Learning Models with Simulation: Project 1

Instructions

Answer all questions. Save the Python code you have written as a Python script or Jupyter Notebook. All plots and discussion could be included in the Jupyter Notebook or be saved separately as a document. Submission your solutions on our Moodle course page.

Bayesian logistic regression on the German credit dataset

This project considers the use of classifiers to do credit risk scoring. We will fit a logistic regression model to classify whether individuals have good or bad credit risks ($Y = 0$ or $Y = 1$ respectively) given a set of attributes. A detailed description of the dataset we will consider is given on this webpage `https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)` and the dataset can be found in the `Project1` folder on Moodle. This dataset, commonly known as the German credit dataset, is considered a benchmark dataset in the statistics and machine learning community to test classifiers. There are 1000 individuals and 24 features extracted from 20 different attributes in this dataset.

1 Reading and formatting dataset

1. Read the German credit dataset using the `read_csv` function from the pandas package. Store the output of the `read_csv` function as the variable `dataset`.

2. The last column of `dataset` contains the credit risk of the 1000 individuals. A value of 1 and 2 indicates good and bad credit risk respectively. We will use data for the first $m = 800$ individuals as our training dataset and the last $n = 200$ individuals as our testing dataset. For convenience, we will use a value of 0 and 1 to indicate good and bad credit risk respectively. Store the credit risk training and testing data as NumPy vectors `ytrain` and `ytest`, respectively.

3. Center and scale the other columns of `dataset` that contain 24 features of the 1000 individuals. Store the first $m = 800$ rows of the centered and scaled features as a NumPy array `xtrain`, and the last $n = 200$ rows as a NumPy array `xtest`.

4. Extend the NumPy arrays `xtrain` and `xtest` by adding a column of ones to the first column.

2 Model specification

The logistic regression model specifies that the probability of an individual having bad credit risk, i.e. $Y = 1$, given his or her features $X = (X_1, \ldots, X_d)$ and a vector of regression coefficients $\beta = (\beta_0, \beta_1, \ldots, \beta_d)$ is

$$\begin{aligned} P(Y = 1|\beta) &= \operatorname{expit}\left(\beta_0 + \sum_{j=1}^{d} \beta_j X_j\right) \\ &= 1 - P(Y = 0|\beta) \end{aligned} \tag{1}$$

where $\operatorname{expit} : \mathbb{R} \to [0, 1]$ is the logistic sigmoid function defined as

$$\operatorname{expit}(z) = \frac{1}{1 + \exp(-z)}, \quad z \in \mathbb{R}. \tag{2}$$

1. To understand the model (1) better, show that the log-odds ratio satisfies

$$\log \frac{P(Y = 1|\beta)}{P(Y = 0|\beta)} = \beta_0 + \sum_{j=1}^{d} \beta_j X_j. \tag{3}$$

2. Using the expression (3), interpret the parameters β_0 and β_j for $j = 1, \ldots, d$.

3. Write down the decision boundary of the logistic regression model (1).

4. Show that the log-likelihood of the $i = 1, \ldots, m$ training data point $x_i = (x_{i,0}, x_{i,1}, \ldots, x_{i,d}) \in \mathbb{R}^{d+1}$ with $x_{i,0} = 1$ and $y_i \in \{0, 1\}$ is

$$\log p(y_i|\beta) = y_i \beta^\top x_i - \log(1 + \exp(\beta^\top x_i)),$$

   where $\beta^\top x_i = \beta_0 + \sum_{j=1}^{d} \beta_j x_{i,j}$.

5. If the training data points $(x_1, y_1), \ldots, (x_m, y_m) \in \mathbb{R}^{d+1} \times \{0, 1\}$ are independent, show that the log-likelihood of the training dataset is

$$\log p(y_1, \ldots, y_m|\beta) = \sum_{i=1}^{m} y_i \beta^\top x_i - \sum_{i=1}^{m} \log(1 + \exp(\beta^\top x_i)). \tag{4}$$

   Write a JAX-compatible function that evaluates the log-likelihood (4) for any $\beta = (\beta_0, \beta_1, \ldots, \beta_d) \in \mathbb{R}^{d+1}$. This function should take as argument `beta` a vector of size $d + 1$ and output a numerical value. Investigate whether the `jit` function from the JAX package helps to speed up log-likelihood evaluations. Use the `jit` implementation if the speedup is significant.

6. Using the `grad` function from the JAX package that performs automatic differentiation, define a function that evaluates the gradient of the log-likelihood $\nabla \log p(y_1, \ldots, y_m|\beta)$ for any $\beta = (\beta_0, \beta_1, \ldots, \beta_d) \in \mathbb{R}^{d+1}$. This function should take as argument `beta` a vector of size $d + 1$ and output a vector of size $d + 1$. Investigate whether the `jit` function from the JAX package helps to speed up gradient of the log-likelihood evaluations. Use the `jit` implementation if the speedup is significant.

7. We consider a Bayesian approach to learn the parameters β. Following Hanson, Branscum and Johnson (2014), we will use the prior distribution

$$\beta \sim N\left(0, \Sigma\right), \quad \Sigma = \frac{\pi^2 m}{3(d+1)}(\mathbf{X}^\top \mathbf{X})^{-1}, \tag{5}$$

where π denotes the mathematical constant, $m = 800$, $d = 24$ and $\mathbf{X} \in \mathbb{R}^{m \times (d+1)}$ denotes the extended training data array `xtrain`. Write a JAX-compatible function that evaluates the log-prior density $\log p(\beta) = \log N(\beta; 0, \Sigma)$ for any $\beta = (\beta_0, \beta_1, \ldots, \beta_d) \in \mathbb{R}^{d+1}$. This function should take as argument `beta` a vector of size $d + 1$ and output a numerical value. Investigate whether the `jit` function from the JAX package helps to speed up log-prior density evaluations. Use the `jit` implementation if the speedup is significant.

8. Using the `grad` function from the JAX package that performs automatic differentiation, define a function that evaluates the gradient of the log-prior $\nabla \log p(\beta)$ for any $\beta = (\beta_0, \beta_1, \ldots, \beta_d) \in \mathbb{R}^{d+1}$. This function should take as argument `beta` a vector of size $d + 1$ and output a vector of size $d + 1$. Investigate whether the `jit` function from the JAX package helps to speed up gradient of the log-prior evaluations. Use the `jit` implementation if the speedup is significant.

9. Since the posterior distribution

$$p(\beta|y_1, \ldots, y_m) \propto p(\beta)p(y_1, \ldots, y_m|\beta) \tag{6}$$

is not a standard distribution, we will rely on Markov chain Monte Carlo methods to sample from it. Using the functions you have written, write a function that evaluates the unnormalized log-posterior density

$$\log \bar{p}(\beta|y_1, \ldots, y_m) = \log p(\beta) + \log p(y_1, \ldots, y_m|\beta)$$

for any $\beta = (\beta_0, \beta_1, \ldots, \beta_d) \in \mathbb{R}^{d+1}$. This function should take as argument `beta` a vector of size $d + 1$ and output a numerical value.

10. We will use Markov chain Monte Carlo algorithms that can exploit gradient information in the following. Using the functions you have written, write a function that evaluates the gradient of the unnormalized log-posterior density

$$\nabla \log \bar{p}(\beta|y_1, \ldots, y_m) = \nabla \log p(\beta) + \nabla \log p(y_1, \ldots, y_m|\beta)$$

for any $\beta = (\beta_0, \beta_1, \ldots, \beta_d) \in \mathbb{R}^{d+1}$. This function should take as argument `beta` a vector of size $d + 1$ and output a vector of size $d + 1$.

3 Markov chain Monte Carlo sampling

1. Implement an independent Metropolis–Hastings algorithm with the prior distribution $p(\beta)$ as the proposal distribution. Initialize the Markov chain using a sample from the prior distribution and run the algorithm for $N = 10,000$ iterations. Report the acceptance rate that you observe and comment on the performance of the algorithm using diagnostic plots.

2. Implement a random walk Metropolis–Hastings algorithm with the proposal transition density $q(\beta'|\beta) = N(\beta'; \beta, \sigma^2 I_{d+1})$. Initialize the Markov chain using a sample from the prior distribution and run the algorithm for $N = 10,000$ iterations. Experiment with the choice of $\sigma \in \{0.002, 0.02, 0.2\}$. Using the acceptance rates that you observe and diagnostic plots, explain which of the three choice of σ is the best.

3. Implement a Metropolis-adjusted Langevin algorithm which is defined by the proposal transition density

$$q(\beta'|\beta) = N\left(\beta'; \beta + \frac{\sigma^2}{2}\nabla \log p(\beta|y_1, \ldots, y_m), \sigma^2 I_{d+1}\right).$$

Initialize the Markov chain using a sample from the prior distribution and run the algorithm for $N = 10,000$ iterations. Experiment with the choice of σ between 0.04 to 0.12. Using the acceptance rates that you observe and diagnostic plots, what is a good choice of σ?

4. Implement a Hamiltonian Monte Carlo algorithm by using the `hamiltonian_dynamics` function in the Python script `hmc.py` (in the `Project1` folder on Moodle) to simulate Hamiltonian dynamics. Initialize the Markov chain using a sample from the prior distribution and run the algorithm for $N = 10,000$ iterations. Experiment with the choice of stepsize between 0.01 to 0.1. Using the acceptance rates that you observe, what is a good choice of stepsize? Using diagnostic plots, compare the performance of the algorithm with 5 or 10 number of steps. To do model predictions in the following, perform a final run of the Markov chain, with the best stepsize that you found and 10 number of steps, for 11,000 iterations and discard the first 1000 iterations as burn-in.

4 Model predictions

1. The Bayesian approach to prediction is based on the predictive probability

$$P(Y = 1|X, y_1, \ldots, y_m) = \int_{\mathbb{R}^{d+1}} \text{expit}\left(\beta_0 + \sum_{j=1}^{d} \beta_j X_j\right) p(\beta|y_1, \ldots, y_m) d\beta,$$

where $X = (X_1, \ldots, X_d)$ denotes the features of an individual in the testing dataset. Given Markov chain Monte Carlo samples $\beta^{(t)}, t = 1, \ldots, N$, explain why it is sensible to approximate the predictive probability by

$$\widehat{P}(Y = 1|X, y_1, \ldots, y_m) = \frac{1}{N}\sum_{t=1}^{N} \text{expit}\left(\beta_0^{(t)} + \sum_{j=1}^{d} \beta_j^{(t)} X_j\right).$$

2. Using the Hamiltonian Monte Carlo samples from your final run in Question 3.4, approximate the predictive probabilities for all $n = 200$ individuals in the testing dataset stored in the NumPy array `xtest`.

3. Using these predictive probabilities, implement the prediction rule

$$\widehat{Y}(X) = \begin{cases} 1, & \text{if } \widehat{P}(Y = 1|X, y_1, \ldots, y_m) > 0.5, \\ 0, & \text{if } \widehat{P}(Y = 1|X, y_1, \ldots, y_m) < 0.5, \end{cases}$$

where $X = (X_1, \ldots, X_d)$ denotes the features of an individual in the testing dataset.

4. Using the actual classification stored in the NumPy vector `ytest`, compute the misclassification rate.

5. In this application, it is worse to classify an individual as having good credit risk when they are bad, than it is to classify an individual as having bad credit risk when they are good. These differences are captured using the following cost function

$$C(\widehat{Y}, Y) = \begin{cases} 5, & \text{if } \widehat{Y} = 0 \text{ and } Y = 1, \\ 1, & \text{if } \widehat{Y} = 1 \text{ and } Y = 0, \\ 0, & \text{otherwise.} \end{cases}$$

   Using the actual classification stored in the NumPy vector `ytest`, compute the average cost.

6. We will compare Bayesian predictions with a maximum likelihood approach that would predict using the rule

$$\hat{Y}(X) = \begin{cases} 1, & \text{if expit} \left(\beta_0^{(\text{MLE})} + \sum_{j=1}^{d} \beta_j^{(\text{MLE})} X_j \right) > 0.5, \\ 0, & \text{if expit} \left(\beta_0^{(\text{MLE})} + \sum_{j=1}^{d} \beta_j^{(\text{MLE})} X_j \right) < 0.5, \end{cases}$$

   where $X = (X_1, \ldots, X_d)$ denotes the features of an individual in the testing dataset, and $\beta^{(\text{MLE})} = (\beta_0^{(\text{MLE})}, \beta_1^{(\text{MLE})}, \ldots, \beta_d^{(\text{MLE})})$ defined as

$$\beta^{(\text{MLE})} = \arg \max_{\beta \in \mathbb{R}^{d+1}} \log p(y_1, \ldots, y_m | \beta)$$

   is the maximum likelihood estimator. Compute the maximum likelihood estimator using the `minimize` function from the SciPy `optimize` subpackage. Initialize the optimization routine using a sample from the prior distribution and pass the gradient of the log-likelihood function by specifying the `jac` parameter. After obtaining the maximum likelihood estimator, implement the prediction rule for all $n = 200$ individuals in the testing dataset stored in the NumPy array `xtest`.

7. Using the actual classification stored in the NumPy vector `ytest`, compute the misclassification rate and average cost under the maximum likelihood approach.

8. Compare the prediction accuracy obtained using the Bayesian and maximum likelihood approaches, and explain your findings.