Machine Learning

| Hyperparameter Sampling Distribution

Prof. Dr. Mostafa Elhosseini

https://youtube.com/@ProfElhosseiniTechVerse

Agenda

- Hyperparameter Sampling Distribution
- Continuous Hyperparameters
- Discrete Hyperparameters
- Categorical Hyperparameters

- How to choose a suitable distribution?
  - Prior Knowledge and Expertise
  - Adaptive Sampling Strategies
  - Practical Considerations
  - scipy.stats module
  - Uniform Distribution
  - Geometric Distribution
  - exponential distribution
  - loguniform distribution
  - Logarithm of Log-Uniform Distribution

Hyperparameter optimization

- Hyperparameters are the **external configurations of the model** that are not learned from the data but must be **set prior** to the training process.
- These parameters significantly **influence the performance** of ML models, affecting their ability to learn from data and make accurate predictions

Hyperparameter optimization

- Choosing the appropriate sampling distribution significantly **impact the performance and efficiency** of machine learning models.
- The choice of distribution often depends on the **nature of the hyperparameter**, **prior knowledge** about the model, and the **specific goals of the optimization process**.

Understanding Hyperparameter Types

- **Continuous Hyperparameters**
  - Learning rate
  - Regularization coefficients.
  - Uniform distributions or
  - Log-uniform distributions

Understanding Hyperparameter Types

- **Continuous Hyperparameters**
  - Learning rate
  - Regularization coefficients.
  - **Uniform distributions** or
  - Log-uniform distributions

  - **Uniform Distribution**: If you assume that the hyperparameter is equally likely to be optimal across a range,

Understanding Hyperparameter Types

- **Continuous Hyperparameters**
  - Learning rate
  - Regularization coefficients.
  - Uniform distributions or
  - **Log-uniform distributions**

  - Log-uniform distributions: For hyperparameters where small changes can have a large impact on model performance and where values span several orders of magnitude (e.g., learning rate)

Understanding Hyperparameter Types

- Discrete Hyperparameters
  - Number of layers in a neural network
  - Number of neighbors in a k-NN algorithm.
  - Uniform discrete distribution

Understanding Hyperparameter Types

- Discrete Hyperparameters
  - Number of layers in a neural network
  - Number of neighbors in a k-NN algorithm.
  - **Uniform discrete distribution**

  - Uniform Distribution: Often, the best approach is to sample uniformly across options since there may be no a priori reason to favor one option over another.

Understanding Hyperparameter Types

- Categorical Hyperparameters
  - Kernel Type in SVM
  - Activation function in a neural network
  - **simple uniform sampling**

  - Uniform Distribution: Often, the best approach is to sample uniformly across options since there may be no a priori reason to favor one option over another.

Prior Knowledge and Expertise

- Leveraging domain knowledge and empirical evidence from similar problems or models can guide the choice of distribution. For instance, if previous experiments indicate that a **certain range or scale of values** is more promising, the distribution can be **centered** around those values.
- Analyzing the sensitivity of the model to different hyperparameters can also inform the choice of distribution. Parameters to which the model is **more sensitive** might require **finer-grained** search strategies.

Adaptive Sampling Strategies

- **Adaptive strategies**, such as those used in **Bayesian optimization**, **adjust the sampling distribution** based on the **results of previous** evaluations. By modeling the performance of the model as a function of hyperparameters, these methods can concentrate the search in more promising regions of the hyperparameter space.
- **Multi-fidelity approaches** can initially use **broader distributions** to identify promising regions quickly and then **refine the search** around those regions with more narrowly focused distributions.

Practical Considerations

- Broader and more explorative → More computational resources

- The choice of distribution may also depend on the optimization goals
  - maximizing accuracy vs. minimizing computational cost

scipy.stats module

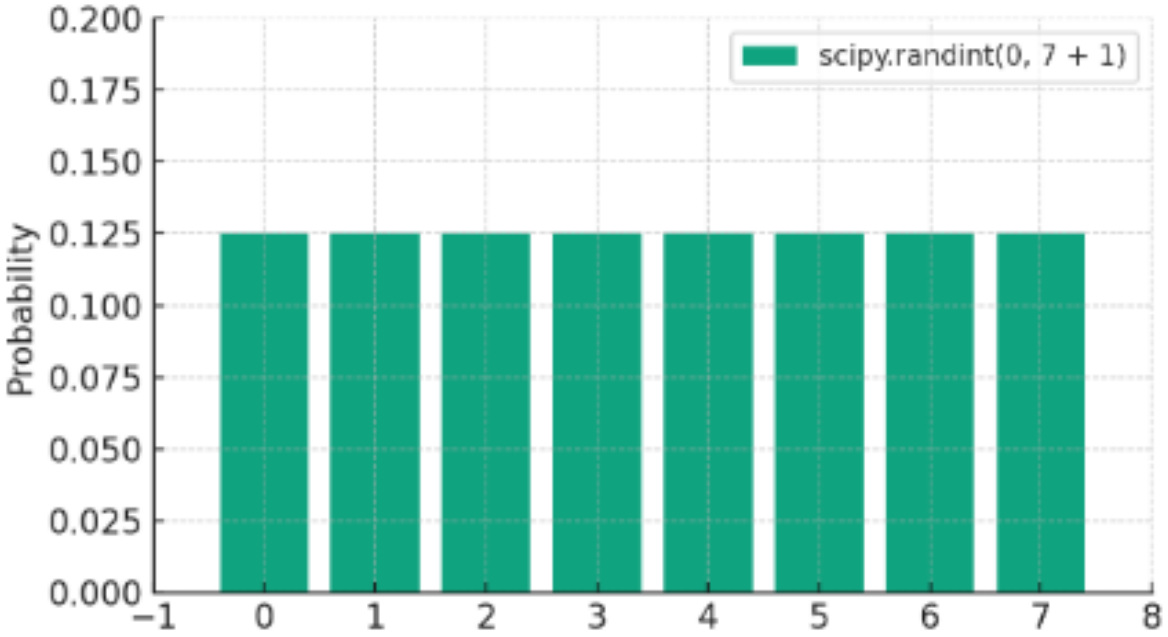
- The **scipy.stats module** in Python offers a **range of statistical distributions**

scipy.stats module

scipy.stats.randint(a, b+1)

- For discrete hyperparameters ranging from **a to b**, inclusive, where each value within that range is equally probable.
- Hyperparameters that have a clear, finite set of integer values (e.g., number of layers in a neural network, number of neighbors in k-NN).
- The uniform probability ensures that **each configuration is given equal consideration**, which is ideal when there's no prior reason to prefer one value over another.

scipy.stats module



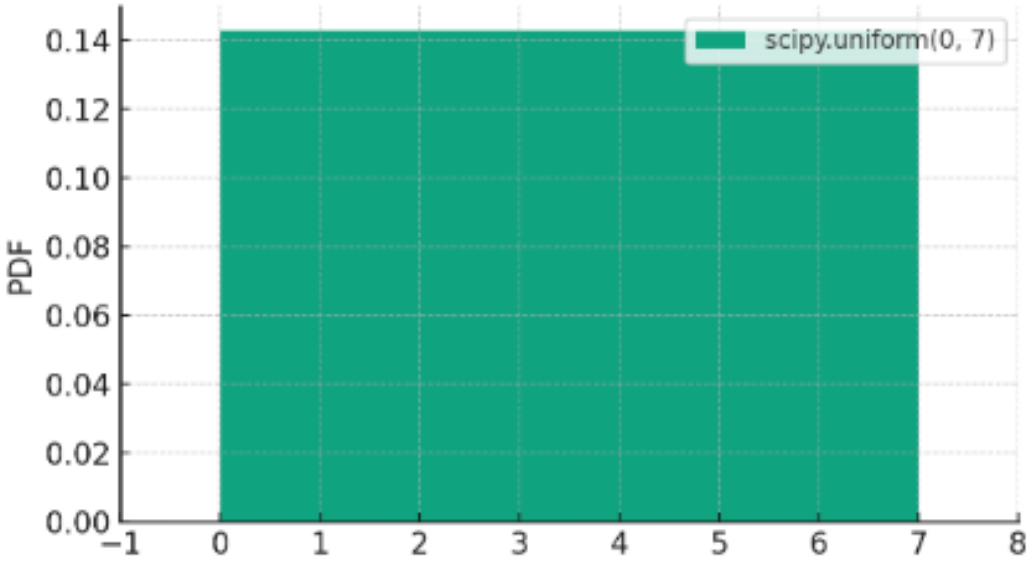
- randint(0, 7 + 1): The probability mass function (PMF) for a discrete uniform distribution, showing equal probabilities for integer values in the range from 0 to 7.

scipy.stats module

scipy.stats.uniform(a, b)

- For continuous hyperparameters, providing a uniform distribution over the interval [a,b).
- Similar to randint but for *continuous* parameters.
- you have no strong priors favoring certain values.
- It's a good default choice for initial explorations, especially when the impact of the hyperparameter on model performance is not well understood.

scipy.stats module



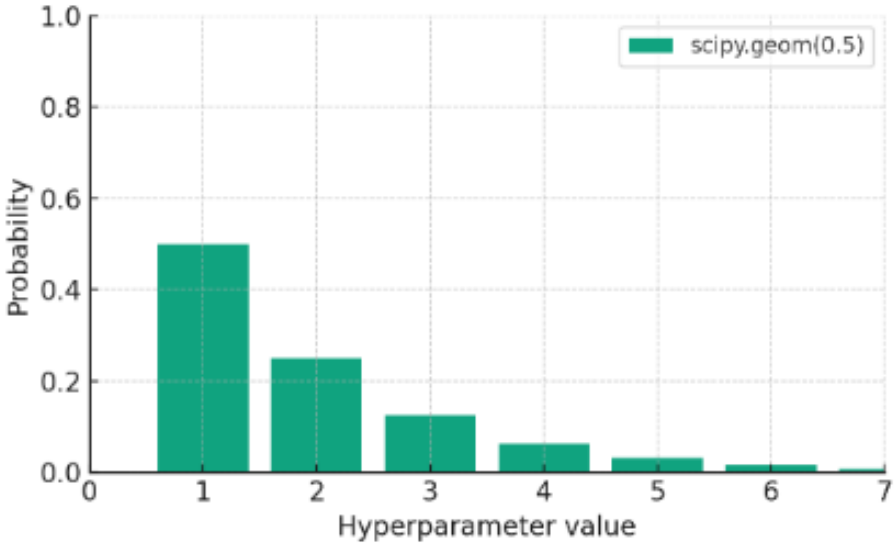
- uniform(0, 7): The probability density function (PDF) for a continuous uniform distribution, illustrating a constant probability density across the interval from 0 to 7.

scipy.stats module

scipy.stats.geom(1 / scale)

- For discrete hyperparameters, when you want to **prioritize values** around a certain scale but **still explore a wide range** of values.
- This can be useful for hyperparameters like the batch size or specific regularization coefficients, where you have a **rough idea** of the **optimal magnitude** but still want to **maintain flexibility**.

scipy.stats module



- geom(0.5): The PMF for a geometric distribution, where the probability of each value decreases exponentially, indicating a preference for lower values.

scipy.stats module

scipy.stats.geom(1 / 1000):

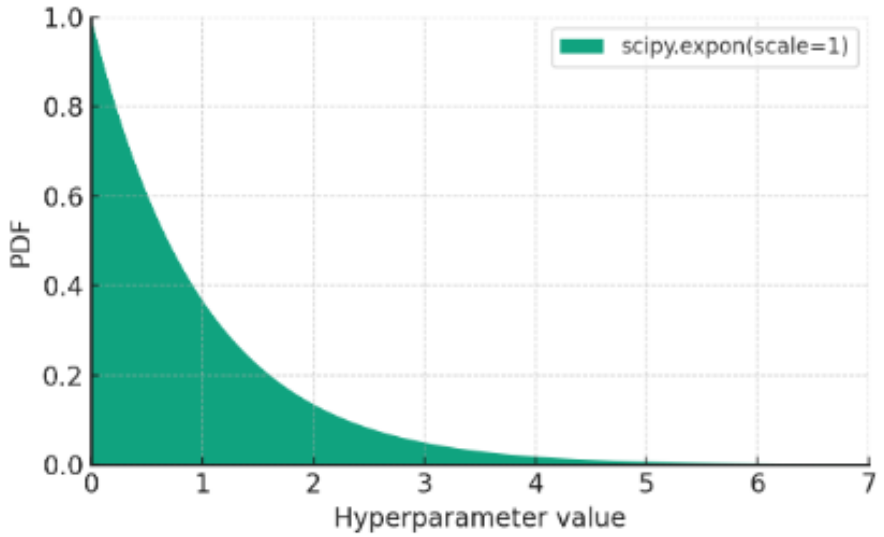
- In this case, we're sampling discrete values with a scale of around 1000, meaning most values will be close to 1000, but some smaller and larger values will also be sampled.
- The samples [1145, 687, 2318, 42, 1029] reflect this behavior. Most values are around 1000, but we also have an outlier value of 2318 and a small value of 42, which could be useful for exploring different scales.

scipy.stats module

scipy.stats.expon(scale)

- For continuous values - The continuous **counterpart** to the **geometric** distribution.
- Analysis: Ideal for hyperparameters where you expect that values around a certain point are more likely but wish to explore both lower and higher values with decreasing probability.
- This can be suitable for learning rates

scipy.stats module



- expon(scale=1): The PDF for an exponential distribution, starting high at zero and decreasing exponentially, highlighting a higher likelihood of lower values.

scipy.stats module

scipy.stats.expon(scale=0.5):

- For this exponential distribution, we specified a scale of 0.5, which means 0.5 is the most likely value, and the distribution decays exponentially as values move away from 0.5.
- The samples [0.21, 1.38, 0.09, 0.69, 0.47] demonstrate this behavior. Most values are close to 0.5, but some larger values like 1.38 are also sampled, albeit with lower probability.

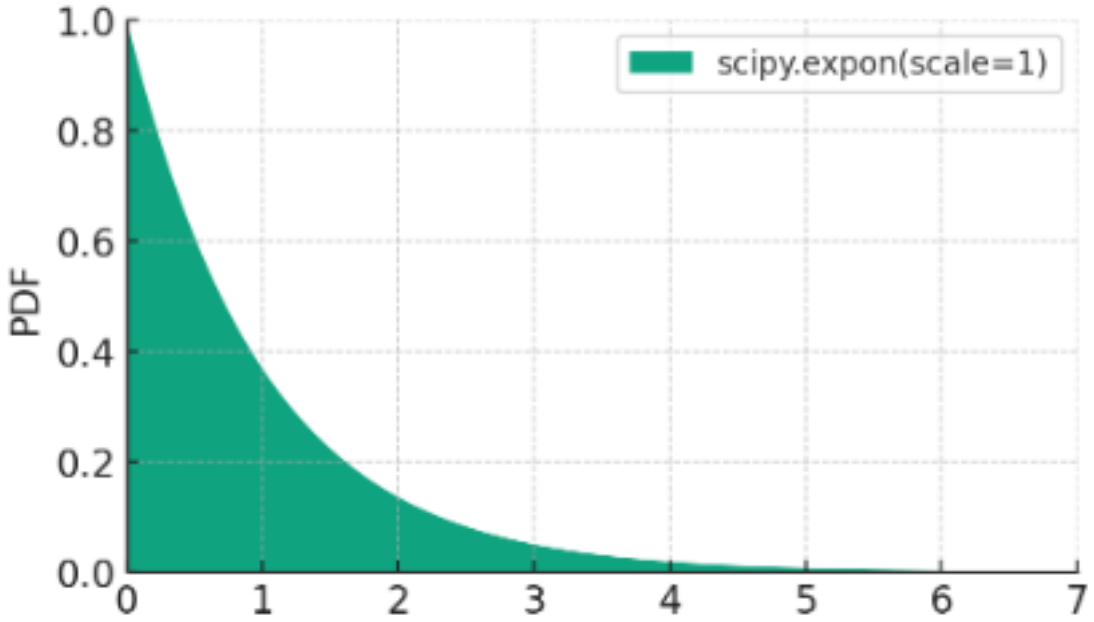
scipy.stats module

Exponential Distribution with scale=1:

- The exponential distribution with a scale parameter of 1 has a PDF that decreases exponentially as the value of the random variable increases.
- Since the PDF is highest at x=1 (with a value of 0.3678794411714423), values close to 1 are more likely to be generated compared to values farther away from 1.
- For example, a value of 0.5 would be more likely to be generated than a value of 2 or 3, as the PDF decays rapidly for larger values.
- Very small values close to 0 are also less likely to be generated, but their probability is higher than very large values.

scipy.stats module

Exponential Distribution with scale=1:



scipy.stats module

scipy.stats.loguniform(a, b)

- The optimal scale of the hyperparameter is largely unknown, offering an equal chance of sampling values across orders of magnitude.
- This distribution is crucial when dealing with hyperparameters that could potentially span several orders of magnitude, and there is significant uncertainty about where the optimal value lies.
- The loguniform distribution ensures that every order of magnitude within the specified range has an equal probability of being sampled, facilitating a broad and unbiased search across scales.

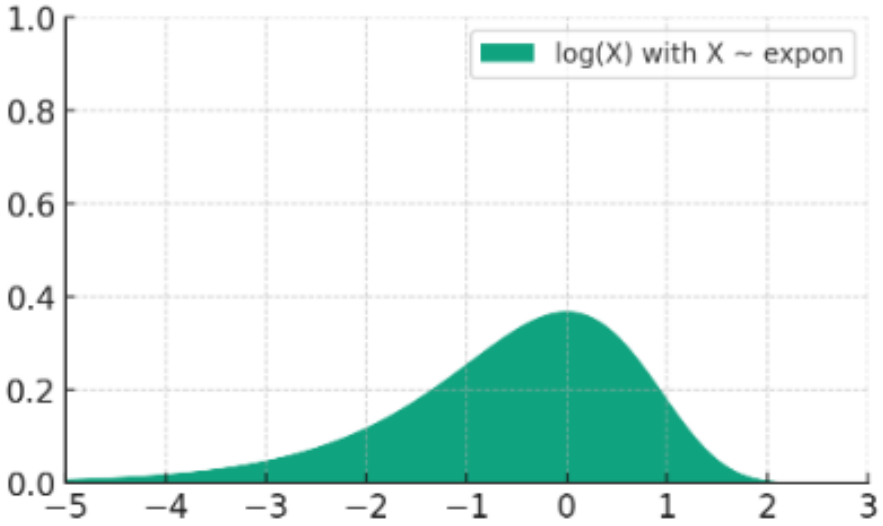
scipy.stats module

Logarithm of Exponential Distribution:

- The logarithmic transformation of the exponential distribution preserves the general shape of the distribution, but on a logarithmic scale.
- Since the PDF is highest at log(x)=0 (or x=1), values close to 1 on the original scale are more likely to be generated.
- However, due to the logarithmic transformation, values smaller than 1 (negative log values) and values larger than 1 (positive log values) have different likelihoods compared to the original exponential distribution.
- Small values close to 0 have a higher probability of being generated on the logarithmic scale, while very large values have a lower probability compared to the original exponential distribution.

scipy.stats module

Logarithm of Exponential Distribution:



scipy.stats module

Log-Uniform Distribution with a=0.001, b=1000:

- The log-uniform distribution has a constant PDF value of 0.0010005 within the specified range of 0.001 to 1000, indicating that all values within this range have an equal likelihood of being generated on a logarithmic scale.
- Since the distribution is uniform on a logarithmic scale, small values close to 0.001 and large values close to 1000 are equally likely to be generated.
- However, on the original linear scale, the probability of generating very small values (e.g., 0.001) or very large values (e.g., 1000) is lower compared to values in the middle of the range.

scipy.stats module

Log-Uniform Distribution with a=0.001, b=1000: