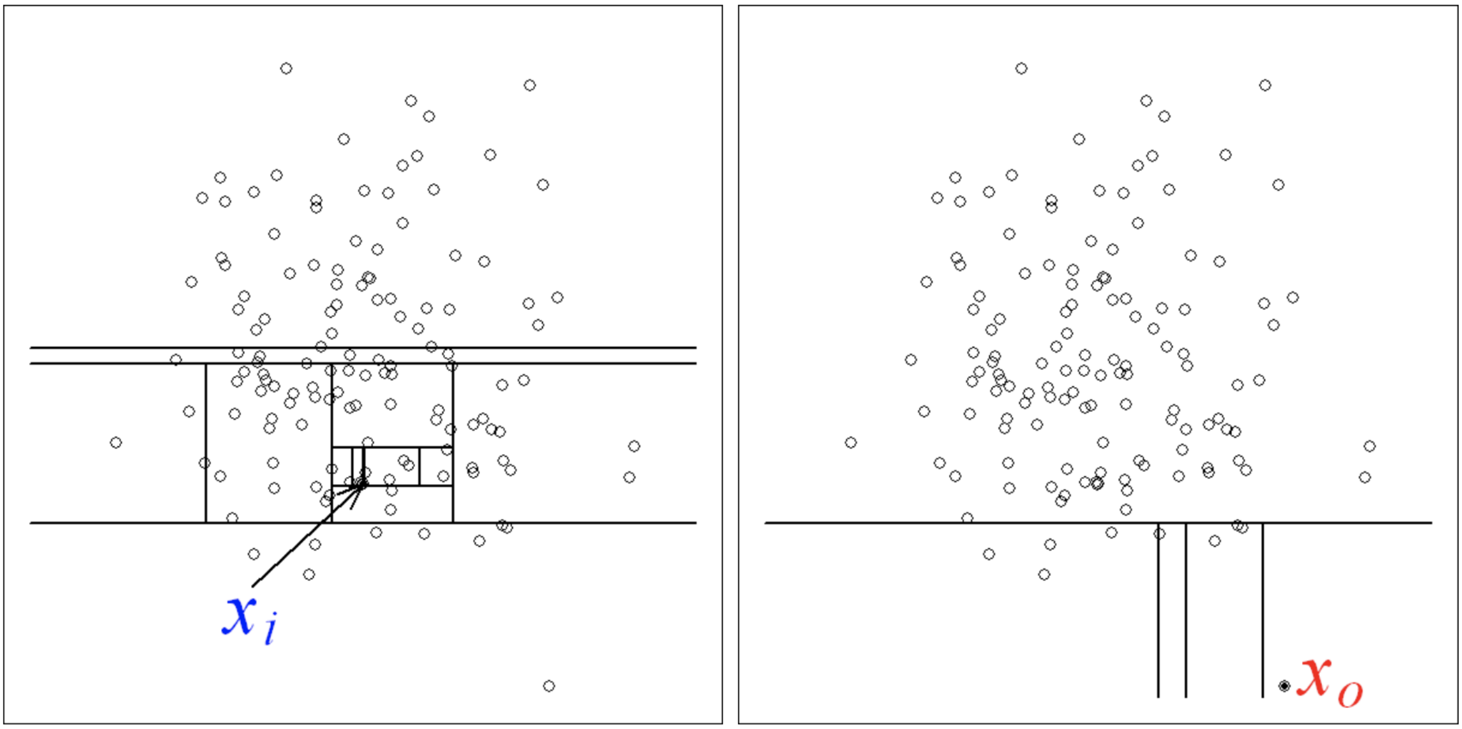
Before we dive into how our algorithm detects outliers, we explain the theory behind the concept and how it relates to our project.

Theory for Isolation Forest

Isolation Forest, like any tree ensemble method, is built on the basis of decision trees. In these trees, partitions are created by first randomly selecting a feature and then selecting a random split value between the minimum and maximum value of the selected feature.

If we try to segregate a point which is obviously a non-outlier, it'll have many points in its round, so that it will be really difficult to isolate. On the other hand, if the point is an outlier, it'll be alone and we'll find it very easily.

Image for the Isolation Forest



Normal data x_i requires more partitions to separate from the rest of the data points. Anomalous data x_0 requires lesser number of partitions to separate from the rest of the data.

https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/icdm08b.pdf

Anomaly Score for Isolation Forest

As with other outlier detection methods, an anomaly score is required for decision making. In the case of Isolation Forest, it is defined as:

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

where *h(x)* is the path length of observation *x*, *c(n)* is the average path length of unsuccessful search in a Binary Search Tree and *n* is the number of external nodes.

Each observation is given an anomaly score and the following decision can be made on its basis:

- A score close to 1 indicates anomalies
- Score much smaller than 0.5 indicates normal observations
- If all scores are close to 0.5 then the entire sample does not seem to have clearly distinct anomalies

K Means

Kmeans algorithm is an iterative algorithm that tries to partition the dataset into *K*pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to **only one group**. It tries to make the inter-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum.

How does K Means help in our project?

- We wanted not just to detect abnormalities, but also **CLASSIFY** them into categories. Just detecting an outlier does not serve much of a purpose – we have to couple that with, what type of outlier it is, and which data set it differs from. This can help the user of our product realize more information about the abnormality, gain deeper insight, and in turn make their product less vulnerable.
- After attaining the anomalous data, we wanted to categorize them so we relied on the K Means categorizing algorithm, to cluster the data automatically.
- We have attached a documentation of our code (made using jupyter notebook on the next pages). The code can be run with the input files to generate the dataset.

Now that the theory has been completed, lets move to the code

Implementation

Anomaly_Detection_final (1)

September 2, 2019

1 Anomaly Detection System

We aim to build an anomaly detection system that relies on the Isolation Forest Algorithm to work on the dataset to mark the anomalous data points. The entire process follows an unsupervised learning approach with clever feature engineering to generate a more sophisticated system

1.0.1 Basic Files Upload

We upload the basic files and process the CSV data into a Pandas DataFrame. The entire process was done on the Google Colab platform, thus we used:

```
from google.colab import files
data = files.upload()
```

to upload it to the Online Jupyter Notebook

```
In [0]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
```

```
In [2]: from google.colab import files
        data = files.upload()
```

```
<IPython.core.display.HTML object>


Saving hackathon.csv to hackathon (2).csv
```

```
In [0]: import io
```

```
In [0]: df = pd.read_csv(io.StringIO(data['hackathon.csv'].decode('utf-8')))
```

1.0.2 Investigating the Data Provided

We begin with elementary data investigation to check the various data points and their data types.

```
In [5]: df
```

Out[5]:

	hour	date	users	event_count	event_name	day
0	0	2019-01-02	1	1	bBCA	0
1	0	2019-01-02	1	1	aO	0
2	0	2019-01-02	1	1	bBCD	0
3	0	2019-01-02	1	6	sD	0
4	0	2019-01-02	1	1	nO	14
5	0	2019-01-02	1	1	bST	0
6	0	2019-01-02	1	1	fL	0
7	0	2019-01-02	1	1	aO	14
8	0	2019-01-02	1	5	aS	0
9	1	2019-01-02	1	1	aO	7
10	1	2019-01-02	1	1	bBCA	0
11	1	2019-01-02	1	2	bST	0
12	1	2019-01-02	1	1	fL	7
13	1	2019-01-02	1	1	fL	0
14	1	2019-01-02	1	1	cFE	0
15	1	2019-01-02	2	2	aO	0
16	1	2019-01-02	1	1	aO	1
17	1	2019-01-02	1	1	aS	1
18	1	2019-01-02	1	1	sD	1
19	1	2019-01-02	1	1	fL	1
20	1	2019-01-02	1	1	aS	0
21	1	2019-01-02	1	1	sD	0
22	2	2019-01-02	2	14	sD	1
23	2	2019-01-02	1	1	bST	0
24	2	2019-01-02	2	2	bBCD	1
25	2	2019-01-02	5	10	bBCA	0
26	2	2019-01-02	1	2	aO	21
27	2	2019-01-02	1	2	aS	45
28	2	2019-01-02	1	2	nO	21
29	2	2019-01-02	1	2	sD	45
...	...	...	...	...	...	...
161179	23	2019-06-30	5	14	bBCA	0
161180	23	2019-06-30	1	3	fL	21
161181	23	2019-06-30	1	1	bST	14
161182	23	2019-06-30	1	1	aO	21
161183	23	2019-06-30	3	3	nR	45
161184	23	2019-06-30	5	15	nR	0
161185	23	2019-06-30	1	3	aS	14
161186	23	2019-06-30	1	6	fL	7
161187	23	2019-06-30	1	1	nR	90
161188	23	2019-06-30	1	2	sD	21
161189	23	2019-06-30	7	7	aO	0
161190	23	2019-06-30	1	1	bST	0
161191	23	2019-06-30	3	3	nR	1
161192	23	2019-06-30	1	1	bST	7
161193	23	2019-06-30	1	4	aS	7
161194	23	2019-06-30	1	2	fL	14

```
161195    23  2019-06-30      4            4        nR    7
161196    23  2019-06-30      6           10        fL    0
161197    23  2019-06-30      1            1      bBCD    7
161198    23  2019-06-30      3            4      bBCD    0
161199    23  2019-06-30      2            2      bBCD   21
161200    23  2019-06-30      1            1        aO    7
161201    23  2019-06-30      2            5        aS   21
161202    23  2019-06-30      2            5      bBCA   21
161203    23  2019-06-30      4            6        nR   14
161204    23  2019-06-30      1            1        sD    0
161205    23  2019-06-30      1            4        aS    0
161206    23  2019-06-30      1            1        sD   14
161207    23  2019-06-30      1            1      bBCA   14
161208    23  2019-06-30      1            2        nR   21

[161209 rows x 6 columns]
```

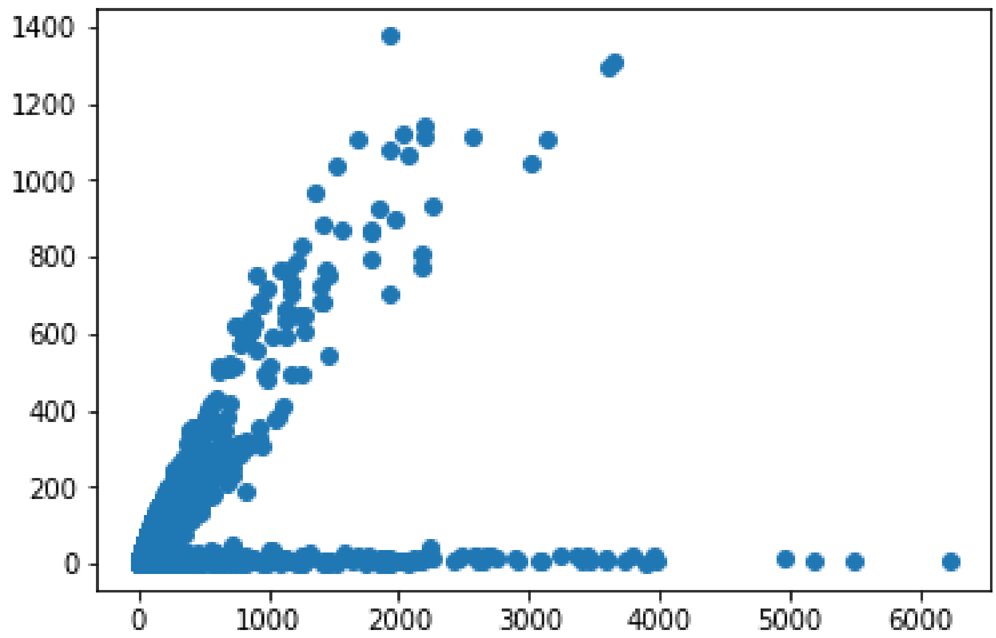
In [6]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 161209 entries, 0 to 161208
Data columns (total 6 columns):
hour           161209 non-null int64
date           161209 non-null object
users          161209 non-null int64
event_count    161209 non-null int64
event_name     161209 non-null object
day            161209 non-null int64
dtypes: int64(4), object(2)
memory usage: 7.4+ MB
None
```

In [0]:
```
df['date'] = pd.to_datetime(df['date'])
```

In [8]:
```
plt.scatter(df['event_count'], df['users'])
```

Out[8]:
```
<matplotlib.collections.PathCollection at 0x7f1f317cf9e8>
```



The above graph gives us a basic insight on to the nature of the outliers in the datapoints. The spread of the points along the $Y = X$ and X *axis* indicate that anomalous data may lie on along those points.

1.0.3 Feature Engineering

In this section, we try to create new distinguishable features from the given ones to provide the system with more valuable and usable datapoints to work on. For example: Here we have mapped each of the *event_names* to a particular number from **0** to **9**.

```
In [9]: print(df.event_name.unique())

['bBCA' 'aO' 'bBCD' 'sD' 'nO' 'bST' 'fL' 'aS' 'cFE' 'nR']


In [0]: a = df.event_name.unique()

In [0]: enum_event_name = {}
        for i in range(len(a)):
          enum_event_name[a[i]] = i

In [12]: print(enum_event_name)

{'bBCA': 0, 'aO': 1, 'bBCD': 2, 'sD': 3, 'nO': 4, 'bST': 5, 'fL': 6, 'aS': 7, 'cFE': 8, 'nR': 9}


In [0]: df['event_index'] = df['event_name'].map(enum_event_name)
```

Here we extract the features provided by the date column of the dataset.

```
In [0]: df['day_of_month'] = df['date'].dt.day
```

```
In [0]: df['month'] = df['date'].dt.month
```

Day of the Week : *0* for *Monday* --> *6* for *Sunday*.

```
In [0]: df['day_of_week'] = df['date'].dt.dayofweek
```

```
In [17]: df #Final Feature Set of the Data
```

```
Out[17]:         hour        date  users  ...  day_of_month  month  day_of_week
         0          0  2019-01-02      1  ...             2      1            2
         1          0  2019-01-02      1  ...             2      1            2
         2          0  2019-01-02      1  ...             2      1            2
         3          0  2019-01-02      1  ...             2      1            2
         4          0  2019-01-02      1  ...             2      1            2
         5          0  2019-01-02      1  ...             2      1            2
         6          0  2019-01-02      1  ...             2      1            2
         7          0  2019-01-02      1  ...             2      1            2
         8          0  2019-01-02      1  ...             2      1            2
         9          1  2019-01-02      1  ...             2      1            2
         10         1  2019-01-02      1  ...             2      1            2
         11         1  2019-01-02      1  ...             2      1            2
         12         1  2019-01-02      1  ...             2      1            2
         13         1  2019-01-02      1  ...             2      1            2
         14         1  2019-01-02      1  ...             2      1            2
         15         1  2019-01-02      2  ...             2      1            2
         16         1  2019-01-02      1  ...             2      1            2
         17         1  2019-01-02      1  ...             2      1            2
         18         1  2019-01-02      1  ...             2      1            2
         19         1  2019-01-02      1  ...             2      1            2
         20         1  2019-01-02      1  ...             2      1            2
         21         1  2019-01-02      1  ...             2      1            2
         22         2  2019-01-02      2  ...             2      1            2
         23         2  2019-01-02      1  ...             2      1            2
         24         2  2019-01-02      2  ...             2      1            2
         25         2  2019-01-02      5  ...             2      1            2
         26         2  2019-01-02      1  ...             2      1            2
         27         2  2019-01-02      1  ...             2      1            2
         28         2  2019-01-02      1  ...             2      1            2
         29         2  2019-01-02      1  ...             2      1            2
         ...      ...         ...    ...  ...           ...    ...          ...
         161179    23  2019-06-30      5  ...            30      6            6
         161180    23  2019-06-30      1  ...            30      6            6
         161181    23  2019-06-30      1  ...            30      6            6
         161182    23  2019-06-30      1  ...            30      6            6
         161183    23  2019-06-30      3  ...            30      6            6
```

```
161184    23 2019-06-30      5  ...            30      6            6
161185    23 2019-06-30      1  ...            30      6            6
161186    23 2019-06-30      1  ...            30      6            6
161187    23 2019-06-30      1  ...            30      6            6
161188    23 2019-06-30      1  ...            30      6            6
161189    23 2019-06-30      7  ...            30      6            6
161190    23 2019-06-30      1  ...            30      6            6
161191    23 2019-06-30      3  ...            30      6            6
161192    23 2019-06-30      1  ...            30      6            6
161193    23 2019-06-30      1  ...            30      6            6
161194    23 2019-06-30      1  ...            30      6            6
161195    23 2019-06-30      4  ...            30      6            6
161196    23 2019-06-30      6  ...            30      6            6
161197    23 2019-06-30      1  ...            30      6            6
161198    23 2019-06-30      3  ...            30      6            6
161199    23 2019-06-30      2  ...            30      6            6
161200    23 2019-06-30      1  ...            30      6            6
161201    23 2019-06-30      2  ...            30      6            6
161202    23 2019-06-30      2  ...            30      6            6
161203    23 2019-06-30      4  ...            30      6            6
161204    23 2019-06-30      1  ...            30      6            6
161205    23 2019-06-30      1  ...            30      6            6
161206    23 2019-06-30      1  ...            30      6            6
161207    23 2019-06-30      1  ...            30      6            6
161208    23 2019-06-30      1  ...            30      6            6

[161209 rows x 10 columns]
```

In [18]:
```python
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 161209 entries, 0 to 161208
Data columns (total 10 columns):
hour            161209 non-null int64
date            161209 non-null datetime64[ns]
users           161209 non-null int64
event_count     161209 non-null int64
event_name      161209 non-null object
day             161209 non-null int64
event_index     161209 non-null int64
day_of_month    161209 non-null int64
month           161209 non-null int64
day_of_week     161209 non-null int64
dtypes: datetime64[ns](1), int64(8), object(1)
memory usage: 12.3+ MB
None
```

In [0]:
```python
from sklearn import preprocessing
```

```
        from sklearn.ensemble import IsolationForest
        from sklearn.decomposition import PCA
```

```
In [0]: features = df[['hour', 'users', 'event_count', 'event_index', 'day', 'day_of_month', 'da
```

1.0.4 Principle Component Analysis (Dimensionality Reduction) for 2 components

We apply Principle Component Analysis to reduce the data points from 8 components to 2 components. This is not only useful for training the system but also helps in visualizing the outcomes.

```
In [21]: pca = PCA(n_components = 2)
         pca_features = pca.fit_transform(features)
         print(pca_features[:,0])
         print(pca_features[:,1])
```

```
[-19.33216276 -19.32604561 -19.31992847 ... -19.7492708  -19.76762224
 -18.94947076]
[-12.5068623  -12.5031032  -12.4993441  ...   0.41395476   0.40267747
   6.97134647]
```

```
In [0]: df['pca_feature_1'] = pca_features[:,0]
        df['pca_feature_2'] = pca_features[:,1]

        features_w_pca = df[['hour', 'users', 'event_count', 'event_index', 'day', 'day_of_month
```

```
In [0]: min_max_scaler = preprocessing.StandardScaler()
        np_scaled = min_max_scaler.fit_transform(features_w_pca)
        features_w_pca = pd.DataFrame(np_scaled)
```

```
In [0]: outliers_fraction = 0.01 #Estimation of the fraction of the outliers
```

```
In [25]: model = IsolationForest(contamination = outliers_fraction)
         model.fit(features_w_pca)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/iforest.py:247: FutureWarning: behaviour
  FutureWarning)
```

```
Out[25]: IsolationForest(behaviour='old', bootstrap=False, contamination=0.01,
                         max_features=1.0, max_samples='auto', n_estimators=100,
                         n_jobs=None, random_state=None, verbose=0, warm_start=False)
```

```
In [26]: features_w_pca['anomaly'] = pd.Series(model.predict(features_w_pca))
         features_w_pca['anomaly'] = features_w_pca['anomaly'].map({1:0, -1:1})
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/iforest.py:415: DeprecationWarning: thre
  " be removed in 0.22.", DeprecationWarning)
```

```
In [27]: print(features_w_pca['anomaly'].value_counts()) # Output 1
```

```
0    159596
1      1613
Name: anomaly, dtype: int64
```

```
In [28]: features_w_pca
```

Out[28]:

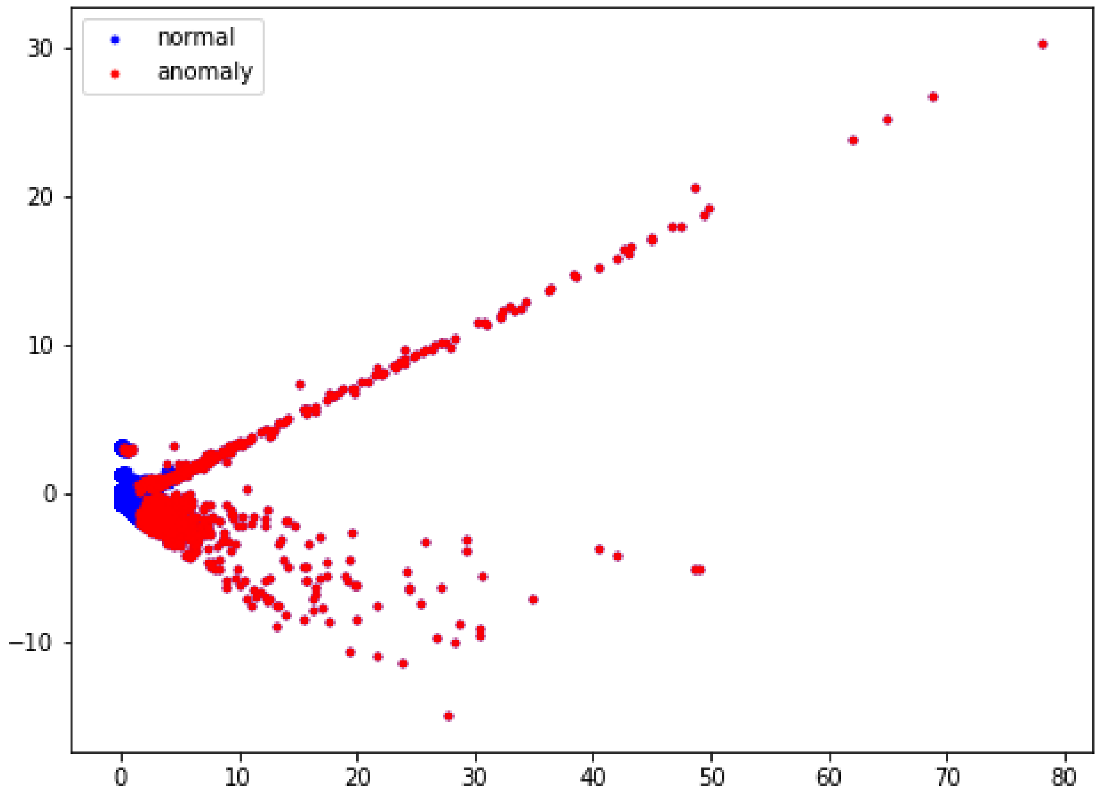
	0	1	2	...	8	9	anomaly
0	-1.685574	-0.321686	-0.242535	...	-0.248640	-0.571846	0
1	-1.685574	-0.321686	-0.242535	...	-0.248562	-0.571674	0
2	-1.685574	-0.321686	-0.242535	...	-0.248483	-0.571502	0
3	-1.685574	-0.321686	-0.176886	...	-0.185511	-0.546384	0
4	-1.685574	-0.321686	-0.242535	...	-0.253854	0.016442	0
5	-1.685574	-0.321686	-0.242535	...	-0.248247	-0.570987	0
6	-1.685574	-0.321686	-0.242535	...	-0.248168	-0.570815	0
7	-1.685574	-0.321686	-0.242535	...	-0.254090	0.015926	0
8	-1.685574	-0.321686	-0.190016	...	-0.197775	-0.550685	0
9	-1.524884	-0.321686	-0.242535	...	-0.251353	-0.278238	0
10	-1.524884	-0.321686	-0.242535	...	-0.248668	-0.572210	0
11	-1.524884	-0.321686	-0.229405	...	-0.235696	-0.566361	0
12	-1.524884	-0.321686	-0.242535	...	-0.250960	-0.277379	0
13	-1.524884	-0.321686	-0.242535	...	-0.248195	-0.571179	0
14	-1.524884	-0.321686	-0.242535	...	-0.248038	-0.570835	0
15	-1.524884	-0.282531	-0.229405	...	-0.233358	-0.584477	0
16	-1.524884	-0.321686	-0.242535	...	-0.248984	-0.530067	0
17	-1.524884	-0.321686	-0.242535	...	-0.248512	-0.529036	0
18	-1.524884	-0.321686	-0.242535	...	-0.248826	-0.529723	0
19	-1.524884	-0.321686	-0.242535	...	-0.248590	-0.529207	0
20	-1.524884	-0.321686	-0.242535	...	-0.248117	-0.571007	0
21	-1.524884	-0.321686	-0.242535	...	-0.248431	-0.571695	0
22	-1.364194	-0.282531	-0.071847	...	-0.082680	-0.482653	0
23	-1.364194	-0.321686	-0.242535	...	-0.248301	-0.571715	0
24	-1.364194	-0.282531	-0.229405	...	-0.233702	-0.542697	0
25	-1.364194	-0.165065	-0.124366	...	-0.124879	-0.597381	0
26	-1.364194	-0.321686	-0.229405	...	-0.244330	0.313988	0
27	-1.364194	-0.321686	-0.229405	...	-0.253334	1.322335	0
28	-1.364194	-0.321686	-0.229405	...	-0.244094	0.314504	0
29	-1.364194	-0.321686	-0.229405	...	-0.253649	1.321647	0
...	...	...	...	...	...	...	...
161179	2.010299	-0.165065	-0.071847	...	-0.074583	-0.574038	0
161180	2.010299	-0.321686	-0.216276	...	-0.231376	0.323222	0
161181	2.010299	-0.321686	-0.242535	...	-0.253848	0.019271	0
161182	2.010299	-0.321686	-0.242535	...	-0.256926	0.312384	0
161183	2.010299	-0.243376	-0.216276	...	-0.235312	1.296197	0
161184	2.010299	-0.165065	-0.058717	...	-0.061296	-0.567502	0
161185	2.010299	-0.321686	-0.216276	...	-0.228533	0.029593	0

```
161186  2.010299 -0.321686 -0.176886  ... -0.188112 -0.249411        0
161187  2.010299 -0.321686 -0.242535  ... -0.283543  3.209790        0
161188  2.010299 -0.321686 -0.229405  ... -0.244190  0.317717        0
161189  2.010299 -0.086754 -0.163756  ... -0.157251 -0.643648        0
161190  2.010299 -0.321686 -0.242535  ... -0.248320 -0.568330        0
161191  2.010299 -0.243376 -0.216276  ... -0.217938 -0.550548        0
161192  2.010299 -0.321686 -0.242535  ... -0.251084 -0.274529        0
161193  2.010299 -0.321686 -0.203146  ... -0.213190 -0.259218        0
161194  2.010299 -0.321686 -0.229405  ... -0.241190  0.024432        0
161195  2.010299 -0.204220 -0.203146  ... -0.205077 -0.311157        0
161196  2.010299 -0.125910 -0.124366  ... -0.121773 -0.610393        0
161197  2.010299 -0.321686 -0.242535  ... -0.251320 -0.275045        0
161198  2.010299 -0.243376 -0.203146  ... -0.205516 -0.588733        0
161199  2.010299 -0.282531 -0.229405  ... -0.241617  0.300117        0
161200  2.010299 -0.321686 -0.242535  ... -0.251398 -0.275217        0
161201  2.010299 -0.282531 -0.190016  ... -0.203488  0.315945        0
161202  2.010299 -0.282531 -0.190016  ... -0.204039  0.314742        0
161203  2.010299 -0.204220 -0.176886  ... -0.182684 -0.007378        0
161204  2.010299 -0.321686 -0.242535  ... -0.248477 -0.568674        0
161205  2.010299 -0.321686 -0.203146  ... -0.210426 -0.553018        0
161206  2.010299 -0.321686 -0.242535  ... -0.254005  0.018927        0
161207  2.010299 -0.321686 -0.242535  ... -0.254241  0.018411        0
161208  2.010299 -0.321686 -0.229405  ... -0.243718  0.318748        0

[161209 rows x 11 columns]
```

In [0]:
```
a = features_w_pca.loc[features_w_pca['anomaly'] == 1, [8,9]]
```

In [30]:
```
plt.figure(figsize=(8,6))
plt.scatter(features_w_pca[8],features_w_pca[9], color='blue', s = 8,label='normal')
plt.scatter(a[8],a[9],color='red', s = 8,label='anomaly')
plt.legend()
```

Out[30]:
```
<matplotlib.legend.Legend at 0x7f1f20f69898>
```



1.0.5 Principle Component Analysis for 3 components

Here we do the same thing as in the previous section; however, we increase the PCA components to 3 which would provide more sophistication to the system thus allowing it to make more complex relatins. At the same time, making the final components to 3 allows us to still visualize the data.

```
In [0]: pca_3 = PCA(n_components = 3)
        pca_3_features = pca_3.fit_transform(features)
```

```
In [0]: df['pca_feature_1'] = pca_3_features[:,0]
        df['pca_feature_2'] = pca_3_features[:,1]
        df['pca_feature_3'] = pca_3_features[:,2]

        features_w_pca_3 = df[['hour', 'users', 'event_count', 'event_index', 'day', 'day_of_mon
```

```
In [0]: np_scaled_3 = min_max_scaler.fit_transform(features_w_pca_3)
        features_w_pca_3 = pd.DataFrame(np_scaled_3)
```

```
In [34]: model_3 = IsolationForest(contamination = outliers_fraction)
         model_3.fit(features_w_pca_3)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/iforest.py:247: FutureWarning: behaviour
  FutureWarning)
```

```
Out[34]: IsolationForest(behaviour='old', bootstrap=False, contamination=0.01,
                         max_features=1.0, max_samples='auto', n_estimators=100,
                         n_jobs=None, random_state=None, verbose=0, warm_start=False)
```

```
In [35]: features_w_pca_3['anomaly'] = pd.Series(model_3.predict(features_w_pca_3))
         features_w_pca_3['anomaly'] = features_w_pca_3['anomaly'].map({1:0, -1:1})
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/iforest.py:415: DeprecationWarning: thre
  " be removed in 0.22.", DeprecationWarning)
```

```
In [36]: print(features_w_pca_3['anomaly'].value_counts()) #Output 2
```

```
0    159596
1      1613
Name: anomaly, dtype: int64
```

```
In [37]: features_w_pca_3
```

Out[37]:

	0	1	2	...	9	10	anomaly
0	-1.685574	-0.321686	-0.242535	...	-0.571846	0.512446	0
1	-1.685574	-0.321686	-0.242535	...	-0.571674	0.511197	0
2	-1.685574	-0.321686	-0.242535	...	-0.571502	0.509947	0
3	-1.685574	-0.321686	-0.176886	...	-0.546384	0.553049	0
4	-1.685574	-0.321686	-0.242535	...	0.016442	0.231650	0
5	-1.685574	-0.321686	-0.242535	...	-0.570987	0.506199	0
6	-1.685574	-0.321686	-0.242535	...	-0.570815	0.504949	0
7	-1.685574	-0.321686	-0.242535	...	0.015926	0.235398	0
8	-1.685574	-0.321686	-0.190016	...	-0.550685	0.539181	0
9	-1.524884	-0.321686	-0.242535	...	-0.278238	0.373800	0
10	-1.524884	-0.321686	-0.242535	...	-0.572210	0.512949	0
11	-1.524884	-0.321686	-0.229405	...	-0.566361	0.515572	0
12	-1.524884	-0.321686	-0.242535	...	-0.277379	0.367553	0
13	-1.524884	-0.321686	-0.242535	...	-0.571179	0.505452	0
14	-1.524884	-0.321686	-0.242535	...	-0.570835	0.502953	0
15	-1.524884	-0.282531	-0.229405	...	-0.584477	0.475616	0
16	-1.524884	-0.321686	-0.242535	...	-0.530067	0.492000	0
17	-1.524884	-0.321686	-0.242535	...	-0.529036	0.484503	0
18	-1.524884	-0.321686	-0.242535	...	-0.529723	0.489501	0
19	-1.524884	-0.321686	-0.242535	...	-0.529207	0.485752	0
20	-1.524884	-0.321686	-0.242535	...	-0.571007	0.504203	0
21	-1.524884	-0.321686	-0.242535	...	-0.571695	0.509201	0
22	-1.364194	-0.282531	-0.071847	...	-0.482653	0.560362	0
23	-1.364194	-0.321686	-0.242535	...	-0.571715	0.507205	0

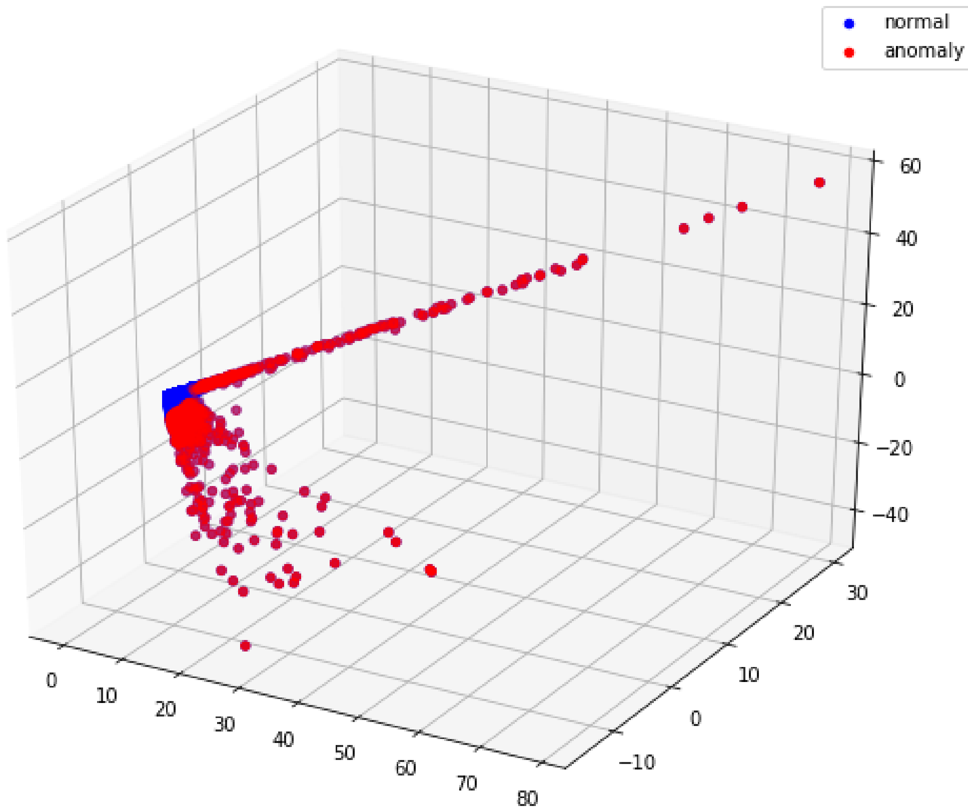
```
24      -1.364194 -0.282531 -0.229405  ... -0.542697  0.455170        0
25      -1.364194 -0.165065 -0.124366  ... -0.597381  0.413469        0
26      -1.364194 -0.321686 -0.229405  ...  0.313988  0.107375        0
27      -1.364194 -0.321686 -0.229405  ...  1.322335 -0.372920        0
28      -1.364194 -0.321686 -0.229405  ...  0.314504  0.103626        0
29      -1.364194 -0.321686 -0.229405  ...  1.321647 -0.367922        0
...           ...       ...       ...  ...       ...       ...      ...
161179   2.010299 -0.165065 -0.071847  ... -0.574038  0.428548        0
161180   2.010299 -0.321686 -0.216276  ...  0.323222  0.089596        0
161181   2.010299 -0.321686 -0.242535  ...  0.019271  0.211005        0
161182   2.010299 -0.321686 -0.242535  ...  0.312384  0.078103        0
161183   2.010299 -0.243376 -0.216276  ...  1.296197 -0.476858        0
161184   2.010299 -0.165065 -0.058717  ... -0.567502  0.426173        0
161185   2.010299 -0.321686 -0.216276  ...  0.029593  0.226246        0
161186   2.010299 -0.321686 -0.176886  ... -0.249411  0.392005        0
161187   2.010299 -0.321686 -0.242535  ...  3.209790 -1.291186        0
161188   2.010299 -0.321686 -0.229405  ...  0.317717  0.084475        0
161189   2.010299 -0.086754 -0.163756  ... -0.643648  0.275300        0
161190   2.010299 -0.321686 -0.242535  ... -0.568330  0.486803        0
161191   2.010299 -0.243376 -0.216276  ... -0.550548  0.389938        0
161192   2.010299 -0.321686 -0.242535  ... -0.274529  0.348904        0
161193   2.010299 -0.321686 -0.203146  ... -0.259218  0.373016        0
161194   2.010299 -0.321686 -0.229405  ...  0.024432  0.218625        0
161195   2.010299 -0.204220 -0.203146  ... -0.311157  0.235655        0
161196   2.010299 -0.125910 -0.124366  ... -0.610393  0.340617        0
161197   2.010299 -0.321686 -0.242535  ... -0.275045  0.352653        0
161198   2.010299 -0.243376 -0.203146  ... -0.588733  0.427255        0
161199   2.010299 -0.282531 -0.229405  ...  0.300117  0.040770        0
161200   2.010299 -0.321686 -0.242535  ... -0.275217  0.353902        0
161201   2.010299 -0.282531 -0.190016  ...  0.315945  0.061133        0
161202   2.010299 -0.282531 -0.190016  ...  0.314742  0.069880        0
161203   2.010299 -0.204220 -0.176886  ... -0.007378  0.115496        0
161204   2.010299 -0.321686 -0.242535  ... -0.568674  0.489302        0
161205   2.010299 -0.321686 -0.203146  ... -0.553018  0.510915        0
161206   2.010299 -0.321686 -0.242535  ...  0.018927  0.213504        0
161207   2.010299 -0.321686 -0.242535  ...  0.018411  0.217252        0
161208   2.010299 -0.321686 -0.229405  ...  0.318748  0.076978        0

[161209 rows x 12 columns]
```

In [0]:
```python
a = features_w_pca_3.loc[features_w_pca_3['anomaly'] == 1, [8,9,10]]
```

In [0]:
```python
from mpl_toolkits.mplot3d import Axes3D
```

In [40]:
```python
fig = plt.figure(figsize=(8,6))
ax = Axes3D(fig)
ax.scatter(features_w_pca_3[8],features_w_pca_3[9], features_w_pca_3[10], color='blue',
ax.scatter(a[8],a[9],a[10],color='red',label='anomaly')
```

```
ax.legend()
plt.show()
```



1.0.6 PCA without normalizing

We realize that some datatypes may not be normalizable, thus we allow the system to train on the discrete values rather than the normalized real values.

```
In [0]: pca_3 = PCA(n_components = 3)
        pca_3_features = pca_3.fit_transform(features)
```

```
In [0]: df['pca_feature_1'] = pca_3_features[:,0]
        df['pca_feature_2'] = pca_3_features[:,1]
        df['pca_feature_3'] = pca_3_features[:,2]

        features_w_pca_3_nnorm = df[['hour', 'users', 'event_count', 'event_index', 'day', 'day_
```

```
In [43]: model_3_nnorm = IsolationForest(contamination = outliers_fraction)
         model_3_nnorm.fit(features_w_pca_3_nnorm)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/iforest.py:247: FutureWarning: behaviour
  FutureWarning)
```

```
Out[43]: IsolationForest(behaviour='old', bootstrap=False, contamination=0.01,
                         max_features=1.0, max_samples='auto', n_estimators=100,
                         n_jobs=None, random_state=None, verbose=0, warm_start=False)
```

```
In [44]: features_w_pca_3_nnorm['anomaly'] = pd.Series(model_3_nnorm.predict(features_w_pca_3_nn
         features_w_pca_3_nnorm['anomaly'] = features_w_pca_3_nnorm['anomaly'].map({1:0, -1:1})
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/iforest.py:415: DeprecationWarning: thre
  " be removed in 0.22.", DeprecationWarning)
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
```

```
In [45]: print(features_w_pca_3_nnorm['anomaly'].value_counts()) #Output 3
```

```
0    159596
1      1613
Name: anomaly, dtype: int64
```

```
In [46]: features_w_pca_3_nnorm
```

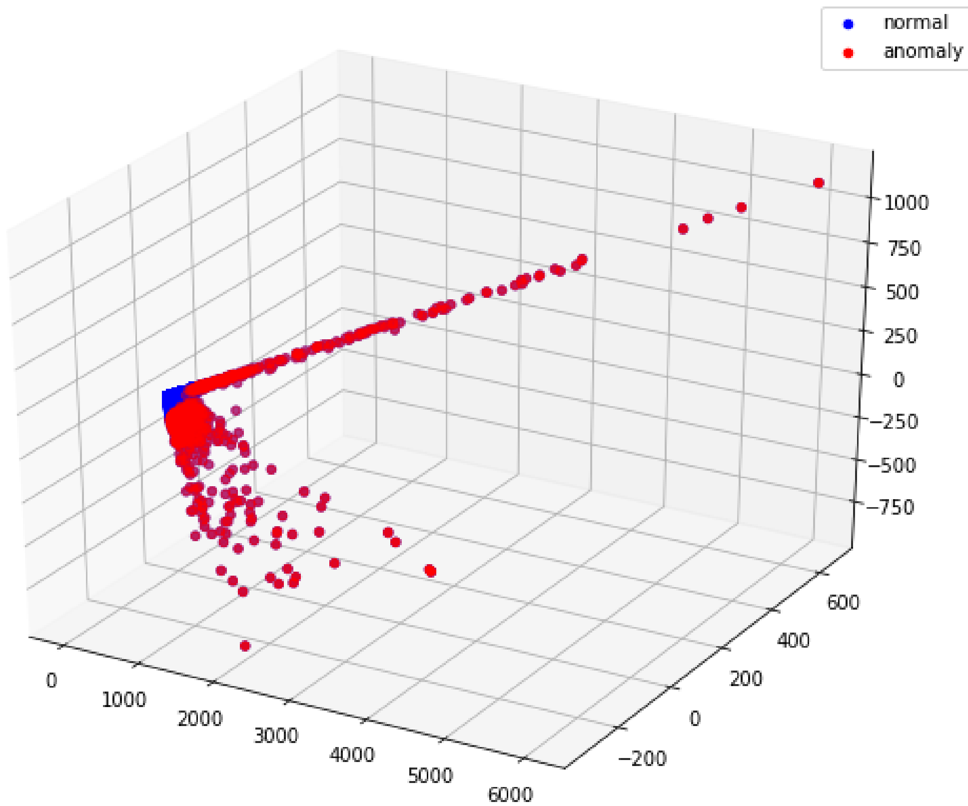
Out[46]:

	hour	users	event_count	...	pca_feature_2	pca_feature_3	anomaly
0	0	1	1	...	-12.506862	10.267715	0
1	0	1	1	...	-12.503103	10.242679	0
2	0	1	1	...	-12.499344	10.217644	0
3	0	1	6	...	-11.949968	11.081250	0
4	0	1	1	...	0.359602	4.641487	0
5	0	1	1	...	-12.488067	10.142537	0
6	0	1	1	...	-12.484308	10.117501	0
7	0	1	1	...	0.348324	4.716594	0
8	0	1	5	...	-12.044055	10.803379	0
9	1	1	1	...	-6.085351	7.489713	0
10	1	1	1	...	-12.514823	10.277792	0
11	1	1	2	...	-12.386905	10.330342	0
12	1	1	1	...	-6.066555	7.364535	0
13	1	1	1	...	-12.492269	10.127577	0
14	1	1	1	...	-12.484751	10.077506	0
15	1	2	2	...	-12.783106	9.529762	0

```
16         1      1         1  ...  -11.593105   9.858036    0
17         1      1         1  ...  -11.570551   9.707821    0
18         1      1         1  ...  -11.585587   9.807964    0
19         1      1         1  ...  -11.574310   9.732857    0
20         1      1         1  ...  -12.488510  10.102542    0
21         1      1         1  ...  -12.503546  10.202685    0
22         2      2        14  ...  -10.556110  11.227787    0
23         2      1         1  ...  -12.503989  10.162690    0
24         2      2         2  ...  -11.869349   9.120082    0
25         2      5        10  ...  -13.065334   8.284533    0
26         2      1         2  ...    6.867239   2.151432    0
27         2      1         2  ...   28.920812  -7.472072    0
28         2      1         2  ...    6.878516   2.076325    0
29         2      1         2  ...   28.905776  -7.371929    0
...      ...    ...       ...  ...         ...        ...  ...
161179    23      5        14  ...  -12.554806   8.586674    0
161180    23      1         3  ...    7.069193   1.795209    0
161181    23      1         1  ...    0.421473   4.227831    0
161182    23      1         1  ...    6.832150   1.564931    0
161183    23      3         3  ...   28.349159  -9.554633    0
161184    23      5        15  ...  -12.411851   8.539081    0
161185    23      1         3  ...    0.647238   4.533216    0
161186    23      1         6  ...   -5.454865   7.854480    0
161187    23      1         1  ...   70.201402 -25.871063    0
161188    23      1         2  ...    6.948792   1.692588    0
161189    23      7         7  ...  -14.077240   5.516094    0
161190    23      1         1  ...  -12.429955   9.753917    0
161191    23      3         3  ...  -12.041042   7.813065    0
161192    23      1         1  ...   -6.004241   6.990874    0
161193    23      1         4  ...   -5.669353   7.473988    0
161194    23      1         2  ...    0.534355   4.380524    0
161195    23      4         4  ...   -6.805329   4.721748    0
161196    23      6        10  ...  -13.349910   6.824824    0
161197    23      1         1  ...   -6.015518   7.065981    0
161198    23      3         4  ...  -12.876192   8.560764    0
161199    23      2         2  ...    6.563868   0.816901    0
161200    23      1         1  ...   -6.019277   7.091017    0
161201    23      2         5  ...    6.910033   1.224908    0
161202    23      2         5  ...    6.883720   1.400158    0
161203    23      4         6  ...   -0.161369   2.314162    0
161204    23      1         1  ...  -12.437473   9.803988    0
161205    23      1         4  ...  -12.095066  10.237031    0
161206    23      1         1  ...    0.413955   4.277902    0
161207    23      1         1  ...    0.402677   4.353010    0
161208    23      1         2  ...    6.971346   1.542374    0

[161209 rows x 12 columns]
```

```
In [0]: a = features_w_pca_3_nnorm.loc[features_w_pca_3_nnorm['anomaly'] == 1, ['pca_feature_1',
```

```
In [48]: fig = plt.figure(figsize=(8,6))
         ax = Axes3D(fig)
         ax.scatter(features_w_pca_3_nnorm['pca_feature_1'],features_w_pca_3_nnorm['pca_feature_
         ax.scatter(a['pca_feature_1'],a['pca_feature_2'],a['pca_feature_3'],color='red', label=
         ax.legend()
         plt.show()
```



1.0.7 K-Means Clustering

We apply K means to cluster the anomalies. We detect the anomalies first then we attempt to use **KMeans Algorithm** to cluster anomalies with similar causes.

```
In [0]: from sklearn.cluster import KMeans
```

```
In [50]: kmeans = KMeans(n_clusters = 4)
         kmeans.fit_transform(a)
```

```
Out[50]: array([[ 134.04096111,  756.87716047, 3604.37684901, 1837.4324616 ],
               [ 182.09637119,  668.56357795, 3506.43401282, 1740.71026207],
```

```
[ 463.0023877 ,  375.99895616, 3177.50574133, 1410.4891043 ],
...,
[  76.54851089,  792.17758099, 3692.71168867, 1907.81396063],
[ 127.91138235,  868.42393039, 3757.43697985, 1977.44775559],
[ 120.89616218,  832.93917838, 3697.0431834 , 1925.72384671]])
```

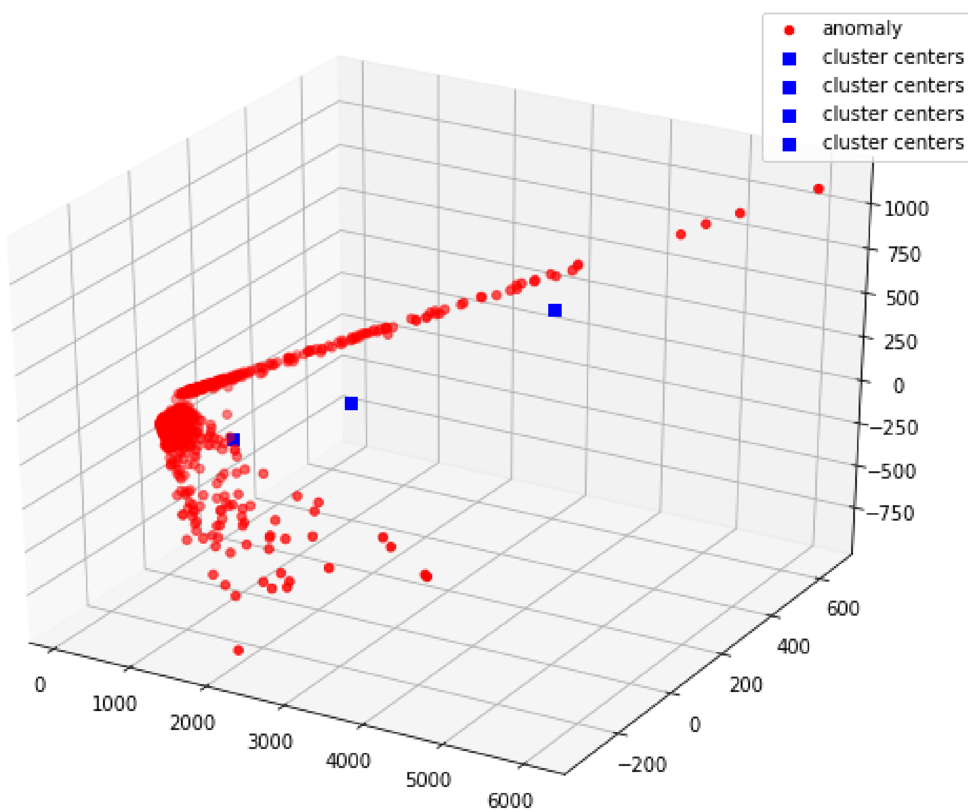
```
In [51]: kmeans.cluster_centers_
```

```
Out[51]: array([[ 266.62241283,  -27.20511812,  -62.82751357],
                [1006.61109242,  -24.59656092, -106.41794692],
                [3828.76862916,  344.2634391 ,  534.78702763],
                [2105.81849916,   95.43026134,   93.57376331]])
```

We used 4 means to cluster the anomalies to 4 different categories of anomalies. The number of the clusters may be increased as per the requirements.

```
In [53]: fig2 = plt.figure(figsize = (8,6))
         ax = Axes3D(fig2)
         ax.scatter(a['pca_feature_1'],a['pca_feature_2'],a['pca_feature_3'],color='red', label=
         for i in kmeans.cluster_centers_:
           ax.scatter(i[0],i[1],i[2], marker = 's', s = 30, color = 'blue', label='cluster cente
         ax.legend()
```

```
Out[53]: <matplotlib.legend.Legend at 0x7f1f199c0f60>
```



1.0.8 Program to Send Email

The code for this section is in the PPT rather than here for the brevity of this file. We use *SMTP to send emails*. The email would be sent to the person in charge through the code given in the PPT **when an anomaly is detected** and *may* contain the following details: 1. K-Mean Clustering Group 2. Time Series KPI Data 3. Similar errors in the past

1.0.9 Real Time Anomaly Checking

The code below demonstrates roughly the pipeline for the data to go through to give the result as anomalous or not.

```
In [ ]: a = input()
        inp = pd.read_csv()
        inp['date'] = inp.to_datetime(inp['date'])
        inp['event_index'] = inp['event_name'].map(enum_event_name)
        inp['day_of_month'] = inp['date'].dt.day
        inp['month'] = inp['date'].dt.month
        inp['day_of_week'] = inp['date'].dt.dayofweek

        inp_features = inp[['hour', 'users', 'event_count', 'event_index', 'day', 'day_of_month'

        pca_3_inp_features = pca_3.fit_transform(inp_features)
        inp['pca_feature_1'] = pca_3_inp_features[:,0]
        inp['pca_feature_2'] = pca_3_inp_features[:,1]
        inp['pca_feature_3'] = pca_3_inp_features[:,2]


        output = model_3_nnorm.predict(inp)
        if output == 1:
          pass
        else:
          send_email(error_id_from_KMeans, some_error) # Send Email to the person-in-charge in c
```

1.0.10 Conclusion

In the 3 models created:

- **2 PCA components**: It gave **1613** anomalous data points.
- **3 PCA components**: It gave also **1613** anomalous data points, which was expected and verified the construction of the system.
- **3 PCA components without normalization**: Since some data points are not normalizable, not normalizing them makes more sense. But we observe that that makes *no difference* to the output of the system. We again get **1613** anomalous points.

Combining this with the initial insight from the graph we can say that the model has successfully pointed out the anomalous data points from the data set.

Also we used:

```
outliers_fraction = 0.01
```

However, this may be changed through the use of a supervised learning algorithm on a small set of known data or using K-Means on a labelled set.

Using the **clustering algorithms** we were able to group the anomalies to *4* different types.

Idea Flow

1. We first investigated the data and analyze to develop basic insights of the data.
2. We then engineered features to make the system more sophisticated and provide it with more non-trivial data points.
3. We then applied PCA (2 components and 3 components) to allow us to visualize the data and to provide for dimensionality reduction.
4. We applied the *Isolation Algorithm* on the data to then find out the anomalies present in the data.
5. After getting the anomalies, we used the *K-Means Clustering Algorithm* to cluster the similsr data and provide categories for the anomalous data.

Issuing alert upon detection of abnormality

After anomaly detection, we have to alert the business owner that there has been some unusual activity. For this, we have written a simple Python script to alert via email using SMTP library.

The code given on the next page requires username and password to be passed as parameters to the function call. This must be replaced for execution.

The SMTP library configures the sender for a particular port and host. The code is pretty self-explanatory for the most part

```
import smtplib
from string import Template
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText


def send_email(id, details):
     message_template = "Abnormal activity has been found
related to account ${ID}. The details of the event are
${DETAIL}"
     s = smtplib.SMTP(host='smtp-mail.outlook.com', port=587)
     s.starttls()
     s.login(MY_ADDRESS, PASSWORD)
#These variables get the value of email address and password of
the account from which alert has to be issues. We have not
included our own, due to privacy's sake.


msg = MIMEMultipart()
message = message_template.substitute(ID = id, DETAIL =
details)
#Gives specific details about the anomaly


msg['From']=MY_ADDRESS
msg['To']=email #Email address to which to send email
msg['Subject']="Unusual activity detected"


msg.attach(MIMEText(message, 'plain'))


s.send_message(msg)
del msg
s.quit()
```

With that, our anomaly detection model is ready. You can use this on your business model, it will detect unusual activity and alert you via email.

We have attached the code alongside, and you can test it on different datasets. Our outputs for the given dataset have been included in the previous slides

Thanks