Non-linear Optimisation Assignment 1

Deadline: 23:59, November 19, 2024

1 Practicalities

This assignment should be made **individually**. Cooperation with others is not allowed: you should never see the code of other students. The use of **generative AI** for this assignment is also not allowed, and will be considered fraud.

Emails or forum questions sent less than 24 hours before the deadline will not be answered. You can ask questions about the content of this assignment during the **tutorial in Week 3**.

Hand in this assignment via **CodeGrade**. If you do not hand in anything before the deadline, you will receive the grade 0. **You can submit to CodeGrade at most five times**; your final submission will determine your grade. Before uploading your code to CodeGrade, make sure you tested and debugged your code on your own device.

Every submission should be a single Python script called 'main.py' that contains all functions. Your code can use the packages numpy, pandas, openpyxl, and/or matplotlib. Whenever this assignment mentions 'vectors' and 'matrices', work with appropriately sized `numpy.ndarray` objects. For vectors, the attribute `ndim` should be one, for matrices it should be two.

All functions should have a **docstring**. This is worth **1 point** in total.

All code has to be implemented for **general inputs** (unless specified otherwise). In other words, your code may not depend on specific characteristics of the provided data set such as the number of observations or the number of features.

All your submissions are automatically checked for **plagiarism**. If similarities with the work of other students are found (which is to be expected for this type of assignment), you may be invited to answer questions about your code. If such a conversation raises no problems, or you are not invited for a conversation, you will get the final grade that CodeGrade returns.

2 Surviving the Titanic

In April 1912, the Titanic set out from Southampton to New York City, picking up passengers in Cherbourg and Queenstown before venturing across the Atlantic. The ship famously never made it to New York. On the night of 14 April, the Titanic hit an iceberg and sank a few hours later. More than 1500 of the estimated 2224 passengers and crew on board perished.

Whether a passenger survived was not just a matter of luck. In this assignment, we will try to predict which passengers were likely to survive.

The data on Canvas is based on the Titanic data set from OpenML with ID 40945. It contains information on 1043 passengers of the Titanic. The features of the data set are described in Table 1.

Feature	Description
name	Passenger name
survived	1 if the passenger survived, 0 otherwise
pclass	Class the passenger was in (1 for first, 2 for second, 3 for third)
female	1 if the passenger was female, 0 otherwise
age	Passenger age in years
sibsp	Number of siblings and spouses on board
parch	Number of parents and children on board
fare	Passenger fare
embarkS	1 if the passenger embarked in Southampton, 0 otherwise
embarkC	1 if the passenger embarked in Cherbourg, 0 otherwise
embarkQ	1 if the passenger embarked in Queenstown, 0 otherwise

Table 1: Description of the passenger data on Canvas

We will try to predict the value of 'survived' by the other variables in the data set – except 'name', because is not numerical. We therefore define N to be the number of passengers, and let

$$y_i = \begin{cases} 1 & \text{if passenger } i \text{ survived} \\ 0 & \text{otherwise,} \end{cases}$$

for all $i = 1, \ldots, N$. The explanatory variables for this passenger are collected in the vector $\boldsymbol{x}_i \in \mathbb{R}^p$. As usual, we gather the information for all passengers

in the objects

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \quad \text{and} \quad X = \begin{bmatrix} \boldsymbol{x}_1^\top \\ \vdots \\ \boldsymbol{x}_N^\top \end{bmatrix}.$$

1. **(0 points)** Load the numerical data in Python. Create the vector $\boldsymbol{y}$ and the matrix X. Make sure this process is easy to repeat, because you will need this data throughout the rest of the assignment to test your code.

3 Logistic regression

Because we are trying to predict the binary variable y_i, we introduce the sigmoid function $\Phi : \mathbb{R} \to (0,1)$ defined by

$$\Phi(t) = \frac{1}{1+e^{-t}}. \tag{1}$$

A plot of Φ is provided in Figure 1. It shows that when $t \ll 0$, then $\Phi(t)$ is nearly zero, and if $t \gg 0$, then $\Phi(t)$ is close to one. This function is therefore suitable to convert some underlying variable to a survival probability. Besides, this function has the useful properties that

$$1 - \Phi(t) = \frac{\Phi(t)}{e^t} = \Phi(-t), \tag{2}$$

which we will use below.

We will model the survival probability of passenger $i = 1, \ldots, N$ by

$$p_i = \Phi\left(w_0 + \boldsymbol{w}^\top \boldsymbol{x}_i\right),$$

where $w_0 \in \mathbb{R}$ and $\boldsymbol{w} \in \mathbb{R}^p$ are model parameters we will determine later. Given the survival probability p_i, the likelihood of observation i is

$$\ell_i := \begin{cases} p_i & \text{if } y_i = 1 \\ 1 - p_i & \text{if } y_i = 0. \end{cases}$$

Because of the simple structure of this likelihood formula, we can write the

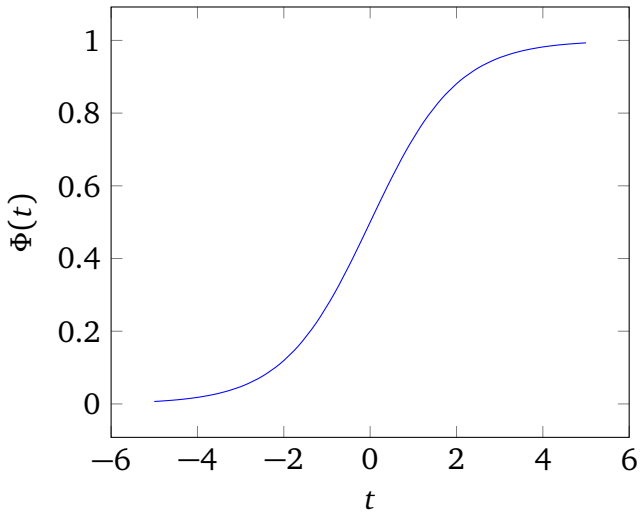

Figure 1: Sigmoid function (1).

log-likelihood function as

$$\begin{aligned} f(w_0, \boldsymbol{w}) &= \sum_{i=1}^{N} \log \ell_i \\ &= \sum_{i=1}^{N} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] \\ &= \sum_{i=1}^{N} \left[y_i \log\left(\frac{p_i}{1 - p_i} \right) + \log(1 - p_i) \right]. \end{aligned}$$

Using both equalities from (2), we can further simplify this to

$$\begin{aligned} f(w_0, \boldsymbol{w}) &= \sum_{i=1}^{N} \left[y_i \log\left(\frac{p_i}{p_i} e^{w_0 + \boldsymbol{w}^\top \boldsymbol{x}_i} \right) + \log\left(\Phi\left(-w_0 - \boldsymbol{w}^\top \boldsymbol{x}_i \right) \right) \right] \\ &= \sum_{i=1}^{N} \left[y_i \left(w_0 + \boldsymbol{w}^\top \boldsymbol{x}_i \right) - \log\left(1 + e^{w_0 + \boldsymbol{w}^\top \boldsymbol{x}_i} \right) \right]. \end{aligned} \tag{3}$$

We will try to **maximize** this log-likelihood over w_0 and $\boldsymbol{w}$. To this end,

note that the gradient of the log-likelihood is

$$\begin{aligned}\nabla f(w_0, \boldsymbol{w}) &= \sum_{i=1}^{N} \left(y_i \begin{bmatrix} 1 \\ \boldsymbol{x}_i \end{bmatrix} - \frac{e^{w_0 + \boldsymbol{w}^\top \boldsymbol{x}_i}}{1 + e^{w_0 + \boldsymbol{w}^\top \boldsymbol{x}_i}} \begin{bmatrix} 1 \\ \boldsymbol{x}_i \end{bmatrix} \right) \\ &= \sum_{i=1}^{N} \left(y_i - \Phi\left(w_0 + \boldsymbol{w}^\top \boldsymbol{x}_i\right) \right) \begin{bmatrix} 1 \\ \boldsymbol{x}_i \end{bmatrix}. \end{aligned} \tag{4}$$

It can be shown that $\Phi'(t) = \Phi(t)[1 - \Phi(t)]$, which implies that the Hessian of the log-likelihood is

$$\nabla^2 f(w_0, \boldsymbol{w}) = -\sum_{i=1}^{N} \Phi\left(w_0 + \boldsymbol{w}^\top \boldsymbol{x}_i\right) \left[1 - \Phi\left(w_0 + \boldsymbol{w}^\top \boldsymbol{x}_i\right)\right] \begin{bmatrix} 1 \\ \boldsymbol{x}_i \end{bmatrix} \begin{bmatrix} 1 \\ \boldsymbol{x}_i \end{bmatrix}^\top. \tag{5}$$

For convenience of notation, we will combine the arguments of the log-likelihood in one vector

$$\bar{\boldsymbol{w}} \equiv \begin{bmatrix} w_0 \\ \boldsymbol{w} \end{bmatrix}.$$

2. **(2 points)** Create a function `func_loglikelihood(wbar, X, y)` in Python that returns the value of (3). The function arguments are $\bar{\boldsymbol{w}} \in \mathbb{R}^{p+1}$, $X \in \mathbb{R}^{N \times p}$, and $\boldsymbol{y} \in \mathbb{R}^N$, respectively.

3. **(2 points)** Create a function `grad_loglikelihood(wbar, X, y)` in Python that returns the value of (4). The function arguments are $\bar{\boldsymbol{w}} \in \mathbb{R}^{p+1}$, $X \in \mathbb{R}^{N \times p}$, and $\boldsymbol{y} \in \mathbb{R}^N$, respectively.

4. **(2 points)** Create a function `hes_loglikelihood(wbar, X)` in Python that returns the value of (5). The function arguments are $\bar{\boldsymbol{w}} \in \mathbb{R}^{p+1}$ and $X \in \mathbb{R}^{N \times p}$, respectively.

4 Wolfe conditions

We will use line search to approximate stationary points of the log-likelihood function. The step lengths will be chosen to satisfy the Wolfe conditions. Algorithm 1 provides pseudocode with which we can find such step lengths.

Algorithm 1 An algorithm to satify the Wolfe conditions

Input: Continuously differentiable $\phi : \mathbb{R} \to \mathbb{R}$ with $\phi'(0) < 0$ that is bounded below; constants $0 < c_1 < c_2 < 1$; initial step length $\alpha_1 > 0$.
Output: Step length $\alpha > 0$ such that $\phi(\alpha) \leq \phi(0) + c_1\alpha\phi'(0)$ and $\phi'(\alpha) \geq c_2\phi'(0)$.

Initialize $\alpha_0 \leftarrow 0$ and $k \leftarrow 1$ and $L \leftarrow 0$ and $U \leftarrow \infty$
while $\phi(\alpha_k) > \phi(0) + c_1\alpha_k\phi'(0)$ or $\phi'(\alpha_k) < c_2\phi'(0)$ **do**
    if $\phi(\alpha_k) > \phi(0) + c_1\alpha_k\phi'(0)$ or $\phi'(\alpha_k) \geq 0$ or $\phi(\alpha_k) \geq \phi(\alpha_{k-1})$ **then**
        $U \leftarrow \alpha_k$ if $\phi'(\alpha_k) \geq 0$; otherwise $L \leftarrow \alpha_k$
        $\alpha_{k+1} \leftarrow 2\alpha_k$ if $U = \infty$; otherwise $\alpha_{k+1} \leftarrow \frac{1}{2}(L + U)$
    else
        $\alpha_{k+1} \leftarrow 2\alpha_k$
    end if
    $k \leftarrow k + 1$
end while
return α_k

5. **(4 points)** Create a function

```
step_length_Wolfe(func, grad, start, direction, param)
```

in Python. The respective function arguments are, for some value of n, the following:

- The function $g : \mathbb{R}^n \to \mathbb{R}$ to minimize;
- The gradient of g;
- The starting point $\boldsymbol{s} \in \mathbb{R}^n$ of the line search;
- The search direction $\boldsymbol{d} \in \mathbb{R}^n$;
- A tuple containing the values c_1, c_2, and α_1, in that order.

The function should return a step length that satisfies the Wolfe conditions for $\phi(\alpha) = g(\boldsymbol{s} + \alpha\boldsymbol{d})$, computed using Algorithm 1.

Hint: Python allows you to define so-called inner functions.

5 Maximizing the log-likelihood

Usually, the log-likelihood can be maximized effectively using Newton's method. For this data set though, the Hessian (5) is not invertible. (Can you figure out why?) Python may still return something if you ask it for the inverse matrix, but you can check that its product with the original Hessian is far from the identity.

Because we cannot invert the Hessian, we will use the BFGS method to maximize the log-likelihood.

6. **(6 points)** Create a function

```
line_search_BFGS(func, grad, x0, H0, iteration_limit, \
epsilon, param)
```

   in Python that performs line search using the BFGS direction. The function should call `step_length_Wolfe` to compute the step lengths using the parameters in `param`. The respective function arguments are, for some value of n, the following:

   - The function to minimize, defined on $\mathbb{R}^n$;
   - The gradient of this function;
   - The starting point of the line search in $\mathbb{R}^n$;
   - The initial approximation of the inverse Hessian;
   - The maximum number of line search steps;
   - The gradient norm tolerance;
   - A tuple containing the values c_1, c_2, and α_1, in that order.

   The function should return a matrix whose columns are the visited iterates, excluding the starting point.

7. **(2 points)** Create a function `logistic_regression(X, y)` in Python that uses `line_search_BFGS` to maximize (3). The function arguments are $X \in \mathbb{R}^{N\times p}$ and $\boldsymbol{y} \in \mathbb{R}^N$, respectively.

   The line search should start from the zero vector. The initial inverse Hessian approximation is the Moore-Penrose pseudo-inverse of the Hessian

at this starting point (you can use `numpy.linalg.pinv`). The iteration limit is 1000, the tolerance is 10^{-4}, $c_1 = 10^{-3}$, $c_2 = 0.9$, and $\alpha_1 = 10^{-2}$.

The function should return a vector $\bar{\boldsymbol{w}}$ that approximately maximizes (3).

6 Predictions

You should now be able to call the function `logistic_regression(X, y)` with the data that you loaded in Exercise 1 to estimate the values of w_0 and $\boldsymbol{w}$. Let us call these estimates $\hat{w}_0$ and $\hat{\boldsymbol{w}}$, respectively.

We can use the estimated parameters to predict who was likely to survive the Titanic's maiden voyage. Our survival prediction for passenger $i = 1, \ldots, N$ is

$$\hat{y}_i = \begin{cases} 1 & \text{if } \Phi\left(\hat{w}_0 + \hat{\boldsymbol{w}}^\top \boldsymbol{x}_i\right) \geq \frac{1}{2} \\ 0 & \text{otherwise.} \end{cases}$$

These predictions can be combined in a vector $\hat{\boldsymbol{y}} \in \mathbb{R}^N$.

To determine how good our predictions are, we can compute the (in-sample) accuracy of our model. This is the fraction of observations that were predicted correctly; in other words, the fraction of observations $i = 1, \ldots, N$ for which $y_i = \hat{y}_i$.

8. **(2 points)** Create a function `logistic_prediction(X, wbar)` in Python that returns a prediction vector $\hat{\boldsymbol{y}}$. The function arguments are $X \in \mathbb{R}^{N \times p}$ and $(\hat{w}_0, \hat{\boldsymbol{w}}) \in \mathbb{R}^{p+1}$, respectively.

9. **(2 points)** Create a function `accuracy(y, yhat)` in Python that returns the accuracy of the model. The function arguments are $\boldsymbol{y} \in \mathbb{R}^N$ and $\hat{\boldsymbol{y}} \in \mathbb{R}^N$, respectively.