A Bayesian Neural Net to Predict Abalone Rings

ISYE 6420 - Bayesian Statistics - Fall 2023
Final Course Project
Georgia Institute of Technology
Spencer Vore

Abstract

This project investigated how a Bayesian Neural Network, implemented as a Markov Chain Monte Carlo network using the Python PyMC library, would perform against a dataset expected to contain non-linearity. The Abalone dataset from the UCI ML Repository was chosen for this investigation [1].

After the data was standardized, one hot encoded, down sampled, and split into training / test sets, three different models were fit. One model was a basic linear regression. The second was neural network containing an input layer with 3 neurons and a single neuron output layer. The third model was a "deep neural network" with a 4 neuron input layer, a 3 neuron hidden layer, and a 1 neuron output. Each neuron in these networks was a Bayesian Generalized Linear model with a ReLU activation function, except for the final layer which had no activation function.

Each model fit the training dataset well, but most had difficulty fitting the out of sample test dataset. Despite more work being needed to predict out of sample data reliably, this project does show that it is indeed possible to implement a small neural network using PyMC.

Introduction

Neural Networks have been used to make predictions using complex datasets and are the state of the art algorithm in many fields. This project investigates how a Bayesian approach can be applied to neural networks. In the paper "Hands-on Bayesian neural networks—A tutorial for deep learning users" [6], a broad range of methods for training and using Bayesian neural networks are presented. This project only scratches the surface of what's presented in that paper, but uses it as inspiration.

One advantage of Bayesian neural nets is they quantify the uncertainty in the model and parameters, as each parameter is modeled by a probability distribution instead of a constant. By examining the probability distributions in the network, a better understanding of the model certainty can be developed. This is a motivation for studying them.

In this project, the starting point is a Bayesian Generalized Linear Model (GLM). A Bayesian GLM solves for the coefficients of a linear equation by assigning a prior distribution to each parameter, and then solves for them using Markov Chain Monte Carlo (MCMC). Since a GLM is essentially a single neuron in a neural network, it's possible to connect multiple of these GLM's together to construct a more complicated model, and a more complicated Markov Chain.

This approach was experimented with over the course of a few weeks by using a free, toy dataset downloaded from the internet. The goal was to create a small but functional Bayesian neural network.

Dataset

For this project, the Abalone dataset was chosen [1], and downloaded from the UC Irvine Machine Learning Repository. It has 8 different predictor variables which can be used to predict the number of rings in an Abalone shell, which corresponds to the Abalone's age. The predictors are mostly numeric, and are various length and weight measurements of the animals. There is also a categorical variable, which classifies each as "Male", "Female", or "Infant".

This dataset was chosen because it appears well suited for this problem. The goal was to experiment with a regression style neural network to predict a response. A dataset with a few different features, which was expected to contain non-linearities, was needed to see if a neural network could improve performance. Otherwise, a linear regression would be the best model for a linear dataset, and a neural network would not be needed. In addition, the size of this dataset is well suited for Markov Chain Monte Carlo (MCMC). Due to the computational requirements of this method, it would be hard to train a neural network on a larger dataset, either in terms of number of data points or dimensionality. The goal was just to explore how MCMC implemented in PyMC could be used to construct a very basic neural network for the purpose of understanding the math and architecture.

Data Preparation

Before any modeling, the dataset was pre-processed. Due to computational limitations of running the computational intensive MCMC process on a laptop, the dataset of about 4100 data points was down sampled. Around only 1500 datapoints could be used for training, or else the MCMC process would just freeze and not start on the available hardware. An additional 3000 data points were used as a test set. The test set is larger than the training set, which is the opposite of how neural networks are usually trained. However, since there were apparent computational limitations and a maximum training size, it was decided the remaining data would be better used as an over sized testing dataset vs. just not using it. This way, most of the original dataset was used for something.

In addition to the train / test split, the data was standardized. Each numerical variable (predictors and response) was centered to it's mean, and scaled to two standard deviations, as shown in on the Georgia Tech ISYE 6420 Course Github [2] and proposed in this paper [3]. The categorical variable "sex" was one hot encoded into 3 different columns with a 0 or 1 indicating if the variable were in one of the three categories (M, F, or I). The one hot encoding allows a numerical regression to be applied to categorical values.

Modeling

General Approach in all Models

The general approach taken in each model was to use the standardized training set as an input. Each neuron is a Bayesian MCMC Generalized Linear Model, sometimes with a ReLU activation function. The linear model is just a single neuron with no activation function.

For reporting, the R^2 value, the Mean Squared Error (MSE), and a linear regression plot of predicted vs. actual values (standardized count of shell rings) are reported. Predicted values are reported for both the training, and test sets. For test sets, values were predicted two ways. A non-sampling approach, where the coefficients are extracted from the trained Bayesian MCMC model and multiplied by standardized input data is the first approach. Then, a sampling approach was used where test data points were determined by resampling from the network. The sampling approach was run twice, to see if results were comparable between different random sampling runs.

The optimal linear model was used as a template neuron for the subsequent neural networks. In the neural networks, there was no experimentation to change the priors determined in the linear model. The priors of the linear model were used as is. This also allows for a more controlled comparison of the approaches.

To execute the MCMC algorithm, PyMC version 5.9.1 was used in Python 3.11. This was run on a Fedora 38 operating system on a

System76 Lemur Pro (2021) laptop. Everything was run on the 11th Gen Intel® Core™ i5-1135G7 × 8 CPU. Each version of PyMC is different, and the library is changing rapidly, so other versions of PyMC could behave very differently. PyMC still feels like an experimental library, and as such can be more difficult and less reliable to use.

Linear Model

A basic generalized linear model (GLM) was first explored as a base neuron for the network. Although it produced clean predictions on the training set, and produced clean predictions on the test data points when resampling was not used, it struggled when the test dataset was resampled. For prediction on this model, coefficients should simply be extracted from the trained model, and multiplied through new data. A summary of these results will be presented in the "Results" section later.

Through tinkering and experimentation, it was found that the resampled test prediction performed less poorly with more informative priors. By moving the standard deviation of the prior normal distributions from 1000 to 5, the resampled predictions achieved the same order of magnitude and approximate scale as the actual predictions. It seemed very sensitive to the priors. This source from the PyMC documentation was used to try to improve the linear model performance on out of sample prediction [4].

In addition, the following example for "Robust Regression" from the PyMC documentation [5] was implemented for some additional improvement. The final model used the Half-Cauchy distribution for the priors on Sigma. A Student t-distribution with 3 degrees of freedom was used for the priors on mu as well as the final likelihood. The idea of using a t-distribution instead of a Normal distribution is that, since it has heavier tails, it's more robust to outliers.

Despite these improvements, the linear model still struggled with resampling out of sample predictions. It was decided results should be reported without using resampling for test predictions.

Two-Layer Neural Net

The code for the GLM was then copied to create a few interconnected neurons in a new PyMC model. This model used three input GLM's as neurons, with a ReLU activation function applied after each neuron learned a linear combination of the input features. The output from these three neurons then fed into a second output layer, containing a single neuron to predict the number of Abalone rings. The output neuron had no activation function, and the linear combination was used as is.

This had comparable performance to the linear model on the test set when predictions were resampled. Adding the non-linear features seemed to improve it's sampling robustness. Predicting the test dataset without resampling had worse performance.

Deep Neural Net

The two layer neural network above was then further expanded to produce a small, fully connected "deep" neural network. The input layer was expanded from three to four neurons. The the outputs from the input layer were fed into a hidden layer with three neurons. Both layers used ReLU as an activation function. Again, a single neuron without any activation function was used as the predictor in the final layer.

This model performed better on the training set but worse on the test set than the simpler neural network, which seems to indicate an overfitting of the training data due to it's increased complexity. With that said, the deep neural network did work well on the training data, and demonstrates that a Bayesian deep neural net can be written using PyMC. More work is needed to perfect it.

Results

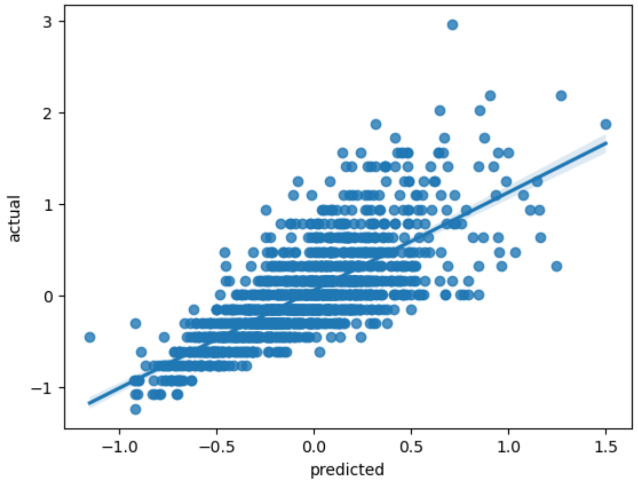
The below table shows the measured results across a variety of metrics, for both the training and test datasets. Note that these values are for a specific run. If you rerun the code, it will produce different values due to the inherent randomness of MCMC.

	Basic linear regression	**2-layer neural network**	**Deep neural network**
Training R^2	0.520	0.593	0.640
Training MSE (sampled)	0.115	0.099	0.085
Test MSE without Resampling	0.124	0.383	3.845
Test MSE with resampling (run 1)	0.254	0.250	0.596
Test MSE with resampling (run 2)	0.205	0.250	0.590
Total Runtime in seconds (training + all testing)	935	2230	5158

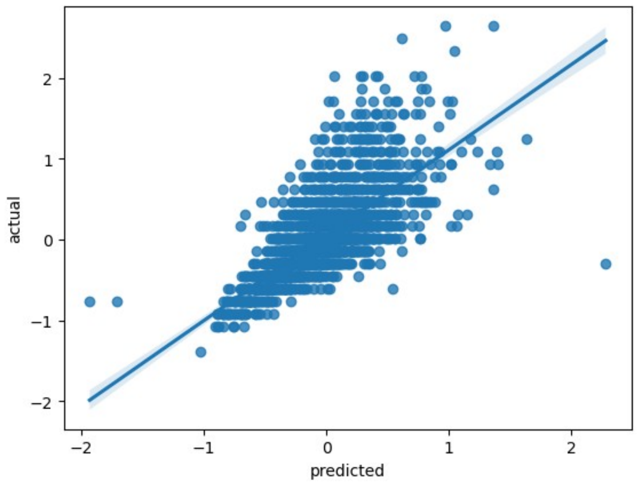
Below are plots showing the linear correlation between the predicted standardized ring values and the actual standardized ring values.

Linear Regression Model Plots

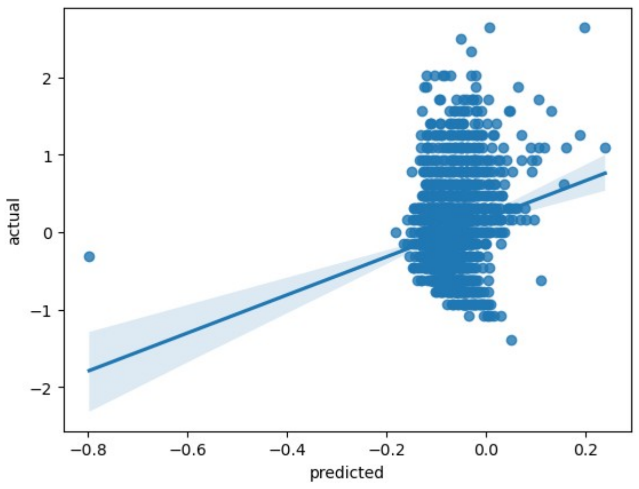
Training:



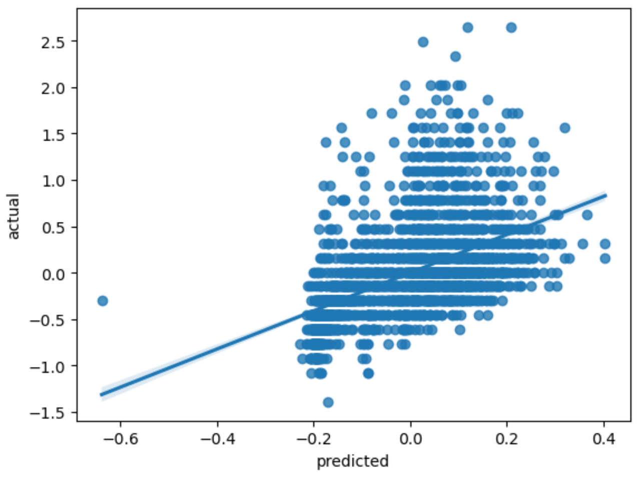
Test using extracted coefficients:



Test by random sampling (run1)



Test by random sampling (run 2)



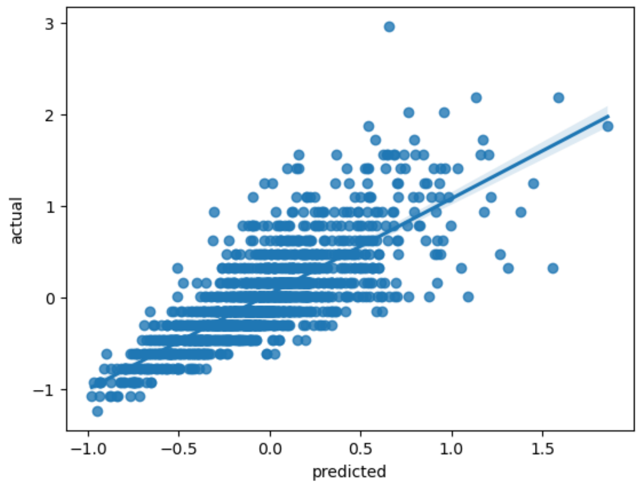
Observations about Linear Model Results

In the above plots, we can see the training predictions work much better. The predicted values are of the same scale as the actual values. Using the extracted coefficients and multiplying them through the test data also produces a clean prediction result, although it has a little wider spread. However, random sampling shows the predicted values are now on a slightly different scale than the actual values. We can also see some outliers, which might be influencing the best fit line. Finally, notice the "stripes" in the x direction. This is because the discrete count of rings was mapped onto a continuous scale, but it still maintains it's discrete levels.

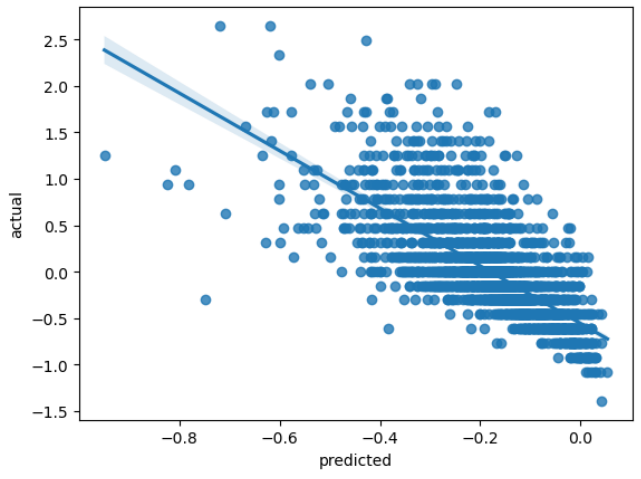
In this situation, predicting using the extracted coefficients appears to produce the best result.

Two-layer Neural Network Plots

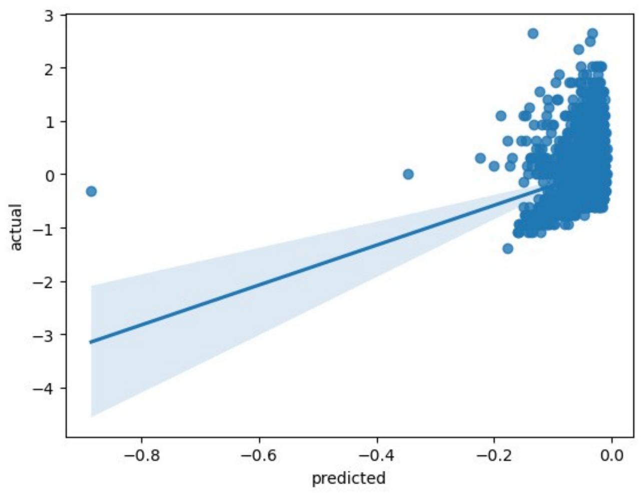
Training



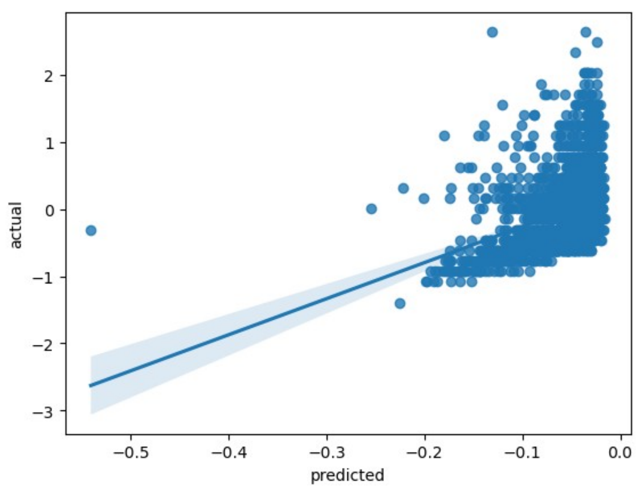
Test using extracted coefficients



Test by random sampling (run 1)



Test by random sampling (run 2)



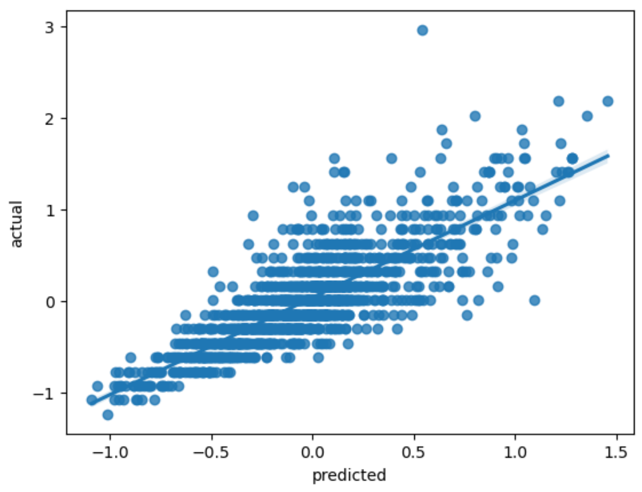
Observations about 2-Layer Neural Network results

Here again, it can be seen that the training dataset produces predictions that are well correlated and of the same scale as the actual values. However, unlike the linear model, sampling seems to produce more accurate predictions than multiplying the test data by the coefficients. For example, note the negative slope of the plot using extracted coefficients for prediction. Also note, for the sampled plots, predicted values don't seem to get much higher than zero. There seems to be a "line" above which the model will never predict in.

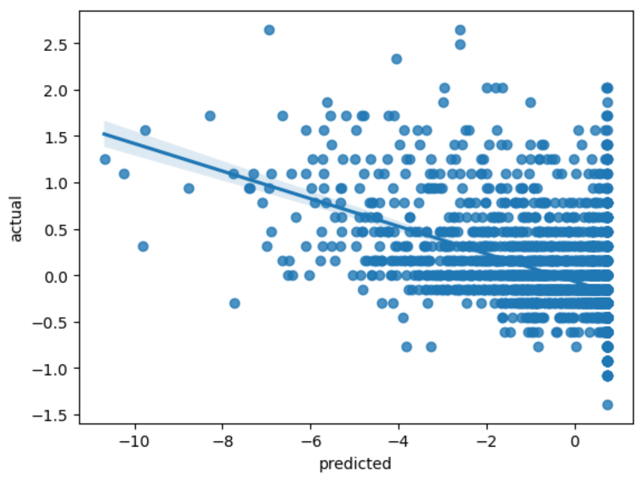
In this situation, the resampling the test dataset appears to produce better results, even if they aren't perfect. The resampling MSE did not improve over the linear model, but it also didn't get much worse.

Deep Neural Network

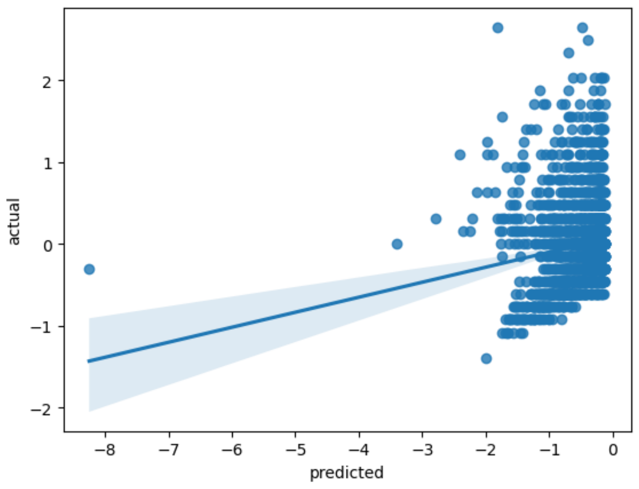
Training



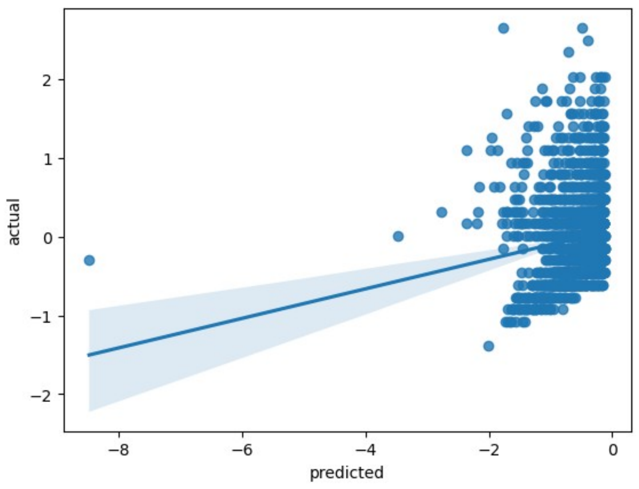
Test using extracted coefficients



Test by random sampling (run 1)



Test by random sampling (run 2)



Observations about Deep Neural Network results

Again, this model produces predictions on the training dataset that align well with the actual values. However, predicting using the extracted coefficients produces a terrible results. One can see from the plot that the range and scale of the predicted values is not the same as the actual values. In this case, random sampling produces much better results. In addition, the random sampling results seem repeatable, as both plots and MSE values are nearly identical.
It is odd there appears to be a "clean cutoff" above which test data points are never predicted at, whether by using coefficients or random sampling on the test set. The predicted scale seen to just stop around 0 or 1. This is not fully understood, but indicates there are still limitations with these predictive models.

In addition, since this model produces the best R^2 and MSE values on it's training dataset out of all the models, but the worst scores on it's test dataset out of all the models, it appears that it is over fitting. Over fitting is a common problem in complex neural networks. For this dataset, it appears a simpler model might be best.

Future Work

More work is needed to get accurate predictions using an out of sample dataset. These models don't appear robust to new data points they were not trained on.

It's possible the specific dataset chosen for this problem is not well suited for this sort of model. It would be interesting to try this approach on some different datasets to see if the same problems arise generally, or if they are specific to this dataset.

The influence of the priors could be better understood. The modeling seemed very sensitive to prior choices, and more experimentation regarding how to choose good priors that produce robust models is needed.

Bayesian neural networks are a large area of study, as outlined in Laurent Jospin's paper " "Hands-on Bayesian neural networks—A tutorial for deep learning users." This paper discusses methods such as approximating the MCMC chain to greatly improve computational efficiency on large networks, or estimating the uncertainty of the network from it's learned prior distributions. Many of these methods could be tried.

More work is needed to reduce the computational efficiency. As we can see from our own process run times, the deep neural network, which only contained a small number of neurons, still took over an hour to run. This approach may not scale well to very large networks, and is more of a theoretical demonstration.

Conclusion

This project was able to demonstrate how it's possible to construct a small neural network out of GLM's using MCMC implemented with PyMC. The MCMC chain is able to solve, and produces good results on the training set.

The models did not perform as well on out of sample test data as desired. In some cases, the model often seemed unstable, and could produce very different test set results just by re-executing the same code with different random seeds. Random sampling seemed more robust with a larger network size, while predicting just by multiplying model coefficients with new data got less reliable the larger the network. The process of manually extracting all the coefficients from the neural net and manually multipling them through the test dataset was complex so there's a higher chance of human error here. More work is needed to better understand predicting using an out of sample test dataset.

The best model ended up being the simple linear model, using just extracted coefficients for prediction instead of resampling the data. The best model for random sampling was the two layer neural network, as it made the random sampling more predictable while maintaining the MSE score.

The deep neural network appears to be overfitting, due to the large difference in it's training vs. test performance. The simple linear model also has the advantage of being the lest computationally expensive to compute.

References

[1] Nash,Warwick, Sellers,Tracy, Talbot,Simon, Cawthorn,Andrew, and Ford,Wes. (1995). Abalone. UCI Machine Learning Repository. https://doi.org/10.24432/C55C7W. Downloaded from: https://archive.ics.uci.edu/dataset/1/abalone

[2] Reding, Aaron, ISYE 6420 Course Github Repository of PyMC examples(Fall 2023), Georgia Tech. Unit 7 Arrhythmia Example. https://areding.github.io/6420-pymc/unit7/Unit7-arrhythmia.html

[3] Gelman, Andrew. "Scaling regression inputs by dividing by two standard deviations." Statistics in medicine 27.15 (2008): 2865-2873. https://stat.columbia.edu/~gelman/research/published/standardizing7.pdf

[4] "Out-of-Sample Predictions". PyMC library documentation example. https://www.pymc.io/projects/examples/en/latest/generalized_linear_models/GLM-out-of-sample-predictions.html

[5] "GLM: Robust Linear Regression." PyMC library documentation example. https://www.pymc.io/projects/examples/en/latest/generalized_linear_models/GLM-robust.html

[6] Jospin, Laurent Valentin, et al. "Hands-on Bayesian neural networks—A tutorial for deep learning users." IEEE Computational Intelligence Magazine 17.2 (2022): 29-48. https://arxiv.org/pdf/2007.06823.pdf