lantern

Release 0.0.0

Jun 09, 2019

Contents

lantern is a cryptanalysis library to assist with the identification and breaking of classical ciphers. The library provides general purpose analysis tools, as well as premade modules to break well known ciphers.

```
from lantern.modules import shift
from lantern import fitness

ciphertext = "iodj{EuxwhIrufhLvEhvwIrufh}"

decryptions = shift.crack(ciphertext, fitness.english.quadgrams)
print(decryptions[0])
```

In short, lantern can be used to:

- **Identify** ciphers from ciphertext
- **Automatically crack** well known ciphers
- **Analyze** ciphertext to assist in the breaking of custom crypto systems

CHAPTER 1

Install

```
pip3 install -U lantern
```

CHAPTER 2

Guide

Coming Soon

CHAPTER 3

API Reference

3.1 Score

Scoring algorithm to return probability of correct decryption. Output range depends on the score functions used.

lantern.**score**(*text*, **score_functions*)

Score `text` using `score_functions`.

Examples

```
>>> score("abc", function_a)
>>> score("abc", function_a, function_b)
```

Parameters

- **text** (*str*) – The text to score
- ***score_functions** (*variable length argument list*) – functions to score with

Returns Arithmetic mean of scores

Raises **ValueError** – If score_functions is empty

3.2 Util

Utility functions to format and marshal data.

lantern.util.**combine_columns**(*columns*)

Combine `columns` into a single string.

Example

```
>>> combine_columns(['eape', 'xml'])
'example'
```

Parameters **columns** (*iterable*) – ordered columns to combine

Returns String of combined columns

lantern.util.**group**(*text*, *size*)
Group `text` into blocks of `size`.

Example

```
>>> group("test", 2)
['te', 'st']
```

Parameters

- **text** (*str*) – text to separate
- **size** (*int*) – size of groups to split the text into

Returns List of n-sized groups of text

Raises **ValueError** – If n is non positive

lantern.util.**iterate_ngrams**(*text*, *n*)
Generator to yield ngrams in `text`.

Example

```
>>> for ngram in iterate_ngrams("example", 4):
...     print(ngram)
exam
xamp
ampl
mple
```

Parameters

- **text** (*str*) – text to iterate over
- **n** (*int*) – size of window for iteration

Returns Generator expression to yield the next ngram in the text

Raises **ValueError** – If n is non positive

lantern.util.**remove**(*text*, *exclude*)
Remove `exclude` symbols from `text`.

Example

```
>>> remove("example text", string.whitespace)
'exampletext'
```

Parameters

- **text** (*str*) – The text to modify
- **exclude** (*iterable*) – The symbols to exclude

Returns `text` with `exclude` symbols removed

`lantern.util.`**`split_columns`**(*text*, *n_columns*)

Split `text` into `n_columns` many columns.

Example

```
>>> split_columns("example", 2)
['eape', 'xml']
```

Parameters

- **text** (*str*) – The text to split
- **n_columns** (*int*) – The number of columns to create

Returns List of columns

Raises **ValueError** – If n_cols is <= 0 or >= len(text)

3.3 Modules

Note: fitness_functions in every module must return a value such that a lower score means the text is closer to the target.

3.3.1 Shift

Automated breaking of the Shift Cipher.

`lantern.modules.shift.`**`crack`**(*ciphertext*, **fitness_functions*, *min_key=0*, *max_key=26*, *shift_function=<function make_shift_function.<locals>.shift_case_sensitive>*)

Break `ciphertext` by enumerating keys between `min_key` and `max_key`.

Example

```
>>> decryptions = crack("KHOOR", fitness.english.quadgrams)
>>> print(''.join(decryptions[0].plaintext))
HELLO
```

Parameters

- **ciphertext** (*iterable*) – The symbols to decrypt
- ***fitness_functions** (*variable length argument list*) – Functions to score decryption with

Keyword Arguments

- **min_key** (*int*) – Key to start with
- **max_key** (*int*) – Key to stop at (exclusive)
- **shift_function** (*function(shift, symbol)*) – Shift function to use

Returns Sorted list of decryptions

Raises

- **ValueError** – If min_key exceeds max_key
- **ValueError** – If no fitness_functions are given

lantern.modules.shift.**decrypt**(*key: int, ciphertext: Iterable[T_co], shift_function: Callable[[int, object], object] = <function make_shift_function.<locals>.shift_case_sensitive>*) → Iterable[T_co]

Decrypt Shift enciphered `ciphertext` using `key`.

Examples

```
>>> ''.join(decrypt(3, "KHOOR"))
HELLO
```

```
>>> decrypt(15, [0xed, 0xbc, 0xcd, 0xfe], shift_bytes)
[0xde, 0xad, 0xbe, 0xef]
```

Parameters

- **key** (*int*) – The shift to use
- **ciphertext** (*iterable*) – The symbols to decrypt
- **shift_function** (*function (shift, symbol)*) – Shift function to apply to symbols in the ciphertext

Returns Decrypted text

lantern.modules.shift.**encrypt**(*key: int, plaintext: Iterable[T_co], shift_function: Callable[[int, object], object] = <function make_shift_function.<locals>.shift_case_sensitive>*) → Iterable[T_co]

Encrypt `plaintext` with `key` using the Shift cipher.

Examples

```
>>> ''.join(encrypt(3, "HELLO"))
KHOOR
```

```
>>> encrypt(15, [0xde, 0xad, 0xbe, 0xef], shift_bytes)
[0xed, 0xbc, 0xcd, 0xfe]
```

Parameters

- **key** (*int*) – The shift to use
- **plaintext** (*iterable*) – The symbols to encrypt
- **shift_function** (*function (shift, symbol)*) – Shift function to apply to symbols in the plaintext

Returns Encrypted text

lantern.modules.shift.**make_shift_function**(*alphabet: Iterable[T_co], operator: Callable[[int, int], int] = <function <lambda>>*) → Callable[[int, object], object]

Construct a shift function from an alphabet.

Examples

Shift cases independently

```
>>> make_shift_function([string.ascii_uppercase, string.ascii_lowercase])
```

Additionally shift punctuation characters

```
>>> make_shift_function([string.ascii_uppercase, string.ascii_lowercase, string.
↪punctuation])
```

Shift entire ASCII range, overflowing cases

```
>>> make_shift_function([''.join(chr(x) for x in range(32, 127))])
```

Parameters **alphabet** (*iterable*) – Ordered iterable of strings representing separate cases of an alphabet

Returns int, symbol: object)

Return type Function (shift

3.3.2 Simple Substitution

Automated breaking of the Simple Substitution Cipher.

lantern.modules.simplesubstitution.**crack**(*ciphertext, *fitness_functions, ntrials=30, nswaps=3000*)

Break `ciphertext` using hill climbing.

Note: Currently ntrails and nswaps default to magic numbers. Generally the trend is, the longer the text, the lower the number of trials you need to run, because the hill climbing will lead to the best answer faster. Because randomness is involved, there is the possibility of the correct decryption not being found. In this circumstance you just need to run the code again.

Example

```
>>> decryptions = crack("XUOOB", fitness.english.quadgrams)
>>> print(decryptions[0])
HELLO
```

Parameters

- **ciphertext** (*str*) – The text to decrypt
- ***fitness_functions** (*variable length argument list*) – Functions to score decryption with

Keyword Arguments

- **ntrials** (*int*) – The number of times to run the hill climbing algorithm
- **nswaps** (*int*) – The number of rounds to find a local maximum

Returns Sorted list of decryptions

Raises

- **ValueError** – If nswaps or ntrails are not positive integers
- **ValueError** – If no fitness_functions are given

lantern.modules.simplesubstitution.**decrypt**(*key*, *ciphertext*)

Decrypt Simple Substitution enciphered `ciphertext` using `key`.

Example

```
>>> decrypt("PQSTUVWXYZCODEBRAKINGFHJLM", "XUOOB")
HELLO
```

Parameters

- **key** (*iterable*) – The key to use
- **ciphertext** (*str*) – The text to decrypt

Returns Decrypted ciphertext

3.3.3 Vigenere

Automated breaking of the Vigenere Cipher.

lantern.modules.vigenere.**crack**(*ciphertext*, **fitness_functions*, *key_period=None*, *max_key_period=30*)

Break `ciphertext` by finding (or using the given) key_period then breaking `key_period` many Caesar ciphers.

Example

```
>>> decryptions = crack("OMSTV", fitness.ChiSquared(analysis.frequency.english.
↪unigrams))
>>> print(decryptions[0])
HELLO
```

Parameters

- **ciphertext** (*str*) – The text to decrypt
- ***fitness_functions** (*variable length argument list*) – Functions to score decryption with

Keyword Arguments

- **key_period** (*int*) – The period of the key
- **max_key_period** (*int*) – The maximum period the key could be

Returns Sorted list of decryptions

Raises

- **ValueError** – If key_period or max_key_period are less than or equal to 0
- **ValueError** – If no fitness_functions are given

lantern.modules.vigenere.**decrypt**(*key*, *ciphertext*)

Decrypt Vigenere encrypted `ciphertext` using `key`.

Example

```
>>> decrypt("KEY", "RIJVS")
HELLO
```

Parameters

- **key** (*iterable*) – The key to use
- **ciphertext** (*str*) – The text to decrypt

Returns Decrypted ciphertext

lantern.modules.vigenere.**key_periods**(*ciphertext*, *max_key_period*)

Rank all key periods for `ciphertext` up to and including `max_key_period`

Example

```
>>> key_periods(ciphertext, 30)
[2, 4, 8, 3, ...]
```

Parameters

- **ciphertext** (*str*) – The text to analyze
- **max_key_period** (*int*) – The maximum period the key could be

Returns Sorted list of keys

Raises **ValueError** – If max_key_period is less than or equal to 0

3.4 Analysis

3.4.1 Frequency

General purpose frequency analysis tools.

lantern.analysis.frequency.**ENGLISH_IC = 0.06505393453880672**
Index of coincidence for the English language.

lantern.analysis.frequency.**chi_squared**(*source_frequency*, *target_frequency*)
Calculate the Chi Squared statistic by comparing `source_frequency` with `target_frequency`.

Example

```
>>> chi_squared({'a': 2, 'b': 3}, {'a': 1, 'b': 2})
0.1
```

Parameters

- **source_frequency** (*dict*) – Frequency map of the text you are analyzing
- **target_frequency** (*dict*) – Frequency map of the target language to compare with

Returns Decimal value of the chi-squared statistic

lantern.analysis.frequency.**english = <lantern.structures.dynamicdict.DynamicDict object>**
English ngram frequencies.

lantern.analysis.frequency.**frequency_analyze**(*text*, *n=1*)
Analyze the frequency of ngrams for a piece of text.

Examples

```
>>> frequency_analyze("abb")
{'a': 1, 'b': 2}
```

```
>>> frequency_analyze("abb", 2)
{'ab': 1, 'bb': 1}
```

Parameters

- **text** (*str*) – The text to analyze
- **n** (*int*) – The ngram size to use

Returns Dictionary of ngrams to frequency

Raises **ValueError** – If n is not a positive integer

lantern.analysis.frequency.**frequency_to_probability**(*frequency_map*, *decorator=<function <lambda>>*)
Transform a `frequency_map` into a map of probability using the sum of all frequencies as the total.

Example

```
>>> frequency_to_probability({'a': 2, 'b': 2})
{'a': 0.5, 'b': 0.5}
```

Parameters

- **frequency_map** (*dict*) – The dictionary to transform
- **decorator** (*function*) – A function to manipulate the probability

Returns Dictionary of ngrams to probability

lantern.analysis.frequency.**index_of_coincidence**(**texts*)

Calculate the index of coincidence for one or more `texts`. The results are averaged over multiple texts to return the delta index of coincidence.

Examples

```
>>> index_of_coincidence("aabbc")
0.2
```

```
>>> index_of_coincidence("aabbc", "abbcc")
0.2
```

Parameters ***texts** (*variable length argument list*) – The texts to analyze

Returns Decimal value of the index of coincidence

Raises

- **ValueError** – If texts is empty
- **ValueError** – If any text is less that 2 character long

3.4.2 Search

Algorithms for searching and optimisation.

lantern.analysis.search.**hill_climb**(*nsteps*, *start_node*, *get_next_node*)

Modular hill climbing algorithm.

Example

```
>>> def get_next_node(node):
...     a, b = random.sample(range(len(node)), 2)
...     node[a], node[b] = node[b], node[a]
...     plaintext = decrypt(node, ciphertext)
...     score = lantern.score(plaintext, *fitness_functions)
...     return node, score, Decryption(plaintext, ''.join(node), score)
>>> final_node, best_score, outputs = hill_climb(10, "ABC", get_next_node)
```

Parameters

- **nsteps** (*int*) – The number of neighbours to visit

- **start_node** – The starting node
- **get_next_node** (*function*) – Function to return the next node the score of the current node and any optional output from the current node

Returns The highest node found, the score of this node and the outputs from the best nodes along the way

3.5 Fitness Functions

3.5.1 Chi Squared

Chi Squared Scoring function.

lantern.fitness.chisquared.**ChiSquared**(*target_frequency*)
Score a text by comparing its frequency distribution against another.

Note: It is easy to be penalised without knowing it when using this scorer. English frequency ngrams are capital letters, meaning when using it any text you score against must be all capitals for it to give correct results. I am aware of the issue and will work on a fix.

Todo: Maybe include paramter for ngram size. Havent had a use case for this yet. Once there is evidence it is needed, I will add it.

Example

```
>>> fitness = ChiSquared(english.unigrams)
>>> fitness("ABC")
-32.2
```

Parameters **target_frequency** (*dict*) – symbol to frequency mapping of the distribution to compare with

3.5.2 Corpus

Score plaintext based on number of words identified are in the corpus.

class lantern.fitness.corpus.**Corpus**(*corpus*)
Scoring function based on existance of words in a corpus.

Todo: This is fairly broken. I'm not happy with this implementation and will be changing it in the future when I revisit weighted mean scoring

---

__call__(*text*)
Score based on number of words not in the corpus.

Example

```
>>> fitness = Corpus(["example"])
>>> fitness("example")
0
```

```
>>> fitness("different")
-2.0
```

Parameters **text** (*str*) – The text to score

Returns Corpus score for text

__init__(*corpus*)

Build function with set of words from a corpus.

Parameters **corpus** (*collection*) – collection of words to use

3.5.3 Ngram

Fitness scoring using ngram frequency.

lantern.fitness.ngram.**NgramScorer**(*frequency_map*)

Compute the score of a text by using the frequencies of ngrams.

Example

```
>>> fitness = NgramScorer(english.unigrams)
>>> fitness("ABC")
-4.3622319742618245
```

Parameters **frequency_map** (*dict*) – ngram to frequency mapping

lantern.fitness.ngram.**english = <lantern.structures.dynamicdict.DynamicDict object>**

English ngram scorers.

3.5.4 Pattern Match

Fitness scoring using pattern matching.

lantern.fitness.patternmatch.**PatternMatch**(*regex*)

Compute the score of a text by determing if a pattern matches.

Example

```
>>> fitness = PatternMatch("flag{.*}")
>>> fitness("flag{example}")
0
```

```
>>> fitness("junk")
-1
```

Parameters **regex** (*str*) – regular expression string to use as a pattern

3.6 Structures

3.6.1 Decryption

Class to group information about a decryption.

Todo: Possibly add more functionality to this class * Equality checking * Formatted plaintext (added spaces) Once there is evidence these things are needed, I will implement them

class lantern.structures.decryption.**Decryption**(*plaintext*, *key*, *score*)
A decryption object, composed of plaintext, a score and the key.

Example

```
>>> decryption = Decryption("example", "key", -10)
>>> decryption.plaintext
example
>>> decryption.key
key
>>> decryption.score
-10
```

__init__(*plaintext*, *key*, *score*)

Parameters

- **plaintext** – The decrypted ciphertext
- **key** – The key which resulted in this decryption
- **score** – The score of this decryption

__lt__(*other*)
Compare decryptions with other decryptions by score.

Parameters **other** – Object to compare with

Returns True if self is less than other, else False

3.6.2 DynamicDict

Class to dynamically create attributes only when they are needed.

Todo: This needs some more functionality. Specifically it doesnt behave like a proper dictionary

class lantern.structures.dynamicdict.**DynamicDict**(*builders={}*)
Dictionary which builds values when they are accessed for the first time.

Example

```
>>> ngrams = DynamicDict({
...     'trigrams': lambda: load_ngrams('trigrams'),
...     'quadgrams': lambda: load_ngrams('quadgrams')
... })
```

Since trigrams and quadgrams are large files, its expensive to load them in if theyre not needed. Using the DynamicDict ensures they are only loaded when they are accessed for the first time.

__getattr__(*name*)
Attempt to build values that are not already created.

__init__(*builders={}*)
Instantiate dict with mapping of keys to builders.

Parameters **builders** (*dict*) – key to function mapping

Python Module Index

l

Index

Symbols

C

D

E

F

G

H

I

K

L

M

N

P

R

S