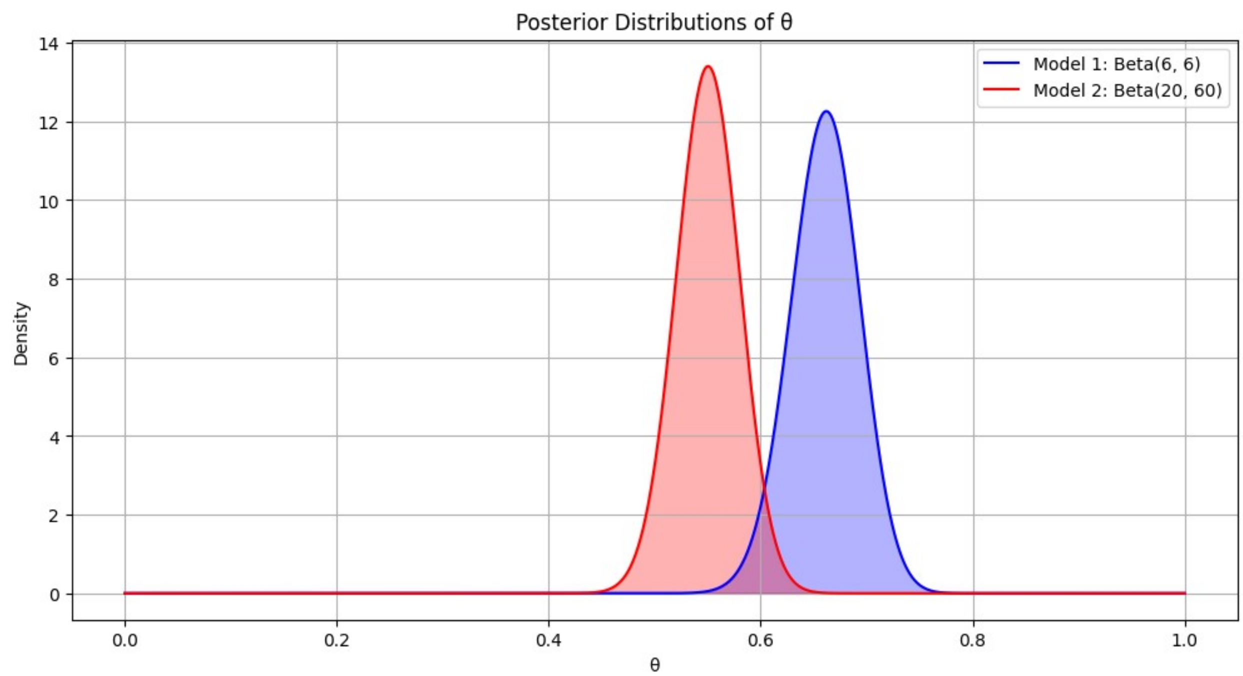
M.Vijay Kumar 220602 Assignment-5

1.1

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

# Given data
y = [10, 15, 15, 14, 14, 14, 13, 11, 12, 16]
n = 20

# Model 1: Prior Beta(6, 6)
alpha1 = 6
beta1 = 6
alpha_post1 = alpha1 + sum(y)
beta_post1 = beta1 + len(y) * n - sum(y)

# Model 2: Prior Beta(20, 60)
alpha2 = 20
beta2 = 60
alpha_post2 = alpha2 + sum(y)
beta_post2 = beta2 + len(y) * n - sum(y)

# Define theta range
theta = np.linspace(0, 1, 1000)

# Calculate posterior densities
posterior1 = beta.pdf(theta, alpha_post1, beta_post1)
posterior2 = beta.pdf(theta, alpha_post2, beta_post2)

# Plotting the posterior distributions
plt.figure(figsize=(12, 6))
plt.plot(theta, posterior1, label='Model 1: Beta(6, 6)', color='blue')
plt.plot(theta, posterior2, label='Model 2: Beta(20, 60)',
color='red')
plt.fill_between(theta, posterior1, alpha=0.3, color='blue')
plt.fill_between(theta, posterior2, alpha=0.3, color='red')
plt.title('Posterior Distributions of θ')
plt.xlabel('θ')
plt.ylabel('Density')
plt.legend()
plt.grid(True)
plt.show()
```



1.2

```
import numpy as np
from scipy.stats import binom

# Given data
y = np.array([10, 15, 15, 14, 14, 14, 13, 11, 12, 16])
n = 20

# Function to compute log pointwise predictive density (lppd) for a
given set of samples
def compute_lppd(samples_posterior, y, n):
    lppd = 0
    for yi in y:
        log_likelihoods = np.log(binom.pmf(yi, n, samples_posterior))
        lppd += np.mean(log_likelihoods)
    return lppd

# Example: Generating posterior samples (replace with actual samples
from posterior)
# Here we generate random samples for illustration purposes
np.random.seed(42)
samples_posterior1 = np.random.beta(6 + np.sum(y), 6 + len(y)*n -
np.sum(y), size=10000)
samples_posterior2 = np.random.beta(20 + np.sum(y), 60 + len(y)*n -
np.sum(y), size=10000)

# Compute lppd for Model 1
lppd_model1 = compute_lppd(samples_posterior1, y, n)
print("Log Pointwise Predictive Density (lppd) for Model 1:",
lppd_model1)
```

```
# Compute lppd for Model 2
lppd_model2 = compute_lppd(samples_posterior2, y, n)
print("Log Pointwise Predictive Density (lppd) for Model 2:",
lppd_model2)
```

```
Log Pointwise Predictive Density (lppd) for Model 1: -
20.701470894031353
Log Pointwise Predictive Density (lppd) for Model 2: -
26.52424266124716
```

1.3

```
lppd_model1 = -20.701470894031353
lppd_model2 = -26.52424266124716

in_sample_deviance_model1 = -2 * lppd_model1
in_sample_deviance_model2 = -2 * lppd_model2

print("In-sample deviance for Model 1:", in_sample_deviance_model1)
print("In-sample deviance for Model 2:", in_sample_deviance_model2)
```

```
In-sample deviance for Model 1: 41.402941788062705
In-sample deviance for Model 2: 53.04848532249432
```

We call it "in-sample deviance" because it quantifies how well a model fits the data it was trained on — the observed data y

y. It's derived from the log pointwise predictive density (lppd), which measures the average log likelihood of the observed data under the model's posterior predictive distribution. By computing the in-sample deviance, we assess how effectively the model captures and predicts the patterns in the data it has seen during training or fitting.

1.4 Model 1 has a lower in-sample deviance (41.40) compared to Model 2 (53.05), indicating that Model 1 is a better fit to the observed data. Therefore, based on in-sample deviance, Model 1 is preferred as it provides a better overall fit to the data compared to Model 2.

1.5

```
import numpy as np
from scipy.stats import binom

# Given data and new data points
y = np.array([10, 15, 15, 14, 14, 14, 13, 11, 12, 16])
y_new = np.array([5, 6, 10, 8, 9])
n = 20

# Function to compute log predictive density (lpd) for a given set of
samples
def compute_lpd(samples_posterior, y_new, n):
    lpd = np.zeros(len(y_new))
    for i, y_new_i in enumerate(y_new):
        log_likelihoods = np.log(binom.pmf(y_new_i, n,
samples_posterior))
```

```
        lpd[i] = np.mean(log_likelihoods)
    return lpd

# Function to compute lppd and out-of-sample deviance
def compute_lppd_and_deviance(samples_posterior, y_new, n):
    lpd = compute_lpd(samples_posterior, y_new, n)
    lppd = np.mean(lpd)
    out_of_sample_deviance = -2 * lppd
    return lppd, out_of_sample_deviance

# Example: Using previously computed posterior samples
samples_posterior1 = np.random.beta(6 + np.sum(y), 6 + len(y)*n -
np.sum(y), size=10000)
samples_posterior2 = np.random.beta(20 + np.sum(y), 60 + len(y)*n -
np.sum(y), size=10000)

# Compute lppd and out-of-sample deviance for Model 1
lppd_model1, deviance_model1 =
compute_lppd_and_deviance(samples_posterior1, y_new, n)
print("Model 1:")
print("Log Pointwise Predictive Density (lppd):", lppd_model1)
print("Out-of-sample deviance:", deviance_model1)

# Compute lppd and out-of-sample deviance for Model 2
lppd_model2, deviance_model2 =
compute_lppd_and_deviance(samples_posterior2, y_new, n)
print("\nModel 2:")
print("Log Pointwise Predictive Density (lppd):", lppd_model2)
print("Out-of-sample deviance:", deviance_model2)

Model 1:
Log Pointwise Predictive Density (lppd): -5.3917818067098064
Out-of-sample deviance: 10.783563613419613

Model 2:
Log Pointwise Predictive Density (lppd): -3.25948684802873
Out-of-sample deviance: 6.51897369605746
```

Therefore, based on the out-of-sample deviance criterion, Model 2 is preferred as it demonstrates better predictive accuracy for the new data compared to Model 1.

1.6

```
import numpy as np
from scipy.stats import binom

# Given data
y = np.array([10, 15, 15, 14, 14, 14, 13, 11, 12, 16])
n = 20

# Function to perform LOO-CV for a given set of samples
def loo_cv(samples_posterior, y, n):
```

```
    lppd = np.zeros(len(y))
    for i in range(len(y)):
        y_train = np.delete(y, i)  # leave out i-th data point
        sample_theta = np.random.beta(6 + np.sum(y_train), 6 +
(len(y_train)*n) - np.sum(y_train), size=10000)
        log_likelihoods = np.log(binom.pmf(y[i], n, sample_theta))
        lppd[i] = np.mean(log_likelihoods)
    loo_cv_score = np.sum(lppd)
    return lppd, loo_cv_score

# Perform LOO-CV for Model 1
lppd_model1, loo_cv_score_model1 = loo_cv(samples_posterior1, y, n)
print("Model 1:")
print("LOO-CV Log Pointwise Predictive Density (lppd):", lppd_model1)
print("LOO-CV Score:", loo_cv_score_model1)

# Perform LOO-CV for Model 2
lppd_model2, loo_cv_score_model2 = loo_cv(samples_posterior2, y, n)
print("\nModel 2:")
print("LOO-CV Log Pointwise Predictive Density (lppd):", lppd_model2)
print("LOO-CV Score:", loo_cv_score_model2)
```

```
Model 1:
LOO-CV Log Pointwise Predictive Density (lppd): [-3.1340338  -
2.1019014  -2.10203961 -1.78653603 -1.7876728  -1.79180804
 -1.75096048 -2.43402422 -1.96924134 -2.70128805]
LOO-CV Score: -21.55950576306247

Model 2:
LOO-CV Log Pointwise Predictive Density (lppd): [-3.14270703 -
2.09922334 -2.09617666 -1.78909195 -1.78825107 -1.78802099
 -1.74788525 -2.43298668 -1.96896538 -2.70017683]
LOO-CV Score: -21.553485183095876
```

2.1

```
import math

# Function to calculate marginal likelihood for Binomial model
def ML_binomial(k, n, a, b):
    binom_coeff = math.factorial(n) / (math.factorial(k) *
math.factorial(n - k))
    numerator = math.factorial(int(k + a - 1)) * math.factorial(int(n
- k + b - 1))
    denominator = math.factorial(int(n + a + b - 1))
    ml = binom_coeff * (numerator / denominator)
    return ml

# Given values
k = 2
n = 10
```

```
# Prior distributions on θ
priors = ["Beta(0.1,0.4)", "Beta(1,1)", "Beta(2,6)", "Beta(6,2)",
"Beta(20,60)", "Beta(60,20)"]
a_values = [0.1, 1, 2, 6, 20, 60]
b_values = [0.4, 1, 6, 2, 60, 20]

# Calculate marginal likelihoods for each prior
marginal_likelihoods = []
for a, b in zip(a_values, b_values):
    ml = ML_binomial(k, n, a, b)
    marginal_likelihoods.append(ml)

# Display results
results = {"Prior": priors, "Marginal Likelihood":
marginal_likelihoods}
for prior, ml in zip(priors, marginal_likelihoods):
    print(f"{prior}: {ml:.10f}")
```

```
Beta(0.1,0.4): 0.6250000000
Beta(1,1): 0.0909090909
Beta(2,6): 0.0047268908
Beta(6,2): 0.0002313863
Beta(20,60): 0.0000000000
Beta(60,20): 0.0000000000
```

2.2

```
import numpy as np
import scipy.stats as stats

# Given data
k = 2
n = 10

# Prior distributions on θ
priors = ["Beta(0.1,0.4)", "Beta(1,1)", "Beta(2,6)", "Beta(6,2)",
"Beta(20,60)", "Beta(60,20)"]
a_values = [0.1, 1, 2, 6, 20, 60]
b_values = [0.4, 1, 6, 2, 60, 20]

# Number of Monte Carlo samples
num_samples = 100000

# Function to estimate marginal likelihood using Monte Carlo
integration
def estimate_marginal_likelihood(k, n, a, b, num_samples):
    theta_samples = np.random.beta(a, b, num_samples)
    likelihoods = stats.binom.pmf(k, n, theta_samples)
    prior_pdf = stats.beta.pdf(theta_samples, a, b)
    marginal_likelihood_estimate = np.mean(likelihoods * prior_pdf)
    return marginal_likelihood_estimate
```

```python
# Calculate marginal likelihood estimates for each prior
marginal_likelihood_estimates = []
for a, b in zip(a_values, b_values):
    ml_estimate = estimate_marginal_likelihood(k, n, a, b,
num_samples)
    marginal_likelihood_estimates.append(ml_estimate)

# Display results
results = {"Prior": priors, "Monte Carlo Estimate":
marginal_likelihood_estimates}
for prior, ml_estimate in zip(priors, marginal_likelihood_estimates):
    print(f"{prior}: {ml_estimate:.10f}")
```

```
Beta(0.1,0.4): 0.0244983752
Beta(1,1): 0.0907509914
Beta(2,6): 0.4720675150
Beta(6,2): 0.0053432046
Beta(20,60): 1.6271040873
Beta(60,20): 0.0030813316
```