Bayesian Statistics : Final Project

T. Amadei, T. Kirscher, A. Klein

January 17, 2024

1 Model

We consider the same model as in the previous assignments : for $t = 1, \ldots, T$,

$$y_t = u_t'\phi + x_t'\beta + \varepsilon_t, \quad \varepsilon_t \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma^2),$$

where x_t is a vector of regressors of dimension k and u_t is a vector of regressors of dimension ℓ. For simplicity, the variance of each regressor is normalized to 1.
We further simplify the model by setting $l = 0$, *i.e.* we do not consider the first vector of regressors $(u_t)_t$. Hence, our model follows the formula:

$$y_t = x_t'\beta + \varepsilon_t, \quad \varepsilon_t \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma^2),$$

The prior for $\theta := (\sigma^2, \phi', \beta') \in \mathbb{R}^{1+\ell+k}$ is specified as in the previous assignements: $\pi(\theta) = \pi(\sigma^2) \times \pi(\phi) \times \pi(\beta \mid \sigma^2, \gamma^2, q)$ with

$$\begin{aligned} \pi\left(\sigma^2\right) &= \frac{1}{\sigma^2} \\ \pi(\phi) &= \text{ flat} \\ \pi\left(\beta_i \mid \sigma^2, \gamma^2, q\right) &\overset{\text{i.i.d.}}{\sim} \begin{cases} \mathcal{N}\left(0, \sigma^2\gamma^2\right) & \text{with probability } q \\ 0 & \text{with probability } 1-q \end{cases} \quad \text{for } i = 1, \ldots, k \end{aligned}$$

and γ^2, q are hyperparameters taking respectively nonnegative values and values in $[0,1]$. In addition, we specify the prior for the hyperparameters q and γ^2 in the following way. First, we specify the mapping $(\gamma^2, q) \mapsto R^2(\gamma^2, q) := \frac{qk\gamma^2\bar{v}_x}{qk\gamma^2\bar{v}_x + 1}$, where $\bar{v}_x := \frac{1}{k}\sum_{j=1}^{k} \widehat{\operatorname{Var}\left(x_{t,j}\right)}$ is the average sample variance of the predictors. Then, the prior for q and R^2 is

$$\begin{aligned} \pi(q) &\sim \text{Beta}(a, b) \\ \pi\left(R^2\right) &\sim \text{Beta}(A, B). \end{aligned}$$

2 Analysis

2.1 Dataset Description

We used the dataset called 'Hill-Valley'. This dataset contains no missing value, and its simplicity and clarity made it commonly used for evaluating the performance of classification algorithms.
Each sample represents a unique "hill" or "valley" configuration in a two-dimensional landscape. Each record represents 100 points on a two-dimensional graph. When plotted in order (from 1 through 100) as the Y co-ordinate, the points will create either a Hill (a "bump" in the terrain) or a Valley (a "dip" in the terrain).

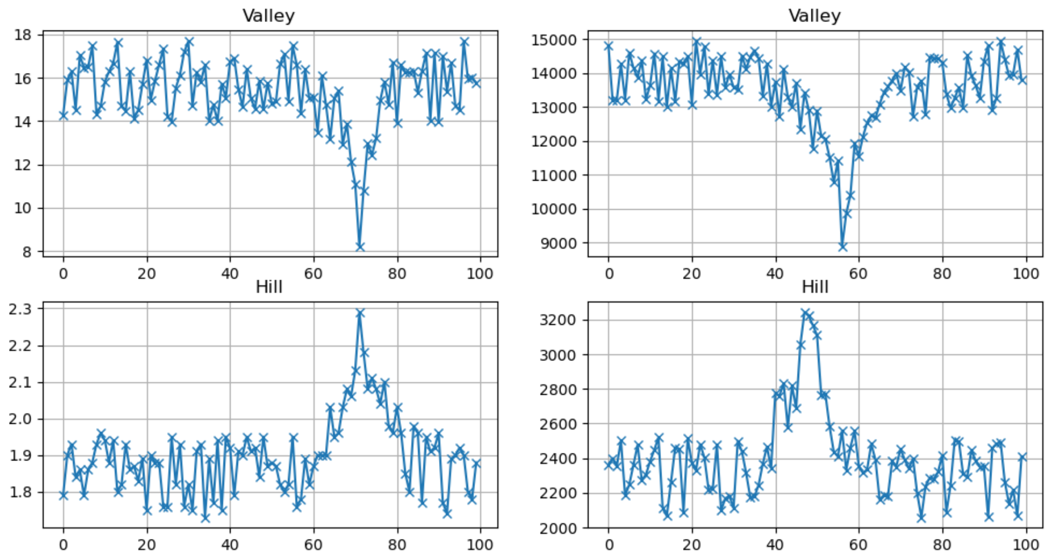

Figure 1: Plotting features from random hills and valleys from the Hill-Valley dataset

If we look at the average feature-wise of all valleys and all hills, we can see that the 'dips' of 'bumps' appear at a rather big range of places, hence each sample is unique.

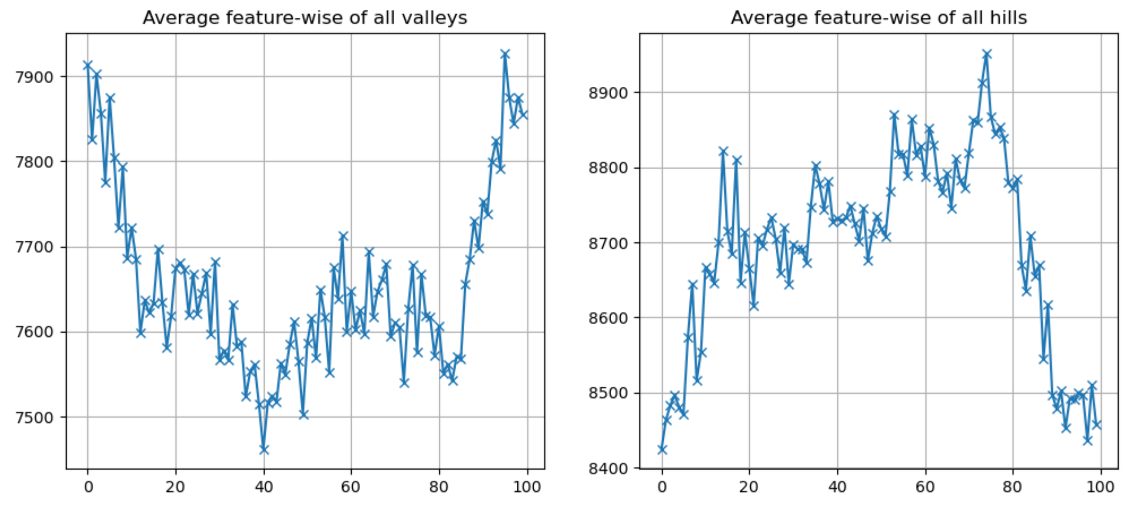

Figure 2: Plotting features from feature-wise averages of hills and valleys

The dataset contains 1212 samples, all containing 100 features and a class, with a perfect distribution between classes: each class contains exactly 50% of all samples.

2.2 Analysis to perform

The goal of our project will be train the model to do a classification: given a new sample with a 100 features, our model should be able to classify it as accurately as possible into the right class, either Valley or Hill.

2.3 From regression to classification

This dataset corresponds to a binary classification. However, our model corresponds to that of a regression. We can then consider that our model will do a regression to approximate the decision boundary of the model, *i.e.* the line that separates the space into two halfspaces, where our model then classifies samples either as valley or hill regarding the halfspace they are in.

Concretely, we will run our regression on Y, the vector containing elements in $\{0, 1\}$, with 0 representing the class 'Valley' and 1 the class 'Hill'. We will then apply a sigmoid function on the vector predicted with our regression, to map those elements into the probability of belonging to class 1 (the probability of belonging to class 0 being 1 minus that probability). Finally, to classify the samples to predict, we will simply classify as 1 those that have a predicted probability higher than $\frac{1}{2}$, otherwise we will classify them as 0.
On top of the classification, this approach also allows us to have access to the predicted probability of belonging to each class, hence giving us some more insight on how confident to model is when making a prediction.

3 Results

We used Python and Jupyter Notebooks to implement our approach. Our code can be found on this GitHub repository.

3.1 Regression results

To get the right values for the parameters of our model, we ran the Gibbs Sampler for 6000 iterations, with a burn-in of the first 1000 iterations. We could then compute the value of Y using

$$\forall t \in \{1, .., T\}, y_t = x_t' \beta$$

We then compute

$$\forall t \in \{1, .., T\}, p_t = \sigma(y_t),$$

where $\forall x \in \mathbb{R}, \sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function, which allows us to map the computed values of Y into probabilities.

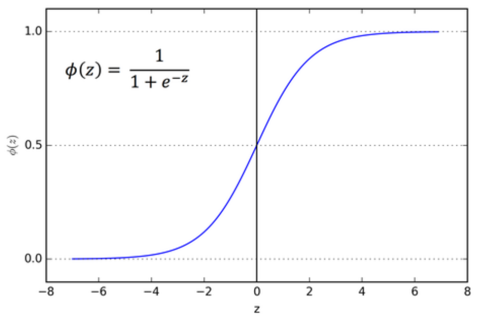

Figure 3: Sigmoid function

We can use those probabilities to compute the log-loss:

$$L\left((y_t)_T, (p_t)_T\right) = -\frac{1}{T} \sum_{t=1}^{T} \left[y_t \log(p_t) + (1 - y_t) \log(1 - p_t)\right]$$

And finally, we get the classification through:

$$\forall t \in \{1, .., T\}, \text{class}_t = \mathbf{1}\left(p_t > \frac{1}{2}\right)$$

Our vector Y contains the classes of the elements of the dataset, with the class 0 representing 'Valley' and the class 1 representing 'Hill'. Our vector X has 1212 samples with 100 features, and it is standardized, *i.e.* the variance of each regressor is normalized to 1.

To run the Gibbs sampler, we also need to set the values of s and R_y. To select the optimal ones, we ran several Gibbs sampler for each possible value, we then computed the log-loss and finally we selected the values that gave the lowest value for the loss. We tested

$$\begin{cases} R_y \in [0.02, 0.14, 0.26, 0.38, 0.5] \\ s \in [5, 10, 25, 50, 100] \end{cases}$$

At the end of our search, we found that the best values were $R_y = 0.02$ and $s = 5$.
For the model with those values, we plotted the evolution of the log-loss and the accuracy through the iterations of the Gibbs sampler. We can see that the curves are quite rugged, but still going in the right directions.

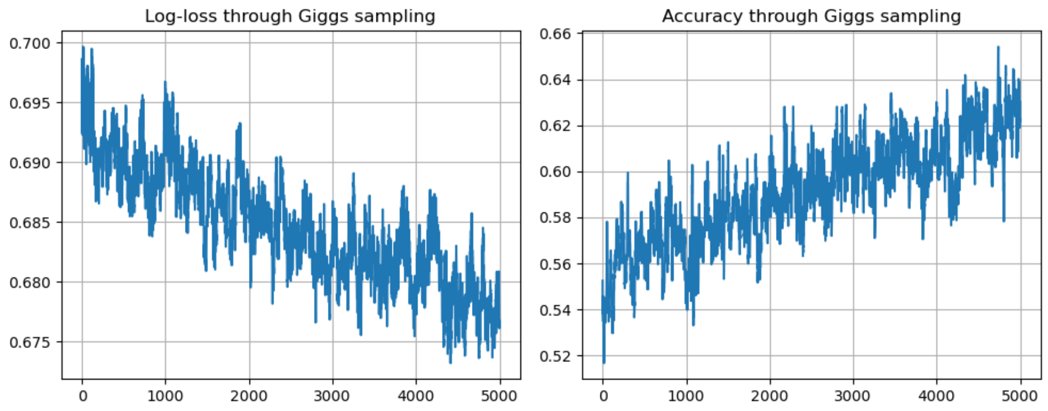

Figure 4: Sigmoid function

After the Gibbs sampler had finished running, the model was able to reach :

$$\begin{cases} \text{log-loss} = 0.676 \\ \text{accuracy} = 63.014\% \end{cases}$$

as we can see on Figure 4 above.

3.2 Comparison to Logistic Regression

Our approach resembles that of a Logistic Regression, with the difference that our model is linear, unlike the Logistic Regression. Hence, for the sake of comparison, we wanted to train a Logistic Regression model onto our data to see what the difference would be.
We did a grid search cross-validation to find the best hyperparameters: after tuning the hyperparameters, our model was able to get an accuracy of around 95.8%, with a log-loss of 0.09.

3.3 Conclusion

Clearly, the model is not linear, thus why the Logistic Regression is able to reach better performances. However, with the addition of Bayesian priors and the Gibbs sampler to compute the values of the parameters of the model, the model is able to perceive some underlying patterns within the data to allow it to outperform a random model by quite some margin.
Finally, we wanted to compare this Bayesian approach to a standard linear regression, mapped into a sigmoid function to then classify the samples. This approach was struggling to get any good result, with or without penalty like Lasso or Ridge, with the best model barely reaching an accuracy of 53%. Hence, we can conclude that the addition of the priors and the Gibbs sampler brings useful information for the model to perform better.

End of document.