Predicting House Heating Load with Bayesian Methods

Jacob Turner
Department of Computer Science, University of Bath

May 17th, 2021

Contents

1 Introduction

A Bayesian linear regression model is applied to predict the heating load of a house based on eight key characteristics. This includes: 'Relative Compactness', 'Surface Area', 'Wall Area', 'Roof Area', 'Overall Height', 'Orientation', 'Glazing Area', and 'Glazing Area Distribution' of the house. In addition, an initial exploration of the data was conducted and a standard linear regression model was applied to predict the target variable as a baseline of performance. Multiple methods were then used to estimate the optimal hyper-parameters for the regression model. Lastly, the Hamiltonian Monte Carlo method was applied to a linear regression and logistic regression model after validating its use on a designed Gaussian distribution.

2 Exploratory Analysis

Initially, based solely on intuition, the task seems feasible. It makes sense that the required power to heat a home would correlate highly to many of the input variables. Specifically, the size and area of the house may seem to have the most predictive power when estimating the homes heating load. However, some of the input parameters may be more relevant than others. Specifically, the house orientation, glazing area, and glazing area distribution seem less relevant and may hold less predictive power. To investigate this hypothesis, a heat-map of the correlation between input variables and the target parameter was implemented, as shown in Figure 1.

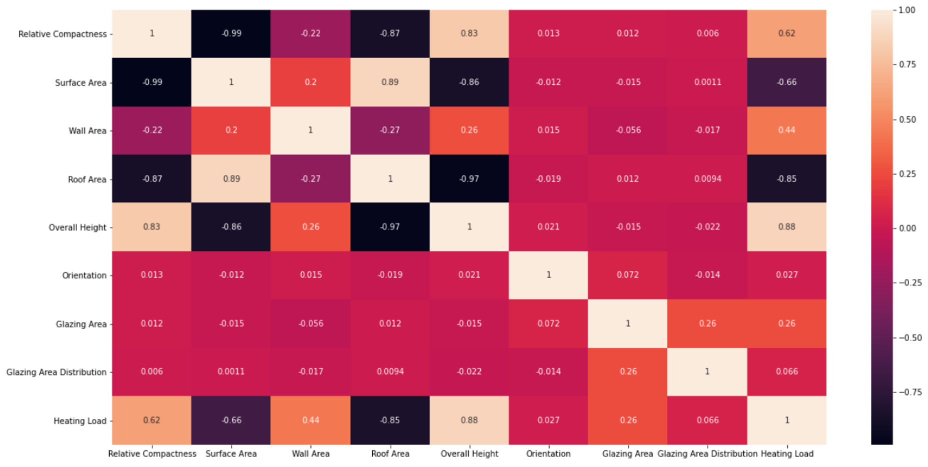

Figure 1: Correlation heat map of the input and target variables.

One can see, the variables in question have a low correlation to the target variable and may be less useful when applying a linear model. This can be investigated further by visualizing the relationship between the input variables and the target as well as their distributions, as shown in Figure 2. This also helps to demonstrate the linearity of the separation between the target variable and each input, though this likely requires a higher dimension. This shows the target variable is likely to be linearly separable and thus a linear regression model would likely produce fairly high predictive performance. This is demonstrated in Figure 8, which shows the train and test results of a standard multiple linear regression model on the data. Here, the coefficient of determination for the linear model on the training and testing data was 0.910, and 0.905, respectively. Alternatively, the Root Mean

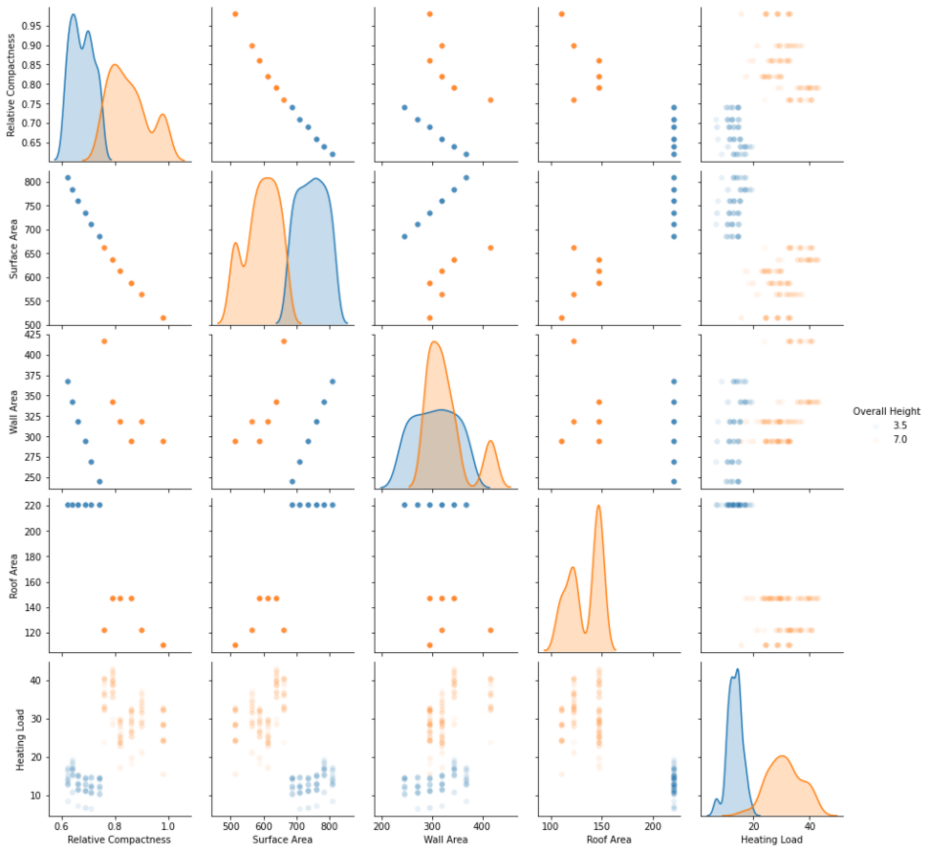

Figure 2: Visualization of the highly correlated features with the target variable and their distributions.

Square Error (RMSE) was 3.0959 and 3.0116. The performance of the linear regression model is fairly high and indicates that the data is likely to be linearly separable.

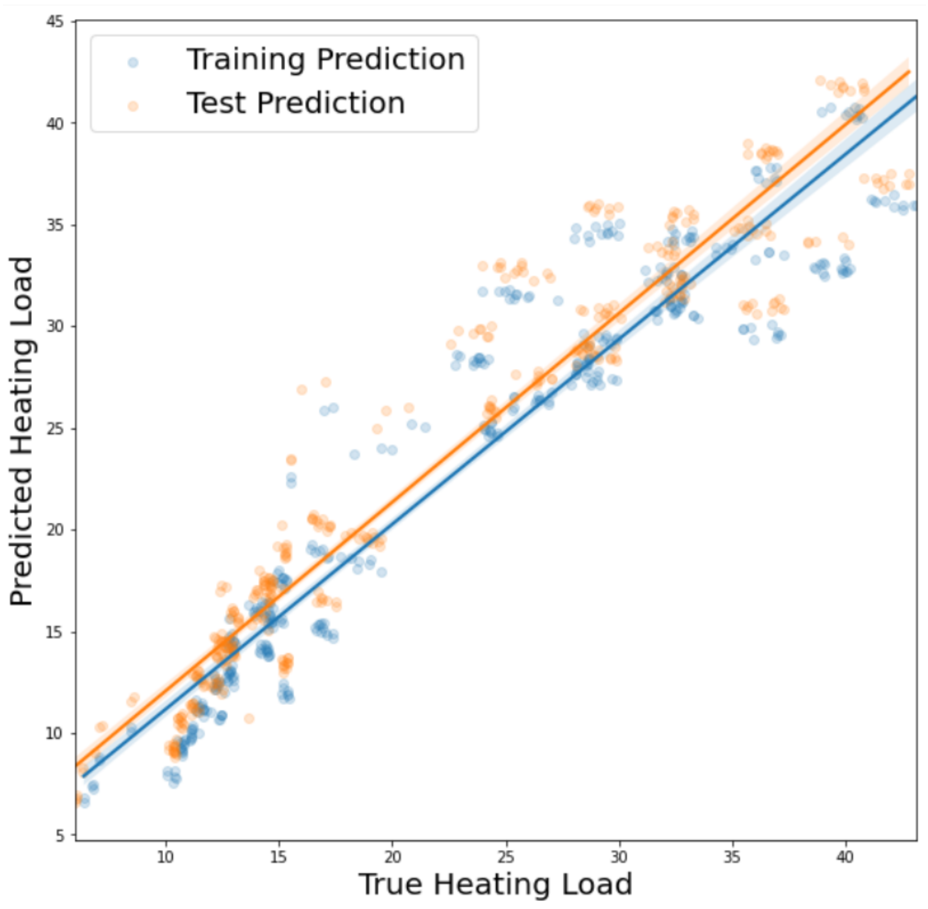

Figure 3: Results of the standard linear regression model applied to the input variables to predict heating load.

3 Bayesian Linear Regression

Bayesian linear regression was applied to the input variables to predict the target, heating load of a house. This model assumes a relationship between the input and target can be modelled with the application of some learned weights w_m to a basis function of x ϕ_x, as shown in equation 1 (Chen, Lecture, 2021).

$$y(x;W) = \sum_{m=1}^{M} w_m \phi_m(x) \tag{1}$$

In this case, the basis function is chosen to be linear ($\phi(x) = x$) because it is more suitable for the higher dimensionality of the problem at hand. The target variable t_n, can then be modeled as the application of these learned weights to the input variables x_n and some random noise or uncertainty ϵ_n (Chen, Lecture, 2021). This relationship is shown in equation 2.

$$t_n = W \cdot x_n + \epsilon_n \tag{2}$$

Penalized Least-Squares (PLS) optimization can be used to find the optimal weights that model the relationship between the training input data and the target data. This can be summarized according to equation 3, where λ refers to a regularization term that can adjust for over-fitting.

$$W_{pls} = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T t \tag{3}$$

The results of the application of the learned weights to the training and test data are shown in Figure 4. The RMSE of this model was about 3.0959, while the coefficient of determination was found to be about 0.905, which produced similar results compared to the standard linear regression model.

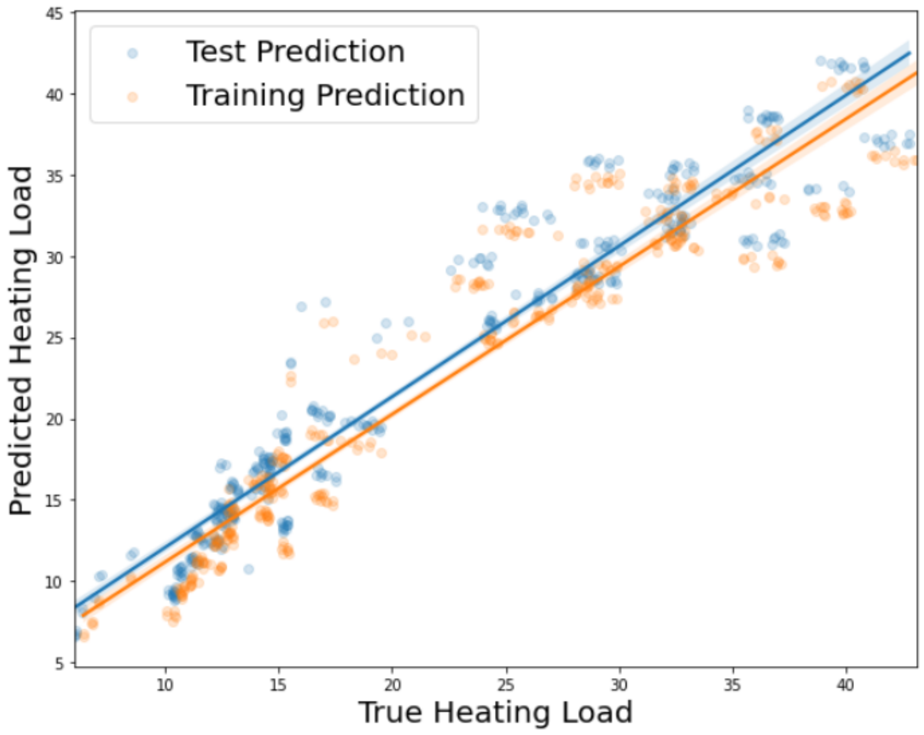

Figure 4: Application of Bayesian linear regression with a linear basis function on the train and test data.

Type-II maximum Likelihood was used to estimate the optimal hyper-parameters (α, σ^2) for the Bayesian regression model. The resulting contour plot of the posterior distribution is shown in Figure 5. This method estimated the most probable hyper-parameters to be $\alpha = 0.0112$, and $\sigma^2 = 9.135$. The results of the application of these optimal hyper-parameters from type-II estimation is shown in Figure 6. The resulting coefficient of determination, and RMSE is 0.907, and 3.067, respectively. These results are slightly better than the standard linear regression model, however the difference is negligible.

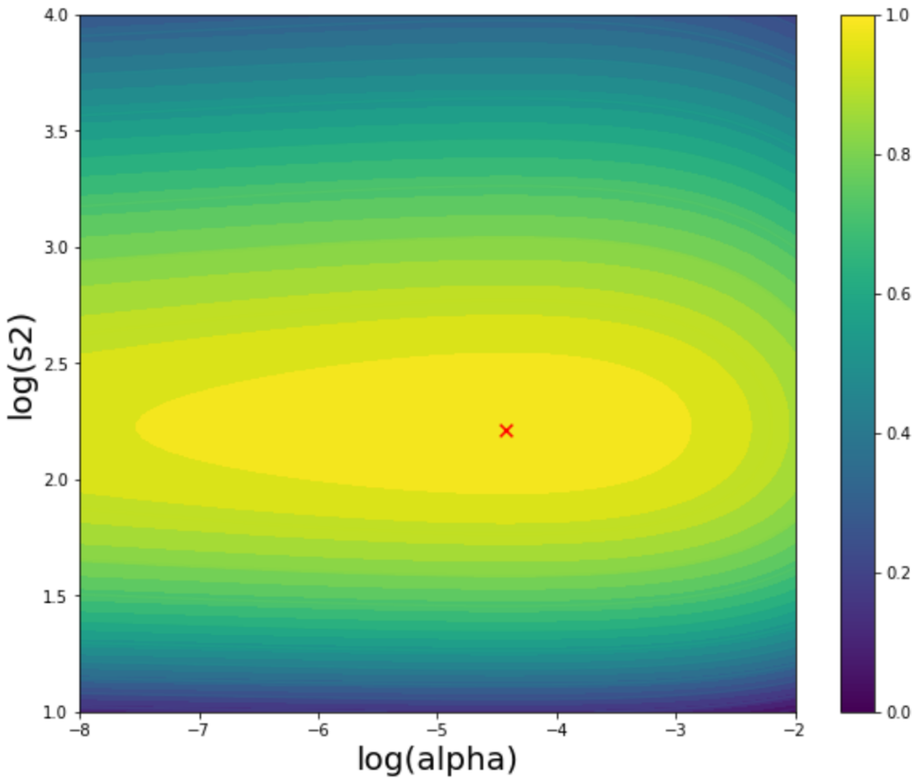

Figure 5: Visualization of the posterior distribution using type-II maximum likelihood estimation for most probable hyper-parameters of Bayesian linear regression model.

4 Hamiltonian Monte Carlo on a Gaussian Distribution

Hamiltonian Monte Carlo (HMC) was applied to sample from a two-dimensional correlated Gaussian distribution, shown in Figure 7. This was a validation exercise for later use with a Bayesian linear regression model. This procedure involves utilizing two functions 'energy func' and its gradient 'energy grad', which are passed to the HMC sampling function for optimal hyper-parameter estimation. In this case, the energy function will be the bi-variate Gaussian distribution using Python Scipy Stats library, and the gradient will be derived manually. The Python implementation for the energy function is shown in code listing 1 below.

```python
def energy_func(x):
    # input mean (mu), and covariance (cov) as parameters to gaussian
    rv = stats.multivariate_normal(mu, cov)

    # return sampled value at provided coordinates
    return -rv.logpdf(np.dstack((x[0],x[1])))
```

Listing 1: Python energy function to sample from a bi-variate Gaussian normal.

The equation for a bi-variate Gaussian distribution is shown in equation 4 [1].

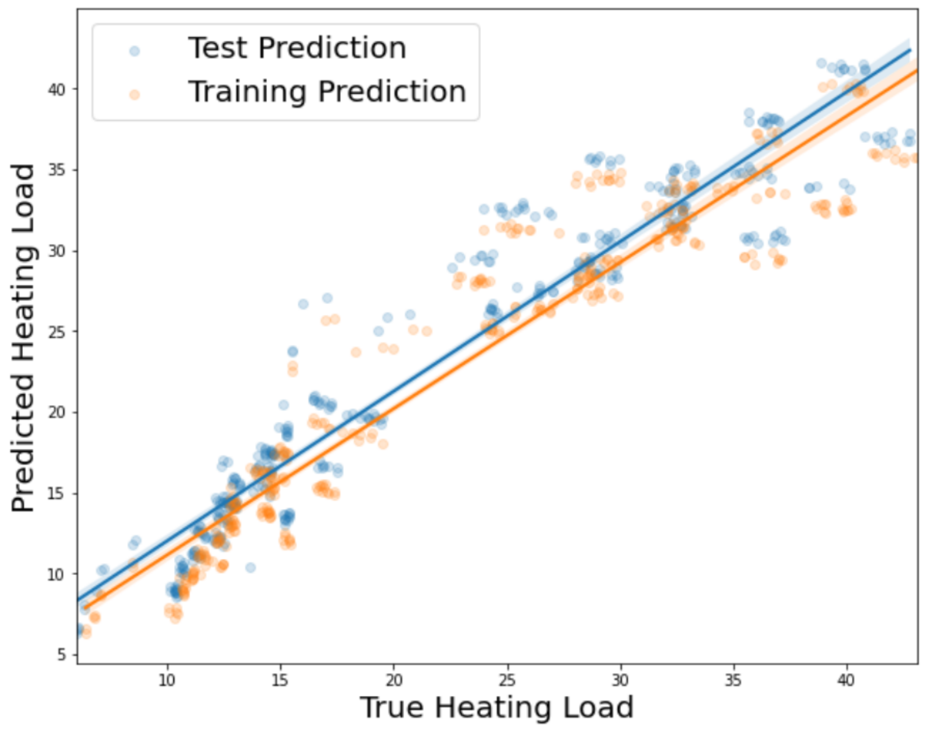

Figure 6: Bayesian linear regression results using the most probable hyper-parameters from type-II estimation.

$$P(x_1, x_2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} exp\left[\frac{-z}{2(1-\rho^2}\right] \tag{4}$$

where,

$$z = \frac{(x_1-\mu_1)^2}{\sigma_1^2} - \frac{2\rho(x_1 = \mu_1)({}_2 - \mu_2)}{\sigma_1\sigma_2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2} \tag{5}$$

The negative log of equation 4 is performed to simplify the gradient calculation. This is shown in equation 6, where ρ refers to the coefficient of determination, or the correlation between x_1 and x_2.

$$-log(P(x_1, x_2) = \left[\frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}}\right] + \frac{1}{2(1-\rho^2)}\left[\frac{(x_1-\mu_1)^2}{\sigma_1^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2}\right] \tag{6}$$

The partial derivative of equation 6 with respect to x_1 and x_2 is calculated, shown in equations 7 and 8.

$$-\frac{\partial log(P(x_1, x_2)}{\partial x_1} = \frac{1}{2(1-\rho^2)}\left[\frac{2(x_1-\mu_1)}{\sigma_1^2} - \frac{2\rho(x_2-\mu_2)}{\sigma_1\sigma_2}\right] \tag{7}$$

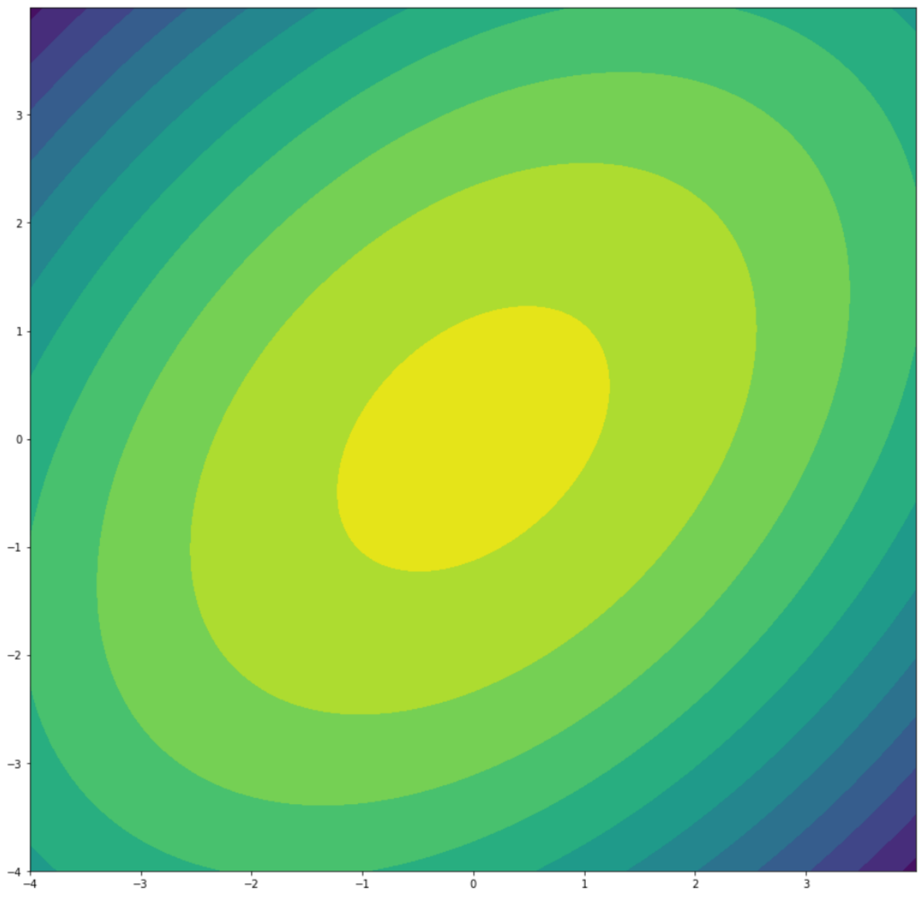

Figure 7: Gaussian normal distribution with a correlation $\rho = 0.4$ between x_1 and x_2.

$$-\frac{\partial log(P(x_1, x_2)}{\partial x_2} = \frac{1}{2(1-\rho^2)}\left[\frac{2(x_2-\mu_2)}{\sigma_2^2} - \frac{2\rho(x_1-\mu_1)}{\sigma_1\sigma_2}\right] \tag{8}$$

In addition, the following co-variance matrix, equation 9, was used to create the normal distribution shown in Figure 7.

$$\Sigma = \begin{pmatrix} \sigma_{x_1}^2 & \rho\sigma_{x_1}\sigma_{x_2} \\ \rho\sigma_{x_1}\sigma_{x_2} & \sigma_{x_2}^2 \end{pmatrix} \tag{9}$$

Finally, the gradient was programmed in a Python function (listing 2), the result of which is shown in Figure 7, where each partial derivative was assigned as an element in a Numpy array.

```python
def energy_grad(x):
    g = np.empty(2)

    F = 1/(2*(1-rho**2))

    # partial wrt x1 and x2 for bivariate gaussian
    g[0] = (F*( 2*(x[0] - mu[0])/(s0**2) - 2*rho*(x[1] - mu[1])/(s0*s1) ))
    g[1] = (F*( 2*(x[1] - mu[1])/(s1**2) - 2*rho*(x[0] - mu[0])/(s0*s1) ))

    return g
```

Listing 2: Energy function

A sample acceptance rate of 82.9% was achieved using 10,000 samples R, a step size ϵ_0 of 1.1, with a burn in rate of 10%, and with 25 cycles L. The resulting Figure 8, shows the accepted samples from the bi-variate Gaussian, along with an over-layed contour plot of the distribution.

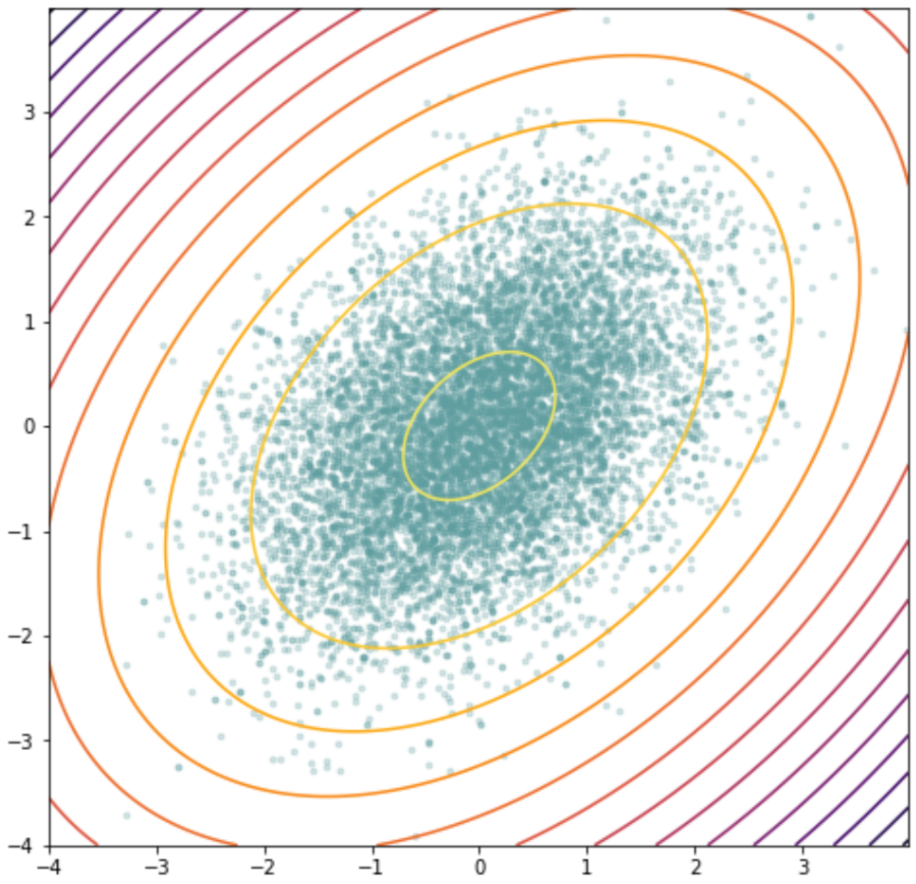

Figure 8: Resulting plot from the HMC sample function depicting the accepted samples from the bi-variate Gaussian.

5 HMC Applied to Linear Regression

HMC was applied to Bayesian linear regression to estimate the optimal parameters W, α, and σ^2 to predict the heating load based on the housing characteristics. This technique uses similar principles as the previous Gaussian example including the implementation of an energy function, which is used to sample from the model likelihood and an energy gradient which returns the partial derivatives for each sampled parameter. The likelihood, shown in equation 10, is derived from a Gaussian distribution, alternatively, the prior over the weights W is shown in equation 11.

$$p(t|W,\sigma^2) = (2\pi\sigma^2)^{-N/2} \prod_{n=1}^{N} exp\left[\frac{t_n - x_n \cdot W}{2\sigma^2}\right] \tag{10}$$

$$p(W|\alpha) = \prod_{m=1}^{M} \left[\frac{\alpha}{2\pi}\right]^{1/2} exp\left[-\frac{\alpha}{2} w_m^2\right] \tag{11}$$

The negative log of equations 10 and 11 was performed to simplify the gradient calculation resulting in equations 12 and 13.

$$-logp(t|W,\sigma^2) = -log[(2\pi\sigma^2)^{-N/2}] + \sum_{n=1}^{N} \frac{[t_n - x_n \cdot W]}{2\sigma^2} \tag{12}$$

$$-logp(W|\alpha) = -log\left[\frac{\alpha}{2\pi}\right]^{1/2} + \frac{\alpha}{2}\sum_{m=1}^{M} w_m^2 \tag{13}$$

The product of the prior and the likelihood, after the natural log is applied, becomes the sum of equations 12 and 13 as shown in equation 14 below.

$$-logP_{posterior} = -logp(t|W,\sigma^2) - logp(W|\alpha) \tag{14}$$

The final energy function can be seen implemented in a Python function in code listing 3, where the first 9 elements of an array are assigned weight values W, and the last two are σ^2, and α, respectively.

```python
def energy_func(x, x_train, y_train):

    # gather weights, noise, and alpha parameters
    w = x[:9]
    s2 = x[9]
    alpha = x[10]

    # width and length of data
    M = x_train.shape[1]
    N = x_train.shape[0]

    # compute posterior mean
    y_post = x_train @ w

    # compute liklihood
    liklihood = -(N/2.0)*np.log( (2.0*np.pi*s2**2) ) \
                - (np.sum((y_train - y_post)**2.0)/(2*s2**2) )

    # compute prior
    prior = (M/2.0)*np.log( (alpha/(2.0*np.pi)) ) \
            - ((alpha/2)*np.sum(w**2.0))

    return -(liklihood + prior)
```

Listing 3: Python energy function which defines the sampling distribution for the likelihood and prior.

The partial derivatives with respect to W, α, and σ^2 is performed on equation 14, the results of which are shown in equations 15, 16, and 17. Where, W is a 9 dimensional vector for each housing characteristic variable with an additive bias to the regression model, to provide another degree of freedom.

$$-\frac{\partial logP_{posterior}}{\partial W} = \sum_{n=1}^{N} \frac{x_n[t_n - x_n \cdot W]}{2\sigma^2} + \alpha W \tag{15}$$

$$-\frac{\partial logP_{posterior}}{\partial \alpha} = -\frac{M}{2\alpha} + \sum_{m=1}^{M} \frac{w_m^2}{2} \tag{16}$$

$$-\frac{\partial logP_{posterior}}{\partial \sigma^2} = \frac{N}{s\sigma^2} - \sum_{n=1}^{N} \frac{[t_n - x_n \cdot W]^2}{2(\sigma^2)^2} \tag{17}$$

The Python implementation for the analytical gradient with respect to these parameters is shown below in code listing 4.

```python
def energy_grad(x, x_train, y_train):

    # define empty array
    g = np.empty(11)

    # collect weights, noise, and alpha parameters
    w = x[:9]
    s2 = x[9]
    alpha = x[10]

    M = x_train.shape[1]
    N = x_train.shape[0]

    y_post = x_train @ w

    # compute gradient for each weight in loop
    for i in range(x_train.shape[1]): # grad weights W
        g[i] = (-1/(2*s2**2))*np.sum( 2*x_train[:,i]*(y_train - y_post) )  + alpha*w[
    i]

    # compute noise and alpha gradients respectively
    g[9] = N/(s2) - (np.sum((y_train - y_post)**2.0) / (s2**3) ) # grad noise (s2)
    g[10] = (-M/(2*alpha) + np.sum(0.5*w**2.0)) # grad alpha

    return g
```

Listing 4: Python gradient of the energy function for the 11 parameters in the bayesian linear regression model.

The resulting parameter estimations from the HMC sampler produced a coefficient of determination of 0.905 and a RMSE of 3.0958 when used in the linear regression model. The hyper-parameters σ^2 and α was found to be 3.05 and 0.0159, respectively. These results were produced by using 10,000 samples R, $L = 100$, and $\epsilon_0 = 0.0026$ with an acceptance of 91.8%. The predicted heating load from this method can be seen plotted against the true heating load below in 10. In addition, a visualization of the accepted HMC samples for the parameters W, α, and σ^2 is shown in Figure 9

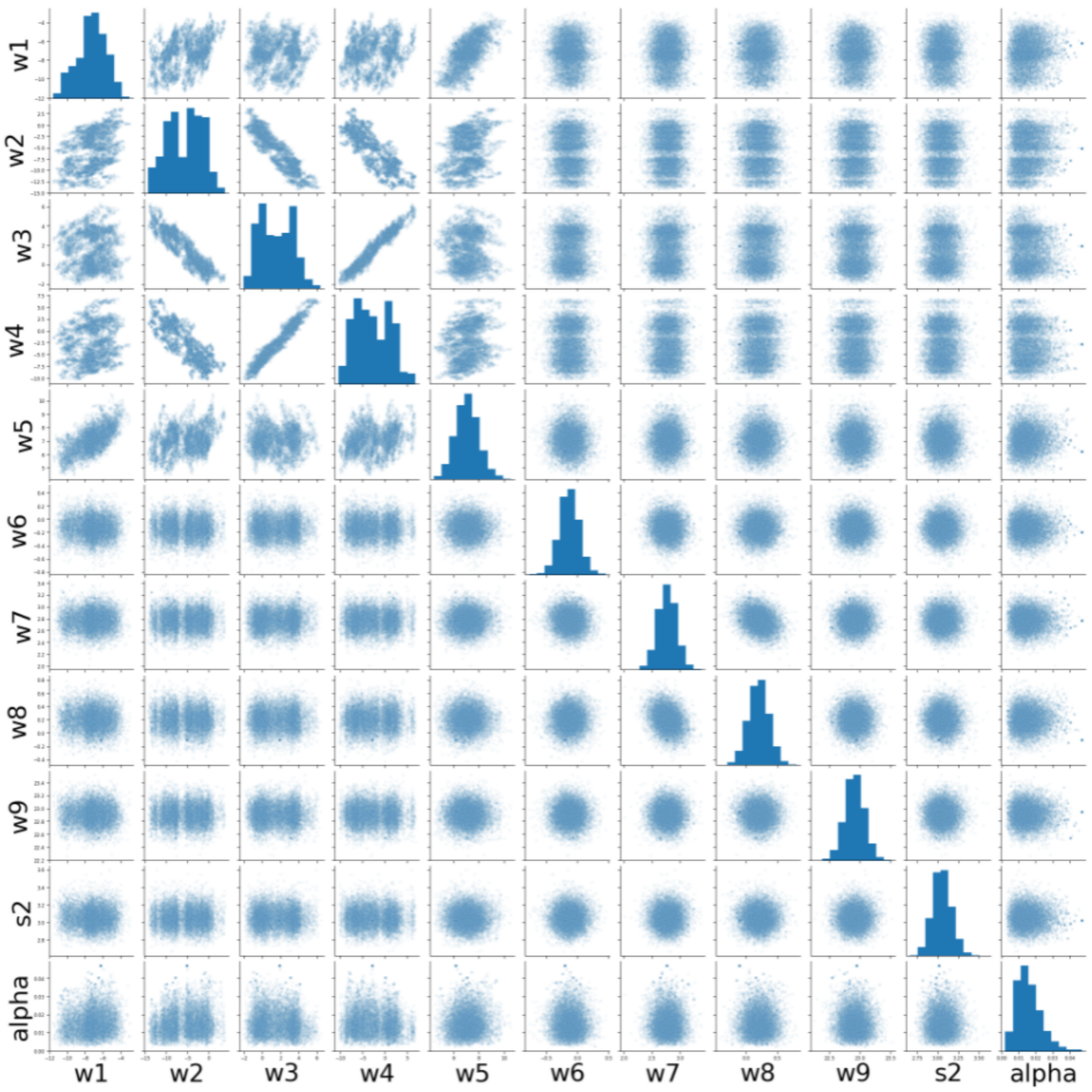

Figure 9: Visualization of the .

6 HMC Applied to Logistic Regression

Finally, HMC was applied to a logistic regression model to estimate the optimal parameters W and α. These were then used to predict the heating load based on the housing characteristics, where a heating load greater than 23.0 is denoted as 'high' and below as 'low'. The same principle's as the previous example were utilized, including the implementation of an energy function, which is used to sample from the model likelihood and an energy gradient which returns the partial derivatives for each sampled parameter. In this case, the Bernoulli likelihood is used with the same normal distribution over the weights, while the mean posterior prediction is linked with a Sigmoid function.

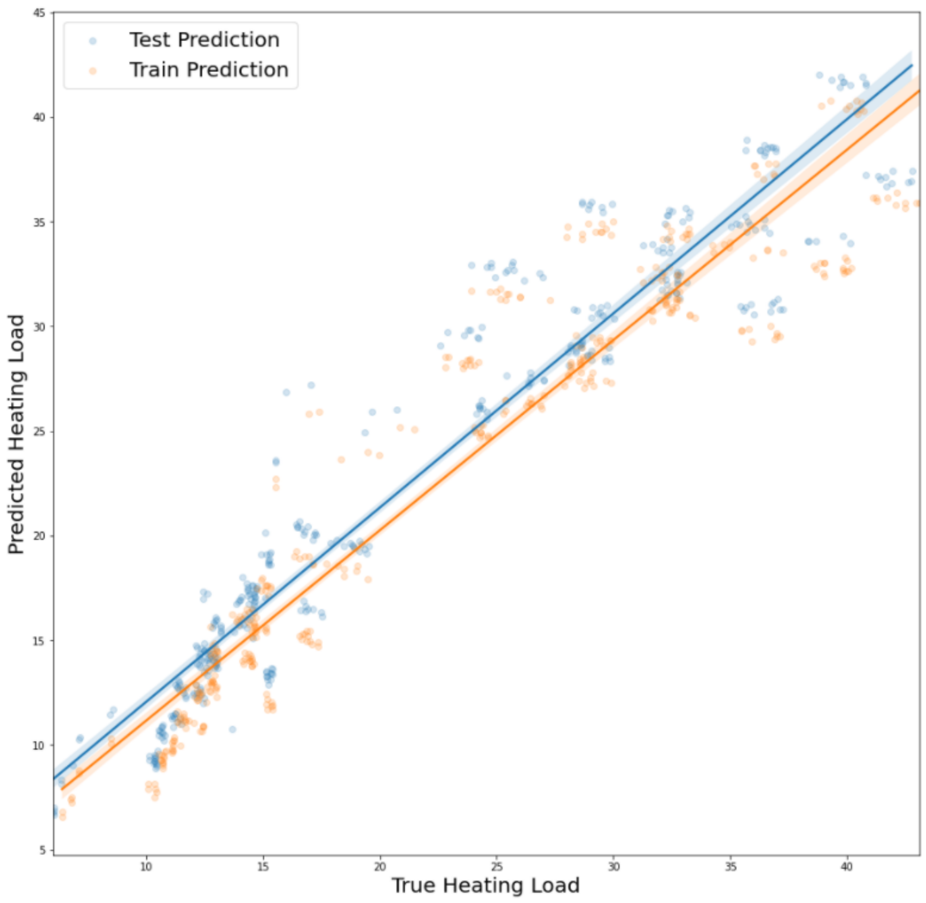

Figure 10: Resulting regression predictions from the HMC sampler using the optimal weights and hyper-parameters.

The likelihood, shown in equation 18, is derived from the Bernoulli distribution with a sigmoid link function, where, the prior over the weights W is also shown in equation 19.

$$p(t|W) = \prod_{n=1}^{N} p_n^y (1-p)^{1-y_n} = \prod_{n=1}^{N} \left[\frac{1}{1+exp(-x_n \cdot W)}\right] \left[\frac{exp(-x_n \cdot W)}{1+exp(-x_n \cdot W))}\right] \tag{18}$$

$$p(W|\alpha) = \prod_{m=1}^{M} \left[\frac{\alpha}{2\pi}\right]^{1/2} exp\left[-\frac{\alpha}{2} w_m^2\right] \tag{19}$$

The negative log of equations 18 and 19 is calculated to simplify the gradient calculation resulting in equations 20 and 21.

$$-logp(t|W) = \sum_{n=1}^{N} x_n \cdot W(y_n - 1) - log(1 + exp(-x_n \cdot W) \tag{20}$$

$$-logp(W|\alpha) = -log\left[\frac{\alpha}{2\pi}\right]^{1/2} + \frac{\alpha}{2}\sum_{m=1}^{M} w_m^2 \tag{21}$$

The product of the prior and the likelihood, after the natural log is applied, becomes the sum of equations 20 and 21 as shown in equation 22 below.

$$-logP_{posterior} = -logp(t|W) - logp(W|\alpha) \tag{22}$$

The energy function, implemented in a Python function is shown in code listing 5, where only the likelihood variable is altered to use the Bernoulli distribution as described previously.

```python
def energy_func(x, x_train, y_train):

    w = x[:9]
    alpha = (x[9])

    M = x_train.shape[1]
    N = x_train.shape[0]

    y_post = x_train @ w

    # neg log of bernoulli
    a = y_post * (y_train - 1)
    b = -np.log( 1 + np.exp(-y_post))

    liklihood = -np.sum(a+b)

    prior = (M/2.0)*np.log( (alpha/(2.0*np.pi)) ) \
            - ((alpha/2)*np.sum(w**2.0))

    return -(liklihood + prior)
```

Listing 5: Python energy function which defines the sampling distribution for the likelihood and prior.

The partial derivatives with respect to W and α is performed on equation 22, the results of which are shown in equations 23 and 24.

$$-\frac{\partial logP_{posterior}}{\partial W} = \sum_{m=1}^{M} x_m y + x_m + \frac{x_m exp(-x \cdot W)}{1 + exp(-x \cdot W)} + \alpha W_m \tag{23}$$

$$-\frac{\partial logP_{posterior}}{\partial \alpha} = -\frac{M}{2\alpha} + \sum_{m=1}^{M} \frac{w_m^2}{2} \tag{24}$$

The implementation for the analytical gradient with respect to these parameters is shown below in Python code listing 6.

```python
def energy_grad(x, x_train, y_train):

    # define empty array
    g = np.empty(10)

    # collect weights, noise, and alpha parameters
    w = x[:9]
    alpha = (x[9])

    M = x_train.shape[1]
    N = x_train.shape[0]

    y_post = x_train @ w

    # compute gradient for each weight in loop from bernoulli
    for i in range(x_train.shape[1]): # grad weights W
        g[i] =  -np.sum(x_train[:,i] * (y_train - 1.0 + np.exp(-y_post)/(1.0 + np.exp
    (-y_post)) ) ) - alpha*w[i]

    g[9] = -(-M/(2*alpha) + np.sum(0.5*w**2.0)) # grad alpha

    return g
```

Listing 6: Python gradient of the energy function for the 11 parameters in the bayesian linear regression model.

HMC sampling applied to logistic regression resulted in a classification accuracy of 97.7%, with an estimated $\alpha = 3.80$. These results were produced by using 10,000 samples R, $L = 100$, and step size $\epsilon_0 = 0.00008$ with an acceptance rate of 88.5 %. A heatmap of the resulting confusion matrix is show in Figure 11 and a visualization of all the HMC samples for the parameters W and α is shown in Figure 12. The results of this model was compared to the standard logistic regression implementation with Sci-Kit Learn Python library. This resulted in the same classification accuracy of 97.7%, validating the HMC implementation for the logistic regression model.

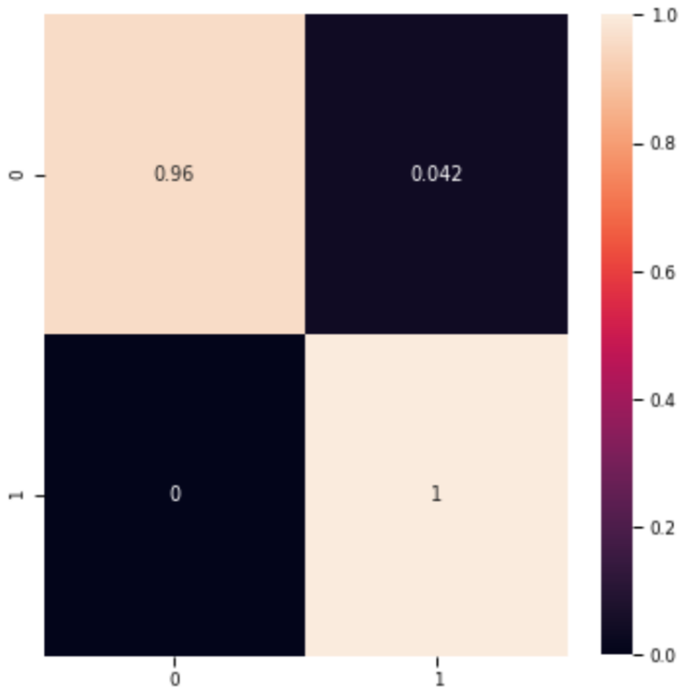

Figure 11: Confusion matrix for the HMC classifiation model.

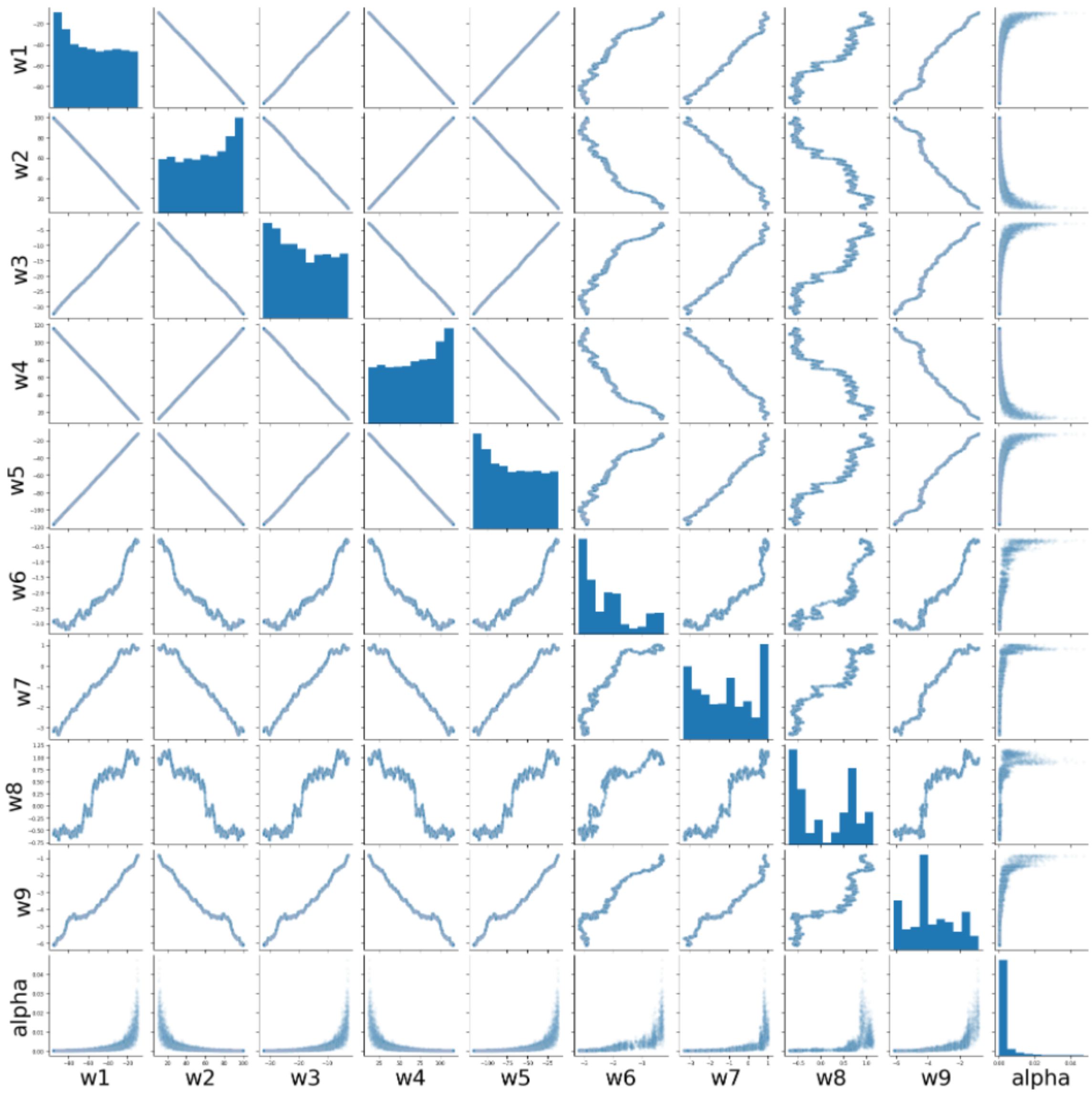

Figure 12: Distribution of accepted samples for model parameter in the logistic regression classifier.

References

[1] Wolfram Alpha. Bivariate normal distribution, 2018.