02477 Practice exam problems (not a full exam set)

Part 1

Consider the following regression model

$$y(x) = f(x) + e = w_0 + w_1 x^2 + w_2 \sin x + w_3 x + e, \tag{1}$$

such that $y_n = f(x_n) + e_n$, where $x_n, y_n \in \mathbb{R}$ are input and targets, respectively. The additive noise $e_n \in \mathbb{R}$ is assumed to i.i.d from a zero-mean Gaussian distribution, i.e. $e_n \sim \mathcal{N}(0, \beta^{-1})$ for $\beta > 0$.

Let $\mathbf{x} = [2.29, -1.8, -0.06, 3.72, 2.6, -5.93, -0.15]$ and $\mathbf{y} = [3.17, -4.53, -0.78, 3.15, 4.76, -1.96, -1.32]$ denote the vector of inputs and targets, respectively, for a dataset with $N = 5$ observations.

Let $\mathbf{w} = [w_0, w_1, w_2, w_3]^T \in \mathbb{R}^4$ denote the parameter vector.

Question 1.1: Compute and report a maximum likelihood estimate for $\mathbf{w}$ and β.

Solution We recognize the model in eq. (1) as a linear model wrt. the parameters w_i, and hence, we can write this in matrix notation as follows

$$y(x) = \mathbf{w}^T \phi(\boldsymbol{x}),$$

where $\phi(x) = \begin{bmatrix} 1 & x^2 & \sin x & x \end{bmatrix}^T$. Moreover, since the noise is i.i.d. Gaussian, this is a linear model with Gaussian likelihood. Therefore, the maximum likelihood solution is given by the solution to the normal equations (i.e. eq. (11.14) in Murphy1 or slide 15 from week 3):

$$\hat{\boldsymbol{w}}_{\text{MLE}} = \left(\boldsymbol{\Phi}^T \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^T \boldsymbol{y} \approx \begin{bmatrix} -0.73 \\ 0.11 \\ 2.36 \\ 1.01 \end{bmatrix}$$

where $\boldsymbol{\Phi} \in \mathbb{R}^{5\times 4}$ is the design matrix.
Similarly, the maximum likelihood estimate for β can be computed via eq. (11.49) in Murphy1:

$$\beta_{\text{MLE}}^{-1} = \frac{1}{N} \sum_{n=1}^{N} (y_n - \hat{\boldsymbol{w}}_{\text{MLE}}^T \phi(x_n))^2 \approx 0.21 \quad \Rightarrow \quad \beta_{\text{MLE}} \approx 4.84$$

The code to obtain the solution is given by

```
# implement design matrix
def design_matrix(x):
    return np.column_stack([np.ones(len(x)), x**2, np.sin(x), x])

# data
x = np.array([ 2.29, -1.8,  -0.06,  3.72,  2.6,  -5.93, -0.15])
y = np.array([ 3.17, -4.53, -0.78,  3.15,  4.76, -1.96, -1.32])

# construct design matrix
Phi = design_matrix(x)

# compute MLE
w_MLE = np.linalg.solve(Phi.T@Phi, Phi.T@y)

# print solution
```

```
print('w_MLE', np.array2string(w_MLE, precision=2))

# MLE for beta
yhat = Phi@w_MLE
sigma2_MLE = np.mean((yhat - y)**2)
beta_MLE = 1./sigma2_MLE
print(f'sigma2_MLE: {sigma2_MLE:3.2f}')
print(f'beta_MLE: {beta_MLE:3.2f}')
```

End of solution

Question 1.2: Compute the posterior predictive distribution $p(y^*|\boldsymbol{y}, x^* = 1)$, where $y^* = y(x^*)$ using a plug-in approximation based on the maximum likelihood estimators for $\boldsymbol{w}$ and β. Report the mean, standard deviation and a 95% credibility interval for y^*

Solution

The posterior predictive distribution with the plugin approximation is given by (from eq. (4.197) in Murphy1)

$$p(y^*|\boldsymbol{y}, x^* = 1) \approx \mathcal{N}(y^*|\hat{\mathbf{w}}_{\text{MLE}}^T \phi(x^*), \hat{\beta}_{\text{MLE}}^{-1}) \approx \mathcal{N}(y^*|2.37, 0.21)$$

Hence, the mean is 2.37, the standard deviation is 0.45, and the 95% credibility interval is $[1.48, 3.26]$. The results can be obtained using the following code

```
from scipy.stats import norm

# compute feature space
xstar = 1
Phi_star = design_matrix(np.array([xstar]))

# compute posterior predictive
ystar_mean = np.dot(w_MLE, Phi_star.ravel())
ystar_std = np.sqrt(1/beta_MLE)
ystar_lower, ystar_upper = norm.interval(0.95, loc=ystar_mean, scale=ystar_std)

# print
print(f'Plug in approximation')
print(f'Mean of y^*:\t{ystar_mean:4.3f}')
print(f'Std dev of y^*:\t{ystar_std:4.3f}')
print(f'95\% interval: [{ystar_lower:4.3f}, {ystar_upper:4.3f}]')
```

End of solution

Next, we impose i.i.d Gaussian priors on all regression coefficients $w_j \sim \mathcal{N}(0, \alpha^{-1})$ for $j = 0, 1, 2, 3$ and assume $\alpha = 1$ and $\beta = \frac{1}{2}$.

Question 1.3: Compute and report the posterior mean and marginal posterior standard deviation for each regression coefficient in w.

Solution

The model is Bayesian linear regression with Gaussian likelihood and Gaussian prior, and hence, the posterior distribution is available in closed-form via the equations eq. (11.120)-(11.122) in Murphy1 or slide 25 from week 3:

$$
\boldsymbol{m} = \beta \mathbf{S}\boldsymbol{\Phi}^T \boldsymbol{y} \approx \begin{bmatrix} -0.56 \\ 0.11 \\ 1.26 \\ 0.99 \end{bmatrix}
$$

$$
\boldsymbol{S} = \left(\alpha \boldsymbol{I} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi}\right)^{-1} \approx \begin{bmatrix} 0.37 & -0.02 & 0.06 & -0.04 \\ -0.02 & 0. & -0.01 & 0.01 \\ 0.06 & -0.01 & 0.49 & -0.02 \\ -0.04 & 0.01 & -0.02 & 0.04 \end{bmatrix}
$$

Hence, the posterior mean for $\boldsymbol{w}$ is given by $\boldsymbol{m}$ above and the posterior standard deviations are given by the square root of the diagonal of $\boldsymbol{S}$: $[0.61, 0.05, 0.7, 0.21]$

The solutions can be computed via the equation above directly or using the code from the course:

```
alpha = 1
beta = 1/2
model = BayesianLinearRegression(Phi, y[:, None], alpha=alpha, beta=beta)
print(f'Post. mean:\t{np.array2string(model.m.ravel(), precision=2,)}')
print(f'Post. cov:\n{np.array2string(model.S, precision=2,)}')
print(f'Post std dev:\t{np.array2string(np.sqrt(np.diag(model.S)), precision=2)}')
```

End of solution

Question 1.4: Compute the analytical posterior predictive density $p(y^*|\boldsymbol{y}, x^*)$ for $x^* = 1$.

Solution

For Bayesian linear regression with the posterior predictive distribution is given by

$$
p(y^*|\boldsymbol{y}, \boldsymbol{x}^* = 1) = \mathcal{N}(y^*|\boldsymbol{m}^T \phi(x^*), \phi(x^*)^T \boldsymbol{S} \phi(x^*) + \beta^{-1}) \approx \mathcal{N}(y^*|1.60, 2.70)
$$

See slide 33 from week 3 or eq. (11.124) in Murphy1.

Again, this can be computed via the equations above directly or the code from the course:

```
ystar_mu, ystar_var = model.predict_y(Phi_star)
print(f'Mean of y^*:\t\t{ystar_mu[0]:3.2f}')
print(f'Variance dev of y^*:\t{ystar_var[0]:3.2f}')
```

End of solution

Question 1.5: State the analytical expression for the marginal likelihood $p(\boldsymbol{y}|\alpha, \beta)$ and compute the value of $\log p(\boldsymbol{y}|\alpha = 1, \beta = \frac{1}{2})$.

Solution

For Bayesian linear regression, the marginal likelihood is given by (see slide 25 from week 3 or eq. (3.38) in Murphy1)

$$
p(\mathbf{y}|\alpha, \beta) = \mathcal{N}(\boldsymbol{y}|\mathbf{0}, \beta^{-1}\mathbf{I} + \alpha^{-1}\boldsymbol{\Phi}\boldsymbol{\Phi}^T)
$$

For $\alpha = 1, \beta = \frac{1}{2}$, this evaluates to

$$
\log p(\mathbf{y}|\alpha = 1, \beta = \frac{1}{2}) = \log \mathcal{N}(\boldsymbol{y}|\mathbf{0}, 2\mathbf{I} + \boldsymbol{\Phi}\boldsymbol{\Phi}^T) \approx -17.24
$$

Again, this can be computed via the equation above or via the code from the course:

```
alpha = 1
beta = 1/2
logZ = model.compute_marginal_likelihood(alpha=alpha, beta=beta)
print(f'Log marginal likelihood for alpha={alpha:3.2f} and beta={beta:3.2f} is log Z = {logZ:3.2f}')
```

End of solution

Consider now the following hyperprior distribution for α and β:

$$p(\alpha, \beta) = \text{Gamma}(\alpha|1, 1)\text{Gamma}(\beta|1, 1) \tag{2}$$

Question 1.6: Use the Metropolis-Hastings algorithm to generate posterior samples from the distribution $p(\alpha, \beta|\boldsymbol{y})$. Run 2 chains for 2000 iterations each. Initialize the first chain using $\alpha = 1$ and $\beta = 1$ and the second chain using $\alpha = 10$ and $\beta = 10$. Choose an appropriate proposal variance and justify your choice. Plot the trace of both parameters.

Solution In order to use the Metropolis-Hastings algorithm to generate samples from the posterior, the first step is to implement the target distribution: $p(\boldsymbol{y}, \alpha, \beta) = p(\boldsymbol{y}|\alpha, \beta)p(\alpha)p(\beta)$. For hyperparameters $(1, 1)$, the logarithm of gamma priors simplifies to

$$\log \text{Gamma}(\alpha|1, 1) = \log \left[\frac{1^1}{\Gamma(1)}\alpha^{1-1}\exp(-\alpha)\right] = -\alpha + \text{constant} \tag{3}$$

and similarly for $\log \text{Gamma}(\beta|1, 1)$. The term $p(\boldsymbol{y}|\alpha, \beta)$ is the marginal likelihood of the model and can be computed as in Question 1.5. This can be implemented as follows

```
# log prior for alpha and beta (we could also simply use gamma.logpdf(x, a=1) instead)
log_gamma_prior = lambda alpha:  -alpha

# prepare model
model = BayesianLinearRegression(Phi, y[:, None], alpha=alpha, beta=beta)

# define log target
def log_target(theta):

    # set hyperparameters
    alpha, beta = theta
    model.alpha = alpha
    model.beta = beta

    # enforce positivity (necessary if you make your own implementation of the gamme prior)
    if alpha <= 0 or beta <= 0:
        return -np.Inf

    # evaluate log prior and log like and return
    log_prior = log_gamma_prior(alpha) + log_gamma_prior(beta)
    log_lik = model.compute_marginal_likelihood(alpha, beta)
    return log_prior + log_lik
```

and then run the sampler:

```
# run two chains
samples1 = metropolis(log_target=log_target, num_params=2, tau=1, num_iter=2000,
theta_init=[1, 1], seed=123)
samples2 = metropolis(log_target=log_target, num_params=2, tau=1, num_iter=2000,
```

```
theta_init=[10, 10], seed=456)

# stack the results and throw the first 500 samples away as warm-up
samples = np.stack((samples1, samples2))[:, 500:, :]

# plot traces
fig, ax = plt.subplots(1, 2, figsize=(20, 6))
ax[0].plot(samples[:, :, 0].T, label='Chain')
ax[0].set(xlabel='Iterations', ylabel='$\\alpha$', title='Trace for $\\alpha$')
ax[1].plot(samples[:, :, 1].T, label='Chain')
ax[1].set(xlabel='Iterations', ylabel='$\\beta$', title='Trace for $\\beta$')
ax[0].legend()
ax[1].legend()
fig.savefig('trace_ab.png', bbox_inches='tight')
```

which yields the trace plot

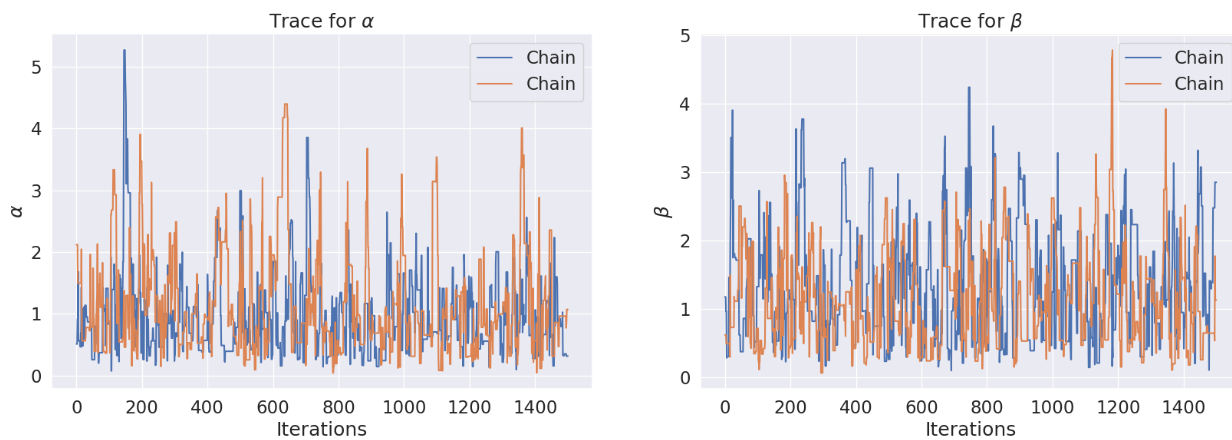

Figure 1: Trace plots for α and β

Using $\tau = 1$ for the proposal distribution and using 500 samples for warm-up yields the following $\hat{R}$ statistics of 1.05 and 1.03, respectively, and hence, mixing does not seem to be problem.

End of solution

Question 1.7: Use the samples to compute a Monte Carlo estimate for the posterior mean of α and β and report the MCSE for both estimates.

Solution

The MC estimator for the posterior mean of α (and similar for β) is given by

$$\mathbb{E}\left[\alpha|\boldsymbol{y}\right] \approx \frac{1}{S}\sum_{i=1}^{S} \alpha^{(i)} \approx 1.05, \tag{4}$$

$$\mathbb{E}\left[\beta|\boldsymbol{y}\right] \approx \frac{1}{S}\sum_{i=1}^{S} \beta^{(i)} \approx 1.22, \tag{5}$$

where $\alpha^{(i)}$ denotes the i'th sample from the MCMC sampler (after combining the chains). To compute the MCSE, we first compute the effective sample S_{eff} for each parameter and then use the equation (from slide

26 week 9)

$$MCSE_\alpha = \frac{1}{\sqrt{S_{\text{eff}}}}\text{stddev}(\alpha^{(i)}) \approx 0.06, \tag{6}$$

$$MCSE_\beta = \frac{1}{\sqrt{S_{\text{eff}}}}\text{stddev}(\beta^{(i)}) \approx 0.05, \tag{7}$$

where $\text{stddev}(\alpha^{(i)}) \approx 0.76$ and $\text{stddev}(\beta^{(i)}) \approx 0.78$ is the standard deviation of the posterior samples for α and β, respectively.

The code for obtaining these results is below:

```
# combine effective sample sizes
Seff = compute_effective_sample_size(samples)

# combine chains
samples_flat = samples.reshape((-1, 2))
# compute mean and standard deviation
m = np.mean(samples_flat, axis=0)
s = np.std(samples_flat, axis=0)
# compute MCSE using effective samples size
MCSE = s/np.sqrt(Seff)

# print
print(f'Mean: {np.array2string(m, precision=2)}')
print(f'Std dev: {np.array2string(s, precision=2)}')
print(f'MCSE: {np.array2string(MCSE, precision=2)}')
```

End of solution

Part 2

Suppose the outcome of $N = 31$ independent Bernoulli trials generated $y = 7$ successes. Let $\theta \in [0, 1]$ denote the probability of success. Assume a Binomial likelihood, i.e. $p(y|\theta) = \text{Bin}(y|N, \theta)$ with the following prior distribution for θ:

$$p(\theta) = \frac{3}{7}\text{Beta}(\theta|2, 10) + \frac{4}{7}\text{Beta}(\theta|10, 2) \tag{8}$$

Question 2.1: Compute the prior probability of the event $\theta > \frac{1}{2}$.

Solution

We can compute this probability in several ways: 1) via the CDF, 2) via sampling or 3) via numerical integration. The most straight-forward is 1)

$$\begin{aligned}
p(\theta > 0.5) &= \int_{0.5}^{1} p(\theta)\mathrm{d}\theta && (9)\\
&= \int_{0.5}^{1} \frac{3}{7}Beta(\theta|2, 10) + \frac{4}{7}Beta(\theta|10, 2)\mathrm{d}\theta && (10)\\
&= \frac{3}{7}\int_{0.5}^{1} Beta(\theta|2, 10)\mathrm{d}\theta + \frac{4}{7}\int_{0.5}^{1} Beta(\theta|10, 2)\mathrm{d}\theta && (11)\\
&= \frac{3}{7}\left[1 - \underbrace{\int_{0}^{0.5} Beta(\theta|2, 10)\mathrm{d}\theta}_{p_1}\right] + \frac{4}{7}\left[1 - \underbrace{\int_{0}^{0.5} Beta(\theta|10, 2)\mathrm{d}\theta}_{p_2}\right] && (12)
\end{aligned}$$

Two the probabilities above can now be easily computed the CDFs of the respective Beta-distributions

$$p(\theta > 0.5) \approx \frac{3}{7}0.01 + \frac{4}{7}0.99 \approx 0.57$$

Code for evaluating the probabilities

```
from scipy.stats import beta as beta_dist
# compute CDF values
p1 = beta_dist.cdf(0.5, 2, 10)
p2 = beta_dist.cdf(0.5, 10, 2)
print(f'p1 = {p1:3.2f}, p2={p2:3.2f}')
# combine and print
p = (3/7)*(1-p1) + (4/7)*(1-p2)
print(f'p(theta > 0 .5) = {p:3.2f}')
```

We can also obtain the same via sampling

```
np.random.seed(0)
N = int(1e6) # sample size
# sample from mixture
z = np.random.binomial(1, p=4/7, size=N)
theta1 = beta_dist.rvs(2, 10,size=N)
theta2 = beta_dist.rvs(10, 2,size=N)
theta = (1-z)*theta1  +theta2*z
# estimate using MCMC and print result
```

```
prob = np.mean(theta > 0.5)
print(f'p(theta > 0 .5) = {prob:3.2f}')
```

Finally, we can also solve it via direct integration in Python (or matlab, Maple or your favourite tool)

```
from scipy.integrate import quad
# define density of distribution of integral
p = lambda theta: 3/7*beta_dist.pdf(theta, 2, 10) + 4/7*beta_dist.pdf(theta, 10, 2)
# integrate from 0.5 to 1
prob2 = quad(p, 0.5, 1)[0]
print(f'p(theta > 0.5) = {prob2:3.2f}')
```

End of solution

Question 2.2: Compute the analytical marginal likelihood $p(y)$ and evaluate $p(y = 7)$.

Solution

We can obtain the marginal likelihood via the sum rule and use linearity of integral to simplify the expression

$$
\begin{aligned}
p(y) &= \int p(y|\theta)p(\theta)\mathrm{d}\theta \\
&= \int \mathrm{Bin}(y|N,\theta)\left[\frac{3}{7}\mathrm{Beta}(\theta|2,10) + \frac{4}{7}\mathrm{Beta}(\theta|10,2)\right]\mathrm{d}\theta \\
&= \int \mathrm{Bin}(y|N,\theta)\frac{3}{7}\mathrm{Beta}(\theta|2,10)\mathrm{d}\theta + \int \mathrm{Bin}(y|N,\theta)\frac{4}{7}\mathrm{Beta}(\theta|10,2)\mathrm{d}\theta \\
&= \frac{3}{7}\int \mathrm{Bin}(y|N,\theta)\mathrm{Beta}(\theta|2,10)\mathrm{d}\theta + \frac{4}{7}\int \mathrm{Bin}(y|N,\theta)\mathrm{Beta}(\theta|10,2)\mathrm{d}\theta
\end{aligned}
$$

We now recognize the two integral as the marginal likelihood of the beta-binomial model. Hence, we can re-use the following result derived in the exercise from Week 1 (or using eq. (4.135) in Murphy1):

$$
p(y) = \int \mathrm{Bin}(y|N,\theta)\mathrm{Beta}(\theta|\alpha_0,\beta_0)\mathrm{d}\theta = \binom{N}{y}\frac{B(y+\alpha_0, N-y+\beta_0)}{B(\alpha_0,\beta_0)}
$$

Hence,

$$
\int \mathrm{Bin}(y|N,\theta)\mathrm{Beta}(\theta|2,10)\mathrm{d}\theta = \binom{N}{y}\frac{B(y+2, N-y+10)}{B(2,10)}
$$

and

$$
\int \mathrm{Bin}(y|N,\theta)\mathrm{Beta}(\theta|10,2)\mathrm{d}\theta = \binom{N}{y}\frac{B(y+10, N-y+2)}{B(10,2)}
$$

yielding

$$
p(y) = \frac{3}{7}\binom{N}{y}\frac{B(y+2, N-y+10)}{B(2,10)} + \frac{4}{7}\binom{N}{y}\frac{B(y+10, N-y+2)}{B(10,2)} \tag{13}
$$

Finally, evaluating it for $N = 31$ and $y = 7$ yields $p(y) \approx 0.03$

```
from scipy.special import beta as beta_fun
from scipy.special import binom

N = 31
```

```
    y = 7

    py = 3/7 * binom(N, y)* beta_fun(y+2, N -y + 10)/beta_fun(2, 10) + 4/7*binom(N,y)*beta_fun(y+10, N-y
    print(f'p(y) = {py:3.2f}')
```

End of solution

Part 3

Consider the generalized linear model with a Poisson likelihood

$$y_n|\boldsymbol{w}, x_n \sim \text{Poisson}(\lambda_n) \tag{14}$$
$$\lambda_n = e^{w_0+w_1x_n} \tag{15}$$
$$\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \alpha^{-1}\boldsymbol{I}), \tag{16}$$

where $\boldsymbol{w} = [w_0, w_1]$ for the following dataset $\mathcal{D} = \{x_n, y_n\}$, for $N = 5$, where $\boldsymbol{x} = [1, 2, 4, 8, 10]$ and $\boldsymbol{y} = [5, 4, 1, 0, 0]$. Assume $\alpha = \frac{1}{4}$.

Question 3.1: Plot the contours of the prior distribution, the log likelihood and the posterior for the ranges $w_0 \in [-3.5, 3.5]$ and $w_1 \in [-3.5, 3.5]$.

Solution

We need to implement a function for evaluating the log prior, the log likelihood and the log joint. We can find inspiration and re-use the code from week 2 ('Grid2D'-class for plotting) and week 8 (Poisson regression) to produce the requested plots:

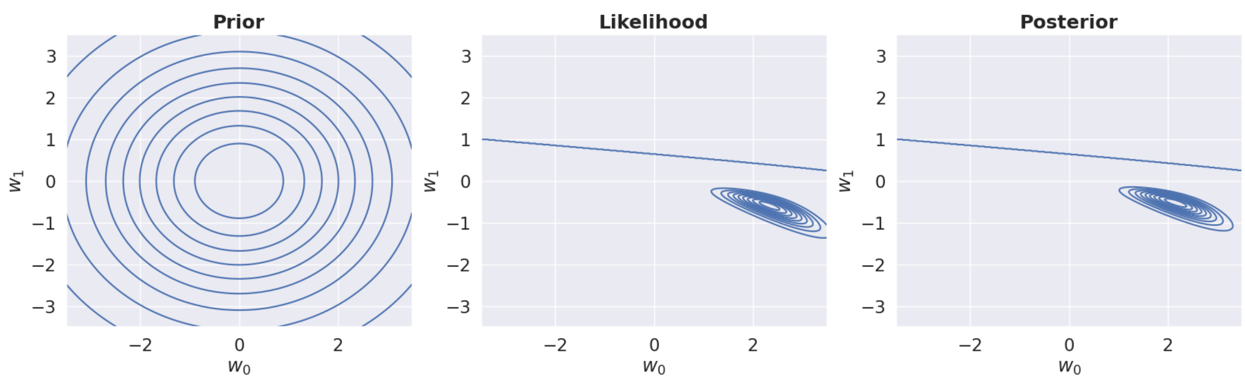

Figure 2: Plot of prior, likelihood and posterior for the Poisson regression model

```
# prep relevant distributions
from scipy.stats import poisson
log_npdf = lambda x, m, v: -0.5*np.log(2*np.pi*v) - 0.5*(x-m)**2/v

class Grid2D(object):
    """ helper class for evaluating the function func on the grid defined by (param1, param2)"""

    def __init__(self, param1s, param2s, func, name="Grid2D"):
        self.param1s = param1s
        self.param2s = param2s
        self.grid_size = (len(self.param1s), len(self.param2s))
```

```
        self.param1_grid, self.param2_grid = np.meshgrid(param1s, param2s, indexing='ij')
        self.func = func
        self.name = name

        # evaluate function on each grid point
        self.values = self.func(self.param1_grid[:, :, None], self.param2_grid[:, :, None]).squeeze()

    def plot_contours(self, ax, color='b', num_contours=10, f=lambda x: x, alpha=1.0, title=None):
        ax.contour(self.param1s, self.param2s, f(self.values).T, num_contours, colors=color, alpha=alpha
        ax.set(xlabel='$w_0$', ylabel='$w_1$')
        ax.set_title(self.name, fontweight='bold')

    @property
    def argmax(self):
        idx = np.argmax(self.values)
        param1_idx, param2_idx = np.unravel_index(idx, self.grid_size)
        return self.param1s[param1_idx], self.param2s[param2_idx]


# implement log likelihood
def log_lik(w0, w1):
    l = poisson.logpmf(y, np.exp(w0 + w1*x)).sum(axis=2)
    return l

# implement log pirior
def log_prior(w0, w1):
    alpha = 1/4
    return (log_npdf(w0, 0, 1/alpha)  + log_npdf(w1, 0,1/alpha)).sum(axis=2)

# log posterior
def log_posterior(w0, w1):
    return log_lik(w0, w1) + log_prior(w0, w1)

# define grid
w0s = np.linspace(-3.5, 3.5, 503)
w1s = np.linspace(-3.5, 3.5, 501)

# evaluate on grid
grid_prior = Grid2D(w0s, w1s, log_prior, name='Log posterior')
grid_loglik = Grid2D(w0s, w1s, log_lik, name='Log likelihood')
grid_posterior  = Grid2D(w0s, w1s, log_posterior, name='Log posterior')

# plot
fig, ax = plt.subplots(1, 3, figsize=(20, 5))
grid_prior.plot_contours(ax[0], f=np.exp)
grid_loglik.plot_contours(ax[1], f=np.exp)
grid_posterior.plot_contours(ax[2], f=np.exp)
```

End of solution

Question 3.2: Write the logarithm of the joint distribution $p(\boldsymbol{y}, \boldsymbol{w})$ and absorb all terms that are constant wrt. $\boldsymbol{w}$ into a constant $K \in \mathbb{R}$.

Solution

First, we write up the joint distribution via the product rue

$$p(\boldsymbol{y},\boldsymbol{w}) = \prod_{n=1}^{N} p(y_n|\boldsymbol{w},x_n)p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{0},\alpha^{-1}\boldsymbol{I}) \prod_{n=1}^{N} \frac{\lambda_n^{y_n}e^{-\lambda_n}}{y_n!} \tag{17}$$

.. then we take the logarithm and absorb all terms, which are constant wrt. $\boldsymbol{w}$ into an additive constant K. We note that $\lambda_n = e^{\boldsymbol{w}^T\boldsymbol{x}_n}$ depends on $\boldsymbol{w}$:

$$\begin{aligned}
\log p(\boldsymbol{y},\boldsymbol{w}) &= \sum_{n=1}^{N} \log p(y_n|\boldsymbol{w},x_n) + \log p(\boldsymbol{w}) && (18)\\
&= -\frac{1}{2}\log(2\pi) - \frac{1}{2}\alpha\boldsymbol{w}^T\boldsymbol{w} + \sum_{n=1}^{N}\log\frac{\lambda_n^{y_n}e^{-\lambda_n}}{y_n!} && (19)\\
&= -\frac{1}{2}\log(2\pi) - \frac{1}{2}\alpha\boldsymbol{w}^T\boldsymbol{w} + \sum_{n=1}^{N}\left[y_n\log\lambda_n - \lambda_n + \log\frac{1}{y_n!}\right] && (20)\\
&= -\frac{1}{2}\alpha\boldsymbol{w}^T\boldsymbol{w} + \sum_{n=1}^{N}\left[y_n\log\lambda_n - \lambda_n\right] + K && (21)\\
&= -\frac{1}{2}\alpha\boldsymbol{w}^T\boldsymbol{w} + \sum_{n=1}^{N} y_n\log\lambda_n - \sum_{n=1}^{N}\lambda_n + K && (22)
\end{aligned}$$

End of solution

Next, assume $\boldsymbol{w}_{MAP} = \left[2.1575, -0.5201\right]^T$ is a MAP estimator for $\boldsymbol{w}$.

Question 3.3: Compute the Hessian of $\log p(\boldsymbol{y},\boldsymbol{w})$ with respect to $\boldsymbol{w}$ and evaluate it at the mode of $p(\boldsymbol{w}|\boldsymbol{y})$.

Solution

To compute the Hessian, we first note that for $\lambda_n = e^{\boldsymbol{w}^T\boldsymbol{x}_n}$ for $\boldsymbol{x_n} = \begin{bmatrix}1 & x_n\end{bmatrix}$, we have

$$\begin{aligned}
\nabla\lambda_n &= \nabla_{\boldsymbol{w}} e^{\boldsymbol{w}^T\boldsymbol{x}_n} = e^{\boldsymbol{w}^T\boldsymbol{x}_n}\boldsymbol{x}_n = \lambda_n\boldsymbol{x}_n && \text{(chain rule)}\\
\nabla\log\lambda_n &= \frac{1}{\lambda_n}\nabla_{\boldsymbol{w}}\lambda_n = \frac{1}{\lambda_n}\lambda_n\boldsymbol{x}_n = \boldsymbol{x}_n && \text{(chain rule)}
\end{aligned}$$

and hence, the gradient becomes

$$\begin{aligned}
\nabla_{\boldsymbol{w}}\log p(\boldsymbol{y},\boldsymbol{w}) &= -\frac{1}{2}\alpha\nabla_{\boldsymbol{w}}\boldsymbol{w}^T\boldsymbol{w} + \sum_{n=1}^{N} y_n\nabla_{\boldsymbol{w}}\log\lambda_n - \sum_{n=1}^{N}\nabla_{\boldsymbol{w}}\lambda_n && (23)\\
&= -\frac{1}{2}\alpha 2\boldsymbol{w} + \sum_{n=1}^{N} y_n\boldsymbol{x}_n - \sum_{n=1}^{N}\lambda_n\boldsymbol{x}_n && (24)\\
&= -\alpha\boldsymbol{w} + \sum_{n=1}^{N}\boldsymbol{x}_n(y_n - \lambda_n) && (25)
\end{aligned}$$

Then we can compute the Hessian as

$$\mathcal{H}(\boldsymbol{w})_{ij} = \frac{\partial^2}{\partial w_i \partial w_j} \log p(\boldsymbol{y}, \boldsymbol{w}) = \frac{\partial}{\partial w_i} \left[-\alpha \boldsymbol{w} + \sum_{n=1}^{N} \boldsymbol{x}_n (y_n - \lambda_n) + K \right]_j \tag{26}$$

$$= -\alpha \mathbb{I}\left[i = j\right] + \sum_{n=1}^{N} \boldsymbol{x}_{n,j} (y_n - \frac{\partial}{\partial w_i} \lambda_n) \tag{27}$$

$$= -\alpha \mathbb{I}\left[i = j\right] - \sum_{n=1}^{N} \lambda_n \boldsymbol{x}_{n,j} \boldsymbol{x}_{n,i}, \tag{28}$$

where $\mathbb{I}\left[\cdot\right]$ is an indicator function. Evaluating the Hessian at the mode yields:

$$\mathbf{H} = \begin{bmatrix} -9.71 & -17.13 \\ -17.13 & -48.3 \end{bmatrix}$$

This can be evaluated using the code

```
H = -alpha*np.identity(2)
lambda_n = lambda xn, w0, w1: np.exp(w0 + w1*xn)
for i in range(2):
    for j in range(2):
        for n in range(len(X)):
            H[i,j] +=  - lambda_n(x[n], w_MAP[0], w_MAP[1])*X[n,j]*X[n,i]

print(np.round(H, 2))
```

This can also be implemented vectorized

```
lambda_n_vector = lambda_n(x, w_MAP[0], w_MAP[1])
X2 =  np.sqrt(lambda_n_vector)[:, None]*X
H2 = -alpha*np.identity(2)- X2.T@X2
```

but the solution based on for loops would be just as fine, so pick the one you are most comfortable with.

End of solution

If you did not answer the previous question, assume the Hessian at the mode is

$$\mathbf{H} = \begin{bmatrix} -9 & -17 \\ -17 & -48 \end{bmatrix} \tag{29}$$

Question 3.4: Construct a Laplace approximation of $p(\boldsymbol{w}|\boldsymbol{y})$.

Solution

The Laplace approximation is given below (slide 35 week 4, or eq. (4.212) in Murphy1)

$$p(\boldsymbol{w}|\boldsymbol{y}) \approx \mathcal{N}(\boldsymbol{w}|\boldsymbol{w}_{\text{MAP}}, \boldsymbol{S}),$$

where $\boldsymbol{S} = -\mathbb{H}^{-1} = \begin{bmatrix} 0.34 & -0.12 \\ -0.12 & 0.06 \end{bmatrix}$.

Note that there has been a lot of confusion about about the sign when computing the covariance matrix, but as a sanity check, we can see that marginal variances are all non-negative in $\mathbf{S}$, so the sign is correct.

As another sanity check, we can see that the variance is large for w_0 compared to w_1 with slight negative correlation, and this observations matches the plots above.

End of solution

Question 3.5: Compute the mean and variance of the posterior predictive probability $p(y^*|\boldsymbol{y}, x^* = 0)$, where $y^* = y(x^*)$ via the Laplace approximation and Monte Carlo sampling. Use $S = 1000$ Monte Carlo samples.

Solution

To solve this, we first generate $S = 1000$ samples from the approximate posterior, $\boldsymbol{w}_{(i)} \sim \mathcal{N}(\boldsymbol{w}|\boldsymbol{w}_{\text{MAP}}, \boldsymbol{S})$, and then for each sample $\boldsymbol{w}_{(i)}$ we compute

$$\lambda^*_{(i)} = e^{\boldsymbol{w}^T_{(i)}\boldsymbol{x}^*}, \tag{30}$$

$$y^*_{(i)}|\lambda^*_{(i)} \sim \text{Poisson}(\lambda^*_{(i)}) \tag{31}$$

where $\boldsymbol{x}^* = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and then we compute the mean and variance for y^*, which gives

$$\mathbb{E}\left[y^*|\boldsymbol{y}\right] \approx 10.25 \tag{32}$$

$$\mathbb{V}\left[y^*|\boldsymbol{y}\right] \approx 54.05 \tag{33}$$

The code is given by

```
# input for predction
xstar = np.array([1, 0])

# generate samples from posterior
num_samples = 1000
w_samples = np.random.multivariate_normal(w_MAP, S, size=(num_samples))

# ancestral sampling to get y^*
lambda_samples = [np.exp(wi[0] + wi[1]*xstar) for wi in w_samples]
ystar_samples = np.random.poisson(lambda_samples)

# compute and print mean and variance
mean = np.mean(ystar_samples)
var = np.var(ystar_samples)
print(f'Mean: {mean:3.2f}')
print(f'Variance: {var:3.2f}')
```

End of solution

Question 3.6: What would happen to the posterior predictive distribution $p(y^*|\boldsymbol{y}, x^* = 0)$ if $\alpha \to \infty$? Explain your reasoning.

Solution

Since α is prior precision, we know that as $\alpha \to \infty$, the prior will concentrate around $\boldsymbol{w} = \mathbf{0}$. Hence, $\lambda^* = e^{\boldsymbol{w}^T\boldsymbol{x}^*}$ will approach $\lambda^* = e^0 = 1$. Therefore, the posterior predictive distribution will become closer and closer to a Poisson1-distribution as α increases.

End of solution