random-mac

May 18, 2019

Contents:

CHAPTER 1

Introduction

A media access control (MAC) address is a 48-bit hexadecimal string. The first eight bits of a MAC address determine many of its properties.

random-mac is a fun, little experiment that uses machine learning (binary classification) to evaluate these properties and identify randomly-generated MAC addresses.

CHAPTER 2

Installing random-mac

random-mac is available on GitHub at https://github.com/critical-path/random-mac.

If you do not have pip version 18.1 or higher, then run the following command from your shell.

```
[user@host ~]$ sudo pip install --upgrade pip
```

To install random-mac with test-related dependencies, run the following command from your shell.

```
[user@host ~]$ sudo pip install --editable git+https://github.com/critical-path/
↪random-mac.git#egg=random-mac[test]
```

To install it without test-related dependencies, run the following command from your shell.

```
[user@host ~]$ sudo pip install git+https://github.com/critical-path/random-mac.git
```

(If necessary, replace `pip` with `pip3`.)

CHAPTER 3

Using random-mac

3.1 Getting started

random-mac requires lists of organzationally-unique identifiers (OUI) and company IDs (CID) assigned by the IEEE. To fetch them from the IEEE's website, run the following commands from your shell.

```
[user@host random-mac]$ curl http://standards-oui.ieee.org/oui/oui.csv > oui.csv
[user@host random-mac]$ curl http://standards-oui.ieee.org/cid/cid.csv > cid.csv
```

3.2 Making a dataset

random-mac's dataset consists of data (samples) and labels (targets). The data include one non-random MAC address for every OUI and CID assigned by the IEEE (currently about 26,000) as well as a configurable number of random MAC addresses. To make a dataset with two random MAC addresses for every non-random address (a 2:1 ratio), run the following command from your Python interpreter.

(Use the ratio that is most appropriate for your needs. Different ratios require different trade-offs. To get a sense of these trade-offs, use scikit-learn's classification reports.)

```
>>> import sklearn.model_selection
>>> import random_mac
>>> data, labels = random_mac.dataset.make(
...   2,
...   oui_file="./oui.csv",
...   cid_file="./cid.csv"
... )
```

3.3 Splitting the dataset

random-mac needs to use some data and labels for training while reserving others for testing. To split data and labels into two different groups, run the following command from your Python interpreter.

```
>>> training_data, testing_data, training_labels, testing_labels = sklearn.model_
↪selection.train_test_split(data, labels)
```

3.4 Making a classifier

random-mac uses a binary classifier. To make, train, and test this classifier, run the following commands from your Python interpreter.

```
>>> classifier = random_mac.classifier.make()
>>> classifier = random_mac.classifier.train(
...   classifier,
...   training_data,
...   training_labels
... )
>>> score = random_mac.classifier.test(
...   classifier,
...   testing_data,
...   testing_labels
... )
>>> print("score = {}%".format(str(int(100 * score))))
score = 83%
```

3.5 Using the classifier

To use the classifier, run the following command from your Python interpreter.

```
>>> address = "a0b1c2d3e4f5"
>>> results = random_mac.is_random_mac(classifier, address)
>>> print(results)
True
```

3.6 Saving and restoring a classifier

To save (pickle) a classifier for future use, run the following command from your Python interpreter.

```
>>> random_mac.classifier.save(
...   classifier,
...   file="./random-mac-classifier.pickled"
... )
```

To restore (unpickle) a classifier, run the following command from your Python interpreter.

```
>>> classifier = random_mac.classifier.restore(
...   file="./random-mac-classifier.pickled"
... )
```

CHAPTER 4

Workflows for random-mac

4.1 Make, train, test, and save

```
# Import modules.

>>> import sklearn.model_selection
>>> import random_mac

# Make a dataset.

>>> data, labels = random_mac.dataset.make(
...   2,
...   oui_file="./oui.csv",
...   cid_file="./cid.csv"
... )

# Split the dataset.

>>> training_data, testing_data, training_labels, testing_labels = sklearn.model_
↪selection.train_test_split(data, labels)

# Make, train, and test a classifier.

>>> classifier = random_mac.classifier.make()
>>> classifier = random_mac.classifier.train(
...   classifier,
...   training_data,
...   training_labels
... )
>>> score = random_mac.classifier.test(
...   classifier,
...   testing_data,
...   testing_labels
... )
```

(continues on next page)

(continued from previous page)

```
 >>> print("score = {}%".format(str(int(100 * score))))
 score = 83%

# Save the classifier.

>>> random_mac.classifier.save(
...   classifier,
...   file="./random-mac-classifier.pickled"
... )
```

4.2 Restore and use

```
 # Import module.

 >>> import random_mac

 # Find a MAC address in a host's ARP cache, a switch's MAC address table, etc.

 >>> address = "aabbccddeeff"

 # Restore the classifier.

 >>> classifier = random_mac.classifier.restore(file="./random-mac-classifier.pickled
↪")

 # Use the classifier.

 >>> result = random_mac.is_random_mac(classifier, address)
 >>> print(result)
True
```

CHAPTER 5

Testing random-mac

To conduct testing, run the following command from your shell.

```
[user@host random-mac]$ pytest --cov --cov-report=term-missing
```

CHAPTER 6

random_mac

6.1 random_mac package

6.1.1 random_mac.classifier module

This module contains classifier-related functions.

random_mac.classifier.**make**()
Retrieve a classifier.

Returns A classifier.

Return type sklearn classifier

random_mac.classifier.**train**(*classifier*, *data*, *labels*)
Train a classifier.

Parameters

- **classifier** (*sklearn classifier*) – The classifier to train.
- **data** (*numpy array*) – The data with which to train the classifier.
- **labels** (*numpy array*) – The labels with which to train the classifier.

Returns The trained classifier.

Return type sklearn classifier

random_mac.classifier.**test**(*classifier*, *data*, *labels*)
Test a classifier.

Parameters

- **classifier** (*sklearn classifier*) – The classifier to test.
- **data** (*numpy array*) – The data with which to test the classifier.
- **labels** (*numpy array*) – The labels with which to test the the classifier.

Returns The results of the test.

Return type float

random_mac.classifier.**save**(*classifier*, *file='random-mac-classifier.pickled'*)
Save (pickle) a classifier.

Parameters

- **classifier** (*sklearn classifier*) – The classifier to save.
- **file** (*str*) – The name of the destination file.

random_mac.classifier.**restore**(*file='random-mac-classifier.pickled'*)
Restore (unpickle) a classifier.

Parameters **file** (*str*) – The name of the source file.

Returns The restored (unpickled) classifier.

Return type sklearn classifier

random_mac.classifier.**is_random_mac**(*classifier*, *address*)
Determine whether a MAC address is random or non-random.

Parameters

- **classifier** (*sklearn classifier*) – The classifier to use.
- **address** (*str (hexadecimal)*) – The address to test.

Returns Whether the given MAC address is random (True) or non-random (False).

Return type bool

6.1.2 random_mac.dataset module

This module contains dataset-related functions.

random_mac.dataset.**get_ieee_assignments**(*file*)
Retrieve OUIs and CIDs.

Parameters **file** (*str*) – The name of a file with information on OUIs and CIDs assigned by the IEEE.

Typical names are *oui.csv* and *cid.csv*.

Returns A list of 24-bit OUIs or CIDs assigned by the IEEE.

Return type list

random_mac.dataset.**make_hexadecimal_digit_strings**(*assignments*)
Make hexadecimal strings based upon OUIs and CIDs.

Parameters **assignments** (*list*) – A list of 24-bit OUIs or CIDs assigned by the IEEE.

Returns A list of 48-bit hexadecimal strings, where each string is the concatenation of a 24-bit OUI/CID and 24 random bits.

Return type list

random_mac.dataset.**make_random_hexadecimal_digit_strings**(*number*)
Make random hexadecimal strings.

Parameters **number** (*int*) – The number of hexadecimal strings to make.

Returns A list of 48-bit hexadecimal strings, where each string is 48 random bits.

Return type list

random_mac.dataset.**get_mac_features**(*digit_string*)
Retrieve the features of a MAC address.

Parameters **digit_string** (*str*) – A 48-bit hexadecimal string with which to instantiate *MediaAccessControlAddress*.

Returns

An eight-tuple with the features of a MAC address.

The features are *type*, *has_oui*, *has_cid*, *is_broadcast*, *is_multicast*, *is_unicast*, *is_uaa*, and *is_laa*.

Return type tuple

random_mac.dataset.**get_features**(*digit_strings*)
Retrieve the features of MAC addresses.

Parameters **digit_strings** (*list*) – A list of 48-bit hexadecimal strings.

Returns A list of tuples, where each tuple contains the features of a MAC address.

Return type list

random_mac.dataset.**normalize_features**(*features*)
Normalize the features of MAC addresses.

Parameters **features** (*list*) – A list of tuples, where each tuple contains the features of a MAC address.

Returns A numpy array with the normalized features of MAC addresses, where normalization means replacing non-numeric with numeric values and converting the container from a list to a numpy array.

Return type numpy array

random_mac.dataset.**make_labels**(*value*, *number*)
Make labels for training and testing of a binary classifier.

Parameters

- **value** (*int*) – The label, where *0* means a non-random MAC addresses and *1* means a random MAC address.
- **number** (*int*) – The number of labels.

Returns A list with the given number of the given label.

Return type list

random_mac.dataset.**normalize_labels**(*labels*)
Normalize labels.

Parameters **labels** (*list*) – A list of labels.

Returns A numpy array with normalized labels, where normalization means converting the container from a list to a numpy array.

Return type numpy array

random_mac.dataset.**make**(*multiple*, *oui_file='./oui.csv'*, *cid_file='./cid.csv'*)
Make a dataset for training and testing purposes.

Parameters

- **multiple** (*int*) – The number of random MAC addresses to create for every non-random MAC address.
- **oui_file** (*str*) – The name of the file with OUIs assigned by the IEEE.
- **cid_file** (*str*) – The name of the file with CIDs assigned by the IEEE.

Returns A tuple with data (features) and labels.

Return type tuple

6.1.3 Module contents

Use machine learning to identify randomly-generated MAC addresses.

random_mac.**is_random_mac**(*classifier*, *address*)

Determine whether a MAC address is random or non-random.

Parameters

- **classifier** (*sklearn classifier*) – The classifier to use.
- **address** (*str (hexadecimal)*) – The address to test.

Returns Whether the given MAC address is random (True) or non-random (False).

Return type bool

CHAPTER 7

Indices

- genindex
- modindex
- search

Python Module Index

r

Index