

Homework 1 - Learning By Example - Spring 2025

Essay Question 1: *There are numerous factors that led to the explosive interest in Machine Learning since 2012.* Write an essay that starts with this sentence and goes on to identify and describe the main forces giving rise to this renewed interest. Be sure to discuss how/why the forces you list led to this interest. Conclude your essay with a single sentence summarizing the main assertion in your essay.

Problem 1: Consider a logistic regression problem whose targets are in the set $Y = \{0, 1\}$. Show that minimizing the negative log likelihood of the dataset is equivalent to minimizing the dataset's empirical risk based on the binary cross-entropy loss function.

Solution: For binary targets $Y = \{0, 1\}$, we let

$$\begin{aligned} Q_{\mathbf{y}|x}(y_k = 0|x_k) &\approx \sigma(s_w(x_k)) \\ Q_{\mathbf{y}|x}(y_k = 1|x_k) &\approx 1 - \sigma(s_w(x_k)) \end{aligned}$$

So the likelihood of the dataset is

$$\mathcal{L}(\mathcal{D} | w) = \prod_{k=1}^N Q_{\mathbf{y}|x}(y_k|x_k) = \prod_{k=1}^N (y_k h_w(x_k) + (1 - y_k)(1 - h_w(x_k)))$$

If we take the negative log-likelihood function this becomes

$$-\log \mathcal{L}(\mathcal{D} | w) = -\sum_{k=1}^N \log (y_k h_w(x_k) + (1 - y_k)(1 - h_w(x_k)))$$

where y_k takes values of 0 or 1. Because of the monotone nature of the log function, we see that maximizing the likelihood is equivalent to minimize the negative log likelihood. But if we divide the negative log-likelihood by N , this is simply the empirical risk,

$$-\frac{1}{N} \log \mathcal{L}(\mathcal{D} | w) = \hat{R}_{\mathcal{D}}[h_w] = -\frac{1}{N} \sum_{k=1}^N (y_k \log h_w(x_k) + (1 - y_k) \log(1 - h_w(x_k)))$$

with the binary cross-entropy loss function

$$L(y, h_w(x)) = -y \log h_w(x) - (1 - y) \log(1 - h_w(x))$$

Problem 2: Consider a learning-by-example problem whose inputs $x \in [0, 2]$ and whose targets $y \in \{-1, +1\}$. Assume the "system" draws samples (x, y) with an equal probability of the sample being in class -1 or $+1$. Assume that the conditional probability of the inputs, x are

$$P(x | y = -1) = \begin{cases} 1 & \text{if } 0 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}, \quad P(x, | y = +1) = \begin{cases} 1/2 & \text{if } 0 \leq x \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

Assume the model has the form $h_w(x) = \text{sgn}(x - w)$ where $w \in \mathbb{R}$ is the weight. Assume the loss function is $L(y, h_w(x)) = \mathbb{1}(y \neq h_w(x))$.

1. Determine the true risk, $R[h_w]$, as a function of w .
2. Determine the optimal model weight, w , that minimizes the model's true risk, $R[h_w]$.

Solution: From the problem statement we know that $\Pr(y = +1) = \Pr(y = -1) = 1/2$ and the conditional densities are

$$p(x | y = -1) = \begin{cases} 1 & 0 \leq x < 1 \\ 0 & \text{otherwise} \end{cases}, \quad p(x | y = +1) = \begin{cases} \frac{1}{2} & 0 \leq x < 2 \\ 0 & \text{otherwise} \end{cases}$$

Let $\hat{y} = h_w(x)$ denote the prediction made by model $h_w(x) = \text{sgn}(x - w)$. This means $\hat{y}(x) = -1$ if $x < w$ and $+1$ if $x \geq w$. The true risk is

$$R[h_w] = \mathbb{E}[\mathbb{1}(y \neq h_w(x))] = \Pr(x > w, y = -1) + \Pr(x < w, y = +1)$$

There are four cases to consider.

1. If $w \leq 0$, then $\Pr(x < w, y = +1) = 0$ since x has support over $[0, 2]$ and $w < 0$. This means $\Pr(x \geq w, y = -1) = \frac{1}{2}$ since $x \geq w$ covers the entire interval $[0, 2]$. This implies $R[h_w] = \frac{1}{2}$ for $w \leq 0$

2. if $0 < w < 1$ then we have

$$\begin{aligned} \Pr(x < w, y = +1) &= \frac{1}{2} \int_0^w p(x | y = +1) dx = \int_0^w \frac{1}{4} dx = \frac{w}{4} \\ \Pr(x > w, y = -1) &= \frac{1}{2} \int_w^2 p(x | y = -1) dx = \frac{1}{2} \int_w^1 dx = \frac{1-w}{2} \end{aligned}$$

So the true risk is

$$R[h_w] = \frac{1}{2} - \frac{w}{4} \text{ for } 0 \leq w < 1$$

3. For $1 \leq w < 2$ we have

$$\Pr(x < w, y = +1) = \frac{1}{2} \int_0^w p(x | y = +1) dx = \frac{1}{2} \int_0^w \frac{1}{2} dx = \frac{w}{4}$$

and $\Pr(x \geq w, y = -1) = 0$ since $p(x | y = -1) = 0$ when $x \notin [0, 1]$. So the true risk is

$$R[h_w] = \frac{w}{4} \text{ when } 1 \leq w < 2.$$

4. For $w > 2$, we have $\Pr(x < w, y = +1) = \frac{1}{2}$ and $\Pr(x > w, y = -1) = 0$ so the true risk is $\frac{1}{2}$.

The true risk, therefore is

$$R[h_w] = \begin{cases} \frac{1}{2} & w \leq 0 \\ \frac{1}{2} - \frac{w}{4} & 0 < w < 1 \\ \frac{w}{4} & 1 \leq w < 2 \\ \frac{1}{2} & w \geq 2 \end{cases}$$

Notebook Assignment 1: We will consider a logistic regression problem that trains a model using a dataset consisting of N input vectors, $x_k \in \mathbb{R}^n$ and N target labels $y_k \in \{0, 1\}$ for $k = 1, \dots, N$. The dataset $\mathcal{D} = (\mathbf{X}, \mathbf{Y})$ consists of two data matrices

$$\mathbf{X} = \begin{bmatrix} x_1 & x_2 & \cdots & x_N \end{bmatrix}, \quad \mathbf{Y} = \begin{bmatrix} y_1 & y_2 & \cdots & y_N \end{bmatrix}^T$$

The model is $h_w : \mathbb{R}^n \rightarrow [0, 1]$ where

$$h_w(x) = \sigma(w^T x) \tag{1}$$

where $w \in \mathbb{R}^n$ is a weight vector and $\sigma(s) = \frac{1}{1 + e^{-s}}$ is a softmax function. The loss function will be the binary cross-entropy function you examined in problem 1.

Such models are often trained using *gradient descent* algorithms. These algorithms update the weights of the model using the equation

$$w_{k+1} \leftarrow w_k - \eta \frac{\partial \hat{R}_{\mathcal{D}}(w)}{\partial w} \tag{2}$$

where $\eta > 0$ is called the *learning rate* and $\hat{R}_{\mathcal{D}}(w)$ is the empirical risk function of the model h_w on the given dataset. The algorithm starts with an initial weight, w_0 and then updates w_k using equation (2) until a termination condition is satisfied. The termination condition is satisfied if either 1) the number of recursions (`iter`) exceed a specific limit (`maxiter`) or 2) that the Euclidean norm of the difference between two successive weight updates

$$|w_{k+1} - w_k| \leq \text{tol}$$

where `tol` is a specified *tolerance* level.

The training data you will use for this notebook exercise is a numpy array

`ring_data.npy`

which you can find a link to in the course's vault. This numpy array has a shape $(3, 2000)$. It may be seen as a matrix whose k th column is the k th sample in the dataset ($k = 1, 2, \dots, 2000$). The first two elements of the k th column is the input $x_k \in \mathbb{R}^2$ and the third element of that column is the target label y_k . If you want to use this matrix in Google Colab, you'll first need to upload it from your local directory into Google Colab session storage.

1. Write two Python functions

```
def Rhat(w, X, Y):
def gradRhat(w, X, Y):
```

whose inputs are the weight vector, w , input data matrix, \mathbf{X} and target matrix \mathbf{Y} . The function `Rhat` returns the empirical loss at w for the dataset $\mathcal{D} = (\mathbf{X}, \mathbf{Y})$. The function `gradRhat` returns the gradient $\frac{\partial \hat{R}_{\mathcal{D}}(w)}{\partial w}$ evaluate at w for dataset $\mathcal{D} = (\mathbf{X}, \mathbf{Y})$. To test your function, compute the empirical risk and gradient of that risk on the given dataset when the components of the model parameter vector, $w = [-1, 1]$.

2. Write a Python function

```
def fit(X, Y, lr, maxiter, tol):
```

whose input arguments are the numpy array of input samples, \mathbf{X} , numpy array of binary targets, \mathbf{Y} , the learning rate (`lr`) η , and the hyperparameters `maxiter` and `tol`. Have this function return the last weight vector, w , obtained after the gradient descent algorithm terminates and a numpy array, `history`, whose entries contain the value of the loss function and the weight vector computed after each recursion of the gradient descent algorithm. Test your function with a learning rate $\eta = 0.1$ for a maximum of 100 iterations (`maxiter`) and a tolerance (`tol`) of 10^{-4} . Then plot the empirical loss as a function of the iteration. Scatter plot for the dataset's input samples, x_k , coloring the class 1 samples blue and the class 0 samples green. On the scatter plot also plot the initial discriminant surface and the final discriminant surfaces obtained after the gradient descent algorithm terminates.

Solution (part 1): We first need to derive an expression for the empirical risk. Because the targets are in $Y = \{0, 1\}$, we have

$$\hat{R}_{\mathcal{D}}(w) = -\frac{1}{N} \sum_{k=1}^N [y_k \log(\sigma(w^T x_k)) + (1 - y_k) \log(1 - \sigma(w^T x_k))]$$

where $\sigma(s) = \frac{1}{1+e^{-s}}$. This means that

$$\begin{aligned} \sigma'(s) = \frac{d\sigma(s)}{ds} &= -\frac{e^s}{(1+e^s)^2} = \frac{1+e^{-s}-1}{(1+e^{-s})^2} \\ &= \frac{1}{(1+e^{-s})} - \frac{1}{(1+e^{-s})^2} = \frac{1}{1+e^{-s}} \left(1 - \frac{1}{1+e^{-s}}\right) = \boxed{\sigma(s)(1-\sigma(s))} \end{aligned}$$

We now take the gradient

$$\begin{aligned} \frac{\partial \hat{R}_{\mathcal{D}}(s)}{\partial w} &= \frac{1}{N} \sum_{k=1}^N \left[-y_k \frac{1}{\sigma(w^T x_k)} \sigma'(w^T x_k) x_k + (1 - y_k) \frac{1}{1 - \sigma(w^T x_k)} \sigma'(w^T x_k) x_k \right] \\ &= \frac{1}{N} \sum_{k=1}^N [-y_k(1 - \sigma(w^T x_k)) x_k + (1 - y_k) \sigma(w^T x_k) x_k] \\ &= \frac{1}{N} \sum_{k=1}^N [-y_k x_k + y_k \sigma(w^T x_k) x_k + \sigma(w^T x_k) x_k - y_k \sigma(w^T x_k) x_k] \\ &= \boxed{\frac{1}{N} \sum_{k=1}^N (\sigma(w^T x_k) - y_k) x_k} \end{aligned}$$

These equations are then used to write expressions for the empirical risk and its gradient.

I ran this in Google Colab. I first uploaded `ring_data.npy` to Google Colab Session Storage. The following script loads the data set and then prints its shape.

```

data = np.load("ring_data.npy")
print(f"data shape = {data.shape}")

_, Ntrain = data.shape

X = data[0:2, 0:Ntrain].reshape(2, Ntrain)
Y = data[2, 0:Ntrain].astype('int').reshape(Ntrain)
indx = np.where(Y == -1)
Y[indx] = 0

indx1 = np.where(Y == 1)
plt.scatter(X[0, indx1], X[1, indx1], c='b')
indx0 = np.where(Y == 0)
plt.scatter(X[0, indx0], X[1, indx0], c='g')
plt.legend(('class 1', 'class 0'))
plt.title('Ring Data')

```

Note that the original target data takes values in $Y = \{-1, +1\}$, but that the empirical risk base don the sigmoid assumes targets $Y = \{0, +1\}$, so I changed the values in Y to reflect this.

I then wrote the two function using the formulae I derived above. My versions of these functions are given below. The last line computes \hat{R} for the dataset using the specified $w = [-1, 1]$. This shows $\hat{R} = .936$ and $\frac{\partial \hat{R}}{\partial w} = [-1.508, -0.605]$.

```

def sigma(s):
    return 1/(1+np.exp(-s))

def gradRhat(w, X, Y):
    nw, N = X.shape
    dRhat_dw = np.zeros(nw)
    for k in range(N):
        xk = X[:, k]
        yk = Y[k]
        dRhat_dw += ((sigma(np.sum(w*xk)) - yk) / N) * xk
    return dRhat_dw

def Rhat(w, X, Y):
    eps = 1.e-10
    nw, N = X.shape
    Rhat = np.zeros(1)
    for k in range(N):
        xk = X[:, k]
        yk = Y[k]
        if (yk >= (1-eps)):
            Rhat += -yk*np.log(sigma(np.sum(w*xk))) / N
        if (yk <= eps):
            Rhat += -(1-yk)*np.log(1-sigma(np.sum(w*xk))) / N
    return Rhat

```

```

w = [-1,1]
print(f'Rhat = {Rhat}')
print(f'gradRhat = {gradRhat}')

# Rhat = [0.93645349]
# gradRhat = [-1.50774038 -0.60466714]

```

Solution (part 2): My version of the training function is

```

def fit(X,Y,lr,maxiter = 500, tol = 1e-6):
    nw,N = X.shape
    w = [-1,1]
    history = np.concatenate([Rhat(w,X,Y),w]).reshape(1,3)
    for _ in range(maxiter):
        diff = -lr * gradRhat(w,X,Y)
        if np.all(np.abs(diff)<=tol):
            break
        w += diff
        entry = np.concatenate([Rhat(w,X,Y),w]).reshape(1,3)
        history = np.vstack((history,entry))
    return w,history

wfinal, xhist = fit(X,Y,lr=0.1,maxiter=100,tol = 1.e-4)

```

The code to generate the plots

```

nk,_ = xhist.shape
plt.figure(1)
plt.plot(range(nk),xhist[:,0])
plt.xlabel('iteration count (iter)')
plt.ylabel('empirical risk Rhat(w|D)')
plt.title('empirical risk of model wk')

wstart = xhist[0,1:3]
x0start = np.arange(-20,30,.5)
x1start = -x0start*wstart[0]/wstart[1]
x0final = np.arange(-20,30,.5)
x1final = -x0final*wfinal[0]/wfinal[1]

import matplotlib.pyplot as plt
indx1 = np.where(Y==1)
plt.figure(2)
plt.scatter(X[0,indx1],X[1,indx1],c='b')
indx0 = np.where(Y==0)
plt.scatter(X[0,indx0],X[1,indx0],c='g')
plt.title('Ring Data with discriminants')

```

```
plt.plot(x0start,x1start,'b--')
plt.plot(x0final,x1final,'b-')
plt.legend(('class +1','class 0','w start','w final'))
```

The output from this cell is shown below in Fig. 1.

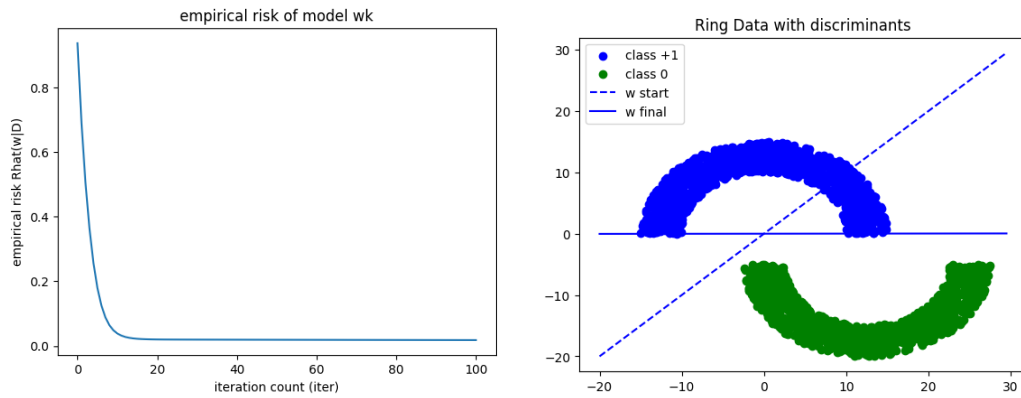


Figure 1: Notebook Result