DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2017

Detection of Web API Content Scraping

An Empirical Study of Machine Learning Algorithms

DINA JAWAD

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION

Detection of Web API Content Scraping

An Empirical Study of Machine Learning Algorithms

DINA JAWAD

Master's in Computer Science
Supervisor at CSC was Sonja Buchegger
Examiner at CSC was Mads Dam
External supervisors were Johan Östlund & Henrik Eriksson

2017-06-20

Abstract

Scraping is known to be difficult to detect and prevent, especially in the context of web APIs. It is in the interest of organisations that rely heavily on the content they provide through their web APIs to protect their content from scrapers. In this thesis, a machine learning approach towards detecting web API content scrapers is proposed. Three supervised machine learning algorithms were evaluated to see which would perform better on data from Spotify's web API. Data used to evaluate the classifiers consisted of aggregated HTTP request data that describes each application having sent HTTP requests to the web API over a span of two weeks. Two separate experiments were performed for each classifier, where the second experiment consisted of synthetic data for scrapers (the minority class) in addition to the original dataset. SMOTE was the algorithm used to perform oversampling in experiment two. The results show that Random Forest was the better classifier, with an MCC value of 0.692, without the use of synthetic data. For this particular problem, it is crucial that the classifier does not have a high false positive rate as legitimate usage of the web API should not be blocked. The Random Forest classifier has a low false positive rate and is therefore more favourable, and considered the strongest classifier out of the three examined.

Sammanfattning

Igenkänning av webb-API-scraping

Scraping är svårt att upptäcka och undvika, speciellt vad gäller att upptäcka applikationer som skrapar webb-APIer. Det finns ett särskilt intresse för organisationer, som är beroende av innehållet de tillhandahåller via sina webb-APIer, att skydda innehållet från applikationer som skrapar det. I denna avhandling föreslås ett tillvägagångssätt för att upptäcka dessa applikationer med hjälp av maskininlärning. Tre maskininlärningsalgoritmer utvärderades för att se vilka som skulle fungera bäst på data från Spotify's webb-API. Data som användes för att utvärdera dessa klassificerare bestod av aggregerade HTTP-request-data som beskriver varje applikation som har skickat HTTP-requests till webb-APIet under två veckors tid. Två separata experiment utfördes för varje klassificerare, där det andra experimentet var utökat med syntetisk data för applikationer som skrapar (minoritetsklassen) utöver det ursprungliga som användes i första experimentet. SMOTE var algoritmen som användes för att generera syntetisk data i experiment två. Resultaten visar att Random Forest var den bättre klassificeraren, med ett MCC-värde på 0,692, utan syntetisk data i det första experimentet. I detta fall är det viktigt att klassificeraren inte genererar många falska positiva resultat eftersom vanlig användning av ett web-API inte bör blockeras. Random Forest klassificeraren genererar få falska positiva resultat och är därför mer fördelaktig och anses vara den mest pålitliga klassificeraren av de tre undersökta.

Acknowledgements

This thesis constitutes the final component for the completion of the *Master of Science in Engineering* degree in the field of *Computer Science* (*Civilingenjörsexamen Datateknik*). Numerous contributions towards the completion of the thesis have been made throughout the course of the project. I would like to thank:

Sonja Buchegger, for your academic guidance, your meticulousness, and your continuous support.

Mads Dam for taking the time to ensure that this thesis reaches an acceptable academic standard.

Johan Östlund, Henrik Eriksson and Kirill Zhdanovich for your interest in this project, your continuous support, and both technical and academic guidance.

Andrew Smith and Michael Thelin for your insight, your energy, and for the opportunity to work with a wonderful group of people.

The Web API team at Spotify, for your technical guidance, your positivity and for making me feel like a part of the team.

Contents

1 Introduction

Definition: *Web API content scraping is the act of collecting a substantial amount of data from a web API without consent from web API providers.*

Scraping is a method used to describe the extraction of data by one program from another program. For instance, the term web scraping describes the extraction of data from websites. The program doing the scraping is generally referred to as a scraper. There are various types of scrapers that act differently but have the common goal of retrieving data. Scraping is generally not illegal and is at times used for legitimate reasons such as collecting data for research or other purposes. However, there are situations where scraping is not permitted, for instance scraping Spotify's web API.

Spotify is a music streaming service that has a web API for retrieving data related to their music content. A web API can be described as an interface that makes data accessible to users. Developers use web APIs to build third party applications. Their applications ("clients") call web API endpoints to retrieve data. Spotify's web API responds with data in JavaScript Object Notation (JSON) format. JSON is a way to represent data in a readable way when transmitting it. These APIs may be exploited for various reasons. Illegitimate clients may scrape content provided by web APIs for financial gains. Taking Spotify's web API as an example, competitors may try and scrape entire curated playlists and use those in their own services without permission to do so. Clients are usually met with rate limiting when too many requests are made to the API. Rate limiting is a method used to limit the number of requests sent to the web API from a specific client. It forces the client to wait a set amount of time before it can continue using the web API. Once the limit is lifted, the client can continue scraping the API and is only completely blocked if manually detected by API providers. It is also possible to avoid getting rate limited, by not overwhelming a web API with too many requests at a time. The detection of illegitimate clients that disregard terms of use and attempt to exploit APIs will help organisations with minimising the occurrence of such activities.

The behaviour of legitimate and illegitimate clients is at times distinguishable. For instance, requests coming from illegitimate clients may be more logical than those from legitimate clients. There are many different behavioural aspects that could be considered. Illegitimate clients could have a request pattern of "sequential" nature. For instance, an illegitimate client may start with a request for some object A and then go on to object AA, AB and so on, where the letters represent the name of a song, album, playlist, etc. Some automated tools such as software agents ("bots") may vary their requests and therefore, other features need to be examined. Some examples of features to examine are request rate and variation in requested data per client. The latter is an especially interesting parameter to consider as the objective of content scraping is to gather data that has not already been obtained, while legitimate clients are more likely to ask for the same data more than once.

1.1 Objective

The motivation behind this project is to enable Spotify's Web API team to block illegitimate users trying to scrape their content. This is achieved by first finding a way to detect them, which is what the focus of this project will be on.

Spotify's music content is their biggest asset. The company has stakeholders and licences that are very central to the business. It is therefore imperative to protect the data, while still being able to provide a web API. The Web API team needs a tool that can help them find clients that do not adhere to the terms of use. The following section of the Developer Terms that each developer using the web API has to accept is relevant for this project:

> "*Do not improperly access, alter or store the Spotify Service or Spotify Content, including (i) using any robot, spider, site search/retrieval application, or other tool to retrieve, duplicate, or index any portion of the Spotify Service or Spotify Content (which includes playlist data) or collect information about Spotify users for any unauthorised purpose;*" [1]

In more general terms, the aim is to investigate if it is possible to detect web API scrapers using machine learning, as this could significantly decrease the need for manual work.

1.2 Problem Definition

Terms of use and rate limiting are not enough to keep clients from scraping web APIs. It is also time consuming to manually identify clients that scrape content. Spotify's web API has many different clients that use the API in different ways, which could make it more difficult to see clear distinctions between legitimate and illegitimate clients. The purpose of this project is to answer the following question: *What supervised machine learning algorithms can be used to detect clients attempting to scrape content from a web API?*

1.3 Delimitations

Online learning is not examined in this thesis. The dataset available for this project is, although updated daily, a static dataset and so the focus is on offline learning. The results of the experiments carried out are specific to the data at Spotify. Additionally, to minimise the chance of false positives, only cases where scraping is done aggressively, that is, where a significant number of requests are made to the web API are considered to be anomalous, as these are the ones where data can be scraped at a high rate. Furthermore, a typical application using the web API may exhibit similar behaviours while for instance requesting all albums of an artist. Such cases should not be classified as anomalous. Finally, a maximum of three algorithms are examined.

2 Background

2.1 Content Scraping

There are a number of techniques and software tools available for content scraping. Internet bots are commonly used to automate tasks involved in extracting web content [2]. A large number of bots form a botnet that can be used to perform tasks in parallel. Bots in a botnet may use different IP addresses when scraping, therefore making it more difficult to detect the botnet. SiteScraper [3] is one example of a scraping tool that extracts data from websites and parses it. This scraper was built with the intention of being able to adapt to changes in websites, and therefore preventing the need for manual change to the source code.

2.2 Machine Learning

Machine learning is a field of study where the objective is to enable computers to learn. Instead of programming a computer to act a certain way, data is provided to it that is interpreted by the computer and used to make predictions. Machine learning has numerous applications and is used in many areas of study. Some examples include medical research [4], search engines [5], speech recognition [6], and intrusion detection systems (IDS) used to detect network and system abuse [7].

The algorithms used to make predictions vary depending on the type of problem that is being solved. For instance, when dealing with a regression problem, the algorithm will need to be able to give predictions on continuous data. On the other hand, a classification problem requires an algorithm to predict the class that an entity belongs to. The predictions made by an algorithm are data driven which means that the nature of that data is important as well as the algorithms used to interpret it. The performance of one algorithm on one dataset can be very different on another dataset. Therefore, algorithms are built based on the nature of the available data.

The data used is separated into a training set and a test set. The training set is used to train the algorithm and build a model that is able to make predictions. The test set is a smaller dataset used to evaluate the model. This is referred to as the holdout method [8]. There are two primary variants of machine learning algorithms, one of which is supervised learning, where the training dataset is labelled, meaning that each observation in the dataset has been classified in some way. For instance, each client using the web API is either labelled as legitimate or illegitimate in the dataset, depending on their behavioural pattern. A label in this case is an extra column in the dataset that for each row in the dataset, states if that row represents a client that is legitimate or a scraper. The labelled data is then used to classify other unlabelled data in the test set [9]. On the other hand, unsupervised learning techniques do not make use of labels and instead rely mainly on clustering methods. Data clustering is used to group data together in order to form clusters. Different clusters represent different types of activity. Clustering in anomaly detection (Section 2.3) aims to expose data points that are not part of any cluster [9]. In semi-supervised learning, only legitimate clients are labelled and compared to unlabelled data in order to detect outliers [9].

2.3 Anomaly Detection

There are a number of subfields in machine learning, one of them being anomaly detection, which concerns finding outliers in data. Anomaly detection is used in IDS, for fraud detection and has many other applications. The algorithms used in anomaly detection can be seen as binary classifiers, where one class is normal, and the other consists of outliers. In the areas where anomaly detection is used, it is often the case that the data used for training and testing is imbalanced, meaning that the majority of the data is representative of normal usage, while only a small percentage contains outliers [10].

There are different categories of anomaly detection techniques, such as supervised, unsupervised and semi-supervised approaches. Other types of anomaly detection techniques rely on signature- and rule-based approaches [11], where static rules define what abnormal activity looks like. The disadvantage of using this approach is that the rules need to be updated regularly and do not necessarily model real behaviour, therefore causing false negatives. Illegitimate clients may also change their request pattern to avoid getting detected. Therefore, it could be more favourable to detect outliers using a model that learns.

2.4 Machine Learning Algorithms

This section introduces a number of machine learning algorithms such as k-Nearest Neighbour, Support Vector Machines, and ensemble learning methods, that may be used to solve the classification problem described in Chapter 1.

In ensemble learning, multiple learners are combined to obtain more accurate predictions. Ensemble learning is useful for resolving the bias-variance trade-off, where bias is the error generated when wrong assumptions are made, causing underfitting, and variance is the error generated due to high sensitivity, causing overfitting. Underfitting refers to the modelling of data when the underlying correlations in the data are not considered, while overfitting happens when the noise in the data is modeled instead of the actual correlations in the data.

2.4.1 k-Nearest Neighbour

One of the most basic machine learning algorithms, k-Nearest Neighbour (kNN) [12] is used for both classification and regression. In classification, the algorithm classifies a data point based on the classification of its k-nearest neighbours. In Figure 2.1, $k = 6$ inside the dotted line and the aim is to classify point p as either a triangle, circle or square. The algorithm predicts that p belongs to the triangle class as there are more triangles in the neighbourhood. One of the biggest advantages of this algorithm is that it is very simple to implement.

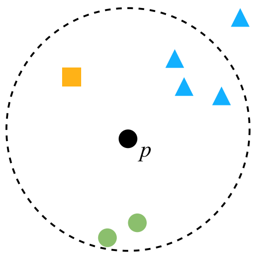

Figure 2.1: Classification using kNN, where each shape represents a different class, and $k = 6$.

2.4.2 Support Vector Machines

Another algorithm used for classification is the Support Vector Machine (SVM) algorithm. A hyperplane is constructed to separate classes in the best way. Hyperplanes are subspaces of a lower dimension than the current space. For instance, when the current space is three-dimensional, the hyperplane will be a two-dimensional plane. The hyperplane is placed as far away from each class as possible. The aim is to maximize the margins (Figure 2.2). The data points on the margins are called support vectors.

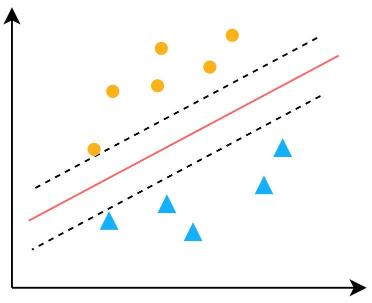

Figure 2.2: Using SVM to maximize margins (represented by dotted lines) around a hyperplane to separate two classes.

For problems that are not linearly separable (Figure 2.3a), a kernel function is used that maps the data in the low dimensional space onto a higher dimensional space, in order to turn the problem into a linearly separable one (Figure 2.3b).

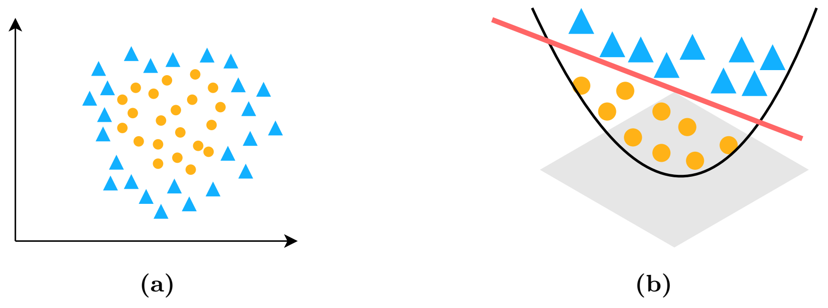
Figure 2.3: Mapping data in (a), a problem that is not linearly separable, onto a higher dimensional space (b), using a kernel function. The data is then separated with a two-dimensional hyperplane.

SVMs are able to work with relatively small datasets and tend to be resistant to overfitting [10] [13].

2.4.3 Random Forest

A number of algorithms in machine learning use decision trees. One such algorithm is Random Forest [14] which is an ensemble method, where random subsets of the training data are used to grow decision trees. Using decision trees can cause high variance and therefore, overfitting. Bootstrap aggregating ("bagging") shown in Algorithm 1, is done to reduce variance.

Algorithm 1: Building a random forest

```
Input  : training data $T$, a set of features $F$, number of features to use $n$,
         number of trees to create $m$
Output: random forest

for $i \leftarrow 1$ to $m$ do
    Take bootstrap sample of $T$: $t$
    Take random subset $f$ of $n$ features from $F$ at each node of decision tree
    Split nodes using $f$        // Split: divide node into sub-nodes
    Build decision tree $i$
end
```

Given a new data point, it is now possible to classify the data point using the "forest". Each tree makes a prediction when trying to classify a data point and a majority vote is taken. The advantage of using many trees is that together, they are not sensitive to noise and therefore the variance is low. However, when there is a high correlation between the trees (i.e. the trees are similar to each other), the error rate increases. This can be solved by adjusting the number of features to find the optimal value [10].

2.4.4 AdaBoost

Adaptive Boosting (AdaBoost) [15] is an ensemble method that combines weak classifiers that converge into a better one. A classifier is considered weak when it is slightly better than random, with an error rate just below 0.5. Each time a model is built, it attempts to learn from the previous model in order to avoid making the same mistakes. This is done by increasing weights on data points that get an incorrect classification while decreasing weights on correct classifications [8]. Decision trees are often used as weak classifiers. Algorithm 2 describes the different steps taken in AdaBoost to classify data [15].

Algorithm 2: The AdaBoost algorithm

```
Input  : A set of training samples $(x_i, y_i)$ where $i = 1, \dots, N$, number of
         iterations $T$, weak classifier $WeakClassifier$
Output: 0 or 1 depending on some threshold

Initialize weight vector $w_i^1$
for $t \leftarrow 1$ to $T$ do
    Set $p^t = \frac{w^t}{\sum_{i=1}^N w_i^t}$
    Hypothesis $h_t = WeakClassifier(p^t)$
    Error of $h_t, \varepsilon_t = \sum_{i=1}^N p_i^t |h_t(x_i) - y_i|$
    Set $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$
    Set new weight vector $w_i^{t+1} = w_i^t \beta_t^{1-|h_t(x_i)-y_i|}$
end
```

AdaBoost is considered to be one of the best classifiers and is able to reduce variance as well as bias [16].

2.5 Data Sampling

As the data handled in this project is imbalanced, detecting outliers with the mentioned algorithms can become difficult. The algorithms may consider the outliers to be noise.

A number of techniques can be used to make imbalanced data more balanced. The Synthetic Minority Over-sampling Technique (SMOTE) is one such technique [17]. It generates synthetic data in order to make the data more balanced [18]. The full SMOTE algorithm is listed in Appendix A, and a summary of the algorithm used to generate synthetic data is presented below:

1. A difference d between a sample and its k-nearest neighbours is calculated.
2. d is multiplied by a random number in the range [0, 1].
3. The result of the multiplication is added to the original sample, creating a point between two samples [17].

This technique is shown to be effective in increasing the accuracy of classifiers where data is imbalanced [17].

2.6 Performance Measure

One way to evaluate the performance of different machine learning algorithms is to use Matthews Correlation Coefficient (MCC) [19]. This value is used to compare the performance of various algorithms.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+TN)(FP+TN)(TP+FP)(TN+TN)}} \tag{2.1}$$

where TP = True Positive, TN = True Negative, FP = False Positive, and FN = False Negative. The output is a number in the range [1, -1], where 1 is a perfect prediction, 0 is considered random and -1 is completely incorrect.

MCC is considered to be a reliable and balanced evaluation method as all four outcomes TP, TN, FP, and FN are considered in the calculation. The MCC value tends to grow slower than other measures such as accuracy. A MCC value of 0.5 is given when 75% of the predictions are correct [20].

2.7 Cross Validation

The disadvantage of using the holdout method to test a classifier is that not all data is used in training and testing. Each portion of the data is used separately to either train the classifier or test it. To get more accurate estimates, cross validation is used [8]. Additionally, Hawkins, Basak and Mills [21] show that using cross validation is more favourable when the number of observations available are few. There are a number of algorithms used for cross validation. The k-fold cross validation algorithm partitions the dataset into k sets (Figure 2.4) and uses k-1 sets in training and the remaining set for testing. This is done k times. Leave One Out cross validation (LOOCV) is a special case of the k-fold algorithm, where k is equal to the number of observations in the dataset. LOOCV is nearly unbiased as the training set will contain only one less observation than the entire dataset [22].

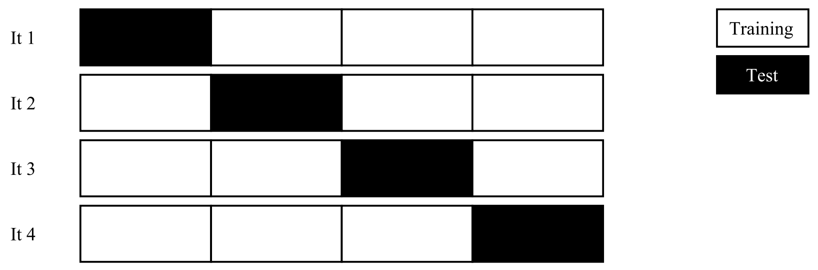

Figure 2.4: k-fold cross validation, where k = 4, It = Iteration, black boxes represent testing sets and white boxes training sets.

3 Related Work

Tripathi, Hubballi and Singh [23] propose a solution for detecting two variations of Slow Hypertext Transfer Protocol (HTTP) Denial-of-Service (DoS) attacks. A DoS attack usually targets servers with the objective of bringing them down, thereby denying legitimate users service. During a Slow HTTP DoS attack, the attacker sends incomplete HTTP requests which causes the server to wait for a certain amount of time for the rest of the request. Tripathi, Hubballi and Singh [23] design an anomaly detection system where probability distributions are compared using The Hellinger Distance (Equation 3.1) to detect this attack, by looking at HTTP traffic.

$$d_H = \sqrt{\frac{1}{2}\sum_{i=1}^{N}(\sqrt{P_i} - \sqrt{Q_i})^2} \tag{3.1}$$

In the training phase, they generate a profile with different attributes and use this profile to compare it with the results from the testing phase. The Hellinger Distance is measured between the probability distributions from the training and testing phases. A threshold is also set and used to determine if some data point is an outlier. For normality, the distance measured needs to be below the threshold [23]. Although it is stated that this solution generates accurate results, there is no evaluation of the performance of the actual learning algorithm. There is also no mention of the true positive and false positive rates, making it difficult to assess the strengths of their model and apply it elsewhere. These points are addressed in this thesis in order to evaluate the algorithms used and compare their performance to one another.

Ingham and Inoue [24] evaluate anomaly intrusion detection algorithms for HTTP. The algorithms evaluated are used to detect different types of attacks, for instance, looking at request length to detect cross-site scripting and character distribution for buffer overflows. They evaluate Deterministic Finite Automata (DFA) and n-grams (often used in natural language processing) and find that these perform better than the other algorithms considered, including request length, character distribution, Mahalanobis distance, Markov model and linear combination. **Lin, Leckie and Zhou** [25] also use n-grams (for incoming HTTP requests) and Deterministic Finite Automata (DFA) for anomaly detection to identify different types of HTTP attacks. They find that DFA performs better than n-grams.

One way to protect a system from attacks is by using an Intrusion Detection System (IDS). An IDS is a monitoring system that creates alerts when unusual activity is detected. This activity is predetermined with for instance signature detection where rules are created to find certain patterns and behaviours. In addition to that, some IDSs use anomaly detection. Unlike in anomaly detection, novel attacks cannot be identified by systems using signature detection [24]. An IDS is often used to monitor network traffic. **Duffield *et al.*** [11] use machine learning algorithms to try and detect the same anomalies that an open source IDS called Snort is able to detect. Snort applies packed based rules to detect anomalies and so works on packet level, while the machine learning algorithms use an aggregated set of data called IP flows. An IP flow is a group of packets that have similar attributes, such as source and destination address. Unlike packets, IP flows do not contain payload and therefore, the machine learning algorithms have less information to work with to generate the same result. The final results of the experiment show that using machine learning algorithms is an effective way to approximate packet-level rules. A comparison was made between AdaBoost and SVM and results show that AdaBoost was the stronger classifier, as it can handle a large number of features [11]. They did not however examine the performance of a bagging algorithm such as Random Forest, which is examined in this thesis and compared to other algorithms.

Haque and Singh [26] present different ways to prevent web scraping. Their suggested solutions include changing class and id names of HTML tags from time to time, giving them a random name as scrapers use these tags to scrape content. This is not applicable to web APIs as changing names of endpoints will create a Denial-of-Service on all clients. They suggest using honeypots to gather information about and detect automated bots and botnets. A honeypot is a form of bait that attracts adversaries in an attempt to gain valuable information about the adversary and take further measures. Haque and Singh's [26] anti-scraping technique relies on the use of black, grey and white lists. These lists contain IP addresses of users of a website. A black list contains blocked IP addresses, a grey list contains IP addresses of suspicious users and a white list contains trusted IP addresses. When entering a site, users in the grey list have to solve a challenge-response test called CAPTCHA ("Completely Automated Public Turing test to tell Computers and Humans Apart"). After some time, the user may be added to the white list. This part of the solution does not scale well. The *grey list* could be very long, and it becomes very important to not include legitimate users in the list. Given the factors provided that add users to the grey list, it is highly likely that the list contains legitimate users that are being checked for no reason. This decreases user experience and therefore chases away legitimate users. In addition to that, CAPTCHA is not always applicable to web APIs as not all calls to a web API are made as a result of a human trying to access some service. They also suggest analysing user activity by looking at URLs visited, and the interval and frequency of visits to the site. This is a solution that can be applied to web APIs using machine learning, which is covered in this thesis.

Chen, Shih and Huang [27] propose a solution for detecting internal anomalies for Firefox OS (FxOS), a web based operating system. There are three layers to the operating system called Gonk (kernel and libraries), Gecko (browser engine), and Gaia (front end). Gecko has web APIs used by Gaia. Their study is focused on events where API calls are made more frequently and consume more resources than usual. An anomaly detection module (ADM) is trained and then used to make predictions on unseen data. The ADM includes three stages: normalisation, characterisation, and vectorisation. A characteristic vector is created in the vectorisation stage and is used in the fourth phase during training, the throttle stage, to create a throttle vector. The characteristic and throttle vectors go through a difference function and the output is a value that is compared to some threshold. An alert is created if the output is larger than the threshold. The results of the experiments show that the ADM gives a false negative rate of 0% and a false positive rate of 12.5%. In this thesis, more features for web APIs in addition to those covered by [27] are considered.

Fraud detection is another area where machine learning is used to detect outliers [10] [28]. **Bhattacharyya *et al.*** [10] evaluate SVM, Random Forest and logistic regression for detecting credit card fraud, and find that Random Forest performs the best. They perform undersampling of the majority class and train their models on four different datasets containing different percentages of the minority class. However, they do not evaluate the algorithms on data where the minority class is oversampled, and do not use more than 15% fraud data in the training sets. **Whitrow *et al.*** [28] also evaluate SVM, Random Forest and logistic regression as well as kNN for detecting credit card fraud. They also find that Random Forest performs better than the other algorithms considered. Both [10] and [28] do not evaluate any boosting algorithms.

In anomaly detection, it is often the case that outliers are rare and therefore, the class that represents illegitimate clients is underrepresented. For instance-based learning such as kNN, each nearest neighbour of some data points is more likely to belong to the majority class as the class grows, leading to misclassification of the minority class [29]. For imbalanced datasets, synthetic data can be injected. **Lundin *et al.*** [30] present a model that uses authentic data to generate synthetic data. They generate synthetic data by simulating attacks. Their methodology involves collecting authentic attack and non-attack data, identify properties in the data such as attack characteristics, and creating an attacker profile to model attackers. The methodology also includes a simulation of the rest of the system, where legitimate clients are modelled as well. However, simulating the entire system is not mandatory. Using synthetic data has its advantages. For instance, the data can be designed to represent any potential attack that is not represented in the authentic data, in order to train a model to detect such an attack.

Khoshgoftaar, Golawala and Hulse [31] use Random Forest to learn from imbalanced data, where the percentage of the minority class is varied between 1.33% and 30%. They create 48 Random Forest learners with different parameters and find that the parameters that give the best results for binary classification of imbalanced data include using 100 trees and $\lfloor \log_2 M + 1 \rfloor$ number of features where M is the total number of attributes in the training dataset. Their Random Forest classifier performed better than six other classifiers including SVM, 2-NN and 5-NN.

Unsupervised learning algorithms are also used in anomaly detection. **Ahmed and Mahmood** [32] present a method for detecting DoS attacks using clustering. Their algorithm is based on a variant of the k-means algorithm, called x-means. In the k-means algorithm, k specifies the number of clusters to find. Clusters are identified by first initialising k cluster centroids randomly, and then assigning each data point in the training data to the nearest centroid. Once that is done, the centroids are moved to the average of the data points assigned to each centroid. This is done iteratively until clusters are formed. With x-means, the number of clusters is not predefined and instead, an estimate of the number of clusters is made. The solution proposed by [32] generated results with an accuracy of 97%, while k-means gave 85%. Another unsupervised learning algorithm called one-class SVM uses unlabeled data, unlike original SVM where data is labeled. The one-class SVM algorithm has been successfully applied in a number of areas [33] [34] [35].

Neural networks have also been used to detect anomalies using one class of data. **Dau, Ciesielski and Song** [36] use Replicator Neural Networks (RNN) with one hidden layer, and train their model on data from the normal class. Another experiment by **Meyer, Leisch and Hornik** [37] where the objective was to compare SVM to other classifiers (including Random Forest, Neural Networks, tree boosting, etc.), ensemble methods performed better than both SVM and neural networks.

Finally, there are numerous algorithms that are based on nearest neighbour [38]. One of these is the Local Outlier Factor (LOF) algorithm [39] which was developed for anomaly detection. In the first part of the algorithm, locality is determined by identifying the k nearest neighbours of a data point. A LOF score is then calculated by examining the local densities of neighbours. The local density of an outlier is lower than that of its k nearest neighbours. Likely outliers get a high LOF score. LOF is able to find outliers in datasets that have varying densities and is considered an anomaly detection technique [9]. Because it takes local density into account, LOF will for instance identify p in Figure 3.1 as an outlier even though the distance from p to a point in the more dense cluster is similar to the distances between the points in the sparse cluster. Although LOF has been successfully applied in different areas [40] [41], it does not perform as well compared to other classifiers and has high computational complexity [42] [32] [43].

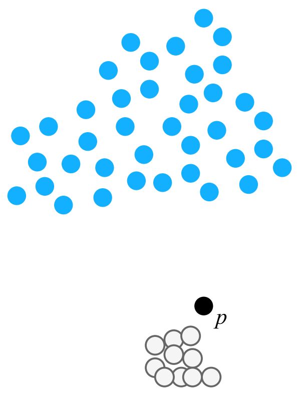

Figure 3.1: Clusters with different densities.

4 System Overview

The web API in question is stateless, and consists of a number of publicly exposed endpoints that can be called by third parties. An endpoint specifies the location of some resource via a Uniform Resource Identifier (URI). Each client making requests to the web API is identified using a unique client ID which is obtained when the application is registered through Spotify's developer site.

4.1 Authorisation Flow

The web API makes use of the OAuth 2.0 Authorization Framework [44] when serving requests. As specified in RFC 6749, the OAuth protocol consists of six steps. The first two steps in the protocol require the client to be authenticated through Spotify's accounts service. Once that is done, the client is issued an access token from the authorisation server. The client is then able to use the access token to authenticate itself when making calls to the resource server.

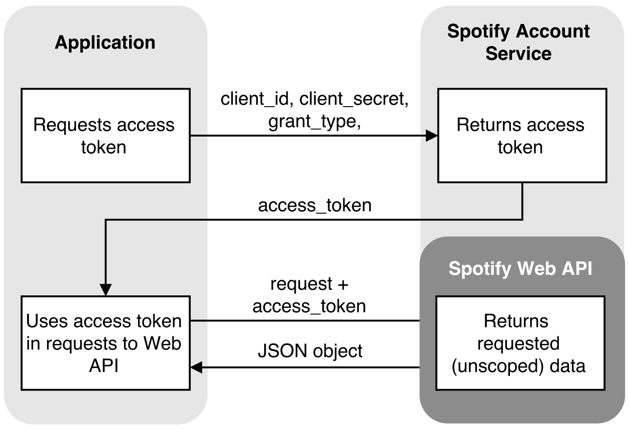

Figure 4.1: Server-to-server communication [45].

The resource server that receives web API requests will only serve requests made by clients with valid access tokens. Figure 4.1 describes the authorisation flow for server-to-server communication, where the requests are anonymous yet authenticated.

Another case to consider is when requests are made on behalf of a user. These requests will, in addition to the client ID, include a user ID. Each user will have a unique user ID. This flow is described in Figure 4.2.

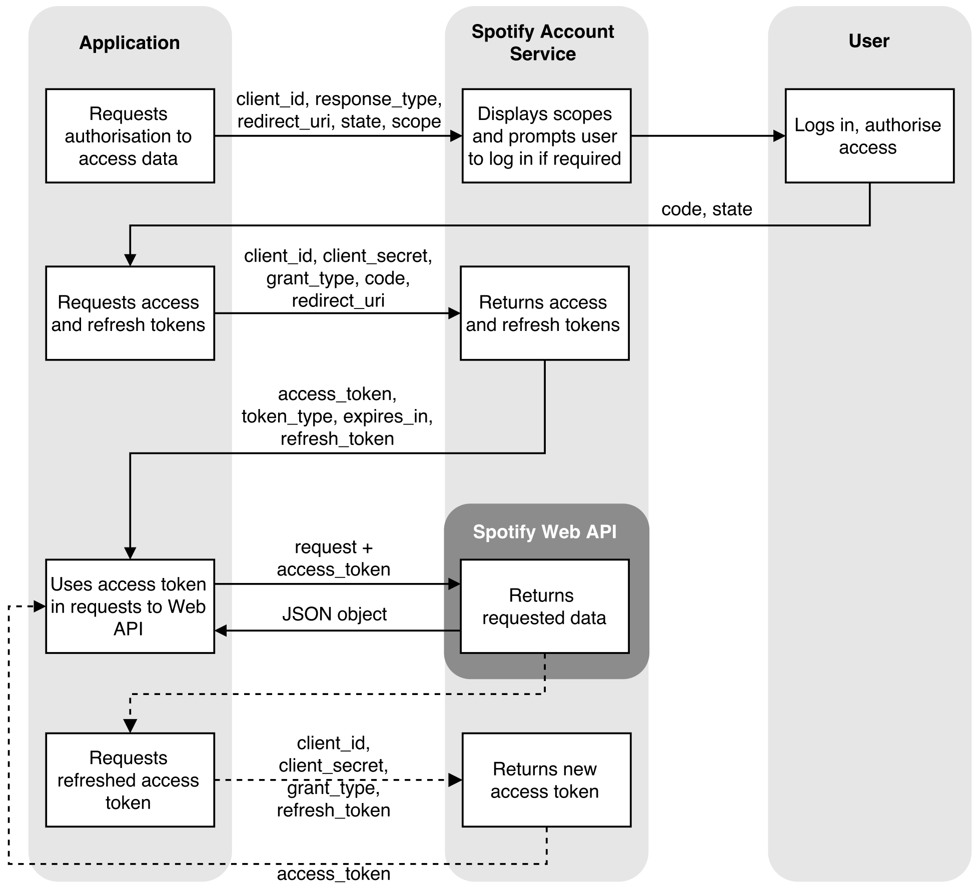
Application
Spotify Account Service
User
Requests authorisation to access data
client_id, response_type, redirect_uri, state, scope
Displays scopes and prompts user to log in if required
Logs in, authorise access
code, state
Requests access and refresh tokens
client_id, client_secret, grant_type, code, redirect_uri
Returns access and refresh tokens
access_token, token_type, expires_in, refresh_token
Spotify Web API
Uses access token in requests to Web API
request + access_token
JSON object
Returns requested data
Requests refreshed access token
client_id, client_secret, grant_type, refresh_token
Returns new access token
access_token

Figure 4.2: Requests made on behalf of a user [45].

4.2 Endpoints

The APIs provided include a number of endpoints that return objects in JSON format. The endpoints provide metadata about objects such as albums, artists, playlists and individual tracks. The *Playlists* endpoint for instance returns a list of track objects, where each object contains metadata about a track, such as the name of the track, metadata about the artist, metadata about the album it belongs to, and relevant external links to the Spotify Web Player, where the track can be played, artist page viewed or album accessed. This endpoint requires a username, meaning that the playlists are originally created by Spotify users, or by playlist curators at Spotify, if for instance the *spotify* username is provided. The *Playlists* endpoint also allows POSTs for the creation of playlists.

Each endpoint has a set of parameters that can be specified in the request. Figure 4.3 shows an example of a request made to the *Search* endpoint.

```
curl -X GET "https://api.spotify.com/v1/search?q=some+search+term&limit=50&
    offset=1000" -H "Accept: application/json" -H "Authorization: Bearer <
    Access Token>"
```

Figure 4.3: HTTP request to the search endpoint.

The maximum number of results returned is set in the limit parameter, while the offset parameter is used to return a set of results starting at a certain index, in this case being index 1000.

Each request made to the endpoints is logged and stored in a database. A request consist of the following data:

- *Timestamp* of the time the request was made.
- *Client ID* used to identify the application sending requests.
- Anonymised *user ID* when requests are made on behalf of a client's user.
- *Type of subscription* (free or premium) associated with the client or user ID.
- *IP address* of client or user.
- *Country* where the request originated.
- *HTTP status code* returned to the client.

- *HTTP request referrer*, found in the HTTP header. Referrers are generally used to see from where a request originated.
- *HTTP request method*, e.g. GET, PUT and so on.
- *User agent string (UAS)*, a string found in the user agent HTTP header field. As specified in RFC 7231, the user agent field should contain "information about the user agent originating the request" [46]. A user agent is a program that acts on behalf of a user.
- *URI* that reveals what data the client has requested for. For example, the URI in the request in Figure 4.3 is

  ```
  /v1/search?q=some+search+term&limit=50&offset=1000
  ```

4.3 Threat Model

There are a number of characteristics that can be used to differentiate between scrapers and legitimate clients. The following threat model devised together with the web API team at Spotify, specifies how requests made by scrapers ("illegitimate requests") may be distinguishable from legitimate requests (and is based on observations and previous experiences):

User Agent

- A request with a browser user agent field but no HTTP referrer.
- Different user IDs all have the same user agent string.

Request Rate

- Constant or little variation in request rate over a long period of time.
- Requests sent with equal intervals.
- Staying just below the rate limit.
- A high average number of requests per second (rps).

URI

- Always returning the maximum amount of results possible, using the limit parameter.

- Requests often contain a high offset value.
- Requests often contain sequenced offset values.
- Frequent use of the offset parameter.
- Systematic modification of the URI in continuous requests.
- Very specific search terms.
- All or most requests ever sent are unique, since data does not need to be acquired more than once.

IP address

- Requests from clients with IP addresses in countries not being served yet.
- Requests from clients with IP addresses belonging to data centres.

Patterns

- Web crawler behaviour: a search for some object followed by several GET requests.
- Similarity in number of requests, across different client IDs.

5 Experimental Setup

In order to train a classifier to detect scrapers, a dataset is constructed using the logged HTTP requests made to the web API. A number of features are created based on the threat model, in addition to others. These features are described in Section 5.1. The libraries used and other implementation details are described in Section 5.2, while Section 5.3 consists of the experiments carried out to obtain the results in Chapter 6. In Section 5.4, a set of requirements for the process of web API content scraping detection are listed. Finally, Section 5.5 describes the limitations in these experiments.

5.1 Data

The data used to create a dataset for the classifiers consists only of GET requests as the act of scraping includes collecting data rather than submitting it. Requests made over a period of two weeks are gathered and used to create the final dataset, which contains one observation per client. The reason for choosing a two-week period for this dataset is because of the sheer amount of data that needs to be processed in order to create a dataset. This is further explained in Section 5.5.

The aim is to classify a client as either legitimate or illegitimate and therefore, a set of features are created for each client ID. These features are listed in Table 5.1. The final dataset contains a list of unique client IDs.

Table 5.1: Features created for the final dataset

Name	Feature Description
r1	Total number of requests made
r2	Percentage of requests made to each of the 30 endpoints (creating 30 features)
r3	Variance of request rate when requests are made
r4	Average request rate when requests are made
r5	Staying just below rate limit, 0 for no, and 1 for yes
ip	Number of unique IPs
uas	Multiple user IDs all have the same user agent string, 0 for no, and 1 for yes
uri1	Percentage of requests made with the limit parameter set to the maximum value
uri2	Percentage of requests containing the offset parameter
uri3	Average offset value
uri4	Number of requests with unique URIs
free	Number of users with a free subscription
prem	Number of users with a premium subscription
stat	Number of requests that received 4XX status codes

5.2 Implementation

Features are created and data sampled using Google's BigQuery API where BigQuery standard SQL is used for querying. The BigQuery package for Python is available through the Google Cloud library [47]. The machine learning algorithms used in the experiments are available through the Scikit Learn package [48], via Python. Finally, the implementation of SMOTE is available through the Imbalanced-learn package [49] and is used for oversampling.

In order to create the final dataset used in the experiments in Section 5.3, a Python script was implemented to automate the process. The script runs a set of queries on a table with two weeks worth of requests. The results from each query is saved in a separate table in BigQuery. Each query generates results for each feature. Once all tables are created, they are joined by the client ID in each table to create one single table with distinct client IDs. Each row represents one client and each column is one feature. Finally, a second script was created to evaluate the three classifiers chosen using the data prepared. Here, the Scikit Learn [48] and Imbalanced-learn [49] packages are used.

5.3 Experiments

The experiments carried out consist of three machine learning algorithms: kNN, Random Forest and AdaBoost, where both Random Forest and AdaBoost used decision trees as weak classifiers. The kNN algorithm is chosen to see how well a very simple distance metric classifier can perform, while Random Forest appears to be a strong and widely used classifier and Adaboost from the same ensemble methods family. SVM is excluded due to its sensitivity towards imbalanced datasets [50].

In the first experiment, these three algorithms are trained using a dataset that has not been oversampled. In the second experiment, oversampling is used to increase the percentage of anomalies in the data up to 20% in hopes that having a more balanced dataset will generate better results. In both experiments, LOOCV is used when training and testing the classifiers. For the second experiment, oversampling is done in each iteration of the cross validation. That is, each time the classifier is retrained using a new training set, oversampling is done on that set before training the classifier. In each iteration of LOOCV, a new observation (test set) is classified. Once the LOOCV has tested each observation on each training set, the resulting predictions for each observation are used to evaluate the performance of the classifier. MCC is calculated along with a confusion matrix, containing the number of false positives, true positives, false negative and true negatives.

5.4 Requirements

The aim of performing these experiments is to find a process that can be deployed and generate useful results. Below follows a set of requirements for the entire process of detecting scrapers:

1. Using the Python script mentioned in Section 5.2 to prepare a dataset for the classifier should be straight forward and should not take an unreasonable amount of time to run.
2. The best performing classifier to be used when trying to detect scrapers should not take an unreasonable amount of time to run.

3. The results generated should not have a high false positive rate. A total of 40 clients to manually inspect is the upper bound. This number includes the set of true positives.
4. The best performing classifier should have a high MCC value, with a lower bound of 0.5, in order to ensure that the classifier is able to correctly label both legitimate and illegitimate clients.

5.5 Limitations

While in theory, more than two weeks worth of request data is available, in practice it is not accessible as it would require a lot of processing which would have exceeded the time given to build this work on. Therefore, a limit of two weeks is set and scrapers that have been found previously are included in that data, in addition to some labelled manually using the threat model.

Given that there was not enough labelled data available that could be classified as illegitimate, synthetic data was generated to see if that has a positive effect on the results. Using synthetic data can introduce biases as it does not necessarily represent actual events. It is after all merely synthetic. Only a very small set of scrapers were available and were labelled manually. These scrapers fit some characteristics in the threat model described in Section 4.3.

Labels for legitimate clients did not exist either. However, data that is not considered illegitimate is labelled as legitimate on the basis that:

1. The web API is mostly used by trusted partners (with approximately 90% of all requests coming from them) where a set of clients are known or assumed to be legitimate.
2. The web API team assumes that the majority of their data consists of legitimate traffic.
3. Only extreme cases of scraping are considered, meaning that clients that send a large number of requests in comparison to others have been labelled manually using the proposed threat model.
4. A classifier that does not generate a high false positive rate is more favourable as legitimate clients should not be blocked.

5. An attempt was made to cluster all data before labelling, to see if separate clusters could be created. This would indicate large variations in client behaviour. However, the results from doing so created only one cluster.

It is still possible that a small percentage of the clients labelled as legitimate are in fact illegitimate, which according to [51] is not necessarily considered to be an issue for anomaly detection.

6 Results

This chapter presents the results gathered from two experiments performed on three machine learning algorithms. A set of 7271 observations were produced when creating the final dataset used for evaluation.

Experiment 1: Without Oversampling

In the first experiment, oversampling was not performed. The three machine learning algorithms kNN, Random Forest and AdaBoost were run with Leave One Out cross validation and a prediction was made for each observation in the dataset. The final MCC value is presented in Table 6.1 along with the percentage of scrapers found in each case.

Table 6.1: Performance of each algorithm, without SMOTE

Classifier	MCC	Percent Outliers Found	TN	FN	TP	FP
kNN	0.665	48%	7245	13	12	1
Random Forest	0.692	48%	7246	13	12	0
AdaBoost	0.670	52%	7244	12	13	2

Experiment 2: With Oversampling

In the second experiment, oversampling with SMOTE was performed. The three machine learning algorithms kNN, Random Forest and AdaBoost were once again run with Leave One Out cross validation oversampling was done each time the dataset was split into a training and testing set. A prediction was made for each observation in the dataset and the final MCC value is presented in Table 6.2 along with the percentage of scrapers found in each case.

Table 6.2: Performance of each algorithm, with synthetic data

Classifier	MCC	Percent Outliers Found	TN	FN	TP	FP
kNN	0.407	72%	7187	7	18	59
Random Forest	0.630	52%	7242	13	13	4
AdaBoost	0.539	64%	7227	9	16	19

7 Discussion

Below follows an analysis of the results from the evaluation in Chapter 6.

The MCC Vaule

In both experiments, Random Forest has the highest MCC value out of the three algorithms, with 0.692 in the first experiment and 0.630 in the second. The AdaBoost classifier is slightly weaker with an MCC value of 0.670 in the first experiment and 0.539 in the second. Finally, kNN is the weakest classifier according to its MCC value, with 0.665 and 0.407 in the first and second experiment, respectively. According to the MCC value, Random Forest appears to be the strongest classifier in both experiments and the change in the MCC value between the two experiments is minimal. On the other hand, kNN has the highest decrease in MCC.

Sythetic Data

One unexpected finding is that all three classifiers were weaker with oversampling in experiment two. Adding synthetic data using SMOTE did not improve the results according to the MCC value. However, with oversampling, a higher percentage of outliers were found, with kNN having the highest change from 48% to 72%. Yet, the kNN classifier in experiment two generated a higher number of false positives, making the classifier weaker, hence the relatively low MCC value and the highest decrease in MCC between experiment one and two. One possible reason for the oversampling not improving the classifiers is that the SMOTE algorithm did not introduce enough variability in the data and possibly decreased the variability of the minority class (illegitimate clients). This could be due to not having enough examples in the minority class. In the first experiment, the AdaBoost classifier had the highest rate of true positives, and the MCC value was slightly lower than that of Random Forest due to a somewhat higher false positive rate. The Random Forest classifier had the least number of false positives in both experiments.

Process Adoption

The results from the experiments show that Random Forest is a suitable classifier for web API request data as it is able to find 48% of outliers while still maintaining a low false positive rate. In order to make use of this classifier, the first step is to train the classifier using a labelled dataset such as the one used when performing the experiments in this thesis. Once the classifier is trained, it can be used to classify unknown data. To do so, another dataset containing unlabelled data should be prepared as described in Section 5.2. The classifier can then be used to label the newly created dataset. Clients that are labelled as scrapers can then be inspected manually to eliminate false positives. Since the process includes a manual inspection, it is less convenient and more time consuming to choose a classifier such as kNN when the false positive rate is as high as it is.

8 Conclusion

Following an evaluation of the performances of the different classifiers, it is evident that the Random Forest classifier was the best fit for the dataset used. All four requirements stated in Section 5.4 were met, allowing for reasonable running time of the dataset creation script as well as the Random Forest classifier with approximately 9 minutes for evaluation using LOOCV, training and classifying unlabelled data, while generating a low false positive rate and an MCC value of 0.692.

8.1 Possible Improvements

A number of steps can be taken in order to build a more accurate classifier for future work. For instance, sampling more data for the minority class can make the training dataset more balanced and avoid misrepresentation of the minority class. It is also possible that labelling data manually, including observations for the majority class (legitimate clients) can help build a stronger classifier. In addition to that, other oversampling or synthetic data generation methods can be explored and compared with SMOTE. Other ways to possibly improve the results include performing feature selection and exploring other types of features. Some examples of features include inspecting IP addresses to see if they belong to specific data centers, or looking for systematic modification of URIs in continuous requests. Various feature selection algorithms can be deployed with a classifier in order to find the best combination of features. Finally, as LOOCV is a highly demanding to run and becomes even more so as the dataset grows, a k-fold algorithm can be used instead, where k is 5 or 10. Another set of machine learning algorithms could also be explored, for instance unsupervised machine learning algorithms instead of supervised ones, to investigate whether clusters can be formed. If so, each cluster could be inspected to see how the clients in those clusters behave. However, since the minority class is significantly smaller than the majority class, it is unlikely that any separate clusters will form.

8.2 Sustainability and Ethics

From an ethical, social and economic standpoint, finding a way to detect scrapers can help prevent theft of intellectual property. It is also in the interest of organisations to take further measures in securing their data. This can increase trust among their users. In terms of ecologically sustainable development, it can be computationally expensive to run the experiments and therefore they may use up a large amount of resources, which can have a negative impact on the environment.

Bibliography

[1] Spotify AB. Spotify Developer Terms of Use, Version 2.1. `https://developer.spotify.com/developer-terms-of-use/`, December 2014.

[2] Michael Schrenk. *Webbots, Spiders, and Screen Scrapers.* No Starch Press, 2 edition, 2007.

[3] Richard Baron Penman, Timothy Baldwin, and David Martinez. Web Scraping Made Simple with SiteScraper, 2009.

[4] Ioannis Kavakiotis, Olga Tsave, Athanasios Salifoglou, Nicos Maglaveras, Ioannis Vlahavas, and Ioanna Chouvarda. Machine Learning and Data Mining Methods in Diabetes Research. *Computational and Structural Biotechnology Journal*, 15:104 – 116, 2017.

[5] T. T. Dang and K. Shirai. Machine Learning Approaches for Mood Classification of Songs toward Music Search Engine. In *2009 International Conference on Knowledge and Systems Engineering*, pages 144–149, October 2009.

[6] L. Deng and X. Li. Machine Learning Paradigms for Speech Recognition: An Overview. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(5):1060–1089, May 2013.

[7] S. Kumar, A. Viinikainen, and T. Hamalainen. Machine learning classification model for Network based Intrusion Detection System. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 242–249, December 2016.

[8] Stuart Russel and Peter Norvig. *Artificial intelligence: A Modern Approach.* Pearson, 3rd edition, n.d. ISBN: 978-1-292-02420-2.

[9] Varun Chandola, Arindam Banerjee and Vipin Kumar. Anomaly Detection: A Survey. Technical report, University of Minnesota, 2007. TR 07-017.

[10] Siddhartha Bhattacharyya, Sanjeev Jha, Kurian Tharakunnel, and J. Christopher Westland. Data Mining for Credit Card Fraud: A Comparative Study. *Decision Support Systems*, 50(3):602 – 613, 2011.

[11] Nick Duffield, Patrick Haffner, Balachander Krishnamurthy and Haakon Ringberg. Rule-Based Anomaly Detection on IP Flows. In *INFOCOM*, Rio de Janeiro, 19-25 April 2009. IEEE.

[12] Pádraig Cunningham and Sarah Jane Delany. k-Nearest Neighbour Classifiers. Technical report, UCD School of Computer Science, 2007. UCD-CSI-2007-4.

[13] Kevin S. Killourhy and Roy A. Maxion. Comparing anomaly detection algorithms for keystroke dynamics. In *2009 IEEE/IFIP International Conference on Dependable Systems Networks*, pages 125–134, June 2009.

[14] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[15] Yoav Freund and Robert E Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139, 1997.

[16] Yanmin Sun, Mohamed S. Kamel, Andrew K.C. Wong, and Yang Wang. Cost-sensitive Boosting for Classification of Imbalanced Data. *Pattern Recognition*, 40(12):3358 – 3378, 2007.

[17] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16(1):321–357, June 2002.

[18] H. He and E. A. Garcia. Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, September 2009.

[19] David M W Powers. Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness and Correlation. Technical report, School of Informatics and Engineering, Flinders University of South Australia, 2007. SIE-07-001.

[20] Mauno Vihinen. How to evaluate performance of prediction methods? Measures and their interpretation in variation effect analysis. *BMC Genomics*, 13(4):S2, 2012.

[21] Douglas M. Hawkins, Subhash C. Basak, and Denise Mills. Assessing Model Fit by Cross-Validation. *J. Chem. Inf. Comput. Sci.,*, 43(2):579–586, January 2003.

[22] Ron Kohavi. A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[23] Nikhil Tripathi, Neminath Hubballi, and Yogendra Singh. How Secure are Web Servers? An Empirical Study of Slow HTTP DoS Attacks and Detection. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*, pages 454–463, August 2016.

[24] Kenneth L. Ingham and Hajime Inoue. *Comparing Anomaly Detection Techniques for HTTP*, pages 42–62. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[25] Li Lin, Christopher Leckie, and Chenfeng Zhou. Comparative Analysis of HTTP Anomaly Detection Algorithms: DFA vs N-Grams. In *2010 Fourth International Conference on Network and System Security*, pages 113–119, September 2010.

[26] A. Haque and S. Singh. Anti-scraping Application Development. In *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 869–874, August 2015.

[27] Borting Chen, Ming Wei Shih, and Yu Lun Huang. An Anomaly Detection Module for Firefox OS. In *2014 IEEE Eighth International Conference on Software Security and Reliability-Companion*, pages 176–184, June 2014.

[28] C. Whitrow, D. J. Hand, P. Juszczak, D. Weston, and N. M. Adams. Transaction Aggregation as a Strategy For Credit Card Fraud Detection. *Data Mining and Knowledge Discovery*, 18(1):30–55, 2009.

[29] Sotiris Kotsiantis and Panagiotis Pintelas. Mixture of Expert Agents for Handling Imbalanced Data Sets. *Annals of Mathematics, Computing and TeleInformatics*, 1(1):46–55, 2003.

[30] Emilie Lundin, Håkan Kvarnström, and Erland Jonsson. *A Synthetic Fraud Data Generation Methodology*, pages 265–277. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[31] T. M. Khoshgoftaar, M. Golawala, and J. V. Hulse. An Empirical Study of Learning from Imbalanced Data Using Random Forest. In *19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007)*, volume 2, pages 310–317, October 2007.

[32] Mohiuddin Ahmed and Abdun Naser Mahmood. Novel Approach for Network Traffic Pattern Analysis using Clustering-based Collective Anomaly Detection. *Annals of Data Science*, 2(1):111–130, 2015.

[33] L. A. Maglaras and J. Jiang. Intrusion Detection in SCADA Systems Using Machine Learning Techniques. In *2014 Science and Information Conference*, pages 626–631, August 2014.

[34] S. Ramyar, A. Homaifar, A. Karimoddini, and E. Tunstel. Identification of Anomalies in Lane Change Behavior Using One-Class SVM. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 4405–4410, October 2016.

[35] E. Burnaev and D. Smolyakov. One-Class SVM with Privileged Information and Its Application to Malware Detection. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 273–280, December 2016.

[36] Hoang Anh Dau, Vic Ciesielski, and Andy Song. *Anomaly Detection Using Replicator Neural Networks Trained on Examples of One Class*, pages 311–322. Springer International Publishing, Cham, 2014.

[37] David Meyer, Friedrich Leisch, and Kurt Hornik. The Support Vector Machine Under Test. *Neurocomputing*, 55(1–2):169 – 186, 2003.

[38] Nitin Bhatia and Vandana. Survey of Nearest Neighbor Techniques. In *International Journal of Computer Science and Information Security*, volume 8, 2010.

[39] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. LOF: Identifying Density-based Local Outliers. *SIGMOD Rec.*, 29(2):93–104, May 2000.

[40] F. Amini and M. Mahdavi. Local Outlier Factor Based Cooperation Spectrum Sensing Scheme for Defending Against Attacker. In *Telecommunications (IST), 2014 7th International Symposium on*, pages 1144–1148, September 2014.

[41] Y. Fu, J. L. Zhou, and Y. Wu. Online Reactive Anomaly Detection over Stream Data. In *2008 International Conference on Apperceiving Computing and Intelligence Analysis*, pages 291–294, December 2008.

[42] F. T. Liu, K. M. Ting, and Z. H. Zhou. Isolation Forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, December 2008.

[43] P. V. Dinh, T. N. Nguyen, and Q. U. Nguyen. An Empirical Study of Anomaly Detection in Online Games. In *2016 3rd National Foundation for Science and Technology Development Conference on Information and Computer Science (NICS)*, pages 171–176, September 2016.

[44] D. Hardt. The OAuth 2.0 Authorization Framework. RFC 6749, RFC Editor, October 2012.

[45] Spotify AB. Spotify Web API Authorization Guide. `https://developer.spotify.com/web-api/authorization-guide/`. Accessed 2017-01-20.

[46] R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231, RFC Editor, June 2014.

[47] Google Cloud. BigQuery API. `https://googlecloudplatform.github.io/google-cloud-python/stable/index.html`, 2014–2017. Accessed 2017-03-31.

[48] Pedregosa et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[49] G. Lemaitre, F. Nogueira, D. Oliveira, and C. Aridas. Imbalanced Learn API. `http://contrib.scikit-learn.org/imbalanced-learn/api.html`, 2016. Accessed 2017-03-15.

[50] Rehan Akbani, Stephen Kwek, and Nathalie Japkowicz. *Applying Support Vector Machines to Imbalanced Datasets*, pages 39–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[51] Andrew Y. Ng. Developing and Evaluating an Anomaly Detection System. `https://www.coursera.org/learn/machine-learning/lecture/Mwrni/developing-and-evaluating-an-anomaly-detection-system`. Accessed 2017-02-17.

Appendix A

Algorithm: SMOTE (T, N, k) as presented by **Chawla** *et al.* [17]

```
Input  : Number of minority class samples $T$; Amount of SMOTE $N\%$; Number
         of nearest neighbors $k$
Output: $(N/100) \times T$ synthetic minority class samples

if $N < 100$ then
    Randomize the $T$ minority class samples
    $T = (N/100) \times T$
    $N = 100$
$N = (int)(N/100)$
$k$ = Number of nearest neighbors
$numattrs$ = Number of attributes
$Sample[\,][\,]$: array for original minority class samples
$newindex$: keeps a count of number of synthetic samples generated, initialized to
 0
$Synthetic[\,][\,]$: array for synthetic samples
for $i \leftarrow 1$ to $T$ do
    Compute $k$ nearest neighbors for $i$, and save the indices in the $nnarray$
    Populate($N$, $i$, $nnarray$)
end
// Function to generate the synthetic samples.
Populate($N$, $i$, $nnarray$):
while $N \neq 0$ do
    Choose a random number between 1 and $k$, call it $nn$. This step chooses one
     of the $k$ nearest neighbors of $i$.
    for $attr \leftarrow 1$ to $numattrs$ do
        Compute: $dif = Sample[nnarray[nn]][attr] - Sample[i][attr]$
        Compute: $gap$ = random number between 0 and 1
        $Synthetic[newindex][attr] = Sample[i][attr] + gap \times dif$
    end
    $newindex{+}{+}$
    $N = N - 1$
end
```