Final Report

Contents

- 1. Executive Summary
- 2. Introduction
- 3. Data Science Methods
- 4. Data Product and Results
- 5. Conclusion and Recommendation
- Citation
- Appendix A: Feature Engineering

An Analytical Framework for Quantifying API Risks

Capstone Partners:

TeejLab and UBC Saunder

Mentor(s):

Gittu George and Gene Lee

Contributors

- Anupriya Srivastava
- Harry Chan
- Jacqueline Chong
- Son Chau

1. Executive Summary

Application Programming Interface (API) is a set of definitions and protocols for building and integrating application software [1]. Due to the heightened focus on data privacy and security, there is an increased risk of using APIs. TeejLab is a research-driven cybersecurity company, aimed at helping organisations with the evolving challenges of the API economy.

Presently, risk is quantified via manual inspection or a rules-based approach. This method is only suitable if we are dealing with few API endpoints and with few features to consider. However, it will prove to be insurmountable and laborious as we scale up the endpoints by several fold, and as we introduce more features - requiring more time and labour to manually label each endpoint. Therefore, we aim to create a machine learning model to quantify risk at each API endpoint based on security and data sovereignty markers.

More tangibly, we created a data pipeline for both preprocessing and machine learning, and documentation on how to use these scripts. This will allow Teejlab to iterate on the present code, add additional security aspects into the machine learning pipeline, and aid in their long-term goal of incorporating an “endpoint security” column on their platform to provide their clientele an additional metric on the API.

2. Introduction

2.1 Capstone Partner's Needs

APIs have been present for decades and are set to grow exponentially, in part due to regulations (in public health or finance) or by industry interoperability (in telecommunications) or disruption (in media or retail) [2]. As a cybersecurity company focused on API management at a global scale, TeejLab aims to tackle this key industry challenge of ***quantifying business risk at the API level***. A manual inspection or a rules-based approach is insufficient to accurately capture various aspects of risk, such as *security, legal, similarity, and data sovereignty*. Security refers to how vulnerable the endpoint is to attack, such as the exposure of personal identifiable information (PII) or financial identifiable information (FII), and how stringent the hosting countries' bylaws are for privacy policy. *Legal aspect* is related to the level of protection for users for data use and distribution. *Similarity* refers to how much user data APIs in the same category are requesting, where one that is requesting for too much data is deemed as less secure. Finally, *data sovereignty* refers to governance of data, which is at risk if they fail certain security tests, such as injection attacks, or if the endpoints are accessible to users.

It is time and labour intensive to inspect every feature and assign a risk label to an API endpoint, let alone quantities of API endpoint in the range of thousands or millions. This process is also static, and would require constant revision to the rules as domain knowledge increases.

2.2 Scope of Project

Based on the dataset provided and time limitation, our team has narrowed the scope to focus on creating a machine learning pipeline to ***quantify the risk of the endpoints of each API based on security and data sovereignty markers***. More specifically, our project aims to create a well-annotated python script for each stage - (1) data pre-processing and feature engineering, and (2) machine learning.

3. Data Science Methods

3.1 Available Data

The dataset provided by TeejLab contains around 2,000 observations of Hypertext Transfer Protocol (HTTP) Requests via third-party APIs. Each row of data represents the full HTTP request made by *TeejLab Services* to the third-party API endpoint, which are also annotated with the level of severity (i.e. High, Medium, Low). The overview of the datasets is shown in Table 1 and the detailed description of the dataset is introduced in Table 2. There are 18 features, four of which are to identify the API, nine are categorical variables, four are text variables, one is binary and the target is an ordinal variable.

Label	Number of the samples
High	4
Medium	876
Low	961
Total	1,841

Table 1 : The statistical summary of the dataset

No.	Column	Type	Description
1	api_endpoint_id	Categorical	Unique id of API Endpoint
2	api_id	Categorical	Unique id of API Service
3	api_vendor_id	Categorical	Unique id of API Vendor
4	request_id	Categorical	Unique id of API request
5	method	Categorical	Category of HTTP method request
6	category	Categorical	Category of API
7	parameters	Text	Variable parts of a resource that determines the type of action you want to take on the resource
8	usage_base	Categorical	Type of the pricing model of API
9	sample_response	Text	Sample HTTP Response in JSON format
10	authentication	Categorical	Authentication method used (e.g. OAuth2.0, Path, None)
11	tagset	Text	Keys of sample response
12	security_test_category	Categorical	Category of security vulnerability test
13	security_test_result	Binary	Result of security vulnerability test
14	server_location	Categorical	Location of server host
15	hosting_isp	Categorical	Internet service provider (ISP) that runs website
16	server_name	Categorical	Name and the version of web server used in API
17	response_metadata	Categorical	API Response Header
18	hosting_city	Text	Location of web hosting
19	Risk label	Ordinal	Severity level of risk (target label)

Table 2 : The detailed description of the columns in the dataset

3.2 Summary of the Exploratory Data Analysis (EDA)

We identified two key issues after performing EDA. The first was insufficient data points for high-risk labels (See Fig. 1 below). This will result in the training model reading too much into the four instances that are labeled as high risk. The training model will remember the patterns. This will be described in further detail in Section 3.3.

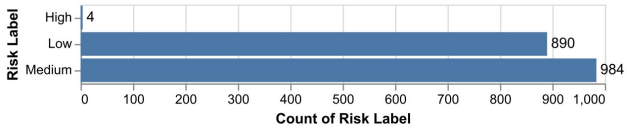

Fig. 1 Histogram of the Risk Label highlighting severe class imbalance

The second issue was that there might be some features hidden from the raw data. Feature engineering techniques are necessary to augment the value of existing data and improve the performance of our machine learning models. This will be described in further detail in Section 3.5.

3.3 Data Augmentation

A major challenge was the imbalanced nature of data, particularly with regard to the class of interest (High Risk). We had only four observations of the High Risk API class in our training set which was insufficient for the models to learn how to predict this class effectively. We attempted these approaches:

- Synthetic Data generation
- Bootstrapping
- Oversampling (SMOTE)

For *Synthetic Data generation*, while it is able to generate new numerical data, the nature of the technique did not allow us to generate columns for text data, such as *response data* and *response metadata*. For *Bootstrapping*, while it was easy to implement, and the overall data size did increase, the underlying distribution and imbalance in Risk labels was not handled.

Hence, we used the `imblearn` package for ***Oversampling***. It is a technique that generates new data by 'clustering' the data points based on the risk labels and generating new samples based on their 'neighbours' information. This technique gave a balanced dataset with sufficient representation of High risk class for further modeling (Fig. 2).

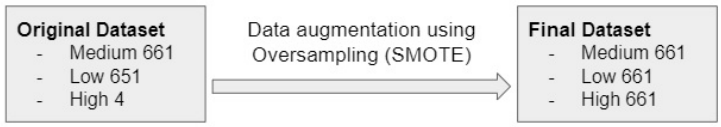

Fig. 2 Transformation of imbalanced training data set to a balanced one

3.4 Evaluation Metric and Acceptance Criteria

As per our discussion with TeejLab, we understood that it is critical to correctly identify the High Risk classes, while also ensuring that not too many Low and Medium risk labels are incorrectly classified as High Risk.

Thus, we selected ***Recall*** [1] as the primary evaluation metric to optimize the models and to maximize the correct identification of High Risk labels. We also considered f1-score as the secondary evaluation metric to ensure that not too many data points were incorrectly classified.

It was agreed that ***Acceptance Criteria would be: Recall |(|geq|) 0.9.***

3.5 Feature Engineering

As mentioned above, the dataset consists of 18 variables, which include identity numbers, strings and text. Based on our domain understanding, it is essential to extract and/or engineer several features - (1) PII and FII extraction, (2) quantify exposure frequency, (3) quantify risk associated with server location, and (4) imputation of security test.

PII and FII are crucial pieces of personal and financial information that can be used to identify, contact or even locate an individual or enterprises [3]. It is imperative that they are transmitted and stored securely. The team attempted several techniques to extract this information, including Amazon's Comprehend, PyPi's piianalyzer package, and Microsoft's PresidioAnalyzer. However, *PresidioAnalyzer* was ultimately chosen as it had high accuracy, was transferable between both PII and FII and was cost-free. By using *PresidioAnalyzer* on the "sample_response" column and defining the pieces of information to pick up on for PII and FII (See Appendix A), we created two binary columns of whether PII and/or FII is exposed by the API endpoint (Fig. 3).

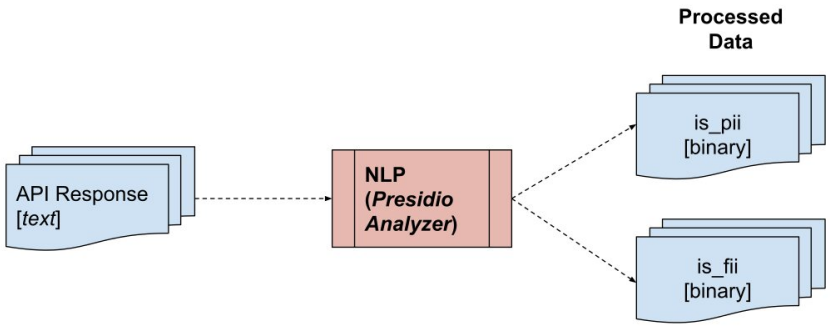

Fig. 3 Extraction of PII and FII from API Response to create two new columns

Second, we had to quantify the exposure of sensitive information. This sensitive information was present in two columns - "parameter" and "response_metadata". These columns have a format similar to a python dictionary (key-value pairs). The former refers to request data while the latter refers to HTTP response headers.

To quantify exposure, we counted the number of keys exposed per API endpoint (See Fig. 4 below). In addition, there were specific security response keys that, if exposed, could lead to a security breach. As such, we extracted 12 security response keys, and created binary columns of whether each of these 12 keys were present or absent (bottom right of Fig. 4). See Appendix A for the list and description of the 12 keys.

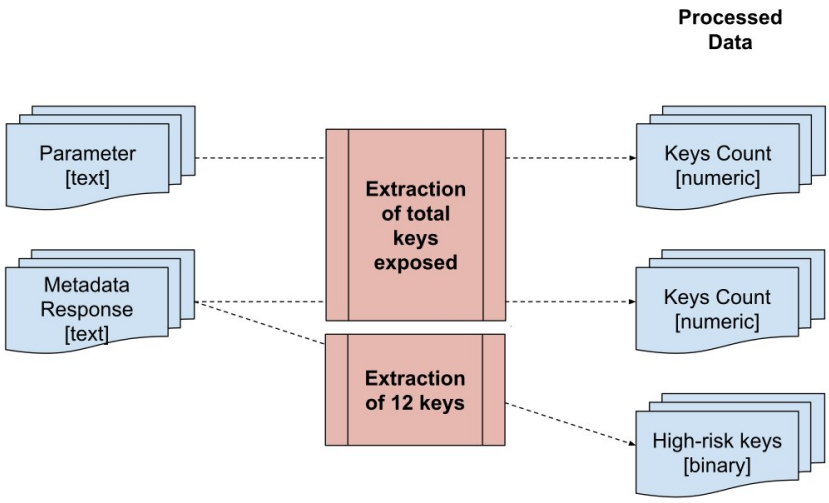

Fig. 4 Extraction of exposure frequency in parameter and metadata response column

Third, we had to quantify the risk associated with the server location (country). Apart from ensuring high loading speed of the API endpoint if it is situated near the user, the infrastructure, technology and the governance associated with the country could also impact the API's security. As a proxy, our team has used the Network Readiness Index [4]. It is an annual index that quantifies the impact of digital technology.

Finally, API exposes application logic and sensitive data, and has unique vulnerabilities and security risks. Ideally, each endpoint would contain all five security tests (as seen in Fig. 5) which TeejLab deems as crucial. However, it is unreasonable for third parties such as TeejLab to request for a temporary shutdown of these API. Based on our discussion with TeejLab, we determined that should a test not be present, it would be deemed as low risk as there was no concern warranting a request for a security test. To capture the subtle difference between a test that passed and a test that was missing, we assigned a value of 0.5 to denote that the specific test was missing, 0 to denote that specific test was employed and passed, and 1 to denote that specific test was employed and failed (Fig. 5).

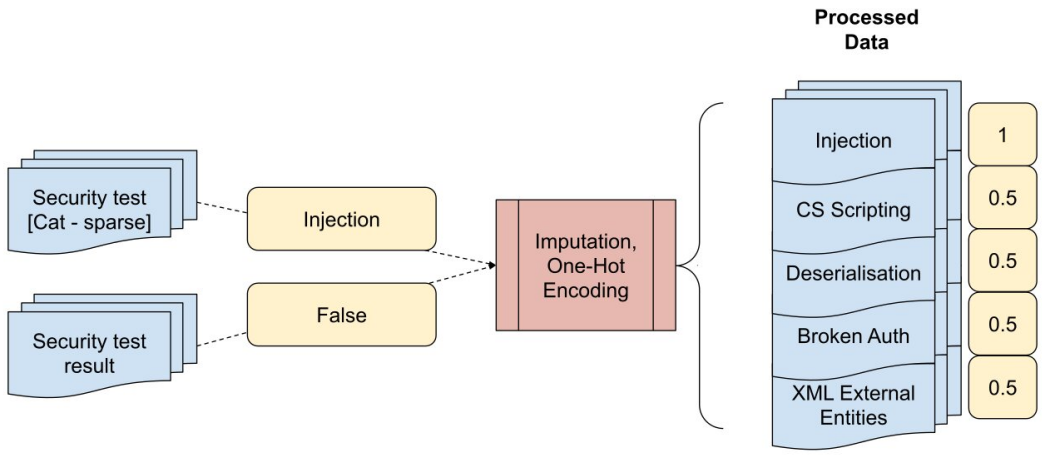

Fig. 5 An example of the processed data when the input is "Injection test failed"

3.6 Supervised Learning

As the risk labels (target column) were provided by TeejLab, we approached this problem as a Supervised Machine Learning task. We attempted the following algorithms:

- Logistic Regression
- Tree-based Models: Decision Tree, Random Forest, XGBoost, CatBoost
- K-Nearest Neighbors (KNN) and Support Vector Machines (SVM)

For each of these models, they were initially trained using the processed data set without any feature engineering. However, the maximum recall value was 0.75. Hyperparameter optimization did not lead to any significant improvement in performance of models. At this stage, Logistic Regression was selected for integration into the prediction pipeline. This is due to its high interpretability and its performance was on par with more complex models. (Refer to the unbolded section of Fig. 6 for comparison). Please refer to the technical report for more details.

To improve the scores, we reviewed the information being captured by the input features, transformed all existing columns and created new features as described in Section 3.5. This resulted in an improvement in the model performance by more than 20% (a high recall score of 0.99), which exceeded the partner's expectations.

Due to feature engineering, the feature set increased from 17 to 53. Thus, we employed feature selection to reduce the dimensionality yet ensuring that our recall score remains high. We attempted the following algorithms:

- Recursive Feature Elimination (RFE)
- RFE with Cross Validation (RFECV)
- SelectFromModel()

RFE is a backward selection method which recursively considers a smaller feature set, RFECV works similarly but eliminates features based on validation scores, while SelectFromModel selects the relevant features based on feature importance. We ultimately chose RFE as it was the most consistent and resulted in the greatest dimensionality reduction from 53 to eight. The improvement of recall score is provided in Fig. 6.

Performance improvement

Evaluation Case	Recall train	Recall test
Processed data + Dummy model	0.317	0.352
Processed data + Decision Tree	0.770	0.742
Processed data + Logistic Regression	0.764	0.748
Hyper-parameter Optimization	0.764	0.747
Feature Engineering + Logistic Regression	**0.999**	**0.995**
Feature Selection + Logistic Regression	**0.998**	**0.998**

*Selected **Logistic Regression** because of high interpretability and strong performance on all metrics*

Fig. 6 Improvement of training and testing recall scores via Featuring Engineering and Feature Selection.

[1] It is the number of true positives divided by the number of true positives plus the number of false negatives". In our case, it is the ratio of the number of cases correctly identified as High Risk compared to the sum of the cases correctly identified as High Risk and the number of cases incorrectly identified as High Risk when it is actually Low or Medium Risk.

4. Data Product and Results

The above is the method and technique employed to respond to the objective of the project: ***to quantify the risk of the endpoints of each API based on security and data sovereignty markers***. To ensure that TeejLab can use our result and deploy an additional column titled "Endpoint Security" on their platform, the following data product will be handed over.

4.1 Data Pipeline

The data product will include scripts for the ***preprocessing*** and ***machine learning*** pipeline. The process is documented with the expectation that Teejlab can reproduce the results with minimal guidance. The data product is accompanied with a Jupyter notebook for how to interpret the results. To elaborate further, the processed data sets will be the output of the preprocessing pipeline, while the model and prediction will be the output of the machine learning pipeline.

This pipeline is modular, making it easy for TeejLab to add new scripts and commands as the domain knowledge evolves. Our pipeline also offers TeejLab the ability to schedule different aspects of the pipeline to be run distinctly or as a whole.

4.2 Auxiliary Script

In addition, we converted TeejLab's rules into a script to automatically label the risk class of each data point at the request of Teejlab. The script will look for specific words or phrases within the API content and label it based on the rule. If or when the domain knowledge changes, TeejLab will be able to adjust the rules accordingly.

As the target labels are defined by the rules in this auxiliary script, should there be new rules, the API risk label scripts will need to be rerun.

4.3 Improvements

To be able to integrate with Teejlab's existing software, the interface has to be user-friendly and easy to maintain. Also, it must be scalable for future growth. Due to time constraints and limited resources, this integration would need to be conducted by TeejLab's team.

There are other areas of improvement regarding our model. For instance, we can leverage machine learning tools to create a model or script that is able to be constantly updated and adjusted based on the new data. However, as this project is at the conception stage of this domain, our model will serve as a starting point for TeejLab to further enhance it as domain knowledge broadens.

5. Conclusion and Recommendation

In this project, we set out to create a tool to ***quantify the risk of the endpoints of each API based on security and data sovereignty markers***. To this end, the team has been successful in creating a model that is not only highly interpretable. More importantly, it scores highly in the acceptance criteria (Recall) - attaining a recall score of 0.99 for both the train and test set.

Despite the team's success, we decided to further investigate the validity of the risk labels. To re-emphasise, API security is a completely novel field and thus there is no absolute or ground truth to their labels. As such, we attempted an unsupervised learning technique.

5.1 Unsupervised Learning

Currently, risk labeling is subject to the capstone partner's expertise. However, data can be clustered in many ways, and there exist a large body of algorithms designed to reveal underlying patterns.

We have built a visual analytics tool using Factor analysis of mixed data (FAMD) and Tensorflow Embedding Projector [5] that helps analyze the similarity between the data points. As seen in Fig. 7, we have found that the data points were not distinct in the 3D plane, indicating that the risk labels might not be sufficiently reliable.

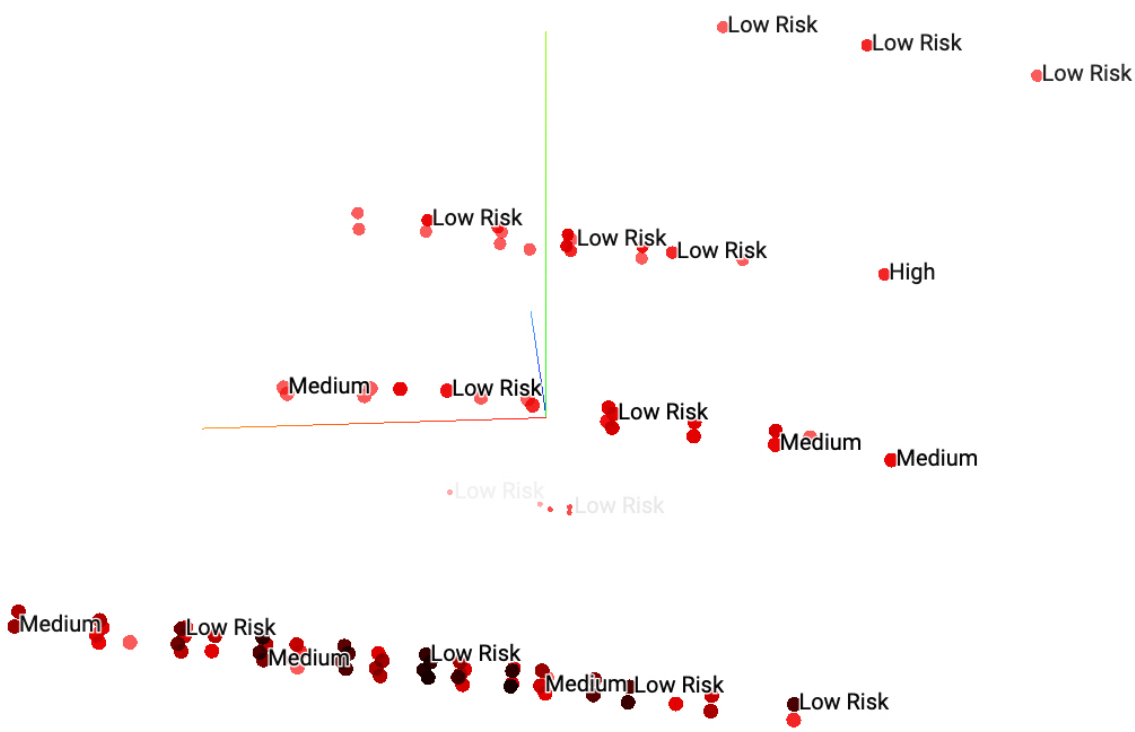

Fig. 7 PCA Visualization of Current Risk Label showing no distinct clusters between low and medium risk labels

Using K-prototypes clustering algorithm, the team attempted to group similar data points according to their metadata. Visually, this algorithm provides more distinguishable clusters that are more equally distributed with clear boundaries (See Fig. 8 below). We can also inspect all the attributes of each data point and compare the similarity within the intra-cluster and inter-cluster boundary (See Fig. 9 below).

Fig. 8 PCA Visualization of Cluster Label showing three distinct clusters along the vertical axis

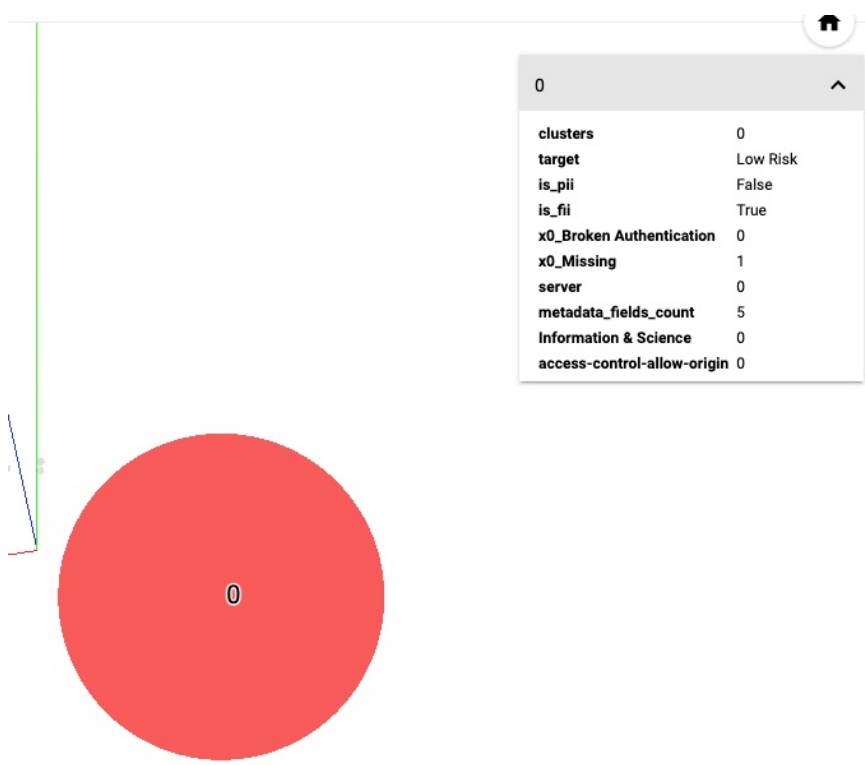

Fig. 9 PCA Visualization of Cluster Label with metadata

In conclusion, our team is highly satisfied with the data product that we will be handing over to TeejLab. While we are aware that some improvements are required before deployment, the team has been most proud of extracting hidden features, quantifying the value of categorical features, and highlighting areas of improvement where our partner has overlooked. This is ultimately a small step in the right direction in this uncharted domain.

Citation

1. What is an api? Oct 2017. URL: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces.
2. Tiffany Xingyu Wang and Matt McLarty. Apis aren't just for tech companies. Apr 2021. URL: https://hbr.org/2021/04/apis-arent-just-for-tech-companies.
3. What is personally identifiable information: pii data security: imperva. Dec 2019. URL: https://www.imperva.com/learn/data-security/personally-identifiable-information-pii/.
4. Network readiness index 2021. URL: https://networkreadinessindex.org/.
5. Nikhil Thorat Daniel Smilkov. URL: https://projector.tensorflow.org/.

Appendix A: Feature Engineering

Tags defined for PII and FII

Personally Identifiable Information (PII) tags	Financially Identifiable Information (FII) tags
PERSON	CREDIT_CARD
LOCATION	CRYPTO
NRP	IBAN_CODE
EMAIL_ADDRESS	US_BANK_NUMBER
MEDICAL_LICENSE	US_ITIN
IP_ADDRESS	US_SSN

Description of High Risk Security Headers These descriptions are incorporated from Cheat Sheet (https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html)

High Risk Security Headers	Description
x-frame-options	It can be used to indicate whether or not a browser should be allowed to render a page in a `<frame>`, `<iframe>`, `<embed>` or `<object>`. Sites can use this to avoid clickjacking attacks, by ensuring that their content is not embedded into other sites.
x-xss-protection	It is a feature of Internet Explorer, Chrome, and Safari that stops pages from loading when they detect reflected cross-site scripting (XSS) attacks.
strict-transport-security	It lets a website tell browsers that it should only be accessed using HTTPS, instead of using HTTP.
expect-ct	It lets sites opt-in to reporting of Certificate Transparency (CT) requirements. Given that mainstream clients now require CT qualification, the only remaining value is reporting such occurrences to the nominated report-uri value in the header. The header is now less about enforcement and more about detection/reporting.
referrer-policy	It controls how much referrer information (sent via the Referer header) should be included with requests.
content-type	It is used to indicate the original media type of the resource (before any content encoding is applied for sending).
set-cookie	It is used to send a cookie from the server to the user agent, so the user agent can send it back to the server later. To send multiple cookies, multiple Set-Cookie headers should be sent in the same response.
access-control-allow-origin	It indicates whether the response can be shared with requesting code from the given origin.
server	It describes the software used by the origin server that handled the request — that is, the server that generated the response.
x-powered-by	It describes the technologies used by the webserver. This information exposes the server to attackers. Using the information in this header, attackers can find vulnerabilities easier.
x-aspnet-version	It provides information about the .NET version.
x-ratelimit-limit	The maximum number of requests available in the current time frame.

By Team API Risks

© Copyright 2022.