# ROB313: Assignment 4 Report

Sidhanth Moolayil
1004101679
April 18, 2021

## 1. Objectives

The purpose of this assignment is to utilize Bayesian inference for a logistic regression model, testing out parameters such as different standard deviations for priors, and choosing a proposal distribution. Finally, a Bayesian linear model will be built for the moana loa dataset using predesigned basis functions.

## 2. Code Structure and Employed Strategies

The structure of the code is formatted with several helper functions answering individual question parts, with the main code below where the user can control which questions should be run.

## 3. Question 1

### 3.A. Part A

The log marginal likelihood for the three prior variances, 0.5, 1, and 2 were approximated using the Laplace approximation and reported in table 1.

Table 1: Approximated Log Marginal Likelihoods for the Different Prior Variances

| Prior Variance | Log Marginal Likelihood |
|---|---|
| 0.5 | -74.82 |
| 1 | -74.51 |
| 2 | -74.81 |

The model using a prior with a variance of 1 was found to be the highest complexity, as indicated by its highest marginal likelihood.

### 3.B. Part B

A gaussian distribution with a variance of 5 was chosen as the proposal distribution. This variance was chosen among a few tested variances, selected based on the criteria of minimizing the negative log likelihood on the validation set. This proposal distribution, when used with a prior with a variance of 1 on the testing set was found to have a testing negative log likelihood of 6.63, and a testing accuracy of 0.73. This proposal distribution is reasonably accurate, as seen in Figure 1 where the sampled posterior overlaps somewhat with the proposal distribution, with higher probability samples arising near the proposal distribution’s peak.

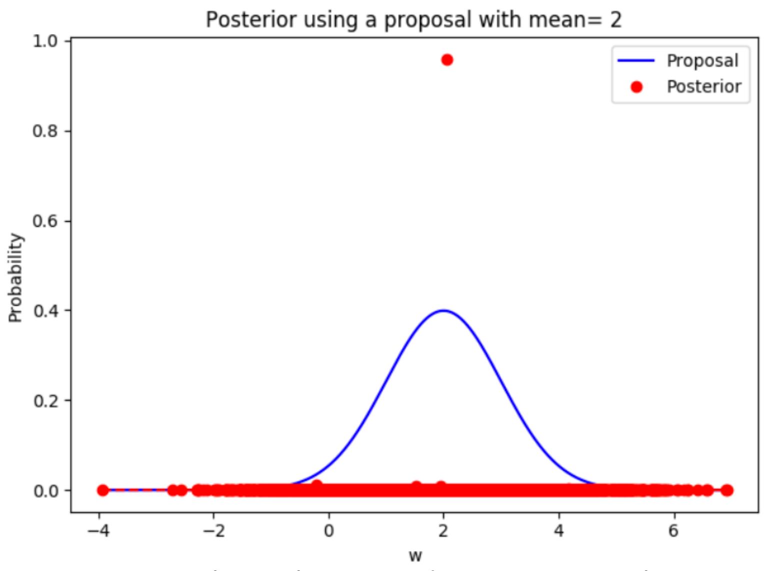
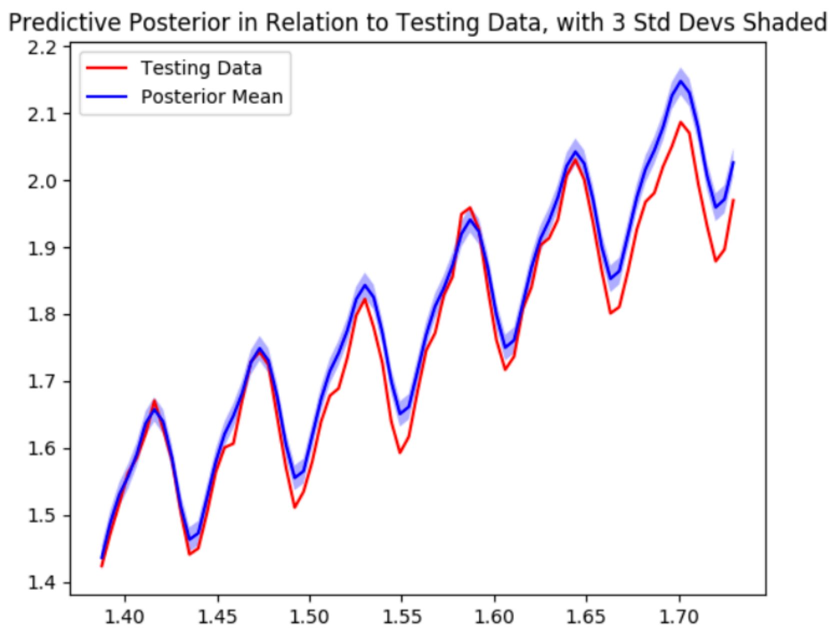


Figure 1. Proposal Distribution and Posterior Samples

4. Question 2

With the designed basis functions for the maona loa dataset, a Bayesian linear model was constructed. The predictive posterior is pictured below in Figure 2 relative to the testing data, with a 99.7% confidence interval shaded.



The quality of the predictive posterior, while visually following the trend of the maona loa dataset, is of a relatively low quality. This is because much of the true observed data still fell outside the 99.7% confidence interval of the posterior. To improve this model, a larger variance for the likelihood function would improve the predictive posterior.

## Appendix

```
from data_utils import load_dataset
import numpy as np
import math
import scipy
from matplotlib import pyplot as plt

# QUESTION 1
# PART A
def sigmoid(z):
    return 1/(1 + np.exp(-1*z))

def log_likelihood(x_prod, y_act):
    ll = np.dot(y_act.T, np.log(sigmoid(x_prod))) + np.dot((1-y_act).T, np.log((1-
sigmoid(x_prod))))
    return ll

def log_likelihood_grad(X, x_prod, y_act):
    ll_grad = np.sum([(y_act[i] - sigmoid(x_prod[i])) * X[i] for i in
range(len(x_prod))], axis=0)
    return ll_grad

def log_likelihood_hessian(X, x_prod):
    temp = np.multiply(sigmoid(x_prod), sigmoid(x_prod) - 1)
    ll_hessian = np.sum([temp[i] * np.outer(X[i], X[i].T) for i in
range(len(x_prod))], axis=0)
    return ll_hessian

def log_prior(w, sigma):
    log_p = -len(w)/2 * np.log(2 * np.pi) - len(w)/2 * np.log(sigma) - 1/(2 * sigma) *
np.dot(w.T, w)
    return log_p

def log_prior_grad(w, variance):
    return -1/variance * w

def log_prior_hessian(w, variance):
    return -1/variance * np.eye(len(w))

def laplace_approximation(x_train, y_train, var):
    w = np.zeros(np.shape(x_train[0]))
    x_prod = np.reshape(np.dot(x_train, w), np.shape(y_train))
    post_grad = log_likelihood_grad(x_train, x_prod, y_train) + log_prior_grad(w, var)

    while max(post_grad) > 10**(-9):
        x_prod = np.reshape(np.dot(x_train, w), np.shape(y_train))
        post_grad = log_likelihood_grad(x_train, x_prod, y_train) + log_prior_grad(w,
var)
        w += 0.001 * post_grad

    post_hessian = log_likelihood_hessian(x_train, x_prod) + log_prior_hessian(w, var)
    marginal_likelihood = log_likelihood(x_prod, y_train) + log_prior(w, var) - (1/2 *
np.log(np.linalg.det(-1 * post_hessian)) - len(post_hessian) / 2 * np.log(2 * np.pi))
    return marginal_likelihood

# PART B
def proposal(mean, variance):
    proposal = np.random.multivariate_normal(mean=mean, cov=np.eye(np.shape(mean)[0])
* variance)
    return proposal
```

```python
def likelihood(x, y):
    likelihood = np.prod([(sigmoid(x[i]) ** y[i]) * ((1 - sigmoid(x[i])) ** (1 -
y[i])) for i in range(len(x))], axis=0)
    return likelihood

def prior_likelihood(w, variance):
    prior = np.prod([1 / math.sqrt(2 * math.pi * variance) * math.exp(-(w[i] ** 2) /
(2 * variance)) for i in range(len(w))], axis=0)
    return prior

def proposal_likelihood(w, variance, mean):
    proposal = np.prod([1 / math.sqrt(2 * math.pi * variance) * math.exp(-((mean[i] -
w[i]) ** 2) / (2 * variance)) for i in range(len(w))], axis=0)
    return proposal

def importance_sampling(x_train, x_valid, x_test, y_train, y_valid, y_test):
    mean = [-2, -1, 0, 1, 2]
    variances = [0.5, 1, 2, 5, 10]

    min_ll = np.inf
    size = 500
    for var in variances:
        w = [proposal(mean, var) for i in range(size)]
        v_pred = np.zeros(np.shape(y_valid))
        vd_pred = np.zeros(np.shape(y_valid))
        for d in range(len(x_valid)):
            temp = np.sum([likelihood(np.dot(x_train, w[j]), y_train) *
prior_likelihood(w[j], var) / proposal_likelihood(w[j], 1, mean) for j in
range(size)])
            v_pred[d] = np.sum([sigmoid(np.dot(x_valid[d], w[i])) *
(likelihood(np.dot(x_train, w[i]), y_train) * prior_likelihood(w[i], var) /
proposal_likelihood(w[i], 1, mean)) / temp for i in range(size)])
            if v_pred[d] > 0.5:
                vd_pred[d] = 1
            elif v_pred[d] < 0.5:
                vd_pred[d] = 0
            else:
                vd_pred[d] = -1

        valid_log_likelihood = -(np.dot(y_valid.T, np.log(v_pred)) + np.dot((1-
y_valid).T, np.log((1-v_pred))))
        if valid_log_likelihood < min_ll:
            min_ll = valid_log_likelihood
            min_acc = (vd_pred == y_valid).sum() / len(y_valid)
            best_var = var

    test_acc, test_nll = test_set_prediction(x_test, x_train, x_valid, y_test,
y_train, y_valid, size, mean, best_var)

    print('Proposal Variance: ', best_var)
    print('Validation Accuracy: ', min_acc)
    print('Validation nll: ', min_ll)
    print('Test Accuracy: ', test_acc)
    print('Test nll: ', test_nll)
    y_train = np.vstack((y_train, y_valid))
    w = [proposal(mean, best_var) for i in range(5000)]
    total = np.sum([likelihood(np.dot(x_train, w[j]), y_train) *
prior_likelihood(w[j], best_var) / proposal_likelihood(w[j], 1, mean) for j in
range(5000)])
    posterior = [(likelihood(np.dot(x_train, w[i]), y_train) * prior_likelihood(w[i],
best_var) / proposal_likelihood(w[i], 1, mean)) / total for i in range(5000)]
```

```
    variance = 1
    for i in range(len(mean)):
        weights = [w[j][i] for j in range(len(w))]
        weights, posterior = zip(*sorted(zip(weights, posterior)))
        z = np.polyfit(weights, posterior, 1)
        z = np.squeeze(z)
        p = np.poly1d(z)
        w_all = np.arange(min(weights), max(weights), 0.001)
        q_w = scipy.stats.norm.pdf(w_all, mean[i], variance)
        plt.figure(i)
        plt.title("Posterior using a proposal with mean= " + str(round(mean[i])))
        plt.xlabel("w")
        plt.ylabel("Probability")
        plt.plot(w_all, q_w, '-b', label="Proposal")
        plt.plot(weights, posterior, 'or', label="Posterior")
        plt.plot(weights, p(weights), "r--")
        plt.legend()
        plt.show()
    return 0

def test_set_prediction(x_test, x_train, x_valid, y_test, y_train, y_valid, size,
mean, best_var):
    x_train = np.vstack((x_train, x_valid))
    y_train = np.vstack((y_train, y_valid))
    t_pred = np.zeros(np.shape(y_test))
    td_pred = np.zeros(np.shape(y_test))
    w = [proposal(mean, best_var) for i in range(size)]
    for d in range(len(x_test)):
        temp = np.sum([likelihood(np.dot(x_train, w[j]), y_train) *
prior_likelihood(w[j], best_var) / proposal_likelihood(w[j], 1, mean) for j in
range(size)], axis=0)
        prediction = np.sum([sigmoid(np.dot(x_test[d], w[i])) *
(likelihood(np.dot(x_train, w[i]), y_train) * prior_likelihood(w[i], best_var) /
proposal_likelihood(w[i], 1, mean))/temp for i in range(size)], axis=0)
        t_pred[d] = prediction
        if t_pred[d] > 0.5:
            td_pred[d] = 1
        elif t_pred[d] < 0.5:
            td_pred[d] = 0
        else:
            td_pred[d] = -1
    test_accuracy = (td_pred == y_test).sum() / len(y_test)
    test_nll = -(np.dot(y_test.T, np.log(t_pred)) + np.dot((1-y_test).T, np.log((1-
t_pred))))
    return test_accuracy, test_nll

# QUESTION 2
def features(x):
    """
    evaluates phi(x)
    Inputs:
    x : (N, 1) input datapoints
    Outputs:
    phi : (N, M) features for each datapoint
    """
    year = 0.057 # equal to one year in input space
    phi = np.hstack(
        # add polynomial terms
        [np.power(x, np.arange(5))]
        # add periodic terms:
```

```python
        + [np.sin(x*2*np.pi*factor / year) for factor in range(1, 11)]
        + [np.cos(x*2*np.pi*factor / year) for factor in range(1, 11)] )
    return phi

def posterior(Phi, t, variance):
    S_N_inv = np.eye(Phi.shape[1]) + (1/variance) * Phi.T.dot(Phi)
    S_N = np.linalg.inv(S_N_inv)
    m_N = (1/variance) * S_N.dot(Phi.T).dot(t)
    return m_N, S_N

def posterior_predictive(Phi_test, m_N, S_N, variance):
    y = Phi_test.dot(m_N)
    y_var = variance + np.sum(Phi_test.dot(S_N) * Phi_test, axis=1)
    return y, y_var

def bayesian_linear_model(x_train, y_train, x_test, y_test):
    variance = 10**(-4)

    Phi_test = features(x_test)

    Phi_N = features(x_train)

    m_N, S_N = posterior(Phi_N, y_train, variance)

    y, y_var = posterior_predictive(Phi_test, m_N, S_N, variance)

    plt.figure()
    plt.plot(x_train, y_train, "g-", label='Training Data')
    plt.plot(x_test, y_test, "r-", label='Testing Data')
    plt.plot(x_test, y, "b-", label='Random Posterior Sample')
    plt.title("Predictive Posterior Mean in Relation to Testing Data, with Training
Data")
    plt.legend()
    plt.show()

    plt.figure()
    plt.plot(x_test, y_test, "r-", label='Testing Data')
    plt.plot(x_test, y, "b", label='Posterior Mean')
    plt.fill_between(x_test.flatten(), (y -
np.reshape(np.sqrt(3*y_var),y.shape)).flatten(), (y +
np.reshape(np.sqrt(3*y_var),y.shape)).flatten(), alpha=0.3, facecolor = "blue")
    plt.title("Predictive Posterior in Relation to Testing Data, with 3 Std Devs
Shaded")
    plt.legend()
    plt.show()


# Question Data Loader Functions
def Q1a():
    x_train, x_valid, x_test, y_train, y_valid, y_test = load_dataset('iris')
    x_train = np.vstack((x_train, x_valid))
    y_train, y_test = np.vstack((y_train[:,(1,)], y_valid[:,(1,)])), y_test[:,(1,)]
    x_train = np.hstack((np.ones((len(x_train),1)),x_train))
    variances = [0.5, 1, 2]
    for var in variances:
        marginal_likelihood = laplace_approximation(x_train, y_train, var)
        print('Variance: ', var, ' Marginal Likelihood: ', marginal_likelihood)

def Q1b():
    x_train, x_valid, x_test, y_train, y_valid, y_test = load_dataset('iris')
    y_train, y_valid, y_test = y_train[:,(1,)], y_valid[:,(1,)], y_test[:,(1,)]
    x_train = np.hstack((np.ones((len(x_train),1)),x_train))
    x_valid = np.hstack((np.ones((len(x_valid),1)),x_valid))
    x_test = np.hstack((np.ones((len(x_test),1)),x_test))
    importance_sampling(x_train, x_valid, x_test, y_train, y_valid, y_test)

def Q2():
    x_train, x_valid, x_test, y_train, y_valid, y_test = load_dataset('mauna_loa')
    x_train = np.vstack((x_train, x_valid))
    y_train = np.vstack((y_train, y_valid))
    bayesian_linear_model(x_train, y_train, x_test, y_test)


Q1a()
Q1b()
Q2()
```