Lab 3: Answers and explanations

Author: Sharon Goldwater
Date: 2014-09-01, updated 2017-09-30

Examining the initial output

`str_list` is a list containing a sequence of random outcomes from the distribution we defined.

`str_counts` is a dictionary containing the counts of each outcome in that sequence.

`str_probs` is a dictionary which is supposed to contain the estimated probability of each outcome, but currently also just contains the counts.

Normalizing a distribution

`normalize_counts` is supposed to normalize a set of counts or weights to create a probability distribution that sums to one, by dividing each count by the sum of all counts. At the moment, it does nothing, just returning the same counts that are passed in.

See our answer code[2] for two possible ways to write the correct function, one with a list comprehension and one with a for loop.

You could implement an error check on this function to make sure the sum of the values in the dictionary is 1, in the same way we did on the true distribution (lines 169-171 in our answer code). You could go further and make sure each value is between zero and one, although this is probably better implemented as a check on the *input* of the function, to make sure someone doesn't pass in crazy values.

Comparing estimated and true probabilities

The type of estimate produced here is called a relative frequency estimate or maximum-likelihood estimate. In general, items that have low probability will be estimated incorrectly because if they happen to appear in the observed data, their probability is usually overestimated; whereas if they happen not to appear their probability is underestimated.

Effects of sample size

The estimated probabilities should be more accurate as the amount of data increases. For the particular distribution we used here, you will be able to set the sample size large enough so that no zero probabilities are estimated (with, say, 500 or certainly 1000 samples). We could likely do the same with natural language if our distribution was over *characters*, as here, because there is only a finite number of characters. However, it isn't possible to do that with

[1] http://creativecommons.org/licenses/by-nc/4.0/.

[2] lab3-sol.py

a distribution over *words* in natural language because there is an infinitely long tail of words with arbitrarily low probabilities, so no matter how much data we have, there will be some words that are theoretically possible but so improbable that they don't appear in the data. (This is even more true if we consider higher-order n-grams over words.)

Computing the likelihood

To see two different implementations of the likelihood function, see our answer code[2].

The exact likelihoods you get will depend on the particular random sequence you generated, but you should find that (for a sequence of 50 characters) the likelihood of the true distribution ranges roughly from 1e-30 to 1e-42, and the likelihood of the estimated distribution ranges roughly from 1e-28 to 1e-39. However, although you can get sequences that vary in probability by 10-12 orders of magnitude, you will *always* find that the probability under the true distribution is *lower* than the probability under the estimated distribution, usually by about three orders of magnitude. This is precisely because the estimated distribution is the maximum-likelihood estimate, i.e., it is the estimate with the highest possible likelihood.

Log likelihood

Note Some students got non-zero answers for 500-character sequences; this may be because the numerical precision of the new lab computers is better than the ones last year when we wrote the lab. But if you increase the length of the sequence enough, you will still get a 0 and the rest of following answer will apply.

The likelihood printed out for a sequence of 500 characters is 0 because the actual probability is so small that the computer is unable to represent its value. (This problem is called `numerical underflow` and is something we need to watch out for when multiplying together many very small probabilities.)

We can't fix the problem by taking the log of the output of the likelihood function, because the output is already zero and taking the log of zero is undefined.

To see how the log likelihood should actually be computed, see our answer code[2].

The log likelihood of a random sequence of 500 characters will again vary, but you should be seeing numbers roughly in the range of -320 to -350.

Introduction to NumPy (optional)

Most answers should be self-explanatory.

The following will give errors:

```
l+1
a+c
l[m]
```

These two treat the `b` (or `c`) array as a set of indices, and return the items in `a` corresponding to those indices:

```
a[b]
a[c]
```

This does the same thing, by implicitly turning `m` into an array:

```
a[m]
```

This one creates a new array, where the item at index `i` is `sum(a[:i+1])`:

```
np.cumsum(a)
```

And `digitize(x,y)` assumes `y` is a set of "bins", and tells you which bin (by index `j`) each `x[i]` falls into, i.e., where `y[j-1] <= ``x[i]` < `y[j]`.