MMCT Python Package Documentation

Version 1.1.2

Chris Walther Andersen

April 7, 2022

Contents

List of Code Listings

1 Introduction

The purpose of this software is to perform a statistical test on a dataset, in order to determine whether the data has been generated from a multinomial distribution. The name of the package, `mmct`, is an abbreviation of "Multinomial Monte Carlo Test". As the name suggests, the statistical test is performed using a Monte Carlo simulation. This document describes how this is done and how to interact with the code to perform a test.

The package is inspired by two other software packages, both of which do the same job: The Python-package met[1] is also a Python-package for performing multinomial tests, however `met` performs an *exact* test. While this strategy is certainly preferred to the Monte Carlo version (which will always be an approximate test), an exact test quickly becomes infeasible as the problem size grows. The other package to have inspired this project is the XNomial[2] package for the R-programming language. `mmct` is basically an XNomial-clone for Python.

2 Mathematical background

Assume we have done an experiment in which N observations of a system that can end up in one of k different states has been made (e.g. rolling a D6 20 times: $N = 20$, $k = 6$). We have tracked the number of times each state occurred and denote them $m_1, m_2, \ldots, m_k$, such that $\sum_{i=1}^{k} m_i = N$.

We want to test the hypothesis that this result is compatible with a multinomial distribution with parameters N and $p_1, p_2, \ldots, p_k$, where p_i is the probability of the state i occurring and $\sum_{i=1}^{k} p_i = 1$. The result of this test should be a p-value for the null hypothesis that $m_1, \ldots, m_k$ are drawn from this multinomial distribution.

2.1 Monte Carlo sampling

The test is performed in two steps: Monte Carlo sampling a bunch of multinomially distributed sets of data, and afterwards determining the likelihood of the original dataset occuring based on the sampled data.

First we write the original data under test as

$$X = \{m_1, m_2, \ldots, m_k\}, \tag{1}$$

[1] `https://pypi.org/project/met/`

[2] `https://cran.r-project.org/web/packages/XNomial/vignettes/XNomial.html`

where m_i is the number of observations in bin i and again $m_1+m_2+\ldots+m_k = N$.

We start the test by sampling from the hypothesised multinomial distribution N times. We will end up with a number of observations in each of the k bins, each drawn with probability p_i, so we end up with

$$X^1 = \left\{m_1^1, m_2^1, \ldots, m_k^1\right\}, \tag{2}$$

such that $m_1^1 + m_2^1 + \ldots + m_k^1 = N$. We save this result, and then start over by sampling another N times into a new result:

$$X^2 = \left\{m_1^2, m_2^2, \ldots, m_k^2\right\}. \tag{3}$$

We continue this process N_S times, ending up with N_S random result vectors $X^j, 1 \leq j \leq N_S$, each generated from the same, hypothesised multinomial distribution. If N_S is large enough, we can now estimate the likelihood that X (the data under test) is drawn from the same distribution by comparing it to the randomly generated samples.

2.2 Sample grading

To do this we need a way of grading our samples to determine what is likely and what is unlikely. The `mmct` package implements two different ways of doing this:

2.2.1 Multinomial probability function

The first and most obvious way would be to grade each sample by its raw probability of occurring according to the probability distribution. This probability is calculates as

$$\begin{aligned} P^j &= \Pr\left(m_1^j, \ldots, m_k^j | N, p_1, \ldots, p_k\right) \\ &= \frac{N!}{m_1^j! \cdots m_k^j!} p_1^{m_1^j} \cdots p_k^{m_k^j}. \end{aligned} \tag{4}$$

Calculating P for all samples and our data under test in this way, we can now rank all the samples from the most likely (largest P) to most unlikely (smallest P) and look where P, the probability of the data under test, fits in. The p-value of our test is simply the number of samples with a probability P^j smaller than P divided by the total number of samples N_S:

$$\text{p-value} = \frac{N_{\text{smaller}}}{N_S} \tag{5}$$

2.2.2 Likelihood ratio

The second way uses likelihood-ratios. For any given Monte Carlo sample X^j we can calculate the likelihood-function $\mathcal{L}$ evaluated as the likelihood of the hypothesised probabilities given the sample X^j, which is simply equation (4). In the spirit of the likelihood-ratio test[3] we can compare this to an alternate hypothesis. The optimal alternate hypothesis is just to use the probabilities observed in the original data X^0, i.e.

$$\begin{aligned}\hat{p}_1 &= \frac{m_1^0}{N}\\ \hat{p}_2 &= \frac{m_2^0}{N}\\ &\cdots\\ \hat{p}_k &= \frac{m_k^0}{N}.\end{aligned}$$

The likelihood function of these probabilities, given the Monte Carlo sample X^j, is

$$\frac{N!}{m_1^j! \cdots m_k^j!}\hat{p}_1^{m_1^j} \cdots \hat{p}_k^{m_k^j} \tag{6}$$

3 Including code samples

Code can be printed using the minted or listings packages, or several other tools. Listing 1 shows an example of typesetting code from an external file.

Alternatively, code can be written directly in your .tex files, as in Listing 2. It's also possible to typeset syntax-highlighted code inline with a number of code listing packages. Check the documentation for the package you're using for more details. For example, section 3.3 of the Minted package documentation[4] gives some examples and guidelines, as does our help article on code highlighting with minted[5]. Forums such as TeX StackExchange[6] and LaTeX Community[7] are also a great source of tips.

[3] https://en.wikipedia.org/wiki/Likelihood-ratio_test
[4] http://texdoc.net/pkg/minted
[5] https://www.overleaf.com/learn/latex/Code_Highlighting_with_minted
[6] https://tex.stackexchange.com/
[7] https://latex.org/forum/

```matlab
function X = BitXorMatrix(A,B)
%function to compute the sum without charge of two vectors

        %convert elements into usigned integers
        A = uint8(A);
        B = uint8(B);

        m1 = length(A);
        m2 = length(B);
        X = uint8(zeros(m1, m2));
        for n1=1:m1
                for n2=1:m2
                        X(n1, n2) = bitxor(A(n1), B(n2));
                end
        end
```

Listing 1: Example from external file

```python
print("Hello World")
```

Listing 2: Example Python code