`RatingsModel` Mathematical Appendix and API Guide

Matthew Kulec

Version 1.1: February 7, 2022

`exact_test`

Let k be the number of distinct categories, $\mathbf{y} = (y_1, y_2, \ldots, y_k)$ be the observed counts for each of the categories, $n = \sum_{i=1}^{k} y_i$ be the observed total number of responses, and $\boldsymbol{\theta} = \left(\frac{y_1}{n}, \frac{y_2}{n}, \ldots, \frac{y_k}{n}\right)$ be the observed proportions for each of the categories. Denote the index with the greatest count as λ, so that $\mathbf{y}[\lambda] = y_\lambda = \max_{1 \le i \le k} y_i$.

An individual rating is modeled as a independent one-hot encoded vector of length k, with 1 being in only one of the k entries and 0 in the remaining $k-1$ entries. For instance, the probability of an individual selecting category l, or assigning a 1 in the lth position (0 everywhere else) in the one-hot encoding is just $\frac{y_l}{n}$. Summing n of these "Bernoulli random vectors" element-wise leads to a random vector that is distributed as $\text{Multinomial}(\boldsymbol{\theta}; n) = \text{Multinomial}(\frac{y_1}{n}, \frac{y_2}{n}, \ldots, \frac{y_k}{n}; n)$. This is the multivariate version of the binomial distribution, whose random variables are sums of

Bernoulli random variables.

Let $\mathbf{X} = (X_1, X_2, \ldots, X_k) \sim \text{Multinomial}(\boldsymbol{\theta}; n) = P(\mathbf{X}|\boldsymbol{\theta}, n)$ be a simulated ratings distribution (random vector), similar to that observed. This is referred to as a "hypothetical event where individual ratings are assigned and aggregated" in the README.md text. The exact p-value, as calculated by the `exact_test` method of the `RatingsModel` class, is then given by the following:

$$1 - \Pr(X_\lambda > \max_{1 \le i \le k, i \ne \lambda} X_i) = 1 - \sum_{x_1=0}^{n} \sum_{x_2=0}^{n-x_1} \sum_{x_3=0}^{n-x_1-x_2} \cdots \sum_{x_{k-1}=0}^{n-(x_1+x_2+\cdots+x_{k-2})} \underbrace{\frac{n!}{x_1! x_2! x_3! \cdots x_k!} \prod_{i=1}^{k} \theta_i^{x_i}}_{P(\mathbf{X}=\mathbf{x}|\boldsymbol{\theta},n)} \cdot \mathbb{1}(x_\lambda > \max_{1 \le i \le k, i \ne \lambda} x_i) \tag{1}$$

Where x_k is just a shorthand for $n - (x_1 + x_2 + \cdots + x_{k-2} + x_{k-1})$ and $\mathbb{1}$ is the indicator function, which evaluates to 1 if the input is true, and 0 if false. Informally speaking, the summation above is performed over the entire support of the multinomial distribution, i.e. size k integer partitions of n (with duplicates; not up to reordering), and discarding terms where x_λ is not greater than the rest of the x_i's.

`monte_carlo_test` with `sample_from_prop_prior = False` and `sample_from_count_prior = False`

From the above, $\Pr(X_\lambda > \max_{1 \le i \le k, i \ne \lambda} X_i)$ can be estimated using a monte-carlo simulation, which is accomplished by the `monte_carlo_test` method of the `RatingsModel` class, setting `sample_from_prop_prior = False` and `sample_from_count_prior = False`. The following describes the mathematical formalism behind the method.

Let S be the Bernoulli random variable:

$$S = \begin{cases} 1 & \text{if } X_\lambda > \max_{1 \le i \le k, i \ne \lambda} X_i \\ 0 & \text{else} \end{cases}$$

Then $\mathbb{E}[S] = \Pr(X_\lambda > \max_{1 \le i \le k, i \ne \lambda} X_i)$. By the Law of Large numbers, $\bar{s} \approx \mathbb{E}[S]$, almost surely in the limit. In the context of the API, we can therefore sample $\mathbf{X}$ `num_samples` many times, and compute the proportion of times $X_\lambda > \max_{1 \le i \le k, i \ne \lambda} X_i$ is true. This sample proportion is just $\bar{s}$, and can be obtained from the method by setting `details = True`. This will return a dictionary whose keys are `"p-value mean"` and `"p-value std"`. The value of the former is $\bar{s}$, while the value of the latter is its standard deviation, $\sqrt{\bar{s}(1-\bar{s})/n}$.

By the Central Limit Theorem, S is asymptotically normal:

$$S \sim \mathcal{N}\left(\bar{s}, \sqrt{\bar{s}(1-\bar{s})/n}\right)$$

A $(1-\alpha)100\%$ confidence interval can also be constructed to gauge the precision of the estimate:

$$\left(\bar{s} - z_{\alpha/2}\sqrt{\bar{s}(1-\bar{s})/n}, \bar{s} + z_{\alpha/2}\sqrt{\bar{s}(1-\bar{s})/n}\right)$$

where $z_{\alpha/2}$ denotes the $1-\alpha/2$ quantile of the standard normal. This confidence interval is obtained by setting `details = False` (the default), and changing the `confidence` parameter, which is 0.95 by default, and equals $1-\alpha$ in the above context.

It is pretty clear that as we increase `num_samples`, we get a better estimate.

Another thing worth pointing out is that if the samples are either all 0 (or all 1), then $\bar{s} = 0$ (or $= 1$), which results in $\text{std}(\bar{s}) = 0$, yeilding a degenerate confidence interval.

`monte_carlo_test` with `sample_from_prop_prior` `=` `True` and `sample_from_count_prior` `=` `False`

This setting of `monte_carlo_test` method imposes a Dirichlet$(\boldsymbol{\alpha}) = P(\boldsymbol{\theta}|\boldsymbol{\alpha})$ prior on $\boldsymbol{\theta}$ (now a random variable, not the sample proportion), where $\boldsymbol{\alpha} = (y_1 + 1, y_2 + 1, \ldots, y_k + 1)$. In a Bayesian context, this parameterization of the prior distribution incorporates information equivalent to a turnout of n, with y_j observations in the jth category, for $j = 1, 2, \ldots, k$. The underbraced term in equation (1) becomes:

$$P(\mathbf{X} = \mathbf{x}|n) = \int_\Omega P(\mathbf{X} = \mathbf{x}|\boldsymbol{\theta}, n) \cdot P(\boldsymbol{\theta}|\boldsymbol{\alpha}) d\boldsymbol{\theta}$$

where $\Omega = \{(\theta_1, \theta_2, \ldots, \theta_k) : \sum_{i=1}^k \theta_i = 1 \text{ and } \theta_i \geq 0 \; \forall i \in \{1, 2, 3, \ldots, k-1, k\}\}$, i.e. the $k-1$ simplex. Although deriving this distribution may be difficult, if not analytically intractable, sampling is actually easy. The procedure imitates the previous section, but the only change is that for each iteration, or every time $\mathbf{X}$ is sampled, $\boldsymbol{\theta}$ is sampled first, yeilding $\tilde{\boldsymbol{\theta}}$. Then, $\mathbf{X}$ is drawn from Multinomial$(\tilde{\boldsymbol{\theta}}; n)$. The process repeats anew the next time $\mathbf{X}$ is sampled.

`monte_carlo_test` with `sample_from_prop_prior = True` and `sample_from_count_prior = True`

This setting expects a `count_prior` on the total number of responses, written as a Python class object, and passed into `RatingsModel`. This could be the in-house class `RightGeometricCountPrior` that comes with the package, or your own.

Whatever the `count_prior` is, for purposes of this exposition, it will be denoted as $P(N|\cdots)$ with $\cdots$ representing any hyperparameters in the distribution. Further, denote the set containing all the possible values N could take as G, i.e. the support of N, with n being a realization of N (not the observed total). The underbraced term in equation (1) becomes:

$$P(\mathbf{X}=\mathbf{x}) = \sum_{n\in G}\int_{\Omega} P(\mathbf{X}=\mathbf{x}|\boldsymbol{\theta},n)\cdot P(\boldsymbol{\theta}|\boldsymbol{\alpha})\cdot P(N=n|\cdots)d\boldsymbol{\theta}$$

Once again, the `monte_carlo_test` programming interface carries over, with the only change being the random sampling. Every time $\mathbf{X}$ is sampled, $\boldsymbol{\theta}$ and N are sampled first, yeilding $\tilde{\boldsymbol{\theta}}$ and $\tilde{N}$, respectively. Then, $\mathbf{X}$ is drawn from Multinomial($\tilde{\boldsymbol{\theta}}$;$\tilde{N}$). The process repeats anew the next time $\mathbf{X}$ is sampled.

`monte_carlo_test` with `sample_from_prop_prior = False` and `sample_from_count_prior = True`

Let $\boldsymbol{\theta}$ represent the observed proportions, as it was defined originally. n is still a random realization of N, and not the observed total. This setting on `monte_carlo_test` also expects a `count_prior` to be passed into

`RatingsModel`. The underbraced term in equation (1) becomes:

$$P(\mathbf{X} = \mathbf{x}|\boldsymbol{\theta}) = \sum_{n \in G} P(\mathbf{X} = \mathbf{x}|\boldsymbol{\theta}, n) \cdot P(N = n| \cdots)$$

Again, the `monte_carlo_test` programming interface carries over, with the only change being the random sampling. Every time $\mathbf{X}$ is sampled, N is sampled first, yeilding $\tilde{N}$. Then, $\mathbf{X}$ is drawn from Multinomial($\boldsymbol{\theta}$;$\tilde{N}$). The process repeats anew the next time $\mathbf{X}$ is sampled.

RightGeometricCountPrior

Probability Mass Function

The probability mass function, which may be called via the `count_pmf` or `pmf` methods, if called from an instance of `RightGeometricCountPrior` (or just `count_pmf` if called from an instance of `RatingsModel`), is defined as:

$$P(N = n|p, m) = \frac{p^{m-n}}{\displaystyle\sum_{i=0}^{m} p^{m-i}} = \frac{p^{m-n}}{\displaystyle\sum_{i=0}^{m} p^{i}} = \left(\frac{1-p}{1-p^{m+1}}\right) p^{m-n}$$

where $0 < p < 1$ is the decay probability and the support is the set $\{0, 1, 2, \ldots, m-1, m\}$. Notice that the masses define a geometric sequence. p is referred to as the decay probability because the chance of sampling $m-1$ is p times the chance of sampling m, . . . the chance of sampling 0 is p^m times the chance of sampling m.

The `count_pmf` and `pmf` methods expect either an integer or a numpy array.

Generating Random Samples

Sampling from this distribution is achieved by calling the `count_rvs` or `rvs` methods, if called from an instance of `RightGeometricCountPrior` (or just `count_rvs` if called from an instance of `RatingsModel`). These expect an integer argument `size`.

Mathematically, these samples are generated by transforming a $U \sim \mathcal{U}(0,1)$ random variable:

$$\left\lceil -\frac{\log\left(\left(\frac{1}{p^{m+1}} - 1\right) U + 1\right)}{\log(p)} - 1 \right\rceil \sim \texttt{RightGeometricCountPrior}(m, p)$$

To derive the above transformation, note that a strictly increasing cumulative distribution function (CDF) evaluated at a discrete random variable is uniformly distributed. To prove this lemma, let T be a discrete random variable and F_T be the corresponding cumulative distribution function which is strictly increasing. We are interested in finding the distribution function of $Q = F_T(T)$:

$$F_Q(q) = \Pr(Q \leq q) = \Pr(F_T(T) \leq q) = \Pr(T \leq F_T^{-1}(q)) = F_T \circ F_T^{-1}(q) = q$$

$F_T^{-1}(q)$ is defined and $F_T \circ F_T^{-1}(q) = q$ follows since the map $F_T(\cdot)$ is strictly increasing, and thus bijective. Now let U be a random variable distributed as a discrete uniform distribution which assigns a mass of $\frac{1}{m+1}$ for each number in the set $\left\{\frac{0}{m+1}, \frac{1}{m+1}, \frac{2}{m+1}, \ldots, \frac{m-1}{m+1}, \frac{m}{m+1}\right\}$. Then the CDF of U evaluated at some point in the set $\frac{j}{m+1}$ is just:

$$F_U\left(\frac{j}{m+1}\right) = \sum_{z=0}^{j} \frac{1}{m+1} = \frac{j}{m+1}$$

Hence, Q and U have the same distribution function, or $F_T(T) \sim \mathcal{U}(0,1)$.

Returning to our derivation, the above implies the following:

$$F_T(T) = \left(\frac{1-p}{1-p^{m+1}}\right) \sum_{i=0}^{T} p^{m-i} = U$$

The remaining steps are purely algebraic:

$$\left(\frac{1-p}{1-p^{m+1}}\right) p^m \left(1 + \left(\frac{1}{p}\right) + \left(\frac{1}{p}\right)^2 + \cdots + \left(\frac{1}{p}\right)^T\right) = U$$

$$\left(\frac{1-p}{1-p^{m+1}}\right) p^m \frac{1-\left(\frac{1}{p}\right)^{T+1}}{1-\frac{1}{p}} = U$$

$$\frac{(1-p)}{1-p^{m+1}} p^{m+1} \frac{1-\left(\frac{1}{p}\right)^{T+1}}{-(1-p)} = U$$

$$\frac{p^{m+1}}{p^{T+1}} \left(\frac{1-p^{T+1}}{1-p^{m+1}}\right) = U$$

$$\frac{1}{1-p^{m+1}} \left[p^{m-T} - p^{m+1}\right] = U$$

$$T = \left\lceil \frac{\log\left(\frac{p^m}{U(1-p^{m+1}) + p^{m+1}}\right) + \log(p) - \log(p)}{\log(p)} \right\rceil$$

$$T = \left\lceil \frac{\log\left(\frac{p^{m+1}}{U(1-p^{m+1}) + p^{m+1}}\right)}{\log(p)} - 1 \right\rceil$$

$$T = \left\lceil \frac{\log\left(\frac{p^{m+1}}{U + p^{m+1}(1-U)}\right)}{\log(p)} - 1 \right\rceil$$

$$T = \left\lceil \frac{\log\left(\frac{p^{m+1}}{U + p^{m+1}U}\right)}{\log(p)} - 1 \right\rceil$$

$$T = \left\lceil -\frac{\log\left(U(1 + p^{-m-1})\right)}{\log(p)} - 1 \right\rceil$$

$$T = \left\lceil -\frac{\log\left(\left(\frac{1}{p^{m+1}} - 1\right)U + 1\right)}{\log(p)} - 1 \right\rceil$$

$$\blacksquare$$

Some steps above used properties of logarithms along with the fact that $1 - U \sim \mathcal{U}(0,1)$. Therefore, substituting U for $1 - U$ and vice versa is appropriate.

Parameter Estimation From a Confidence Interval

Parameters p and m can be estimated given a $(1-\alpha)100\%$ confidence interval $[l, m]$. This is done by using the `from_interval` class method on `RightGeometricCountPrior`, or during the instantiation of `RatingsModel` with `RightGeometricCountPrior` chosen as the `count_prior`. The argument `concentration` represents $(1-\alpha)$ and is between 0 and 1, default 0.95. Arguments `left_endpoint` and `right_endpoint` represent l and m, respectively.

Lastly, argument `maxiter` represents the number of iterations to run for Newton's method, or the bisection method, which is used as a fallback, with a default setting of 100.

We first find the polynomial that is used by the above root-finding algorithms. We wish to have a probability mass of $1-\alpha$ under the interval $[l,m]$:

$$1-\alpha = \left(\frac{1-p}{1-p^{m+1}}\right)\sum_{i=l}^{m} p^{m-i} = \left(\frac{1-p}{1-p^{m+1}}\right)\left(\frac{1-p^{m-l+1}}{1-p}\right) = \frac{1-p^{m-l+1}}{1-p^{m+1}}$$

$$f(p) = p^{m-l+1} - (1-\alpha)p^{m+1} - \alpha = 0$$

Besides 1, there is another root between 0 and 1. We look for a starting point where $f'(p)$ changes sign. To the right of this starting point, $f'(p) < 0$ and the tangent line updates decend closer and closer to 1. On the left side of the starting point, $f'(p) > 0$, or the slope is negative from right to left, and it will converge to the root. This starting point is:

$$f'(p) = (m-l+1)p^{m-l} - (m+1)(1-\alpha)p^{m} = 0$$

$$p^{*} = \sqrt[l]{\frac{m-l+1}{(1-\alpha)(m+1)}}$$

We can speed up convergence by selecting a point closer to the root and a bit to the left of p^{*}, call it p^{**}, where the slope is increasing left to right:

$$f''(p) = (m-l+1)(m-l)p^{m-l+1} - (1-\alpha)(m+1)mp^{m-1} = 0$$

$$p^{**} = \sqrt[l]{\frac{(m-l+1)(m-l)}{(1-\alpha)(m+1)m}}$$

This is the actual starting point used in the code.