Balancing Data Sampling Using Weighted Probabilities

Introduction
When working with real-world datasets, the distribution of different categories is often unbalanced. Some categories may appear more frequently than others, making it difficult to train unbiased models or conduct fair analyses. One way to address this issue is through weighted sampling, where each data sample is assigned a probability of being selected based on category frequencies.

This report presents two methods for assigning sample weights and evaluates their effectiveness in achieving a balanced category distribution.

Problem Definition
Goals:
1. Assign Weights to Data Samples: Each sample contains multiple categories, and we need a method to compute an appropriate weight for it.
2. Perform Weighted Sampling: Use computed weights to sample data such that the final sampled dataset achieves a more uniform category distribution.

Challenges:
- The dataset is not uniformly distributed across categories.
- Some samples belong to multiple categories.
- The sampling process must allow both "with replacement" and "without replacement" options.

Weight Assignment Methods
We implemented two different methods to assign weights to samples:

1. Least-Frequency Weighting
- Each sample receives a weight equal to the inverse of the frequency

of the least occurring category in the sample.
- This ensures that samples containing rare categories have a higher probability of selection.

2. Sum-Inverse-Frequency Weighting
- Each sample's weight is the sum of the inverse frequencies of all categories it contains.
- Samples that include multiple rare categories receive even higher weights, further boosting their probability of being chosen.

Weighted Sampling Process
Once weights were assigned, we performed sampling using the following approach:
- Samples were selected based on computed probabilities.
- Sampling was conducted with replacement to allow repeated selection of the same sample.
- The number of selected samples was set to 50 (out of 100 total samples).

Evaluating Balance in Sampled Data
We used two statistical metrics to measure how well the sampling methods balanced category distribution:

1. Variance of Category Proportions
   - Measures how much the category proportions deviate from their mean.
   - Lower variance indicates a more balanced distribution.

2. KL Divergence from Uniform Distribution
   - Measures how different the sampled distribution is from an ideal uniform distribution.
   - Lower values indicate a more uniform spread of categories.

Results
| Metric | Original Data | Least-Frequency Sampling | Sum-Inverse-Frequency Sampling |

Variance	0.000462	0.000194	0.000157
KL Divergence	0.0227	0.0093	0.0077

Key Insights:
- Both sampling methods significantly reduced variance, making the dataset more balanced.
- Sum-Inverse-Frequency Sampling achieved the most uniform category distribution, as shown by the lowest KL divergence value.

Conclusion & Recommendations
- Weighted sampling is a powerful technique to improve category balance in datasets.
- Sum-Inverse-Frequency Sampling provides the best balance and should be preferred in cases where uniformity is critical.
- These methods can be applied to real-world problems such as bias correction in AI models, survey data balancing, and fair sampling in medical studies.

Further Exploration
- Extend sampling methods to support additional constraints, such as stratification.
- Explore adaptive re-weighting strategies based on real-time data distribution.
- Test methods on larger datasets to assess scalability.

Code Implementation
Below is the Python implementation of the discussed methods:

```python
import numpy as np
from collections import Counter

def assign_weights_least_frequency(data, categories):
    num_samples = len(data)
    category_counts = Counter(cat for row in data for cat in row)
    weights = np.array([1 / min(category_counts[cat] for cat in row) for row in data])
    return weights / np.sum(weights)

def assign_weights_sum_inverse_frequency(data, categories):
    num_samples = len(data)
    category_counts = Counter(cat for row in data for cat in row)
    weights = np.array([sum(1 / category_counts[cat] for cat in row) for row in data])
    return weights / np.sum(weights)

def weighted_sampling(data, weights, sample_size, replacement):
    indices = np.random.choice(len(data), size=sample_size, replace=replacement, p=weights)
    return [data[i] for i in indices]
```