lec08

February 12, 2026

1 Lecture 8: Bayesian Priors and the Meaning of Probability

Data 145, Spring 2026: Evidence and Uncertainty
Instructors: Ani Adhikari, William Fithian

Please run the setup cell below before reading.

---

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
from scipy.special import betaln

plt.style.use('fivethirtyeight')
%matplotlib inline

# Color scheme for Bayesian lectures (Scheme B, same as lec07)
# Black = data (likelihood)
# Blue = beliefs (prior and posterior; prior dashed, posterior solid)
# Red = asymptotic approximations (BvM normal approx, dashed)
COLOR_LIKELIHOOD = 'black'       # Data / likelihood
COLOR_PRIOR = 'steelblue'        # Prior belief (dashed)
COLOR_POSTERIOR = 'steelblue'    # Posterior belief (solid)
COLOR_APPROX = 'firebrick'      # Asymptotic approximations (dashed)
COLOR_TRUE = '#000000'           # True parameter value

# Shades of blue for comparing multiple posteriors
MULTI_BLUES = ['#1b4f72', '#2e86c1', '#5dade2', '#85c1e9']

np.random.seed(145)
```

1.1 Introduction

In Lecture 7 we developed the **Bayesian framework**: starting from a prior $\pi(\theta)$ and a likelihood $f_\theta(x)$, we computed the posterior $\pi(\theta \mid x) \propto f_\theta(x) \cdot \pi(\theta)$. We saw three conjugate fami-

lies (Beta-Binomial, Gamma-Exponential, Normal-Normal) and established the **Bernstein–von Mises theorem**: for large n, the posterior concentrates around the MLE regardless of the prior.

But we also noted a word of caution: unlike the likelihood model (which we can check against data), the prior is **uncheckable**. When n isn't very large — or the prior is sufficiently strong — the choice of prior can seriously affect our conclusions.

1.1.1 Today's roadmap

Where should priors come from? What does it even mean for θ to "have a distribution"? We'll explore four approaches (which are not mutually exclusive):

1. **Subjective Bayes**: The prior encodes personal degrees of belief
2. **Objective Bayes**: Flat priors and the Jeffreys prior — minimizing the influence of the prior
3. **Convenience priors**: Conjugate priors chosen for computational tractability
4. **Hierarchical Bayes**: Let the data from many similar problems inform the prior

The hierarchical approach will motivate the **Gibbs sampler**, a computational tool for sampling from complex posterior distributions.

1.2 1. Motivation: Uncheckable Priors

1.2.1 How strong is too strong?

In Lecture 7, we put three different priors on the earthquake rate λ — Gamma$(1, 1)$, Gamma$(1, 20)$, and Gamma$(1, 365)$ — and found that the posteriors were virtually identical. That was reassuring: it suggested the prior doesn't matter.

Recall from Lecture 7 that for the Gamma–Exponential model, the posterior mean of λ is $(\alpha + n)/(\beta + \sum x_i)$. We can interpret α as a "prior sample size" (number of pseudo-observations) and β as the "prior sum of interarrival times":

- Gamma$(1, 20)$: one pseudo-interarrival of 20 days (prior mean rate $= 0.05$/day, i.e., one earthquake every 20 days)
- Gamma$(1, 365)$: one pseudo-interarrival of 365 days (prior mean rate ≈ 0.003/day, about one per year)
- Gamma$(10, 3650)$: ten pseudo-interarrivals averaging 365 days — the same prior mean as Gamma$(1, 365)$, but with ten times the "prior sample size"

The first two priors are very weak — just a single pseudo-observation each. The third encodes the same guess about the rate as the second, but with more conviction. Below we compare all three to see whether this extra strength makes a difference.

[2]:
```
# Load earthquake data (same as lec07)
eq_data = pd.read_csv('../../demos/lec01_earthquakes/data/
 ↪california_earthquakes_declustered.csv')
mainshocks = eq_data[eq_data['is_mainshock']].sort_values('time').
 ↪reset_index(drop=True)
timestamps = pd.to_datetime(mainshocks['time'], format='ISO8601')
interarrivals = timestamps.diff().dt.total_seconds().dropna().values / 86400

n_eq = len(interarrivals)
```

```
sum_x = np.sum(interarrivals)
mle_lambda = 1 / np.mean(interarrivals)

print(f"Number of interarrival times: {n_eq}")
print(f"MLE rate: {mle_lambda:.4f} per day")

# Three priors of varying strength
priors = [
    (1,   20,   'Gamma(1, 20): weak, mean 0.05',       MULTI_BLUES[0]),
    (1,   365,  'Gamma(1, 365): weak, mean 0.003',     MULTI_BLUES[1]),
    (10,  3650, 'Gamma(10, 3650): stronger, mean 0.003', MULTI_BLUES[2]),
]

# Print posterior means
print(f"\n{'Prior':<42s} {'Prior mean':<12s} {'Post. mean':<12s}")
print("-" * 66)
for alpha_p, beta_p, label, _ in priors:
    post_mean = (n_eq + alpha_p) / (beta_p + sum_x)
    print(f"{label:<42s} {alpha_p/beta_p:<12.4f} {post_mean:<12.4f}")

# --- Figure 1 ---
fig, ax = plt.subplots(figsize=(10, 5))
lam_grid = np.linspace(0.025, 0.055, 500)

# Compute max posterior height for rescaling priors
max_post_height = 0
for alpha_p, beta_p, label, color in priors:
    pa = n_eq + alpha_p
    pb = beta_p + sum_x
    pdf = stats.gamma.pdf(lam_grid, a=pa, scale=1/pb)
    max_post_height = max(max_post_height, np.max(pdf))

for alpha_p, beta_p, label, color in priors:
    pa = n_eq + alpha_p
    pb = beta_p + sum_x
    post_pdf = stats.gamma.pdf(lam_grid, a=pa, scale=1/pb)
    prior_pdf = stats.gamma.pdf(lam_grid, a=alpha_p, scale=1/beta_p)

    # Rescale prior so its peak is ~25% of max posterior height
    if np.max(prior_pdf) > 0:
        prior_rescaled = prior_pdf * (max_post_height / np.max(prior_pdf)) * 0.
 ↪25
    else:
        prior_rescaled = prior_pdf

    ax.plot(lam_grid, prior_rescaled, color=color, linewidth=1.5,␣
 ↪linestyle='--', alpha=0.5)
```

```
    ax.plot(lam_grid, post_pdf, color=color, linewidth=2.5, label=label)

ax.axvline(mle_lambda, color=COLOR_LIKELIHOOD, linestyle=':', linewidth=1.5,
           label=f'MLE: {mle_lambda:.4f}')

ax.set_xlabel(r'$\lambda$ (earthquakes per day)', fontsize=11)
ax.set_ylabel('Density', fontsize=11)
ax.set_title('Three Priors and Their Posteriors for the Earthquake Rate',
             fontsize=13, fontweight='bold')
ax.legend(fontsize=9)
plt.tight_layout()
plt.show()
```

```
Number of interarrival times: 613
MLE rate: 0.0366 per day

Prior                                          Prior mean   Post. mean
-----------------------------------------------------------------------
Gamma(1, 20): weak, mean 0.05                  0.0500       0.0366
Gamma(1, 365): weak, mean 0.003                0.0027       0.0359
Gamma(10, 3650): stronger, mean 0.003          0.0027       0.0305
```

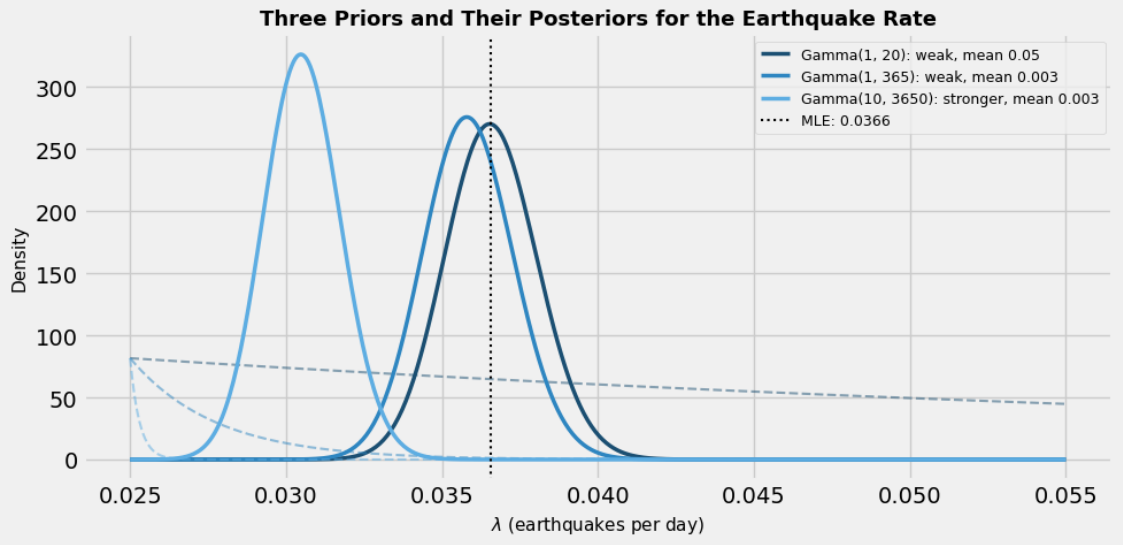

Figure 1. *Three Gamma priors (dashed, rescaled) and their posteriors (solid) for the California earthquake rate λ, with $n = 614$ interarrival times. The weak priors Gamma(1, 20) (dark blue) and Gamma(1, 365) (medium blue) yield virtually identical posteriors — the same reassuring result from Lecture 7. But the stronger Gamma(10, 3650) prior (light blue), which has the same mean but ten times the "prior sample size," noticeably pulls the posterior to the left of the MLE (black dotted). Unlike the exponential model itself, which we can check against data, there is no diagnostic to tell us our prior is wrong.*

1.2.2 The prior is uncheckable

We have an intuition that the Gamma$(10, 3650)$ prior is "too strong" — but how would we check? When we assessed the exponential model in Lecture 1, we could plot a histogram and compare it to the fitted density. For the prior, we have at most one "observation" of λ — and even that we don't observe directly. Come to think of it, what does it even *mean* to say λ has a distribution?

This question leads us to ask: **where should priors come from?** We'll discuss four approaches — subjective, objective, convenience, and hierarchical priors.

1.3 2. Epistemic Probability and Subjective Bayes

1.3.1 Two kinds of uncertainty

Consider two statements about earthquakes:

- "The waiting time until the next M ≥ 4 earthquake" — this feels genuinely random (**aleatory** uncertainty)
- "The rate λ at which earthquakes arrive in California" — this is a fixed fact about the world that we happen not to know (**epistemic** uncertainty)

Aleatory uncertainty is uncertainty about whether something will *happen* — randomness inherent in a physical process whose outcome hasn't yet been determined. **Epistemic uncertainty** is uncertainty about whether something *is true* — the fact is already settled, we just don't know it.

People argue about what kinds of things we should assign probabilities to. Here are some examples of events, ranging from most aleatory to most epistemic, that people may be more or less comfortable assigning probabilities to:

Example	Discussion
Outcome of rolling a die (most aleatory)	Most people agree on 1/6 regardless of philosophy — justified by repeated trials, physical symmetry, or rational indifference.
Whether a radiation treatment cures a cancer patient	Requires identifying a comparison group. As you condition on more specific patient characteristics (stage, age, family history, genetics), the reference class shrinks until potentially only one individual remains.
Whether the Democratic candidate wins the next presidential election	Every presidential election is a one-off event. Anyone qualified to assess it can identify important aspects that are more or less unprecedented.
Whether a subatomic particle has the mass predicted by a theory	Not the probability of something happening, but of something being *true* about the world — a fact established since the universe's creation.
Whether P = NP	A purely mathematical question, independent of empirical data. It may or may not be resolved in our lifetimes, yet people assign probabilities to it.

Example	Discussion
The 20th digit of $\sqrt{2}$ (most epistemic)	A deterministic fact, verifiable by calculation — yet without time to compute, one might assign 10% probability if forced to bet.

But is the distinction really so crisp? (See the covered-coin demo in lecture.)

1.3.2 Subjective Bayes

In the **subjective Bayesian** view, the prior $\pi(\theta)$ encodes a specific person's **degrees of belief** about θ before seeing data. These beliefs can differ between people, and that's fine — the posterior $\pi(\theta \mid X)$ tells *you* what *you* should believe after seeing the data, given what *you* believed before.

Pros: The subjective approach is philosophically clean. It can incorporate genuine expertise — for example, a seismologist's beliefs about earthquake rates — and provides a coherent framework for updating those beliefs in light of data.

Cons: If different people start with different priors, they get different posteriors. In practice, it is hard to elicit a full prior distribution from an expert. And when you write a paper, do you want the conclusion to be "in my personal opinion, the parameter is probably this"? Or do you want it to be, at least nominally, observer-agnostic?

1.4 3. Objective Bayes: Flat Priors and the Jeffreys Prior

1.4.1 Flat priors

Can we find a prior that represents "no prior information"? The simplest idea: use a **flat prior** $\pi(\theta) \propto 1$.

We've already seen this: the Uniform$(0, 1)$ prior for the Binomial, and the flat prior for the Normal mean. Flat priors are widely used and practically appealing: with a flat prior, the posterior mode equals the MLE.

But flat priors have two issues:

1. **Not always normalizable**: A flat prior on $(0, \infty)$ (e.g., for an exponential rate λ) doesn't integrate to a finite value. This isn't always fatal — the posterior can still be proper — but it requires care.
2. **Not reparameterization-invariant**: This is the deeper problem. A flat prior on θ is *not* flat on $g(\theta)$ — it depends on the parameterization!

1.4.2 The reparameterization problem

Consider the Binomial model. Let p be the success probability and $\eta = \log(p/(1-p))$ the log-odds.

- A flat prior on p: $\pi(p) \propto 1$ for $p \in (0, 1)$
- What does this imply for η? By the change-of-variables formula: $\pi(\eta) = \pi(p) \cdot |dp/d\eta| = \frac{e^\eta}{(1+e^\eta)^2}$

Recall: Change of variables for densities

If p has density $f_p(p)$ and $\eta = g(p)$ is a monotone transformation, the density of η is:

$$f_\eta(\eta) = f_p(p) \cdot \left|\frac{dp}{d\eta}\right|$$

where $p = g^{-1}(\eta)$. **Intuition**: If $dp/d\eta$ is large, a small interval in η maps to a large interval in p — so probability mass is "stretched out," and the density on p gets *multiplied* (not divided) by $|dp/d\eta|$.

This is the **standard logistic density** — not flat at all! It concentrates near $\eta = 0$ (i.e., $p = 1/2$).

Conversely, a flat prior on η would imply a non-flat, U-shaped prior on p.

Let's see this concretely.

[3]:
```
# Figure 2: Uniform-on-p prior in both parameterizations
n_samples = 1000
p_samples = np.random.uniform(0, 1, n_samples)
eta_samples = np.log(p_samples / (1 - p_samples))  # logit transform

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4.5))

# Left panel: p-parameterization
ax1.hist(p_samples, bins=30, density=True, color=COLOR_PRIOR, alpha=0.4,
         edgecolor='white', label='Histogram (1000 samples)')
ax1.axhline(1.0, color=COLOR_PRIOR, linewidth=2.5, linestyle='--',
            label='Uniform(0,1) density')
ax1.set_xlabel('$p$', fontsize=12)
ax1.set_ylabel('Density', fontsize=11)
ax1.set_title('Uniform-on-$p$ prior\n($p$-parameterization)',
              fontsize=12, fontweight='bold')
ax1.set_xlim(0, 1)
ax1.legend(fontsize=9)

# Right panel: eta-parameterization
eta_grid = np.linspace(-8, 8, 500)
logistic_density = np.exp(eta_grid) / (1 + np.exp(eta_grid))**2

ax2.hist(eta_samples, bins=40, density=True, color=COLOR_PRIOR, alpha=0.4,
         edgecolor='white', label='Histogram (1000 samples)')
ax2.plot(eta_grid, logistic_density, color=COLOR_PRIOR, linewidth=2.5,
         linestyle='--', label='Induced density (logistic)')
ax2.set_xlabel(r'$\eta = \log(p/(1-p))$', fontsize=12)
ax2.set_ylabel('Density', fontsize=11)
ax2.set_title('Uniform-on-$p$ prior\n($\eta$-parameterization)',
              fontsize=12, fontweight='bold')
ax2.set_xlim(-8, 8)
ax2.legend(fontsize=9)
```

```
plt.tight_layout()
plt.show()
```

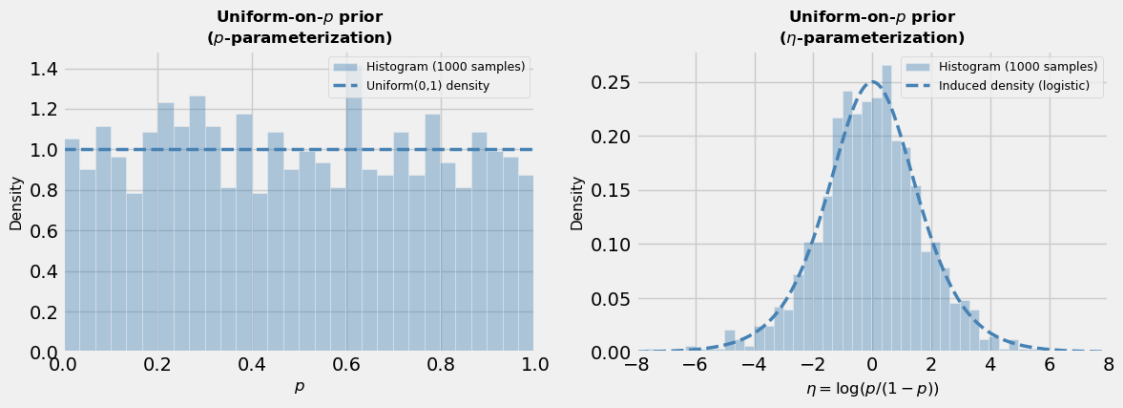

Figure 2. *A Uniform(0, 1) prior on p, displayed in both parameterizations. Left: the prior is flat on p (blue dashed line), and the histogram of 1000 samples confirms this. Right: the same 1000 samples transformed to the log-odds scale $\eta = \text{logit}(p)$. The induced density on η (blue dashed curve) is the standard logistic density, which concentrates near $\eta = 0$ (i.e., $p = 1/2$). A prior that is "uninformative" on p is informative on η.*

[4]:
```
# Figure 3: Two "noninformative" priors - density curves only (no histograms)
# The flat-on- prior is improper, so we show density curves rather than␣
 ↪samples.

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4.5))

# --- Left panel: p-parameterization ---
p_grid = np.linspace(0.005, 0.995, 500)

# Flat-on-eta induced density on p: proportional to 1/(p(1-p))
# Rescale so the visual area approximately matches the uniform (which␣
 ↪integrates to 1)
flat_eta_unnorm = 1.0 / (p_grid * (1 - p_grid))
area = np.trapz(flat_eta_unnorm, p_grid)
flat_eta_on_p = flat_eta_unnorm / area  # now integrates to ~1 over the␣
 ↪displayed range

ax1.plot(p_grid, flat_eta_on_p, color=COLOR_APPROX, linewidth=2.5,
         linestyle='--', label='Flat-on-$\\eta$: $\\propto 1/(p(1-p))$')
ax1.axhline(1.0, color=COLOR_PRIOR, linewidth=2, linestyle='--', alpha=0.7,
            label='Uniform-on-$p$: density')
ax1.set_xlabel('$p$', fontsize=12)
ax1.set_ylabel('Density', fontsize=11)
```

```
ax1.set_title('Two "noninformative" priors\n($p$-parameterization)',
              fontsize=12, fontweight='bold')
ax1.set_xlim(0, 1)
ax1.set_ylim(0, 6)
ax1.legend(fontsize=9)

# --- Right panel: eta-parameterization ---
eta_grid = np.linspace(-8, 8, 500)

# Flat-on-eta: constant (improper)
ax2.axhline(0.1, color=COLOR_APPROX, linewidth=2.5, linestyle='--',
            label='Flat-on-$\\eta$: $\\propto 1$ (improper)')

# Uniform-on-p induced density on eta: logistic density
logistic_density = np.exp(eta_grid) / (1 + np.exp(eta_grid))**2
ax2.plot(eta_grid, logistic_density, color=COLOR_PRIOR, linewidth=2,
         linestyle='--', alpha=0.7, label='Uniform-on-$p$: induced density')

ax2.set_xlabel(r'$\eta = \log(p/(1-p))$', fontsize=12)
ax2.set_ylabel('Density', fontsize=11)
ax2.set_title('Two "noninformative" priors\n($\\eta$-parameterization)',
              fontsize=12, fontweight='bold')
ax2.set_xlim(-8, 8)
ax2.legend(fontsize=9)

plt.tight_layout()
plt.show()
```

```
/tmp/ipykernel_197/2294386289.py:12: DeprecationWarning: `trapz` is deprecated.
Use `trapezoid` instead, or one of the numerical integration functions in
`scipy.integrate`.
  area = np.trapz(flat_eta_unnorm, p_grid)
```

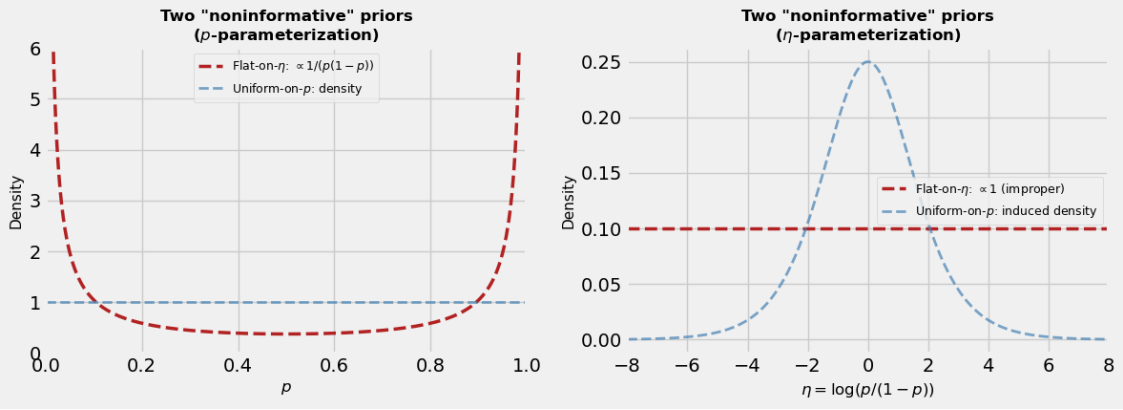

Figure 3. *Two "noninformative" priors for the binomial success probability, shown side by side in both parameterizations. Blue (dashed): Uniform on p. Red (dashed): flat on the log-odds $\eta = \text{logit}(p)$, which is improper (does not integrate to 1). Left panel: in the p-parameterization, the flat-on-η prior (red) is proportional to $1/(p(1-p))$ (rescaled so the minimum at $p = 1/2$ equals 1). It is U-shaped, concentrating near $p = 0$ and $p = 1$, and diverges to ∞ at the boundaries. Right panel: in the η-parameterization, the flat-on-η prior is constant (red), while the uniform-on-p prior (blue) induces the standard logistic density concentrating near $\eta = 0$. The two "noninformative" priors disagree — "flat" depends on the coordinate system.*

1.4.3 The lesson

"Flat" is not a well-defined concept for continuous parameters — it depends on the coordinate system. These two "noninformative" priors give different posteriors, so which one is really noninformative?

1.4.4 The Jeffreys prior: reparameterization invariance

Jeffreys' idea: Choose the prior $\pi(\theta) \propto \sqrt{I(\theta)}$, where $I(\theta)$ is the Fisher information.

Key property (stated without proof): The Jeffreys prior is **invariant under reparameterization**. If we change from θ to $\eta = g(\theta)$, the Jeffreys prior in η is just the change-of-variables transformation of the Jeffreys prior in θ. It doesn't matter which parameterization we work in — we always get the same answer.

1.4.5 Computing Jeffreys priors: examples

Binomial(n, p). The Fisher information is $I(p) = n/(p(1-p))$, so the Jeffreys prior is $\pi(p) \propto p^{-1/2}(1-p)^{-1/2}$. This is the Beta$(1/2, 1/2)$ distribution — not flat on p or on the log-odds η, but something in between.

Exponential(λ). The Fisher information is $I(\lambda) = 1/\lambda^2$, so the Jeffreys prior is $\pi(\lambda) \propto 1/\lambda$. This is **improper** — it doesn't integrate to a finite value — but the posterior is proper once we observe data. In the pseudo-data interpretation, the Jeffreys prior can be viewed as the limit of Gamma$(\varepsilon, \varepsilon\bar{x})$ as $\varepsilon \to 0$ for any pseudo-mean $\bar{x} > 0$: it corresponds to *zero* pseudo-observations. In this sense, the Jeffreys prior is even weaker than the flat prior $\pi(\lambda) \propto 1$, which corresponds to Gamma$(1, 0)$ — one pseudo-observation of $x = 0$. As a result, the Jeffreys posterior concentrates on the MLE, whereas the flat-prior posterior converges to a slightly shrunk estimator (reminiscent of Laplace's estimator from Lecture 6).

Normal$(\mu, \sigma^2$ known$)$. The Fisher information is $I(\mu) = 1/\sigma^2$ (constant), so $\pi(\mu) \propto 1$. Here the Jeffreys prior agrees with the flat prior — because the Normal location family is translation-invariant: shifting μ doesn't change the shape of the likelihood, only its location.

1.4.6 Limitations and philosophical tension

Jeffreys priors can be improper, and for multiparameter models they can behave poorly. In practice, they are most useful as a principled default for simple models.

There's a deeper tension: the subjective posterior had the problem that it was someone's personal opinion. The objective posterior has the opposite problem — it's *nobody's* opinion. So why should

we care about it? A practical answer: sometimes we just want to get out of the way of Bernstein–von Mises and let the data speak, without anyone suspecting we cooked up the prior to get the answer we wanted.

1.5 4. Convenience Priors

A **convenience prior** is one that's computationally easy to work with — typically a conjugate prior. This is compatible with multiple philosophies:

- **Informative**: Choose a conjugate prior that matches your prior mean and variance (e.g., a seismologist encoding beliefs via a Gamma prior)
- **Objective**: Use a flat or Jeffreys prior, if it happens to be conjugate (e.g., flat prior for the Normal mean)
- **Weakly informative**: Choose conjugate parameters that gently constrain θ without strongly influencing the posterior (e.g., Beta$(1, 1)$, Gamma$(1, \text{small})$)

The key advantage is **computational tractability**: conjugate priors give closed-form posteriors.

If we were truly trying to elicit our subjective beliefs, they probably wouldn't correspond to a conjugate family. For example, if I pull a coin out of my pocket, I probably think there's a very high chance it's a normal coin ($p \approx 0.5$), but a small chance it's a trick coin (p near 0 or 1). That belief — a spike at 0.5 with small bumps near 0 and 1 — is not a Beta density. (See the board sketch in lecture.)

But the convenience prior approach is especially appealing when we expect the prior doesn't matter much for the inference — then we might as well pick something easy to compute with.

1.6 5. Hierarchical Bayes

1.6.1 Motivation: many related problems

So far, we've been choosing a prior based on philosophy or convenience. But what if we have **many similar problems** whose data can help inform the prior?

1.6.2 Example: baseball batting averages (Efron & Morris, 1975)

There are 18 baseball players in our dataset, each with a true batting average p_i. For player i, we observe X_i hits in n_i at-bats: $X_i \mid p_i \sim \text{Binomial}(n_i, p_i)$.

The **naive approach** estimates each player independently by the MLE $\hat{p}_i = X_i/n_i$. But early in the season, every player has few at-bats, leading to noisy estimates. Roberto Clemente's 18-for-45 (0.400) makes him look like a superstar, while Max Alvis's 7-for-45 (0.156) makes him look terrible — but how much should we trust these numbers?

1.6.3 The hierarchical model

Model the p_i's as drawn from a common **population distribution**:

- **Hyperparameters**: $\alpha, \beta > 0$ (unknown)

- **Parameters**: $p_1, \ldots, p_m \mid \alpha, \beta \overset{\text{iid}}{\sim} \text{Beta}(\alpha, \beta)$
- **Data**: $X_i \mid p_i \sim \text{Binomial}(n_i, p_i)$, independently

The population distribution $\text{Beta}(\alpha, \beta)$ acts as a **learned prior** — we don't specify α and β in advance; the data across all players help us learn them.

1.6.4 Shrinkage toward the group mean

The posterior for each p_i "borrows strength" from all the other players, pulling each estimate toward the group average. This is **shrinkage** — the same idea from Lecture 6 (Laplace's estimator), but now the shrinkage target is learned from the data. With season-ending batting averages as ground truth, we can check whether the shrinkage estimates are actually better predictions.

[5]:
```
# Efron & Morris (1975) dataset: 18 MLB players from the 1970 season
# Each player had exactly 45 at-bats early in the season
# Season-ending batting averages provide ground truth

batting = pd.DataFrame({
    'player': [
        'Clemente',    'F Robinson', 'Howard',     'Johnstone',
        'Berry',       'Spencer',    'Kessinger',  'Alvarado',
        'Santo',       'Swoboda',    'Petrocelli', 'Rodriguez',
        'Scott',       'Unser',      'Williams',   'Campaneris',
        'Munson',      'Alvis',
    ],
    'hits': [
        18, 17, 16, 15,
        14, 14, 13, 12,
        11, 11, 10, 10,
        10, 10, 10,  9,
         8,  7,
    ],
    'at_bats': [45] * 18,
    'season_ba': [
        .352, .306, .283, .238,
        .276, .274, .266, .224,
        .267, .233, .261, .225,
        .296, .258, .251, .279,
        .302, .183,
    ],
})

batting['mle'] = batting['hits'] / batting['at_bats']
m = len(batting)

print(f"Number of players: {m}")
print(f"At-bats per player: {batting['at_bats'].iloc[0]}")
print(f"MLE range: {batting['mle'].min():.3f} to {batting['mle'].max():.3f}")
```

```
print(f"Overall batting average: {batting['hits'].sum() / batting['at_bats'].
  ↪sum():.3f}")
print()
print(batting[['player', 'hits', 'at_bats', 'mle', 'season_ba']].
  ↪to_string(index=False))
```

```
Number of players: 18
At-bats per player: 45
MLE range: 0.156 to 0.400
Overall batting average: 0.265

    player  hits  at_bats      mle  season_ba
  Clemente    18       45 0.400000      0.352
F Robinson    17       45 0.377778      0.306
    Howard    16       45 0.355556      0.283
 Johnstone    15       45 0.333333      0.238
     Berry    14       45 0.311111      0.276
   Spencer    14       45 0.311111      0.274
 Kessinger    13       45 0.288889      0.266
  Alvarado    12       45 0.266667      0.224
     Santo    11       45 0.244444      0.267
   Swoboda    11       45 0.244444      0.233
Petrocelli    10       45 0.222222      0.261
 Rodriguez    10       45 0.222222      0.225
     Scott    10       45 0.222222      0.296
     Unser    10       45 0.222222      0.258
  Williams    10       45 0.222222      0.251
Campaneris     9       45 0.200000      0.279
    Munson     8       45 0.177778      0.302
     Alvis     7       45 0.155556      0.183
```

This is the classic dataset from Efron & Morris (1975): 18 MLB players from the 1970 season, each with exactly $n = 45$ at-bats early in the season. The MLEs (hits/45) range from Clemente's 0.400 to Alvis's 0.156. The `season_ba` column records each player's batting average at the end of the full season — this serves as approximate "ground truth" for evaluating our estimates.

1.7 6. The Gibbs Sampler

1.7.1 Why we need computation

In the hierarchical model, the posterior $\pi(p_1, \dots, p_m, \alpha, \beta \mid X_1, \dots, X_m)$ has no closed-form expression. We can't just recognize the functional form as we did with conjugate models. We need a computational method to **sample** from this posterior.

1.7.2 The Gibbs sampler

Instead of sampling all parameters at once, we cycle through them one at a time, sampling each from its **full conditional distribution** — the distribution of that parameter given all the others

and the data.

For our hierarchical model, we reparameterize from (α, β) to $\mu = \alpha/(\alpha+\beta)$ (population mean) and $\sigma^2 = \mu(1-\mu)/(\alpha+\beta+1)$ (population variance), with a flat hyperprior on (μ, σ^2). This is more interpretable and reduces correlation between the hyperparameters. In each iteration:

- **Step (a)**: Sample each p_i from its conjugate Beta posterior (given $\alpha = \alpha(\mu, \sigma^2)$ and $\beta = \beta(\mu, \sigma^2)$) — easy!
- **Step (b)**: Sample μ from $\pi(\mu \mid \sigma^2, p_1, \ldots, p_m)$ — one-dimensional, computed via grid approximation
- **Step (c)**: Sample σ^2 from $\pi(\sigma^2 \mid \mu, p_1, \ldots, p_m)$ — same

The p_i updates are fast (closed-form Beta draws). The μ and σ^2 updates require grid approximation, but each is just a one-dimensional problem.

[6]:
```
def mu_sigma2_to_alpha_beta(mu, sigma2):
    """Convert (mu, sigma2) parameterization to (alpha, beta)."""
    kappa = mu * (1 - mu) / sigma2 - 1  # concentration = alpha + beta
    return mu * kappa, (1 - mu) * kappa

def log_conditional_mu(mu, sigma2, p_vec):
    """Log full conditional for mu (population mean) given sigma2 and p_1,...
 ↪,p_m."""
    m = len(p_vec)
    kappa = mu * (1 - mu) / sigma2 - 1
    if kappa <= 0:
        return -np.inf
    alpha = mu * kappa
    beta = (1 - mu) * kappa
    return (alpha - 1) * np.sum(np.log(p_vec)) + (beta - 1) * np.sum(np.log(1 -␣
 ↪p_vec)) - m * betaln(alpha, beta)

def log_conditional_sigma2(sigma2, mu, p_vec):
    """Log full conditional for sigma2 (population variance) given mu and p_1,..
 ↪.,p_m."""
    m = len(p_vec)
    kappa = mu * (1 - mu) / sigma2 - 1
    if kappa <= 0:
        return -np.inf
    alpha = mu * kappa
    beta = (1 - mu) * kappa
    return (alpha - 1) * np.sum(np.log(p_vec)) + (beta - 1) * np.sum(np.log(1 -␣
 ↪p_vec)) - m * betaln(alpha, beta)

def sample_from_log_grid(log_func, grid, *args):
    """Sample from a 1D distribution via grid approximation.

    Evaluates log_func on the grid, exponentiates with numerical stability,
    normalizes to a PMF, and draws one sample.
```

```
    """
    log_vals = np.array([log_func(g, *args) for g in grid])
    log_vals -= np.max(log_vals)  # numerical stability
    probs = np.exp(log_vals)
    probs /= probs.sum()
    return np.random.choice(grid, p=probs)
```

[7]:
```
# --- Gibbs sampler for the beta-binomial hierarchical model ---
# Parameterized by mu (population mean) and sigma2 (population variance),
# with flat hyperpriors on (mu, sigma2). More interpretable than (alpha, beta).

n_iter = 5000
burn_in = 1000

# Grids for mu and sigma2
mu_grid = np.linspace(0.01, 0.99, 300)
sigma2_grid = np.linspace(0.00005, 0.01, 300)

# Data
hits = batting['hits'].values
n_trials = batting['at_bats'].values

# Initialize
mu_current = 0.265
sigma2_current = 0.005
p_current = batting['mle'].values.copy()
p_current = np.clip(p_current, 0.01, 0.99)

# Storage
mu_trace = np.zeros(n_iter)
sigma2_trace = np.zeros(n_iter)
p_traces = np.zeros((n_iter, m))

for t in range(n_iter):
    # Step (a): Sample each p_i from Beta(X_i + alpha, n_i - X_i + beta)
    alpha_current, beta_current = mu_sigma2_to_alpha_beta(mu_current,␣
 ↪sigma2_current)
    for i in range(m):
        a_post = hits[i] + alpha_current
        b_post = n_trials[i] - hits[i] + beta_current
        p_current[i] = np.random.beta(a_post, b_post)

    # Step (b): Sample mu via grid approximation
    mu_current = sample_from_log_grid(
        log_conditional_mu, mu_grid, sigma2_current, p_current)

    # Step (c): Sample sigma2 via grid approximation
```

```
    sigma2_current = sample_from_log_grid(
        log_conditional_sigma2, sigma2_grid, mu_current, p_current)

    mu_trace[t] = mu_current
    sigma2_trace[t] = sigma2_current
    p_traces[t, :] = p_current

print(f"Gibbs sampler complete: {n_iter} iterations")
```

```
Gibbs sampler complete: 5000 iterations
```

[8]:
```
# Figure 4: Trace plots for mu (population mean) and sigma2 (population␣
↪variance)
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 6), sharex=True)

ax1.plot(mu_trace, color=MULTI_BLUES[0], linewidth=0.5, alpha=0.7)
ax1.axvline(burn_in, color=COLOR_APPROX, linestyle='--', linewidth=1.5,
            label=f'Burn-in ({burn_in} iter)')
ax1.set_ylabel(r'$\mu$ (population mean)', fontsize=13)
ax1.legend(fontsize=10)
ax1.set_title('Gibbs Sampler Trace Plots', fontsize=13, fontweight='bold')

ax2.plot(sigma2_trace, color=MULTI_BLUES[1], linewidth=0.5, alpha=0.7)
ax2.axvline(burn_in, color=COLOR_APPROX, linestyle='--', linewidth=1.5)
ax2.set_ylabel(r'$\sigma^2$ (population variance)', fontsize=13)
ax2.set_xlabel('Iteration', fontsize=11)

plt.tight_layout()
plt.show()
```

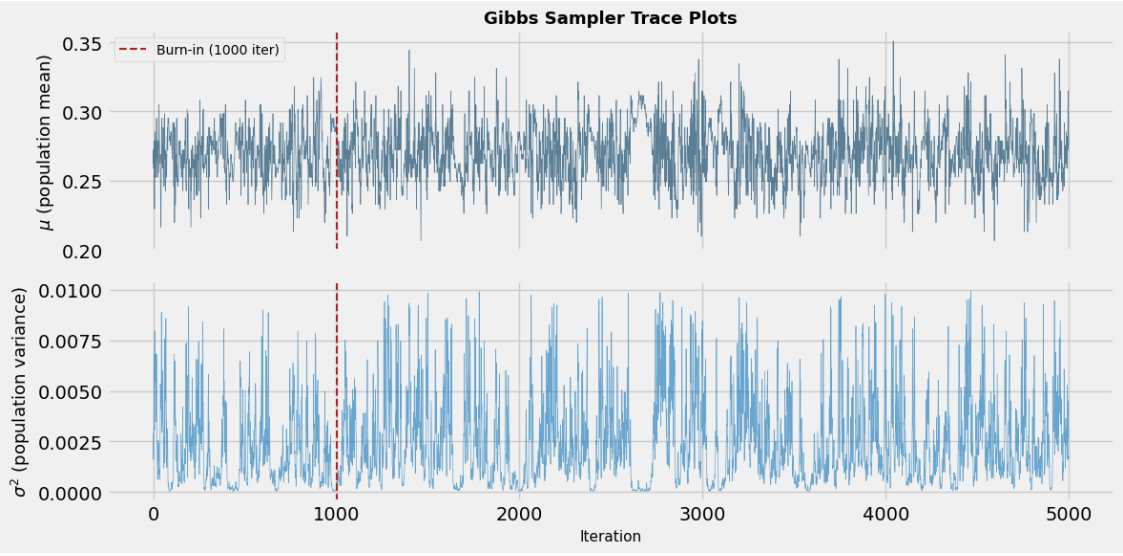

Figure 4. *Trace plots for the hyperparameters μ (population mean, top) and σ^2 (population variance, bottom) from the Gibbs sampler applied to the beta-binomial hierarchical model. The red dashed line marks the end of the burn-in period (1000 iterations). After burn-in, both chains appear stationary, indicating convergence to the posterior distribution.*

[9]:
```
# Posterior summaries from post-burn-in samples
mu_post = mu_trace[burn_in:]
sigma2_post = sigma2_trace[burn_in:]
p_post = p_traces[burn_in:, :]

posterior_means = p_post.mean(axis=0)
group_mean = mu_post.mean()

print(f"Posterior mean of mu (group mean):     {mu_post.mean():.3f}")
print(f"Posterior mean of sigma2 (pop. var.):  {sigma2_post.mean():.5f}")
print(f"Posterior SD of batting averages:      {np.sqrt(sigma2_post.mean()):.
 ↪3f}")
print()
print(f"{'Player':<14s} {'MLE':<8s} {'Post. mean':<12s} {'Season BA':<10s}␣
 ↪{'Shrinkage'}")
print("-" * 58)
for i in range(m):
    shrink = batting['mle'].iloc[i] - posterior_means[i]
    print(f"{batting['player'].iloc[i]:<14s} "
          f"{batting['mle'].iloc[i]:<8.3f} {posterior_means[i]:<12.3f} "
          f"{batting['season_ba'].iloc[i]:<10.3f} {shrink:+.3f}")

# MSE comparison: MLE vs season BA, posterior mean vs season BA
mse_mle = np.mean((batting['mle'].values - batting['season_ba'].values)**2)
mse_post = np.mean((posterior_means - batting['season_ba'].values)**2)
print(f"\nMSE comparison (predicting season-ending BA):")
print(f"  MLE:             {mse_mle:.5f}")
print(f"  Posterior mean:  {mse_post:.5f}")
print(f"  Improvement:     {(1 - mse_post/mse_mle)*100:.1f}%")
```

```
Posterior mean of mu (group mean):     0.270
Posterior mean of sigma2 (pop. var.):  0.00249
Posterior SD of batting averages:      0.050

Player         MLE      Post. mean   Season BA  Shrinkage
----------------------------------------------------------
Clemente       0.400    0.311        0.352      +0.089
F Robinson     0.378    0.305        0.306      +0.073
Howard         0.356    0.298        0.283      +0.058
Johnstone      0.333    0.290        0.238      +0.043
Berry          0.311    0.282        0.276      +0.030
Spencer        0.311    0.283        0.274      +0.028
Kessinger      0.289    0.276        0.266      +0.013
```

```
Alvarado        0.267    0.270        0.224      -0.003
Santo           0.244    0.261        0.267      -0.017
Swoboda         0.244    0.262        0.233      -0.017
Petrocelli      0.222    0.255        0.261      -0.032
Rodriguez       0.222    0.255        0.225      -0.033
Scott           0.222    0.255        0.296      -0.033
Unser           0.222    0.255        0.258      -0.033
Williams        0.222    0.254        0.251      -0.032
Campaneris      0.200    0.248        0.279      -0.048
Munson          0.178    0.241        0.302      -0.063
Alvis           0.156    0.233        0.183      -0.078

MSE comparison (predicting season-ending BA):
  MLE:             0.00327
  Posterior mean:  0.00098
  Improvement:     69.9%
```

[10]:
```python
# Figure 5: Shrinkage plot - MLE vs posterior mean, with season-ending BA as␣
 ↪ground truth
# Red circles + lines show MLE errors (shifted left); blue circles + lines show␣
 ↪posterior mean errors (shifted right)
fig, ax = plt.subplots(figsize=(10, 9))

# Determine plot range
all_vals = np.concatenate([batting['mle'].values, posterior_means,␣
 ↪batting['season_ba'].values])
lo = max(0, min(all_vals) - 0.03)
hi = min(1, max(all_vals) + 0.03)

offset = 0.005  # horizontal shift so red and blue lines don't overlap

# 45-degree line (no shrinkage)
ax.plot([lo, hi], [lo, hi], color=COLOR_LIKELIHOOD, linestyle='--',
        linewidth=1.5, alpha=0.5, label='No shrinkage ($y = x$)')

# Group mean line
ax.axhline(group_mean, color=COLOR_POSTERIOR, linestyle='--', linewidth=1.5,
           alpha=0.5, label=f'Group mean ({group_mean:.3f})')

for i in range(m):
    mle_i = batting['mle'].iloc[i]
    pm_i = posterior_means[i]
    sba_i = batting['season_ba'].iloc[i]

    # Red line + circle (shifted left): MLE to season BA
    ax.plot([mle_i - offset, mle_i - offset], [mle_i, sba_i],
            color=COLOR_APPROX, linewidth=1, alpha=0.5)
```

```
    ax.scatter(mle_i - offset, mle_i, c=COLOR_APPROX, s=50,
               edgecolors='white', linewidths=0.5, zorder=4)

    # Blue line + circle (shifted right): posterior mean to season BA
    ax.plot([mle_i + offset, mle_i + offset], [pm_i, sba_i],
            color=COLOR_POSTERIOR, linewidth=1, alpha=0.5)
    ax.scatter(mle_i + offset, pm_i, c=COLOR_POSTERIOR, s=70,
               edgecolors='white', linewidths=0.5, zorder=5)

    # Black x: season-ending BA (centered)
    ax.scatter(mle_i, sba_i, c='black', s=50, marker='x', linewidths=1.5,␣
 ↪zorder=6)

# --- Non-overlapping player labels ---
# Collect (name, point_x, point_y) sorted by posterior mean
label_info = [(batting['player'].iloc[i], batting['mle'].iloc[i] + offset,␣
 ↪posterior_means[i])
              for i in range(m)]
label_info.sort(key=lambda t: t[2])

# Compute adjusted y-positions with minimum vertical separation
min_sep = 0.013
adj_ys = [label_info[0][2]]
for j in range(1, len(label_info)):
    y = max(label_info[j][2], adj_ys[-1] + min_sep)
    adj_ys.append(y)

# Center the label block to minimize overall drift
drift = np.mean(adj_ys) - np.mean([t[2] for t in label_info])
adj_ys = [y - drift for y in adj_ys]

# Re-enforce minimum separation after centering
for j in range(1, len(adj_ys)):
    if adj_ys[j] - adj_ys[j-1] < min_sep:
        adj_ys[j] = adj_ys[j-1] + min_sep

# Place labels in a right-side column with connecting lines
label_x = hi - 0.005
for j, (name, px, py) in enumerate(label_info):
    ax.annotate(name, xy=(px, py), xytext=(label_x, adj_ys[j]),
                fontsize=7.5, color=COLOR_POSTERIOR, va='center', ha='right',
                arrowprops=dict(arrowstyle='-', color=COLOR_POSTERIOR, alpha=0.
 ↪25, lw=0.5))

# Legend entries for marker types
ax.scatter([], [], c=COLOR_APPROX, s=50, edgecolors='white', label='MLE (early␣
 ↪season)')
```

```
ax.plot([], [], color=COLOR_APPROX, linewidth=1, alpha=0.5, label='MLE error')
ax.scatter([], [], c=COLOR_POSTERIOR, s=70, edgecolors='white',␣
 ↪label='Posterior mean')
ax.plot([], [], color=COLOR_POSTERIOR, linewidth=1, alpha=0.5, label='Posterior␣
 ↪mean error')
ax.scatter([], [], c='black', s=50, marker='x', linewidths=1.5,␣
 ↪label='Season-ending BA (ground truth)')

ax.set_xlabel('MLE ($\\hat{p}_i = X_i / 45$)', fontsize=12)
ax.set_ylabel('Estimate / Ground Truth', fontsize=12)
ax.set_title('Shrinkage: MLEs vs Posterior Means\n(Efron & Morris, 1975)',
             fontsize=13, fontweight='bold')
ax.set_xlim(lo, hi)
ax.set_ylim(lo, hi)
ax.set_aspect('equal')
ax.legend(fontsize=9, loc='upper left')

plt.tight_layout()
plt.show()
```

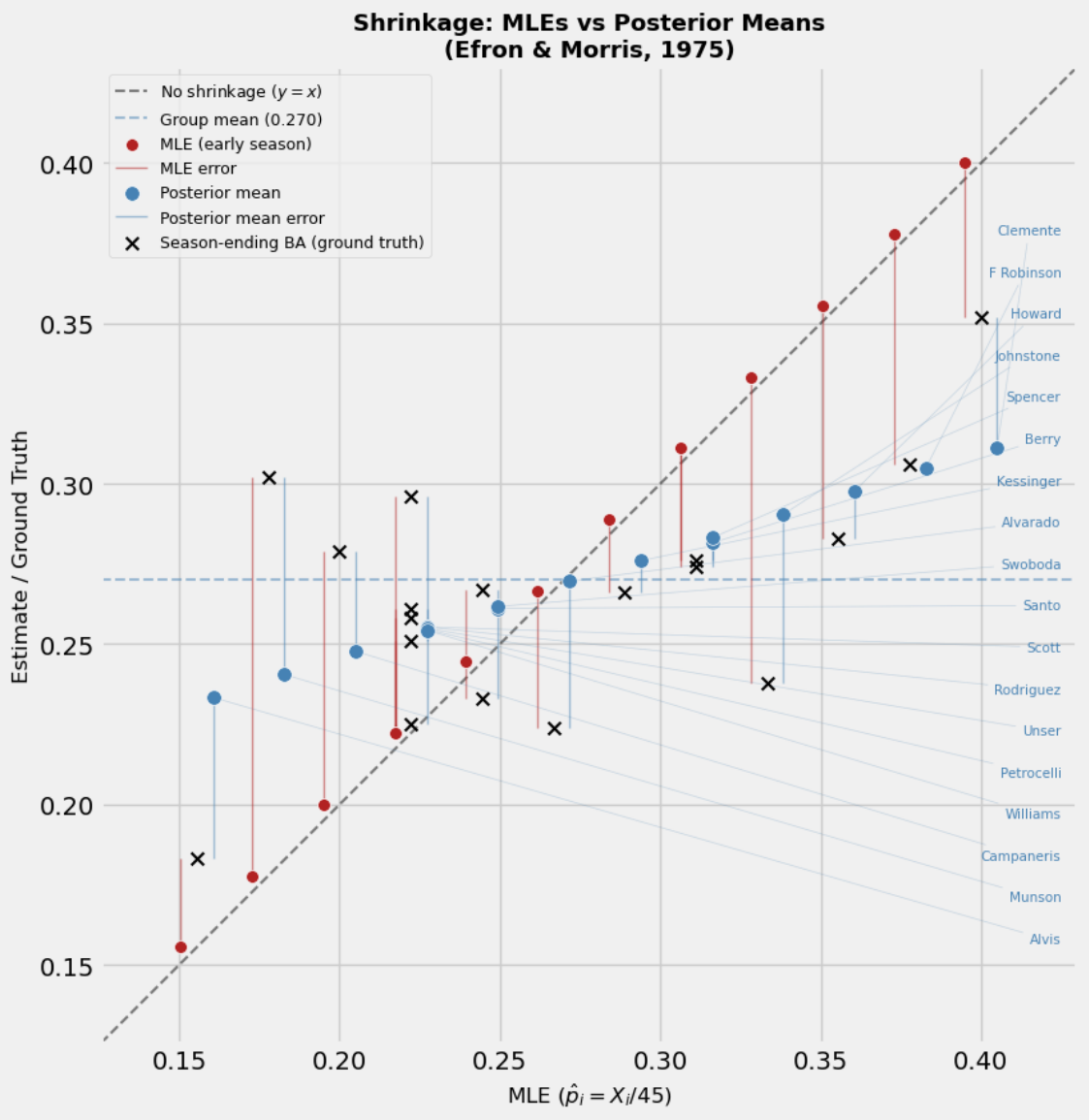

Figure 5. *Shrinkage in the beta-binomial hierarchical model applied to the Efron & Morris (1975) data. Each player's x-coordinate is the early-season MLE (hits/45). Red circles sit on the $y = x$ line (i.e., the MLE "prediction"); blue circles show posterior means; black × markers show season-ending batting averages (ground truth). Red lines connect each MLE to the ground truth, and blue lines connect each posterior mean to the ground truth — shorter lines mean smaller prediction errors. For most players the blue lines are shorter than the red lines, confirming that shrinkage toward the group mean (blue dashed) produces better predictions of season-ending performance.*

1.7.3 Why shrinkage makes sense

All 18 players have the same number of at-bats ($n = 45$), so each gets the same degree of shrinkage toward the group mean. The key insight is that early-season batting averages are noisy — 45 at-bats is not a lot — and shrinkage toward the group average produces better predictions of

season-ending performance. The MSE comparison above confirms this: the posterior means are substantially closer to the season-ending batting averages than the raw MLEs.

This is the same idea as Lecture 6: Laplace's estimator $(X + k)/(n + 2k)$ shrinks toward $1/2$ (an arbitrary, fixed target). The hierarchical model does the same thing, but the shrinkage target is **learned from the data** — it's the overall group batting average, not a predetermined value.

1.7.4 Why the Gibbs sampler works

Each conditional update **preserves** the target distribution $\pi(p_1, \ldots, p_m, \alpha, \beta \mid X)$. If the current state is a draw from the joint posterior and we resample one coordinate from its conditional, the result is still a draw from the joint posterior. Since each step preserves the target, the full cycle does too — so the posterior is a **stationary distribution** of the Markov chain. The chain also needs to be irreducible and aperiodic to guarantee convergence, but these hold under mild conditions.

1.8 7. Summary

1.8.1 Four approaches to priors

Approach	Key Idea	Pros	Cons
Subjective Bayes	Prior encodes personal degrees of belief	Can incorporate expertise	Whose beliefs? Hard to justify scientifically
Objective Bayes	Flat or Jeffreys prior; minimize prior influence	Observer-agnostic	"Nobody's opinion"; Jeffreys can be improper
Convenience priors	Conjugate priors for tractability	Closed-form posteriors	May not match actual beliefs
Hierarchical Bayes	Learn the prior from data across related problems	Shrinkage; borrows strength	Computationally harder; needs many related problems

1.8.2 Key technical results

- **Jeffreys prior**: $\pi(\theta) \propto \sqrt{I(\theta)}$. Computed for Binomial $\to$ Beta$(1/2, 1/2)$; Exponential $\to$ $\pi(\lambda) \propto 1/\lambda$; Normal location $\to$ flat
- **Hierarchical Bayes**: The population distribution serves as a learned prior; the Gibbs sampler provides a computational tool for posterior inference
- **Gibbs sampler**: Cycle through parameters, sampling each from its full conditional. Preserves the target distribution because resampling one coordinate from its conditional doesn't change the joint.

1.8.3 Key takeaways

1. Priors are uncheckable — unlike the model, there's no diagnostic to tell you your prior is wrong
2. Flat priors are a simple noninformative default, but they're not reparameterization-invariant; the Jeffreys prior $\pi(\theta) \propto \sqrt{I(\theta)}$ resolves this ambiguity
3. Hierarchical Bayes lets the data inform the prior across many similar problems, producing shrinkage estimators that borrow strength across units
4. The Gibbs sampler is a practical MCMC algorithm for posterior computation: cycle through coordinates, sample each from its full conditional

Next time (Lecture 9): What if the model itself is wrong? Model misspecification and Kullback-Leibler divergence.