Lesson on Probability Distributions in NumPy

This lesson will guide you through working with four fundamental probability distributions using NumPy: Normal, Uniform, Binomial, and Poisson. We'll progress from basic concepts to advanced applications.

Prerequisites

* Basic Python knowledge
* NumPy installed (pip install numpy)

1. Introduction to Probability Distributions

Probability distributions describe how probabilities are distributed over the values of a random variable. NumPy's random module provides tools to work with these distributions.

python

Copy

Download

import numpy as np

import matplotlib.pyplot as plt

2. Uniform Distribution

Basic Concepts

* All outcomes in a range are equally likely
* Defined by minimum and maximum values

Beginner Level

python

Copy

Download

# Generate 1000 uniform random numbers between 0 and 1

uniform\_data = np.random.uniform(0, 1, 1000)

# Plotting

plt.hist(uniform\_data, bins=30, density=True)

plt.title('Uniform Distribution (0, 1)')

plt.show()

Intermediate Level

python

Copy

Download

# Generate uniform distribution with different parameters

uniform\_custom = np.random.uniform(low=-5, high=5, size=1000)

# Calculate statistics

mean = np.mean(uniform\_custom)

std = np.std(uniform\_custom)

print(f"Mean: {mean:.2f}, Standard Deviation: {std:.2f}")

# Probability density function (PDF) at a point

def uniform\_pdf(x, a, b):

return 1/(b-a) if a <= x <= b else 0

pdf\_value = uniform\_pdf(2, -5, 5)

print(f"PDF at x=2: {pdf\_value}")

Advanced Level

python

Copy

Download

# Inverse transform sampling for custom distributions

def inverse\_transform\_sampling(size=1000):

u = np.random.uniform(0, 1, size)

return np.sqrt(u) # Example transformation

samples = inverse\_transform\_sampling()

plt.hist(samples, bins=30, density=True)

plt.title('Transformed Uniform Distribution')

plt.show()

3. Normal (Gaussian) Distribution

Basic Concepts

* Bell-shaped curve
* Defined by mean (μ) and standard deviation (σ)

Beginner Level

python

Copy

Download

# Generate normal distribution

normal\_data = np.random.normal(loc=0, scale=1, size=1000)

# Plotting

plt.hist(normal\_data, bins=30, density=True)

plt.title('Standard Normal Distribution (μ=0, σ=1)')

plt.show()

Intermediate Level

python

Copy

Download

# Generate with different parameters

normal\_custom = np.random.normal(loc=5, scale=2, size=1000)

# Calculate probability between two values

from scipy.stats import norm

prob = norm.cdf(2, loc=5, scale=2) - norm.cdf(1, loc=5, scale=2)

print(f"Probability between 1 and 2: {prob:.4f}")

# Percentiles

percentile\_95 = np.percentile(normal\_custom, 95)

print(f"95th percentile: {percentile\_95:.2f}")

Advanced Level

python

Copy

Download

# Multivariate normal distribution

mean = [0, 0]

cov = [[1, 0.5], [0.5, 1]]

mv\_normal = np.random.multivariate\_normal(mean, cov, 1000)

# Plot

plt.scatter(mv\_normal[:, 0], mv\_normal[:, 1])

plt.title('Bivariate Normal Distribution')

plt.xlabel('X')

plt.ylabel('Y')

plt.show()

# Kernel Density Estimation

from scipy.stats import gaussian\_kde

kde = gaussian\_kde(normal\_data)

x = np.linspace(-4, 4, 100)

plt.plot(x, kde(x), label='KDE')

plt.hist(normal\_data, bins=30, density=True, alpha=0.5)

plt.title('Normal Distribution with KDE')

plt.legend()

plt.show()

4. Binomial Distribution

Basic Concepts

* Discrete distribution of number of successes in n trials
* Parameters: n (trials), p (success probability)

Beginner Level

python

Copy

Download

# Flip a fair coin 10 times, repeat experiment 1000 times

binomial\_data = np.random.binomial(n=10, p=0.5, size=1000)

# Plotting

plt.hist(binomial\_data, bins=range(12), density=True, align='left')

plt.title('Binomial Distribution (n=10, p=0.5)')

plt.xticks(range(11))

plt.show()

Intermediate Level

python

Copy

Download

# Different parameters

binomial\_custom = np.random.binomial(n=20, p=0.3, size=1000)

# Normal approximation (for large n)

mu = 20 \* 0.3

sigma = np.sqrt(20 \* 0.3 \* 0.7)

normal\_approx = np.random.normal(mu, sigma, 1000)

# Compare

plt.hist(binomial\_custom, bins=range(22), density=True, alpha=0.5, label='Binomial')

plt.hist(normal\_approx, bins=30, density=True, alpha=0.5, label='Normal Approx')

plt.title('Binomial vs Normal Approximation')

plt.legend()

plt.show()

Advanced Level

python

Copy

Download

# Bayesian inference with binomial distribution

def bayesian\_update(prior, data):

# Data is (successes, failures)

posterior = np.random.beta(prior[0] + data[0], prior[1] + data[1], 10000)

return posterior

prior = (1, 1) # Uniform prior

data = (8, 2) # 8 successes, 2 failures

posterior = bayesian\_update(prior, data)

plt.hist(posterior, bins=30, density=True)

plt.title('Posterior Distribution of Success Probability')

plt.xlabel('Probability')

plt.show()

5. Poisson Distribution

Basic Concepts

* Models rare events over time/space
* Parameter λ (average rate)

Beginner Level

python

Copy

Download

# Generate Poisson data (λ=3)

poisson\_data = np.random.poisson(lam=3, size=1000)

# Plotting

plt.hist(poisson\_data, bins=range(15), density=True, align='left')

plt.title('Poisson Distribution (λ=3)')

plt.xticks(range(14))

plt.show()

Intermediate Level

python

Copy

Download

# Relationship between Poisson and Binomial

# For large n and small p, Binomial ≈ Poisson

n = 1000

p = 0.003

lambda\_ = n \* p

binomial\_approx = np.random.binomial(n, p, 1000)

poisson\_exact = np.random.poisson(lambda\_, 1000)

plt.hist(binomial\_approx, bins=range(15), density=True, alpha=0.5, label='Binomial')

plt.hist(poisson\_exact, bins=range(15), density=True, alpha=0.5, label='Poisson')

plt.title('Poisson as Limit of Binomial')

plt.legend()

plt.show()

Advanced Level

python

Copy

Download

# Poisson process simulation

def simulate\_poisson\_process(rate, T):

times = []

t = 0

while t < T:

dt = np.random.exponential(1/rate)

t += dt

if t < T:

times.append(t)

return np.array(times)

events = simulate\_poisson\_process(rate=0.5, T=100)

# Plot event occurrences

plt.eventplot(events, orientation='horizontal')

plt.title('Poisson Process Events')

plt.xlabel('Time')

plt.yticks([])

plt.show()

# Inter-arrival times should be exponential

inter\_arrival = np.diff(events)

plt.hist(inter\_arrival, bins=30, density=True)

plt.title('Inter-Arrival Times')

plt.show()

6. Comparing Distributions

python

Copy

Download

# Generate samples from all distributions

samples = {

'Uniform': np.random.uniform(0, 1, 1000),

'Normal': np.random.normal(0, 1, 1000),

'Binomial': np.random.binomial(10, 0.5, 1000),

'Poisson': np.random.poisson(3, 1000)

}

# Plot comparison

fig, axes = plt.subplots(2, 2, figsize=(10, 8))

for (name, data), ax in zip(samples.items(), axes.flat):

ax.hist(data, bins=30, density=True)

ax.set\_title(name)

plt.tight\_layout()

plt.show()

7. Advanced Applications

Monte Carlo Integration

python

Copy

Download

def monte\_carlo\_integrate(func, a, b, n\_samples=10000):

x = np.random.uniform(a, b, n\_samples)

y = func(x)

integral = (b - a) \* np.mean(y)

return integral

# Estimate integral of sin(x) from 0 to π

result = monte\_carlo\_integrate(np.sin, 0, np.pi)

print(f"Monte Carlo estimate of ∫sin(x)dx from 0 to π: {result:.4f}")

print(f"Exact value: {2:.4f}")

Bootstrapping

python

Copy

Download

def bootstrap\_ci(data, stat\_func=np.mean, n\_boot=10000, ci=95):

boots = []

for \_ in range(n\_boot):

sample = np.random.choice(data, size=len(data), replace=True)

boots.append(stat\_func(sample))

lower = (100 - ci) / 2

upper = 100 - lower

return np.percentile(boots, [lower, upper])

data = np.random.normal(0, 1, 100)

ci = bootstrap\_ci(data)

print(f"95% CI for mean: [{ci[0]:.2f}, {ci[1]:.2f}]")