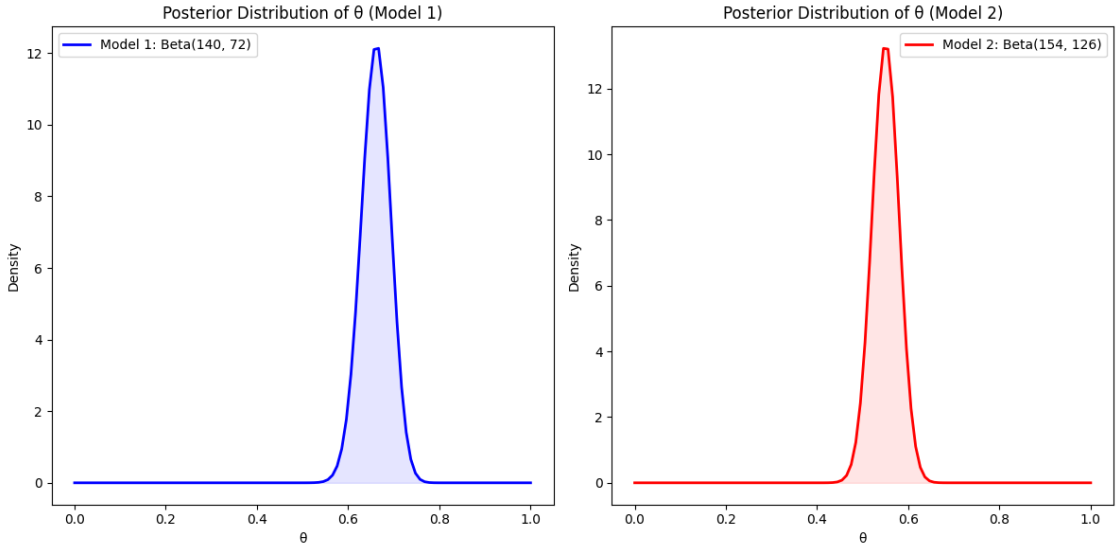
assignment-5

July 11, 2024

[1]:
```
#Exercise 1.1

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import beta

# Given data
data = [10, 15, 15, 14, 14, 14, 13, 11, 12, 16]

# Compute the number of successes and failures
num_successes = np.sum(data)  # Sum of data points
num_failures = len(data) * 20 - num_successes  # Total trials - sum of successes

# Model 1: Beta(6, 6)
alpha1 = 6 + num_successes
beta1 = 6 + len(data) * 20 - num_successes

# Model 2: Beta(20, 60)
alpha2 = 20 + num_successes
beta2 = 60 + len(data) * 20 - num_successes

# Create a range of theta values
theta_values = np.linspace(0, 1, 100)  # 100 points between 0 and 1

# Compute the posterior distributions for each model
posterior_model1 = beta.pdf(theta_values, alpha1, beta1)
posterior_model2 = beta.pdf(theta_values, alpha2, beta2)

# Plot the posterior distributions
plt.figure(figsize=(12, 6))

# Model 1 Plot
plt.subplot(1, 2, 1)
plt.plot(theta_values, posterior_model1, 'b-', lw=2, label=f'Model 1:␣
 ↪Beta({alpha1}, {beta1})')
plt.fill_between(theta_values, 0, posterior_model1, color='blue', alpha=0.1)
plt.title('Posterior Distribution of   (Model 1)')
```

```
plt.xlabel(' ')
plt.ylabel('Density')
plt.legend()

# Model 2 Plot
plt.subplot(1, 2, 2)
plt.plot(theta_values, posterior_model2, 'r-', lw=2, label=f'Model 2:␣
 ↪Beta({alpha2}, {beta2})')
plt.fill_between(theta_values, 0, posterior_model2, color='red', alpha=0.1)
plt.title('Posterior Distribution of   (Model 2)')
plt.xlabel(' ')
plt.ylabel('Density')
plt.legend()

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



[12]:
```
# Exercise 1.2 and 1.3
import numpy as np
from scipy.stats import beta, binom

# Given data
data = [10, 15, 15, 14, 14, 14, 13, 11, 12, 16]

# Compute the number of successes and failures
num_successes = np.sum(data)  # Sum of data points
num_failures = len(data) * 20 - num_successes  # Total trials - sum of successes
```

```python
# Define the number of posterior samples
N = 10000

# Model 1: Beta(6, 6)
alpha1 = 6 + num_successes
beta1 = 6 + len(data) * 20 - num_successes

# Model 2: Beta(20, 60)
alpha2 = 20 + num_successes
beta2 = 60 + len(data) * 20 - num_successes

# Draw samples from the posterior distributions
samples_model1 = beta.rvs(alpha1, beta1, size=N)
samples_model2 = beta.rvs(alpha2, beta2, size=N)

# Compute the log predictive densities for each data point for both models
log_pred_density_model1 = np.zeros(len(data))
log_pred_density_model2 = np.zeros(len(data))

for i, y in enumerate(data):
    # Predictive density for each data point given  j
    pred_density_model1 = binom.pmf(y, 20, samples_model1)
    pred_density_model2 = binom.pmf(y, 20, samples_model2)

    # Compute log of the average predictive density
    log_pred_density_model1[i] = np.log(np.mean(pred_density_model1))
    log_pred_density_model2[i] = np.log(np.mean(pred_density_model2))

# Compute the log pointwise predictive density (lppd) for each model
lppd_model1 = np.sum(log_pred_density_model1)
lppd_model2 = np.sum(log_pred_density_model2)

# Compute the in-sample deviance for each model
deviance_model1 = -2 * lppd_model1
deviance_model2 = -2 * lppd_model2

# Output the results
print(f"Log Pointwise Predictive Density (lppd) for Model 1: {lppd_model1:.4f}")
print(f"Log Pointwise Predictive Density (lppd) for Model 2: {lppd_model2:.4f}")
print(f"In-Sample Deviance for Model 1: {deviance_model1:.4f}")
print(f"In-Sample Deviance for Model 2: {deviance_model2:.4f}")

# Compare which model is better
if deviance_model1 < deviance_model2:
    print("Model 1 is a better fit to the data.")
else:
    print("Model 2 is a better fit to the data.")
```

```
print("In-sample deviance measures how well the model fits the training data.␣
 ↪The term in-sample refers to the data used to fit the model in this case,␣
 ↪the training data. The deviance is a measure of goodness-of-fit, where lower␣
 ↪values indicate a better fit. By calculating the in-sample deviance, we␣
 ↪assess the model's performance on the data it was trained on, which provides␣
 ↪insight into how well the model captures the underlying patterns in the data.
 ↪ This measure is crucial because a good fit to the training data is a␣
 ↪prerequisite for making reliable predictions on new, unseen data.")
```

```
Log Pointwise Predictive Density (lppd) for Model 1: -20.3690
Log Pointwise Predictive Density (lppd) for Model 2: -25.8963
In-Sample Deviance for Model 1: 40.7381
In-Sample Deviance for Model 2: 51.7926
Model 1 is a better fit to the data.
In-sample deviance measures how well the model fits the training data. The term
in-sample refers to the data used to fit the model in this case, the training
data. The deviance is a measure of goodness-of-fit, where lower values indicate
a better fit. By calculating the in-sample deviance, we assess the model's
performance on the data it was trained on, which provides insight into how well
the model captures the underlying patterns in the data. This measure is crucial
because a good fit to the training data is a prerequisite for making reliable
predictions on new, unseen data.
```

[4]:
```
# Exercise 1.4
#Answer: Model 1 is a better fit to the data.
#Explanation: The in-sample deviance is calculated as -2×lppd-2×lppd. Since␣
 ↪Model 1 has a higher log pointwise predictive density (lppd) compared to␣
 ↪Model 2, it results in a lower in-sample deviance. A lower deviance␣
 ↪indicates a better model fit.
# Compute the in-sample deviance for each model
deviance_model1 = -2 * lppd_model1
deviance_model2 = -2 * lppd_model2

# Output the results
print(f"In-Sample Deviance for Model 1: {deviance_model1:.3f}")
print(f"In-Sample Deviance for Model 2: {deviance_model2:.3f}")

# Compare which model is better
if deviance_model1 < deviance_model2:
    print("Model 1 is a better fit to the data.")
else:
    print("Model 2 is a better fit to the data.")
```

```
In-Sample Deviance for Model 1: 40.734
In-Sample Deviance for Model 2: 51.864
Model 1 is a better fit to the data.
```

[5]:
```python
# Exercise 1.5
import numpy as np
from scipy.stats import beta, binom

# Given new data points
new_data = [5, 6, 10, 8, 9]

# Model parameters
data = [10, 15, 15, 14, 14, 14, 13, 11, 12, 16]
num_successes = np.sum(data)
num_failures = len(data) * 20 - num_successes  # Total trials - sum of successes

# Define the number of posterior samples
N = 10000

# Model 1: Beta(6, 6)
alpha1 = 6 + num_successes
beta1 = 6 + len(data) * 20 - num_successes

# Model 2: Beta(20, 60)
alpha2 = 20 + num_successes
beta2 = 60 + len(data) * 20 - num_successes

# Draw samples from the posterior distributions
samples_model1 = beta.rvs(alpha1, beta1, size=N)
samples_model2 = beta.rvs(alpha2, beta2, size=N)

# Compute the log predictive densities for each new data point for both models
log_pred_density_new_model1 = np.zeros(len(new_data))
log_pred_density_new_model2 = np.zeros(len(new_data))

for i, y in enumerate(new_data):
    # Predictive density for each data point given  j
    pred_density_new_model1 = binom.pmf(y, 20, samples_model1)
    pred_density_new_model2 = binom.pmf(y, 20, samples_model2)

    # Compute log of the average predictive density
    log_pred_density_new_model1[i] = np.log(np.mean(pred_density_new_model1))
    log_pred_density_new_model2[i] = np.log(np.mean(pred_density_new_model2))

# Compute the log pointwise predictive density (lppd) for each model
lppd_new_model1 = np.sum(log_pred_density_new_model1)
lppd_new_model2 = np.sum(log_pred_density_new_model2)

# Compute the out-of-sample deviance for each model
deviance_new_model1 = -2 * lppd_new_model1
deviance_new_model2 = -2 * lppd_new_model2
```

```
# Output the results
print(f"Log Pointwise Predictive Density (lppd) for New Data using Model 1:␣
 ↪{lppd_new_model1:.4f}")
print(f"Log Pointwise Predictive Density (lppd) for New Data using Model 2:␣
 ↪{lppd_new_model2:.4f}")
print(f"Out-of-Sample Deviance for Model 1: {deviance_new_model1:.4f}")
print(f"Out-of-Sample Deviance for Model 2: {deviance_new_model2:.4f}")

# Compare which model is better
if deviance_new_model1 < deviance_new_model2:
    print("Model 1 is better at predicting new data.")
else:
    print("Model 2 is better at predicting new data.")
```

```
Log Pointwise Predictive Density (lppd) for New Data using Model 1: -25.2022
Log Pointwise Predictive Density (lppd) for New Data using Model 2: -15.7932
Out-of-Sample Deviance for Model 1: 50.4044
Out-of-Sample Deviance for Model 2: 31.5863
Model 2 is better at predicting new data.
```

[6]:
```
# Exercise 1.6

import numpy as np
from scipy.stats import beta, binom

# Given data points
data = [10, 15, 15, 14, 14, 14, 13, 11, 12, 16]

# Define the number of posterior samples
N = 10000

# Compute number of successes and failures for the full dataset
num_successes = np.sum(data)
num_failures = len(data) * 20 - num_successes  # Total trials - sum of successes

# Define arrays to store log predictive densities
log_pred_density_loo_model1 = []
log_pred_density_loo_model2 = []

# Perform LOO-CV
for i in range(len(data)):
    # Leave one out
    loo_data = data[:i] + data[i+1:]
    loo_num_successes = np.sum(loo_data)
    loo_num_failures = len(loo_data) * 20 - loo_num_successes
```

```
    # Fit Model 1 on the remaining 9 data points
    alpha1 = 6 + loo_num_successes
    beta1 = 6 + len(loo_data) * 20 - loo_num_successes
    samples_model1 = beta.rvs(alpha1, beta1, size=N)

    # Fit Model 2 on the remaining 9 data points
    alpha2 = 20 + loo_num_successes
    beta2 = 60 + len(loo_data) * 20 - loo_num_successes
    samples_model2 = beta.rvs(alpha2, beta2, size=N)

    # Compute log predictive density for the left-out data point
    y_i = data[i]
    pred_density_model1 = binom.pmf(y_i, 20, samples_model1)
    pred_density_model2 = binom.pmf(y_i, 20, samples_model2)

    log_pred_density_loo_model1.append(np.log(np.mean(pred_density_model1)))
    log_pred_density_loo_model2.append(np.log(np.mean(pred_density_model2)))

# Compute average log pointwise predictive density (lppd) for each model
lppd_loo_model1 = np.mean(log_pred_density_loo_model1)
lppd_loo_model2 = np.mean(log_pred_density_loo_model2)

# Compute LOO-CV deviance for each model
deviance_loo_model1 = -2 * lppd_loo_model1
deviance_loo_model2 = -2 * lppd_loo_model2

# Output the results
print(f"Average Log Pointwise Predictive Density (lppd) for LOO-CV using Model␣
 ↪1: {lppd_loo_model1:.4f}")
print(f"Average Log Pointwise Predictive Density (lppd) for LOO-CV using Model␣
 ↪2: {lppd_loo_model2:.4f}")
print(f"LOO-CV Deviance for Model 1: {deviance_loo_model1:.4f}")
print(f"LOO-CV Deviance for Model 2: {deviance_loo_model2:.4f}")

# Compare which model is better
if deviance_loo_model1 < deviance_loo_model2:
    print("Model 1 is better based on LOO-CV.")
else:
    print("Model 2 is better based on LOO-CV.")
```

```
Average Log Pointwise Predictive Density (lppd) for LOO-CV using Model 1:
-2.1104
Average Log Pointwise Predictive Density (lppd) for LOO-CV using Model 2:
-2.7250
LOO-CV Deviance for Model 1: 4.2209
LOO-CV Deviance for Model 2: 5.4501
Model 1 is better based on LOO-CV.
```

[16]:
```python
#Exercise 2.1
import math  # Import the math library to use the factorial function

# Function to calculate the marginal likelihood
def ML_binomial(k, n, a, b):
    # Compute the marginal likelihood using the binomial likelihood and Beta␣
 ↪prior
    ML = (math.factorial(n) / (math.factorial(k) * math.factorial(n - k))) * (
        math.factorial(k + a - 1) * math.factorial(n - k + b - 1) / math.
 ↪factorial(n + a + b - 1)
    )
    return ML

# Parameters for the binomial model
k = 2  # Number of successes
n = 10  # Number of trials

# Define the list of Beta priors to use
priors = {
    "Beta_1_1": (1, 1),
    "Beta_2_6": (2, 6),
    "Beta_6_2": (6, 2),
    "Beta_20_60": (20, 60),
    "Beta_60_20": (60, 20)
}

# Compute and print the marginal likelihood for each Beta prior
for name, (a, b) in priors.items():
    marginal_likelihood = ML_binomial(k, n, a, b)
    print(f"Marginal Likelihood for {name} (Beta({a}, {b})):␣
 ↪{marginal_likelihood:.2e}")
```

```
Marginal Likelihood for Beta_1_1 (Beta(1, 1)): 9.09e-02
Marginal Likelihood for Beta_2_6 (Beta(2, 6)): 4.73e-03
Marginal Likelihood for Beta_6_2 (Beta(6, 2)): 2.31e-04
Marginal Likelihood for Beta_20_60 (Beta(20, 60)): 5.08e-21
Marginal Likelihood for Beta_60_20 (Beta(60, 20)): 1.51e-23
```

[17]:
```python
#exercise 2.2
import numpy as np
from scipy.stats import binom, beta, norm, uniform

# Parameters for the binomial model
k = 2
n = 10

# Define the list of Beta priors to use
```

```
priors = {
    "Beta_0.1_0.4": (0.1, 0.4),
    "Beta_1_1": (1, 1),
    "Beta_2_6": (2, 6),
    "Beta_6_2": (6, 2),
    "Beta_20_60": (20, 60),
    "Beta_60_20": (60, 20)
}

# Function to estimate the marginal likelihood using Monte Carlo Integration
def estimate_marginal_likelihood(a, b, k, n, num_samples=10000, proposal_sd=0.
 ↪08):
    theta_samples = np.zeros(num_samples)  # Initialize Markov Chain Monte␣
 ↪Carlo (MCMC) samples
    theta_samples[0] = 0.4  # Starting value for theta
    ML_estimate = 0
    i = 1

    while i < num_samples:
        # Propose a new value for theta
        proposed_theta = np.random.normal(theta_samples[i - 1], proposal_sd)

        # Check if the proposed value is within the valid range
        if 0 < proposed_theta < 1:
            # Compute the posterior probabilities for the proposed and current␣
 ↪theta values
            post_new = binom.pmf(k, n, proposed_theta) * beta.
 ↪pdf(proposed_theta, a, b)
            post_prev = binom.pmf(k, n, theta_samples[i - 1]) * beta.
 ↪pdf(theta_samples[i - 1], a, b)

            # Compute the proposal density ratio
            proposal_density_ratio = (post_new * norm.pdf(theta_samples[i - 1],␣
 ↪proposed_theta, proposal_sd)) / (post_prev * norm.pdf(proposed_theta,␣
 ↪theta_samples[i - 1], proposal_sd))

            # Acceptance criterion
            acceptance_prob = min(proposal_density_ratio, 1)
            if acceptance_prob > np.random.uniform(0, 1):
                theta_samples[i] = proposed_theta  # Accept the new value
                # Add the likelihood of the observed data under the accepted␣
 ↪theta
                likelihood = (binom.pmf(k, n, proposed_theta) * beta.
 ↪pdf(proposed_theta, a, b)) / norm.pdf(proposed_theta, theta_samples[i - 1],␣
 ↪proposal_sd)
```

```
                ML_estimate += likelihood  # Update the marginal likelihood␣
↪estimate
                i += 1  # Move to the next iteration

    # Compute the average likelihood
    marginal_likelihood = ML_estimate / num_samples
    return marginal_likelihood

# Compute the marginal likelihood for each Beta prior and print the results
results = {}
for name, (a, b) in priors.items():
    marginal_likelihood = estimate_marginal_likelihood(a, b, k, n)
    results[name] = marginal_likelihood
    print(f"Marginal Likelihood for {name}: {marginal_likelihood:.2e}")
```

```
Marginal Likelihood for Beta_0.1_0.4: 4.08e-02
Marginal Likelihood for Beta_1_1: 8.98e-02
Marginal Likelihood for Beta_2_6: 2.14e-01
Marginal Likelihood for Beta_6_2: 1.01e-02
Marginal Likelihood for Beta_20_60: 4.10e-01
Marginal Likelihood for Beta_60_20: 1.16e-03
```

[ ]: