Bayesian Machine Learning

Stephen Charles

Final Project
The University of Bath
May 2022

Abstract

The final piece of assessed coursework involves the evaluation of Bayesian modelling methods on a real multivariate regression task. The guiding objectives are to derive a good predictor for data derived from an "energy efficiency" data set, and to estimate which of the input variables are relevant for prediction. In particular, the exercise focuses on approximating (and averaging over) posterior distributions using the Hamiltonian Monte Carlo (HMC), Variational Inference (VI), and Gaussian Processes.

Experiments will be based mainly on existing (or supplied) analytic code and techniques you have already learned. You will of course need to write all the relevant code in the relevant cells of the Jupyter Notebook to process the data, apply the methods appropriately, extend them in places, and ultimately calculate and output the necessary results. A key part of the assessment is to compile, present and critique all those results effectively within an "individual project report" document. For this exercise, your code in the Jupyter Notebook will also be auto-marked. Marks will be awarded based on the report, code, and (potentially) group contributions.

Contents

List of Figures

List of Tables

List of Code

Chapter 1

Exploratory Analysis

The energy efficiency data set is distributed by the University of Oxford, and available for download at the UCI Machine Learning Repository. It is a multivariate dataset containing 768 examples, comprising of 1 constant bias and 8 input variables $x_0, x_1, x_2, \cdots, x_8$, where x_0 is the constant bias, and the rest represent basic architectural parameters for buildings.

For the purposes of modeling, we chose to pre-process the data to standardise the inputs to have 0 mean and a standard deviation of 1. The *const* column was preserved to be 1. The data was split into two sets, *'ee-train.csv'* for the training stage, and *'ee-test.csv'* for the testing stage.

Previously, analysis (Tsanas and Xifara, 2012) for this dataset used both random forest and classical linear regression models, which had a mean average error of 0.51 and 1.42 respectively.

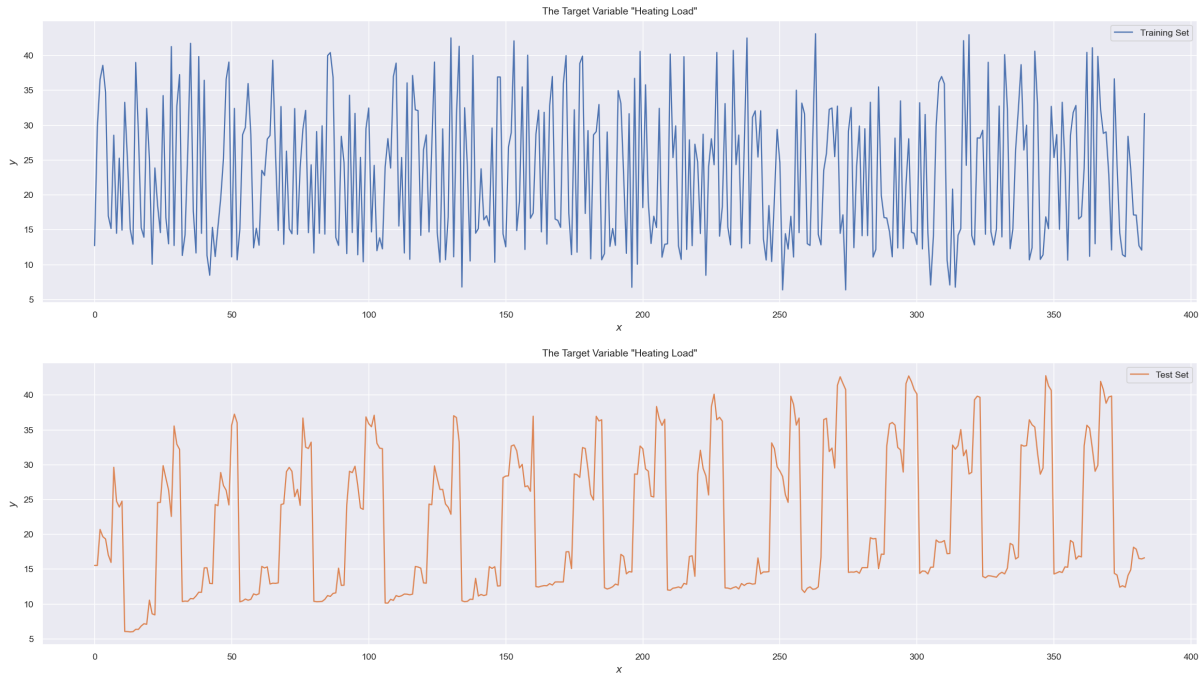

Figure 1.1: The training and test sets for the target variable *Heating Load*.

1.1 Correlation

The data appears non-Gaussian, thus we used the correlation coefficient (Spearman) as a metric to determine the association between the target variable and the dependants. In a sense we can gauge an estimation of the linearity using this method. Plots shown in Figure 1.2 were computed to illustrate the correlation between the target variable *Heating Load* and the 8 input variables.

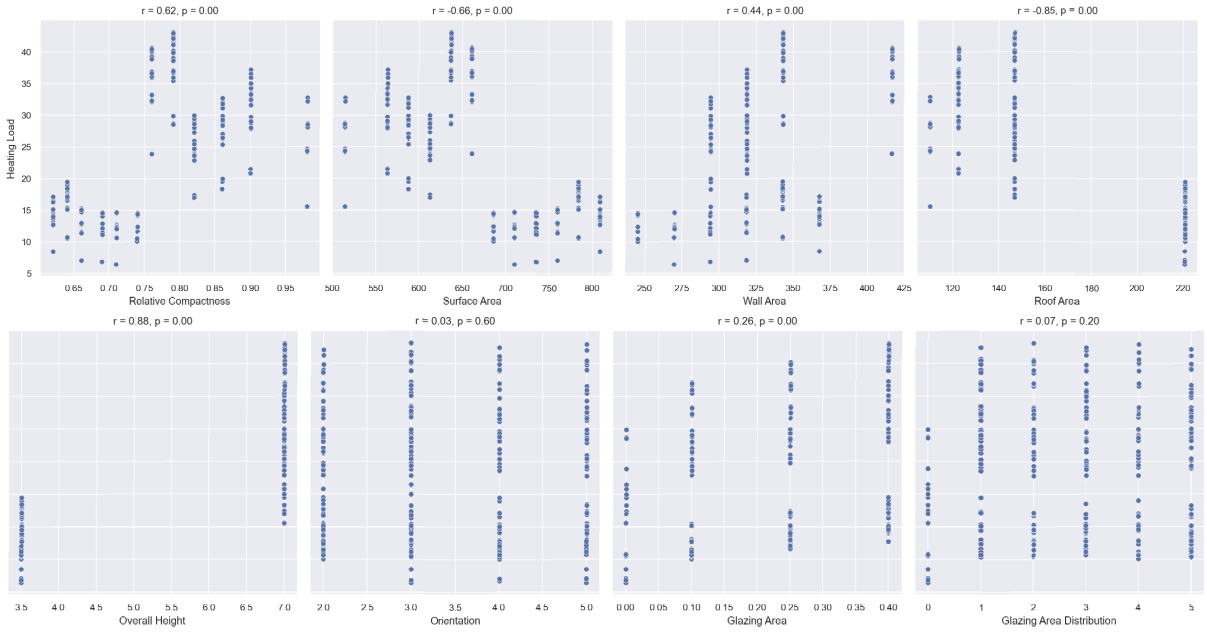

Figure 1.2: The correlation between heating load and the other dependent variables.

The least correlated and thus least important in terms of their feature importance to the model were *Orientation, Glazing Area and Glazing Area Distribution*. Conversely, the most important appear to be *Relative Compactness, Roof Area and Overall Height* in this analysis, in agreement with Tsanas and Xifara (2012).

Figure 1.3 shows a correlation heatmap between all features. Positive and negative correlation is defined the same as before, corresponding to the colormap indicated by the colorbar. The final row for *Heating Load* gives us the same information shown as before in Figure 1.2.

Figure 1.3: The correlation between all features.

1.2 Ordinary Least Squares

Ordinary Least Squares regression is a method where the coefficients of linear regression equations can be estimated which describe multiple independent variables and a dependent variable. Figure 1.4 shows the results of such an approach, and Table 1.1. From the OLS regression results we can see that the most correlated features are x_1, x_2, x_4 and x_5, corresponding to *Relative Compactness, Surface Area, Roof Area* and *Overall Height*, in agreement with our visual inspection of the correlation.

Table 1.1: The RMSE and MAE of the train and test sets for OLS.

Metric	Train Error
RMSE	3.0115517876503617
MAE	2.1306794414069143
Metric	Test Error
RMSE	2.8435880167333694
MAE	2.069010093808354

OLS Regression Results

Dep. Variable:	y	R-squared:	0.910
Model:	OLS	Adj. R-squared:	0.909
Method:	Least Squares	F-statistic:	544.8
Date:	Sun, 15 May 2022	Prob (F-statistic):	1.72e-192
Time:	09:16:16	Log-Likelihood:	-968.22
No. Observations:	384	AIC:	1952.
Df Residuals:	376	BIC:	1984.
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	22.9207	0.155	147.581	0.000	22.615	23.226
x1	-7.2346	1.654	-4.373	0.000	-10.488	-3.982
x2	-3.9422	1.206	-3.269	0.001	-6.314	-1.571
x3	0.7560	0.312	2.420	0.016	0.142	1.370
x4	-4.2319	1.091	-3.880	0.000	-6.376	-2.087
x5	7.2040	0.858	8.401	0.000	5.518	8.890
x6	-0.1252	0.156	-0.803	0.423	-0.432	0.182
x7	2.7702	0.162	17.129	0.000	2.452	3.088
x8	0.2041	0.161	1.267	0.206	-0.113	0.521

Omnibus:	7.903	Durbin-Watson:	1.890
Prob(Omnibus):	0.019	Jarque-Bera (JB):	11.852
Skew:	0.109	Prob(JB):	0.00267
Kurtosis:	3.832	Cond. No.	8.07e+15

Figure 1.4: The correlation between heating load and the other dependent variables. x_3, x_6 and x_8 have the highest p-values, which suggests those are the least important to predicting our target variable.

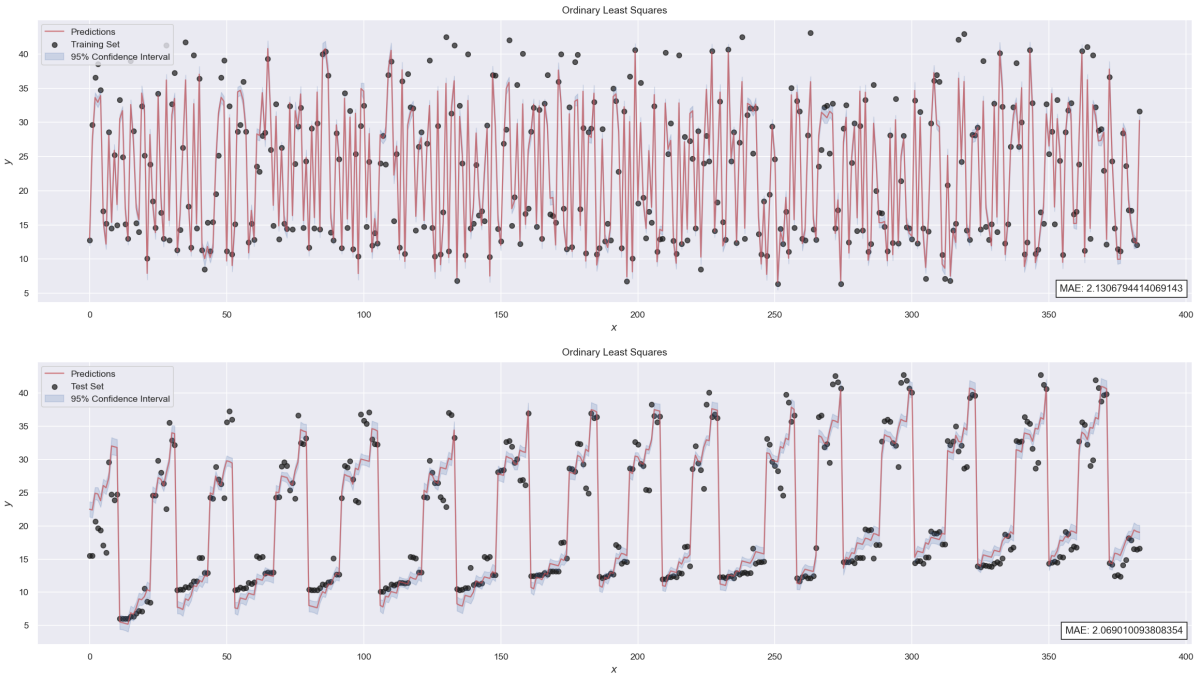

Figure 1.5: The predictions against the training and test set for OLS.

Chapter 2

Bayesian Linear Regression

Define a standard linear regression model with an unknown coefficient set $\mathbf{w}$.

- $\mathbf{w}$ is assumed to have a Gaussian prior $\mathcal{N}(0, \sigma_w^2)$, and the precision as $\alpha = \frac{1}{\sigma_w^2}$.
- The problem can be modelled using additive Gaussian noise $\mathcal{N}(0, \sigma_\epsilon^2)$, and the precision as $\beta = \frac{1}{\sigma_\epsilon^2}$.
- The unknown hyperparameter set consists of $\theta = (\sigma_w^2, \sigma_\epsilon^2) = (\alpha, \beta)$.
- With observation $\mathcal{D}$, the posterior we want to estimate can be written as $p(\mathbf{w}, \theta|\mathcal{D})$, in our case this is y, *Heating Load*.

2.1 Type-II Maximum Likelihood

The full posterior we require is $p(\mathbf{w}, \theta|\mathcal{D})$ as defined above, where $\theta = (\sigma_w^2, \sigma_\epsilon^2) = (\alpha, \beta)$. The Type-II maximum likelihood is used to infer the hyperparameters α and β, and the posterior in our case for this dataset is

$$p(\mathbf{w}, \alpha, \beta|y) \equiv p(\mathbf{w}|\alpha, \beta, y)p(\alpha, \beta|y). \tag{2.1}$$

The second term cannot be inferred directly, thus to find the most probable alpha and beta we maximise

$$p(\alpha, \beta|y) = \frac{p(y|\alpha, \beta)p(\alpha)p(\beta)}{p(y)} \tag{2.2}$$

where

$$\alpha, \beta = \text{argmax} \log p(y|\alpha|\beta). \tag{2.3}$$

The precision is

$$\Sigma = \beta \boldsymbol{I} + \alpha^{-1} \boldsymbol{\Phi} \boldsymbol{\Phi}^T \tag{2.4}$$

$$\alpha = \frac{1}{\sigma_w^2} \tag{2.5}$$

$$\beta = \frac{1}{\sigma_\epsilon^2}. \tag{2.6}$$

```python
def compute_log_marginal(X, y, alph, beta):
    """Type 2 maximum liklihood"""
    cov = 1 / beta * np.identity(X.shape[0]) + np.matmul(alph ** (-1) * X, X.T)
    return stats.multivariate_normal.logpdf(y, cov=cov, allow_singular=True)
```

Listing 2.1: The log marginal function.

The log marginal then becomes

$$\frac{1}{(2\pi)^{\frac{n}{2}}\Sigma^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(y-\mu)^T\Sigma^{-1}(y-\mu)\right). \tag{2.7}$$

We wish to maximise the marginal likelihood $p(y|\alpha,\beta)$ to approximate the most probable values for α and β. For this, 100 uniformly distributed samples across the range -5 to 0 were iterated over using `compute log marginal` to determine the most likely values shown in Figure 2.1 and Table 2.1. The predictions were made via the `compute posterior` function. The errors in the train and test sets are displayed in Table 2.2.

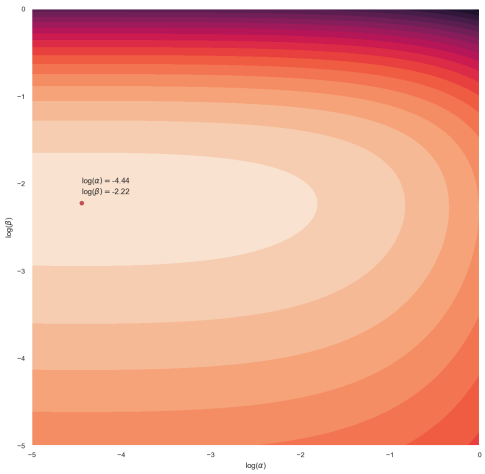

Figure 2.1: For clarity, the posterior is shown in log space. The most probable values were $\alpha = 0.117$ and $\beta = 0.108$.

Table 2.1: The most probable values for the Type-II Maximum Likelihood.

Priors	
α	0.01174362845702136
β	0.10836802322189586
$\log(\alpha)$	-4.444444444444445
$\log(\beta)$	-2.22222222222222237
Log-Likelihood	-1001.4576252255179

```python
def compute_log_marginal(PHI, y, alph, s2):
    '''Compute the log of the marginal likelihood for BLR model.'''
    def woodbury_inverse(I, U, V):
        '''Woodbury Identity'''
        return np.identity(U.shape[0]) - U @ np.linalg.inv(I + V @ U) @ V

    s2 = 1 / s2
    I = np.identity(PHI.shape[1])
    U = 1 / (alph * s2) * PHI
    V = PHI.T

    const = np.log(2 * np.pi) * (len(y) / 2)
    p1 = -(np.log(np.diagonal(np.linalg.cholesky((s2 * np.identity(PHI.shape[1]) \
         + (PHI.T @ (alph ** -1 * PHI)))))).sum() \
         + np.log(np.sqrt(s2)) * (PHI.shape[0] - PHI.shape[1]))
    p2 = - 0.5 * y.T @ ((1 / s2) * woodbury_inverse(I, U, V)) @ y
    lgp = const + p1 + p2
    return lgp
```

Listing 2.2: The efficient woodbury inverse log marginal function.

Table 2.2: The RMSE and MAE of the train and test sets for BLR.

Metric	Train Error
RMSE	3.011551809679529
MAE	2.130668537391788
Metric	Test Error
RMSE	2.8063010631019174
MAE	1.9907650699570147

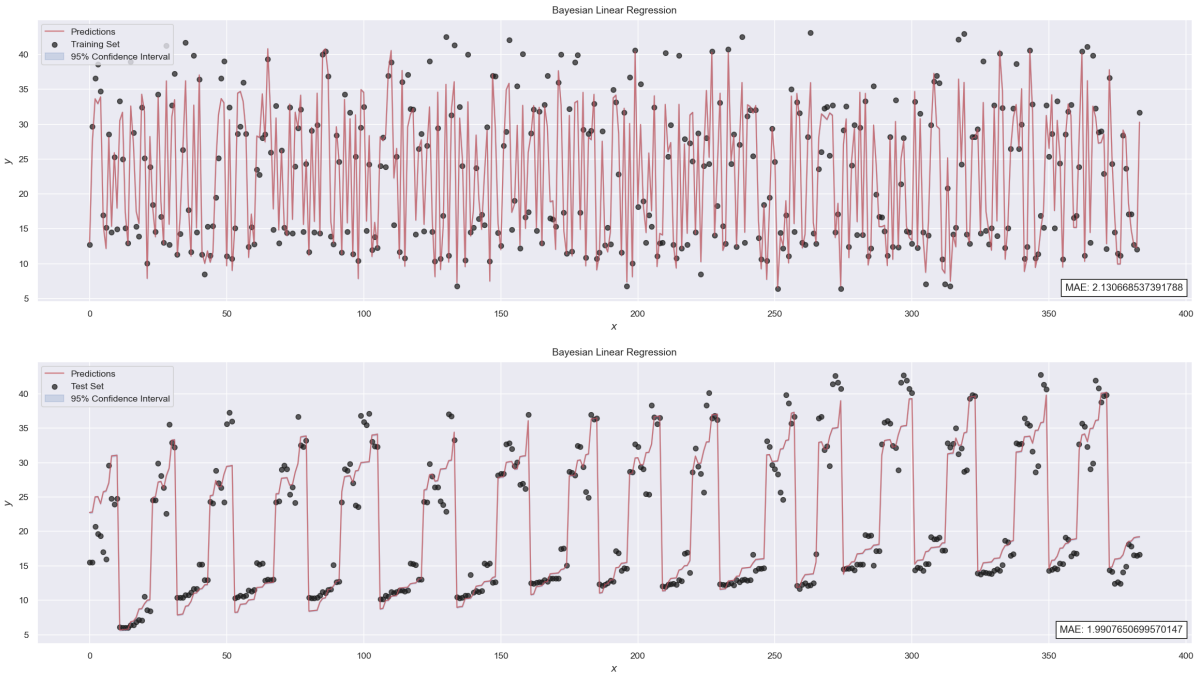

Figure 2.2: The predictions against the training and test set for BLR.

2.2 Variational Inference

Variational Inference is a method for approximating distributions, casting inference as an optimization problem. The main goal is to find a good proposal distribution $\mathcal{Q}(\theta)$ which is the best match for an unknown posterior distribution or objective $\mathcal{P}(\theta)$. The Kullback-Leibler divergence is used to analyse similarity between the proposal and posterior, to determine similarities between both.

In this project, the proposal is given by

$$\mathcal{Q}(\mathbf{w}, \alpha, \beta) = \mathcal{Q}(\mathbf{w}, \beta)\mathcal{Q}(\alpha), \tag{2.8}$$

with priors

$$\mathcal{Q}(\mathbf{w}) = \mathcal{N}(\mathbf{w}|0, (\alpha\beta)^{-1}) \tag{2.9}$$
$$\mathcal{Q}(\alpha) = \mathcal{G}(\alpha|a_0^{\alpha}, |b_0^{\alpha}) = \mathcal{G}(\alpha|a, b) \tag{2.10}$$
$$\mathcal{Q}(\beta) = \mathcal{G}(\beta|a_0^{\beta}, |b_0^{\beta}) = \mathcal{G}(\beta|c, d), \tag{2.11}$$

where

$$\mu_N = \tfrac{a_N}{b_N}\mathbf{I} + \mathbf{X}^T\mathbf{X} \tag{2.12}$$
$$\Sigma_N = V_N\mathbf{X}^T\mathbf{y} \tag{2.13}$$
$$a_N = a_0 + \tfrac{K}{2} \tag{2.14}$$
$$b_N = \tfrac{1}{2}b_0(\tfrac{c_N}{d_N})\Sigma_N^T\Sigma_N + Tr(\mu_N) \tag{2.15}$$
$$c_N = c_0 + \tfrac{N}{2} \tag{2.16}$$
$$d_N = \tfrac{1}{2}d_0(||y - x\Sigma_N||^2 + \Sigma_N^T(\tfrac{a_N}{b_N}))\Sigma_N \tag{2.17}$$
$$\tag{2.18}$$

The floats a_N, b_N, c_N, d_N give us the hyper-parameters α and β, and are defined by

$$\alpha = \tfrac{a_N}{b_N} \tag{2.19}$$
$$\beta = \tfrac{c_N}{d_N}. \tag{2.20}$$

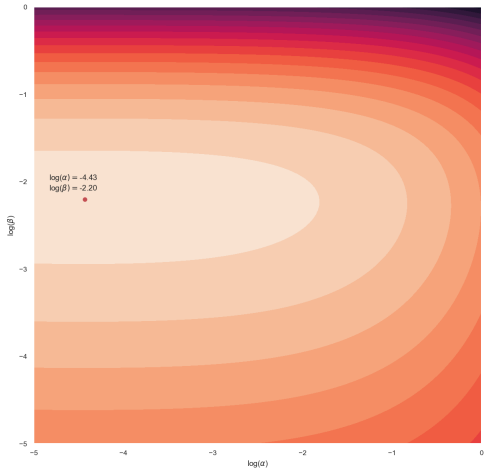

Figure 2.3: For clarity, the posterior is shown in log space. The expected values were $\alpha = 0.012$ and $\beta = 0.110$.

Table 2.3: The most probable values for VI.

Priors	
α	0.011916753508509775
β	0.11026038962261975
$\log(\alpha)$	-4.42981001110939
$\log(\beta)$	-2.2049105321553415
Log-Likelihood	-1001.4975203139636

Table 2.4: The RMSE and MAE of the train and test sets for VI.

Metric	Train Error
RMSE	3.0115518111318993
MAE	2.1306681834494188
Metric	Test Error
RMSE	2.8435839397385267
MAE	2.0689828677240825

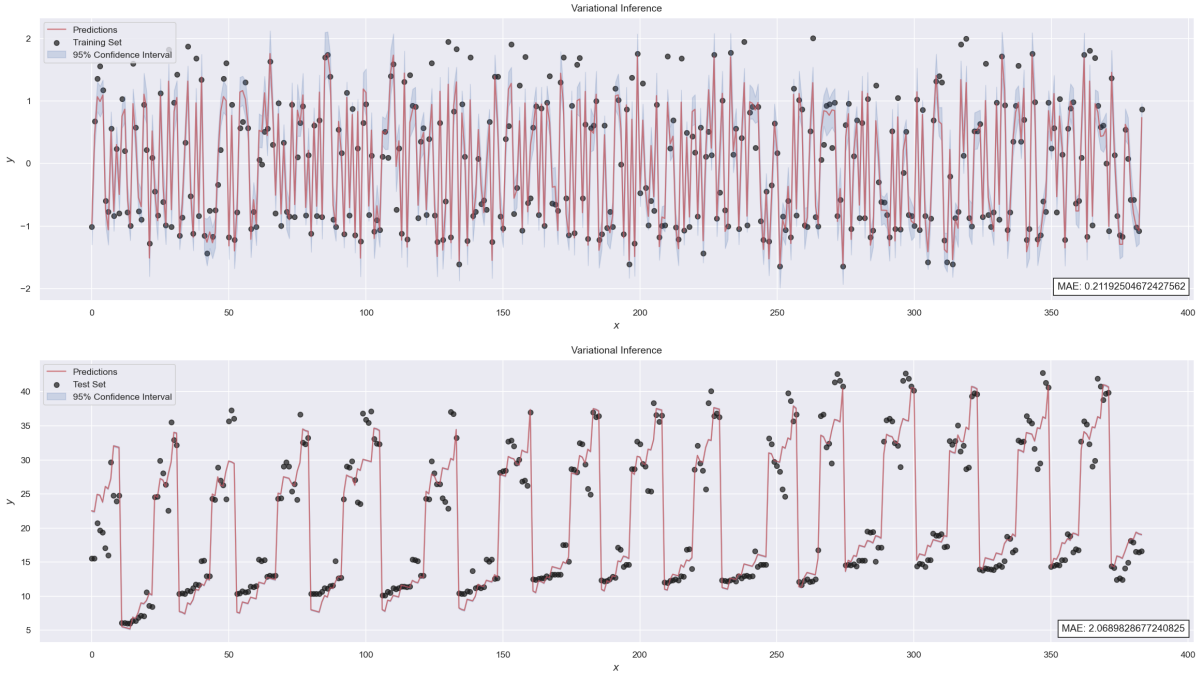

Figure 2.4: The predictions against the training and test set for VI.

Chapter 3

Verify HMC on a 2D Gaussian

The Hamiltonian Monte Carlo method aims to make the random walk in algorithms more efficient. The 2D Gaussian proposed is shown in Figure 3.1 with mean 0 and variance 1. The mathematical formula for this is

$$f(x,y) = A\exp\left(-\left(\frac{(x-x_0)^2}{2\sigma_X^2} + \frac{(y-y_0)^2}{2\sigma_Y^2}\right)\right). \tag{3.1}$$

The energy function is calculated via

$$\Sigma = \begin{pmatrix} \sigma_{00}^2 & \sigma_{10}^2 \\ \sigma_{01}^2 & \sigma_{11}^2 \end{pmatrix} \tag{3.2}$$

$$\Sigma^{-1} = \frac{1}{\sigma_{00}^2\sigma_{11}^2}\begin{pmatrix} \sigma_{11}^2 & -\sigma_{10}^2 \\ -\sigma_{01}^2 & \sigma_{00}^2 \end{pmatrix} \tag{3.3}$$

$$x^T\Sigma^{-1}x = \frac{\sigma_{11}^2x_0^2 - 2\sigma_{10}^2x_1x_0 + \sigma_{00}^2x_1^2}{(\sigma_{00}\sigma_{11})^2 - (\sigma_{01}\sigma_{10})^2} \tag{3.4}$$

$$\gamma = (\sigma_{00}\sigma_{11})^2 - (\sigma_{01}\sigma_{10})^2 \tag{3.5}$$

$$\frac{\partial(x^T\Sigma^{-1}x)}{\partial x_0} = -\frac{\sigma_{11}^2x_0}{\gamma} + \frac{\sigma_{10}^2x_1}{\gamma} \tag{3.6}$$

$$\frac{\partial(x^T\Sigma^{-1}x)}{\partial x_1} = \frac{\sigma_{10}^2x_0}{\gamma} - \frac{\sigma_{00}^2x_1}{\gamma}. \tag{3.7}$$

Table 3.1: The values used to obtain HMC results.

Hyper-Parameters	
R	5000
L	20
eps	0.36

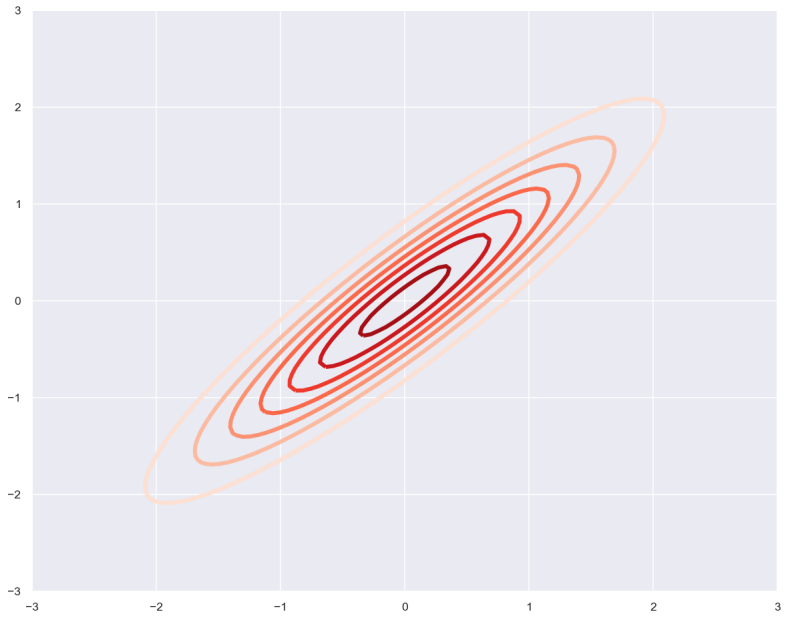

Figure 3.1: A 2D Gaussian with mean 0 and variance 1.

```
1 def energy_func(x, covar):
2     """Energy function. Returns Neglogpdf."""
3     return np.negative(stats.multivariate_normal.logpdf(x, cov=covar))
4
5 def energy_grad(x, covar):
6     """Gradient function."""
7     return np.linalg.inv(covar) @ x
```

Listing 3.1: Designed functions `energy_func` and `energy_grad`

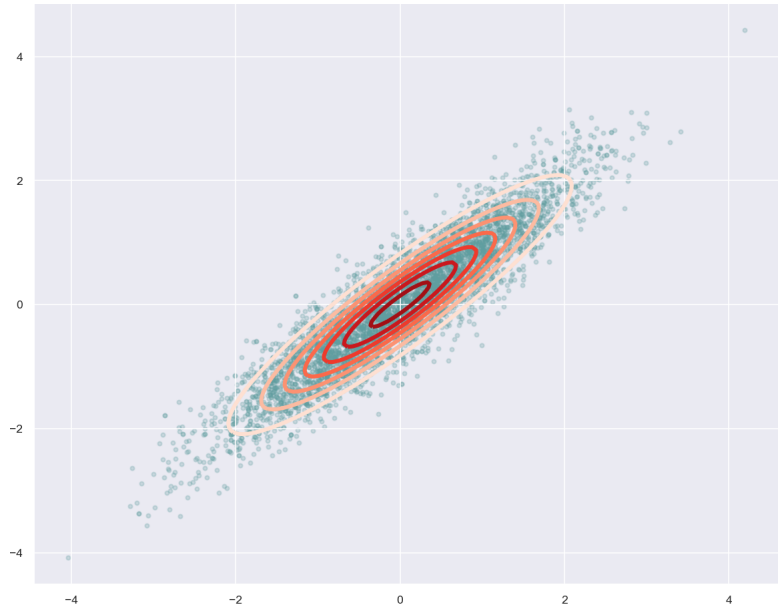

Figure 3.2: HMC Distribution fitted.

```
Calc.          Numeric       Delta          Acc.
    -2.91819       -2.91819  -3.195266e-10  10
     1.41705        1.41705   8.923884e-11  11
|----------|  0% accepted [ 3 secs to go ]
|#---------| 92% accepted [ 3 secs to go ]
|##--------| 92% accepted [ 2 secs to go ]
|###-------| 91% accepted [ 2 secs to go ]
|####------| 91% accepted [ 2 secs to go ]
|#####-----| 91% accepted [ 1 secs to go ]
|######----| 90% accepted [ 1 secs to go ]
|#######---| 90% accepted [ 1 secs to go ]
|########--| 90% accepted [ 1 secs to go ]
|#########-| 90% accepted [ 0 secs to go ]
|##########| 90% accepted [ 0 secs to go ]
HMC: R=5000 / L=20 / eps=0.36 / Accept=90.4%
```

Figure 3.3: The acceptance rate over iterations.

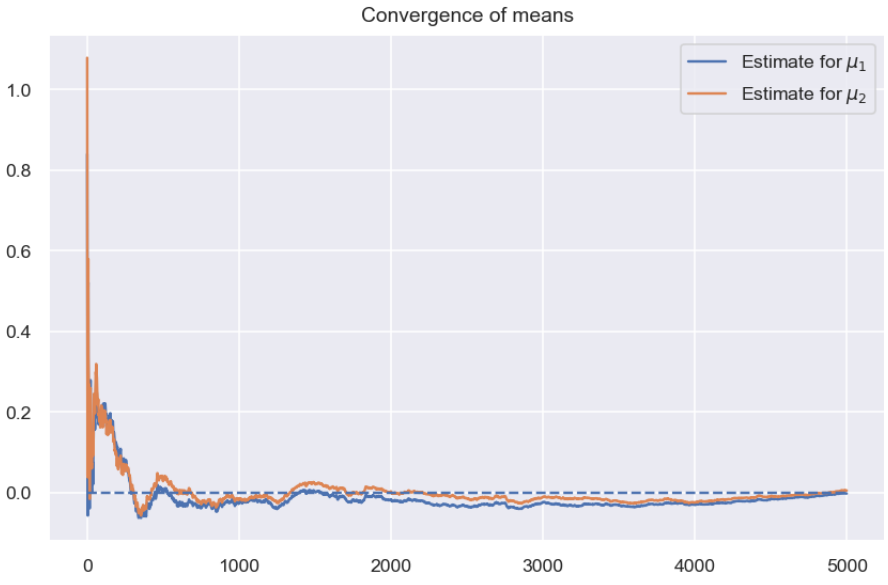

Figure 3.4: The convergence over 5000 samples.

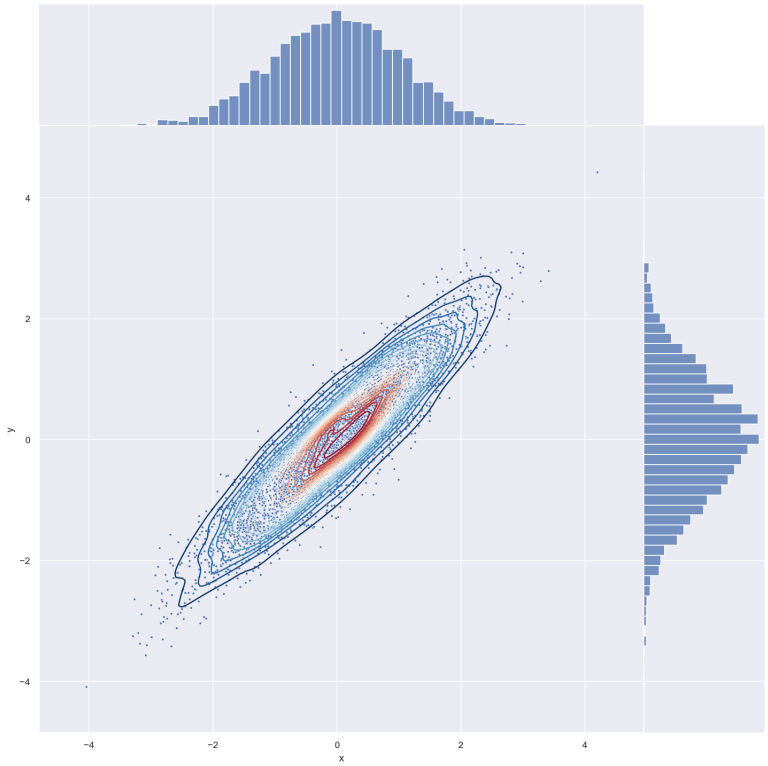

Figure 3.5: Verification of the HMC model to a simple Gaussian. Both histograms demonstrate a typical Gaussian distribution across the samples accepted.

Chapter 4

Apply HMC to the Linear Regression Model

The posterior distribution is defined as

$$p(\boldsymbol{w}|\alpha,\beta,\boldsymbol{y}) = p(\boldsymbol{y}|\boldsymbol{w},\beta,\alpha)p(\boldsymbol{w}|\alpha) \tag{4.1}$$

The energy function is calculated via

$$p(\boldsymbol{y}|\boldsymbol{X},\beta) = \left(\frac{\sqrt{\beta}}{\sqrt{2\pi}}\right)^n \exp\left(-\frac{\beta}{2}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w})^T(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w})\right) \tag{4.2}$$

$$p(\boldsymbol{w}|\alpha) = \left(\frac{\sqrt{\alpha}}{\sqrt{2\pi}}\right)^m \exp\left(-\frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}\right) \tag{4.3}$$

$$\mathbb{L} = -\left(\frac{n}{2}\log\beta - \frac{n}{2}\log 2\pi - \frac{\beta}{2}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w})^T(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w}) + \cdots \right. \tag{4.4}$$

$$\left. \cdots \frac{m}{2}\log\alpha - \frac{m}{2}\log 2\pi - \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}\right) \tag{4.5}$$

$$\frac{\partial\mathbb{L}}{\partial\alpha} = -\frac{m}{2\alpha} + \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} \tag{4.6}$$

$$\frac{\partial\mathbb{L}}{\partial\beta} = -\frac{n}{2\beta} + \frac{1}{2}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w})^T(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w}) \tag{4.7}$$

$$\frac{\partial\mathbb{L}}{\partial\boldsymbol{w}} = -\beta(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w})^T\boldsymbol{X} + \alpha\boldsymbol{w}^T. \tag{4.8}$$

Table 4.1: The hyper-parameters used to obtain HMC LR results.

Hyper-Parameters	
R	20000
L	100
eps	0.000075

```python
def energy_func_lr(hps, x, y):
    """Energy function."""
    alpha = hps[0]
    beta = hps[1]
    w = hps[2:]
    N, M = x.shape
    a = (N / 2 * np.log(beta)) - (N / 2 * np.log(2*np.pi)) \
        - beta / 2 * (np.sum((y - x @ w).T @ (y - x @ w)))
    b = (M / 2 * np.log(alpha)) - (M / 2 * np.log(2*np.pi)) \
        - (alpha / 2) * (np.sum(w.T @ w))
    neglgp = - (a + b)
    return neglgp

def energy_grad_lr(hps, x, y):
    """Gradient function returns arr of
    partial derivatives of the energy function."""
    alpha = hps[0]
    beta = hps[1]
    w = hps[2:]
    N, M = x.shape
    grad_alpha = - (M / (2 * alpha)) + (np.sum(w.T @ w) / 2)
    grad_beta = - (N / (2 * beta)) + (np.sum(((y - x @ w).T @ (y - x @ w))) / 2)
    grad_w = (alpha * w) + beta * (y - x @ w) @ x
    g = np.array([grad_alpha, grad_beta] + list(grad_w))
    return g
```

Listing 4.1: Designed functions `energy_func` and `energy_grad` for HMC LR.

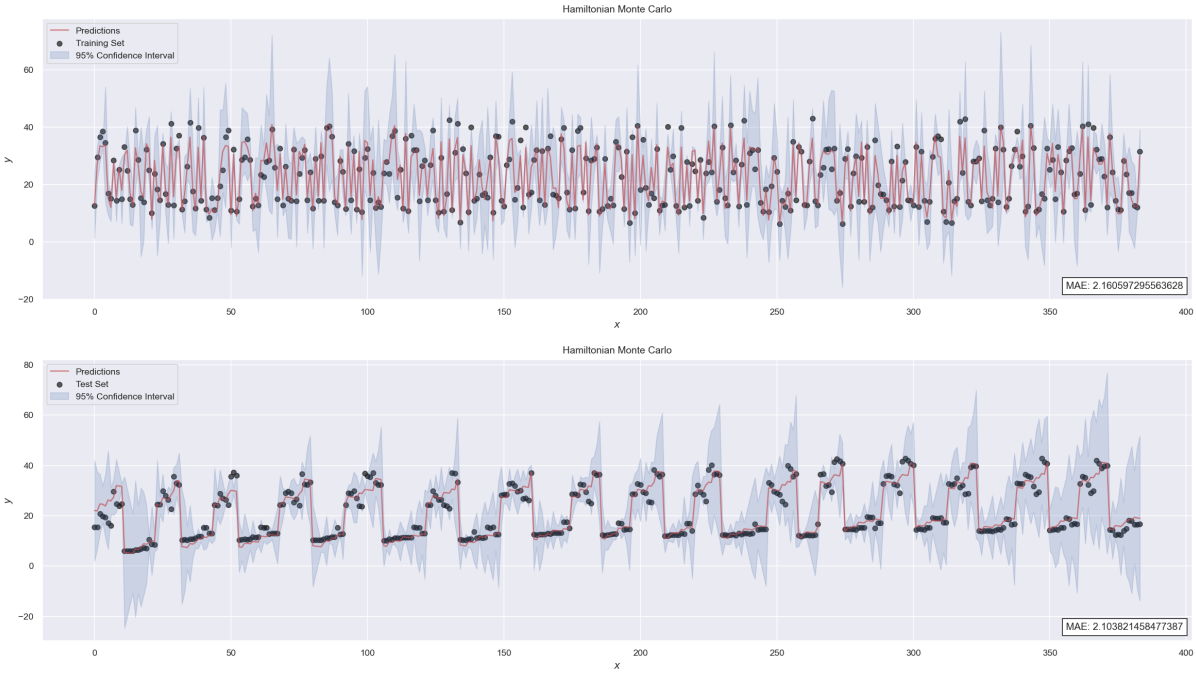

Figure 4.1: The predictions against the training and test set for HMC LR.

Table 4.2: The optimal values of the unknown terms for HMC LR.

Optimal Values	
α	0.014306838372461161
β	0.10854886876405839
Bias	0.20568049399730945

Table 4.3: The RMSE and MAE of the train and test sets for HMC LR.

Metric	Train Error
RMSE	3.018637231485769
MAE	2.160597295563628
Metric	Test Error
RMSE	2.852187436445753
MAE	2.103821458477387

```
Calc.          Numeric       Delta          Acc.
     290.633       290.633  -3.982234e-08  10
    -178.667      -178.667  -4.401591e-08  10
     2.28905       2.28905   2.599009e-09   9
   -0.722124     -0.722124  -1.016047e-08   8
    -0.39343      -0.39343  -2.445999e-08   8
   0.0755569      0.075557   2.518834e-08   7
   -0.422401     -0.422401  -1.829638e-08   8
    0.719567      0.719567  -7.127348e-08   8
  -0.0125016    -0.0125017  -6.799340e-08   6
    0.276659      0.276659  -7.174795e-08   7
   0.0203778     0.0203778  -1.893499e-08   7
|----------|  0% accepted [ 63 secs to go ]
|#---------| 90% accepted [ 56 secs to go ]
|##--------| 91% accepted [ 50 secs to go ]
|###-------| 92% accepted [ 43 secs to go ]
|####------| 90% accepted [ 37 secs to go ]
|#####-----| 90% accepted [ 31 secs to go ]
|######----| 90% accepted [ 24 secs to go ]
|#######---| 90% accepted [ 18 secs to go ]
|########--| 89% accepted [ 12 secs to go ]
|#########-| 90% accepted [ 6 secs to go ]
|##########| 90% accepted [ 0 secs to go ]
HMC: R=20000 / L=100 / eps=0.0025 / Accept=89.6%
```

Figure 4.2: Acceptance rates for HMC LR.

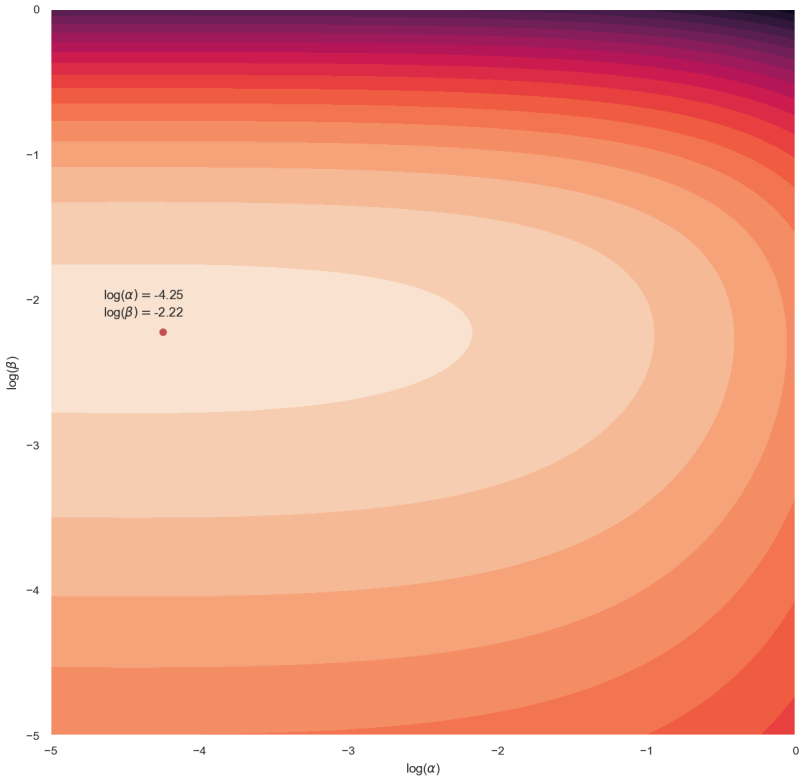

Figure 4.3: The posterior for HMC LR. The best α was 0.014 and β 0.108.

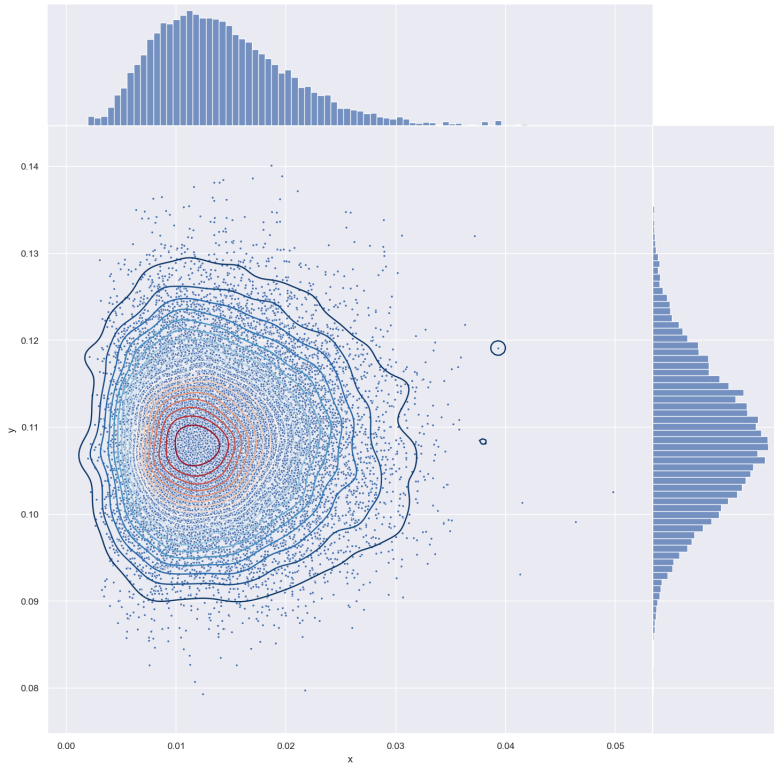

Figure 4.4: The posterior HMC samples fitted to BLR. The data is clearly non-gaussian, verified by the accepted samples.

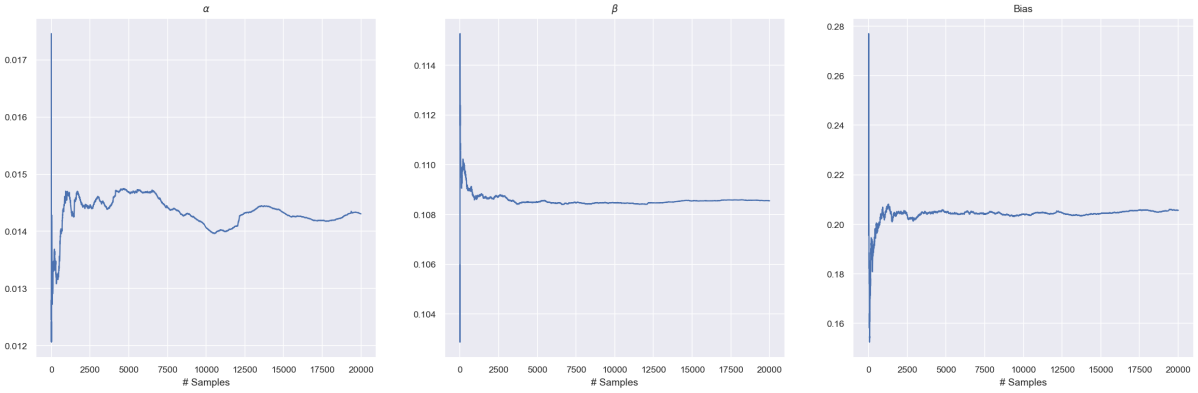

Figure 4.5: The optimal values of the unknown terms for HMC LR.

Chapter 5

Apply GP to the Linear Regression Model

A Gaussian Process is stochastic (collection of random variables), where every finite linear combination of random variables is normally distributed. The Gaussian Process itself is the joint distribution of all the normally distributed random variables. The prior mean is assumed to be constant and zero, which we have achieved through our preprocessing step with `StandardScaler`.

A note about the kernel choices made for GP:

- RBF - Explains a long term, smooth rising trend.
- WhiteKernel - Explains the correlated noise components.
- ExpSineSquared - Explains periodicity. To allow for variance in periodicity, multiply with an RBF Kernel.

5.1 Default Kernel

The default kernels hyper-parameters are optimised during fitting and uses

```
ConstantKernel(1.0, constant_value_bounds="fixed" *
RBF(1.0, length_scale_bounds="fixed").
```

Listing 5.1: Default Kernel for the Gaussian Process.

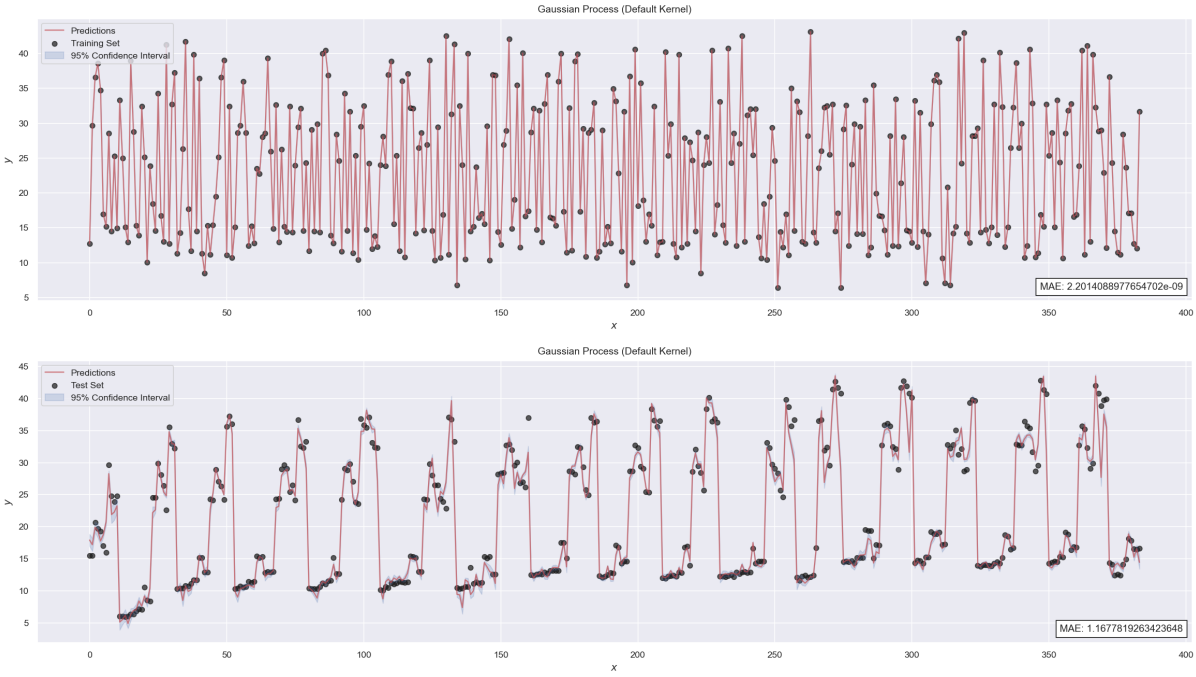

Figure 5.1: The predictions against the training and test set for GP (Default Kernel).

The hyper-parameters of the kernel are optimized during fitting of `GaussianProcessRegressor` by maximizing the log-marginal-likelihood (LML) based on the passed optimizer.

5.2 Custom RBF Kernel

The first kernel I tried was to modify the parameters of the RBF kernel

```
kernel = 1.0 * RBF(length_scale=0.5, length_scale_bounds=(1e-3, 1e2))
```

Listing 5.2: Custom RBF Kernel for the Gaussian Process.

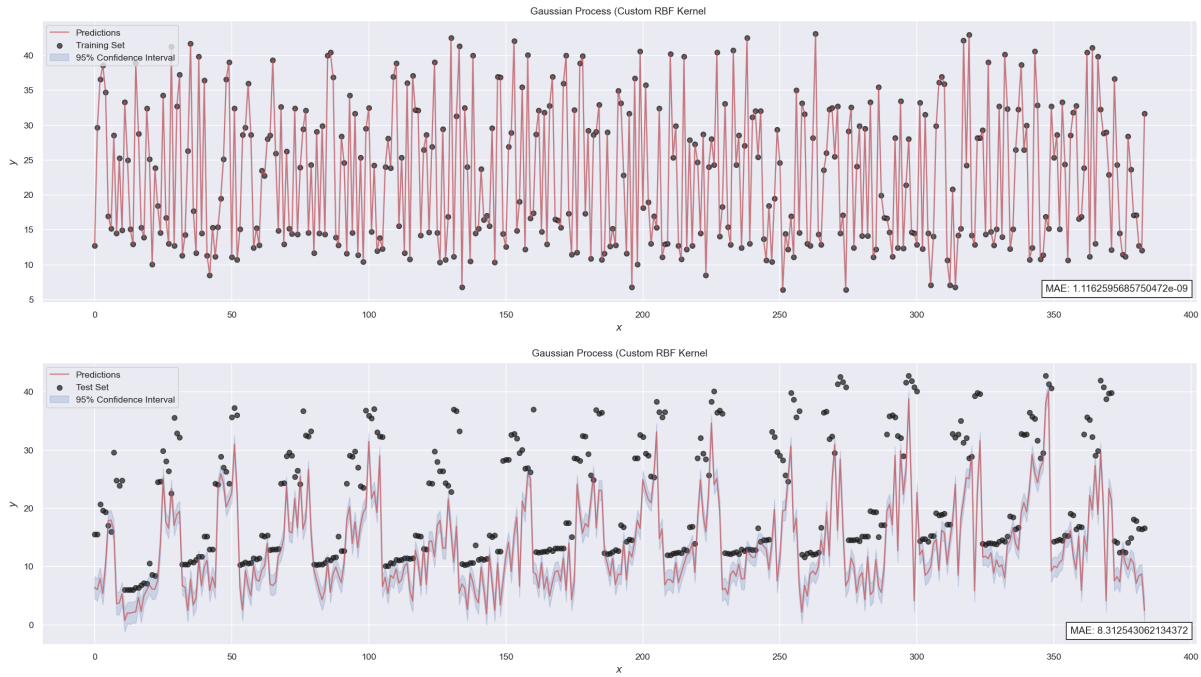

Figure 5.2: The predictions against the training and test set for GP (Custom RBF Kernel).

5.3 Custom RBF Kernel + White Noise

The `sklearn` (Pedregosa et al., 2011) documentation states that:

> "The noise level in the targets can be specified by passing it via the parameter alpha, either globally as a scalar or per datapoint. Note that a moderate noise level can also be helpful for dealing with numeric issues during fitting as it is effectively implemented as Tikhonov regularization, i.e., by adding it to the diagonal of the kernel matrix. An alternative to specifying the noise level explicitly is to include a WhiteKernel component into the kernel, which can estimate the global noise level from the data."

So i chose to use additive noise in the form of WhiteKernel, which allows the model to learn the noise level of the data.

```
kernel = 1.0 * RBF(length_scale=0.5, length_scale_bounds=(1e-3, 1e2)) +
WhiteKernel(noise_level=1e-4, noise_level_bounds=(1e-22, 1e2))
```

Listing 5.3: Custom RBF-WN Kernel for the Gaussian Process.

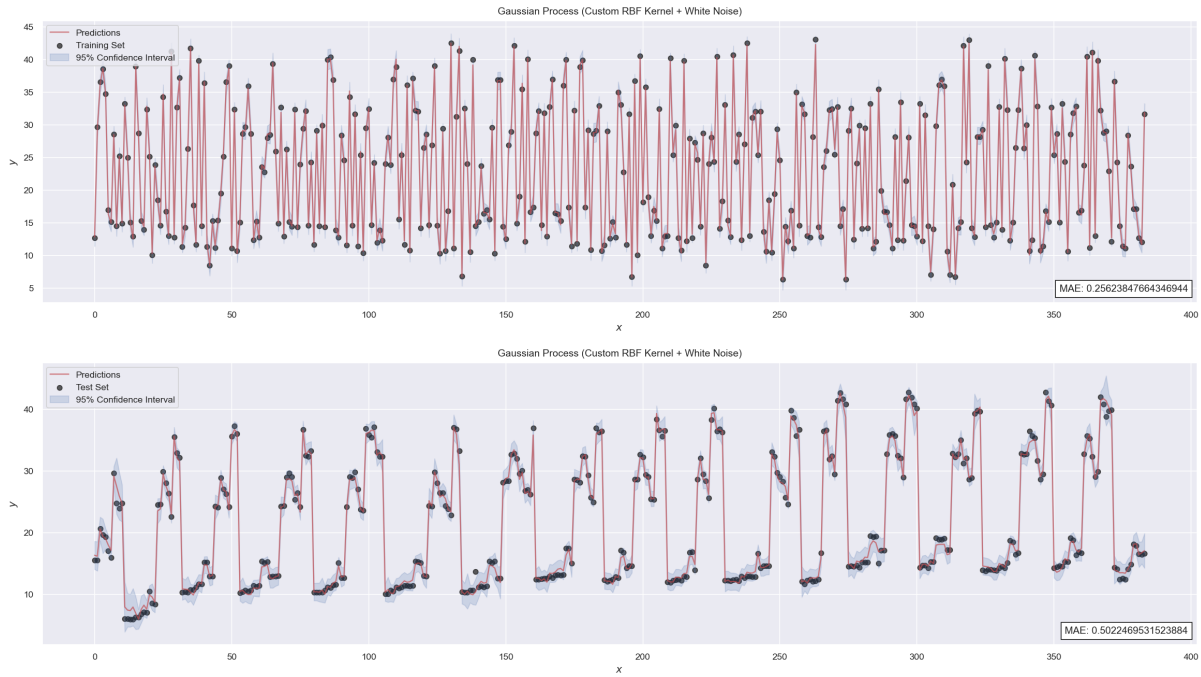

Figure 5.3: The predictions against the training and test set for GP (Custom RBF + WN Kernel).

5.4 Gaussian Process Bayesian Neural Network

For this model I used `tensorflow-probability` (Abadi et al., 2015) to implement a Bayesian Neural Network, where the linear weights are replaced with a probability distribution. The performance is notable worse than the previous method, as initialising the kernel proved to be more difficult than simple additive modes for noise in `sklearn`. I chose not to investigate this further as the model complexity seemed unnecessary, but it could be an avenue for future research.

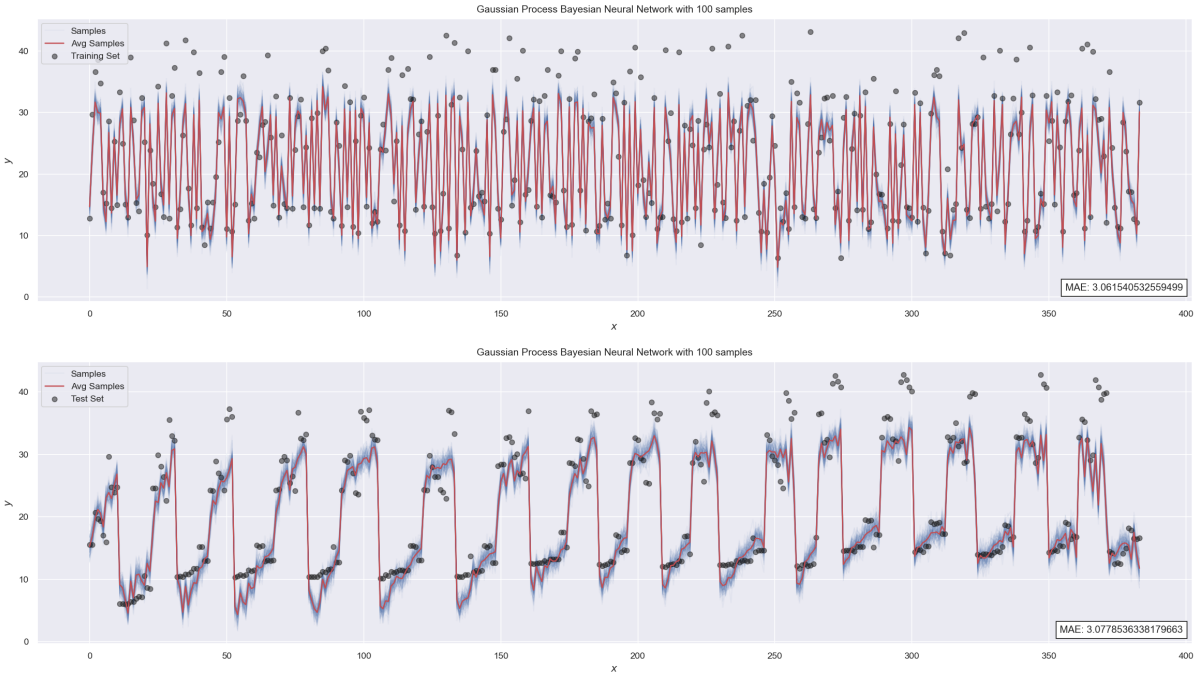

Figure 5.4: The predictions against the training and test set for the BNN.

5.5 Approaches Compared

The initial kernels showed clear signs of overfitting (Table 5.1), as the errors are orders of magnitude apart from the test errors in Table 5.2. Adding noise to the kernel allowed the model to properly generalise and avoid memorising the training set.

Table 5.1: The RMSE and MAE of the training set for GP.

Metric	Default Kernel
RMSE	3.826778857177952e-09
MAE	2.2014088977654702e-09
	Custom RBF Kernel
RMSE	1.3779388124263726e-09
MAE	1.1162595685750472e-09
	Custom RBF + WN Kernel
RMSE	0.32481282810488643
MAE	0.25623847664346944
	BNN
RMSE	13.239257476369245
MAE	3.061540532559499

Table 5.2: The RMSE and MAE of the test set for GP.

Metric	Default Kernel
RMSE	1.8900458253349346
MAE	1.1677819263423648
	Custom RBF Kernel
RMSE	10.924232516393255
MAE	8.312543062134372
	Custom RBF + WN Kernel
RMSE	0.6718571298232691
MAE	0.5022469531523884
	BNN
RMSE	13.432235335150008
MAE	3.0778536338179663

Chapter 6

Results

The results across all models shown in Table 6.1 for the mean average error, suggest that the most accurate model is the Gaussian Process with an RBF + WN kernel. The others have very similar performance, with Hamiltonian Monte Carlo having the second best accuracy likely due to the sampling allowing the most optimal alpha and beta to be found.

The Bayesian Neural Network has a similar mean average error to the other models, but a much higher root mean squared error in Table 6.2, suggesting this model is not well optimised and has large outliers.

Table 6.1: The MAE for all fitted and evaluated models.

Model	Train Error (MAE)	Test Error (MAE)	Absolute Difference
OLS	2.1306794414069143	2.069010093808354	0.06166934759856035
BLR	2.130668537391788	1.9907650699570147	0.13990346743477322
VI	2.1306681834494188	2.0689828677240825	0.06168531572533631
HMC	2.160597295563628	2.103821458477387	0.05677583708624079
GP	0.25623847664346944	0.5022469531523884	0.246008476508919
BNN	3.061540532559499	3.0778536338179663	0.01631310125846719

Table 6.2: The RMSE for all fitted and evaluated models.

Model	Train Error (RMSE)	Test Error (RMSE)	Absolute Difference
OLS	3.0115517876503617	2.8435880167333694	0.16796377091699233
BLR	3.011551809679529	2.8063010631019174	0.20525074657761166
VI	3.0115518111318993	2.8435839397385267	0.16796787139337255
HMC	3.018637231485769	2.852187436445753	0.16644979504001567
GP	0.32481282810488643	0.6718571298232691	0.34704430171838263
BNN	13.239257476369245	13.432235335150008	0.1929778587807629

The model with the smallest absolute difference between the train and test set for RMSE and MAE was OLS, which suggests it is fairly well generalised with fewer outliers than the other models. The MAE is robust to outliers, thus gives us a better idea of the model generalization without noise.

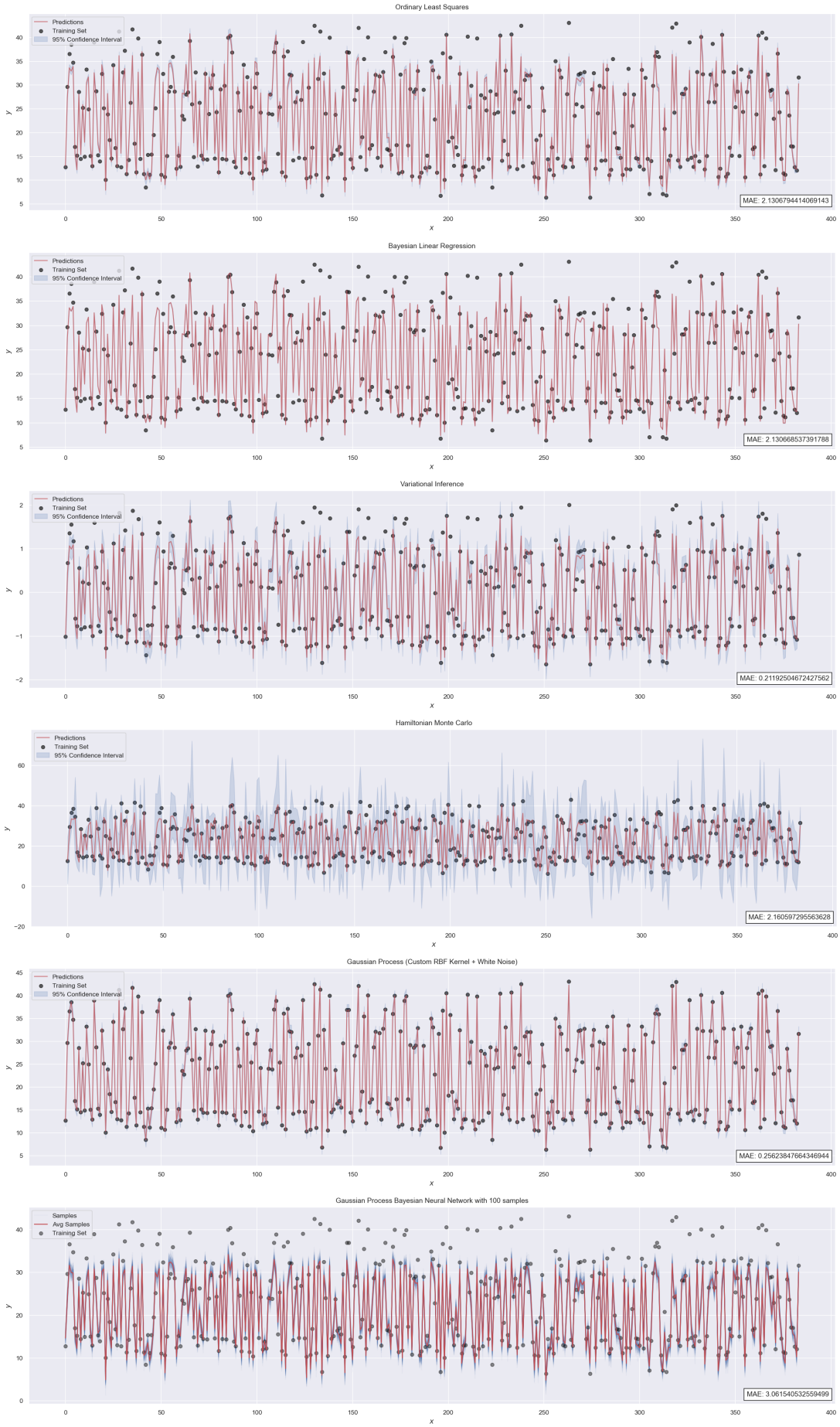

Figure 6.1: The predictions against the training set for all models.

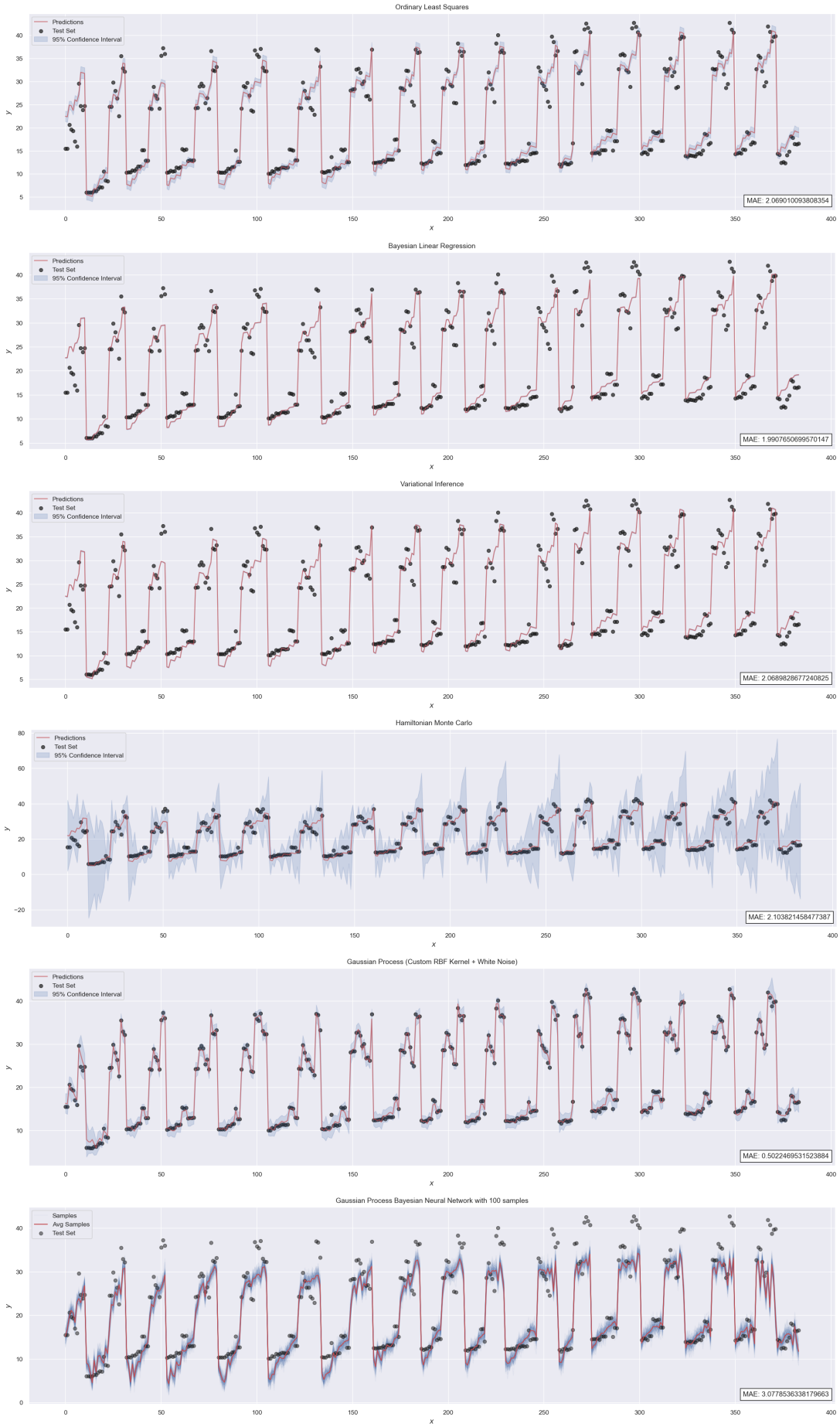

Figure 6.2: The predictions against the test set for all models.

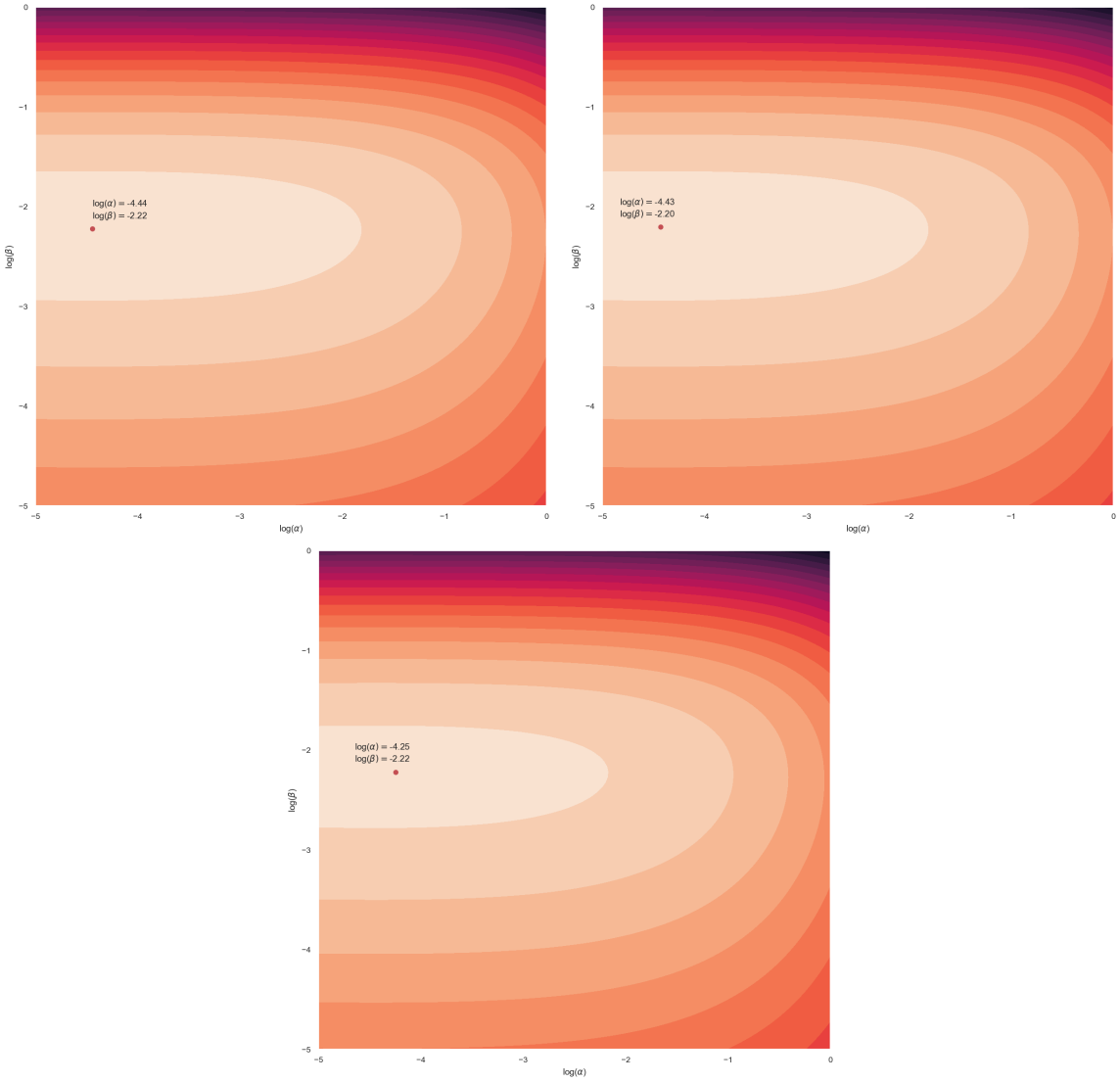

Figure 6.3: The posterior for BLR, VI and HMC with best α and β values.

Chapter 7

Conclusion

Figure 4.4 demonstrates that there is a bias in the distribution on the x and y axes, where the skew is higher along x. This deviation suggests that the dataset itself is non-gaussian, as opposed to Figure 3.5, and linear models are potentially suitable for prediction.

Of all the methods used, shown in Figure 6.2, the *Gaussian Process* is the clear winner, likely due to the ability to refine a custom kernel to allow insights into what the distribution might be to be given almost like a prior. Accuracy across the board was quite uniform (Table 6.1), which is to be expected given their model similarities. The advantage of *Variational Inference* was the improved computation time to find the optimal α and β, however this is mitigated by using the *Woodbury* method in Listing 2.2 for computing the posterior for BLR. *Hamiltonian Monte Carlo* had an extra overhead of computation required to sample from the posterior, but this resulted in a slight accuracy boost compared to the previous methods.

The least performing model was the *Gaussian Process Bayesian Neural Network*, but with limited time constraints it became beyond the scope of this analysis. Future work could involve taking a review (Shridhar, Laumann and Liwicki, 2019) of this research area, in particular the kernel used for the network to try and incorporate noise, as was the case for improvement seen in the standard *Gaussian Process*.

We report a mean average error of 0.2562 for our best model, the *Gaussian Process* (Table 6.1), in contrast to 0.51 reported using *Random Forests* by Tsanas and Xifara (2012). Advancements in machine learning algorithm implementation, and the mass adoption of python have allowed the community to create tools and packages to allow much faster prototyping of different models since the initial paper by Tsanas and Xifara (2012).

The difference in the training and test accuracy was taken to assess the level of overfitting and underfitting. The best absolute difference for the *mean average error* (robust to outliers) was found to be *ordinary least squares* and for the *mean squared error* (amplifies the value associated with outliers), the *Gaussian Process*. This suggests that GP had less outliers and thus a better spread of the data compared to the other models.

We have shown that a stochastic approach to this dataset gave us access to confidence intervals, of which a comparison is illustrated in Figure 6.2. We can see that in general the confidence interval in the *Gaussian Process* has a much smaller range, thus we can be more confident in the predicted values for this model.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems [Online]. Software available from tensorflow.org. Available from: https://www.tensorflow.org/.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* [Online], 12, pp.2825–2830. Available from: https://scikit-learn.org/.

Shridhar, K., Laumann, F. and Liwicki, M., 2019. A Comprehensive guide to Bayesian Convolutional Neural Network with Variational Inference [Online]. [Online], pp.1–38. 1901.02731, Available from: http://arxiv.org/abs/1901.02731.

Tsanas, A. and Xifara, A., 2012. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and buildings* [Online], 49, pp.560–567. Available from: https://doi.org/10.1016/j.enbuild.2012.03.003.