API MISUSE OR ABUSE

SOLUTION PROTOTYPE

Team : Ctrl Alt Delete

DATA PREPROCESSING

Step 1: Parsing the access log file

The general format of the access log data we used was:

[IP] - - [Timestamp] [URL] [Status Code] [Return Size] [Referer] [User Agent]

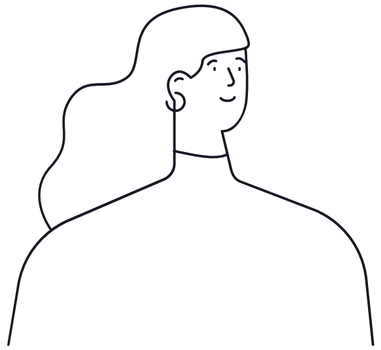
Hence we used Python **RegEx** to parse each line of the log file and create an initial DataFrame

```
HTTP/1.1" 200 203023 "http://sem
monitorama-2013/" "Mozilla/5.0 (
AppleWebKit/537.36 (KHTML, like
Safari/537.36"
83.149.9.216 - - [17/May/2015:10
/presentations/logstash-monitora
HTTP/1.1" 200 171717 "http://sem
monitorama-2013/" "Mozilla/5.0 (
AppleWebKit/537.36 (KHTML, like
Safari/537.36"
83.149.9.216 - - [17/May/2015:10
/presentations/logstash-monitora
HTTP/1.1" 200 26185 "http://semi
monitorama-2013/" "Mozilla/5.0 (
```

Step 2: Feature Extraction:

The following features were extracted from the parsed dataframe:

1. Request success label (Success codes are of the form 2xx)
2. GET request label from Request URL
3. Date of request from Timestamp



FEATURE ENGINEERING

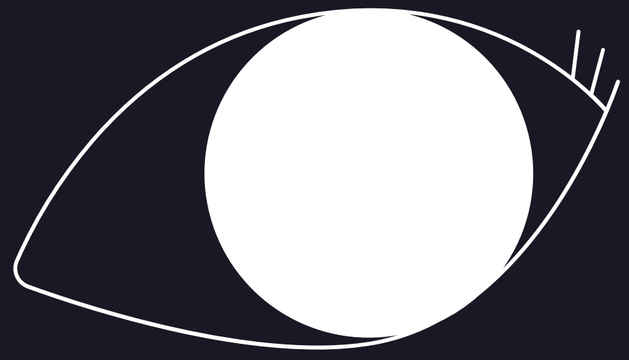
The following features were produced using the data extracted from the log file:

1. Time interval between two successive requests per IP address - Very small time difference should indicate anomalous request ideally.
2. Mean number of requests per day per IP address.
3. Total number of successful requests made by each IP address.
4. Total number of GET requests per IP address.
5. Standard deviation of request return size per IP address.

Python Libraries used for this purpose:

1. Numpy
2. Pandas

	IP	Total Requests	Daily Mean	GET requests	Successful requests	Mean Return Size	Return Size Std	Mean Time Difference
0	1.22.35.226	6	6.0	6	6	1.338050e+04	1.943990e+04	7.000000
1	100.2.4.116	6	3.0	6	6	1.811173e+07	2.803655e+07	1201.666667
2	100.43.83.137	84	21.0	84	56	1.505974e+04	1.469133e+04	2485.226190
3	101.119.18.35	33	33.0	33	32	7.144242e+04	1.136064e+05	1.696970

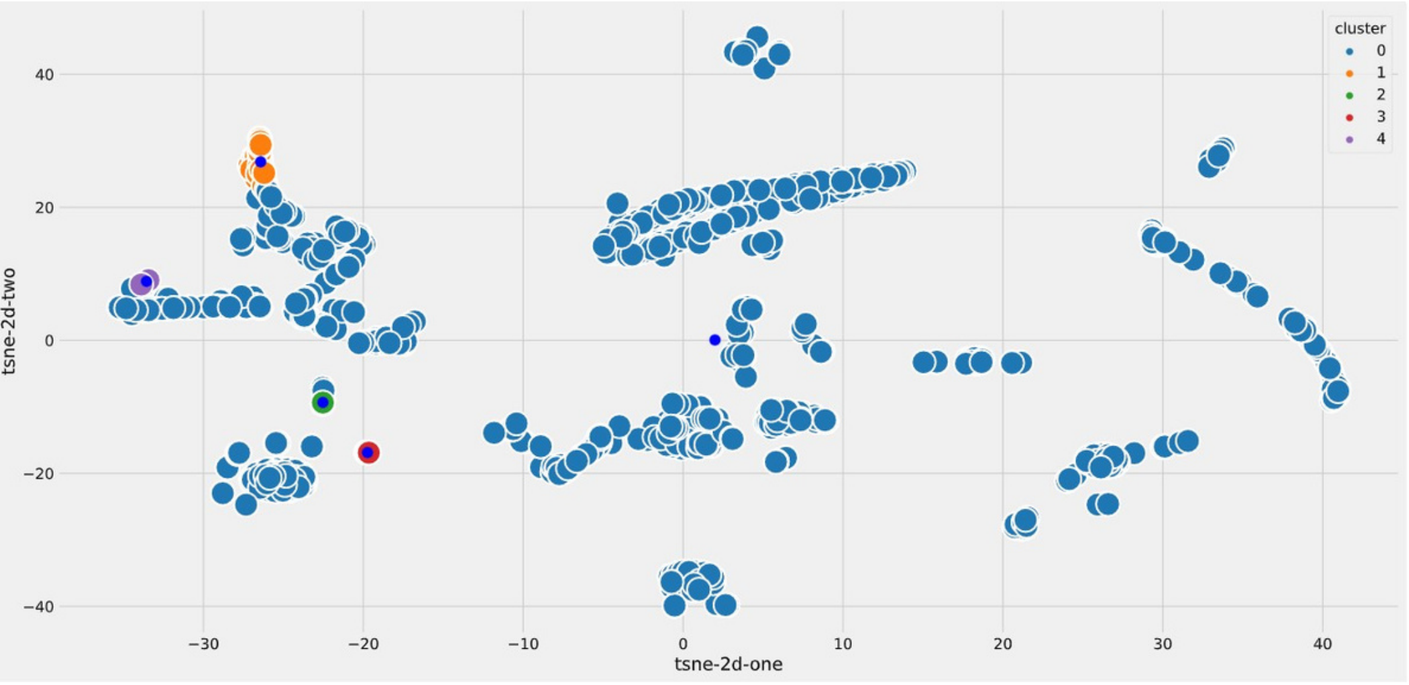


UNSUPERVISED ML ALGORITHMS USED FOR IMPLEMENTATION

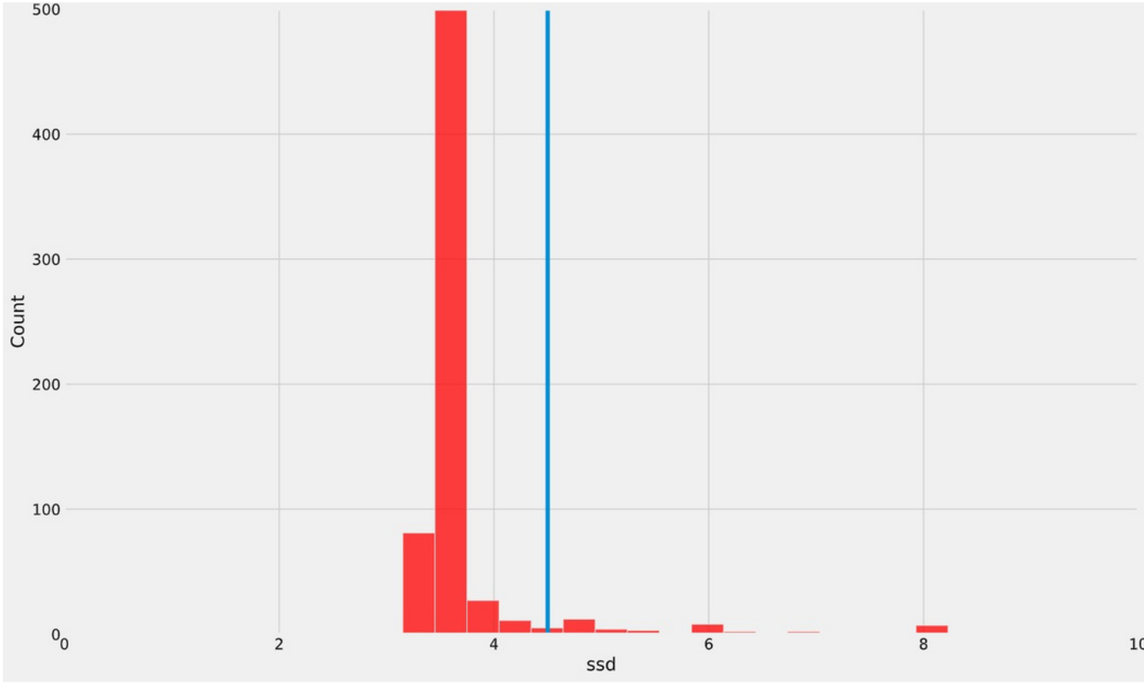
K-MEANS CLUSTERING

K-Means is an **unsupervised learning algorithm**. This algorithm looks for a fixed number of clusters in the given dataset. Clusters are just collection of data points that aggregate together due to some similarities between them. We define 'k' centroids, which are the centers of the clusters and each of the data points are allocated to these centroids. First some random centroids are chosen and then iterations are performed to optimize the position of the centroids. The creation and optimization of centroids stops when either of the two given conditions is achieved: **centroids are stabilized or the no. of iterations defined has been achieved.**

In the final solution, **'0' indicates absence of k-means anomaly and '1' indicates its presence.**

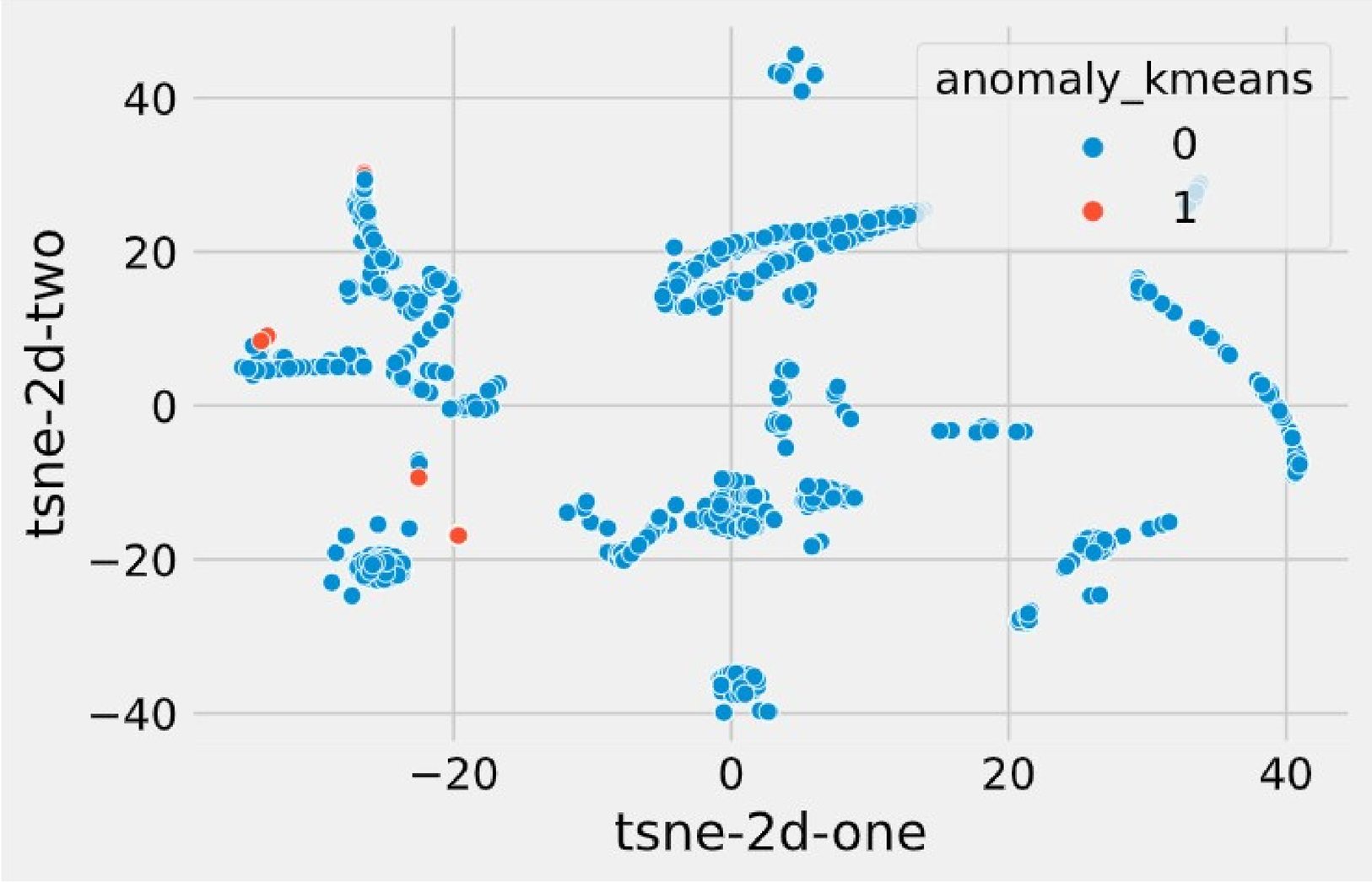


The distribution of the data points according to clusters



Histogram plotted on the basis of sum of squared distances to find the cutoff value for anomalous requests. The value obtained was 4.7.

SCATTERPLOT VISUALISATION



Here we have made the scatterplot visualization of the anomalous requests. The cutoff value obtained in the case of k-means was **4.7** i.e. if the sum squared distance value is greater than or equal to 4.7, it would mean that the request is an anomalous request detected by k-means. Here the labelling **'0'** indicates that the particular data point has **no k-means anomaly** and **'1'** indicates that it does **have k-means anomaly**.

ISOLATION FOREST

Isolation Forest, or iForest for short, is a tree-based anomaly detection algorithm. It is based on modeling the normal data in such a way to isolate anomalies that are both few in number and different in the feature space.

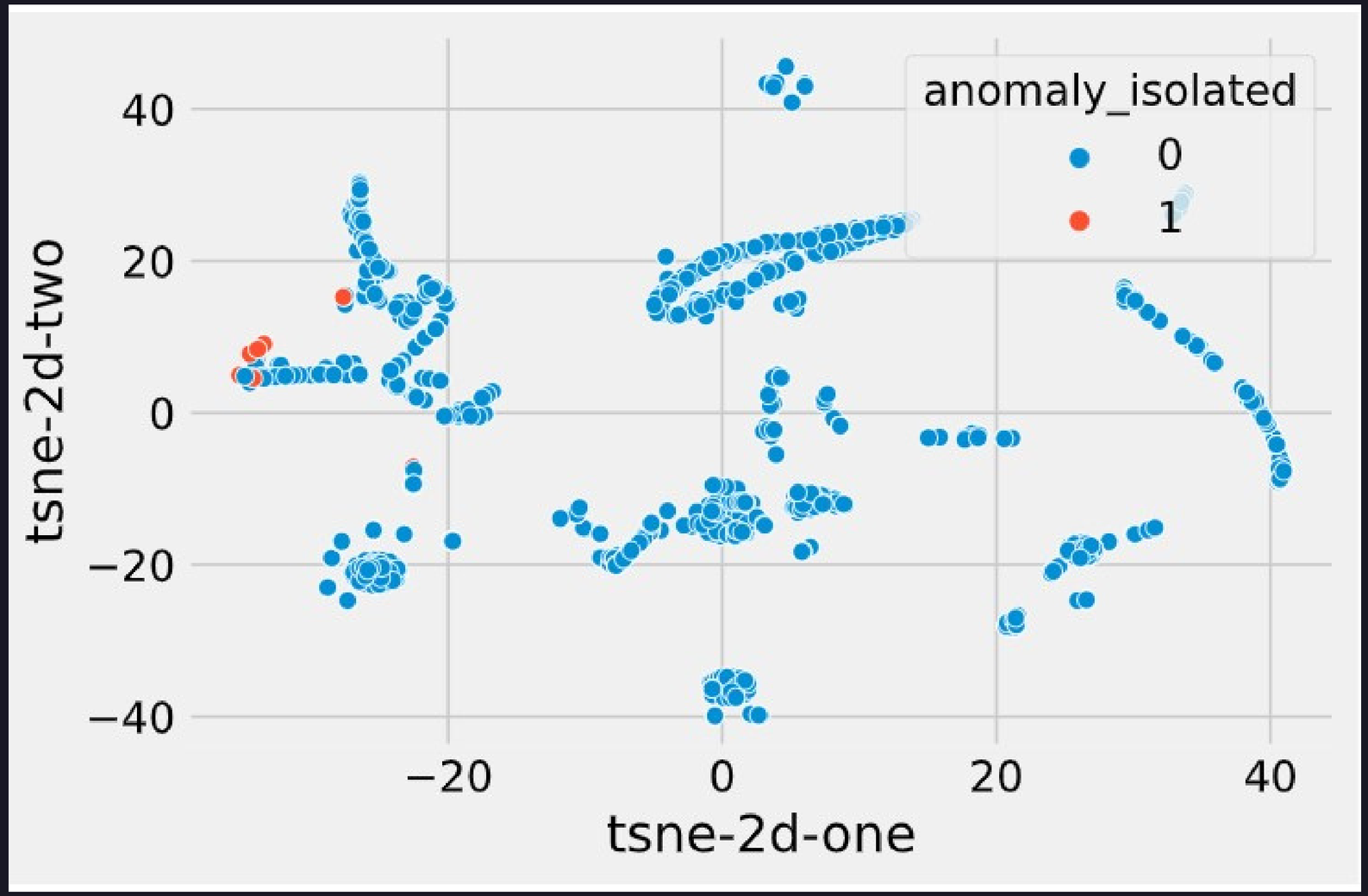
A tree structure can be constructed effectively to isolate every single instance. Because of their susceptibility to isolation, anomalies are isolated closer to the root of the tree; whereas normal points are isolated at the deeper end of the tree.

Perhaps the most important hyperparameters of the model are the "**n_estimators**" argument that **sets the number of trees to create** and the "**contamination**" argument, which is used to help **define the number of outliers** in the dataset.

We know the contamination is about 0.016 percent (by k-means) positive cases to negative cases, so we can set the "contamination" argument to be 0.01 approx.

The model will predict an inlier with a label of +1 and an outlier with a label of -1, therefore, the labels of the test set must be changed before evaluating the predictions.

SCATTERPLOT VISUALISATION



Here we have made the scatterplot visualization of the anomalous requests based on isolation forest algorithm which is an unsupervised learning algorithm. Here the labelling **'0'** indicates that the particular data point has **no isolation forest anomaly** and **'1'** indicates that it does **have isolation forest anomaly**.

ONE CLASS SVM

For anomaly detection, a one-class classifier is fit on a training dataset that only has examples from the normal class. Once prepared, the model is used to classify new examples as either normal or not-normal, i.e. outliers or anomalies. SVM, algorithm developed initially for binary classification can be used for one-class classification. **When modeling one class, the algorithm captures the density of the majority class and classifies examples on the extremes of the density function as outliers. This modification of SVM is referred to as One-Class SVM.**

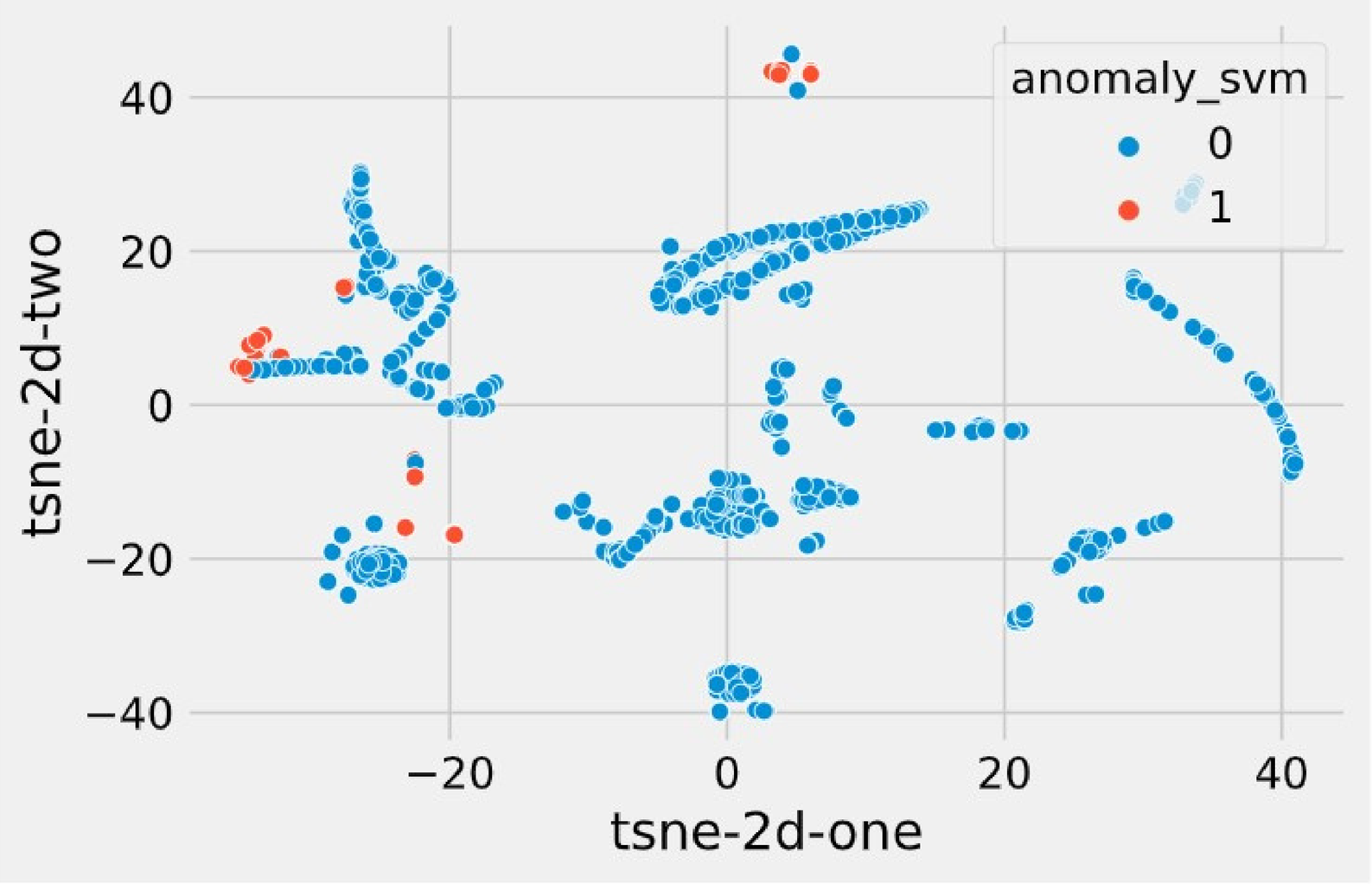
It provides a hyperparameter **"nu" that controls the sensitivity of the support vectors** and should be tuned to the approximate ratio of outliers in the data, e.g. 0.01%.

Once fit, the model can be used to identify outliers in new data.

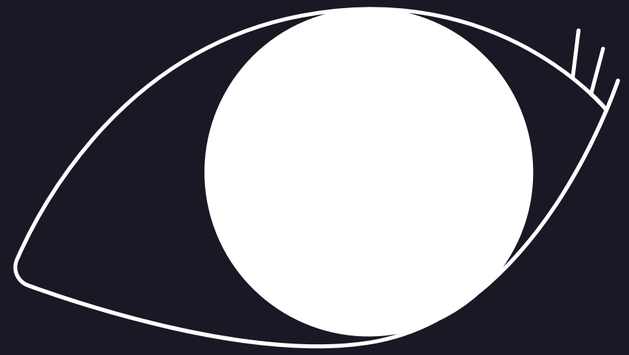
When calling the predict() function on the model, it will output a **+1 for normal examples, so-called inliers, and a -1 for outliers.**

If we want to evaluate the performance of the model as a binary classifier, we must change the labels in the test dataset from 0 and 1 for the majority and minority classes respectively, to +1 and -1.

SCATTERPLOT VISUALISATION



Here we have made the scatterplot visualization of the anomalous requests detected by one class SVM. This is also an unsupervised learning algorithm. Here the labelling **'0'** indicates that the particular data point has **no one class SVM anomaly** and **'1'** indicates that it does **have one class SVM anomaly**.



FURTHER IMPROVEMENTS: DETECTING TYPES OF API MISUSE USING SUPERVISED ALGORITHMS & DETECTING OTHER FEATURES

FAKE ACCOUNT CREATION

Fake accounts can be detected from the log data that is available to us. If **a large number of accounts use the same IP address**, this can be an indicator of a fake account creation. Another indication could be the **creation of 2 accounts of the same user from different IP addresses**. Another indication could be that **a particular account has been accessed from two different devices.**

Such a malicious practice can be detected by **Random Forest** algorithm. But this algorithm has its downsides like inefficiency to handle the categorical variables which has different number of levels. Also, when there is an increase in the number of trees, the algorithm's time efficiency takes a hit.

So to tackle these, we propose **Gradient Boosting** algorithm. This algorithm gives us an output even if some inputs are missing. This is the major reason for choosing this algorithm. Due to the use of this algorithm we will be able to get **highly accurate results**.

ACCOUNT TAKEOVER

Even with valid credentials, an attacker behaves differently than a legitimate user. An account takeover attacker will penetrate a site with stolen credentials and then move quickly. Perhaps they buy e-gift certificates and email them to an address they control.

Legitimate user takes more time than a criminal. They browse a bit. They click on some reviews. They tend to linger on certain interesting web pages before adding a product to the shopping cart. So the time spent on each site is very less which can be **determined from the Timestamp** in the log data.

Account Takeover attempts by spotting patterns of behaviour that indicate suspicious behaviour. This includes spotting indicators of an upcoming attack, such as large amounts of fake account creations that can be used to camouflage an account takeover, as well as the attack itself. ATO is considered an anomaly detection challenge, so various state-of-art unsupervised Machine Learning algorithms such as **LOF, PCA, one-class SVM, and Isolation Forest** help find abnormal patterns of a user's behavior in order to detect unauthorized actions. They work as a litmus test to find anomalies in the field of normal behavior. These algorithms group abnormal behavior data points together in a dense cluster than differs from clusters of normal behaviors.

CREDENTIAL STUFFING

Credential Stuffing can also be detected from the log data that is available to us. If **a large number of requests are made by the same IP address to access a particular account at the same Timestamp**, this can be an indicator of a credential stuffing. Another indication could be that **an attempt has been made to access particular account from two different devices or two different IP addresses.**

Such a malicious practice can be detected by **Cat Boost** algorithm. This algorithm gives outstanding results even with limited data. CatBoost supports numerical, categorical, and text features but has a good handling technique for categorical data. The CatBoost algorithm has quite a number of parameters to tune the features in the processing stage. "Boosting" in CatBoost refers to the gradient boosting machine learning and is a machine learning technique for regression and classification problems.

BOT SCRAPING DETECTION

Machine learning detection: We can use **both supervised and unsupervised machine learning** to identify new bot patterns in real time.

For maximum predictive accuracy, we analyze data sets with different granularities per request, per session, per IP, at different time scales (days, weeks), on one or multiple customers's traffic. With supervised machine learning, our objective is to predict a label (here: bot or human) by using labeled data from the past.

In bot detection, we use **unsupervised machine learning (based on unlabeled data from the past) to look for outliers and predict new bot patterns**. Based on the premise that bots and humans have different fingerprints and/or behavioral patterns, we can detect abnormal behaviors or traffic patterns.

We encode signals related to the user behavior into a set of features to which we can apply outlier detection. This helps us identify, for example, u**sers with suspicious client side events. If the user's mouse movements, timestamps, or distance between events are unusual,** we check whether other client-side events look human or not. We constantly monitor the entire traffic to a customer website and extract features from it, such as t**he number of requests per country, most frequent user agents, and most frequent browsers**.

SUPERVISED ALGORITHMS

We plan on improving our model by **labeling** the existing data. To detect anomalies in the existing data, we can look for the following indications:

- A large no. of IP addresses trying to access the same URL in a limited time frame.
- Same IP address trying to access multiple URLs at the same time.

Once we label the data, we plan on using **Random Forest** algorithm and **Support Vector Machine** algorithm. Many experiments have also shown that **Random Forest is a suitable classifier for web API request data as it maintains a low false positive rate.**

We also have planned to assign some values based on a relative scale to each of these parameters. For example, we believe that trying to access the same URL by many different users is more threatening and assign it a larger score. Then the final score for each entry will be calculated and we will be assigning a range for the values. For eg. if the maximum score is 50, then all values from 35-50 could be considered as highly anomalous, values from 25-35 mildly anomalous and the values below 25 as not anomalous.