Learning the Multinomial Distribution with Python

Authored by

Mohammed looti

November 1, 2025

The **Multinomial Distribution** stands as a cornerstone concept within **probability theory**, providing a crucial generalization of the simpler, yet widely used, **Binomial Distribution**. While the binomial model is strictly confined to scenarios involving only two possible, mutually exclusive outcomes--traditionally labeled as "success" or "failure"--the multinomial distribution extends this framework to accommodate any fixed number, k, of distinct and independent outcomes.

This powerful statistical tool is specifically designed to calculate the exact probability of observing a particular combination of counts for each of the k outcomes across a predetermined number of independent trials. A fundamental requirement for its application is that the probability associated with each specific outcome must remain constant throughout all trials. This makes the multinomial distribution indispensable for modeling complex real-world phenomena where observations naturally fall into more than two predefined categories. Examples span a vast range of disciplines, including genetic research (modeling allele frequencies), political science (analyzing voting patterns among multiple candidates), quality control (categorizing defects), and advanced machine learning techniques (classification error analysis).

Decoding the Mathematical Formula

When a **random variable**, denoted X, is defined to follow a multinomial distribution, our goal is typically to calculate the precise likelihood of achieving a specific set of counts ($x_1, x_2, ldots, x_k$) given a total of n trials. This calculation relies entirely on knowing the fixed probabilities ($p_1, p_2, ldots, p_k$) corresponding to each of the k possible outcomes. It is mathematically essential that the sum of these individual outcome probabilities always equates exactly to one ($sum_{i=1}^{k} p_i = 1$).

The formal mathematical expression, which yields the Probability Mass Function (PMF) for this distribution, elegantly combines principles of counting and probability. The formula is structured around two main components: the multinomial coefficient, derived from **combinatorics**, which accounts for the number of ways the specific counts can occur; and the product of the probabilities, which accounts for the inherent likelihood of that specific sequence of outcomes.

The probability $P(X = (x_1, ldots, x_k))$ is formally expressed as:

$$P(X) = frac{n!}{x_{1}! x_{2}! cdots x_{k}!} cdot p_{1}^{x_{1}} p_{2}^{x_{2}} cdots p_{k}^{x_{k}}$$

A clear understanding of the parameters within this equation is crucial for its correct application in statistical modeling:

n: Represents the **total number of independent trials** or events conducted in the experiment.
xi: Denotes the **specific count** of times outcome i occurs. Critically, the sum of all x_i must

perfectly match the total number of trials, n.
pi: Indicates the fixed, underlying **probability** that outcome i occurs during any single trial.
k: Defines the total number of **distinct and mutually exclusive outcomes** available within the sample space of the experiment.

While the manual derivation of this formula can be complex and computationally taxing, especially for large n, modern data science practice relies heavily on optimized statistical libraries. Libraries such as **SciPy** in the Python ecosystem provide highly efficient functions that automate the calculation of the multinomial probability mass function, allowing analysts to focus on model interpretation rather than arithmetic complexity.

Leveraging SciPy for Multinomial Calculations

To transition the theoretical framework of the multinomial distribution into practical data analysis using Python, we utilize the powerful tools available in the `scipy.stats` module. This module provides specialized functions necessary for statistical computations, most notably for calculating the **Probability Mass Function** (PMF). The PMF is essential here because the multinomial distribution deals with discrete, specific counts, giving us the exact probability for a defined set of outcomes.

The core function we employ is `multinomial.pmf()`. This function efficiently handles the complex factorial and exponentiation calculations defined by the mathematical formula, returning the desired probability. Before executing this function, the user must correctly structure three primary input parameters, ensuring they align with the definition of the multinomial experiment.

The required input parameters for the `multinomial.pmf()` function are structured as follows:

The `x` array: This list or array encapsulates the specific outcome counts ($x_1, x_2, ldots, x_k$) whose probability is being sought. It represents the specific realized result of the experiment.
The `n` parameter: This integer represents the total number of independent trials performed. Crucially, this value must exactly match the sum of all elements within the `x` array ($sum x_i = n$).
The `p` array: This list or array defines the fixed probabilities ($p_1, p_2, ldots, p_k$) associated with each of the k distinct outcomes. As noted previously, the sum of these probabilities must equal 1.0.

The following case studies demonstrate the utility of `multinomial.pmf()` across various real-world scenarios, illustrating how these parameters are defined and applied in practice.

Case Study 1: Analyzing Categorical Election Data

A classic application of the multinomial distribution involves modeling voter preferences in an

election featuring multiple candidates. Let us analyze a local mayoral election where three candidates--Candidate A, Candidate B, and Candidate C--are competing. Based on extensive historical data and pre-election polling, we establish the following fixed, underlying probabilities of voter preference:

Candidate A (p_1): 10% (0.10)
Candidate B (p_2): 40% (0.40)
Candidate C (p_3): 50% (0.50)

We decide to conduct a small, random sample survey of 10 registered voters ($n=10$). The specific question is: What is the exact probability that we observe a breakdown where exactly 2 voters supported Candidate A, 4 supported Candidate B, and 4 supported Candidate C? This situation perfectly aligns with the requirements of the multinomial distribution, featuring a fixed number of independent trials (voters) and three mutually exclusive outcomes (A, B, or C).

To solve this using Python, we define our inputs: the total number of trials ($n=10$), the desired outcome counts ($x=$), and the pre-established outcome probabilities ($p=$). We then leverage the SciPy library to calculate the **multinomial** probability.

from scipy.stats import multinomial

Define the parameters: x (counts), n (total trials), p (probabilities)
counts =
n_trials = 10
probabilities =

# Calculate the multinomial probability using the PMF
multinomial.pmf(x=counts, n=n_trials, p=probabilities)

0.05040000000000001

The result of the calculation, approximately **0.0504** (or just over 5%), demonstrates the specific nature of this distribution. Obtaining this exact combination of counts in a small sample is relatively unlikely. This outcome highlights a key feature of the multinomial PMF: it provides the probability for one highly specific point in the sample space, rather than a cumulative range of outcomes.

Case Study 2: Modeling Sampling with Replacement (The Urn Problem)

The classical "urn problem" in probability provides an excellent conceptual model for understanding sampling with replacement, a process perfectly suited for the multinomial distribution. Imagine an urn containing 10 marbles categorized by color, where the colors and their

respective probabilities are fixed:

Yellow Marbles: 6 (Probability $p_1 = 0.6$)
Red Marbles: 2 (Probability $p_2 = 0.2$)
Pink Marbles: 2 (Probability $p_3 = 0.2$)

If we conduct four independent trials ($n=4$), where a marble is selected, its color noted, and it is immediately replaced, the probability distribution remains constant across all trials. We are interested in the probability of a highly specific outcome: that all four selections result in a yellow marble. This requires setting the counts for the other colors to zero.

For this scenario, the desired counts vector `x` must reflect 4 yellow marbles and 0 of the other two colors, yielding $x = $. The total number of trials n is 4, and the probabilities p are constant at $$. The following Python implementation calculates this probability directly:

from scipy.stats import multinomial

Define the parameters: x (counts), n (total trials), p (probabilities)
counts =
n_trials = 4
probabilities =

# Calculate the multinomial probability using the PMF
multinomial.pmf(x=counts, n=n_trials, p=probabilities)

0.1295999999999999

The calculated probability of drawing four yellow marbles consecutively, with replacement, is approximately 0.1296. This example is instructive as it shows how the multinomial function seamlessly handles scenarios where several of the desired counts (x_i) are zero, provided the fundamental requirement that the sum of all counts equals the total number of trials (n) is met.

Case Study 3: Modeling Outcomes in Competitive Games

The multinomial distribution proves highly valuable for modeling competitive environments or sports results where the possible outcomes extend beyond a simple win/loss dichotomy to include events like a tie or draw. Consider a series of 10 competitive chess games between two students, Student A and Student B. Based on their historical performance, the probabilities for a single game are fixed and known:

Student A wins (p_1): 0.5
Student B wins (p_2): 0.3

The game results in a tie (p_3): 0.2

If these 10 games ($n=10$) are independent trials, we aim to find the probability of observing a very specific final scoreline: 4 wins for A, 5 wins for B, and 1 game resulting in a tie. This requires applying the **Probability Mass Function** to this set of specific counts, utilizing the total trials and the inherent game probabilities.

We structure our inputs accordingly: the total number of games ($n=10$), the specific outcome counts ($x=$), and the inherent probabilities ($p=$). The resulting calculation provides the likelihood of this exact sequence occurring under these defined conditions.

from scipy.stats import multinomial

Define the parameters: x (counts), n (total trials), p (probabilities)
counts =
n_trials = 10
probabilities =

# Calculate the multinomial probability
multinomial.pmf(x=counts, n=n_trials, p=probabilities)

0.03827249999999997

The resulting probability, approximately **0.0383**, confirms that achieving this precise score distribution--4 wins for A, 5 wins for B, and exactly one tie--is a statistically rare event given the initial assumptions about player skill. This demonstrates how the multinomial function can quantify the likelihood of complex, multi-state outcomes.

Summary and Best Practices for Implementation

The **Multinomial Distribution** provides a powerful and flexible statistical framework for accurately calculating the probabilities of specific counts in experiments involving multiple discrete outcomes ($k > 2$). By mastering the use of the `multinomial.pmf()` function available within the **SciPy** library, data scientists can efficiently model complex categorical data scenarios, bypassing the need for manual, tedious calculation of the full combinatorial probability formula.

Successful and accurate implementation of the multinomial distribution hinges on strict adherence to its underlying assumptions. Before applying the distribution, analysts should confirm the following critical prerequisites:

The experimental trials must be **independent** of one another, meaning the outcome of one trial does not influence the outcome of the next.

The outcome probabilities ($p_1, p_2, ldots, p_k$) must **remain fixed** throughout the entire sequence of trials.
The sum of the desired outcome counts ($x_1 + x_2 + ldots + x_k$) must precisely equal the total number of trials (n).
The sum of the outcome probabilities must strictly equal 1.0 ($sum p_i = 1$).

By diligently adhering to these guidelines, the multinomial distribution becomes an indispensable tool in the analyst's toolkit, providing clarity and quantitative rigor to problems rooted in categorization and the modeling of multiple discrete outcomes across various scientific and commercial fields.

For those interested in exploring related concepts or diving deeper into the statistical foundations of this topic, further documentation and academic resources are readily available.