NLTK Tutorial: Probability

Edward Loper

Table of Contents

1. Experiments and Samples

The `nltk.probability` module can be used to model probablistic phenomena. Most probablistic phenomena can be thought of as experiments. An *experiment* is any process which leads to a well-defined outcome. For example, rolling a die is an experiment whose possible outcomes are 1, 2, 3, 4, 5, and 6.

A *sample* is any possible outcome of a given experiment. In `nltk.probability`, any immutable Python value or object can be a sample. Typical samples are strings, integers, Tokens, and tuples. We can use a simple Python procedure to define the experiment of rolling a die; its samples are 1, 2, 3, 4, 5, and 6:

```
>>> import random
>>> def roll():
...     return random.choice( [1, 2, 3, 4, 5, 6] )
>>> roll()
4
```

2. Frequency Distributions

A frequency distribution records the number of times each outcome of an experiment has occured. For example, a frequency distribution could be used to record the frequency of each word type in a document. Frequency distributions are encoded by the `FreqDist` class, which is defined by the `nltk.probability` module.

2.1. Constructing a Frequency Distribution

The `FreqDist` constructor creates a new empty frequency distribution:

```
>>> freq_dist = FreqDist()
<FreqDist with 0 outcomes>
```

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

```
>>> freq_dist = FreqDist()
>>> for token in document:
...     freq_dist.inc(token.type())
```

2.2. Using a Frequency Distribution

Once we construct a frequency distribution that records the outcomes of an experiment, we can use it to examine a number of interesting properties of the experiment. This section describes the most important accessors that are defined for frequency distributions.

We can find the number of times a given sample occured with the `count` method:

```
# How many times did "the" occur?
>>> freq_dist.count('the')
```

The `freq` method returns the frequency of a given sample:

```
# What was the frequency of the word "the"?
>>> freq_dist.freq('the')
0.012
```

We can find the total number of sample outcomes recorded by a frequency distribution with the `N` method:

```
# How many words were counted?
>>> freq_dist.N()
500
```

The `samples` method returns a list of all samples that have been recorded as outcomes by a frequency distribution:

```
# What words were encountered?
>>> freq_dist.samples()[:5]
['happy', 'but', 'the', 'in', 'of']
```

We can find the sample with the greatest number of outcomes with the `max` method:

```
>>> freq_dist.max()
# What was the most common word?
'the'
```

See the reference documentation for `FreqDistI` for more information on how to use frequency distributions.

2.3. Example: Word Lengths

In this section, we use a `FreqDist` to examine the distribution of word lengths in a corpus. In particular, we find the distribution of word lengths for...

- All words in a corpus.
- Words that end in vowels.
- Words following words that end in vowels.

In each case, we construct a frequency distribution whose samples are word lengths; and plot the results.

To begin with, we load a corpus from a text file:

```
>>> corpus = open('corpus.txt').read()
>>> tokens = WSTokenizer().tokenize(corpus)
```

2.3.1. All words in a corpus

To find the first distribution, we examine each token in the corpus, and find the length of its token. This length is the "outcome" for our experiment, so we use `inc()` to increment its count in a frequency distribution.

```
# What is the distribution of word lengths in a corpus?
>>> freq_dist = FreqDist()
>>> for token in tokens:
...     freq_dist.inc(len(token.type()))
```

To plot the results, we first create a sorted list of all word lengths that were encountered in the document. We then construct a list of points, where the x coordinate is the word length, and the y coordinate is the frequency with which that word length is used:

```
# Plot the results.
>>> wordlens = freq_dist.samples()
>>> wordlens.sort()
>>> points = [(l, freq_dist.freq(l)) for l in wordlens]
>>> Plot(points)
```

Note: We are currently using a fairly simple class to plot functions. We will likely replace it with a more advanced plotting system in the future.

2.3.2. Words ending in vowels

For the second distribution, we only care about word lengths for words that end in vowels. This specification of which word lengths we care about is known as a *condition*. In the next section, we will explore ways of encoding the frequency distributions for a single experiment under a variety of related conditions.

To find the second distribution, we examine each token in the corpus. If it ends in a vowel, then we increment the count for its length in a frequency distribution.

```
# For this example, we define vowels as "a", "e", "i", "o", and "u"
>>> VOWELS = ('a', 'e', 'i', 'o', 'u')

# What is the distribution of word lengths for words that
# end in vowels?
>>> freq_dist = FreqDist()
>>> for token in tokens:
...     if token.type()[-1].lower() in VOWELS:
...         freq_dist.inc(len(token.type()))

# Plot the results
>>> wordlens = freq_dist.samples()
>>> wordlens.sort()
>>> points = [(l, freq_dist.freq(l)) for l in wordlens]
>>> Plot(points)
```

2.3.3. Words following words ending in vowels

For the third distribution, we only care about word lengths for words following words that end in vowels. We can use a boolean variable, `ended_in_vowel`, to keep track of this condition. Initially, its value is zero; and after we examine each token, we update its value for use with the next token.

```
# What is the distribution of word lengths for words following
# words that end in vowels?
>>> ended_in_vowel = 0 # Did the last word end in a vowel?
>>> freq_dist = FreqDist()
>>> for token in tokens:
...     if ended_in_vowel:
...         freq_dist.inc(len(token.type()))
...     ended_in_vowel = token.type()[-1].lower() in VOWELS

# Plot the results
>>> wordlens = freq_dist.samples()
>>> wordlens.sort()
>>> points = [(l, freq_dist.freq(l)) for l in wordlens]
>>> Plot(points)
```

3. Conditional Frequency Distributions

A *condition* specifies the context in which an experiment is performed. Often, we are interested in the effect that conditions have on the outcome for an experiment. For example, we might want to examine how the distribution of a word's length (the outcome) is affected by the word's initial letter (the condition). Conditional frequency distributions provide a tool for exploring this type of question.

A *conditional frequency distribution* is a collection of frequency distributions for the same experiment, run under different conditions. The individual frequency distributions are indexed by the condition. Conditional frequency distributions are represented using the `ConditionalFreqDist` class, which is defined by the `nltk.probability` module.

The `ConditionalFreqDist` constructor creates a new empty conditional frequency distribution:

```
>>> cfdist = ConditionalFreqDist()
<ConditionalFreqDist with 0 conditions>
```

To access the frequency distribution for a condition, use the indexing operator:

```
>>> cfdist['a']
<FreqDist with 0 outcomes>

# Record the word lengths of some words starting with 'a'
>>> for word in 'apple and arm'.split():
...     cfdist['a'].inc(len(word))

# Of words starting with 'a', how many are 3 characters long?
>>> cfdist['a'].freq(3)
0.66667
```

There is no need to explicitly specify the set of conditions used by a conditional frequency distribution. Whenever you use the indexing operator to access the frequency distribution for a new condition, `ConditionalFreqDist` automatically creates a new empty `FreqDist` for it. To list the conditions which have been accessed for a conditional frequency distribution, use the `conditions` method:

```
>>> cfdist.conditions()
['a']
```

3.1. Example: Conditioning on a Word's Initial Letter

In this section, we use a `ConditionalFreqDist` to examine how the distribution of a word's length is affected by the word's initial letter. To begin, we load a corpus from a text file, and create an empty `ConditionalFreqDist`:

```
>>> corpus = open('corpus.txt').read()
>>> tokens = WSTokenizer().tokenize(corpus)

>>> cfdist = ConditionalFreqDist()
```

We then examine each token in the corpus, and determine its initial letter (the condition) and its word length (the outcome). We use the indexing operator to access the frequency distribution for the condition, and use the `inc()` method to increment its count for the outcome.

```
# How does initial letter affect word length?
>>> for token in tokens:
...     outcome = len(token.type())
...     condition = token.type()[0].lower()
...     cfdist[condition].inc(outcome)
```

We can construct a plot to show the frequency of various word lengths (outcomes) for a given intial letter (condition):

```
# Plot the distribution of word lengths for words starting with 'a'
>>> wordlens = cfdist['a'].samples()
>>> wordlens.sort()
>>> points = [(l, cfdist['a'].freq(l)) for l in wordlens]
>>> Plot(points)
```

We can also construct a plot to show how the frequency of a word length (outcome) depends on the initial letter (condition):

```
# Plot the frequency of 3-letter-words for each initial letter.
>>> conditions = cfdist.conditions()
>>> conditions.sort()
>>> points = [(ord(c), cfdist[c].freq(3)) for c in conditions]
>>> Plot(points)
```

3.2. Prediction

Conditional frequency distributions are often used for prediction. *Prediction* is the problem of deciding a likely outcome for a given run of an experiment. The decision of which outcome to predict is usually based on the context in which the experiment is performed. For example, we might try to predict a word's type (outcome), based on the type of the word that it follows (context).

To predict the outcomes of an experiment, we first examine a representative *training corpus*, where the context and outcome for each run of the experiment are known. When presented with a new run of the experiment, we simply choose the outcome that occured most frequently for the experiment's context.

We can use a `ConditionalFreqDist` to find the most frequent occurance for each context. First, we record each outcome in the training corpus, using the context that the experiment was under as the condition. Then, we can access the frequency distribution for a given context with the indexing operator, and use the `max()` method to find the most likely outcome.

3.2.1. Example: Predicting Words

In this section, we use a `ConditionalFreqDist` to predict a word's type, based on the type of the word that it follows. To begin, we load a corpus from a text file, and create an empty `ConditionalFreqDist`:

```
>>> corpus = open('corpus.txt').read()
>>> tokens = WSTokenizer().tokenize(corpus)

>>> cfdist = ConditionalFreqDist()
```

We then examine each token in the corpus, and increment the appropriate sample's count. We use the variable `context` to record the type of the preceeding word.

```
>>> context = None  # The type of the preceeding word
>>> for token in tokens:
...     outcome = token.type()
...     cfdist[context].inc(outcome)
...     context = token.type()
```

Note: Sometimes the context for an experiment is unavailable, or does not exist. For example, the first token in a text does not follow any word. In these cases, we must decide what context to use. For this example, we use `None` as the context for the first token. Another option would be to simply discard the first token.

Once we have constructed a conditional frequency distribution for the training corpus, we can use it to find the most likely word for any given context:

```
>>> cfdist['prediction'].max()
'problems'
>>> cfdist['problems'].max()
'in'
>>> cfdist['in'].max()
'the'
```

We can set up a simple loop to generate text, by using the most likely token for each word as the context for the next word:

```
>>> word = 'prediction'
>>> for i in range(15):
...     print word,
...     word = cfdist[word].max()
prediction problems in the frequency distribution of the
frequency distribution of the frequency distribution of
```

Note: This simple approach to text generation tends to get stuck in loops, as demonstrated by the text generated above. A more advanced approach would be to randomly choose each word, with more frequent words chosen more often.

4. Probability Distributions

This section is under construction. For information about probability distributions, see the reference documentation for the `ProbDistI` interface. The reference documentation includes pointers to all of the classes that currently implement the `ProbDistI` interface. Also, see the reference documentation for `ConditionalProbDist`, which is used to encode conditional probability distributions.

Index