COGMOD HWII

Itamar Oren-Naftalovich, Annabelle Choi

February 22, 2025

I

Explanation: The following statements are false:

- **False**: Discrete random variables use probability mass functions (or PMFs)
- **False**: The set of possible realizations of a random variable is its *support*
- **False**: The expected value of a discrete random variable can't be part of its support. Example: The expectation of a die roll is $\mathbb{E}[X] = 3.5$, which isn't in the support $\{1, 2, 3, 4, 5, 6\}$
- **False**: Bayesian models sample from the prior first and then from the likelihood, not the other way around
- **False**: The posterior $p(\theta \mid y)$ is a valid probability density function, so its integral over all θ is 1

II

Steps:

1. **Expectation:**
$$\mathbb{E}[\text{Return}] = (0.8 \times 0.01) + (0.2 \times -0.10) = -0.012 \quad \text{(-1.2\%)}$$
The expectation is negative, so it's not rational to invest.

2. **Minimal Probability p:** need to find p so that the expectation is non-negative:
$$0.01p - 0.10(1-p) \geq 0 \implies p \geq \frac{10}{11} \approx 90.9\%.$$

3. **Limitation:** Expectations don't account for risk/variance or the possibility of catastrophic losses in single-shot decisions. For a funny example of irrational investment decisions, you can see this example, which I don't think any model could have predicted this.

III

Derivations:

1. **Variance Identity:**
$$\text{Var}[X] = \mathbb{E}[(X-\mu)^2] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$$

2. **Scaling Property:** Shifting by β does not affect it:
$$\text{Var}[\alpha X + \beta] = \alpha^2 \text{Var}[X]$$

3. **Transformation to $\mathcal{N}(3, 5)$:** If $X \sim \mathcal{N}(0, 1)$, then:
$$\tilde{X} = 5X + 3 \quad \text{(scale by 5, shift by 3)}$$

IV

Calculation:

- **Prior:** $P(T) = \frac{1}{3}$, $P(L) = \frac{2}{3}$.
- **Likelihood:**
  - If true: $P(\text{"Yes"}|T) = \frac{1}{3}$ (truth)
  - If false: $P(\text{"Yes"}|L) = \frac{2}{3}$ (lie)
- **Posterior:**

$$P(T|\text{"Yes"}) = \frac{\frac{1}{3} \times \frac{1}{3}}{\frac{1}{3} \times \frac{1}{3} + \frac{2}{3} \times \frac{2}{3}} = \frac{1/9}{5/9} = \frac{1}{5} = 20\%$$

V

Prompt:

- Superman and Batman are roommates. They had a guest over and they discovered crumbs all over the place, mostly around the couch, the kitchen, and the gym. The two superheroes are the only suspects and the guest must find those who did not clean up after themselves.
- calculation is shown in the mainHW2P5.py and calculateHW2P5.py
- In conclusion: As the value of N increases, the simulated probability is closer to the analytic probability.

Simulated Probabilities

Table 1: Simulated Probabilities for $N = 1,000$

Couch	Kitchen	Gym	Culprit	Probability
False	False	False	Superman	10.10%
False	False	False	Batman	8.50%
False	False	True	Superman	1.80%
False	False	True	Batman	3.30%
False	True	False	Superman	19.50%
False	True	False	Batman	13.10%
False	True	True	Superman	3.60%
False	True	True	Batman	4.70%
True	False	False	Superman	3.80%
True	False	False	Batman	6.20%
True	False	True	Superman	0.90%
True	False	True	Batman	2.40%
True	True	False	Superman	8.70%
True	True	False	Batman	7.70%
True	True	True	Superman	1.70%
True	True	True	Batman	4.00%

Table 2: Simulated Probabilities for $N = 10,000$

Couch	Kitchen	Gym	Culprit	Probability
False	False	False	Superman	8.18%
False	False	False	Batman	9.09%
False	False	True	Superman	1.82%
False	False	True	Batman	3.46%
False	True	False	Superman	19.22%
False	True	False	Batman	13.13%
False	True	True	Superman	4.41%
False	True	True	Batman	5.37%
True	False	False	Superman	3.53%
True	False	False	Batman	5.87%
True	False	True	Superman	0.70%
True	False	True	Batman	2.35%
True	True	False	Superman	9.00%
True	True	False	Batman	8.37%
True	True	True	Superman	2.16%
True	True	True	Batman	3.34%

Table 3: Simulated Probabilities for $N = 100,000$

Couch	Kitchen	Gym	Culprit	Probability
False	False	False	Superman	8.40%
False	False	False	Batman	8.49%
False	False	True	Superman	2.12%
False	False	True	Batman	3.61%
False	True	False	Superman	19.79%
False	True	False	Batman	12.31%
False	True	True	Superman	4.99%
False	True	True	Batman	5.33%
True	False	False	Superman	3.57%
True	False	False	Batman	5.62%
True	False	True	Superman	0.91%
True	False	True	Batman	2.35%
True	True	False	Superman	8.52%
True	True	False	Batman	8.42%
True	True	True	Superman	2.00%
True	True	True	Batman	3.55%

Analytic Probabilities

Table 4: Analytic Probability

Couch	Kitchen	Gym	Culprit	Probability
False	False	False	Superman	8.40%
False	False	False	Batman	8.40%
False	False	True	Superman	2.10%
False	False	True	Batman	3.60%
False	True	False	Superman	19.60%
False	True	False	Batman	12.60%
False	True	True	Superman	4.90%
False	True	True	Batman	5.40%
True	False	False	Superman	3.60%
True	False	False	Batman	5.60%
True	False	True	Superman	0.90%
True	False	True	Batman	2.40%
True	True	False	Superman	8.40%
True	True	False	Batman	8.40%
True	True	True	Superman	2.10%
True	True	True	Batman	3.60%

VI

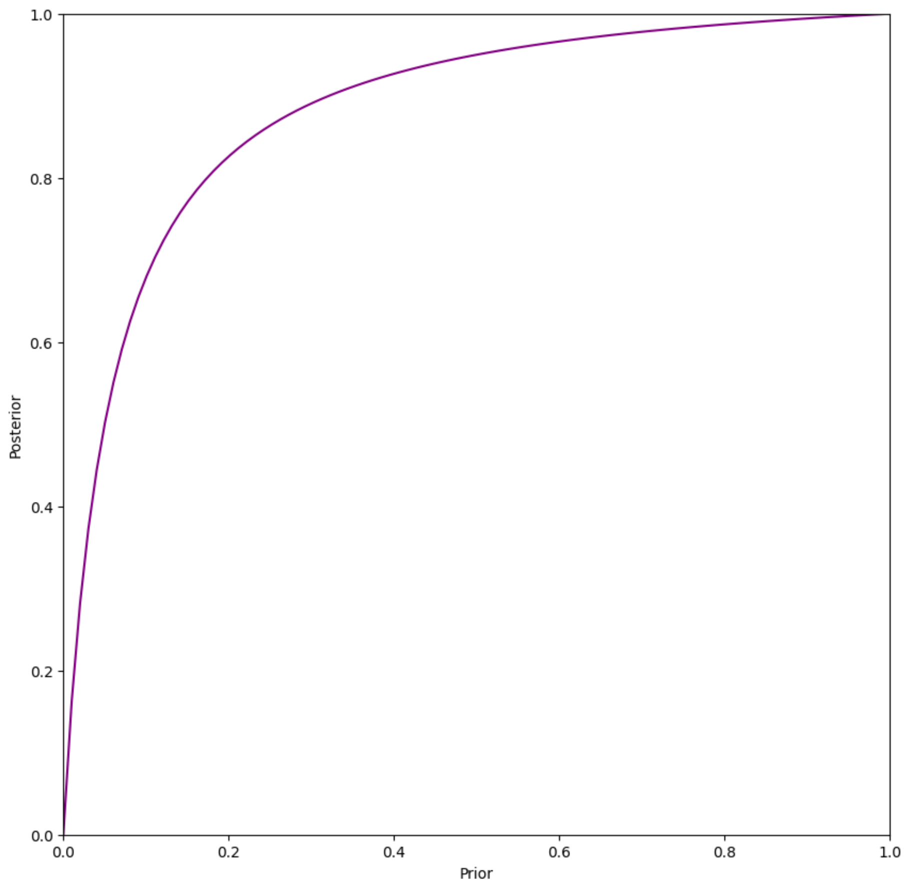
Posterior Calculation:

$$P(D|+) = \frac{0.95 \times 0.01}{0.95 \times 0.01 + 0.10 \times 0.99} \approx 8.76\%$$

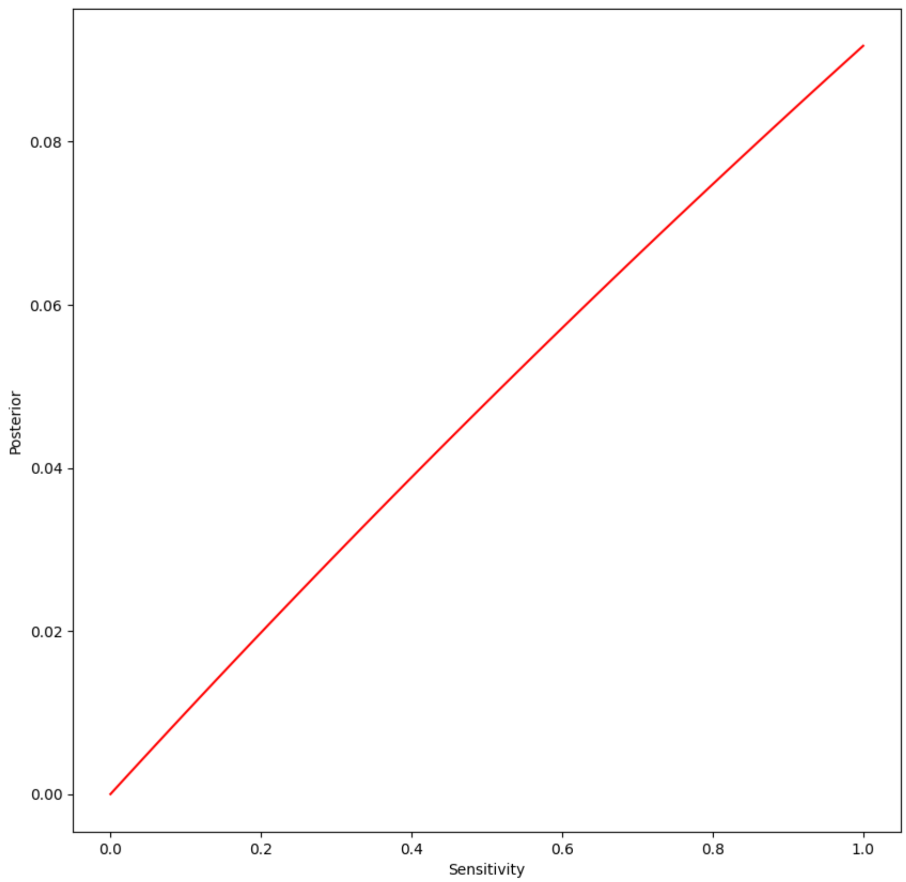
code is provided in hw2num6.ipynb

Graph Code Dump:

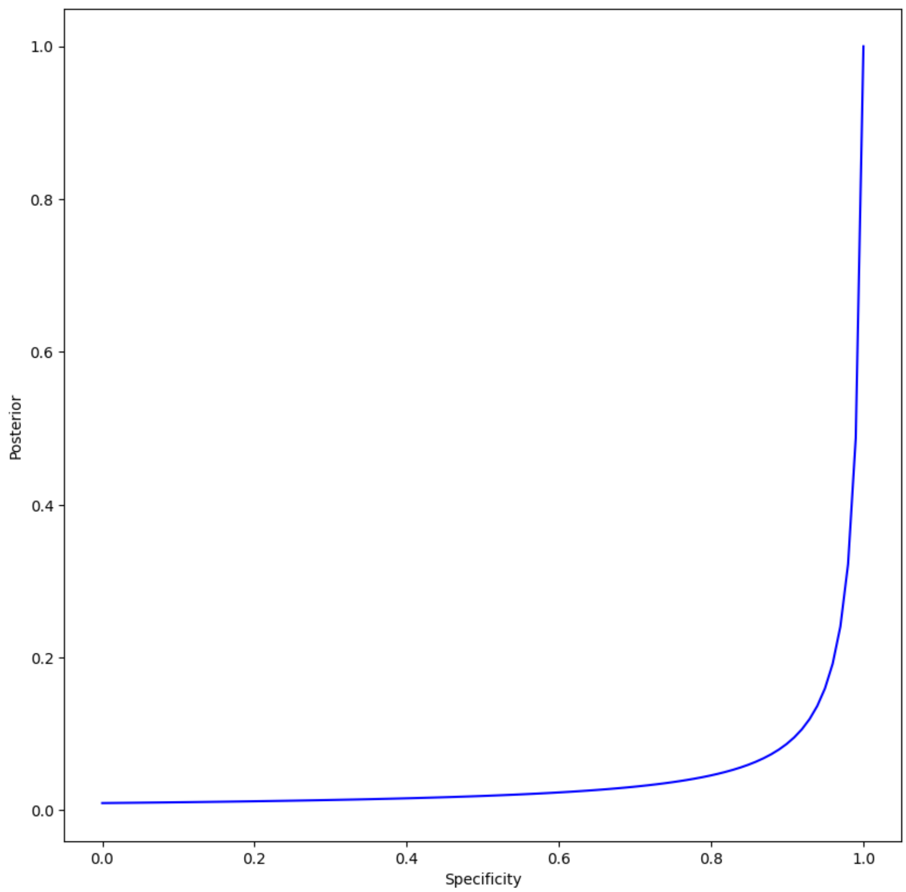
```
import numpy as np
import matplotlib.pyplot as plt

SENSITIVITY = 0.95
SPECIFICITY = 0.95

def posterior(prior, sensitivity, specificity):
    return (sensitivity * prior) / (sensitivity * prior + (1 - specificity) * (1 - prior))

priors = np.linspace(0, 1, 100)
posteriors = posterior(priors, SENSITIVITY, SPECIFICITY)

plt.figure(figsize=(10, 10))
plt.plot(priors, posteriors, color = 'purple')
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.xlabel('Prior')
plt.ylabel('Posterior')
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt

prior = 0.01
specificity = 0.90

def posterior_prob(prior, sensitivity, specificity):
    return (sensitivity * prior) / ((sensitivity * prior) + ((1 - specificity) * (1 - prior)))

sensitivities = np.linspace(0, 1, 100)
posteriors = [posterior_prob(prior, sens, specificity) for sens in sensitivities]

# Plot
plt.figure(figsize=(10, 10))
plt.plot(sensitivities, posteriors, color = 'red')
plt.xlabel('Sensitivity')
plt.ylabel('Posterior')
plt.show()
```

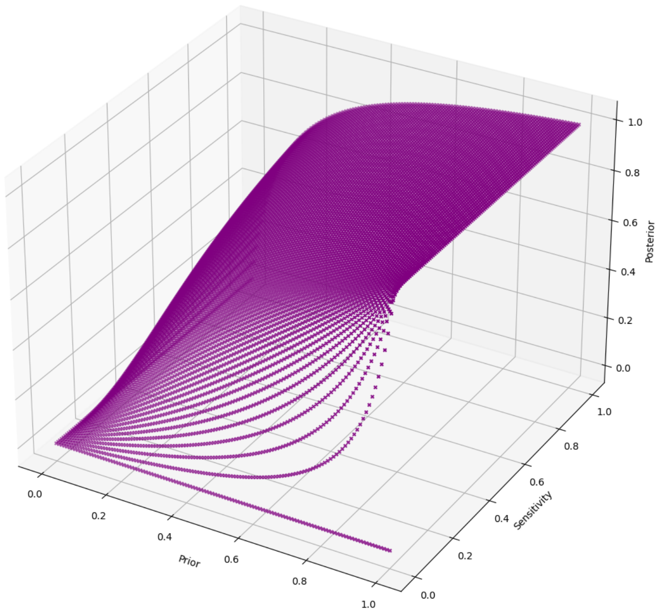


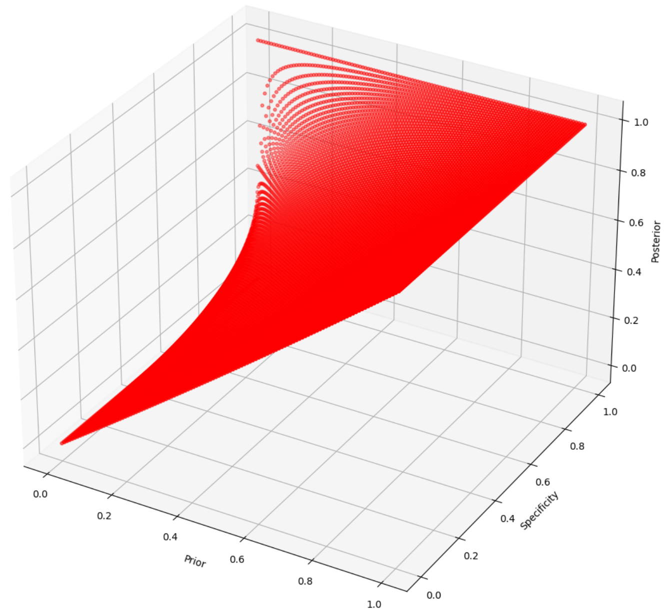

```
import numpy as np
import matplotlib.pyplot as plt

prior = 0.01
sensitivity = 0.95

def posterior_prob(prior, sensitivity, specificity):
    return (sensitivity * prior) / ((sensitivity * prior) + ((1 - specificity) * (1 - prior)))

# Vary specificity from 0 to 1
specificities = np.linspace(0, 1, 100)
posteriors = [posterior_prob(prior, sensitivity, sp) for sp in specificities]

# Plot
plt.figure(figsize=(10, 10))
plt.plot(specificities, posteriors, color = 'blue')
plt.xlabel('Specificity')
plt.ylabel('Posterior')
plt.show()
```

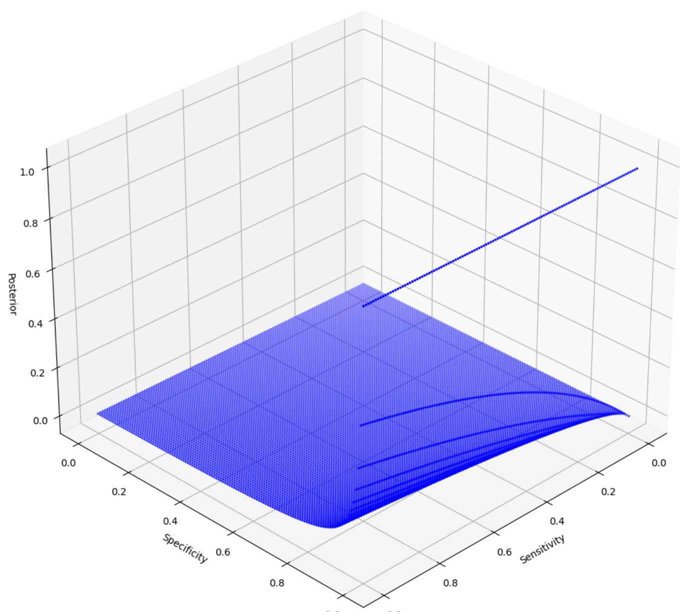


Posterior
Specificity
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

0.1 3D Scatter Plot







1. **Prior up → Posterior up:** A higher prior increases the likelihood of having the disease.
2. **Sensitivity up → Posterior up:** Better detection reduces false negatives.
3. **Specificity up → Posterior up:** Fewer false positives improve confidence in the test result.

VII —— Generated by ChatGPT free version

0.2 Introduction

This section presents a JavaScript implementation of a function `multivariateNormalDensity(x, mu, Sigma)` which computes the density of a D-dimensional vector x given a D-dimensional mean vector μ and a $D \times D$ covariance matrix

Σ. The density is defined as

$$f(x) = \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu) \right).$$

0.3 JavaScript Implementation

The following code uses ES modules and the `mathjs` library for matrix operations.

```
import { det, inv, subtract, dot, multiply, exp, sqrt, pow, pi } from ’mathjs’;

// function that computes the multivariate normal density
export function multivariateNormalDensity(x, mu, Sigma) {
  // compute dimension
  const d = x.length;
  // compute the determinant of the covariance matrix
  const sigmaDet = det(Sigma);
  // compute the normalization factor: 1 / sqrt((2pi)^d * det(Sigma))
  const normFactor = 1 / sqrt(pow(2 * pi, d) * sigmaDet);
  // compute the difference vector
  const diff = subtract(x, mu);
  // compute the quadratic form: diff’ * inv(Sigma) * diff
  const quadForm = dot(diff, multiply(inv(Sigma), diff));
  // compute the exponent part
  const exponent = -0.5 * quadForm;
  return normFactor * exp(exponent);
}
```

0.4 Testing the Function

Below are test cases for various parameterizations:

- **Spherical Gaussian**: shared variance across dimensions (zero covariance).
- **Diagonal Gaussian**: different variance for each dimension (zero covariance).
- **Full Covariance Gaussian**: non-zero covariance with different variances.

```
import { multivariateNormalDensity } from ’./multivariateNormalDensity.js’;

// spherical gaussian: shared variance, zero off-diagonals
const muSpherical = [0, 0];
const SigmaSpherical = [
  [1, 0],
  [0, 1]
];
const x1 = [0, 0];
console.log(’spherical␣gaussian␣at␣[0,0]:’, multivariateNormalDensity(x1, muSpherical, SigmaSpherical));

// diagonal gaussian: different variance for each dimension, still zero covariance
const muDiagonal = [1, 2];
const SigmaDiagonal = [
  [1, 0],
  [0, 4]
];
const x2 = [1, 2];
console.log(’diagonal␣gaussian␣at␣[1,2]:’, multivariateNormalDensity(x2, muDiagonal, SigmaDiagonal));

// full covariance gaussian: non-zero off-diagonals
const muFull = [0, 0];
const SigmaFull = [
  [1, 0.3],
  [0.3, 1]
];
const x3 = [0, 0];
console.log(’full␣covariance␣gaussian␣at␣[0,0]:’, multivariateNormalDensity(x3, muFull, SigmaFull));
```

0.5 Comparison with SciPy

I ran the above and got

```
spherical gaussian at [0,0]: 0.15915494309189535
diagonal gaussian at [1,2]: 0.07957747154594767
full covariance gaussian at [0,0]: 0.1668397135325737
```

I then ran SciPy's `scipy.stats.multivariate_normal.pdf` for the same parameterizations:

```
spherical gaussian with [0,0]: 0.15915494309189535
diagonal gaussian with [1,2]: 0.07957747154594767
full covariance gaussian with [0,0]: 0.1668397135325737
```

Comparison Table

Distribution Type	**Point Evaluated**	**JS Result**	**Python Result**
Spherical Gaussian	[0,0]	0.15915494309189535	0.15915494309189535
Diagonal Gaussian	[1,2]	0.07957747154594767	0.07957747154594767
Full Covariance Gaussian	[0,0]	0.1668397135325737	0.1668397135325737

Table 5: Comparison of multivariate normal density results from JavaScript and Python

Note: I had Deepseek create this table from the outputs because I struggled with correctly aligning it

0.6 Conclusion

I think the LLM did a really good job, it also added the needed libraries and formatting to style the code, which was nice.