COMP541 - LAB #1

Deep Learning course at Koc University, Spring 2019

In this exercise, you're supposed to preprocess Boston Housing Dataset, so that we can use it in some machine learning models like linear regression later.

The housing dataset has housing related information for 506 neighborhoods in Boston from 1978. Each neighborhood is represented using 13 attributes such as crime rate or distance to employment centers. The goal is to predict the median value of the houses given in $1000's.

EXERCISE 0

In order to use some necessary functions, we need to import some modules. Just insert the following line as first line or cell,

```
using DelimitedFiles, Statistics, Random
```

`Statistics` contains statistical procedures like `mean` and `std`, `DelimitedFiles` contains our data read procedure functions (`readdlm`) and Random is for random numbers (`rand`, `Random.seed!` etc.).

EXERCISE 1

First download, and then read the file. You need to download the data within Julia notebook (please have a look: `readdlm`, `download` functions of Julia by typing e.g. `@doc readdlm`). If you look at the data, you see that each house is represented with 13 attributes separated by whitespaces and there are 506 lines in total. Here's the link to the dataset.

```
506×14 Array{Float64,2}:
 0.00632  18.0   2.31  0.0  0.538  6.575  …  296.0  15.3  396.9    4.98  24.0
 0.02731   0.0   7.07  0.0  0.469  6.421     242.0  17.8  396.9    9.14  21.6
 0.02729   0.0   7.07  0.0  0.469  7.185     242.0  17.8  392.83   4.03  34.7
 0.03237   0.0   2.18  0.0  0.458  6.998     222.0  18.7  394.63   2.94  33.4
 0.06905   0.0   2.18  0.0  0.458  7.147     222.0  18.7  396.9    5.33  36.2
 0.02985   0.0   2.18  0.0  0.458  6.43   …  222.0  18.7  394.12   5.21  28.7
 0.08829  12.5   7.87  0.0  0.524  6.012     311.0  15.2  395.6   12.43  22.9
 0.14455  12.5   7.87  0.0  0.524  6.172     311.0  15.2  396.9   19.15  27.1
 0.21124  12.5   7.87  0.0  0.524  5.631     311.0  15.2  386.63  29.93  16.5
 0.17004  12.5   7.87  0.0  0.524  6.004     311.0  15.2  386.71  17.1   18.9
 0.22489  12.5   7.87  0.0  0.524  6.377  …  311.0  15.2  392.52  20.45  15.0
 0.11747  12.5   7.87  0.0  0.524  6.009     311.0  15.2  396.9   13.27  18.9
 0.09378  12.5   7.87  0.0  0.524  5.889     311.0  15.2  390.5   15.71  21.7
 ⋮                                  ⋮    ⋱          ⋮
 0.27957   0.0   9.69  0.0  0.585  5.926     391.0  19.2  396.9   13.59  24.5
 0.17899   0.0   9.69  0.0  0.585  5.67   …  391.0  19.2  393.29  17.6   23.1
 0.2896    0.0   9.69  0.0  0.585  5.39      391.0  19.2  396.9   21.14  19.7
 0.26838   0.0   9.69  0.0  0.585  5.794     391.0  19.2  396.9   14.1   18.3
 0.23912   0.0   9.69  0.0  0.585  6.019     391.0  19.2  396.9   12.92  21.2
 0.17783   0.0   9.69  0.0  0.585  5.569     391.0  19.2  395.77  15.1   17.5
 0.22438   0.0   9.69  0.0  0.585  6.027  …  391.0  19.2  396.9   14.33  16.8
 0.06263   0.0  11.93  0.0  0.573  6.593     273.0  21.0  391.99   9.67  22.4
 0.04527   0.0  11.93  0.0  0.573  6.12      273.0  21.0  396.9    9.08  20.6
 0.06076   0.0  11.93  0.0  0.573  6.976     273.0  21.0  396.9    5.64  23.9
 0.10959   0.0  11.93  0.0  0.573  6.794     273.0  21.0  393.45   6.48  22.0
 0.04741   0.0  11.93  0.0  0.573  6.03   …  273.0  21.0  396.9    7.88  11.9
```

EXERCISE 2

The resulting data matrix should have 506 rows representing neighborhoods and 14 columns representing the attributes. The last attribute is the median house price to be predicted, so let's separate it. Also, take transpose of this data matrix to make data convenient with common mathematical notation (deep learning people represent instances/samples as column vectors mostly). We will use Julia's array indexing operation to split the data array into input x and output y. (Hint: you may want to reshape y array into a matrix with size 1x506, use reshape procedure for this purpose)

```
13×506 Array{Float64,2}:
  0.00632    0.02731    0.02729  …    0.06076    0.10959    0.04741
 18.0        0.0        0.0           0.0        0.0        0.0
  2.31       7.07       7.07         11.93      11.93      11.93
  0.0        0.0        0.0           0.0        0.0        0.0
  0.538      0.469      0.469         0.573      0.573      0.573
  6.575      6.421      7.185    …    6.976      6.794      6.03
 65.2       78.9       61.1          91.0       89.3       80.8
  4.09       4.9671     4.9671        2.1675     2.3889     2.505
  1.0        2.0        2.0           1.0        1.0        1.0
296.0      242.0      242.0         273.0      273.0      273.0
 15.3       17.8       17.8    …    21.0       21.0       21.0
396.9      396.9      392.83        396.9      393.45     396.9
  4.98       9.14       4.03          5.64       6.48       7.88
```

EXERCISE 3

As you can see, input attributes have different ranges. We need to normalize attributes by subtracting their mean and then dividing by their standard deviation (hint: take means and standard deviations of row vectors). The mean and std functions calculate mean and standard deviation values of x. Calculate mean and standard deviation values. Perform normalization on input data.

```
13×506 Array{Float64,2}:
 -0.419367  -0.416927  -0.416929  …  -0.413038  -0.407361  -0.41459
  0.284548  -0.48724   -0.48724      -0.48724   -0.48724   -0.48724
 -1.28664   -0.592794  -0.592794      0.115624   0.115624   0.115624
 -0.272329  -0.272329  -0.272329     -0.272329  -0.272329  -0.272329
 -0.144075  -0.73953   -0.73953       0.157968   0.157968   0.157968
  0.413263   0.194082   1.28145   …   0.983986   0.724955  -0.362408
 -0.119895   0.366803  -0.265549      0.796661   0.736268   0.434302
  0.140075   0.556609   0.556609     -0.772919  -0.667776  -0.61264
 -0.981871  -0.867024  -0.867024     -0.981871  -0.981871  -0.981871
 -0.665949  -0.986353  -0.986353     -0.802418  -0.802418  -0.802418
 -1.45756   -0.302794  -0.302794  …   1.1753     1.1753     1.1753
  0.440616   0.440616   0.396035      0.440616   0.402826   0.440616
 -1.0745    -0.491953  -1.20753      -0.982076  -0.864446  -0.668397
```

Important Note on Random Number Generation

Before generating random numbers, strings etc., you need to set a seed, because Julia uses a pseudo random number generator. In pseudo random number generators you set a seed and you obtain some certain random number generation order based on that seed. If you don’t set a seed, the results you obtain in the next exercises will be different. When you fail in some part, run the cells again starting from the cell or line you set random seed.

EXERCISE 4

It is necessary to split our dataset into training and test subsets so we can estimate how good our model will perform on unseen data. There are 506 house

in our dataset. Let's take 400 of them randomly, use them as training data. Let the rest be test data. In the end, you will have 4 different arrays: xtrn, ytrn, xtst and ytst.

Use randperm function to split our dataset into train and test sets. Note that, results will differ since usage of randperm function introduces randomness. If you want to overcome this randomness, set a seed by using Random.seed! function. In this exercise, please set seed as 1 just before randperm call and you will get exactly the same results. Use @doc macro to see documentation about randperm and Random.seed! (e.g. type @doc randperm to Julia REPL or notebook).

You can see the results of the split operation below.

```
xtrn
```

```
13×400 Array{Float64,2}:
 -0.410443   -0.113706    -0.414262   …  -0.41168    -0.405218   -0.408378
 -0.48724    -0.48724      1.01346       2.08539    -0.48724    -0.48724
 -1.26477    -0.180279    -0.740017     -1.37701    -0.375604   -0.164245
 -0.272329   -0.272329    -0.272329     -0.272329   -0.272329   -0.272329
 -0.575564   -0.0922961   -1.00792      -1.24006    -0.299411   -0.0664067
 -0.968713   -1.86678     -0.823542   …  -0.570203    0.269515    0.612518
  0.75403    -1.09329     -1.42723      -1.77893     1.01337     0.462722
 -0.383311   -0.605802     1.3514        3.28405    -0.64688    -0.53072
 -0.752178   -0.637331    -0.981871     -0.637331   -0.522484   -0.407638
 -1.27709    -0.618482    -0.618482      0.0163931  -0.143809    0.140995
 -0.302794   -0.0256513   -0.718509   …  -0.0718418   1.12911    -0.302794
  0.37599    -0.068175     0.40907       0.390558    0.422433    0.426267
  0.185818   -0.00182931  -0.0312367    -0.681      -0.0536423  -0.349117
```

```
ytrn
```

```
1×400 Array{Float64,2}:
 26.4  16.1  17.1  19.0  21.7  17.4  …  23.9  18.3  8.8  18.6  19.8  22.8
```

```
xtst
```

```
13×106 Array{Float64,2}:
 -0.411772   1.25847   -0.41599   …   0.241611  -0.403706  -0.405478
 -0.48724   -0.48724    0.970582      -0.48724   -0.48724    1.44223
  2.11552    1.01499   -0.735644       1.01499    0.405697  -1.12192
 -0.272329  -0.272329  -0.272329      -0.272329  -0.272329  -0.272329
  0.227006   0.365083  -1.0502         0.244266  -1.01568   -1.01568
 -0.399413   0.807503   0.434612  …  -0.242855  -0.703988   0.386221
  0.551536   1.11639   -1.00093        0.398776  -0.375678  -1.40236
 -0.758719  -1.1063     0.805741      -0.118318   1.1991     0.366459
 -0.867024   1.6596    -0.292791       1.6596    -0.637331  -0.522484
 -1.30676    1.52941   -0.470147       1.52941   -0.707483  -0.0607412
  0.297683   0.805778  -1.08803   …    0.805778  -1.13422   -1.50375
  0.22998   -1.94221    0.428019       0.394392   0.440616   0.286609
  0.226428   0.998022  -0.44154        0.325853   0.446283  -1.13331
```

```
ytst
```

```
1×106 Array{Float64,2}:
 20.3  27.5  22.0  30.7  19.4  24.5  …  24.8  42.3  16.3  19.1  20.3  29.8
```

EXERCISE 5

Our data is ready to be used. This week, we will not deal with the training of a model, but let’s look at how good a randomly initialized linear regression model performs on our processed data.

Basically, we need to use some weights with whom we’re going to multiply the attributes of houses so that we can predict the price of that house. Neighborhoods are represented with 13 attributes and we need to predict the prices which is a single number. We need to have a weight matrices with size of 1x13. We also use a bias value which is 0.

To create weight matrix, we will sample from normal distribution with zero mean and a small standard deviation. In this tutorial, our standard deviation value is equal to 0.1. Use randn function to create a random weight matrix whose values are sampled from a unit normal distribution (mean=0, standard

deviation=1). Multiply our weight matrix by 0.1 which is our desired standard deviation. We will not use bias in this tutorial.

```
w - weight matrix
1×13 Array{Float64,2}:
 0.0426637  0.0378471  -0.169487  …  -0.0208622  -0.066244  -0.0257344
```

EXERCISE 6

Now, we have input and weights. Let's write a function to predict price. Implement a function takes weight matrix and neighborhood attributes as input and outputs a single value, house price prediction. Simply perform a matrix multiplication inside this function and return the output vector. We call this function as predict function.

Call predict function and store the output as ypred.

```
ypred
1×400 Array{Float64,2}:
 0.227025  -0.0623266  -0.0604358  …  0.207288  0.0592333  -0.0221864
```

ypred is an 1x400 dimensional array/matrix. Each value in this array is the model's price prediction for an average house in corresponding neighborhood.

EXERCISE 7

Let's implement a loss function which is called as Mean Squared Error (MSE),

$$J = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

In this function we calculate J, our loss value, average of squared difference between real price values and predicted price values.

Implement MSE loss function which takes weight matrix, input matrix (xtrn or xtst) and ground truth prices (ytrn or ytst). Make first parameter of loss function weight matrix, it's not crucial, but make it a habit. Helpful functions: sum, mean, size, abs2, .* You don't have to use all of them. Use abs2 with dot syntax as `abs2.(x)` if you're using it. Calculate the loss value for both splits by using your MSE loss function.

`(train loss/test loss): (297.3036503276774,299.0172855668773)`

EXERCISE 8

Lastly, let's find in how many of them, the model predicts the price with an error less than average error. Measure the absolute difference between the predicted price and correct price for each neighborhood and compare those differences with the square root of the loss value calculated in previous exercise. Use sqrt function (with dot syntax, e.g. `sqrt.(x)`) to take square roots. Perform this step for only training set. The result should be `108`.