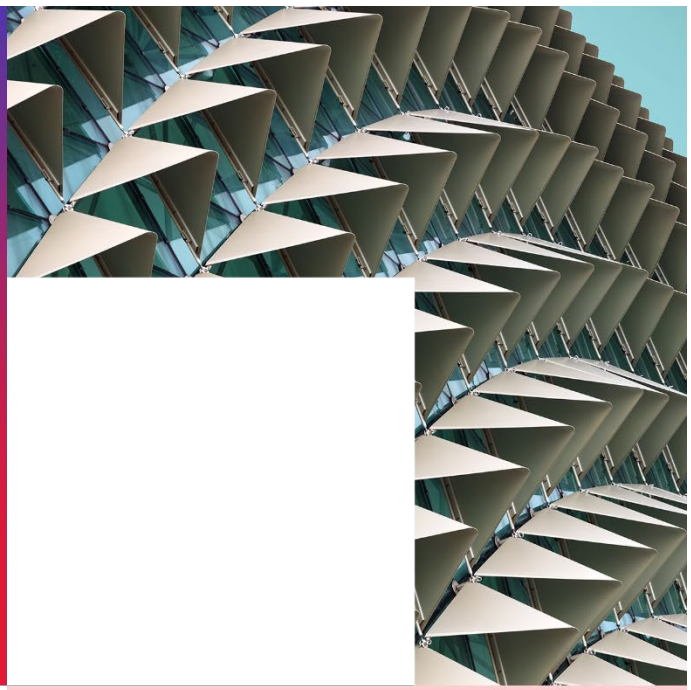


Using machine learning to improve software identification

Modern solutions to legacy challenges

Ileana Gutierrez, David Crawford

2024-10-09

Table of contents

1 Introduction

1.1 Software identification challenges

Software developers have several robust software identification methods to choose from, including Common Platform Enumerations (CPE), Package URLs (PURL), Software Identification Tags (SWID) and Common Vulnerability and Exposures (CVE).

However, substantial gaps in software product identification still exist. The gaps will persist until the software development community, encompassing both open-source and commercial developers (including cyber tool vendors) work together to establish standards for software identification. It is crucial to ensure consistency in the location of installed software identification information on devices. It is also unlikely that any one of the proposed standards will be adopted as a single standard for software identification.

From our perspective, the specific method used is not of utmost importance. What truly matters is that the producer of the software implements one or more of the core methods (CPE, CVE, PURL or SWID). Secondarily, software asset inventory tools, and vulnerability tools need to support these standards. Additionally, it is crucial to adopt a standardized set of post-installation locations, which may vary depending on the platform, where cybersecurity tools and other applications can reliably locate the identification and inventory artifacts.

It is evident that a fallback method will be necessary to create a reasonably accurate representation of the software identification in a generic format, using the discovered vendor, product, version, etc. The Department of Homeland Security's Continuous Diagnostic and Mitigation (CDM) program has recognized the need for an alternative and has implemented a more generic approach. However, it is crucial to address the inconsistencies discussed below.

Right now, while CPE is the presumed standard, few of the end point management or other tools used to provide software asset management accurately return CPE values that can be matched to the master CPE dictionary. Certain tools provide results in different formats of CPE, such as CPE 2.2 or CPE 2.3. However, even with wildcarding, the actual values of these CPEs do not match the ones in the master list.

On the other hand, some tools return the expected attributes--vendor, product, version, etc.--but not in the form of a CPE-like object. Moreover, there is a notable lack of consistency in how the values for the attributes are represented, even within data from the same provider. The ability to directly match entries in the inventory of installed software items on an endpoint with the master CPE list greatly simplifies upstream activities, such as matching CVEs returned from a vulnerability scan to a specific software product. However, when there is no match or the match rate is low, it creates significant limitations in upstream activities.

There is no easy fix to this, but coming to a consensus about both how software is to be identified on a device, and where that identification can be located by other tools would be a good starting point.

1.2 Capstone structure overview

Over the 2023-2024 academic school year, CGI sponsored a cohort of George Mason University Cybersecurity Engineering seniors. The cohort was made up of Kiet Hoang Nguyen, Faris Issa, Arman Makar, Shaho Ahmed and Beni Mahato. The cohort worked on developing machine learning models to correlate software inventory data gathered from several cybersecurity tools with a pre-defined dictionary of known software products. The results of the cohort's research were documented in their paper titled "Develop Machine Learning Models using AWS SageMaker"[i]. CGI Federal provided the GMU students an Amazon SageMaker environment to develop these machine learning techniques.

During the first semester of the capstone project the students focused on learning how to use AWS SageMaker. SageMaker was also a new tool for some on the CGI team so there was a learning progression for both parties. The students followed a tutorial that AWS provided to get started with SageMaker. Team CGI provided the students with the domain, appropriate permissions and SageMaker workspace. Once the students started to get comfortable with the workspace, the datasets were introduced to them. With the datasets, the students started training their artificial intelligence model, tested different algorithms and researched ways to make SageMaker more efficient for the task at hand. As the end of the semester approached, they were able to create their first prototype.

In the following semester the students made substantial progress with their model development. The initial plan was to use supervised machine learning to develop these models, but it was found to be more efficient to use unsupervised learning techniques due to the nature of the datasets. The students tried several approaches: a machine learning (ML) method, a non-ML method, and a hybrid approach mixing both. In the end, the students determined that the hybrid approach resulted in a higher match rate than either the ML approach or the non-ML approach. The hybrid approach used the ML method to pick from the top 1,000 choices for a given search parameter and the non-ML method (fuzzy matching) to find the best match. The shortcomings of the ML only approach were not investigated in depth. A basic analysis suggested that the ML approach could be improved by adjusting parameters of the model, combined with additional tuning. Time limitations precluded experimentation to determine if the initial analysis was correct.

2 GMU research

2.1 Problem statement

The current methods of mapping software inventory data with a standardized list of software products are inefficient and are susceptible to errors. The current limitations are due to a range of issues, including the varied naming conventions used by different vendors and scalability issues due to the large data volumes. The research conducted in the GMU capstone project provides a potential solution to the problem above, leveraging unsupervised machine learning techniques to automate the correlation process.

2.2 Concept of operations and requirements

The students were tasked with using AWS SageMaker to train a machine learning model to match discovered software inventory data with a large (1.25M), curated dictionary of software data. The goal being to match as closely as possible based the following values: vendor, product, version, update, and edition. They used several steps to reach their desired outcome. They started with researching different machine learning algorithms for software inventory data correlation to level set. Once comfortable with SageMaker, the students used the datasets

for training the models and deployment. The students were then able to develop a machine learning data-matching model to correlate software inventory data with a standardized dictionary of known products.

2.3 Methodology

The students took time to thoroughly study the datasets provided by CGI, which included software inventory data gathered from several cybersecurity tools and the master dictionary of known software products. These datasets are extensive in size, so it was another hurdle for the students to keep in mind when they were researching their algorithm selection and model development process. It was around this time, they found that the best approach would be to use unsupervised learning instead of the supervised learning due to the nature of the data.

Once the students had clear understanding of the objectives of the project and the SageMaker environment, they began writing Python code using Jupyter Notebook within the SageMaker studio. At this time the students had a full understanding of the datasets, so they were able to decide on the algorithms they found were best for the project. They used fastText's Skip-Gram model due to its efficiency in text representation combined with the Approximate Nearest Neighbor (Annoy) library for efficient indexing and matching.

The different techniques they used in the project were also a factor in their success: they created a cleaner software ID by merging five attributes into a unified identifier. They also used fuzzy matching “best match” techniques to refine and enhance matching accuracy at the post-processing stage.

2.4 SageMaker walkthrough

This section focuses on the technical aspect of how the students used SageMaker to yield their successful results. The first step the students had to take before training the model was preparing the data. This was done by removing the headers and converting the data the model uses from comma-separated values (CSV) format to text. After preparing the data, the students were able to proceed to training the model. The code snippet at right shows how the students used unsupervised learning to train their chosen model, the fastText's Skip-Gram model. We also see other training parameters, such as the epoch, minCount, and learning rate (lr). The epoch is the number of complete cycles through the entire training dataset. The minCount indicates the minimum times a word needs to be present in the dataset to be included in training. The learning rate (lr) is a hyperparameter that controls the degree of change to the model in response to the estimated error whenever the model weights are changed.

```
import fasttext

model = fasttext.train_unsupervised(
    'fasttext_training_text.txt',
    model = 'skipgram',
    epoch = 10,
    minCount = 1,
    lr = 1,
)
# Use cbow for continuous bag of words model
# Use skipgram for skipgram model

model.save_model('fasttext_model.bin')
```

After training the data, the next step was to build the Annoy index. First, string embeddings, also known as vectors, had to be generated from the fastText model and training data. From there the students built the Annoy index with a choice of trees and metrics. This Annoy library is used to construct an efficient indexing system due to its capability to use angular distance for nearest neighbor calculations. This gives the students the ability to keep a high level of precision in the similarity searches.

```python
import numpy as np
def get_embedding(model, string):
    return model.get_sentence_vector(string)

string_embeddings = np.array([get_embedding(model, string.strip()) for string in strings_list])
```

```python
from annoy import AnnoyIndex

def build_index(embeddings, n_trees=10):
    f = embeddings.shape[1]
    index = AnnoyIndex(f, 'angular')

    for i, vector in enumerate(embeddings):
        index.add_item(i, vector)

    index.build(n_trees)
    return index
```

```python
annoy_index = build_index(string_embeddings, 50)
```

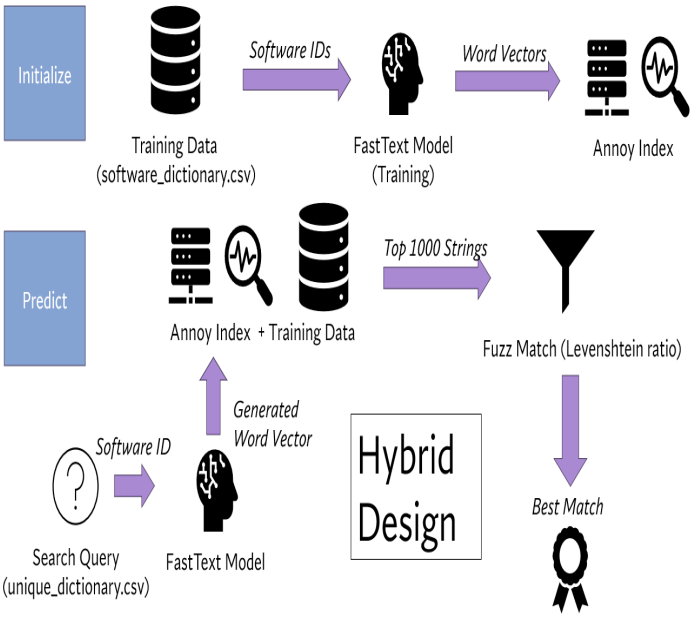
The last step was to find the best match. The students used the Annoy Index to return the top_k matches, in which they set to 1000. From there they were able to use the fuzzy matching algorithm on the top_k results and the candidate with the best score got returned as the best match.

```python
def find_best_match(annoy_index, model, query_string, strings_list, top_k=5):
    best_score = 0
    model_score = 0
    best_match = None

    top_k_matches = find_most_similar_annoy(annoy_index, model, query_string, strings_list, top_k)
```

```python
#print(f"Top {top_k} matches:\n")
for candidate, similarity in top_k_matches:
    #print(candidate)
    score = detailed_match(query_string, candidate)
    #print(score)
    if score > best_score:
        best_score = score
        best_match = candidate
        model_score = similarity
    elif score == best_score:
        if similarity > model_score:
            best_score = score
            best_match = candidate
            model_score = similarity

#print("\n")
if best_match:
    return best_match, best_score, model_score
else:
    return None, 0
```

2.5 Designing, training and evaluation, Results

To create the prototype, the students had to prepare the data, select the appropriate algorithm (fastText Model) for the training, and utilize the Annoy Index for efficient nearest neighbor searches. After coming up with the top matches from the Annoy Index, in our case the top 1000 matches, they applied the fuzzy matching algorithm to the top results. From these results, the candidate with the best score was returned as the best match. The reason the students refer to it as their hybrid method is because the ML method picks from the top 1000 choices and the non-ML method finds the best match. The “hybrid method” approach the students came up with provides a suitable combination of accuracy and efficiency for the software inventory matching. It far exceeds the match-accuracy compared to previous methods of software matching.



2.6 Conclusion

The capstone project conducted by George Mason University's Cybersecurity Engineering students with guidance from our CGI Federal team showed great promise, representing a significant step towards using machine learning to address challenges in the efficiency managing software inventory data. By developing this unsupervised learning model, the results showed a highly effective match rate between cybersecurity tool data with a standardized software product list. This project is just a steppingstone; further improvements can be explored to enhance software identification in our evolving cybersecurity landscape. CGI intends to continue exploring methods utilizing machine learning and text matching schemes in this use case and others, in pursuit of greater matching between observed values reported by cyber security tools, and the related curated/master databases.

[i] “Develop Machine Learning Models in AWS SageMaker”, Nguyen, K., Issa, F., Markar, A., Ahmed S., Mahato, B., George Mason University, 2024, unpublished.