KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Machine Learning

Second cycle, 30 credits

Machine learning for detecting fraud in an API

ANNA SÁNCHEZ ESPUNYES

Stockholm, Sweden, 2022

Machine learning for detecting fraud in an API

ANNA SÁNCHEZ ESPUNYES

Master's Programme, Machine Learning, 120 credits
Date: July 4, 2022

Supervisor: Ying Liu
Examiner: Amir H. Payberah
School of Electrical Engineering and Computer Science

Abstract

An Application Programming Interface (API) provides developers with a high-level framework that abstracts the underlying implementation of services. Using an API reduces the time developers spent on implementation, and it encourages collaboration and innovation from third-party developers. Making an API public has a risk: developers might use it inappropriately. Most APIs have a policy that states which behaviors are considered fraudulent. Detecting applications that fraudulently use an API is a challenging problem: it is unfeasible to review all applications that make requests. API providers aim to implement an automatic tool that accurately detects suspicious applications from all the requesting applications.

In this thesis, we study the possibility of using machine learning techniques to detect fraud in Web APIs. We experiment with supervised learning methods (random forests and gradient boosting), clustering methods such as Density-Based Spatial Clustering of Applications with Noise (DBSCAN), and ensemble methods that combine the predictions of supervised learning methods and clustering methods. The dataset available contains data gathered when a developer creates an application and data collected when the application starts making HTTP requests. We derive a meaningful representation from the most important textual fields of the dataset using Sentence-BERT (S-BERT). Furthermore, we experiment with the predictive importance of the S-BERT embeddings. The method that achieves the best performance in the test set is an ensemble method that combines the results from the gradient boosting classifier and DBSCAN. Furthermore, this method performs better when using the S-BERT embeddings of the textual data of the applications, achieving an f1-score of 0.9896.

Keywords

Fraud detection, Machine learning, Supervised learning, Sentence-BERT, Web API

Sammanfattning

Ett API (Application Program Interface) ger utvecklare ett högnivåramverk som abstraherar den underliggande implementationen av tjänster. Användning av ett API reducerar tiden utvecklare lägger på implementation, och uppmuntrar samarbete med och innovation av tredjeparts-utvecklare. Att göra ett API publikt har ett risk: utvecklare kan utnyttja den på olämpliga sätt. De flesta API:erna har ett policy som beskriver beteenden som räknas som bedrägliga. Upptäckandet av applikationer som använder ett API på ett bedrägligt sätt är ett icke-trivialt problem, det är omöjligt att undersöka alla applikationer som skickar begäran till API:et. API leverantörerna siktar ständigt på att skapa ett automatiskt verktyg för att exakt upptäcka applikationer misstänkta för bedrägeri.

I denna avhandling undersöks möjligheten av användning av maskininlärning för att upptäcka bedrägeri i Web API. Vi experimenterar med övervakad inlärningsmetoder (random forests och gradient boosting), klustring metoder som Density-Based Spatial Clustering of Applications with Noise (DBSCAN) och ensemble metoder som kombinerar prediktionerna av övervakad inlärningsmetoder och klustring metoder. Det tillgängliga datasetet innehåller data samlat när en utvecklare skapar en applikation och när den börjar skicka HTTP begäran. Vi härleder en meningsfull representation från de viktigaste textfälten i datasetet med hjälp av Sentence-BERT (S-BERT). Dessutom experimenterar vi med den prediktiva betydelsen av S-BERT-inbäddningarna. Metoden som uppfyller den bästa prestandan i testsetet är ett ensemble metod som kombinerade resultaten från gradient boosting klassificeraren och DBSCAN. Denna metod presterar även bättre vid användning av S-BERT-inbäddnignarna av applikationernas textdata och därav uppnår ett f1-score på 0.9896.

Nyckelord

Bedrägeriupptäckt, Maskininlärning, Övervakad inlärning, Sentence-BERT, Web API

Acknowledgments

Foremost, I would like to express my deepest gratitude to Caleb Fleming and Emil Bøje Lind Pedersen for giving me this opportunity. Your invaluable and constant advice, guidance and support have been extremely useful throughout all the process. I would also like to thank all the members in modulator and playx-insights for their encouragement and Mohammad Mahid Omar for his help and support.

I would also like to thank Ying Liu and Amir H. Payberah for supervising my thesis. Their insightful suggestions and help have helped keep the project on track.

Last but not least, I would also like to thank all my friends, family, and my partner for their unconditional support.

Contents

List of Figures

List of Tables

List of acronyms and abbreviations

API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
BoW	Bag of Words
CART	Classification and Regression Trees
DBSCAN	Density-based spatial clustering of applications with noise
DoS	Denial of service
GB	Gradient Boosting
MCC	Matthews Correlation Coefficient
ML	Machine Learning
NLP	Natural Language Processing
RF	Random Forest
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SVM	Support Vector Machine
TF-IDF	Term-Frequency-Inverse Document Frequency

Chapter 1

Introduction

An Application Programming Interface (API) is a set of tools and services that allows two applications to interact with each other [1]. The goal of an API is to abstract the logic of a program and provide a high-level framework for developers that want to create an application. Using an API reduces the time spent on implementation, thanks to software component reusability.

From an API provider perspective, making an API public has multiple advantages: it encourages collaboration and drives innovation from third-party developers. Nevertheless, making an API public has a risk: developers may inappropriately use the API. An example of fraudulent behavior is performing the denial of service (DoS) attack. In this type of attack, fraudsters aim to overwhelm the system by flooding it with fake requests [2]. APIs have protection against this type of fraud with rate-limiting. The goal of rate-limiting is to limit the number of requests an application can make to the API, making it difficult to perform a DoS attack. Fraudsters have found alternatives to circumvent the rate-limiting, for example, by creating multiple applications. APIs providers aim to find an automatic tool that detects applications that fraudulently use the API.

Spotify, one of the largest audio streaming services, offers a Web API that allows third-party developers to integrate Spotify's content into their applications [3]. There is a wide range of applications that use this API, for example, applications that explore audio features or applications that customize playlist recommendations. In order for a developer to create an application in Spotify's Web API, they must first accept the developer policy [4]. This policy informs which use cases are accepted in the API and also which behaviors are considered fraudulent. An example of a prohibited use case is the creation of bots that artificially increase the number of streams of

a song.

The vast majority of applications use the API for legitimate purposes. Nevertheless, *some* applications do violate the developer policy and terms of service. Fraudulent clients are blocked from the API, removing further access. It is observed that these applications often follow similar patterns of requesting behavior and represent themselves with similar textual features that are provided both at the time of application create and throughout its operational lifecycle. This creates the possibility to use machine learning (ML) models to improve the detection of misuse in Spotify's Web API.

1.1 Problem

Web APIs are protected from fraudsters with rate-limiting. Furthermore, APIs include a developer policy that states which behaviors are considered fraudulent. In this thesis, we want to explore the opportunity of using ML to detect fraud in Web APIs. More specifically, the goal of this thesis is to answer the following question: **Is it possible to use ML models to automate the process of detecting fraud in Spotify's Web API?**

1.2 Goals

The goal of this thesis is to implement a ML system that detects fraud. This objective can be divided into the following sub-goals:

1. Applications have textual features such as the name and description of the applications. These fields have not been used in previous research in the field. The first sub-goal is to derive a meaningful representation from the text features.

2. The second sub-goal is to implement and evaluate a set of machine learning models and systems that predict whether an application is legitimate.

3. The third sub-goal is to experiment with the predictive importance of the representation of the textual features and the requesting behavior.

1.3 Delimitations

The first sub-goal of this thesis is to derive a representation from the textual features of a client. In the background chapter, we review different natural language processing techniques that are useful for solving this task. Evaluating the performance of all the presented techniques is out of the scope of this thesis. Moreover, the second sub-goal of the thesis is to implement and evaluate machine learning models that detect fraud. For this sub-goal, we consider three different algorithms. Online learning is out of the scope of the thesis.

1.4 Outline of the thesis

The thesis has the following structure:

- Chapter 2 provides a brief overview of fraud and describes the ML algorithms used in the methodology of the thesis.
- Chapter 3 explains the methodology used to solve the problem of detecting fraud.
- Chapter 4 presents the experiments and the results obtained.
- Chapter 5 presents a discussion of the results.
- Chapter 6 provides the conclusions of the thesis and the discussion of the future work.

Chapter 2

Background

This chapter describes the background knowledge necessary to understand the methodology of this thesis. Section 2.1 provides a brief overview of fraud. Section 2.2 describes the machine learning algorithms used in the methodology of this thesis, including supervised learning algorithms, unsupervised learning algorithms, and natural language processing techniques. Moreover, section 2.3 includes an overview of related work.

2.1 Fraud

The rise of the Internet has worsened an already existing problem: fraud. Collins dictionary defines fraud as ”something or someone that deceives people in a way that is illegal or dishonest” [5]. The Internet provides fraudsters with a wide range of new tools to commit fraud. There are two actions that an organization or person can take against fraud: fraud prevention and fraud detection [6]. Fraud prevention aims to stop fraud in advance. An example of a fraud prevention measure is setting a strong password. Fraud prevention is usually a trade-off between effectiveness and convenience. Fraud detection consists of detecting fraud as fast as possible. In general, fraudsters are adaptive and find ways to circumvent fraud prevention measures. In this situation fraud detection plays an important role.

Fraud in an API

An API is a software intermediary that allows two applications to interact with each other [1]. The goal of an API is to provide a high-level framework that abstracts the underlying implementation of the services. The services offered

by an API are used by software developers and not end-users. Programmers build the clients using the API by *calling* specific services, also known as endpoints.
There are three types of API release policies [7]

1. Private: the use of the API is internal.
2. Partner: the API is shared with strategic partners.
3. Public: the API is shared with everyone.

Sharing an API with a broader audience forms a community of developers that typically collaborate, helping in the development of new use-cases and driving innovation. The risk of making an API public is that hackers or fraudsters can easily commit fraud or build applications that violate the terms of the API.

Botnets are a sophisticated example of fraud in an API. A Botnet is a network of machines that attacks an API under the control of a botmaster [8]. The botnet objective is to abuse an API through denial-of-service attacks [9], financial fraud, or data scraping. APIs are protected against these attacks with rate-limiting, in which the number of calls that a specific user or client can make to the API in a specified period of time is limited. As botnets evolved, bots within a network may use different IP addresses, making it difficult to detect and rate-limit.

2.2 Machine Learning

Machine learning is the study of algorithms that can automatically improve by using data. ML algorithms are present in a wide range of applications, like computer vision, speech recognition, and medicine. An example of an application is detecting whether an image contains a cat. To classify the images successfully, the machine learning model needs to examine examples of images that contain cats and examples of images that do not have cats in it. The phase of examining pictures is known as training, and the images are known as *training set*. Once the model is trained, it needs to be evaluated on new unseen images. The set of images used for evaluation is known as *test set*. Generalization is the ability of a model to accurately predict unseen data.

There are three approaches in machine learning: supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, the training data is labeled. Using the previous example, the training data is

a set of images with the corresponding labels: 1 if the image contains a cat and 0 if contrary. In unsupervised learning the dataset is unlabeled. The goal of unsupervised learning is to cluster the unlabeled data in different groups to discover hidden patterns. The third type of machine learning problem is reinforcement learning (RL). In this type of method, a machine is trained by trial and error to maximize a reward. This thesis mainly focuses on supervised and unsupervised learning techniques and exploring RL techniques is outside the scope of this thesis.

2.2.1 Supervised learning

In this section, we review supervised learning algorithms that are used in the methodology of this thesis. The goal of supervised learning is to build a system that is able to learn the mapping between the input data and the corresponding output while accurately predicting the output for an unseen input [10]. There are two types of supervised learning problems: classification and regression. Detecting whether an image contains a cat is a classification problem: a discrete number of categories is assigned to every input. In regression problems, the output that is assigned to every input is a continuous variable. In this thesis, we use supervised learning for classification purposes to flag clients that are likely to be fraudulent.

Finding a function that maps the inputs and the target values, both capturing the singularities of the data and generalizing well to other data is a central problem in supervised learning. It is often referred to as the bias-variance trade-off. The bias error occurs when a model makes erroneous simplifying assumptions and leads to underfitted models. An example of a model that is likely to be highly biased is a linear algorithm. The variance error, however, estimates the sensitivity to changing the training set. Models that have high variance are known as overfitted. There is a trade-off between trying to minimize both errors at the same time (see Figure 2.1).

Random forests

Random forests is a supervised learning method that uses decision trees for solving classification and regression tasks. A decision tree, also known as Classification and Regression Trees (CART) is a well-known method used in statistics for solving supervised learning problems [11]. As shown in Figure 2.2, a node acts as a decision-making node based on an attribute of the data. The edges represent the decision made by the node. The final node, also known as leaf node, holds the prediction for the target variable.

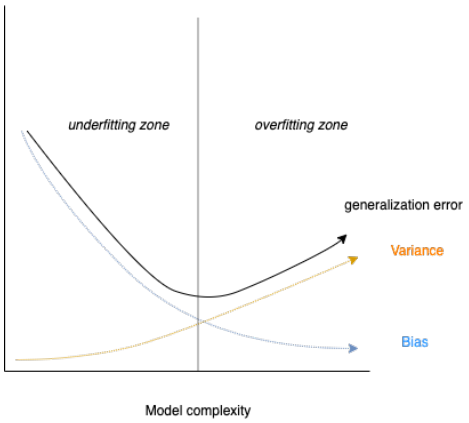

Figure 2.1: Trade-off between variance and bias error

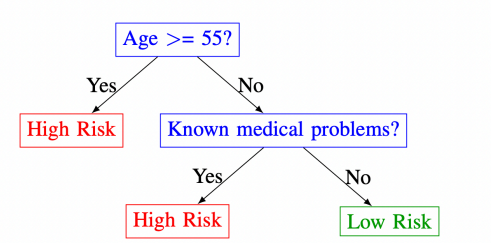

Figure 2.2: Example of decision tree for simple insurance classification [12]

Random forests are an *ensemble* learning method, which means that it uses multiple algorithms in order to achieve superior performance. In random forests, several decision trees are built, and the output of a classification task is the class that is agreed by the most number of trees. The problem with using a single decision tree for classification purposes is that it is likely to overfit. Random forests can be used as a technique to reduce variance.

Random forests use a modification of the method Bootstrap Aggregation (also known as bagging) for training b trees. In the bagging algorithm, b training subsets are created from sampling the training data with replacement. For each subset of training data, a classifier is trained. After training, the majority vote of the b trees determines the output class of a classification task. Random forests introduce a modification in the method: feature bagging. Instead of training the classifiers using all the features, random forests use a subset of features. The goal of using a subset of features is to reduce the correlation between estimators.

Gradient boosting

Another supervised learning method that uses ensemble learning and decision trees to solve classification and regression tasks is gradient boosting. In this method, several weak learner classifiers are combined to build a strong classifier. A classifier is considered weak if it performs only slightly better than a random classifier.

Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$ of n samples [13], the goal of gradient boosting is to find a function $\hat{F}(x)$ that approximates the mapping from the inputs to the outputs. In Equation 2.1 this function is expressed as the weighted sum of all weak learners $h_i(x)$.

$$\hat{F}(x) = \sum_{i=1}^{M} \gamma_i h_i(x) + \text{ const.} \tag{2.1}$$

A differentiable loss function $L(y, F(x))$ is defined where $F(x)$ represents the predicted value and y represents the observed value. The first step of the iterative algorithm is to make an initial guess expressed as

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, \gamma\right) \tag{2.2}$$

The value γ represents the first prediction. The next step is to calculate the derivative of the loss with respect to the predicted value, also known as pseudo-residuals.

$$r_{im} = -\left[\frac{\partial L\left(y_i, F\left(x_i\right)\right)}{\partial F\left(x_i\right)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n \tag{2.3}$$

The pseudo-residuals are used to build the training set $\{(x_i, r_{inn})\}_{i=1}^{n}$ that is used to train a weak learner $h_m(x)$. The next step of the algorithm is to solve the following optimization problem

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}\left(x_i\right) + \gamma h_m\left(x_i\right)\right) \tag{2.4}$$

With this calculation, the function can be updated

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) \tag{2.5}$$

The gradient boosting algorithm can be performed for $m = 1, .., M$ iterations, being M the number of created trees. The final model is given by F_M.

2.2.2 Unsupervised learning

In this section, we review unsupervised learning techniques used in the methodology of the thesis, namely clustering techniques. The goal of clustering is to group similar data points while keeping dissimilar data points in separate groups [14]. In this thesis, clustering algorithms are used to group applications and detect anomalous groups. There are five different types of clustering algorithms:

1. **Connectivity-based clustering**. This is also known as hierarchical clustering and is based on the idea that data points that are close to each other are more related than data points that are far from each other [15]. There are two types of connectivity-based algorithms: agglomerative and divisive. Agglomerative algorithms use a bottom-up approach: each sample starts having its own cluster and clusters keep merging as moving up the hierarchy. The divisive algorithms use a top-down approach. All samples start being part of one cluster and split as moving down the hierarchy.

2. **Centroid-based clustering**. In this type of clustering, the goal is to find k cluster centers that assign each data point to the closest cluster center, while minimizing the squared distances. This problem is known to be NP-hard and it is common to use approximations such as the k-means algorithm. One disadvantage of these algorithms is that the number of clusters must be defined in advance.

3. **Distribution-based clustering**. This type of clustering relies on the assumption that samples from the same cluster are from the same distribution. A well-known distribution-based algorithm is Gaussian mixture model, which relies on the assumption that the distributions are Gaussian [16]. These algorithms have a strong theoretical foundation but usually suffer from overfitting.

4. **Density-based clustering**. In this type of clustering technique, clusters are defined as areas of high density. Data samples that are in areas of low density are usually considered noise. Similar to connectivity-based clustering algorithms, close samples are clustered together, but there is also a density threshold requirement. The most known algorithm is

Density-based spatial clustering of applications with noise (DBSCAN) [17].

5. **Grid-based clustering**. Grid-base techniques divide the data space into a finite number of cells. To assign clusters, the density of each cell is calculated and then compared to the density of the neighbors. The computational complexity of these algorithms is low. A popular grid-based clustering algorithm is Statistical Information Grid Clustering Algorithm (STING) [18].

DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) is an algorithm that clusters together high-dense areas [17]. Data points in low-dense areas are considered outliers or noise. Let the parameter ϵ be the radius of the neighborhood and let the parameter minPts define the number of points that are needed to have a dense area. In this clustering algorithm, points can be classified into three types: core points, reachable points, and outliers or noise points.

- **Core points**. Points that have at least minPts points inside the radius ϵ (including the point itself).
- **Reachable points**. Points that have less than minPts points inside the radius ϵ. Apart from that, the points need to be reachable from a core point.
- **Outliers or noise points**. Points that have less than minPts points inside the radius ϵ and can not be reached from a core point.

All data points within a cluster are density-connected. Two points p and q are density-connected if there is a third data point o that makes p and q reachable from o. It is important to note that all points that are density-connected from a cluster, are also part of the cluster. The first step of the DBSCAN algorithm is to identify all core points, and afterward, identify all points that are density-connected to the core points. The next step is to assign all non-core points that are inside the radius of a cluster to a cluster. If the non-core point is not inside the radius of any cluster it gets assigned to be noise.

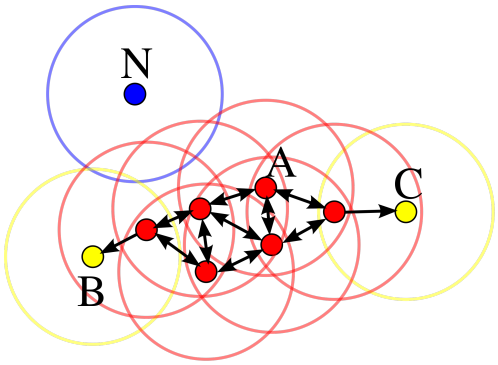

Figure 2.3: Diagram showing DBSCAN with minPts=4 [19]. The red points represent the core points, the yellow points represent reachable points and the blue point is an outlier.

2.2.3 Natural language processing

Natural Language Processing (NLP) is a field in machine learning that uses computational techniques to analyze and represent large amounts of textual data with human-like processing [20]. NLP can be applied in a wide range of applications such as:

- Information Extraction: recognizes and extracts the key elements of a text.
- Summarization: summarizes a long text into a short text.
- Machine Translation: translates text from one language to another one.
- Question-Answering: provides an answer to a user's query.

In this thesis, natural language processing is used as a technique for representing textual features, a fundamental problem that aims to numerically represent a text in order to make it mathematically computable [21].

Bag of words

A simple and effective technique for text representation is Bag-of-Words (BoW) [22]. This method uses a vector of word counts to represent a text. The following text can be represented as a vector of word counts:

I like history: I find history interesting.

`[I:2, like:1, history:2, find:1, interesting:1]`

In this method, the frequency of words is used as a weight. In the representation of the sentence, the word `history` is an important word having a weight of 2. Nevertheless, the word `I` also has a high weight, although it is just a common pronoun in English. Another shortcoming of the BoW method is that the order of words is not preserved in this model. A more sophisticated method is Term-Frequency-Inverse Document Frequency.

TF-IDF

Term-Frequency-Inverse Document Frequency (TF-IDF) is a statistical method that models the importance of a word in a document in a collection of documents [23]. TF-IDF is the multiplication of the term frequency of term t in a document d with the inverse document frequency of term t (see Equation 2.6). The term frequency is the raw count of the term t in the document d. The inverse document frequency captures how rare a is word relative to the entire document set D (see Equation 2.7). The improvement of this method with respect to BoW is that the weight of each word not only takes into account the occurrences of the word in the document but also how common a word is in the whole corpus. Although TF-IDF solves the problem of giving low importance to common words, it does not capture word order. Another disadvantage is that it is highly dependent on the corpus, and the representation of a document can be very sparse. Furthermore, similar to BoW, this technique does not capture the semantics of the words.

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D) \tag{2.6}$$

$$\text{idf}(t, D) = \ln\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right) \tag{2.7}$$

Recurrent Neural Networks

A neural network is a computer system that imitates the human brain [24]. The system consists of interconnected units named neurons that interact with each other. Nodes are organized in layers having three types of layers: the input layer, the hidden layers, and the output layer. The input layer brings the initial data to the network, whereas the output layer predicts an output. In between the input and output layer, there are the hidden layers. Information flows through the layers, going from the input layer, then the hidden layers, and finally to the output layer.

Each node in the neural network is connected to the previous layer and posterior layer. The strength of the connection is represented by a weight. The learning of a neural network consists of adjusting the weights of the connections of neurons so that the network can produce the desired output when an input is presented to the network. There are different algorithms that can be used to train a neural network, but the most popular one is backpropagation. This technique uses gradient descent for adjusting the weights.

A recurrent Neural Networks (RNN) is a type of neural network that can handle sequential data. This particularity has made RNNs well-suited for language modeling tasks and state-of-the-art for many years. RNNs are very similar to traditional neural networks, and the main difference is the presence of feedback loops. These loops allow information to flow through the time steps and enable the modeling of sequential data.

Due to this sequential nature, RNNs are impractical to parallelize, being computationally inefficient. Furthermore, RNNs are inadequate for modeling long-term dependencies due to the vanishing gradients problem [25]. In this situation, the propagated gradients in backpropagation tend to diminish, preventing the neural network weights from updating.

Transformers

Transformers solve this problem by solely relying on the attention mechanism and not using a sequential approach [26]. The attention mechanism is able to find global dependencies between the input and the output while allowing parallelization. Transformers architecture is based on the encoder-decoder structure shown in Figure 2.4. In the following paragraphs there is a description of each of the components.

The **encoder** is the component located in the left part of Figure 2.4. This block has $N = 6$ identical layers that internally have two sub-layers: a multi-head self-attention mechanism and a fully-connected neural network. Both sub-layers use residual connection succeeded by layer normalization.

The **decoder** is the component located in the right part of Figure 2.4. Like the encoder, the decoder also has $N = 6$ identical layers. The only difference is that the decoder has three sub-layers instead of two, introducing a sub-layer that performs multi-head self-attention mechanism to the output of the encoder.

Attention is one of the key mechanisms of transformers. When the model processes a word, attention looks into the other words of the sentence to take

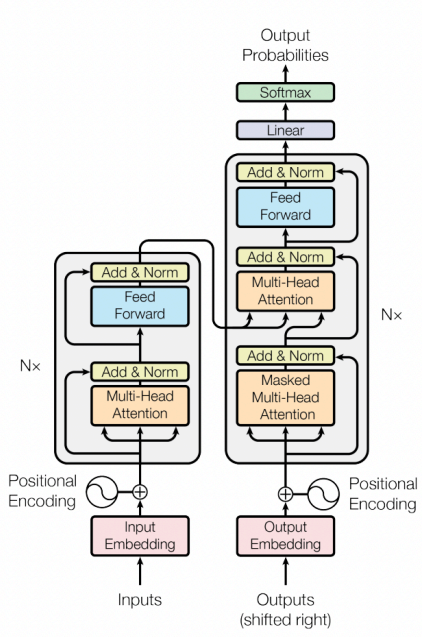

Figure 2.4: Architecture of the transformers [26]

into account their relationship with the processed word, and consequently produce a better embedding. Mathematically, attention is a function that takes three inputs (the queries Q, the keys K and the values V) to generate one output. These three inputs are calculated by multiplying the embedding of each word by three learnable matrices. [26] proposes a particular type of attention named Scaled Dot-Product Attention. In this type of attention, the output of attention is calculated as in Equation 2.8, where d_k represents the dimension of the vector of queries and keys.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.8}$$

Multi-headed attention improves attention by learning information from multiple representation subspaces. The idea is to run in parallel eight attention layers, each layer is also known as head. Each head has different learnable weight matrices, leading to different outputs. The final output of the multi-headed attention is the concatenation of the output of each head, multiplied by a learnable matrix W^O (see Equation 2.9).

$$\begin{aligned} \text{MultiHead}\,(Q,K,V) &= \text{Concat}\,(\,\text{head}_{\,1},\ldots,\,\text{head}_{\,\text{h}})\,W^{O} \\ \text{where head}_{\,\text{i}} &= \text{Attention}\,\left(QW_i^Q, KW_i^K, VW_i^V\right) \end{aligned} \tag{2.9}$$

BERT

BERT (Bidirectional Encoder Representations from Transformers) was published in 2018 by Google AI Language and produced state-of-the-art results on 11 NLP tasks [27]. The key innovation of BERT is the use of bidirectional pre-training of Transformers for language representations. Traditional directional methods read the text sequentially from left-to-right. Instead, BERT reads the text in both directions, being able to learn the context of a word by taking into account all its surrounding. The following paragraphs explain the two steps of BERT: pre-training and fine-tuning.

The **pre-training** is performed on two unsupervised tasks using unlabeled data. The first task is known as Masked LM and it involves masking a percentage of random words. The model uses these tokens as the target and tries to predict them. The second task is known as Next Sentence Prediction (NSP) and consists of predicting whether two sentences are consecutive.

The other step is **fine-tuning** of the model to a specific downstream task. By fine-tuning all parameters end-to-end, BERT is able to be fine-tuned to many downstream tasks. One advantage of decoupling the pre-training and the fine-tuning is that this last step is significantly less computationally expensive. A single pre-trained model can be fined-tuned to many different downstream tasks.

Generating a sentence embedding using BERT has been commonly done by averaging the BERT output layer, also known as BERT embedding. An alternative is using the output of the first token (CLS token). This method has demonstrated bad performance, and has inspired other methods such as Sentence-BERT [28].

Sentence-BERT is a modified version of BERT that uses siamese and triplet networks to create semantically meaningful sentence embeddings [28]. Siamese networks are a class of neural networks that have two or more identical networks, sharing the same weights. The goal is to work in tandem to produce comparable outputs and is particularly useful in similarity problems.

2.3 Related work

In this section we review related work in the field of machine learning for fraud detection.

Machine learning techniques are widely used in the field of fraud detection. A common approach when labeled data is available is to use supervised learning techniques. For instance, **Bhattacharyya et al.** [29] use support vector machines, random forests, and logistic regression to detect credit card fraud. They compare the performance of the three approaches - using a labeled dataset. The method that achieves best performance in the defined evaluation metrics is the random forests. In their method, the authors also include undersampling to reduce the class imbalance between the fraud and non-fraud class. One of the main limitations of this work is that the timestamp of the credit card transaction is unavailable and the features of the dataset are only derived from past research and not empirical results.

When labeled data is not available, it is common to use unsupervised learning techniques as labeling techniques. **Anowar et al.** [30] propose a clustering method that classifies fraud in online auctions. Since no labeled data is available, this work uses hierarchical clustering as a labeling technique. This clustering technique has advantages with respect to other clustering methods: the number of clusters does not need to be specified in advance and it handles noisy data well. The chosen labeling technique consists on classifying each fraud pattern between low or high depending on the average value in the cluster. Each fraud pattern gets assigned a weight determined by how useful the fraud pattern is. Depending on the score and the weight of the pattern, the cluster is classified as suspicious or normal. To solve the problem of imbalanced data the paper explores a wide range of sampling techniques such as SMOTE, NearMiss and ClusterCentroids [31], [32], [33]. Anowar et al. [30] use Support Vector Machines (SVM) model and Randomized Search Cross Validation (CV) to find the best data sampling technique.

A common method in the field of fraud detection is to combine the predictions of a supervised learning method and unsupervised learning method using an ensemble method. This strategy is inspired by the *best-of-both-worlds* principle proposed by Micenková et al. [34]. Classification algorithms tend to perform better than clustering algorithms when predicting the class of a sample. Nevertheless, classifiers do not perform well when the labeled data available is not reliable. Clustering algorithms do not directly assign a label to every cluster, but since similar samples are clustered together, one could assign the same label to a cluster. Following this approach,

Chakraborty [35] proposes an ensemble method named *EC3* that merges the predictions of clustering and classification algorithms. Furthermore, the paper also introduces a modification of EC3 for imbalanced datasets called *iEC3*. The new proposed methods outperform all the 12 defined baselines (including standalone classifiers and homogeneous ensemble classifiers) in all the datasets. Similarly, **Carcillo et al.** [36] also implement a hybrid method that combines supervised and unsupervised learning techniques to detect credit card fraud. The paper combines both methods by calculating outlier scores at different levels of granularity, and including these features in the supervised learning classifier.

Previous work in fraud detection of an API focuses on traffic anomaly detection. **Shon et al.** [37] use a hybrid machine learning approach to distinguish between normal and fraudulent traffic in a network by using Support Vector Machines (SVMs) - a widely used technique in the field of anomaly detection. Nevertheless, this supervised learning method requires labeled data which is not available. A solution to this problem is the one-class SVM, an unsupervised learning technique that does not require labeled data. The downside of this method is that it produces a high false positive rate. The paper proposes an *Enhanced SVM*, an unsupervised learning method that is able to produce a low false positive rate. The method is a combination of self-organized feature map, passive TCP/IP stack fingerprinting and genetic algorithms.

Kromkowski et al. [38] use advanced statistical models to detect anomalies in the network traffic. They compare the performance of a Seasonal Autoregressive Integrated Moving Average (SARIMA) times series model and Long Short-Term Memory (LSTM) Autoencoder model for anomaly detection. Since both models report high false positive rates, the authors propose to ensemble both models. The results of the ensemble method significantly reduce the false positive rate, by only predicting the data as anomalous if both individual models flag it as anomalous.

Baye et al. [39] use a Gaussian distribution approach for outlier detection in the context of an API. This method labels as *normal* all the traffic within a standard deviation range of a Gaussian of all traffic. All traffic not included within a standard deviation range is considered *abnormal*. The labeled dataset is used to train a SVM classifier using the following features: bandwidth consumption, IP address, connection duration, number of consecutive requests, number of HTTP requests and number of HTTP responses. The method achieves a f1-score of 0.964 in detecting anomalies in the API.

Jaward [40] explores three supervised learning algorithms to detect scraping in Spotify's Web API. The data used to train the algorithms is the aggregated HTTP request data over the span of two weeks. Some of the relevant features that are considered are the total number of requests, the percentage of requests to each endpoint, the average request rate, the variance of request rate, the number of users with premium and free subscription and the number of unique IP. Since not enough labeled data is available, the work uses synthetic data. This work also experiments with oversampling the minority class using SMOTE - although the results show that this method does not improve performance. The model that achieves best performance is the random forests with a Matthews Correlation Coefficient (MCC) of 0.692 [41].

Chapter 3

Methodology

The purpose of this chapter is to provide an overview of the methodology used in this thesis. Section 3.1 provides an overview of the problem and the methodology. Section 3.2 focuses on describing the data available. Section 3.3 details the architecture. Section 3.4 describes the framework selected to evaluate the proposed methodology.

3.1 Overview

We want to investigate the opportunity for Web APIs to employ ML to automatically detect fraud by using the information on previous fraudulent applications. More specifically, the goal of this thesis is to answer the following research question: *Is it possible to use machine learning models to automate the process of detecting fraud in Spotify's Web API?*.

The dataset available in this thesis is a combination of numerical and textual features that describe important characteristics of each application. There are two textual fields, the name and the description of the client, that are useful for fraud detection. These fields provide information about the use-case of the application, and consequently are important to incorporate in the machine learning model. For this reason, we need to find a numerical representation for these fields.

The main requirement for the final machine learning model is that it needs to have the right balance between precision and recall, meaning that we want to minimize as much as possible the need for manual reviews of fraudulent applications, but at the same time, we want to flag all possible fraudulent clients. As seen in the related work section, a common approach to address this requirement in fraud detection is to use ensemble methods. This technique

combines the predictions of multiple individual models in order to achieve better performance and more robustness than using the individual models.

Inspired by [35], [42], [43] we combine the predictions of a supervised learning classifier and a clustering model in an ensemble method. Supervised learning models tend to perform better than unsupervised learning models in classification problems. Nevertheless, supervised learning classifiers make the assumption that samples are drawn from an independent and identical distribution (i.i.d), meaning that the inter-dependencies between samples are not considered [44]. On the other hand, clustering methods consider the relationship between samples, although a label is not directly assigned. Combining the predictions of both methods in an ensemble method can yield a better overall performance.

The overall summary of the methodology is the following:

1. **Textual Representation**. Two of the most important features used for fraud detection are the name and description of an application. The goal of this block is to generate a meaningful data representation of these fields.

2. **Data preprocessing**. The goal of this block is to perform data cleaning and data transformation to the rest of the features.

3. **Supervised learning classifier**. The main objective is to find clients that are likely to be fraudulent based on past data of fraudulent clients.

4. **Clustering classifier**. The objective is to group similar applications into clusters. All clients that belong to a cluster that contains more than 90% of fraudulent clients are flagged as fraudulent.

5. **Ensemble method**. This block combines the predictions of the classification block and the clustering block. An application gets predicted as fraudulent depending on the prediction of the supervised learning classifier and the clustering block.

6. **Evaluation of the system**. The goal is to evaluate the system by using metrics such as accuracy, precision, recall, and f1-score.

3.2 Dataset

This section describes the dataset used to train the models used for fraud detection. The dataset consists of the most relevant features used for detecting fraud.

The dataset is the combination of fraudulent clients and clients we hypothesize are not fraudulent. It includes all the clients that have been flagged as fraudulent in the past, as well as all clients that have been active for more than one year and have not been flagged as fraudulent. We assume with very high confidence that these active, unflagged clients are not fraudulent. We split the dataset into 70 % of training data and 30 % of test data. The split is based on clients and the class distribution between classes is the same in both sets.

Each row of this dataset represents the information of a client. There are two types of features: the features we gather when the application is created, and the features that are obtained when the application starts making requests to the API. When a user creates a client for use with the API they must specify a name and description for their client. Once the application is created, the owner may specify a *redirect uri*. This is the location or address where the user is redirected after the authorization server has granted authorization to the application.

Other features that can be derived at the moment of creation of a client are related to the owner of the application. We derive features that indicate whether the owner's email is valid, verified, and temporary, as well as the number of applications that are owned by the owner. A user that creates fraudulent clients tends to use invalid emails, not verified and/or temporary emails. Furthermore, a user that creates a high amount of clients is likely to create a botnet. It is also useful to know the number of clients created in the same hour as the application. A high number may indicate the creation of a botnet.

Other features are obtained when the application starts making requests to the API. We derive features regarding the monthly number of GET, PUT and POST requests to all the different endpoints. The hypothesis is that fraudulent clients tend to have similar behavior and consequently request the same endpoints with similar frequency. From the requesting data, we also derive the service class feature. A service class represents the parent grouping of similar endpoints. For example, a fictional "playlist service" might contain endpoints allowing the requester to retrieve playlist metadata, create or update new or existing playlists, subscribe to another user's playlist, etc. If a client makes more than 70% of their total requests to endpoints belonging to the same service, this service is assigned as the service class of the client. From the requesting data, we can also derive application daily active users (DAU) and the number of subscribed / unsubscribed users using the application.

A summary of all the described features can be found in Table 3.1, as well

as the data type of each feature.

Table 3.1: Summary of the features from the dataset

Name of the feature	Description of the feature
client_id	Unique identifier of the application (String)
name	Name of the application (String)
description	Description of the application (String)
n_redirect_uris	# clients sharing redirect uri (Integer)
is_email_valid	0 if email is not valid, 1 otherwise (Boolean)
is_email_verified	0 if email is not verified, 1 otherwise (Boolean)
is_email_temporary	0 if email is not temporary, 1 otherwise (Boolean)
n_clients_owner	# clients owned by the owner (Integer)
days_owner_client_create	# days between creation of owner and client (Integer)
n_clients_create_hour	# clients created the same hour (Integer)
GET_endpoint_X	# GET requests to a specific endpoint X (Integer)
PUT_endpoint_X	# PUT requests to a specific endpoint X (Integer)
POST_endpoint_X	# POST requests to a specific endpoint X (Integer)
service_class	Service class (String)
current_dau	Current daily active users using the client (Integer)
premium_prod_users	Number users using the premium product (Integer)
free_prod_users	Number of users using the free product (Integer)
deleted	0 if client is not fraudulent, 1 otherwise (Boolean)

3.3 Architecture

In this section we present the architecture used to detect misuse. The high-level overview of the system can be seen in Figure 3.1.

Textual representation block

The goal of this block is to find a representation of the textual features of a client, namely using the name and the description. Applications having a similar name and description should have a close representation, whereas clients with a different name and description should have a distinct representation.

In section 2.2, we reviewed the most important methods for data representation such as TF-IDF. The main disadvantage of this method is that the word order and the context of the word are not considered in the data representation. This requirement is important when finding a meaningful

representation of the name and the description of a client. In section 2.2, we also reviewed RNNs as a technique for data representation. The main shortcoming of this method is the inability to model long-term dependencies, as well as being computationally inefficient.

Sentence-BERT, a modified version of BERT, is the chosen method for representing the name and the description of a client. This technique is able to find semantically meaningful sentence embeddings, taking into account the word order and the context. More specifically, we use a pre-trained Sentence-BERT model named *distiluse-base-multilingual-cased-v1*. This pre-trained model is multilingual, meaning that similar inputs in different languages have an embedding that is close in the vector space. Our data includes more than 20 languages, although the main language is English. This pre-trained model is optimized for 15 languages, including the languages present in our dataset.

As seen in Figure 3.1, the input of this block is the name and the description of a client. The first step is to concatenate both fields and input the concatenation to *distiluse-base-multilingual-cased-v1*. The output of this block is the sentence embedding of both the name and the description of length 512. Another alternative could be to derive two different sentence embeddings for the name and the description. This alternative is not explored during this thesis due to time constraints but could be one path for future work.

Data preprocessing

The goal of this block is to derive a clean dataset from the data available. The steps are the following:

1. Data cleaning: fill the missing values from the numerical features with 0.

2. One-hot encoding: this step one-hot-encodes the categorical feature *service_class*.

3. Scaler: this step scales the features individually by the maximum absolute value of the training set. The same scaler is used to scale the test set.

Supervised learning classifier

This block aims to find which clients are likely to be fraudulent based on past data. As described in section 3.2, we have labeled data available of clients that are fraudulent and clients that are very likely to be non-fraudulent. By using

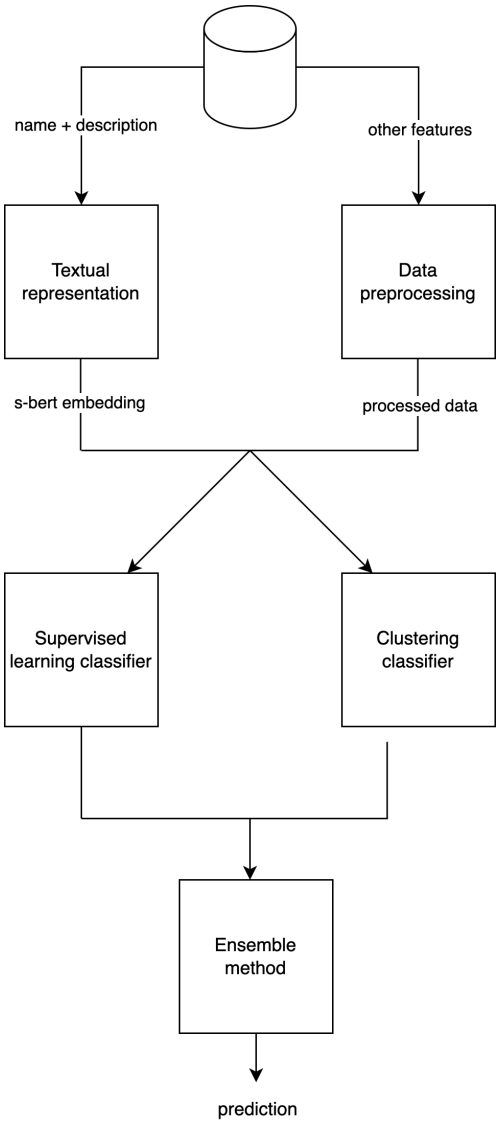

Figure 3.1: Architecture of the method including the textual representation block, data preprocessing block, supervised learning classifier, clustering classifier and ensemble method block.

this data, the goal of this block is to train a classifier that is able to distinguish between fraudulent and non-fraudulent clients.

When training the supervised learning classifier, the input data is the concatenation of the S-BERT embedding and the processed data. We will run different experiments on which features should be included as part of the input data. The output of this block is the prediction of the classifier on whether a

specific client is fraudulent. In section 2.2, we reviewed different methods for classification. In this block, we will experiment with random forests and gradient boosting, two methods that have shown good performance in similar problems (see section 2.3).

Random forests algorithm is an ensemble learning method that fits a number of decision trees. Random forests have a number of hyper-parameters need to be specified in order to build the model.

- Number of estimators: number of decision trees used in the algorithm. Using a large number of estimators does not degrade the performance of the model, but increases the time complexity of the algorithm. In this thesis we have experimented with 60, 80, 100.

- Maximum depth: this hyper-parameter defines the maximum depth of a decision tree. Depth is defined as the largest path between the root node and the leaf node. In this thesis we have experimented with depth two, depth three and depth four.

- Minimum samples required to split an internal node: this parameter defines the minimum number of observations required in a node in order to be able to split. For example, in case this parameter is equal to three, that means that if a node has more than three samples, and it is not a pure node, then it can be split into further nodes. In this thesis we have experimented with minimum samples equal to two and three.

- Information gain criterion. This criterion defined the function used to measure the quality of the split. In this thesis we only experiment with gini.

Gradient boosting algorithm also has some hyper-parameters that also need to be specified.

- Loss: defines the loss function that needs to be optimized. In this thesis we only experiment with logistic loss.

- Learning rate: parameter used to scale the contribution of each of the individual trees. The weight, also known as shrinkage is the learning rate. In this thesis we experiment with a learning rate of 0.25, 0.5, 1.0.

- Number of estimators: number of decision trees used in the algorithm. In this thesis we experiment with 60, 80, 100.

- Maximum depth: defines the maximum depth of a decision tree. In this thesis we experiment with depth two, depth three and depth four.
- Minimum samples required to split an internal node: minimum number of observations required in a node in order to be able to split. In this thesis we only experiment with minimum samples equal to two.

After the models are trained, we evaluate the performance of each model. We input the test data to the model, and predict the output class for each individual client. The performance of the model can be computed when comparing the prediction of the model and the true label of each application.

Clustering classifier

This block groups together similar clients based on the data available. The objective is to analyze the output clusters, more specifically all clusters that contain fraudulent clients. If a cluster has 90% or more fraudulent clients, then all the non-fraudulent clients within the cluster are suspected of fraud.

To evaluate this block, we input the training set and the test set to the DBSCAN algorithm. Taking into account the formed clusters, we calculate for each cluster the percentage of fraudulent clients in the train set out of all training set clients. If the percentage of fraudulent clients is higher or equal to 90%, then all the test clients in the cluster are flagged as fraud. If the cluster has less than 90% of fraudulent clients, the test set clients are flagged as non-fraudulent.

One of the main challenges of clustering is to determine the number of clusters in advance. In the context of this problem, this number depends on a wide range of factors such as seasonal trends or even arbitrary reasons. This is the main motivation of the methodology chosen for the clustering block: DBSCAN. As described in section 2.2, this clustering algorithm does not require the number of clusters in advance. Another advantage of this method is that not all samples are assigned to a cluster. This is important in the context of this problem: our main goal is to detect groups of fraudulent clients. DBSCAN provides a strong method for grouping clients but does not force clients to belong to a cluster.

DBSCAN algorithm has a number of hyper-parameters need to be specified in order to build the model.

- Epsilon: defines the maximum distance between two samples to be considered in the same neighborhood. In this thesis we experiment with an epsilon of 0.1, 0.3 and 0.5.

- Number of samples: defines the minimum number of samples in a neighborhood of a point in order to be considered a core point. In this thesis we experiment with 10, 20, 30.
- Distance metric: defines the metric used when calculating distances. In this thesis we only experiment with euclidean distance metric.

Ensemble method

The goal of the ensemble method is to combine the output of the supervised learning classifier and the clustering classifier to improve the results of the individual models. The selected approach for the ensemble method is the majority vote; the final prediction is the one that receives more votes from the individual models.

If both models predict that a client is fraudulent, the output of the ensemble is the positive class. Similarly, if both models predict that a client is non-fraudulent, the final output is the negative class. There are two different configurations in case the individual models do not predict the same class. One option (config 1) is that the ensemble method only predicts that a client is fraudulent only if both individual models predict the positive class for that client. The other configuration of the ensemble (config 2) predicts a positive class in case one of the individual models predicts that a client is fraudulent. In this thesis we experiment with both configurations (see Table 3.2).

Table 3.2: Possible results of the ensemble from individual models

Output SL (1)	Output UL (2)	(3) Ensemble config 1	(4) Ensemble config 2
Fraud	Fraud	Fraud	Fraud
Fraud	Not fraud	Not fraud	Fraud
Not fraud	Fraud	Not fraud	Fraud
Not fraud	Not fraud	Not fraud	Not fraud

3.4 Evaluation framework

In this section, the evaluation metrics are described. The results of evaluating a binary classification can be divided into 4 groups: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). An outcome is true positive if the model correctly predicts the positive class. An outcome is a true negative if the model correctly predicts the negative class. In terms of

errors in predictions, we have two types: false positives and false negatives. An outcome is a false positive if the model incorrectly predicts the positive class, for instance by predicting that a client is fraudulent when it is not. A false negative is the opposite of this error; the model incorrectly predicts the negative class, for example by predicting that a client is not fraudulent when it actually is.

We use the following evaluation metrics: accuracy, precision, recall, and F1 score. Accuracy is the fraction of correct predictions out of all the predictions made by the model.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision is the fraction of true positives out of all the samples that were predicted as positives and is a useful metric when the cost of a false positive is very high.

$$Precision = \frac{TP}{TP + FP}$$

Recall, also known as sensitivity, is the fraction of true positives out of all positive samples. This metric measures the ability of the model to identify relevant data.

$$Recall = \frac{TP}{TP + FN}$$

There is a trade-off between precision and recall. That means that one metric can be improved at the expense of worsening the other. F1 score considers both precision and recall, and it is the harmonic mean of precision and recall. It is an evaluation metric that is used when a balance between precision and recall is needed.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN}$$

Chapter 4

Experiments and Results

Chapter 3 provided an overview of the methodology. In this chapter we experiment with different models and features in order to quantify their impact when detecting fraud in an API.

The architecture described in chapter 3 consists of an ensemble method that combines the predictions of a supervised learning classifier and a clustering classifier. Due to time complexity, we evaluate the performance of the blocks in the different experiments separately. All the experiments are evaluated in the same test set. The original dataset is split into 70% training data and 30% test data and the evaluation is performed in the metrics described in section 3.4.

4.1 Supervised learning block

The two algorithms considered for the supervised learning classifier are the random forest (RF) classifier and the gradient boosting (GB) classifier. After a grid search, the best performing hyper-parameters for RF are 100 estimators, a maximum depth of 4 and the minimum samples required to split an internal node equal to 2. The optimal hyper-parameters for the GB classifier are 100 estimators, a learning rate of 1.0 and a maximum depth of 2.

We perform three different experiments regarding the selection of features of the classifier.

1. In the first experiment we only consider the Sentence-BERT embedding of the name and the description as the input of the classifier.

2. In the second experiment we use all available features except the Sentence-BERT embedding of the name and description.

3. In the third experiment consists of using all available features.

The results of the the three experiments using the RF and GB can be found in Table 4.1.

Table 4.1: Results of the supervised learning experiments

Results	RF (1)	GB (1)	RF (2)	GB (2)	RF (3)	GB (3)
Accuracy	0.9754	0.9827	0.9602	0.9867	0.9827	**0.9904**
Precision	0.9791	0.9769	0.9944	0.9844	**0.9951**	0.9882
Recall	0.9522	0.9753	0.8951	0.9786	0.9567	**0.9853**
F1-score	0.9655	0.976	0.9421	0.9815	0.9756	**0.9867**

We now compare the performance results of gradient boosting and random forests. In the first experiment gradient boosting obtains better performance than random forests in terms of accuracy, recall and f1-score. Similarly, when taking into account all features except the Sentence-BERT embedding (experiment two), gradient boosting also obtains better performance than random forests in accuracy, recall and f1-score. In the third experiment, we also observe that random forests only perform better than gradient boosting in terms of precision.

The best results are achieved when we use as input data both behavioral features and Sentence-BERT embedding of the name and the description of the clients. More specifically, for experiment three we obtain a f1-score of **0.9867** when using gradient boosting.

4.2 Unsupervised learning block

The algorithm considered for the unsupervised learning block is DBSCAN. The hyper-parameters used for this algorithm are an epsilon of 0.5, a minimum number of samples in a neighborhood of 10 and an euclidean distance metric.

We perform four different experiments regarding the selection of features of the clustering algorithm.

1. In the first experiment we only consider the Sentence-BERT embedding of the name and the description as the input of the clustering algorithm.

2. In the second experiment we only consider the 2-dimensional representation of the Sentence-BERT embedding of the name and the description as the input of the clustering algorithm.

3. In the third experiment we use all available features except the Sentence-BERT embedding of the name and description.

4. In the fourth experiment we use the 2-dimensional Sentence-BERT embedding of the name and the description, in composition with the rest of the available features.

The results of these experiments can be found in Table 4.2.

Table 4.2: Results of the clustering experiments

Results	DBSCAN (1)	DBSCAN (2)	DBSCAN (3)	DBSCAN (4)
Accuracy	0.6531	0.968	0.7567	**0.9741**
Precision	0.9781	0.968	0.9812	**0.9946**
Recall	0.0419	**0.9426**	0.3339	0.9334
F1-score	0.0804	0.9551	0.4982	**0.963**

The results show that using the Sentence-BERT embedding as the input data yields to poor performance. When using a low-dimensional representation of the embedding the performance significantly improves in terms of accuracy, precision and f1-score. The results of the third experiment, when not considering the Sentence-BERT embedding, are low compared to the results in the second experiment. We obtain the best results in terms of accuracy, precision and f1-score when considering all the features, including the Sentence-BERT embedding. We achieve a f1-score of **0.963**.

4.3 Ensemble block

In a hard voting classifier, also known as majority voting, every individual model votes for a class and the majority wins. We experiment in four different ways to combine the output of the individual classifiers.

1. The first setting consists of using the same output as the supervised learning classifier that obtained best performance.

2. The second setting consists of using the same predictions as the clustering block that obtained best performance.

3. The third configuration only flags a client as fraudulent if both blocks have flagged the client as fraudulent.

4. The last configuration flags a client as fraudulent if one or both models have flagged the client as fraudulent.

The results from these experiments can be found in Table 4.3.

Table 4.3: Results from experiments ensemble method

Results	SL (1)	UL (2)	Ensemble (3)	Ensemble (4)
Accuracy	0.9904	0.9741	0.9672	**0.9925**
Precision	0.9882	0.9946	**0.999**	0.9857
Recall	0.9853	0.9334	0.9103	**0.9936**
F1-score	0.9867	0.963	0.9526	**0.9896**

The best results across all metrics are achieved when combining the supervised learning classifier and the clustering results. More specifically, the configuration that achieves higher performance in terms of f1-score is when the ensemble flags a client as fraudulent if both models have predicted the positive class for the application. This configuration achieves a f1-score of **0.9896**.

Chapter 5

Discussion

In this chapter, we provide a discussion of the results reported in chapter 4.

5.1 Supervised learning block

Gradient boosting performs significantly better than random forests in accuracy, recall and f1-score in all the performed experiments. Nevertheless, random forests perform slightly better than gradient boosting in terms of precision. As explained in section 3.4, there is a trade-off between precision and recall - and the f1-score metric considers a balance between both metrics. In the context of our problem, there is a balance between detecting all fraudulent clients and limiting the number of manual revisions. This is the main reason that motivates the choice of gradient boosting as the algorithm for the supervised learning classifier.

In terms of features, the results of the experiments show that the Sentence-BERT embedding of the name and description has a high predictive performance when predicting whether a client is fraudulent. If we compare the results obtained when using all features except the Sentence-BERT embeddings (experiment 2) and using all features including Sentence-BERT embeddings (experiment 3), we observe a significant increase in all the performance metrics in experiment 3. Due to these results, all features are considered as the input to the supervised learning classifier, including the Sentence-BERT embeddings.

5.2 Unsupervised learning block

The results from Table 4.2 show that DBSCAN using the high-dimensional Sentence-BERT embedding yields to poor performance across all metrics. When analyzing the results of the clusters for this experiment, we observe that most of the positive samples do not get assigned into a cluster, and are considered outliers or noise points. Since all noise points are not considered fraudulent, the reported results for accuracy are very low. One hypothesis that may explain the results is that DBSCAN for the chosen distance metric is not able to find high-dense areas within the high-dimensional embeddings. This hypothesis motivates the following experiment that applies DBSCAN to the low-dimensional representation of these embeddings.

The results of applying DBSCAN to the 2-dimensional Sentence-BERT embeddings show a significant performance improvement compared to using the high-dimensional embeddings. DBSCAN can find high-density areas containing fraudulent applications, and only a few fraudulent clients do not get assigned to a cluster.

In the next experiment, we apply DBSCAN to all available features except textual embeddings. The results report poor performance in terms of accuracy, recall, and f1-score, proving that these features are not enough to assign all fraudulent clients to clusters. The high precision shows that these features are useful when clustering a specific type of fraud. These results motivate the next experiment: using both the low-dimensional Sentence-BERT embeddings and the rest of the features. The results of this last experiment show the best performance in terms of accuracy, precision, and f1-score, motivating the choice of these features as the input of the clustering classifier.

5.3 Ensemble block

The best performance in terms of precision is achieved when the ensemble method flags a client as fraudulent only in the case of positive class in both the individual models. This result is expected since precision measures the fraction of relevant samples among the retrieved instances. In this setting since both individual models need to flag the application as fraudulent, the precision is very high (0.999). The downside of having a high precision is that the recall is significantly low, having a high number of false negatives.

Since we want to have a balance between precision and recall, we will analyze the f1-score results. The best performance in terms of accuracy, recall,

and f1-score is achieved when the ensemble method flags a client as fraudulent if any of the individual blocks predicts a positive class for the application (see Table 4.3). The performance in these metrics is superior to any of the individual models, proving that the ensemble method improves the prediction performance. Due to the results, the chosen configuration is an ensemble method that flags a client as fraudulent when any of the individual models predicts a positive class for the application.

Chapter 6

Conclusions

Making an API public helps developers reduce the time spent on implementing, but comes with the risk that developers might attempt to use the API in a way that is violating of the terms of service. This research aims to implement a machine learning system that detects misuse in the context of an API, making use of the available labeled dataset containing textual and behavioral information of the fraudulent applications.

We experiment with the use of supervised learning techniques (random forests and gradient boosting), clustering techniques (DBSCAN), and ensemble methods that combine the predictions of the supervised learning classifier and the clustering algorithm. Furthermore, we also experiment with the use of different features, more specifically with the predictive importance of the Sentence-BERT embeddings derived from the textual data of the dataset.

The results of the experiments are evaluated on a test set in four different metrics: accuracy, precision, recall, and f1-score. The method that achieves the best performance is an ensemble method that both takes into account the predictions of a gradient boosting classifier and the predictions of DBSCAN. Furthermore, this method achieves the best results when the behavioral data is used in combination with the textual Sentence-BERT embeddings – achieving an f1-score of 0.9896 and proving the predictive importance of these features.

6.1 Future work

In this section, we will focus on future work. The performance of the proposed model is very high, which proves that the proposed method is able to detect fraud accurately. Nevertheless, there are several areas that could help achieve even a better performance.

The immediate next step is to focus on extracting more features from the requesting data. For instance, focusing on modeling the IP, the user agent or model the request rate of a client [45]. Additionally, the overall quality of the dataset could be improved by manually labeling non-fraudulent clients instead of making the assumption that if a client has not been flagged as fraudulent in the past year then it is from the negative class. Another direction could be to explore techniques to deal with positive and unlabeled data (PU learning). This type of method is widely used in fraud detection and could improve the performance of the model.

In this work, we have experimented with Sentence-BERT embeddings as a method to represent textual data. Future work could explore if better performance is achieved in the case of fine-tuning Sentence-BERT or if a different textual representation method is used. Also, future work could experiment with obtaining separate Sentence-BERT embeddings for the name and description. Future work could explore other clustering techniques apart from DBSCAN, as well as over-sampling and under-sampling techniques for imbalanced datasets.

The downside of relying on labeled data to detect fraud is that we will only detect similar fraudulent applications to the ones that have already been flagged. Future work could emphasize unsupervised learning approaches to detect anomalous behavior.

References

[1] M. Reddy, *API Design for C++*. Elsevier, 2011. [Pages 1 and 5.]

[2] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni, "Analysis of a denial of service attack on tcp," in *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No. 97CB36097)*. IEEE, 1997, pp. 208–223. [Page 1.]

[3] "Web api | spotify for developers," https://developer.spotify.com/documentation/web-api/, (Accessed on 05/21/2022). [Page 1.]

[4] "Spotify developer policy | spotify for developers," https://developer.spotify.com/policy/, (Accessed on 05/21/2022). [Page 1.]

[5] "Fraud definition and meaning | collins english dictionary," https://www.collinsdictionary.com/dictionary/english/fraud, (Accessed on 05/21/2022). [Page 5.]

[6] R. J. Bolton and D. J. Hand, "Statistical fraud detection: A review," *Statistical science*, vol. 17, no. 3, pp. 235–255, 2002. [Page 5.]

[7] "What is an api?" https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces, (Accessed on 02/27/2022). [Page 6.]

[8] M. Feily, A. Shahrestani, and S. Ramadass, "A survey of botnet and botnet detection," in *2009 Third International Conference on Emerging Security Information, Systems and Technologies*. IEEE, 2009, pp. 268–273. [Page 6.]

[9] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, "Inferring internet denial-of-service activity," *ACM Transactions on Computer Systems (TOCS)*, vol. 24, no. 2, pp. 115–139, 2006. [Page 6.]

[10] P. Cunningham, M. Cord, and S. J. Delany, "Supervised learning," in *Machine learning techniques for multimedia*. Springer, 2008, pp. 21–49. [Page 7.]

[11] W.-Y. Loh, "Classification and regression trees," *Wiley interdisciplinary reviews: data mining and knowledge discovery*, vol. 1, no. 1, pp. 14–23, 2011. [Page 7.]

[12] S. Suthaharan, "Decision tree learning," in *Machine Learning Models and Algorithms for Big Data Classification*. Springer, 2016, pp. 237–269. [Pages vii and 8.]

[13] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794. [Page 9.]

[14] I. H. Witten and E. Frank, "Data mining: practical machine learning tools and techniques with java implementations," *Acm Sigmod Record*, vol. 31, no. 1, pp. 76–77, 2002. [Page 10.]

[15] O. Maimon and L. Rokach, "Data mining and knowledge discovery handbook," 2005. [Page 10.]

[16] G. Xuan, W. Zhang, and P. Chai, "Em algorithms of gaussian mixture model and hidden markov model," in *Proceedings 2001 International Conference on Image Processing (Cat. No. 01CH37205)*, vol. 1. IEEE, 2001, pp. 145–148. [Page 10.]

[17] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "Density-based spatial clustering of applications with noise," in *Int. Conf. Knowledge Discovery and Data Mining*, vol. 240, 1996, p. 6. [Page 11.]

[18] W. Wang, J. Yang, R. Muntz *et al.*, "Sting: A statistical information grid approach to spatial data mining," in *Vldb*, vol. 97. Citeseer, 1997, pp. 186–195. [Page 11.]

[19] "Dbscan - wikipedia," https://en.wikipedia.org/wiki/DBSCAN, (Accessed on 03/17/2022). [Pages vii and 12.]

[20] E. D. Liddy, "Natural language processing," 2001. [Page 12.]

[21] N. Liu, B. Zhang, J. Yan, Z. Chen, W. Liu, F. Bai, and L. Chien, "Text representation: From vector to tensor," in *Fifth IEEE International*

Conference on Data Mining (ICDM'05). IEEE, 2005, pp. 4–pp. [Page 12.]

[22] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: a statistical framework," *International Journal of Machine Learning and Cybernetics*, vol. 1, no. 1, pp. 43–52, 2010. [Page 12.]

[23] A. Rajaraman and J. D. Ullman, *Mining of massive datasets*. Cambridge University Press, 2011. [Page 13.]

[24] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, p. e00938, 2018. [Page 13.]

[25] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001. [Page 14.]

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017. [Pages vii, 14, and 15.]

[27] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018. [Page 16.]

[28] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019. [Page 16.]

[29] S. Bhattacharyya, S. Jha, K. Tharakunnel, and J. C. Westland, "Data mining for credit card fraud: A comparative study," *Decision support systems*, vol. 50, no. 3, pp. 602–613, 2011. [Page 17.]

[30] F. Anowar and S. Sadaoui, "Detection of auction fraud in commercial sites," *Journal of theoretical and applied electronic commerce research*, vol. 15, no. 1, pp. 81–98, 2020. [Page 17.]

[31] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002. [Page 17.]

[32] I. Mani and I. Zhang, "knn approach to unbalanced data distributions: a case study involving information extraction," in *Proceedings of workshop on learning from imbalanced datasets*, vol. 126. ICML, 2003, pp. 1–7. [Page 17.]

[33] M. M. Rahman and D. Davis, "Cluster based under-sampling for unbalanced cardiovascular data," in *Proceedings of the World Congress on Engineering*, vol. 3, 2013, pp. 3–5. [Page 17.]

[34] B. Micenková, B. McWilliams, and I. Assent, "Learning outlier ensembles: The best of both worlds–supervised and unsupervised," in *Proceedings of the ACM SIGKDD 2014 Workshop on Outlier Detection and Description under Data Diversity (ODD2). New York, NY, USA*. Citeseer, 2014, pp. 51–54. [Page 17.]

[35] T. Chakraborty, "Ec3: Combining clustering and classification for ensemble learning," in *2017 IEEE international conference on data mining (ICDM)*. IEEE, 2017, pp. 781–786. [Pages 18 and 22.]

[36] F. Carcillo, Y.-A. Le Borgne, O. Caelen, Y. Kessaci, F. Oblé, and G. Bontempi, "Combining unsupervised and supervised learning in credit card fraud detection," *Information sciences*, vol. 557, pp. 317–331, 2021. [Page 18.]

[37] T. Shon and J. Moon, "A hybrid machine learning approach to network anomaly detection," *Information Sciences*, vol. 177, no. 18, pp. 3799–3821, 2007. [Page 18.]

[38] P. Kromkowski, S. Li, W. Zhao, B. Abraham, A. Osborne, and D. E. Brown, "Evaluating statistical models for network traffic anomaly detection," in *2019 systems and information engineering design symposium (SIEDS)*. IEEE, 2019, pp. 1–6. [Page 18.]

[39] G. Baye, F. Hussain, A. Oracevic, R. Hussain, and S. A. Kazmi, "Api security in large enterprises: Leveraging machine learning for anomaly detection," in *2021 International Symposium on Networks, Computers and Communications (ISNCC)*. IEEE, pp. 1–6. [Page 18.]

[40] D. Jawad, "Detection of web api content scraping: An empirical study of machine learning algorithms," 2017. [Page 19.]

[41] B. W. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme," *Biochimica et Biophysica Acta (BBA)-Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975. [Page 19.]

[42] J. Gao, F. Liang, W. Fan, Y. Sun, and J. Han, "A graph-based consensus maximization approach for combining multiple supervised and unsupervised models," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 1, pp. 15–28, 2011. [Page 22.]

[43] X. Ao, P. Luo, X. Ma, F. Zhuang, Q. He, Z. Shi, and Z. Shen, "Combining supervised and unsupervised models via unconstrained probabilistic embedding," *Information Sciences*, vol. 257, pp. 101–114, 2014. [Page 22.]

[44] X.-Y. Zhang, P. Yang, Y.-M. Zhang, K. Huang, and C.-L. Liu, "Combination of classification and clustering results with label propagation," *IEEE Signal Processing Letters*, vol. 21, no. 5, pp. 610–614, 2014. [Page 22.]

[45] "Automated detection of automated traffic," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/herley [Page 40.]

TRITA-EECS-EX- 2022:00

For DIVA

{
"Author1": { "Last name": "Sánchez Espunyes",
"First name": "Anna",
"Local User Id": "annase2",
"E-mail": "annase2@kth.se",
"organisation": {"L1": "School of Electrical Engineering and Computer Science",
}
},
"Degree1": {"Educational program": "Master's Programme, Machine Learning, 120 credits"
,"programcode": "TMAIM"
,"Degree": "Master's Programme, Machine Learning"
,"subjectArea": "Machine Learning"
},
"Title": {
"Main title": "Machine learning for detecting fraud in an API",
"Language": "eng" },
"Alternative title": {
"Main title": "",
"Language": "swe"
},
"Supervisor1": { "Last name": "Liu",
"First name": "Ying",
"Local User Id": "yinliu",
"E-mail": "yinliu@kth.se",
"organisation": {"L1": "School of Electrical Engineering and Computer Science",
"L2": "Division of Software and Computer Systems" }
},
"Examiner1": { "Last name": "Payberah",
"First name": "Amir H.",
"Local User Id": "payberah",
"E-mail": "payberah@kth.se",
"organisation": {"L1": "School of Electrical Engineering and Computer Science",
"L2": "Division of Software and Computer Systems" }
},
"National Subject Categories": "10201, 10206",
"Other information": {"Year": "2022", "Number of pages": "xi,46"},
"Series": { "Title of series": "TRITA-EECS-EX" , "No. in series": "2022:00" },
"Opponents": { "Name": "A. B. Normal & A. X. E. Normalè"},
"Presentation": { "Date": "2022-03-15 13:00"
,"Language":"eng"
,"Room": "via Zoom https://kth-se.zoom.us/j/ddddddddddd"
,Åddress": Isafjordsgatan 22 (Kistagången 16)"
,"City": "Stockholm" },
"Number of lang instances": "2",
"Abstract[eng]": €€€€

%Fraud in the context of an Application Programming Interface is a difficult problem to prevent and detect: it is unfeasible to manually review all requesting applications. API providers aim to implement an automatic tool that accurately detects suspicious applications from all existing clients. In this thesis, we use a machine learning approach to solve fraud detection in Spotify's Web API. The available data is a combination of textual and numerical data of fraudulent and non-fraudulent applications. We experiment with different supervised learning methods (random forests and gradient boosting), clustering methods (DBSCAN), and ensemble methods that combine the predictions of the supervised learning methods and the clustering methods. We also experiment with different features, focusing on proving the predictive importance of the Sentence-BERT embeddings derived from the textual data of the dataset. The method that achieves best performance in the test set is an ensemble method that combines the results from the gradient boosting classifier and DBSCAN. Furthermore, this method performs better when using the Sentence-BERT embeddings of the textual data of the applications, achieving an f1-score of 0.9896.

An Application Programming Interface (API) provides developers with a high-level framework that abstracts the underlying implementation of services. Using an API reduces the time developers spent on implementation, and it encourages collaboration and innovation from third-party developers. Making an API public has a risk: developers might use it inappropriately. Most APIs have a policy that states which behaviors are considered fraudulent. Detecting applications that fraudulently use an API is a challenging problem: it is unfeasible to review all applications that make requests. API providers aim to implement an automatic tool that accurately detects suspicious applications from all the requesting applications.

In this thesis, we study the possibility of using machine learning techniques to detect fraud in Web APIs. We experiment with supervised learning methods (random forests and gradient boosting), clustering methods such as Density-Based Spatial Clustering of Applications with Noise (DBSCAN), and ensemble methods that combine the predictions of supervised learning methods and clustering methods. The dataset available contains data gathered when a developer creates an application and data collected when the application starts making HTTP requests. We derive a meaningful representation from the most important textual fields of the dataset using Sentence-BERT (S-BERT). Furthermore, we

experiment with the predictive importance of the S-BERT embeddings. The method that achieves the best performance in the test set is an ensemble method that combines the results from the gradient boosting classifier and DBSCAN. Furthermore, this method performs better when using the S-BERT embeddings of the textual data of the applications, achieving an f1-score of 0.9896.

€€€€,
"Keywords[eng]": €€€€
Fraud detection, Machine learning, Supervised learning, Sentence-BERT, Web API €€€€,
"Abstract[swe]": €€€€

Ett API (Application Program Interface) ger utvecklare ett högnivåramverk som abstraherar den underliggande implementationen av tjänster. Användning av ett API reducerar tiden utvecklare lägger på implementation, och uppmuntrar samarbete med och innovation av tredjeparts-utvecklare. Att göra ett API publikt har ett risk: utvecklare kan utnyttja den på olämpliga sätt. De flesta API:erna har ett policy som beskriver beteenden som räknas som bedrägliga. Upptäckandet av applikationer som använder ett API på ett bedrägligt sätt är ett icke-trivialt problem, det är omöjligt att undersöka alla applikationer som skickar begäran till API:et. API leverantörerna siktar ständigt på att skapa ett automatiskt verktyg för att exakt upptäcka applikationer misstänkta för bedrägeri.

I denna avhandling undersöks möjligheten av användning av maskininlärning för att upptäcka bedrägeri i Web API. Vi experimenterar med övervakad inlärningsmetoder (random forests och gradient boosting), klustring metoder som Density-Based Spatial Clustering of Applications with Noise (DBSCAN) och ensemble metoder som kombinerar prediktionerna av övervakad inlärningsmetoder och klustring metoder. Det tillgängliga datasetet innehåller data samlat när en utvecklare skapar en applikation och när den börjar skicka HTTP begäran. Vi härleder en meningsfull representation från de viktigaste textfälten i datasetet med hjälp av Sentence-BERT (S-BERT). Dessutom experimenterar vi med den prediktiva betydelsen av S-BERT-inbäddningarna. Metoden som uppfyller den bästa prestandan i testsetet är ett ensemble metod som kombinerade resultaten från gradient boosting klassificeraren och DBSCAN. Denna metod presterar även bättre vid användning av S-BERT-inbäddnignarna av applikationernas textdata och därav uppnår ett f1-score på 0.9896.

€€€€,
"Keywords[swe]": €€€€
Bedrägeriupptäckt, Maskininlärning, Övervakad inlärning, Sentence-BERT, Web API €€€€,
}