IBM Security

Resilient SOAR Platform

Resilient Machine Learning Function

Reference Guide

V1.0

Date: February 2020

Resilient SOAR Platform
Resilient Machine Learning Function
Reference Guide

Platform Version	Publication	Notes
1.0	February 2020	Initial publication.

Table of Contents

Overview 4

- Architecture 5
- Dependencies 5

NLP of Machine Learning 5

- Word embeddings 6
- Sentence embeddings 7
  - SIF 7
  - Remove principle component 7

Resilient NLP 7

- Build an NLP model 7
- Use an NLP model to find similarity 8
- Rebuild the NLP model 9

Technical Details 9

- Preprocess raw data 9
  - Parsing 9
  - Tokenization 9
  - Lemmatization 9
  - Compositional training 9
  - Skip-gram 10
- Keywords 10

Advanced Features 10

- Number of features 10
- Other gensim settings 10

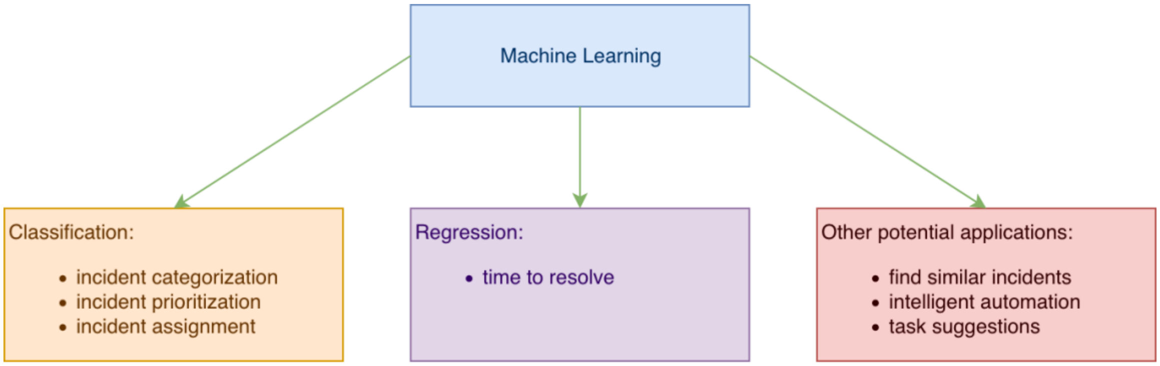
Reference 11

Overview

This is a reference guide for the Resilient Machine Learning Function (fn_resilient_ml). It is a companion guide to the *Resilient Machine Learning Function User Guide*.

Machine learning can extract knowledge from historical incidents stored in a Resilient platform. This knowledge normally includes useful pattern information of the customer environment. More importantly, it can include the experience of security analysts who have worked on those historical incidents. In some sense, a machine learning model learns from the previous decisions the security analysts made. Thus, machine learning can be very useful in assisting security analysts to make quick and proper response.

The following graph shows possible applications of machine learning in an incident response system like the Resilient platform. Please note that this release supports “find similar incidents” only. Other potential applications are not supported for this release.



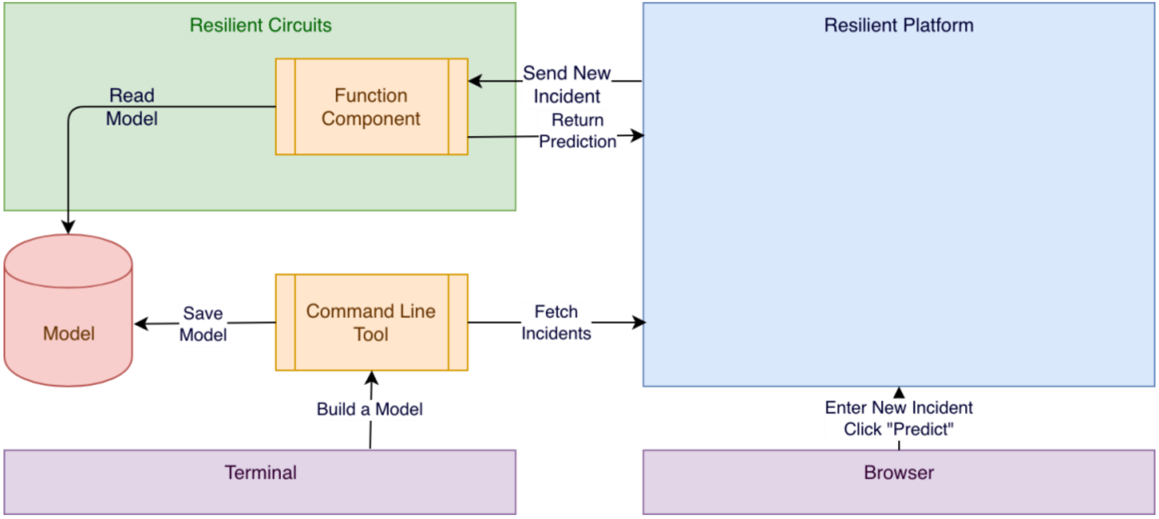
The Resilient Machine Learning Function is highly customized for the incident response system. It is fully integrated to the Resilient platform to provide the best user experience, and can be incorporated into a custom workflows in flexible ways. In addition, user data is processed locally, thus customers do not need to be concerned about transmitting sensitive data to the cloud when a machine learning model is built.

This guide includes background information about features supported by the Resilient Machine Learning Function. In addition, it includes recommendations to resolve common issues Resilient platform users might face.

Architecture

The Resilient Machine Learning Function contains two components.

- A command line tool, called res-ml, to build a machine learning model.
- A function component to compute similarity using a saved machine learning model.



Dependencies

The res-ml tool is a Python based application. It depends on the following third-party Python packages for machine learning support:

- Scikit-learn 0.21.3: An open source Python package for machine learning with BSD license. It is built on Numpy and SciPy. The res-ml tool uses the PCA algorithms supported by scikit-learn.
- Pandas 0.25.1: An open source Python package for easy-to-use data structures and related analysis tools. It also has a BSD license. The res-ml tool uses Pandas to manipulate datasets, which are a collection of incidents. This includes extracting features from incidents, filtering samples, and up-sampling training dataset.
- Numpy 1.17.1: A fundamental package for scientific computing with Python with BSD license.
- Scipy 1.3.1: An open source Python package for mathematics, science, and engineering with BSD license.
- Nltk 3.4.5: Natural Language Toolkit. Apache license.
- Beautifulsoup4 4.8.0: A library that makes it easy to scrape information from HTML format files. This integration uses it to pre-process training data, which might contain HTML/XML tags. MIT license.
- Gensim 3.8.0: GNU LGPLv2.1 license. This integration uses the word2vec model of gensim.

Note that some of the python packages require python3. As a result, Resilient Machine Learning Integration supports python3 only.

NLP of Machine Learning

Natural Language Processing (NLP) is a branch of Machine Learning and Artificial Intelligence. It is about the interaction between computers and human language. An NLP model can be trained to digest and analyze textual information. Textual information here refers to the name, description, and resolution summary of historical incidents. By understanding the textual information, a model can then compute the similarity between two incidents.

One use case of the NLP model is to first use historical incidents as training data to build an NLP model. A model trained like this has the ability to understand textual information of incidents. Later, create a new incident with description data. The textual information of the new incident can be used by the NLP model to find similar (historical) incidents.

Word embedding is one of the successful NLP models. Resilient machine learning integration uses word embeddings to analyze incidents.

Word embeddings

The linguistic theory[1] behind word embedding is that similar words can appear in similar context. For example, the following two sentences have similar meaning:

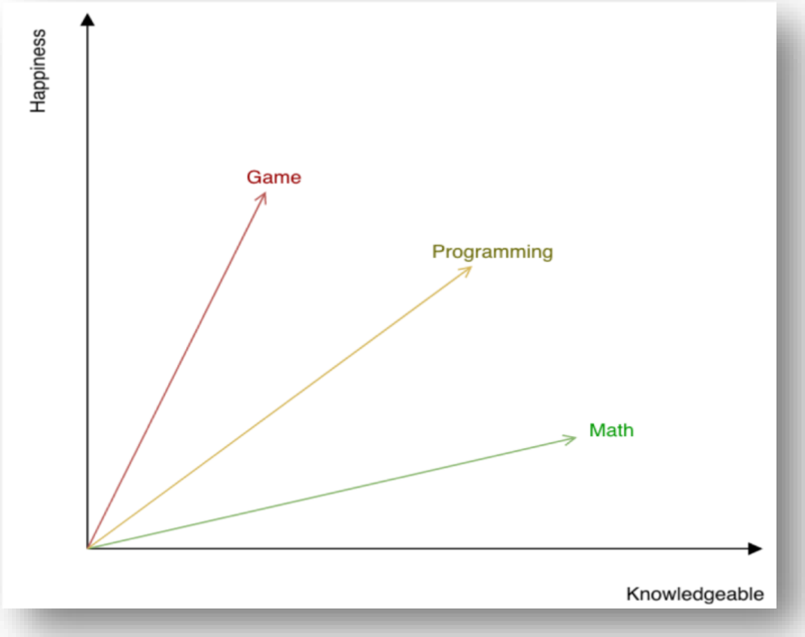
- Malware has been detected on a laptop
- Virus has been detected on a laptop

Here "malware" and "virus" appear in similar context. They are then similar words within the context of the training dataset, even though they mean completely different things in other contexts.

Word embedding is a learned representation in such a way that similar words have similar representation. A representation can be a vector space with pre-defined number of dimensions. For example, Resilient Machine Learning Integration uses 50 as the default value for dimensions. An NLP model trained like this converts each word into a 50-dimensional vector of real numbers.

Resilient Machine Learning Integration uses the gensim word2vec package to train an NLP model. It is an unsupervised algorithm involving a two-layer shallow neural network.

To understand how a word can be represented by a multi-dimensional vector of real numbers, it is helpful to imagine that each dimension of the vector space represents a meaningful feature that a human can understand. Here each word is a combination (vector sum) of those meaningful features. Similar words should then have similar combination of those meaningful features, and thus stay close to each other in the vector space. A simplified example is shown in the following image. Here, only two dimensions are used so that it can be easily drawn. Here "programming" contains a bigger portion of the "Knowledgeable" feature than "Game", and thus closer to "Math".



The similarity between words can then be defined as the dot product of the corresponding normalized vectors. Similar words stay close to each in the vector space, and thus have bigger similarity.

In reality, users specify the number of features only, an NLP model figures out what features to use. The features identified by an NLP model, in general, are combinations of something a human being can understand.

Note that since similarity is defined as the dot product of two normalized vectors, the range of a similarity is between 0 and 1, with 1 meaning two words being exactly the same. The similarity is always positive because all the vectors are in the first quadrant.

Sentence embeddings

As mentioned previously, gensim word2vec can convert words into vectors. Similarity between words can be computed as the dot product of corresponding vectors. The next step is to compute similarity between sentences.

A simple approach is to represent a sentence using the sum of the vectors of all the words in the sentence. This is also called sentence embeddings. This simple approach can give useful results, with small enhancements, according to this research result[2] from Princeton.

There are two enhancements:

- Use Smooth Inverse Frequency (SIF) to lower down the contribution of common words
- Remove the principle component from each sentence vector

SIF

When word vectors are summed up to get the sentence vector, a weight factor is assigned to each word vector

$$\frac{a}{a + wc}$$

Here $a = 10^{-4}$ is a small constant, and wc is the word count. The larger the word count, the smaller the contribution.

To understand this, imagine a word that appears in all the incidents. This word is then not very useful in distinguishing different incidents.

Remove principle component

According to the research of the Princeton paper[2], the sentence vectors obtained by summing word vectors using SIF still share a common principle component (vector component). This is caused by some common words shared by all the sentences. By removing this shared principle component, the model can give a more accurate result.

Resilient Machine Learning Integration uses the PCA algorithm of scikit-learn to obtain the principle component from the sentence vectors.

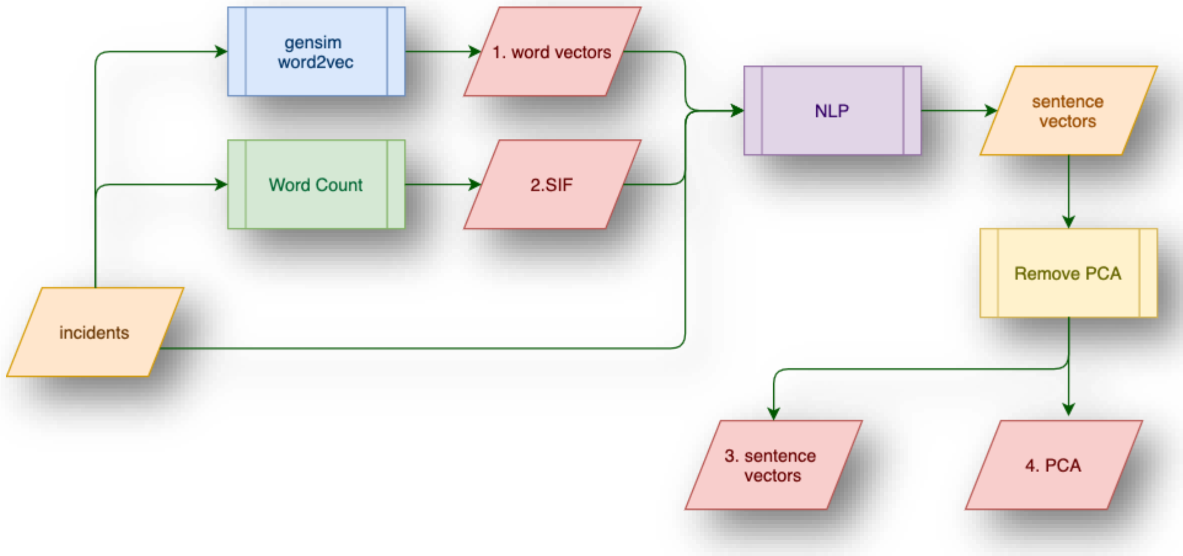
Resilient NLP

There are two separate processes:

- Build an NLP model
- Use a pre-built NLP model to find similarity

Build an NLP model

The steps to build an NLP model can be summarized into the following data flow diagram.



The steps include:

- The raw data (incidents) is used as training dataset to build a gensim word2vec model. The result is saved into a .txt file. By default, this file is named resilient-w2v.txt.
- The raw data is used to get word count. The data is saved into a file called resilient-sif.pkl.
- The raw data and the above generated data are used to get sentence vectors.
- PCA is then computed using the sentence vectors. PCA data is saved into resilient-pca.json.
- PCA is removed from each sentence vector, and all sentence vectors are saved into resilient-vec.json.

Note this is an unsupervised learning algorithm, thus the whole dataset is used to train the NLP model.

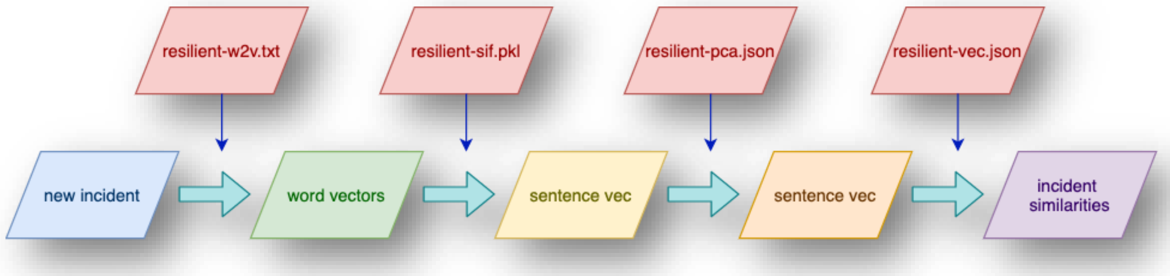
Use an NLP model to find similarity

As shown previously, an NLP model is saved into 4 files:

- resilient-w2v.txt: The word2vec model in text format. This is basically a word-vector dictionary.
- resilient-sif.pkl: The word count used for SIF. This is a word-count dictionary saved in Python pickle format.
- resilient-pca.json: The principle component vector saved in JSON format.
- resilient-vec.json: Strictly speaking, this is not part of the NLP model. This stores the computed vectors for all the (historical) incidents. It is an incident_id-vector dictionary in JSON format.

To find similarity for a new incident, the above four files are used in the following steps:

- Use the resilient-w2v.txt file to find the vector for each word of the new incident.
- Use the resilient-sif.pkl file to find the weight factor for each word.
- Sum up the word vector to get the sentence vector.
- Remove PCA vector (saved in resilient-pca.json) from the sentence vector above.
- Compute the dot product of this new sentence vector and each sentence vector is saved in resilient-vec.json.
- Find the top similarities.



One small detail here. The resilient-vec.json file saves all the vectors for those incidents created before the NLP model is built. Incidents created after the model is built are not included in the resilient-vec.json file. But they should be taken into consideration as well when searching for similar incidents. Thus, the function component looks for the highest incident ID saved in resilient-vec.json, and then downloads incidents with an ID higher than that. Those newly downloaded incidents are used in the search for similar incidents.

Also note that resilient-vec.json is created when the model is built. If an incident is created before the model is built, it is included in the file. However, if it is modified later, the modification does not take effect in the search until the model is rebuilt.

Rebuild the NLP model

Once users accumulate more incidents, they should rebuild the NLP model to make use of the newly available data.

Technical Details

This section is about how word2vec is configured within Resilient Machine Learning Integration.

Preprocess raw data

The raw data, meaning the textual data, of the incidents need to be preprocessed before it can be used to train an NLP model, as follows:

Parsing

The description of some of the incidents are in HTML format. A python package called beautifulsoup4 is used to remove HTML tags. Punctuations, accents, and umlauts are removed from sentences, since they are not relevant to the training. Other non-alphabetic characters are also removed in the steps.

Tokenization

Tokenization refers to converting sentences like “Malware has been detected on a laptop” into a list of words [“malware”, “has”, “been”, “detected”, “on”, “a”, “laptop”].

In this step, all the words are converted to lower cases. In addition, python NLTK package is used to remove normal English stop words. These stop words do not carry much relevant information which can be used to distinguish different incidents.

The current Resilient Machine Learning Integration supports English only.

Lemmatization

Lemmatization means reverting a word back to its lemma (base of the word). For example, “better” is reverted back to “good”. The WordNetLemmatizer of NLTK is used to return a word to its base form.

Compositional training

Idioms in the dataset need special attention. Idioms have meaning that are different from the combination of the individual words they contain. For example, “give up” has a meaning that cannot be derived easily from “give” and “up”.

To handle idioms, the approach is to combine words into one. The idiom “give up” is converted into “give_up” in this step.

Skip-gram

The gensim word2vec package can support two architectures:

- Continuous Bag of words
- Skip-gram

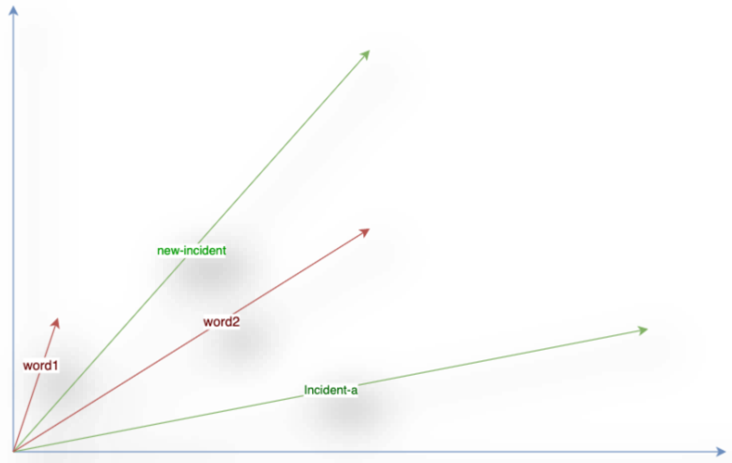
For Resilient Machine Learning Integration, skip gram is used.

The skip-gram approach means selecting a window of words, which means a number of words in continuous order. For this integration, a window is set to be 10. Now given a target word, skip-gram tries to predict the probabilities of other words appearing within this window. The objective is then to maximize the average log-probability.

The skip-gram approach is good for a small training dataset. This is good for typical Resilient users.

Keywords

When looking for similar incidents, all the words in the description of the new incident are summed into a sentence vector. To see which words in the new incident contribute the most to the similarity, one approach is compute the dot product of each word vector against the sentence vector of the other incident. The word with the highest word-sentence similarity is the keyword used in the search for similar incidents.



Assume that incident-a has been identified as a similar incident for new-incident, and new-incident is made up of word1 and word2 as shown above. Here the vector for word2 is closer to incident-a than word1. In other words, the dot product between word2 and incident-a is bigger than the one of word1. Therefore, word2 contributes more to the overall similarity than word1, and it can be thought as a keyword used in the search.

Advanced Features

This section is for advanced users to fine-tune the NLP model. There are several configuration settings.

Number of features

The default number of features for the word2vec model is set to 50. In general, users can increase this once they collect more data. 100 to 300 are also reasonable choices. This can be easily done by modifying the app.config file.

Other gensim settings

There are other settings for gensim, which are kept in a python file called nlp_settings.py under the lib/nlp folder.

```
    BIGRAM_MIN_COUNT = 3
    BIGRAM_THRESHOLD = 10

    # parameters for word2vec
    # https://radimrehurek.com/gensim/models/word2vec.html
    WORD2VEC_WINDOW = 10
    WORD2VEC_MIN_COUNT = 2
    WORD2VEC_SAMPLE = 1e-3
    WORD2VEC_ALPHA = 0.03
    WORD2VEC_MIN_ALPHA=0.0007
    WORD2VEC_NEGATIVE = 20
    WORD2VEC_PROGRESS_PER = 10000
    WORD2VEC_EPOCHS = 30
    WORD2VEC_REPORT_DELAY = 1
    WORD2VEC_FEATURE_SIZE = 50
```

For advanced users, use the following command to install the package then modify the above file to fine-tune the word2vec model. Please refer to this link for the meanings of the parameters.

```
pip install -e
```

Reference

1. Distributional Structure, by Zelling Harris: https://www.tandfonline.com/doi/pdf/10.1080/00437956.1954.11659520
2. A simple but tough-to-beat baseline for sentence embeddings, by S. Arora, Y. Liang, T. Ma: https://openreview.net/pdf?id=SyK00v5xx