NLTK Documentation

nltk.probability.FreqDist

class nltk.probability.FreqDist [source]

Bases: `Counter`

A frequency distribution for the outcomes of an experiment. A frequency distribution records the number of times each outcome of an experiment has occurred. For example, a frequency distribution could be used to record the frequency of each word type in a document. Formally, a frequency distribution can be defined as a function mapping from each sample to the number of times that sample occurred as an outcome.

Frequency distributions are generally constructed by running a number of experiments, and incrementing the count for a sample every time it is an outcome of an experiment. For example, the following code will produce a frequency distribution that encodes how often each word occurs in a text:

```
>>> from nltk.tokenize import
>>> from nltk.probability im
>>> sent = 'This is an examp
>>> fdist = FreqDist()
>>> for word in word_tokenize
...     fdist[word.lower()] +=
```

An equivalent way to do this is with the initializer:

```
>>> fdist = FreqDist(word.lo
```

__init__(*samples=None*) [source]

Construct a new frequency distribution. If `samples` is given, then the frequency distribution will be initialized with the count of each object in `samples`; otherwise, it will be initialized to be empty.

In particular, `FreqDist()` returns an empty frequency distribution; and `FreqDist(samples)` first creates an empty frequency distribution, and then calls `update` with the list `samples`.

Parameters

samples (*Sequence*) – The samples to initialize the

frequency distribution with.

N() [source]

Return the total number of sample outcomes that have been recorded by this FreqDist. For the number of unique sample values (or bins) with counts greater than zero, use `FreqDist.B()`.

Return type

int

update(**args*, ***kwargs*) [source]

Override `Counter.update()` to invalidate the cached N

setdefault(*key*, *val*) [source]

Override `Counter.setdefault()` to invalidate the cached N

B() [source]

Return the total number of sample values (or “bins”) that have counts greater than zero. For the total number of sample outcomes recorded, use `FreqDist.N()`. (FreqDist.B() is the same as len(FreqDist).)

Search

NLTK Team

Return type

int

hapaxes() [source]

Return a list of all samples that occur once (hapax legomena)

Return type

list

Nr(*r*, *bins=None*) [source]

r_Nr(*bins=None*) [source]

Return the dictionary mapping r to Nr, the number of samples with frequency r, where Nr > 0.

Parameters

bins (*int*) – The number of possible sample outcomes. `bins` is used to calculate Nr(0). In particular, Nr(0) is `bins-self.B()`. If `bins` is not specified, it defaults to `self.B()` (so Nr(0) will be 0).

Return type

int

freq(*sample*) [source]

Return the frequency of a given sample. The frequency of a sample is defined as the count of that sample divided by the total number of sample outcomes that have been recorded by this FreqDist. The count of a sample is defined as the number of times that sample outcome was recorded by this FreqDist. Frequencies are always real numbers in the range [0, 1].

Parameters

sample (*any*) – the sample whose frequency should be returned.

Return type

float

max() [source]

Return the sample with the greatest number of outcomes in this frequency distribution. If two or more samples have the same number of outcomes, return one of them; which sample is returned is undefined. If no outcomes have occurred in this frequency distribution, return None.

Returns

The sample with the maximum number of outcomes in this frequency distribution.

Return type

any or None

plot(*args, title='', cumulative=False, percents=False, show=True, **kwargs) [source]

Plot samples from the frequency distribution displaying the most frequent sample first. If an integer parameter is supplied, stop after this many samples have been plotted. For a cumulative plot, specify cumulative=True. Additional `**kwargs` are passed to matplotlib's plot function. (Requires Matplotlib to be installed.)

Parameters

- **title** (*str*) – The title for the graph.
- **cumulative** (*bool*) – Whether the plot is cumulative. (default = False)
- **percents** (*bool*) – Whether the plot uses percents instead of

counts. (default = False)
- **show** (*bool*) – Whether to show the plot, or only return the ax.

tabulate(**args, **kwargs*) [source]

Tabulate the given samples from the frequency distribution (cumulative), displaying the most frequent sample first. If an integer parameter is supplied, stop after this many samples have been plotted.

Parameters

- **samples** (*list*) – The samples to plot (default is all samples)
- **cumulative** – A flag to specify whether the freqs are cumulative (default = False)

copy() [source]

Create a copy of this frequency distribution.

Return type

FreqDist

pprint(*maxlen=10, stream=None*) [source]

Print a string representation of this FreqDist to ‘stream’

Parameters

- **maxlen** (*int*) – The maximum number of items to print
- **stream** – The stream to print to. stdout by default

pformat(*maxlen=10*) [source]

Return a string representation of this FreqDist.

Parameters

maxlen (*int*) – The maximum number of items to display

Return type

string

__new__(***kwargs*)

clear() → None. Remove all items from D.

elements()

Iterator over elements repeating each as many times as its count.

```
>>> c = Counter('ABCABC'
>>> sorted(c.elements())
['A', 'A', 'B', 'B', 'C'
```

Knuth's example for prime factors of 1836: 2**2 * 3**3 * 17**1 >>> prime_factors = Counter({2: 2, 3: 3, 17: 1}) >>> product = 1 >>> for factor in prime_factors.elements(): # loop over factors ... product *= factor # and multiply them >>> product 1836

Note, if an element's count has been set to zero or is a negative number, elements() will ignore it.

classmethod fromkeys(*iterable*, *v=None*)

Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, */*)

Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

most_common(*n=None*)

List the n most common elements and their counts from the most common to the least. If n is None, then list all element counts.

```
>>> Counter('abracadabra
[('a', 5), ('b', 2), ('r
```

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, default is returned if given, otherwise KeyError is raised

popitem()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

subtract(*iterable=None*, */*, ***kwds*)

Like dict.update() but subtracts counts instead of replacing them. Counts can be reduced below zero. Both the inputs and outputs are allowed to contain zero and negative counts.

Source can be an iterable, a dictionary, or another Counter instance.

```
>>> c = Counter('which')
>>> c.subtract('witch')
>>> c.subtract(Counter('\
>>> c['h']
0
>>> c['w']
-1
```

values() → an object providing a view on D's values