Data preprocessing

The provided data are standardized using sklearn standard scaler. The data set is check for mean of zero and standard deviation of one as shown in Figure1. The bias is also added to the independent variable x as shown in Figure2.

```
#check mean of zero and std 1
print(np.isclose(np.mean(x_train,axis=0),[0]*x_train.shape[1]).all())
print(np.isclose(np.mean(x_test,axis=0),[0]*x_test.shape[1]).all())
print(np.isclose(np.std(x_train,axis=0),[1]*x_train.shape[1]).all())
print(np.isclose(np.std(x_test,axis=0),[1]*x_test.shape[1]).all())
```

Figure1: Validate zero mean and one standard deviation of standardized data set

```
#add bias to input x
x_train=np.concatenate((x_train,np.ones(x_train.shape[0])[:,np.newaxis]),axis=1)
x_test=np.concatenate((x_test,np.ones(x_test.shape[0])[:,np.newaxis]),axis=1)
```

Figure2: Add bias to the data set

Initial analysis

Not every independent feature x is important in predicting the dependent variable y. Therefore, correlation is the basic way to determine the relationship between a pair of variables of how one variable change with another. Note that correlation does not imply causation. For example, the more you exercise the more you lose weight, but not vice versa. Positive correlation indicates 2 variables change in same direction while negative correlation indicates both of them change in opposite direction. From correlation plot as shown in Figure3, the most un-important variable are orientation, glazing area, and glazing area distribution as their correlation to dependent variable y is very small such that the absolute value is less than 0.3.

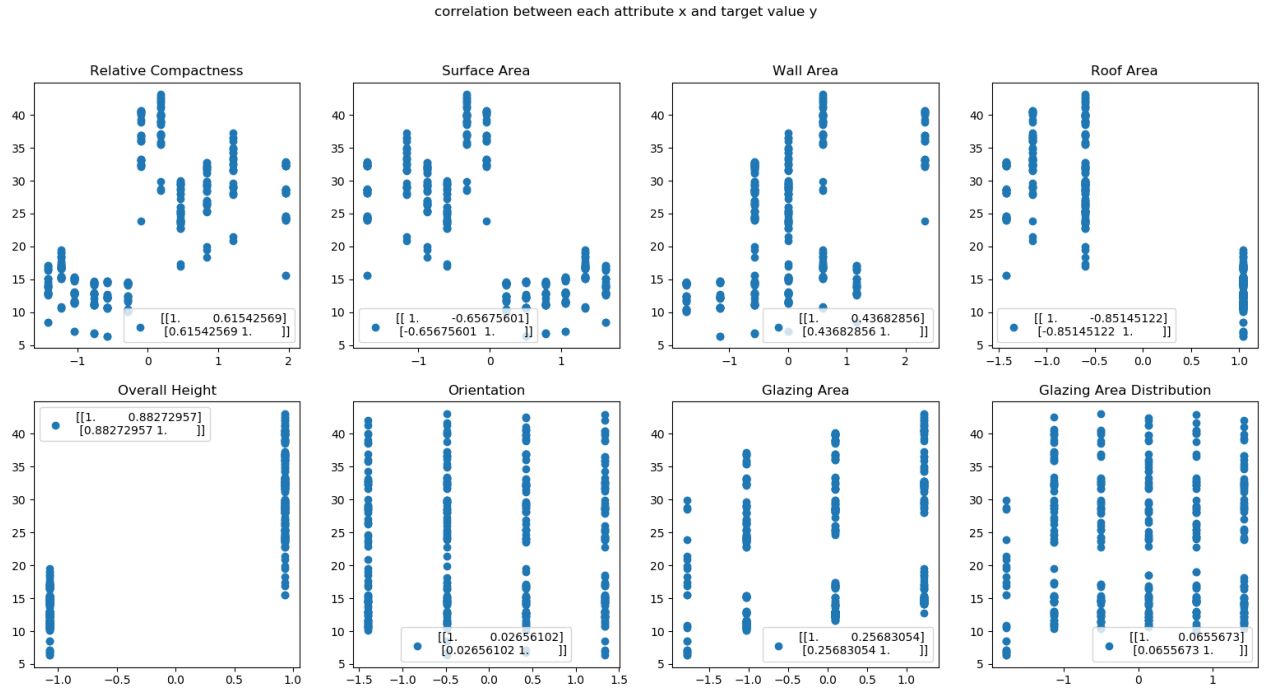
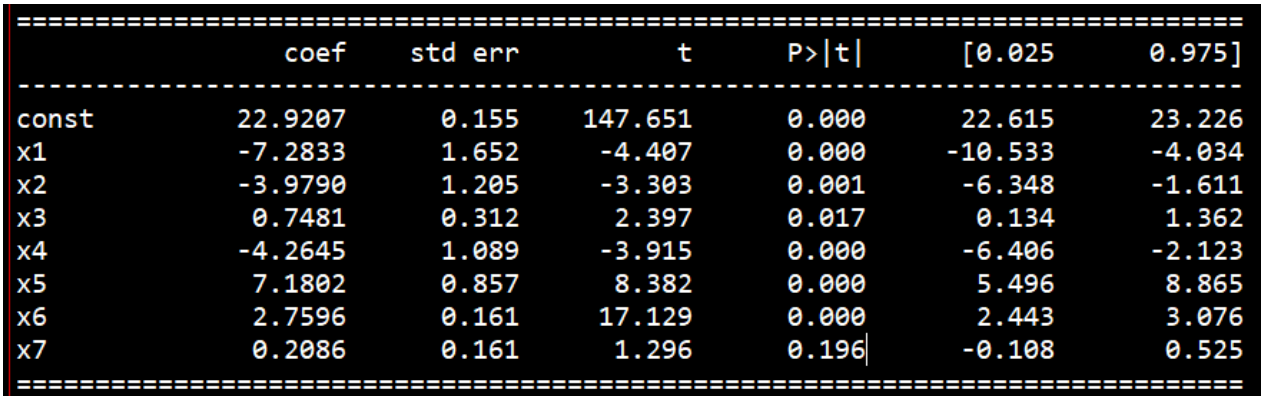


Figure3: Correlation plot between each independent variable x to dependent variable y

Another method is to use selection process such as backward elimination. This process aims to find the set of variables (features) that can best explain the dependent variable y. The process begins with considering all variables and then removing a variable that have P-value higher than significant level one at a time until no variable can be removed. By performing backward elimination with significant level of 0.05, the first variable to be removed was orientation that have P-value of 0.423 shown in Figure4 which is higher than significant level. The second variable was glazing area orientation with P-value of 0.196 shown in Figure5. After these 2 variables have been remove¸ no variable has p-value more than significant level shown in Figure6. This indicate that orientation and glazing area is not that important in a sense that the model can still have a good prediction without these 2 features.

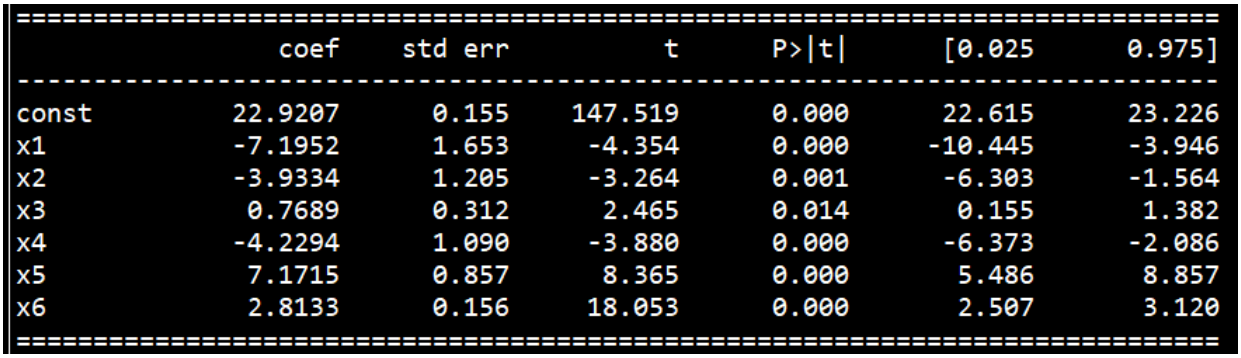
	coef	std err	t	P>|t|	[0.025	0.975]
const	22.9207	0.155	147.581	0.000	22.615	23.226
x1	-7.2346	1.654	-4.373	0.000	-10.488	-3.982
x2	-3.9422	1.206	-3.269	0.001	-6.314	-1.571
x3	0.7560	0.312	2.420	0.016	0.142	1.370
x4	-4.2319	1.091	-3.880	0.000	-6.376	-2.087
x5	7.2040	0.858	8.401	0.000	5.518	8.890
x6	-0.1252	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

Figure4: Backward elimination 1



	coef	std err	t	P>|t|	[0.025	0.975]
const	22.9207	0.155	147.651	0.000	22.615	23.226
x1	-7.2833	1.652	-4.407	0.000	-10.533	-4.034
x2	-3.9790	1.205	-3.303	0.001	-6.348	-1.611
x3	0.7481	0.312	2.397	0.017	0.134	1.362
x4	-4.2645	1.089	-3.915	0.000	-6.406	-2.123
x5	7.1802	0.857	8.382	0.000	5.496	8.865
x6	2.7596	0.161	17.129	0.000	2.443	3.076
x7	0.2086	0.161	1.296	0.196	-0.108	0.525

Figure5: Backward elimination 2



	coef	std err	t	P>|t|	[0.025	0.975]
const	22.9207	0.155	147.519	0.000	22.615	23.226
x1	-7.1952	1.653	-4.354	0.000	-10.445	-3.946
x2	-3.9334	1.205	-3.264	0.001	-6.303	-1.564
x3	0.7689	0.312	2.465	0.014	0.155	1.382
x4	-4.2294	1.090	-3.880	0.000	-6.373	-2.086
x5	7.1715	0.857	8.365	0.000	5.486	8.857
x6	2.8133	0.156	18.053	0.000	2.507	3.120

Figure6: Backward elimination 3

The train and test set error after performing least squared linear regression are shown in Table1 for the case that consider all variable as well as the backward elimination case.

Table1: Result after performing least square linear regression

Case	RMSE train	RMSE test
Consider all features	3.0115517876503612	3.0958865845448686
After remove glazing area and orientation	3.020837151333591	3.110716297089577

Since the data have more than 2 dimensions, the train and test plot accuracy are generated in terms of correlation between actual value of dependent variable y and the predicted value as shown in Figure7.

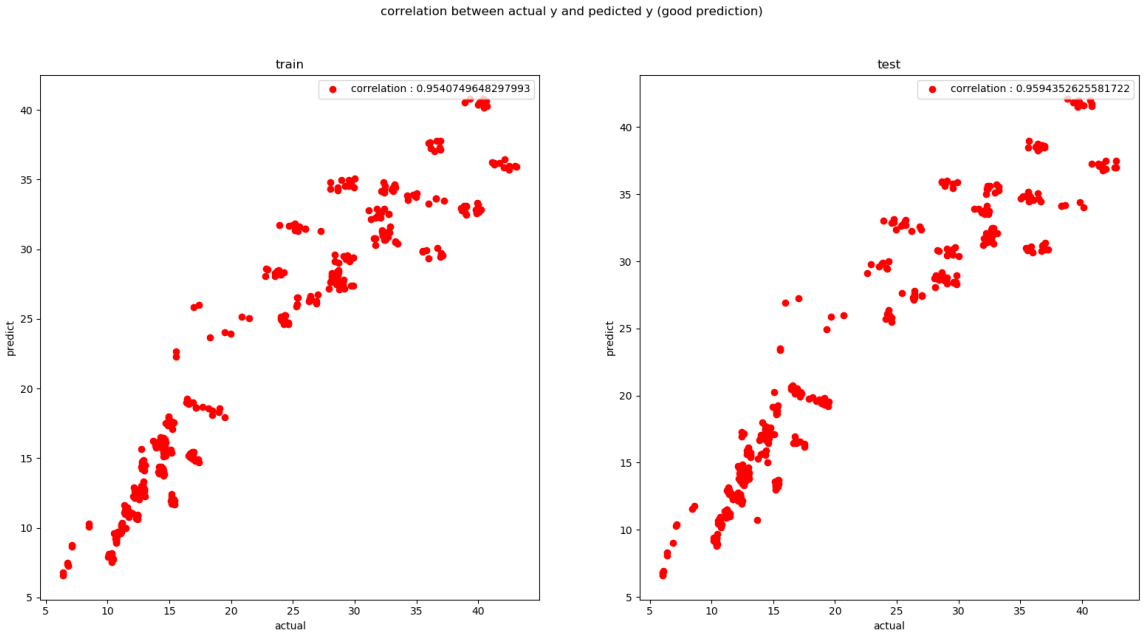


Figure7: Correlation between actual value of dependent variable y and the predicted value

Type-II maximum likelihood

Since the full posterior that we want is p(w,α,s2|D), where α is inverse variance of w and s2 is noise variance, which cannot be compute analytically. The type 2 maximum likelihood can be used to approximate or infer the value of hyper-parameter α and s2. First the full posterior can be re-written as a product of weight posterior and probability of α and s2 given the dataset as shown:

$$p(w,\alpha,s2|y) = p(w|\alpha,s2,y)p(\alpha,s2|y)$$

The weight posterior is normally distributed with mean of $(x^Tx+s2\ \alpha I)^{-1}x^Ty$ and covariance of $s2(x^Tx+s2\ \alpha I)^{-1}$. However, the second probability again cannot be analytically computed. Using the concept of type 2 maximum likelihood¸ the best value of inverse weight variance α and noise variance s2 can be approximate by maximizing the marginal likelihood p(y| α,s2) by assuming flat uninformative prior over log α and log s2. The term marginal likelihood is the probability of dataset given specific hyper-parameter α and s2 where this likelihood is normally distributed with zero mean and covariance of $s2I + \alpha^{-1}xx^T$. To summarize, the procedure of type 2 maximum likelihood is to try a range of value of hyper-parameter α and s2 and the best hyper-parameters are the combination of value that maximize marginal likelihood probability.

```
#marginal log likelihood
def log_marginal_likelihood(x,y,alph,s2):
    cov=s2*np.identity(x.shape[0])+np.matmul((alph**-1)*x,x.T)

    log_p=stats.multivariate_normal.logpdf(y,cov=cov,allow_singular=True)
    return log_p
```

Figure8: Log marginal likelihood function

The marginal likelihood is computed using the function "log_marginal_likelihood" as shown in Figure8 that is the log pdf of normal distribution with zero mean and covariance of $s2I + \alpha^{-1}xx^T$. The range of log alpha and log s2 were chosen to be from -25 to -5 and -10 to 10; respectively, using numpy linspace. The process iterates to all these values of parameter. The estimate value of weights can be obtained from mean of the weight posterior distribution $((x^Tx+s2\ \alpha I)^{-1}x^Ty)$.

After performing type 2 maximum likelihood¸ the max log likelihood obtained is -49.9791556301691 where best log alpha is-5.0 (alpha is 0.006737946999085467) and best log s2 is -10.0 (s2 is 4.5399929762484854e-05) . All parameters are shown in Table2.

Table2: Value of most probable parameters from type 2 maximum likelihood

Parameter	Value
Best alpha (inverse weight variance)	0.006737946999085467
Best s2 (noise variance)	4.5399929762484854e-05
W1	-7.23462662
W2	-3.94213963
W3	0.75594473
W4	-4.23192596
W5	7.2039515
W6	-0.12516927
W7	2.77021895
W8	0.20406264
W9	22.92070311

In order to see the effect of hyper-parameter. 3 cases are conducted to show the importance of choosing good parameter of α and s2. Since regularization parameter (λ) is the product of α and s2, the effect of regularization parameter is shown in Table3 to distinguish between good and bad hyper-parameter selection. If the value of λ is too large, this will lead to underfitting and too small λ lead to over fit. It is cleared that, by optimally chose the value of alpha and s2, this can decrease the RMSE of train and test set as shown in Figure9 and Figure10.

Table3: Demonstrate importance of good hyper-parameter selection

Model	α	s2	RMSE train	RMSE test
Type 2 ML	0.0673	4.53999e-05	3.0116	3.0959
Over fit	0	0	3.0828	3.0977
Under fit	10	10	5.9431	4.81124

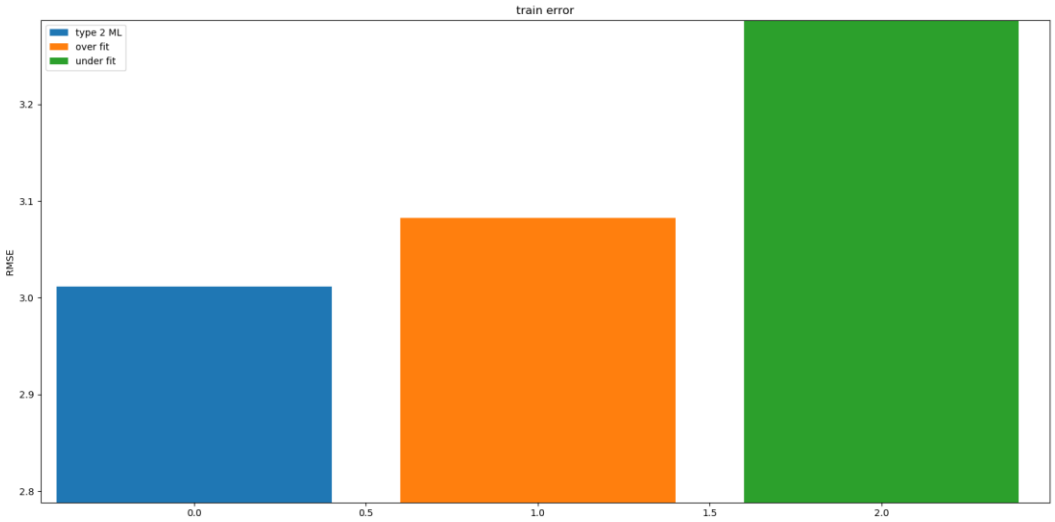


Figure9: Demonstrate importance of good hyper-parameter selection in terms of training set error

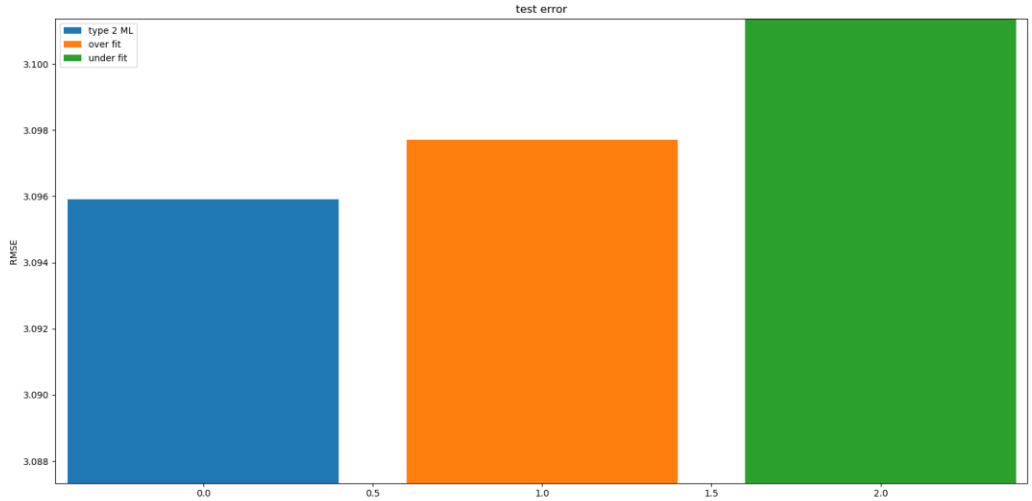


Figure10: Demonstrate importance of good hyper-parameter selection in terms of test set error

The obtained value of log alpha and log s2 is marked on the contour plot with red color dot. The model performed well to achieve maximum log marginal likelihood and marginal likelihood as shown in Figure11, Figure12 and Figure13. However, by visualize the posterior (P(w, α,s2|y) contour plot in Figure14 , the region that the value located is not the best region. The model can do better if the obtained value of log alpha and log s2 fall into yellow region. To conclude, the model does well in terms of maximize marginal log likelihood but not posterior.

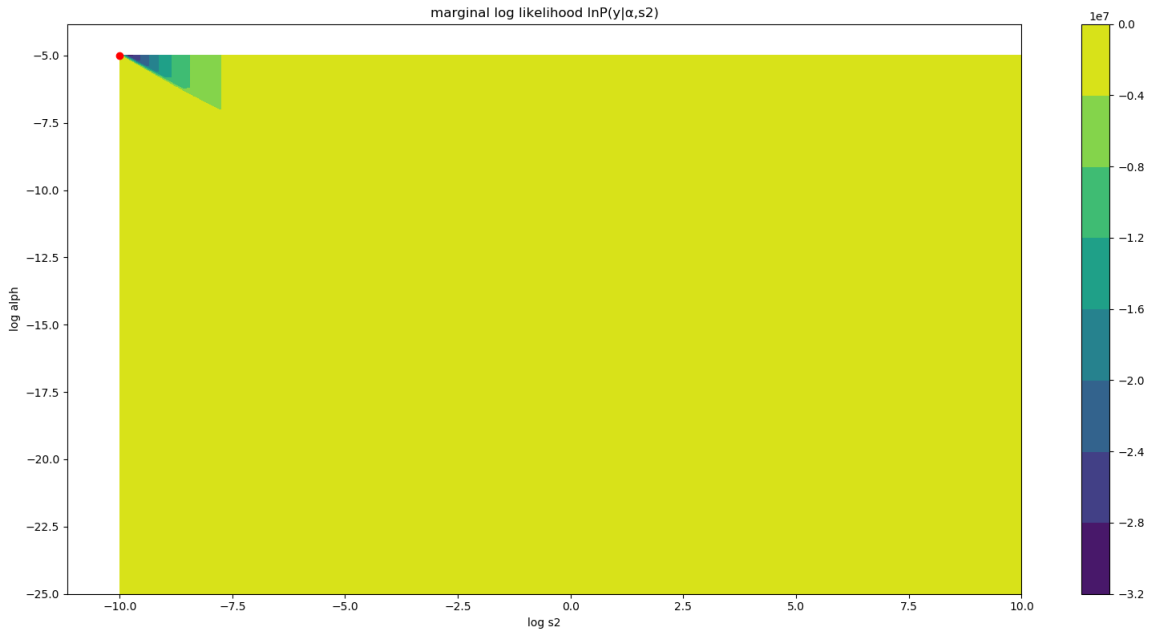


Figure11: Marginal log likelihood

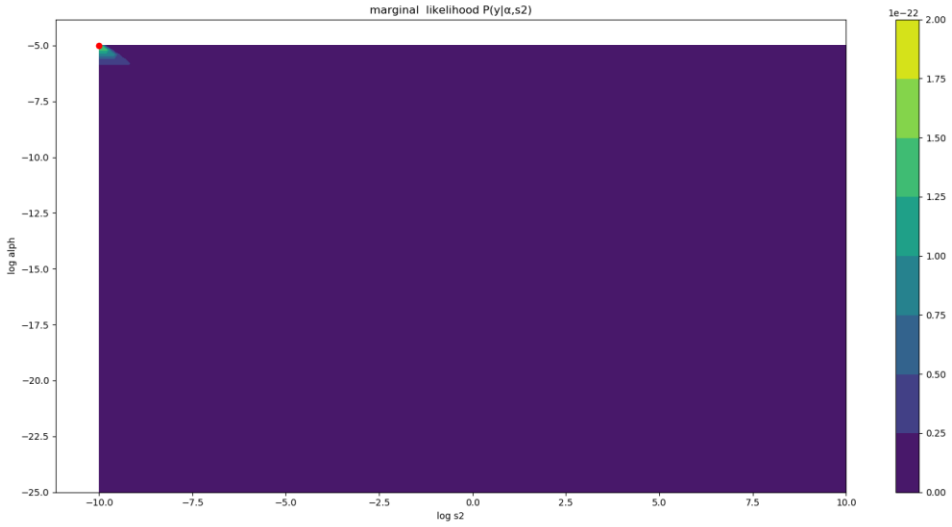


Figure12: Marginal likelihood

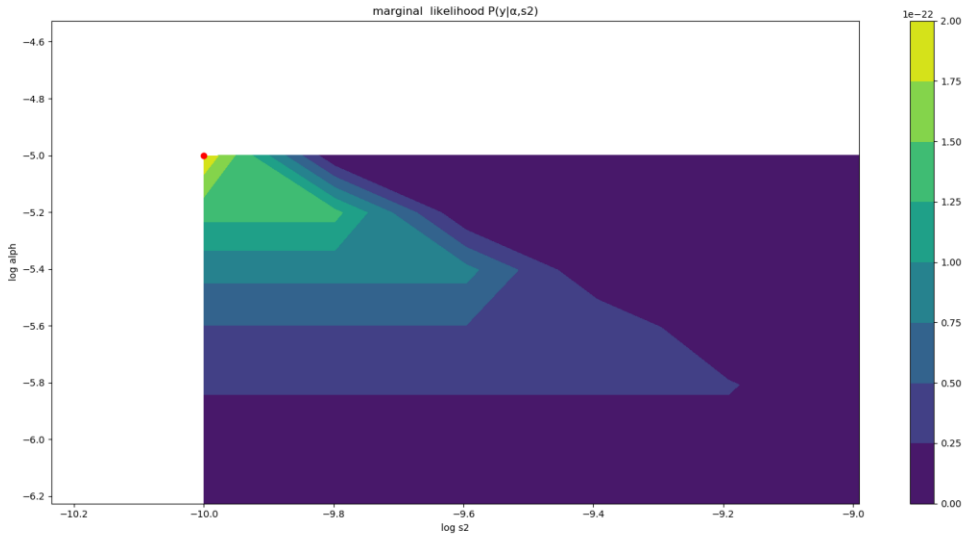


Figure13: Marginal likelihood (zoom in)

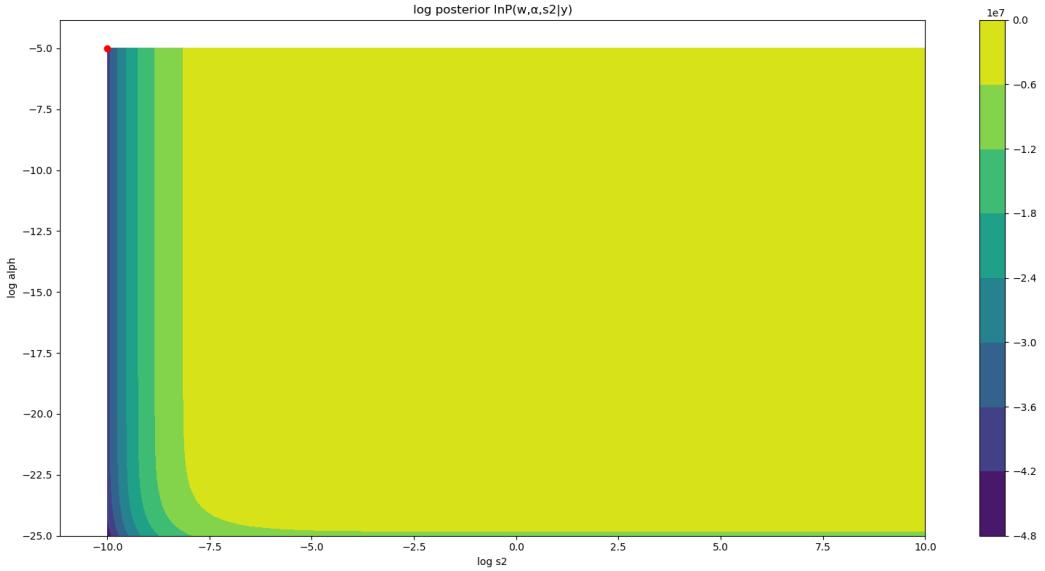


Figure14: Posterior distribution of type 2 maximum likelihood

Variational Inference

Variational method is one of deterministic approximation method. The aim is to find good proposal distribution Q(θ) that is most similar to unknown posterior distribution P(θ). To determine the similarity or difference between these two distributions, KL (Kullback-Leibler) divergence, also known as relative entropy, is used to measure the difference between two distribution based on the observation sample from both distributions. The best case is to have KL divergence to be zero. Later, due to the constant term and since proposal distribution Q(θ) depends on set of hyper-parameters, minimizing KL divergence is

equal to maximizing ELBO (Evidence Lower Bound) which is equal to finding best hyper-parameter set that maximize ELBO.

Mean field theory is applied to avoid complex joint distribution since sometime the problem contains more than one hyper-parameter to consider by breaking down one joint distribution to many small independent distributions where each distribution depends only on single hyper-parameter.

The optimal proposal can be obtained using Factorized distribution solution where this update rule is similar to Gibbs sampling in a sense that the update of particular hyper-parameter depends on all hyper-parameters except itself. Finally, the mean of proposal distribution is the estimate for the hyper-parameter.

The variational inference in this project follow procedure described in [1], where prior of w, λ, and α are $N(w|0,(\lambda\,\alpha)^{-1})$, $Gam(\lambda|a_0^{\lambda}\,b_0^{\lambda})$, and $Gam(\alpha\,|a_0^{\alpha}\,b_0^{\alpha})$ where λ is inverse noise variance (1/s2) and α is inverse weight variance. The proposal distribution is factorized in the way as:

$$q(w, \alpha, \lambda)= q(w, \lambda)\, q(\alpha)$$

Where priors are:

$$q(w)=N(w|0,(\lambda\,\alpha)^{-1})$$

$$q(\lambda)=Gam(\lambda|a_0^{\lambda},b_0^{\lambda})$$

$$q(\alpha)=Gam(\alpha\,|a_0^{\alpha},b_0^{\alpha}).$$

Where

$V_N= a_N^{\lambda} / b_N^{\lambda} I + x^T x$

$W_N=V_N x^T y$

$a_N^{\lambda} = a_0^{\lambda} + (N/2)$

$b_N^{\lambda} = b_0^{\lambda}\, 0.5(||y-xW_N||^2 + W_N^T (a_N^{\lambda} / b_N^{\lambda})\, W_N$

$a_N^{\alpha} = a_0^{\alpha} + (K/2)$

$b_N^{\alpha} = b_0^{\alpha}\, 0.5((a_N^{\lambda} / b_N^{\lambda})\, W_N^T\, W_N + tr(V_N))$

The variable a_0^{α}, b_0^{α}, a_0^{λ}, and b_0^{λ} are the hyper-parameter for α and λ where the values of all variables are chosen to be one.

These procedure is transformed to 2 functions which are “update_w_and_lambda” and “update_alph” functions shown in Figure15. Since the prior distribution suggested by [1] of λ and α are inverse Gamma. Therefore, the value input in Gamma function of stats library should be shape (a) and scale (1/rate) where rate are b_N^{λ} and b_N^{α}.

```python
#update prob of w and lambda
def update_w_and_lambda(a_n_alph,b_n_alph,x,y):
    v_n=np.linalg.inv((a_n_alph/b_n_alph)*np.identity(x.shape[1])+np.matmul(x.T,x))
    w_n=np.squeeze(np.matmul(np.matmul(v_n,x.T),y[:,np.newaxis]))

    num_sample=x.shape[0]
    a_n_lambda=a_0_lambda+(num_sample/2)
    value1=np.linalg.norm(y-np.squeeze(np.matmul(x,w_n[:,np.newaxis])))**2
    value2=np.squeeze(np.matmul(w_n[:,np.newaxis].T,(a_n_alph/b_n_alph)*w_n[:,np.newaxis]))
    b_n_lambda=b_0_lambda+0.5*(value1+value2)

    #calculate prob
    w=w_n
    lam=a_n_lambda/b_n_lambda
    prob_w=stats.multivariate_normal.pdf(w,w_n,v_n/lam)

    prob_lambda=stats.gamma.pdf(x=lam,a=a_n_lambda,scale=1/b_n_lambda)
    return prob_w,w_n,v_n,prob_lambda,a_n_lambda,b_n_lambda

#update prob alph
def update_alph(a_0_alph,b_0_alph,w_n,v_n,a_n_lambda,b_n_lambda,x):
    d=x.shape[1]
    a_n_alph=a_0_alph+(d/2)
    value=np.squeeze(np.matmul((a_n_lambda/b_n_lambda)*w_n[:,np.newaxis].T,w_n[:,np.newaxis]
    b_n_alph=b_0_alph+0.5*(value+np.trace(v_n))

    #calculate prob
    alph=a_n_alph/b_n_alph
    prob_alph=stats.gamma.pdf(x=alph,a=a_n_alph,scale=1/b_n_alph)
    return prob_alph,a_n_alph,b_n_alph
```

Figure15: Update functions for variational inference

The total log of factored proposal distribution and RMSE of training and test set are calculated in each iteration which are shown in Figure16 and Figure17.

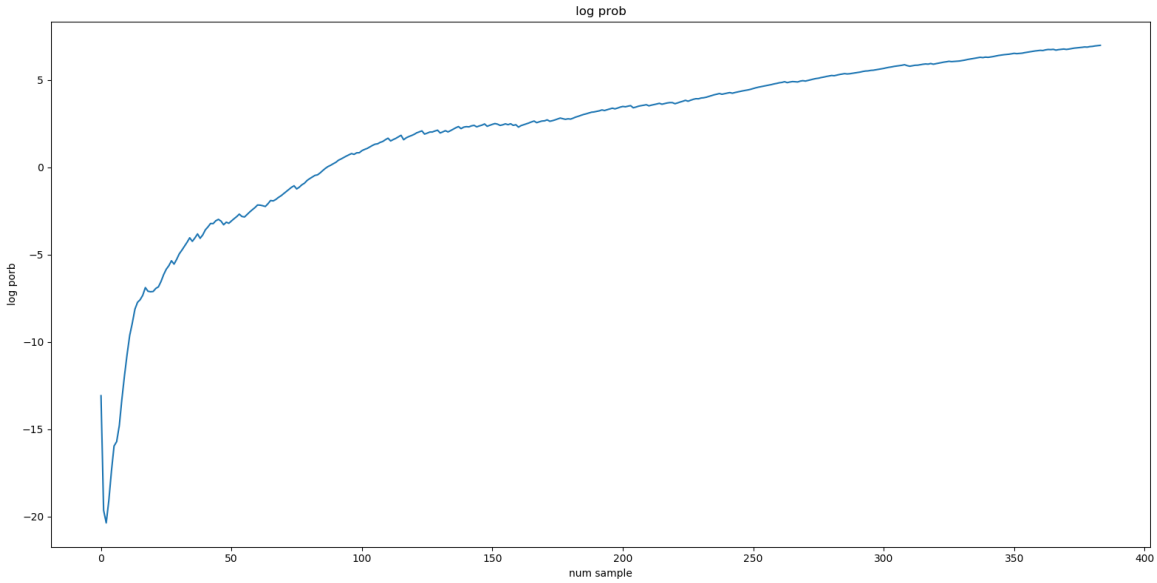


Figure16: Total log of factored proposal distribution with respect to number of samples

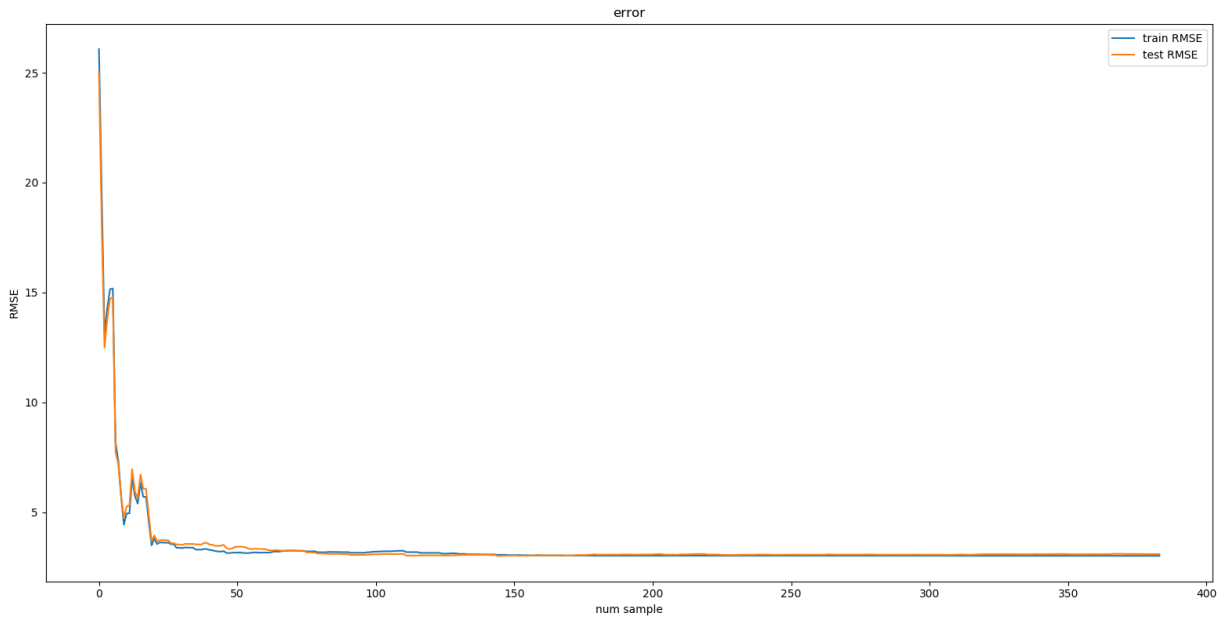


Figure17: RMSE of training and test set with respect to number of samples

The model performed well in terms of error as well as continuously increase the log probability. In Figure17, both training and test error decrease rapidly for first 20 samples and remain stable after 50 sample onwards. Also in Figure18 and Figure19 the obtained value of alpha and s2 located in good position on posterior contour plot where the color is yellow which indicate highest value. The difference between 2 curves is only that the Figure19 plot in better way since some value of s2 and alpha did not get sampled in Variational Inference process as there are some white space on the region. Figure19 used numpy linspace in order to have smoother contour plot with similar range of values.

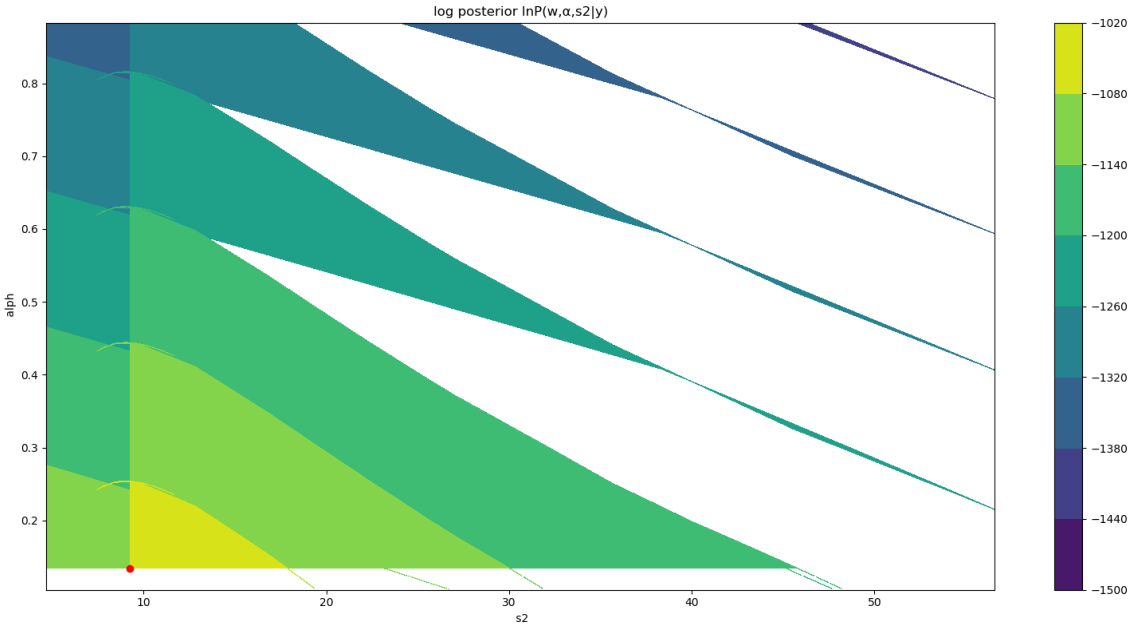


Figure18: Posterior distribution of Variational inference

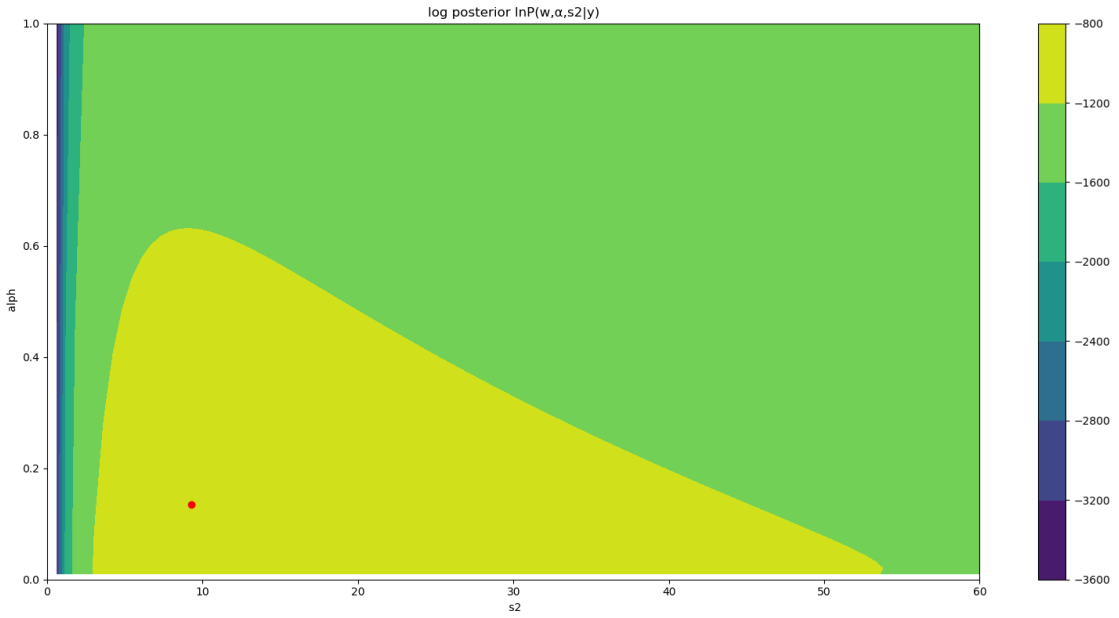


Figure19: Posterior distribution of Variational inference (smooth)

The best value can be obtained by taking the mean of proposal distribution at the final iteration. Therefore best alpha of λ, α, and w are $a_N{}^{\lambda}$/ $b_N{}^{\lambda}$, $a_N{}^{\alpha}$/ $b_N{}^{\alpha}$, and W_N ;respectively, where best s2 is the inverse of best λ. All values obtained are shown in Table4 and errors are shown in Table5.

Table4: Value of most probable parameters from Variational inference

Parameter	Value
Best alpha (inverse weight variance)	0.13462209254849022
Best s2 (noise variance)	9.260917427758185
W1	-6.86459553
W2	-3.68826174
W3	0.81494196
W4	-4.01007459
W5	7.31383394
W6	-0.12623419
W7	2.77138242
W8	0.20247917
W9	22.91265232

Table5: RMSE of training and test set of Variational inference

Model	RMSE train	RMSE test
Variational inference	3.011767383050821	3.0918273471103173

Hamiltonian Monte Carlo (HMC) on a simple Gaussian

Hamiltonian Monte Carlo or Hybrid Monte Carlo made used of Hamiltonian dynamic concept in order to make the random walk in Gibbs sampling and metropolis hasting algorithm more efficient

The example design in this problem is correlated bivariate normal with mean and variance of 0 and 1 with correlation of 0.9 as shown in Figure20.

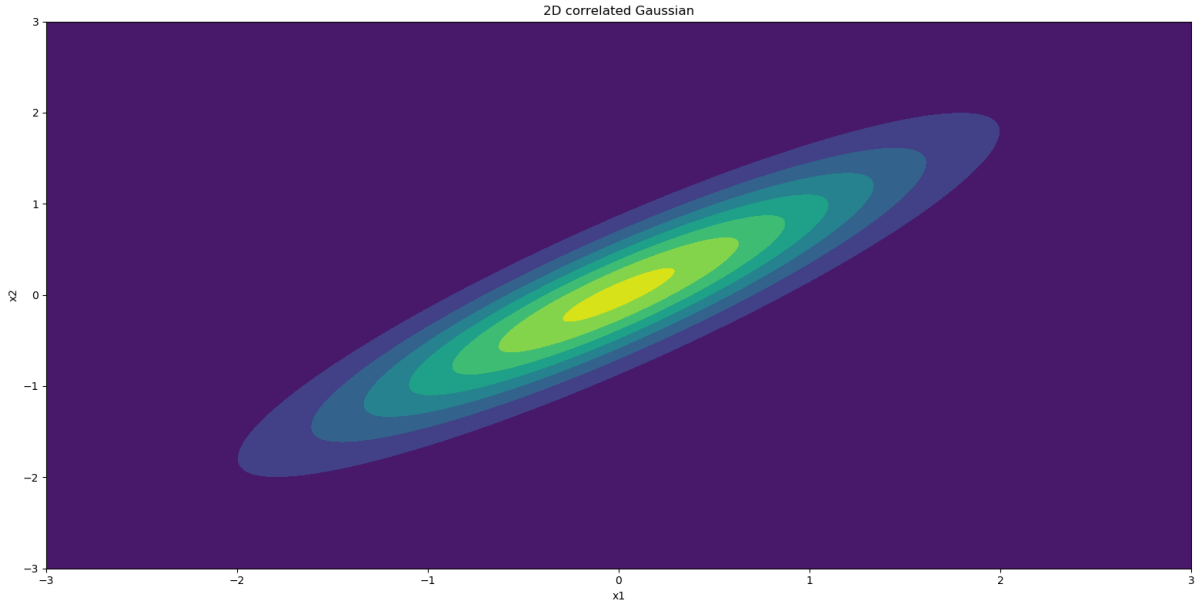


Figure20: 2-dimensional Gaussian example with mean and variance of 0 and 1 with correlation of 0.9

Design and validation of energy function

Energy function input the value of x1 and x2 and output energy value that is the negative log probability. The energy function, shown in Figure21, is validated by compare with the value obtained from stats library as shown in Table6. The difference is zero which confirmed that there is no error. The energy equation can be determined by the following:

$$P(x1,x2) = (2pi)^{-1} (1-\rho)^{-1} \exp[(-x_1^2+2\rho x_1 x_2 - x_2^2)/(2(1-\rho^2)]$$

$$Energy = -\ln(P(x1,x2))$$

$$= \ln(2pi) + 0.5\ln(1-\rho^2) + [(x_1^2-2\rho x_1 x_2 +x_2^2)/(2(1-\rho^2)]$$

```python
#energy function
def energy_function(x,f):
    x0=x[0]
    x1=x[1]


    a=-np.log(2*np.pi)
    b=np.log((1-(correlation**2)))/2
    c=((-x0**2)+(2*correlation*x0*x1)-(x1**2))/(2*(1-(correlation**2)))
    log_p=a-b+c
    return -log_p
```

Figure21: Energy function for 2-dimensional Gaussian example

Table6: Validate energy function for 2-dimensional Gaussian example

	Coding	Stats library	Absolute difference
P(x1,x2)	0.00032165685084216075	0.00032165685084216075	0.0

Design and validation of gradient function

For gradient function, the function takes the input of x1 and x2 and return the corresponding gradient for each variable as shown in Figure22. Gradient can be determined by taking the derivative of energy function with respect to each variable. The gradient function is validated using check gradient function, the result is shown in Table7. The equation of gradient can be calculated as shown:

$$\text{Gradient wrt x1} = \frac{d\ energy}{d\ x1} = (2x_1 - 2\rho x_2)/(2(1-\rho^2))$$

$$\text{Gradient wrt x2} = \frac{d\ energy}{d\ x2} = (2x_2 - 2\rho x_1)/(2(1-\rho^2))$$

```python
#grad
def grad_funtion(x,f):
    x0=x[0]
    x1=x[1]

    grad1=((2*x0)-(2*correlation*x1))/(2*(1-(correlation**2)))
    grad2=((2*x1)-(2*correlation*x0))/(2*(1-(correlation**2)))

    return np.array([grad1,grad2])
```

Figure22: Gradient function for 2-dimensional Gaussian example

Table7: Validate gradient function for 2-dimensional Gaussian example

	Calculated	Numeric	Delta	Accuracy
X1	11.2954	11.2954	-4.116973e-10	11
X2	-11.8463	-11.8463	-1.740030e-10	11

It is clear that, the value of accuracy is more than 6 and value of delta is very low. This indicate that there is no error in coding or mathematical calculation.

Hyper-parameter selection

The value of hyper-parameter L used is 25 as it is sufficient for small example while R is chosen to be 10000. With this value of L, the hyper-parameter epsilon is tuned such the way that the percent acceptance is close to 80 percent as possible. This is done by keep increasing the value of epsilon until the percent acceptance is lower than 80 percent for the first time, then the value of epsilon when the percent acceptance is higher than 80% for the last time is selected. As shown in Figur23, Figure24, and Figure25, the procedure is done by narrowing down the range of epsilon from 0 to 1¸ 0.41 to 0.5, and 0.48 to 0.5. The value of epsilon chosen is 0.487 with percent acceptance of 80.36% because if the epsilon is increased to 0.489, the percent acceptance will fall to 79.41%

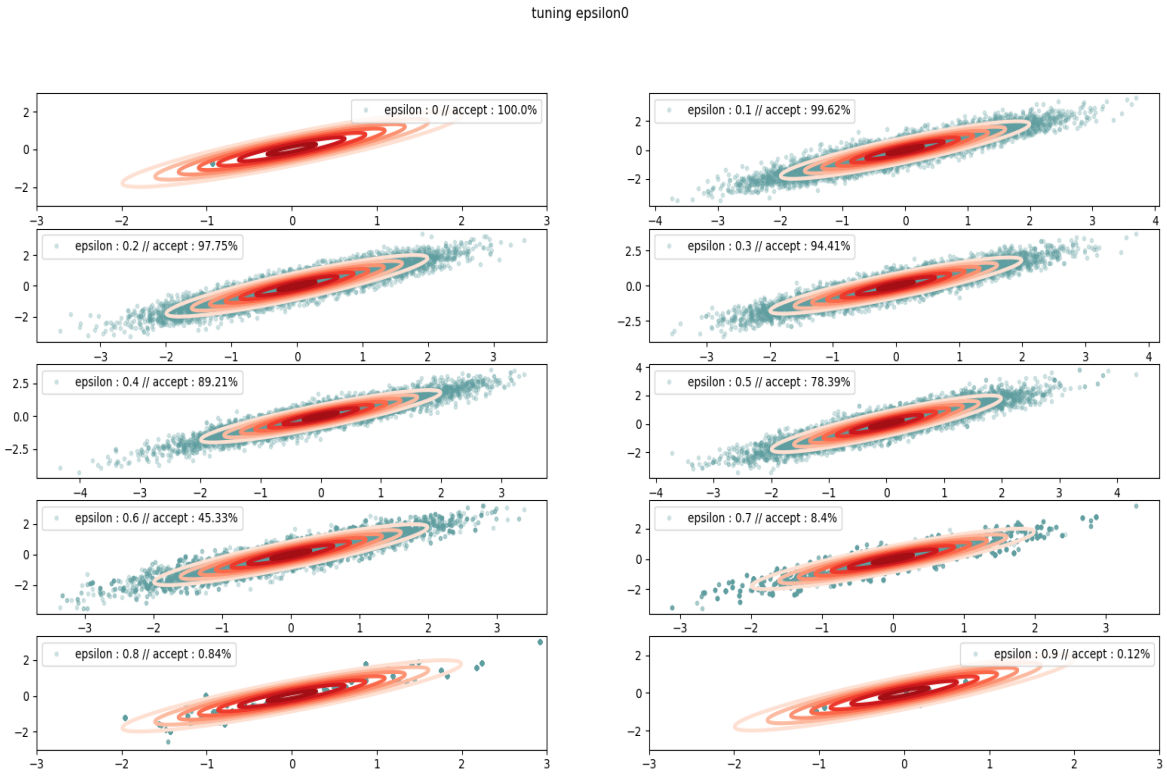


Figure23: Finding best epsilon from 0 to 1

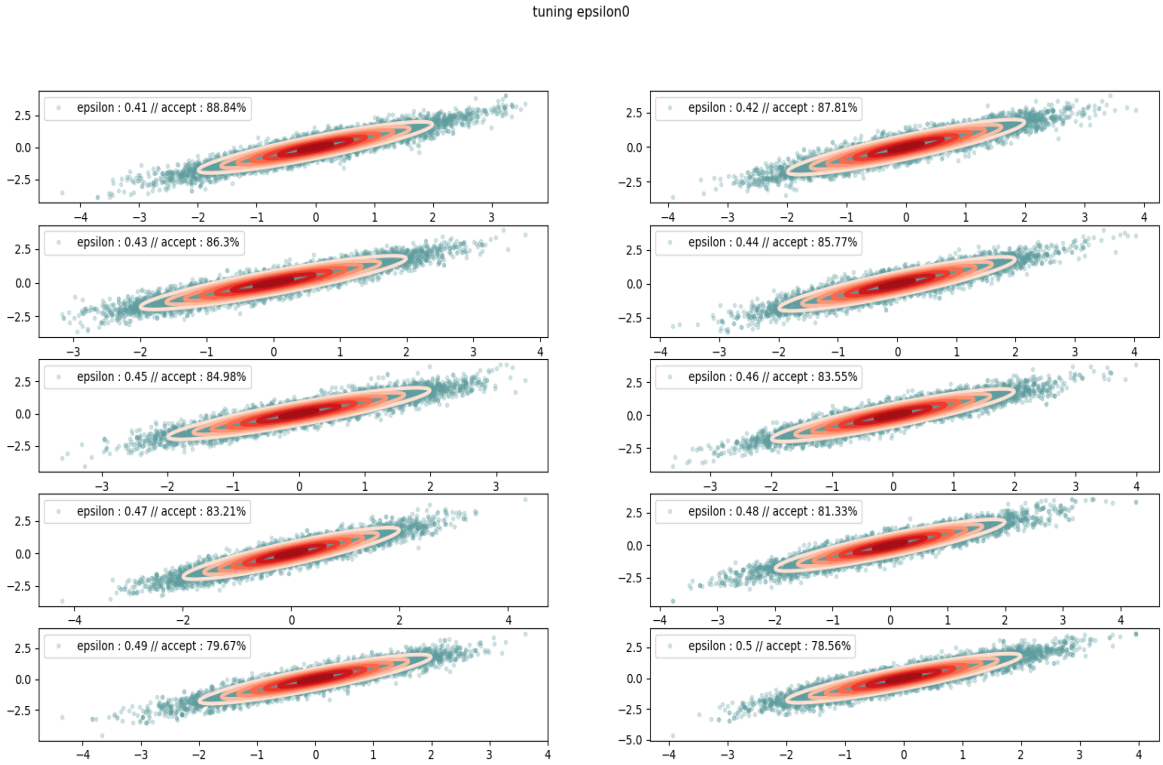


Figure24: Finding best epsilon from 0.41 to 0.5

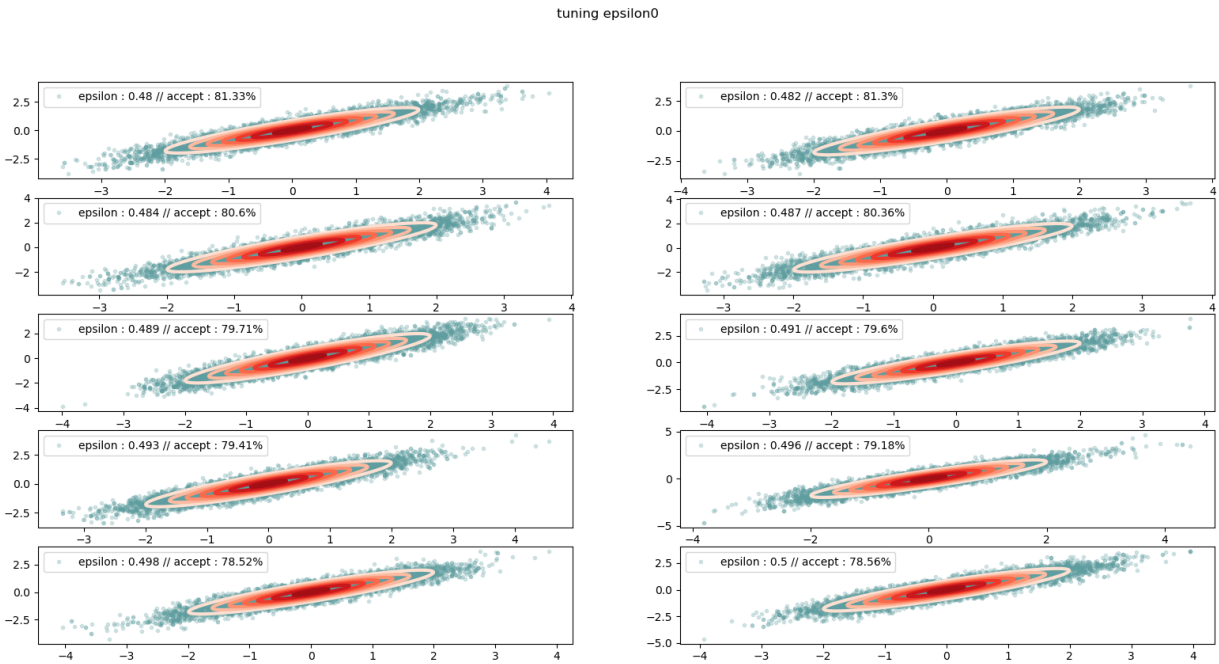


Figure25: Finding best epsilon from 0.48 to 0.5

Validation of HMC algorithm

The HMC algorithm is validate as shown in Figure26, the sample are located around the pattern of bivariate gaussian distribution with zero mean which indicate that HMC actually sampling from this 2-dimensional Gaussian example. Also, the estimated value converges to true value which is zero as shown in Figure27 and Figure28. The estimations are done by averaging each sample with all the sample before this sample using numpy cumulative sum.

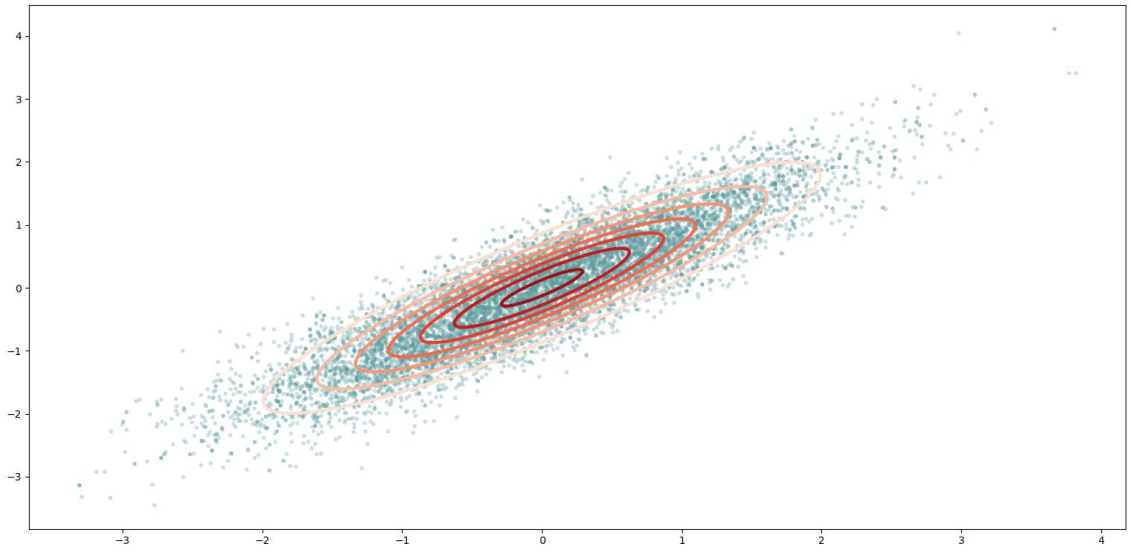


Figure26: Sampling from 2-dimensional Gaussian example using HMC

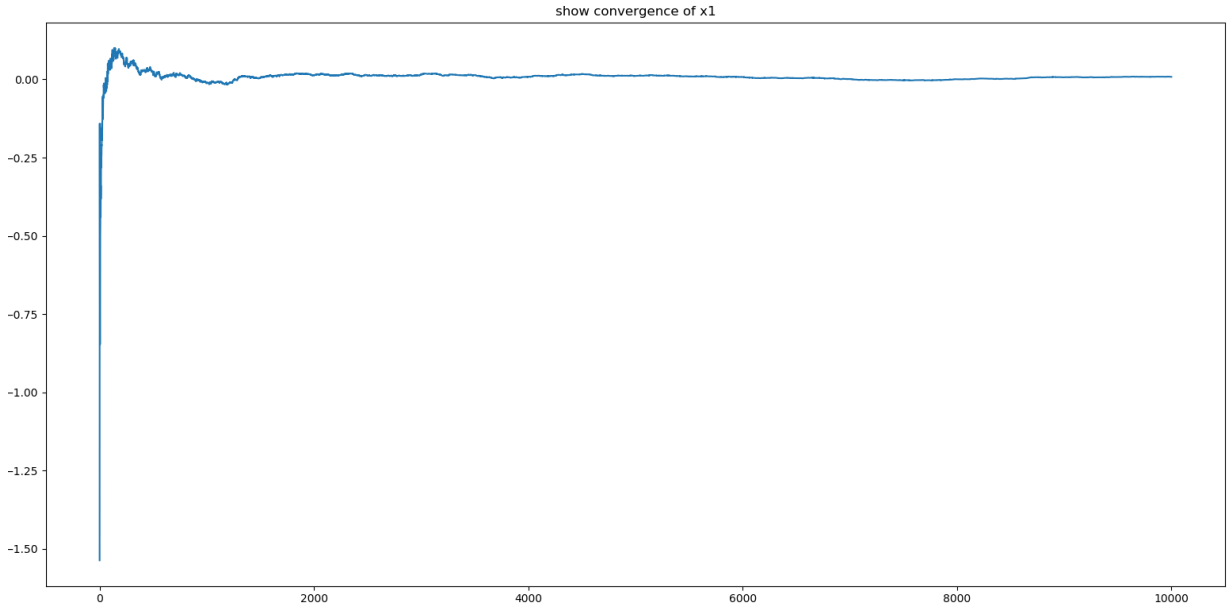


Figure27: Convergence of unknown parameter x1

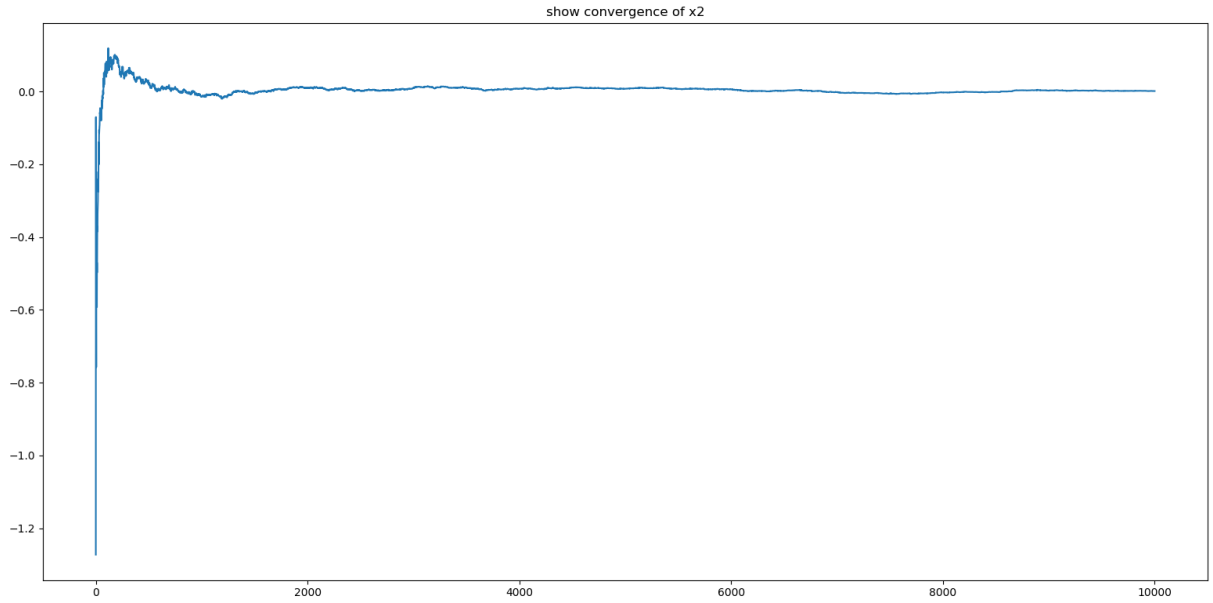


Figure28: Convergence of unknown parameter x2

HMC

Design and validation of energy function

The posterior that we are sample from is P(w, α,s2|y) which is equal to product of P(y|w,s2), P(w| α), P(α),and P(s2). The inverse weight variance hyper-prior and noise variance prior are assumed to follow inverse gamma distribution with default parameter of shape and rate of 10^-2 and 10^-4; respectively, as recommended by [2]. The log posterior can be written as:

$$\ln P(w, \alpha, s2|y) = a + b + c + d$$

where:

$$a = \ln P(y|w,s2) = -\frac{N\, ln(2pi\ s2)}{2} - \frac{sum(y - x^T w)^2}{2\ s2}$$

$$b = \ln P(w|\ \alpha) = -\frac{M}{2} \ln\left(\frac{\alpha}{2pi}\right) - \frac{\alpha\ \mathrm{sum}(w^2)}{2}$$

$$c = \ln P(\alpha) = \ln\left(\frac{b0^{a0}}{Gamma(a0)}\right) - (a0-1)\ln(\alpha) - \frac{b0}{\alpha}$$

$$d = \ln P(s2) = \ln\left(\frac{b0^{a0}}{Gamma(a0)}\right) + (a0-1)\ln(s2) - b0s2$$

$$a0 = 10^{-2}$$

$$b0 = 10^{-4}$$

Therefore, the energy function, as shown in Figure29 is equal to negative log probability which can be written as:

$$\text{Energy} = - a - b - c - d$$

To validate the accuracy of energy function, this is done by comparing the value with output from stats library for 4 terms consists of log likelihood (lnP(y|w,s2)), log weight prior (lnP(w| α)), log hyper prior inverse weight variance (lnP(α)), and log noise variance prior (lnP(s2)). This can be compared by convert energy back to log probability. The results are shown in Table8 where the difference is very small and verified the accuracy of the function.

Table8: Validate energy function for HMC linear regression

	Coding	Stats library	Absolute difference
lnP(y|w,s2)	-296357.6249960258	-296357.6249960263	4.656612873077393e-10
lnP(w| α)	-10.09966444922638	-10.099664449226378	1.7763568394002505e-15
lnP(α)	-3.9531513325728818	-3.953151332572883	1.3322676295501878e-15
lnP(s2)	-3.8395808667784257	-3.8395808667784266	8.881784197001252e-16

```python
#energy function
def energy_function(x0,f):
    x=f[0]
    y=f[1]
    s2=x0[0]
    alph=x0[1]
    w=x0[2:]

    #common term
    N=x.shape[0]
    M=x.shape[1]
    term=(b0**a0)/gamma(a0)
    energy=0

    #a
    second_term=sum((y-np.matmul(x,w))**2)/(2*s2)
    a=-0.5*N*np.log(2*np.pi*s2)-second_term
    energy-=a

    #b
    b=0.5*M*np.log((alph/(2*np.pi)))-sum(alph*(w**2)/2)
    energy-=b

    #c
    c=np.log(term)-(a0-1)*np.log(alph)-(b0/alph)
    energy-=c

    #d
    d=np.log(term)+((a0-1)*np.log(s2))-(b0*s2)
    energy-=d

    return energy
```

Figure29: Energy function for HMC linear regression

Design and validation of gradient function

Gradient can be calculated by derivative of Energy function respect to each parameter as shown in following equations:

$$\text{Gradient of a wrt w} = \frac{d\,a}{\mathrm{d}\,w} = \frac{-2(y-xw)^2(-x)}{2\,s2}$$

$$\text{Gradient of a wrt } \alpha = \frac{d\,a}{\mathrm{d}\,\alpha} = 0$$

$$\text{Gradient of a wrt s2} = \frac{d\,a}{\mathrm{d}\,s2} = \frac{-N\ln(2pi)}{2} + \frac{sum(y-xw)^2}{2\,s2^2}$$

$$\text{Gradient of b wrt w} = \frac{d\,b}{\mathrm{d}\,w} = -\frac{2w\alpha}{2}$$

$$\text{Gradient of b wrt } \alpha = \frac{d\,b}{\mathrm{d}\,\alpha} = \frac{M}{2\alpha} - \frac{sum(w^2)}{2}$$

$$\text{Gradient of b wrt s2} = \frac{d\,b}{\mathrm{d}\,s2} = 0$$

$$\text{Gradient of c wrt w} = \frac{d\,c}{\mathrm{d}\,w} = 0$$

$$\text{Gradient of c wrt } \alpha = \frac{d\,c}{\mathrm{d}\,\alpha} = \frac{-(a0-1)}{\alpha} + \frac{b0}{\alpha^2}$$

$$\text{Gradient of c wrt s2} = \frac{d\,c}{\mathrm{d}\,s2} = 0$$

$$\text{Gradient of d wrt w} = \frac{d\,d}{\mathrm{d}\,w} = 0$$

$$\text{Gradient of d wrt } \alpha = \frac{d\,d}{\mathrm{d}\,\alpha} = 0$$

$$\text{Gradient of d wrt s2} = \frac{d\,d}{\mathrm{d}\,s2} = \frac{(a0-1)}{s2} - b0$$

The gradient function, as shown in Figure30, is again validated using check gradient function. The results are shown in Table9. All gradients show accuracy of higher than 6 which indicate the absent of error in coding and calculation.

Table9: Validate gradient function for HMC linear regression

	Calculated	Numeric	Delta	Accuracy
S2	-31224.3	-31224.3	6.105620e-06	10
α	1.55016	1.55016	-1.411203e-07	8
W0	-1204.22	-1204.22	2.490060e-05	8
W1	1297.55	1297.55	6.914611e-06	9
W2	-996.629	-996.629	1.170854e-05	8
W3	1742.71	1742.71	2.422461e-05	8
W4	-1820.65	-1820.65	-1.494205e-05	9
W5	302.197	302.197	-2.285516e-05	8
W6	-545.26	-545.26	-2.332535e-06	9
W7	-207.931	-207.931	3.188797e-05	7
W8	-4426.91	-4426.91	2.204899e-05	9

```
def grad_function(x0,f):
    x=f[0]
    y=f[1]
    s2=x0[0]
    alph=x0[1]
    w=x0[2:]
    #common term
    N=x.shape[0]
    M=x.shape[1]
    output_w=np.array([0]*len(w))
    output_s2=0
    output_alph=0
    #a wrt to w
    a_w=-2*np.matmul((y-np.matmul(x,w)),-x)
    output_w=output_w-(a_w/(2*s2))
    #a wrt alph
    a_alph=0
    output_alph-=a_alph
    #a wrt s2
    term=(sum((y-np.matmul(x,w))**2))*(-(s2**-2))
    a_s2=(-0.5*N/s2)-(term/2)
    output_s2-=a_s2
    #b wrt w
    b_w=-alph*2*w/2
    output_w=output_w-b_w
    #b wrt alph
    b_alph=(M/(2*alph))-sum((w**2))/2
    output_alph-=b_alph
    #b wrt s2
    b_s2=0
    output_s2-=b_s2
    #c wrt alph
    c_alph=(-(a0-1)/alph)+(b0/(alph**2))
    output_alph-=c_alph
    #d wrt s2
    d_s2=((a0-1)/(s2))-b0
    output_s2-=d_s2
    return np.array([output_s2,output_alph]+output_w.tolist())
```

Figure30: Gradient function for HMC linear regression

Hyper-parameter selection

The model used the value of R for 10000 samples with L and epsilon equal to 100 and 0.0035: respectively. The final percent acceptance is 81.6% where these hyper-parameters are tuned using the same method as described in Task 3.

Validation of HMC algorithm

The Figure31, Figure32, and Figure33 show the convergence of estimate parameter within first 2000 sample

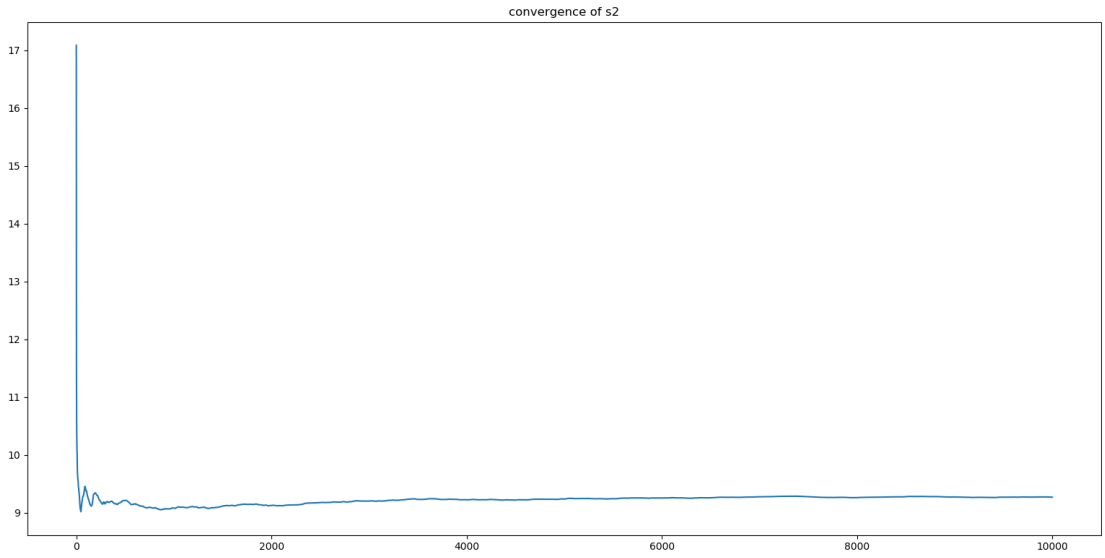


Figure31: Convergence of unknown parameter s2

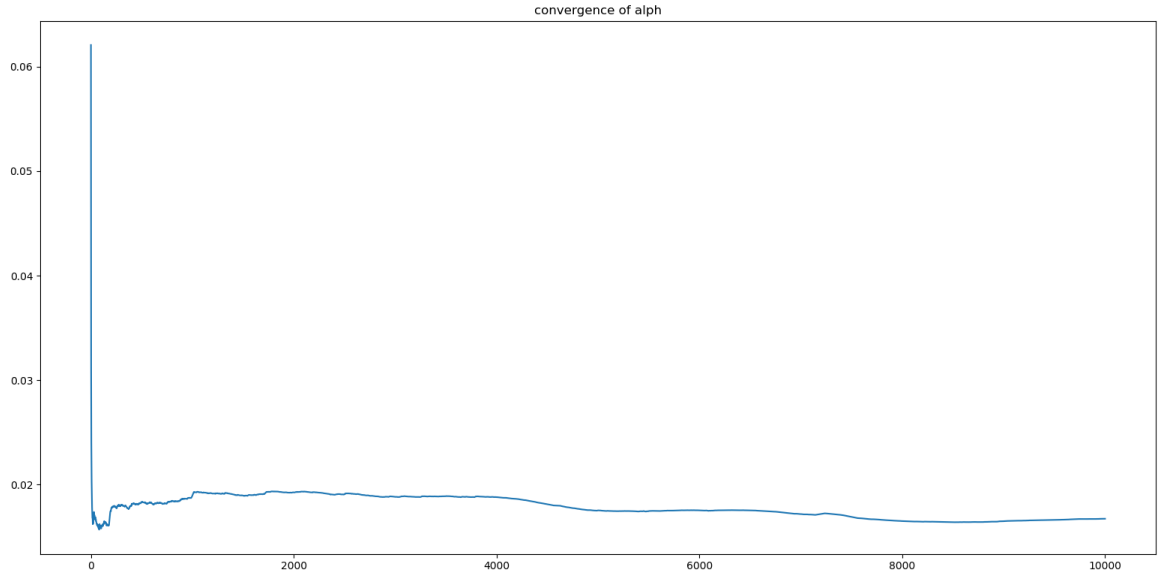


Figure32: Convergence of unknown parameter α

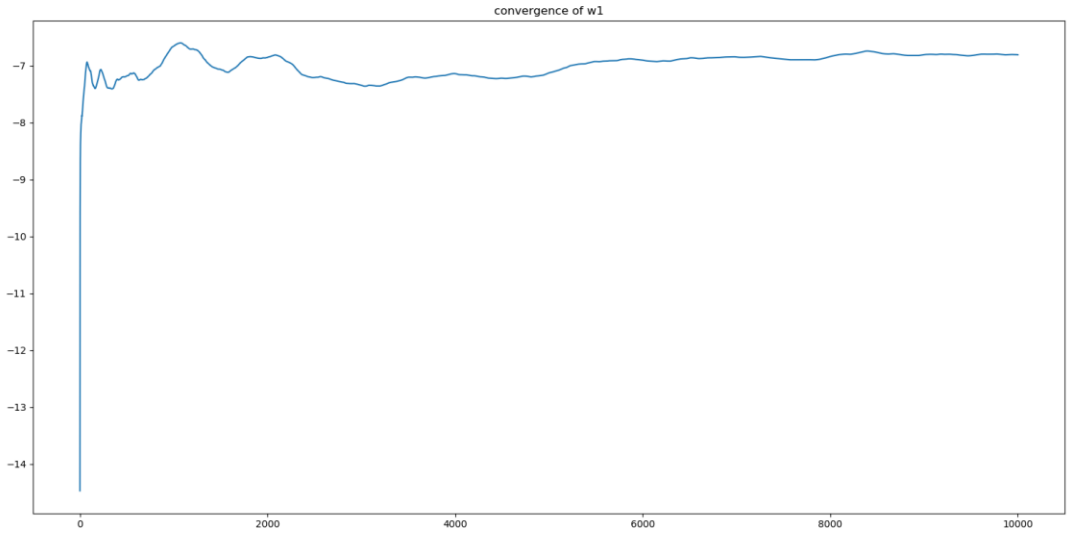


Figure33: Convergence of unknown parameter w1

Result

The model also performs well since the location of converged estimate parameter located at highest yellow region of posterior contour plot as shown in Figure34. The final converged values and errors are shown in Table10 and Table11: respectively.

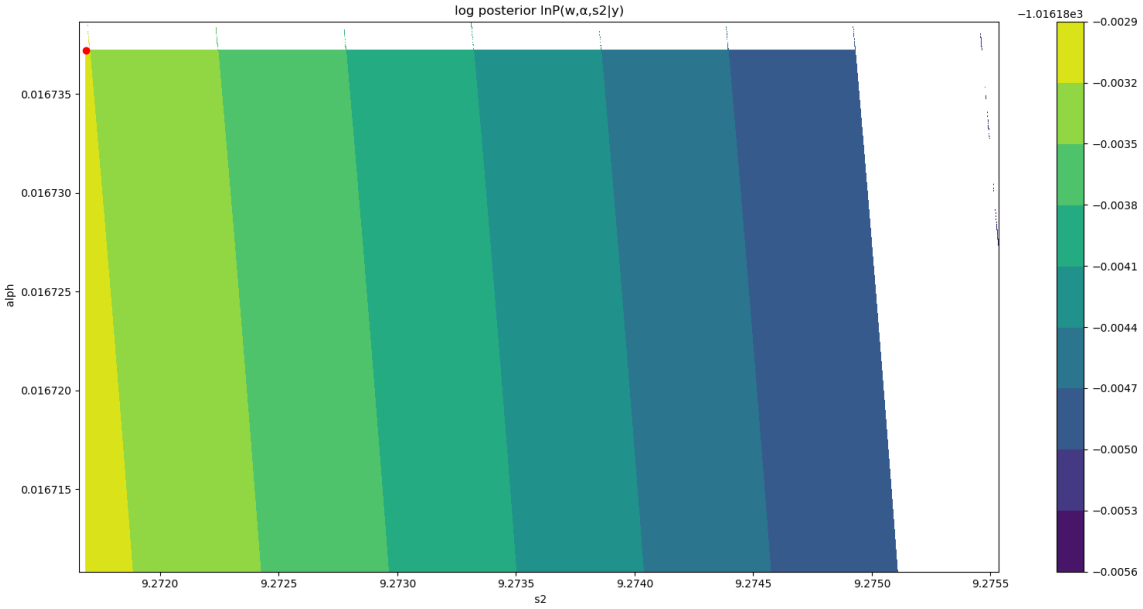


Figure34: Posterior distribution of HMC linear regression

Table10: Value of most probable parameters from HMC linear regression

Parameter	Value
Best alpha (inverse weight variance)	0.016737224846942945
Best s2 (noise variance)	9.271688191719882
W1	-6.80564271
W2	-4.54897081
W3	1.25374272
W4	-3.00750759
W5	7.37815691
W6	-0.12444385
W7	2.76752209
W8	0.20993861
W9	22.91402406

Table11: RMSE of training and test set of HMC linear regression

Model	RMSE train	RMSE test
HMC	3.0118549466697635	3.0887831894624247

Model comparison

Four method have been implemented in this project; the results are then compared in term of errors. (note that linear regression has same value of train and test set error as type 2 maximum likelihood

model). Even though HMC model achieved lower value of log posterior as shown in Figure35 when compare to Variational inference method, the model shows smallest gap between train and test and error and lowest test set error as shown in Figure37 and Figure38 which indicate that the model has lower chance of overfitting than other methods. While type 2 maximum likelihood method and linear regression seems to be the worse approach since the model shows highest test set error, largest gap that indicate the most overfit and lowest log posterior.

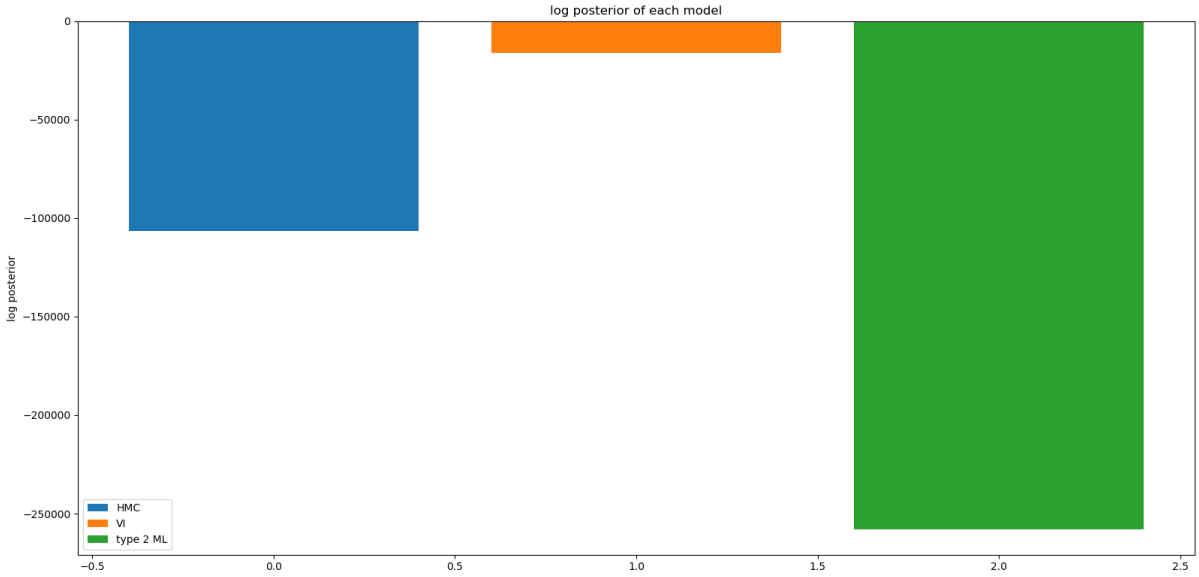


Figure35: Comparison of log posterior between type 2 ML, VI, and HMC

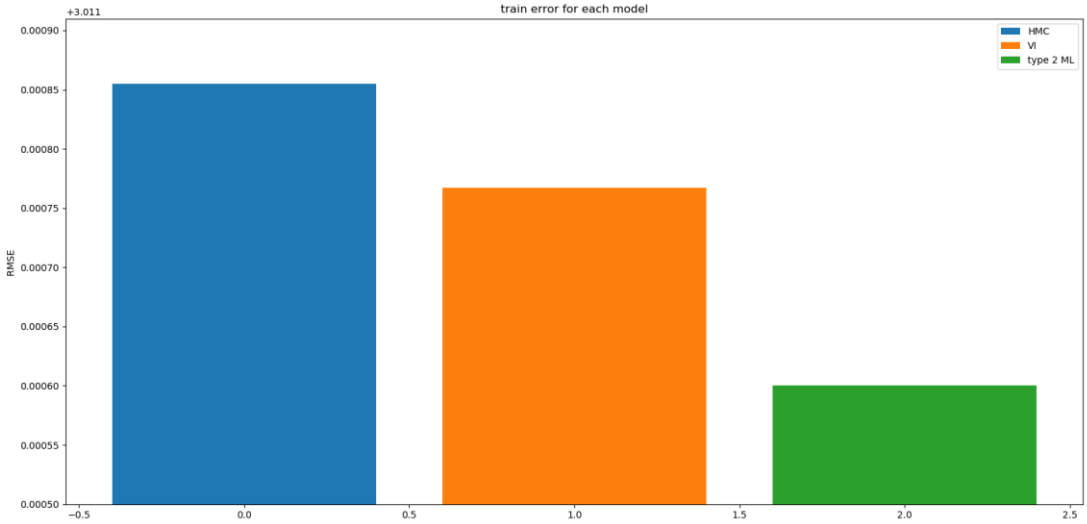


Figure36: Comparison of training set error between type 2 ML, VI, and HMC

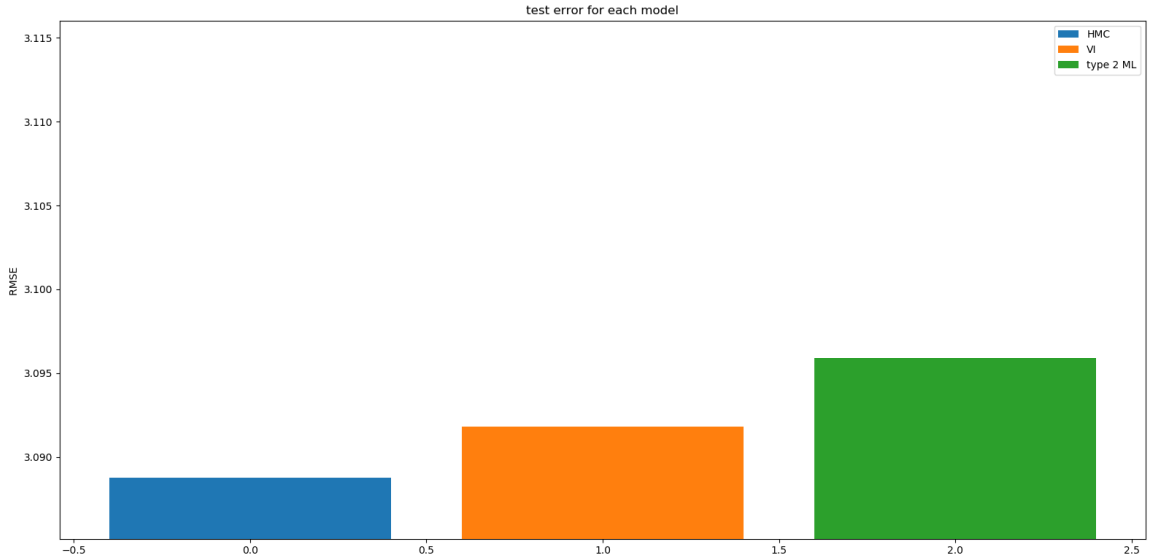


Figure37: Comparison of test set error between type 2 ML, VI, and HMC

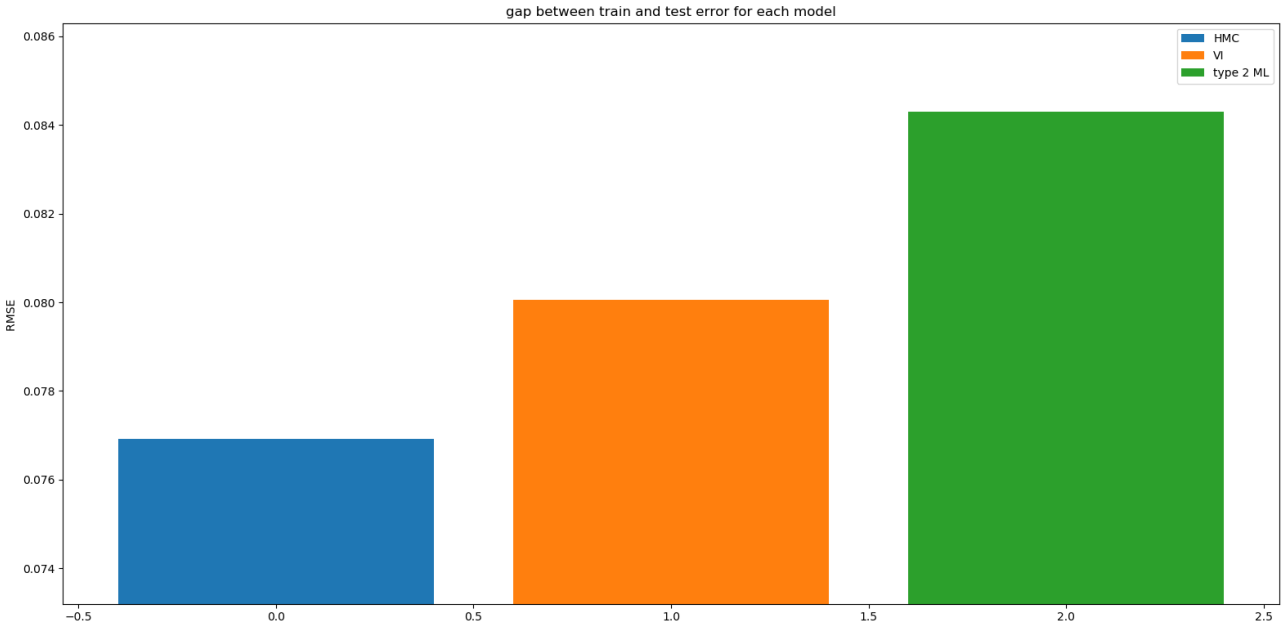


Figure38: Comparison of the gap between training and test set error between type 2 ML, VI, and HMC

HMC classification

Design of energy function

In order to apply to classification problem, only the likelihood term (term a mentioned in previous task) is changed to Bernoulli likelihood instead of Gaussian which can be written as:

$$a = \text{sum}\ (y \ln (\sigma(z)) + ((1-y) \ln (1- \sigma(z)))$$

The energy function is shown in Figure39.

```
def sigmoid_func(w,x):
    z=np.matmul(x,w)
    output=1/(1+np.exp(-z))
    return output
def energy_function(x0,f):
    x=f[0]
    y=f[1]
    s2=x0[0]
    alph=x0[1]
    w=x0[2:]

    #common term
    N=x.shape[0]
    M=x.shape[1]
    term=(b0**a0)/gamma(a0)
    energy=0

    #a
    p=sigmoid_func(w,x)
    a=sum((y*np.log(p))+((1-y)*np.log(1-p)))
    energy-=a

    #b
    b=0.5*M*np.log((alph/(2*np.pi)))-sum(alph*(w**2)/2)
    energy-=b

    #c
    c=np.log(term)-(a0-1)*np.log(alph)-(b0/alph)
    energy-=c

    #d
    d=np.log(term)+((a0-1)*np.log(s2))-(b0*s2)
    energy-=d

    return energy
```

Figure39: Energy function for HMC classification

Design and validation of gradient function

Then, the gradient of the term a can be obtained by taking the derivative of the term a with respect to each parameter as shown:

$$\text{Gradient of a wrt w} = \frac{d\,a}{\mathrm{d\,w}} = (y - \sigma(z))x$$

$$\text{Gradient of a wrt } \alpha = \frac{d\,a}{\mathrm{d}\,\alpha} = 0$$

$$\text{Gradient of a wrt s2} = \frac{d\,a}{\mathrm{d}\,s2} = 0$$

The Table12 show the validation of gradient function using check gradient function.

Table12: Validate gradient function for HMC classification

	Calculated	Numeric	Delta	Accuracy
S2	1.93353	1.93353	-9.313653e-09	9
α	2.46489	2.46489	2.111256e-09	10
W0	-92.2064	-92.2064	6.779386e-08	10
W1	98.5624	98.5624	5.551726e-08	10
W2	-0.278128	-0.278128	-3.880381e-08	7
W3	99.426	99.426	-4.051097e-08	10
W4	-102.512	-102.512	4.219586e-08	10
W5	-0.770226	-0.770226	-5.027356e-08	8
W6	95.8498	95.8498	-2.774472e-08	10
W7	34.7743	34.7743	7.637083e-08	9
W8	3.34895	3.34895	1.494941e-07	8

Hyper-parameter selection

The classification is done using epsilon of 0.00038, L of 100, and R of 10000 samples with percent acceptance of 79.5%

Validation of HMC algorithm

The Figure40, Figure41, and Figure42 show the convergence of estimate parameter. However, the convergence is not as strong as HMC linear regression.

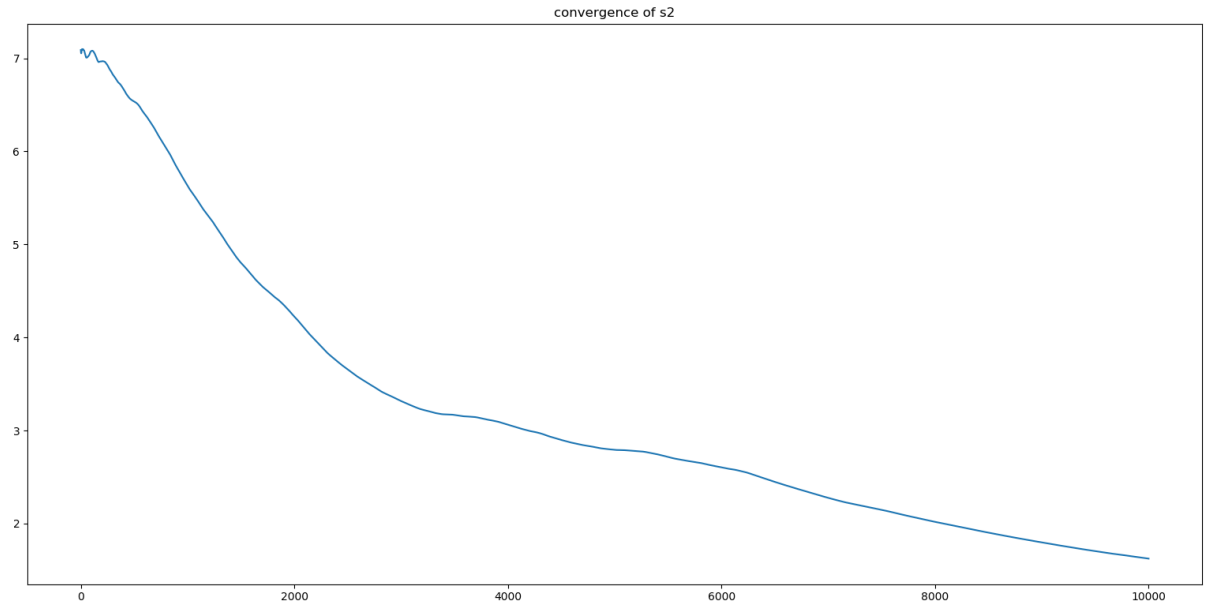


Figure40: Convergence of unknown parameter s2

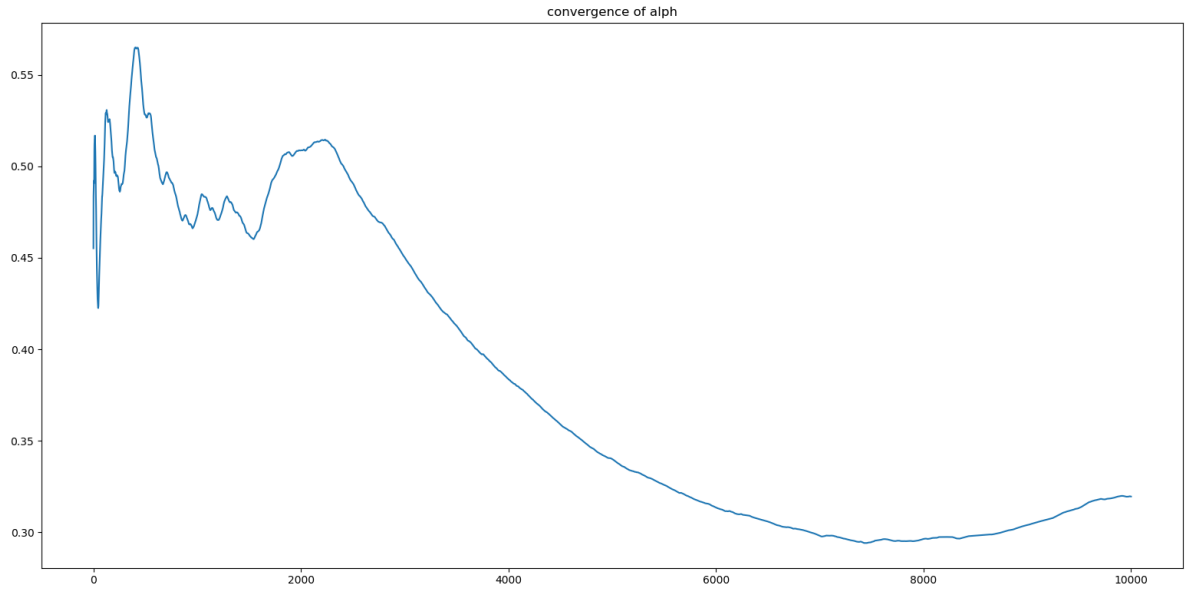


Figure41: Convergence of unknown parameter α

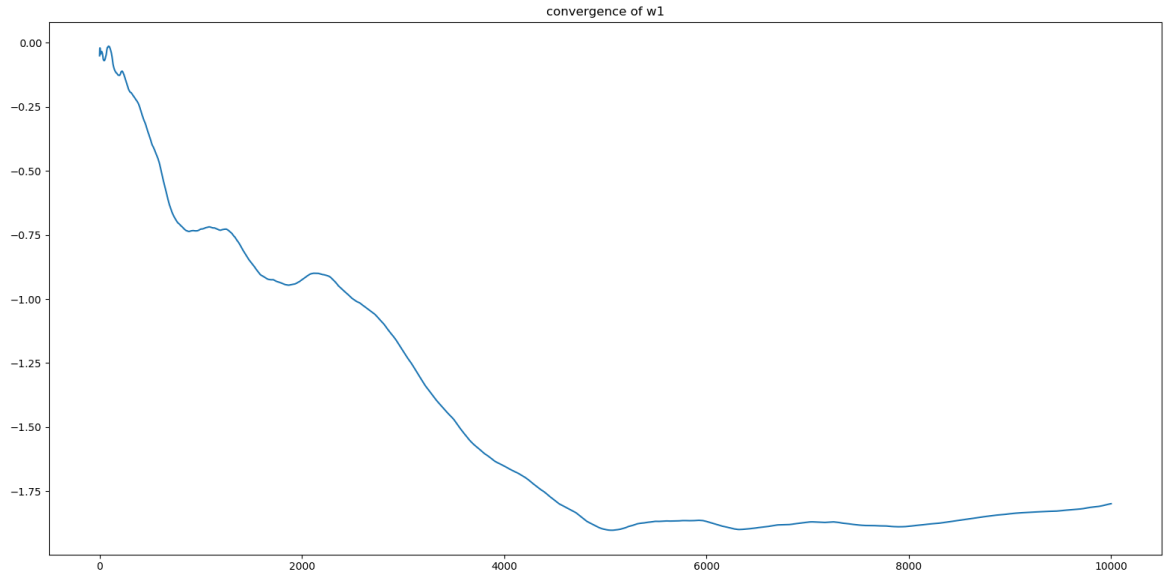


Figure42: Convergence of unknown parameter w1

Result

Model perform well since value of unknown parameter falls in yellow area shown in Figure43. The model achieved about 2 percent misclassification in test set as shown in Table14. The errors are shown in Table13.

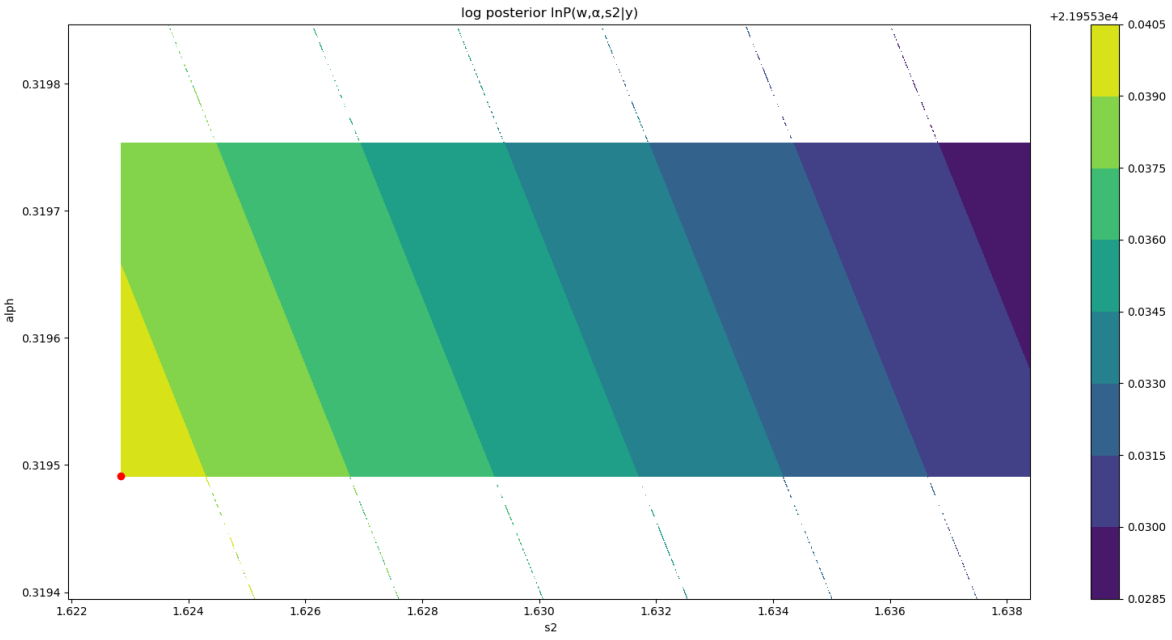


Figure43: Posterior distribution of HMC classification

Table13: Value of most probable parameters from HMC classification

Parameter	Value
Best alpha (inverse weight variance)	0.31949118879611615
Best s2 (noise variance)	1.6228463598273044
W1	-1.79868115
W2	-1.04325573
W3	0.34165174
W4	-2.5761905
W5	5.08983475
W6	0.06827902
W7	2.66850828
W8	0.78963554
W9	-0.49590057

Table14: RMSE of training and test set of HMC classification

Case	Accuracy train	Accuracy test
HMC classification	0.9817708333333334	0.9791666666666666

Summary

To sum up, two method have been used to identify the relevant of each variable. From correlation plot, we found that orientation, glazing area, and glazing area are not as relevant compare to other variables. Using backward elimination with significant level of 0.5, the result implies the irrelevant of orientation and glazing area orientation.

Four techniques have been implemented in this project: linear regression, type 2 maximum likelihood, variational inference, and HMC. HMC shows lowest overfit as the gap between train test error is the lowest which implies the generality of the model. However, least square linear regression and type 2 maximum likelihood model shows largest gap of error as well as highest test set error which indicate that this is the worse model.

References

[1] Murphy, K.P., 2012. *Machine learning: a probabilistic perspective*. MIT press.

[2] Drugowitsch, J., 2013. Variational Bayesian inference for linear and logistic regression. *arXiv preprint arXiv:1310.5438*.