ML4ENG Coursework – Part 2

Deadline for submission (enforced by Keats): December 14, 2022 at 16:00.

1 General Guidelines

In this coursework, you will work with

1. Binary classification using deterministic and soft predictors.
2. Binary classification using multiple layers.

To start, download the data sets `dataset_heart_attack.mat` and the template file `cw2_template.m` from the module's Keats website. Once this is done:

3. Change the `cw2_template.m` to your k number. In the following, we will refer to this file as `k12345678.m`.
4. Open the `k12345678.m` file with your MATLAB editor [1]. Note that the file contains a preamble, referred to as main body, which you should **not** modify, and the definition of several functions.
5. Follow the Instructions (Section 3 of this document) to fill in the details of the functions in the template file. The functions in the k12345678.m file have been numbered according to the numbered list below in Section 3 (Instructions).
6. Make sure each function's output has the correct size.
7. You are encouraged to use in each function previous functions that you wrote. It will not always be possible.
8. Once you have written the functions, verify that the file `k12345678.m` runs without errors when the file is included in a folder containing **only** the file itself and the data sets.
9. Check that **no MATLAB toolbox** was used. You can type in the command window `license('inuse')` and verify only standard `matlab` functions are used.
10. Check no display lines are printed when running `k12345678.m` beside the discussion parts that are detailed in Section 3. Any variable that will be printed have a penalty of 2 points. As a reminder, place ';' at the end of each assignment line to avoid printing.
11. If running file `k12345678.m` raises any error, the coursework may be graded 0.
12. Submit only the `k12345678.m` file on Keats. No other files are allowed.

2 DATA SET

The file `dataset_heart_attack.mat` [2] contains a data set $\mathcal{D} = \{x_n, t_n\}_{n=1}^{N}$, which consists of $N = 303$ examples. Each example consists of:

1. Input vector x_n in $\mathbb{R}^{13}$, encompassing $d = 13$ medical features.
2. Its corresponding binary label $t_n \in \{0,1\}$, where 1 stands for high chance of heart attack and 0 for low chance as diagnosed by a medical expert.

The data is loaded into the workspace to have

Name	**Size**	**Type**	**Description**
`t`	$N \times 1$	Logical	Diagnosis (binary label): 1 = high chance of heart attack and 0 = low chance.
`X`	$N \times d$	Double	Data matrix (samples vectors as rows)
`x_titles`	$1 \times d$	String	Description for the *d* features in x

The n-th input sample vector is denoted as

$$x_n = \begin{bmatrix} x_n^{(1)} & \dots & x_n^{(d)} \end{bmatrix}^T,$$

and the inputs of the data sets are given by stacking up N samples

$$X = \begin{bmatrix} x_1^T \\ \vdots \\ x_N^T \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & \cdots & x_1^{(d)} \\ \vdots & \ddots & \vdots \\ x_N^{(1)} & \cdots & x_N^{(d)} \end{bmatrix} \in \mathbb{R}^{N \times d}.$$

Note the superscript denotes entry index and not power (as indicated by the use of the parenthesis). The labels are also stacked up, forming the vector

$$t = \begin{bmatrix} t_1 & t_2 & \dots & t_N \end{bmatrix}^T,$$

The entries of the vector `x_titles` annotate the d features.

We will focus on two feature vectors. The first vector of features, named A, consists of all inputs, together with a bias, and hence it includes $D_A = d + 1 = 14$ scalar features. The second set of features, named B, includes all inputs, a bias, and their squares, for a total of $D_B = 2d + 1 = 27$ scalr features.

Accordingly, the two feature vectors for a given input x_n are given as

$$u_A(x_n) = \begin{bmatrix} 1, & x_n^{(1)}, & x_n^{(2)}, & \dots, & x_n^{(d-1)}, & x_n^{(d)} \end{bmatrix}^T \in \mathbb{R}^{14}$$

$$u_B(x_n) = \begin{bmatrix} 1, & x_n^{(1)}, & \dots, & x_n^{(d)}, & \left(x_n^{(1)}\right)^2, & \left(x_n^{(2)}\right)^2, \dots, & \left(x_n^{(d-1)}\right)^2, & \left(x_n^{(d)}\right)^2 \end{bmatrix}^T \in \mathbb{R}^{27}$$

For a set of N samples, we stack the features as usual into the feature matrices

$$U_A = \begin{bmatrix} u_A(x_1)^T \\ \vdots \\ u_A(x_N)^T \end{bmatrix} \in \mathbb{R}^{N \times D_A} \quad , \quad U_B = \begin{bmatrix} u_B(x_1)^T \\ \vdots \\ u_B(x_N)^T \end{bmatrix} \in \mathbb{R}^{N \times D_B}.$$

You are provided with two functions to produce these, using `U = mapping_A(X)` and `U = mapping_B(X)` respectively. In them, you can see that we also add bias and scale each input, so all of them will reside in the segment [-1,+1].

We use the dimension D to represent either D_A or D_B when dealing with general functions.

3 Instructions for Completing the Coursework

Section 1 – Perceptron

To complete this section, we will work on 6 functions that will enable the training of a discriminative linear model for binary classification via the perceptron algorithm. Two model classes will be investigated, one with feature mapping $u_A(x)$ and the other with feature vector $u_B(x)$. Accordingly, the model parameter vector is of size D_A or D_B, and all functions below are applicable to an arbitrary feature size D. The main body loads the data set, and sends the entire data set `X` to all of the next functions.

1. [5 points] Design the function

   ```
   function t_hat= perceptron_predict(X, map_func, theta)
   ```

   that takes as input a data matrix `X`$\in \mathbb{R}^{N \times d}$, a feature mapping function `map_func` (this will be called with `mapping_A` or `mapping_B`), as well as a model parameter vector `theta`$\in \mathbb{R}^{D}$. The function returns the perceptron output `t_hat`$\in \mathbb{R}^{N}$, as a column vector of hard predictions $\hat{t}(x|\theta) = [\hat{t}(x_1|\theta), \hat{t}(x_2|\theta), \dots, \hat{t}(x_N|\theta)]^T$ applied to the N inputs using the selected feature vector. To avoid ambiguity, assume `step(0.0)=0` here.

2. [5 points] Design the function

   ```
   function cm = classification_margin(X, t, map_func,
   theta)
   ```

   that takes as input a data matrix `X`$\in \mathbb{R}^{N \times d}$, the targets vector `t`$\in \{0,1\}^{N}$, a feature mapping function `map_func`, and a model parameter vector `theta`$\in \mathbb{R}^{D}$. The function return a vector of the classification margins `cm`$\in \mathbb{R}^{N}$, as the column vector $\left[t_1^{\pm}(\theta^T u(x_1)), t_2^{\pm}(\theta^T u(x_2)), \dots, t_N^{\pm}(\theta^T u(x_N))\right]^T$.

3. [5 points] Design the function

   ```
   function grad_theta = perceptron_gradient(X, t, map_func,
   theta)
   ```

that takes as input a data matrix `X` $\in \mathbb{R}^{N\times d}$, the targets vector `t` $\in \{0,1\}^N$, a feature mapping function `map_func` that produces D-lengthed features, and a model parameter vector `theta` $\in \mathbb{R}^D$. The function returns a matrix of size $N \times D$, with the n-th row representing the transpose of the gradient of the n-th point $\nabla_\theta l\big(t_n, \hat{t}(x_n|\theta)\big)$ of the surrogate loss of the perceptron algorithm, that is the gradient of the hinge-at-zero loss at the above sample pair. The gradient matrix is represented as

$$\begin{bmatrix} \Big(\nabla_\theta l\big(t_1, \hat{t}(x_1|\theta)\big)\Big)^T \\ \cdots \\ \Big(\nabla_\theta l\big(t_N, \hat{t}(x_N|\theta)\big)\Big)^T \end{bmatrix}.$$

4. [5 points] Design the function

   `function loss = hinge_at_zero_loss(X, t, map_func, theta)`

   that takes as input a data matrix `X` $\in \mathbb{R}^{N\times d}$, the targets vector `t` $\in \{0,1\}^N$, a feature mapping function `map_func` that produces D-lengthed features, and a model parameter vector `theta` $\in \mathbb{R}^D$. The function returns the empirical hinge-at-zero loss $L_{\mathcal{D}} \in \mathbb{R}$ in the variable `loss` of the perceptron predictions using the given features and model parameter vector.

5. [10 points] Design the function

   `function theta_mat = perceptron_train_sgd(X, t, map_func, theta_init, I, gamma)`

   that takes as input a training data matrix `X` $\in \mathbb{R}^{N\times d}$, its targets vector `t` $\in \{0,1\}^N$, a feature mapping function `map_func` that produces D-lengthed features, an initial parameter `theta_init` $\in \mathbb{R}^D$; the number `I` $\in \mathbb{N}$ of training iterations; and learning rate `gamma` > 0. The function runs over `I` training iterations of the perceptron algorithm with mini-batch size of 1. It returns a matrix, with model parameter vectors along these iterations as columns in the matrix `theta_mat` $\in \mathbb{R}^{D\times(I+1)}$. The first column equals `theta_init`, the second column is after one update, and so on. The mini-batch for the i-th iteration has a single sample n_i that is chosen via the cyclic scan

   $$n_i = 1 + (i-1)\ mod\ N.$$

   This means that the first update uses the first input x_1 (as the first row of `X`) and first entry of `t`, the second update uses the second sample and its label, and so forth.

The main body trains the two different model classes perceptrons, and plots the loss for each model.

6. [10 points] Add up to two lines of text in `function discussion()` addressing the question: According to the produced graphs, can we tell how fitted the trained models (whether overfitted, underfitting or well fitted)? State which, or write what we should have done differently in order to be able to tell.

Section 2 – Logistic Regression

To complete this section, 4 functions will be coded to implement the training of discriminative probabilistic linear model for binary classification by using logistic regression. The same two model classes with feature mapping $u_A(x)$ and $u_B(x)$ are considered, and so is the data set. All functions must be written to an arbitrary model parameter number D.

7. [5 points] Design the function
   `function logit= logistic_regression_logit(X, map_func, theta)`
   that takes as input a data matrix `X`$\in \mathbb{R}^{N\times d}$, a feature mapping function `map_func` that produces D-lengthed features, and a model parameter vector `theta`$\in \mathbb{R}^{D}$. The function returns the logit column vector `logit`$\in \mathbb{R}^{N}$ of a logistic regression output, outputting the vector $[\theta^T u(x_1), \theta^T u(x_2), \dots, \theta^T u(x_N)]^T$.
8. [10 points] Design the function
   `function grad_theta = logistic_regression_gradient(X, map_func, t, theta)`
   that takes as input a data matrix `X`$\in \mathbb{R}^{N\times d}$, the targets vector `t`$\in \{0,1\}^{N}$, a feature mapping function `map_func` that produces D-lengthed features, and a model parameter vector `theta`$\in \mathbb{R}^{D}$. The function returns a matrix of size $N \times D$, with the n-th row representing the transpose of the gradient of the n-th point $\nabla_\theta(-\log p(t = t_n|x_n, \theta))$ of the logistic loss. The gradient matrix is represented as
   $$\begin{bmatrix} (\nabla_\theta(-\log p(t = t_1|x_1, \theta)))^T \\ \dots \\ (\nabla_\theta(-\log p(t = t_N|x_N, \theta)))^T \end{bmatrix}.$$
   It may be useful to use the `sigmoid()` function provided in the auxiliary functions.
9. [10 points] Design the function
   `function loss = logistic_loss(X, t, map_func, theta)`
   that takes as input a data matrix `X`$\in \mathbb{R}^{N\times d}$, the targets vector `t`$\in \{0,1\}^{N}$, a feature mapping function `map_func` that produces D-lengthed features, and a model parameter vector `theta`$\in \mathbb{R}^{D}$. The function returns the empirical logistic loss `loss`$\in$

$\mathbb{R}$ of logistic regression predictions using the given features and model parameter vector.

10. [10 points] Design the function

    `function theta_mat = logistic_regression_train_sgd (X_tr, t_tr, map_func, theta_init, I, gamma, S)`

    having the same description of inputs and outputs as the above function `perceptron_train_sgd()`, with the difference of using the logistic loss for the considered logistic regression. Moreover, the SGD uses now mini-batches of `S` samples in each mini-batch, following the cyclic scan rule over indices of

    $$\mathcal{S}_i = 1 + (S \cdot (i-1) + [0:S-1] + 1) \; mod \; N.$$

    The first update uses this rule with $i = 1$, the last with I.

The main body trains the two different model classes logistic regressions, plots the losses and shows the decision rules for some representative iterations. Use them to gain insights and validate your code.

Section 3 – Neural network

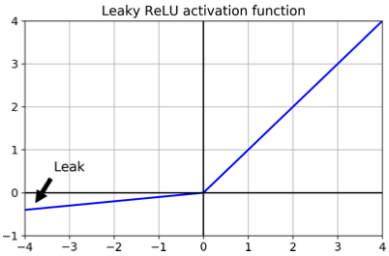
We now consider a 3-layer neural network model for binary classification, with input features of size D, number of neurons in the first hidden layer D_1 and in the second hidden layer D_2, and Leaky ReLU activations of the hidden layers. To represent the model parameters $\theta = \{W^1, W^2, w^3\}$, we use a MATLAB struct [3] to group the matrices. The model parameters can be accessed using the struct fields. For a struct named `theta`, use `theta.W1`, `theta.W2` and `theta.w3` to access them as regular variables. The functions to be designed must be written to account for a general 3-layers of arbitrary sizes each. You can assume the number of layers is 3. We use the notations in the lecture slides, and denote the input features as x rather than u as in the previous sections. This section does not use any data set. The main body calls for the listed function for two different input feature vectors, using the same model parameter vector θ as provided within the main body.

11. [5 points] Design the function

    `function out= leaky_ReLU(in)`

    that takes as input a vector `in`, and outputs a leaky ReLU with 0.1 leak coefficient activation, outputting a vector of the same size with the entry-wise activation following the rule

    $$h(a) = \begin{cases} a & a \geq 0 \\ 0.1a & a < 0 \end{cases}.$$



12. [5 points] Design the function

    `function out= grad_leaky_ReLU(in)`

    that computes entry-wise the gradient of the non linear activation function leaky ReLU with leaky coefficient of 0.1 over any size vector or matrix `in`, outputting a vector or matrix `out` of the same size. As it is not defined when an input entry is 0, we choose arbitrarily for this case the output to be 1.

13. [15 points] Design the function

    `function [logit, grad_theta] = nn_logit_and_gradient(x, t, theta)`

    that takes as input one sample `x` $\in \mathbb{R}^d$ and a struct `theta` with fields `W1,W2,w3`. Based on these two arguments, it produces the logit `logit` $\in \mathbb{R}$ of the abovementioned non-linear model with leaky ReLU activations for the corresponding weights. Furthermore, it combines the corresponding target `t` $\in \{0,1\}$ to produce a struct `grad_theta`, with inner fields (`W1,W2,w3`), containing the gradients $\nabla_\theta(-\log\, p(t|x,\theta))$ of the logistic loss.

4 REFERENCES

[1] https://uk.mathworks.com/academia/tah-portal/kings-college-london-30860095.html

[2] https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset

[3] https://www.mathworks.com/help/matlab/ref/struct.html