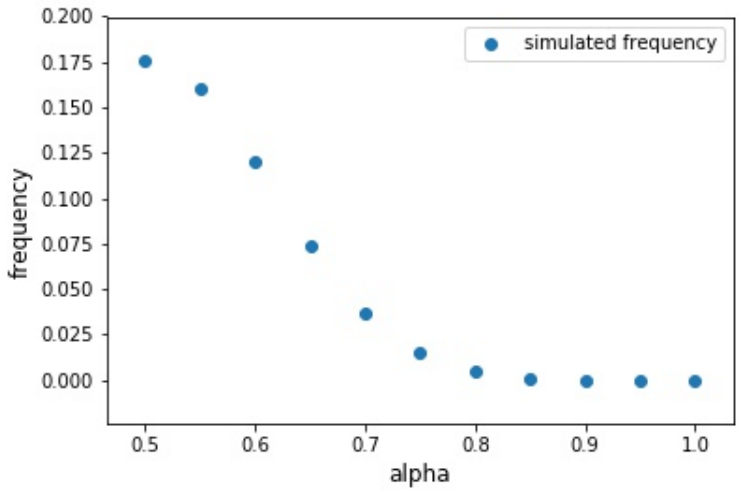
1 Illustration of Hoeffding's Inequality

I have simulated 1000000 repetitions of the experiment of drawing 20 i.i.d Bernoulli random variables $X_1, ..., X_{20}$ with bias $\frac{1}{2}$, and plotted the empirical frequency of observing $\overline{X} = \frac{1}{20}\sum_{i=1}^{20} X_i \geq \alpha$ for $\alpha \in \{0.5, 0.55, ..., 0.95, 1.0\}$. It is clear that since we only draw 20 random variables, and all of them have a value of 0 or 1, then the $\overline{X}$ will be a multiple of 0.05 between 0 and 1. Therefore, it would not provide additional information to choose a finer granularity of α than the chosen one. Here is my plot:



As expected from the bias of each X_i being $\frac{1}{2}$, the simulated frequency becomes lower and lower the further from 0.5 the value of α becomes. Due to the central limit theorem and the fact that we are simulating a big number of experiments, we should in fact be observing something that approximates the right half of a normal distribution with mean 0.5. This corresponds well with the actual plot.

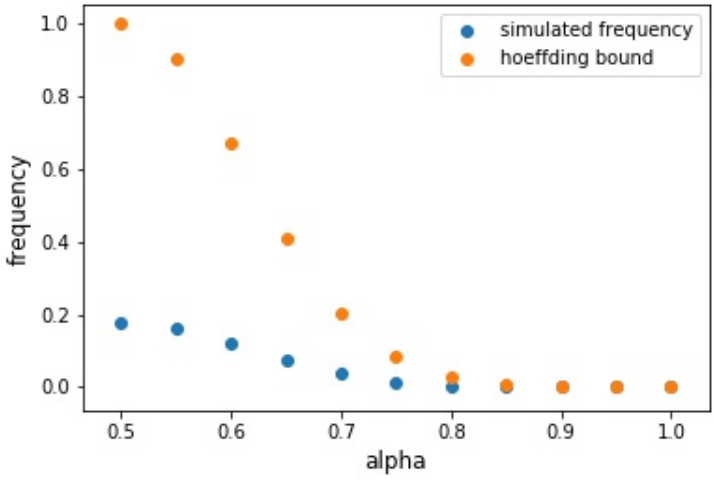
By substituting $\epsilon = \alpha - \mu$ and $\mathbb{E}(X_i) = \mu$ in the formulation of Hoeffding's Inequality in **corrolary 2.4**, we get:

$$\mathbb{P}\left(\sum_{i=1}^{n} X_i - n\mu \geq \alpha - \mu\right) \leq e^{2n(\alpha-\mu)^2} \tag{1}$$

which is equivalent to

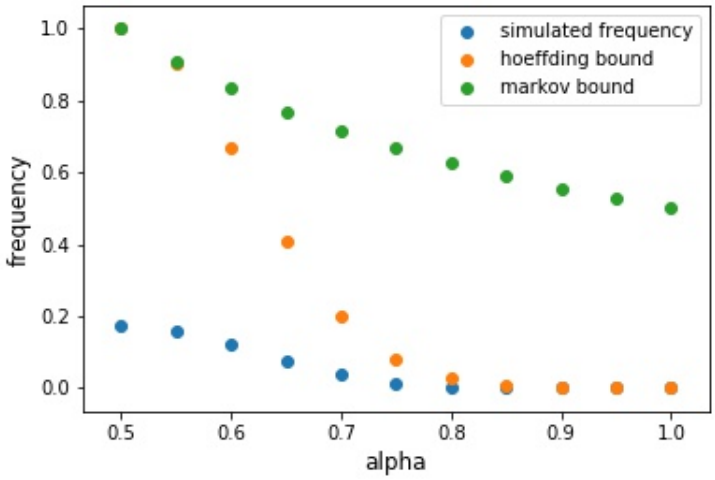
$$\mathbb{P}\left(\overline{X} \geq \alpha\right) \leq e^{2n(\alpha-\mu)^2} \tag{2}$$

By setting $\mu = 0.5$, we can use this formula to calculate the Hoeffding bound on the probability of observing $\overline{X} \geq \alpha$ as a function of α. Here is my plot of this bound and the simulated frequency of the event actually happening:



As expected, we see that the bound is always larger than the simulated frequency. We also see that the larger α becomes, the tighter the Hoeffding also becomes compared to the simulated frequency.

By substituting $\epsilon = \alpha$ and $\mathbb{E}(X_i) = 0.5$ in the formulation of Markov Inequality in **corrolary 2.1**, we can calculate the Markov bound on the probability of observing $\overline{X} \geq \alpha$ as a function of α. Here is my plot:



From this plot, we can see that Hoeffdings Inequality gives tighter bounds than Markovs Inequality.

We can calculate the probability that $\overline{X} = 1$ as

$$P(\overline{X} = 1) = \left(\frac{1}{2}\right)^{20} \approx 9.54 \cdot 10^{-7} \tag{3}$$

The Hoeffding bound on this probability is approximately $4.54 \cdot 10^{-5}$.

Using the binomial distribution, we can find the probability getting 1 or 0 successes on 20 tries with 0.5 success probability on each try. This is equivalent to the probability of $\overline{X} \geq 0.95$. The probability is approximately $2.01 \cdot 10^{-5}$. The Hoeffding bound on this probability is approximately $3.03 \cdot 10^{-4}$.

2 The effect of scale (range) and normalization of random variables in Hoeffding's Inequality

Let all the assumptions of **corollary 2.4** be true for some random variables $X_1, ..., X_n$. Set $a_i = 0$ and $b_i = 1$ for all $i \in \{1, ..., n\}$. By the assumptions of **corollary 2.4** and our definition of a_i and b_i, we now have that all the assumptions of **theorem 2.2** are true for $X_1, ..., X_n$. We can therefore use **theorem 2.2** to conclude that for all $\varepsilon > 0$

$$\mathbb{P}\left\{\sum_{i=1}^{n} X_i - \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] \geq \varepsilon\right\} \leq e^{-2\varepsilon^2 / \sum_{i=1}^{n}(b_i - a_i)^2} \tag{4}$$

and

$$\mathbb{P}\left\{\sum_{i=1}^{n} X_i - \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] \leq -\varepsilon\right\} \leq e^{-2\varepsilon^2 / \sum_{i=1}^{n}(b_i - a_i)^2} \tag{5}$$

Since the assumptions of the corollary states that $\mathbb{E}(X_i) = \mu$ for all $i \in \{1, ..., n\}$, then we know that

$$\mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = n\mu \tag{6}$$

Since $b_i - a_i = 1$ for all $i \in \{1, ..., n\}$, we also know that

$$\sum_{i=1}^{n}(b_i - a_i)^2 = n \tag{7}$$

From line (XX-XX) we therefore get that for all $\varepsilon > 0$

$$\mathbb{P}\left\{\sum_{i=1}^{n} X_i - n\mu \geq \varepsilon\right\} \leq e^{-2\frac{\varepsilon^2}{n}} = e^{-2n\left(\frac{\varepsilon}{n}\right)^2} \tag{8}$$

and

$$\mathbb{P}\left\{\sum_{i=1}^{n} X_i - n\mu \leq -\varepsilon\right\} \leq e^{-2\frac{\varepsilon^2}{n}} = e^{-2n\left(\frac{\varepsilon}{n}\right)^2} \tag{9}$$

From this it clearly follows that for all $\varepsilon > 0$

$$\mathbb{P}\left\{\frac{1}{n}\sum_{i=1}^{n} X_i - \mu \geq \frac{\varepsilon}{n}\right\} \leq= e^{-2n\left(\frac{\varepsilon}{n}\right)^2} \tag{10}$$

and

$$\mathbb{P}\left\{\frac{1}{n}\sum_{i=1}^{n} X_i - \mu \leq -\frac{\varepsilon}{n}\right\} \leq= e^{-2n\left(\frac{\varepsilon}{n}\right)^2} \tag{11}$$

If $\varepsilon > 0$, then $\tilde{\varepsilon} = \frac{\varepsilon}{n} > 0$ for all $n \in \mathbb{N}$. We can therefore now conclude that for all $\tilde{\varepsilon} > 0$

$$\mathbb{P}\left\{\frac{1}{n}\sum_{i=1}^{n} X_i - \mu \geq \tilde{\varepsilon}\right\} \leq= e^{-2n\tilde{\varepsilon}^2} \tag{12}$$

and

$$\mathbb{P}\left\{\frac{1}{n}\sum_{i=1}^{n} X_i - \mu \leq -\tilde{\varepsilon}\right\} \leq= e^{-2n\tilde{\varepsilon}^2} \tag{13}$$

We have now proven that **corollary 2.4** follows from **theorem 2.2**.

3 Probability in Practice

Question 1: We assume that the probability of each passenger showing up is 0.95, and that event of any passenger showing up is independent of whether any other of the passengers showed up or not. This means that the event of all 100 passengers showing up is equivalent to flipping a coin with bias 0.95 100 times and getting heads each time. The probability of this happening is

$$\left(\frac{95}{100}\right)^{100} \approx 0.006 \tag{14}$$

The flight company should therefore expect to get an overbooked flight on around 6 out 1000 departures.

Question 2: Maybe I have misunderstood the assignment text, but I think this question is kind of weird. As I understand the question, we should assume that the event of whether a passenger with a flight reservation shows up or not is drawn from a bernoulli distribution with a given probability p of showing up, and a probability $1 - p$ of not showing up. Using this assumption, we have to bound the probability that the following two events both happen:

- Event A: We i.i.d sample 1000 flight reservations from the distribution, and the passengers show up in exactly in 95 percent of them.
- Event B: We i.i.d sample 100 flight reservations from the distribution, and the passengers show up in all of them.

Since event A and event B are independent, then $P_p(A \wedge B) = P_p(A)P_p(B)$, where p is the probability of a single passenger showing up for a reservation. $P_p(A)P_p(B)$ probability is going to be very low, however, since the probability of observing *exactly* 95 percent of the passengers in the 10000 reservations not showing up is very low, no matter what p is. To be precise, this probability is equal to the value of the binomial distribution of 9500 success in 10000 experiments with a success probability p on each experiment. I denote this value by $\text{bin}(9500, 10000, p)$. In other words, we have:

$$P_p(A) = \text{bin}(9500, 10000, p) \tag{15}$$

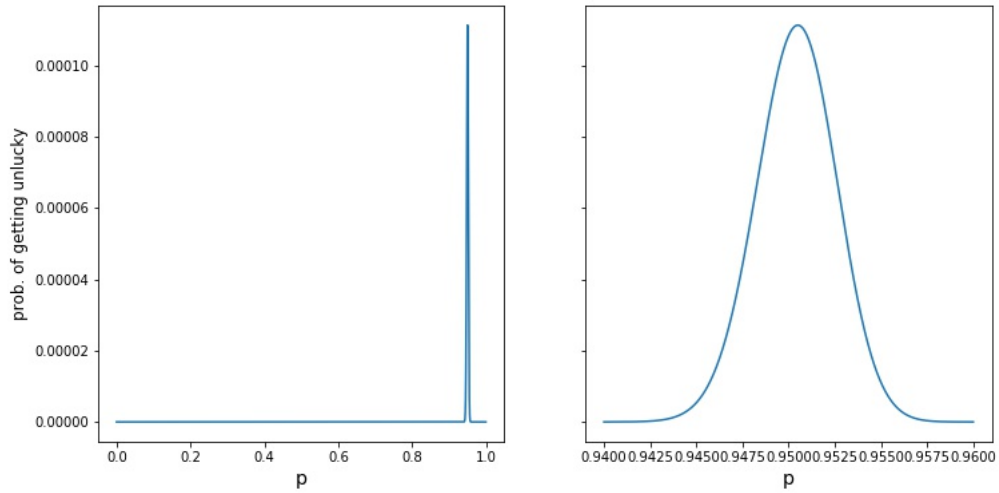
As in question 1, we also have that:

$$P_p(B) = p^{100} \tag{16}$$

All in all, we therefore have that:

$$P_p(A \wedge B) = P_p(A)P_p(B) = \text{bin}(9500, 10000, p)p^{100} \tag{17}$$

Here are two plots of the probability of the flight company getting unlucky, that is $P_p(A \wedge B)$, against the value of p. The plot on the right is just a zoomed in version of the plot on the left, only focusing on p from 0.94 to 0.96:



The value of $P_p(A \wedge B)$ is maximized for $\tilde{p} = 0.95005$ with a value of $P_{\tilde{p}}(A \wedge B) = 0.00011$. This means that for all $p \in [0, 1]$, we have that $P_p(A \wedge B) \leq 0.00011$. However, this says very little about anything interesting, since the low probability just comes from the fact that we are taking A as the event of observing *exactly* 95 percent of the passengers not showing up. A flight company would be completely uninterested in this probability, since they would never have a decision procedure of implementing the reservation policy of overbooking with 1 passenger, if and only if they observe exactly event A happening.

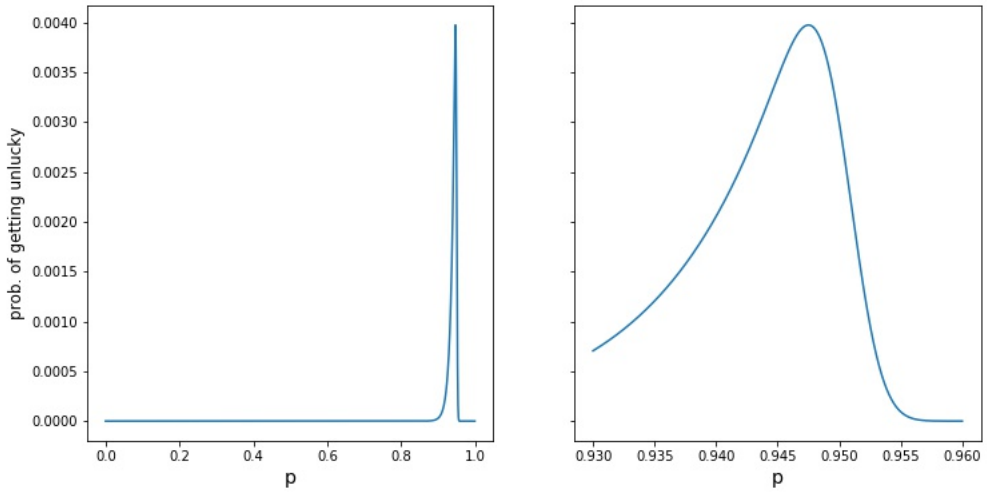
Another slighty more interesting, but still not very realistic scenario, is to ask for the probability of observing both event $\hat{A}$ and event B, where

- Event $\hat{A}$: We i.i.d sample 1000 flight reservations from the distribution, and the passengers show up in exactly in less than or exactly 95 percent of them.
- Event B: We i.i.d sample 100 flight reservations from the distribution, and the passengers show up in all of them.

Here we have that $P_p(\hat{A})$ is just the value of the cumulative binomial distribution of 9500 success in 10000 experiments with a success probability p on each experiment. I denote this value by $\text{cumbin}(9500, 10000, p)$. We therefore have:

$$P_p(\hat{A} \wedge B) = P_p(\hat{A})P_p(B) = \text{cumbin}(9500, 10000, p)p^{100} \tag{18}$$

Here are two plots of the probability of the flight company getting unlucky in this setting against the value of p:



The value of $P_p(\tilde{A} \wedge B)$ is maximized for $\tilde{p} = 0.94475$ with a value of $P_{\tilde{p}}(\hat{A} \wedge B) = 0.0039$. This means that for all $p \in [0, 1]$, we have that $P_p(\hat{A} \wedge B) \leq 0.0039$.

4 Logistic Regression

4.1 Cross-entropy measure

Let $\mathcal{X}$ be some sample space, and let $\mathcal{Y}$ be the label space $\{-1, 1\}$, and assume that we want to learn the distribution of the labels y conditioned on the value of a sample x, that is we want to learn the conditional probability $P(y|x)$ for $y \in -1, 1$ and all $x \in \mathcal{X}$. Also, assume that the distribution $P(y|x)$ can be parametrized by choosing w in some parameter space $\mathcal{W}$. That is, by choosing $w \in \mathcal{W}$ we get the value of $P_w(y|x)$ for $y \in -1, 1$ and all $x \in \mathcal{X}$. In this context, the learning problem becomes to come up with a method for choosing some parameter $\hat{w} \in \mathcal{W}$ and hereby a corresponding distribution $P_{\hat{w}}(y|x)$, which somehow is our best guess of the true distribution of y conditioned on x. The information we have available to base this choice on is some finite, labeled sample $S = \{(x_1, y_1), ..., (x_N, y_N)\}$, where each y_i is assumed to have been sampled from $P_w(y|x_i)$ and all of them independently from each other.

The maximum likelihood method for choosing $\hat{w}$ solves this problem by defining the likelihood function L_S for the given sample S as

$$L_S(w) = \prod_{(x_n, y_n) \in \mathcal{S}} P_w(y_n|x_n) = \prod_{n=1}^{N} P_w(y_n|x_n) \tag{19}$$

and then saying that we should choose $\hat{w} \in \mathcal{W}$ such that $L_{\mathcal{S}}$ is maximized.

Since the function $-\ln$ is monotonically decreasing, this strategy is equivalent[1] to choosing $\hat{w} \in \mathcal{W}$ such that the function

$$f_S(w) = -\ln\left(\prod_{n=1}^{N} P_w(y_n|x_n)\right) = \sum_{n=1}^{N}(-\ln P_w(y_n|x_n)) \tag{20}$$

is minimized.

Since $y \in \{-1, 1\}$, then we can write $P_w(y|x)$ as

$$P_w(y|x) = \tag{21}$$
$$[[y=1]]P_w(y=1|x) + [[y=1]]P_w(y=-1|x) = \tag{22}$$
$$[[y=1]]h_w(x) + [[y=1]](1-h_w(x)) \tag{23}$$

where we simply have defined $h_w(x) = P(y=1|x)$. We therefore have that

$$-\ln P_w(y_n|x_n) = \tag{24}$$
$$-\ln([[y=1]]h_w(x) + [[y=-1]](1-h_w(x))) = \tag{25}$$
$$[[y=1]](-\ln(h_w(x)) + [[y=-1]](-\ln(1-h_w(x))) = \tag{26}$$
$$[[y=1]]\left(\ln\left(\frac{1}{h_w(x)}\right)\right) + [[y=-1]]\left(\ln\left(\frac{1}{1-h_w(x)}\right)\right) \tag{27}$$

By line (2) and line (6-9), we now get that

$$f_S(w) = \sum_{n=1}^{N}\left[[[y_n=1]]\left(\ln\left(\frac{1}{h_w(x_n)}\right)\right) + [[y_n=-1]]\left(\ln\left(\frac{1}{1-h_w(x_n)}\right)\right)\right] \tag{28}$$

As I have already said, we will end up with the same $\hat{w}$ for a given sample S, if we minimize $f_S(w)$ as if we maximize $L_S(w)$. If we had started by saying that we would like to estimate the probability $h_w(x) = P_w(y=1|x)$ by choosing $\hat{w}$ such that we minimize the error function $f_S(w)$ defined as in line (10), we would have ended up with the same estimates of $P(y|x)$ for $y \in \{-1, 1\}$ and $x \in \mathcal{X}$, as if we have used the maximum likelihood method. These two strategies are therefore equivalent.

If we now focus on logistic regression and set $h_w(x) = \Theta(w^T x) = \frac{e^{w^T x}}{1+e^{w^T x}}$, we know from the book that $1 - h_w(x) = h_w(-x)$, which means that

$$f_S(w) = \sum_{n=1}^{N}\left[[[y_n=1]]\left(\ln\left(\frac{1}{h_w(x_n)}\right)\right) + [[y_n=-1]]\left(\ln\left(\frac{1}{h_w(-x_n)}\right)\right)\right] = \tag{29}$$
$$\sum_{n=1}^{N}\ln\left(\frac{1}{h_w(y_n x_n)}\right) = \sum_{n=1}^{N}\ln\left(\frac{1}{\frac{e^{y_n w^T x}}{1+e^{y_n w^T x}}}\right) = \sum_{n=1}^{N}\ln\left(\frac{1+e^{y_n w^T x}}{e^{y_n w^T x}}\right) = \tag{30}$$
$$\sum_{n=1}^{N}\ln\left(1+e^{-y_n w^T x}\right) \tag{31}$$

[1] I define two strategies to be equivalent, if and only if they end up choosing the same $\hat{w}$ for all possible samples $S = \{(x_1, y_1), ..., (x_N, y_N)\}$.

Minimizing this function gives the same w as minimizing:

$$\frac{1}{N} f_S(w) = \frac{1}{N} \sum_{n=1}^{N} \ln\left(1 + e^{-y_n w^T x}\right) \tag{32}$$

This is exactly the function that the book says in statement (3.9) that we should minimize, when we search for the weights in logistic regression.

4.2 Logistic regression loss gradient

In the algorithm for logistic regression, we determine the parameter $w \in \mathbb{R}^m$ in our final hypothesis

$$h_w(x) = P_w(y = 1|x) = \frac{e^{w^T x}}{1 + e^{w^T x}} \tag{33}$$

using a given labeled sample $S = \{(x_1, y_1), ..., (x_N, y_N)\}$ by minizing the function

$$E_{in}(w) = \frac{1}{N} \sum_{n=1}^{N} \ln(1 + e^{-y_n w^T x_n}) \tag{34}$$

We use gradient descent to do this, which means we have to find the loss gradient $\nabla E_{in}(w)$. We can use matrix calculus to to differentiate E_{in}

$$D_w(E_{in}(w)) = D_w\left(\frac{1}{N} \sum_{n=1}^{N} \ln(1 + e^{-y_n w^T x_n})\right) = \tag{35}$$

$$\frac{1}{N} \sum_{n=1}^{N} D_w\left(\ln(1 + e^{-y_n w^T x_n})\right). \tag{36}$$

Let us now focus on the inside of the sum

$$D_w\left(\ln(1 + e^{-y_n w^T x_n})\right) = \frac{D_w(1 + e^{-y_n w^T x_n})}{1 + e^{-y_n w^T x_n}} = \tag{37}$$

$$\frac{D_w(e^{-y_n w^T x_n})}{1 + e^{-y_n w^T x_n}} = \frac{e^{-y_n w^T x_n} D_w(-y_n w^T x)}{1 + e^{-y_n w^T x_n}} = \tag{38}$$

$$-y_n x_n \frac{e^{-y_n w^T x_n}}{1 + e^{-y_n w^T x_n}} = -y_n x_n \Theta(-y_n w^T x_n) \tag{39}$$

Line (13-14) and line (15-17) now gives us that

$$D_w(E_{in}(w)) = \frac{1}{N} \sum_{n=1}^{N} D_w\left(\ln(1 + e^{-y_n w^T x_n})\right) = \frac{1}{N} \sum_{n=1}^{N} -y_n x_n \Theta(-y_n w^T x_n) \tag{40}$$

which is what the exercise asked us to show.

4.3 Logistic regression implementation

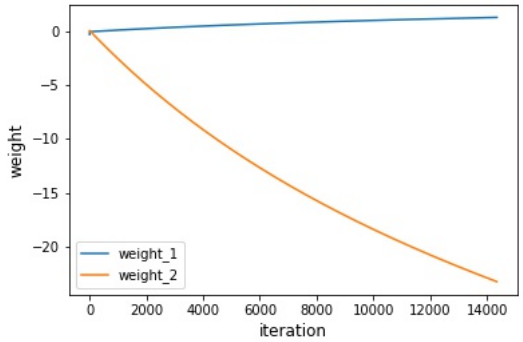
My implementation is written in python with the use of the scipy library. It has a little more code than strictly necessary for two reasons. First, I have experimented with the book's proposal of using a combination of the norm of the gradient and the training error made by the classifier resulting from the weigths in the given iteration as a stopping criterion for the algorithm. This means that I have to construct the classifier from the weights and calculate the resulting error on the training set in each iteration of the algorithm. Second, I log information from each iteration in order to be able to investigate how the algorithm evolves through time.

As input, my main function *log_reg* takes a scipy 2d-array as x (the sample) and a scipy 1d-array as y (the labels). Additionally, the user can specify some more arguments to control the hyper-parameters of the algorithm. As output, the function returns a dictionary with the weights found by the algorithm, the classifier given by these weigths and some arrays that log the history of different, interesting variables through the iterations of the algorithm. Here is my source code:

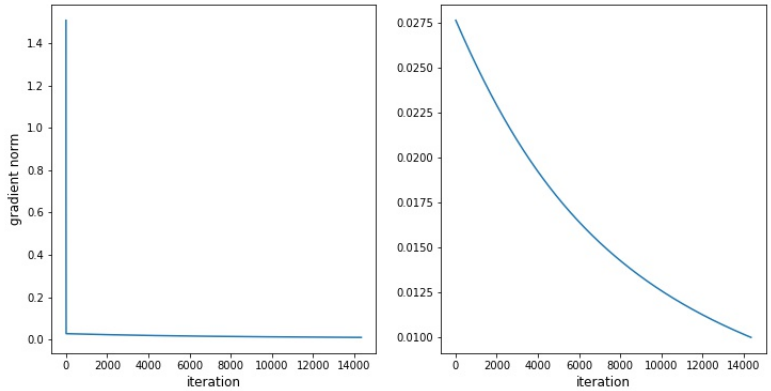
```
import scipy as scp
import scipy.linalg as lalg
-------------------------------------------------------------------------------
def log_reg(x, y, rate = 1, gnorm_stop = 0.01, error_stop = 0.05,
            iterations_stop = 1000000, start_weigths = None, seed = 1989) :

    scp.random.seed(seed)
    if (start_weigths == None) : w = scp.random.normal(size = x.shape[1])
    else : w = start_weights

    n = x.shape[0]; ws = []; gnorms = []; es = []; i = 1

    while (i <= iterations_stop) :

        ws.append(w)
        g = gradient(x, y, w, n); gnorm = lalg.norm(g); gnorms.append(gnorm)
        c = make_classifier(w); e = error(x,y,c,n); es.append(e)

        if terminate(e, error_stop, gnorm, gnorm_stop) : break

        w = w + (-g) * rate; i = i + 1

    return {'classifier' : c, 'weights' : w, 'gnorm_history' : scp.array(gnorms),
            'weight_history' : scp.array(ws), 'error_history' : scp.array(es)}
-------------------------------------------------------------------------------
def gradient(x, y, w, n) :
    num = x * y[:,scp.newaxis]
    den = scp.dot(x, w) * y
    den = scp.exp(den) + 1
    quotient = num / den[:,scp.newaxis]
    avg = scp.sum(quotient, axis = 0) / n
    return - avg
-------------------------------------------------------------------------------
def make_classifier(w) :
    def classifier(x) :
        p = scp.exp(scp.dot(x,w))/(1 + scp.exp(scp.dot(x,w)))
        if (p > 0.5) : return 1
        else : return -1
    return classifier
-------------------------------------------------------------------------------
def error(x, y, classifier, n) :
    pred = scp.apply_along_axis(classifier, 1, x)
    e = abs((pred - y)) / 2
    e = scp.sum(e) / n
    return e
-------------------------------------------------------------------------------
def terminate(e, error_stop, gnorm, gnorm_stop) :
    if ((gnorm < gnorm_stop) and (e < error_stop )) : return True
    else : return False
-------------------------------------------------------------------------------
```

4.4 Iris Flower Data

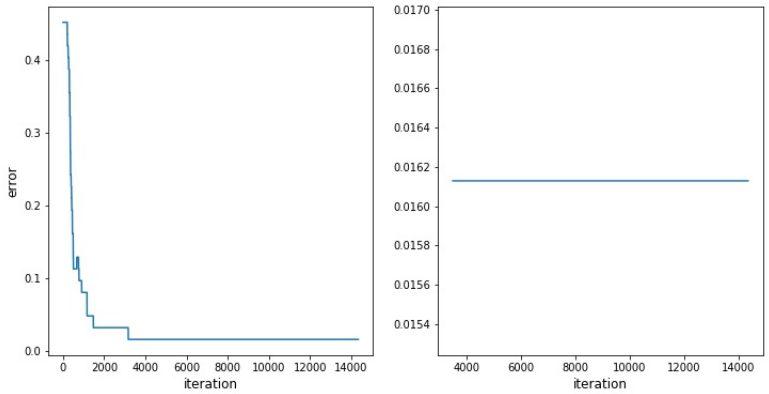
As I mentioned in the last section, I have experimented with using both the norm of the gradient and size of the training error as the stopping criterion of my algorithm. Specifically, I have said that the algorithm should stop iterating, when the norm of the gradient is less than 0.01, and the training error - calculated as the fraction of wrongly classified points - is less than 0.05. If this criterion has not been fulfilled after the 1000000^{th} iteration, the algorithm should also stop. These hyper parameters can be changed by specifying arguments to my function. If no other start weights are specified, then my implementation samples to start weights from a standard normal distribution. Here is a plot showing how the values of the weights develop through the iterations of the algorithm:



When we look at this plot, it does not look like the value of the weights have converged, when the algorithm stopped. Here are two plots of the norm of gradient through the iterations, where I have cut away the 10 first iterations in the second plot:



Here, we also see from the second plot that the norm of the gradient is still in the process of getting even smaller, when the algorithm stops. When we look at the plot of the development of the training error through the iterations, we see something different:



After around iteration number 3500, the training error does not improve, but is constant with the value of 0.0161. This corresponds to 1 of 62 training examples being wrongly classified. In terms of only getting a low training error, the majority of the iterations are therefore wasted. On the hand, if we wanted to get our weights as close as possible to the weights maximizing the likelihood of the data in the model space of logistic regression, then we should have continued the iterations, since the weights have not converged yet.

All in all, my implementation of logistic regression with the chosen hyper-parameters end up giving me the weights $(1.31, -23.28)$. The classifier constructed from these weights gives me a test error of 0.038. This corresponds to 1 of the 26 test examples being wrongly classified.