ML Laboratory 01: Linear Regression

1. Objective

Students should understand and be able use a linear regression model in Matlab

2. Theoretical aspects

Regression vs Classification

A typical job in Machine Learning is to **predict** an output y from some given data $(x_1, ...x_n)$:

$$X = \begin{bmatrix} x_1 & x_2 & ... & x_N \end{bmatrix} \rightarrow y$$

In **supervised learning**, we are given many examples (input-output pairs) out of which we need to deduce the prediction rule.

Depending on the meaning of y, we can have:

- **classification**: y is a number representing a category (0 = cat, 1 = dog, 2 = bird). The numbers have no real meaning as numbers, whey are just numerical labels for representing the categories.
- **regression**: y is a number which actually means a numerical result (it is the price of a house, or a probability, etc.). Regression is also known as "curve fitting".

Linear regression: the model

The linear regression model: the output is assumed to be a **linear combination** of the inputs.

$$y \approx w_1 x_1 + w_2 x_2 + ... + w_N x_N + b.$$

The coefficients w_i and b are parameters we need to estimate/find. "Learning" means finding good values for the parameters, which get the job done.

Linear regression: the parameters

The parameters of the linear regression model are the **weights** $w_1, w_2, ...w_N$ and the **bias** value b (also known as the **intercept**).

Cost function (loss function)

Given some parameters w_i and b, how do we know if they are good?

For an input vector X:

- we compute the **prediction**:

$$\hat{y} = w_1 x_1 + w_2 x_2 + ... + w_N x_N + b$$

- we compare the prediction against the true result ("**ground-truth**") with the **cost function** (also known as **loss function**):

$$J = (\hat{y} - y)^2$$

The cost function defines what is good and what is bad, depending on its result (the cost): - cost is small => prediction is good - cost is big => prediction is bad

If we have many data (input-output pairs), the overall cost is the average of the cost for each entry:

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

The cost function can be anything. Here, and typically for linear regression, we have the **quadratic cost function** (also known as "least squares", "ℓ_2 norm", ...). This is typical good cost function for regression, but not so good for classification problems.

$$(\hat{y} - y)^2$$

Other cost functions can be used, and they lead to different results (sometimes better, sometimes worse, depending on the problem).

Matrix form of linear regression

The linear regression problem can be written in matrix form as follows:

$$\begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^N \end{bmatrix} \approx \begin{bmatrix} \hat{y}^1 \\ \hat{y}^2 \\ \vdots \\ \hat{y}^N \end{bmatrix} = \begin{bmatrix} x_1^1 & x_2^1 & x_3^1 & ... & x_{11}^1 & 1 \\ x_1^2 & x_2^2 & x_3^2 & ... & x_{11}^2 & 1 \\ \vdots & \vdots & \vdots & ... & \vdots & \vdots \\ x_1^N & x_2^N & x_3^N & ... & x_{11}^N & 1 \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \\ b \end{bmatrix}$$

Naming the matrices and vectors as Y, $\hat{Y}$, X, W, we have:

$$Y \approx \hat{Y} = X \cdot W$$

Note two important things:

- We can treat b just like another weight, which multiplies some constant value 1. We can extend the input matrix with a column of 1's, and consider b just like the 12-th weight in W.
- The same weights appear in all linear combinations. They are the unknowns in this linear equation system.
- This is a **massively overcomplete** equation system. There is probably no exact solution, but there exists a **least-squares solution**, i.e. the solution vector W which makes the predictions $\hat{Y}$ **as close as possible** to the true values Y (i.e. minimum cost).

How to train linear regression?

Training = **learning** = finding good values for the unknown parameters.

For linear regression, we can do it in three ways:

1. Closed form solution

When the cost function is the quadratic, the best solution can actually be found analytically (this may be the only such case in the whole of Machine Learning :)):

$$W_{optimal} = X^{\dagger}Y = (X^*X)^{-1} X^* \cdot Y$$

This is not true anymore if we change the cost function.

The sign * signifies **transposition**.

2. Matlab function doing the job for us

Linear regression can be fitted in Matlab using the function `fitml()`:

```
mdl = fitlm(X,Y)  % X are the inputs, Y is the target vector, mdl is a model object
```

3. Optimization with Gradient Descent

The cost function J is a function like any other, and it depends on the parameters w_i.

We can compute the derivative of J with respect to each parameter, $\frac{dJ}{dw_i}$.

The derivative tells us how the cost J changes for a small increase in the parameter w_i. We want to reduce the cost function. If the derivative is positive, we'll make w_i a little smaller. If it is negative, we'll increase it a bit.

Gradient Descent optimization procedure:

1. set μ = step size = small (e.g. 0.001)
2. initialize parameters w_i somehow (random)
3. Repeat:

- compute predictions and cost J
- compute derivative of cost with respect to parameters $\frac{dJ}{dw_i}$.
- update each parameter like:

$$w_i = w_i - \mu \cdot \frac{dJ}{dw_i}$$

If we group all derivatives in a vector, this is known as the **gradient** vector:

$$\nabla W = \begin{bmatrix} \frac{dJ}{dw_1} & \frac{dJ}{dw_2} & \dots & \frac{dJ}{dw_k} \end{bmatrix}.$$

In matrix form, the update rule can be written as:

$$W = W - \mu \nabla W.$$

For linear regression with the quadratic cost function, it can be shown that the gradient is equal to:

$$\nabla W = X^*(\hat{Y} - Y)$$

3. Practical work

The data used in this example comes from here: https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009

The data contains 11 numerical chemical measurements for some different brands of red wines, together with a quality score indicated by buyers. The job is to determine how do the measured parameters influence the quality score. The ultimate goal is to predict the quality for some new type of red wine, based on its measurements.

Inputs:

- 1 - fixed acidity
- 2 - volatile acidity
- 3 - citric acid
- 4 - residual sugar
- 5 - chlorides
- 6 - free sulfur dioxide
- 7 - total sulfur dioxide
- 8 - density
- 9 - pH
- 10 - sulphates
- 11 - alcohol

Outputs:

- 12 - quality

Let's load the data first

```
Data = readmatrix('winequality-red.csv');
X = Data(:,1:11);   % 11 columns for the inputs
Y = Data(:,12);     % 1 column for the output
N = size(Data,1);   % The number of wines in the set (1599)
```

Extend the X matrix so we can treat the bias b as just another weight.

```
X = [X ones(N,1)]
```

Let's initialize the weights to some random values

```
W = randn(12, 1)   % a column vector
```

Task 1: Compute and show the cost function with the above weights

```
%======================
% Your code here
%======================
```

Task 2: Compute the solution with the closed-form formula

- **Question**: According to the model parameters, which is the most important factor in determining the perceived quality of the wine?

```
%======================
% Your code here
%======================
```

Task 3: Compute the solution with the Matlab function `fitml()` and check that it is the same

```
%======================
% Your code here
%======================
```

Task 4: Implement optimization with Gradient Descent

```
%======================
% To fill in
%======================

W = randn(12, 1);           % initialize parameters randomly

number_of_epochs = 1000;    % set number of iterations

for iter = 1:number_of_epochs

    % Compute predictions:
    Ypred = ...

    % Compute cost:
    J(iter) = 1/N * ...
```

```
    % Compute derivatives according to the given formula
    dW = ...

    % Update the weights
    mu = 0.0001;           % try multiple values here
    W = W - mu * dW;

    % Store the weights history
    W_hist(:,iter) = W;
end

% Plot the error and the evolution of the weights
plot(J)
figure
plot(W_hist)
```

Task 5: Repeat, with normalized input data

In many machine learning tasks, it is better to normalize the input data. Normalization means, in this context, to make each column (each feature) zero-mean and unit-variance, by subtracting the mean and dividing to the standard deviation:

$$x \to \frac{x - \mu}{\sigma},$$

where μ is the mean and σ is the standard deviation of every input column (σ^2 is the variance).

Do this preprocessing and repeat the training from above. You can use the Matlab function `normalize()` for this purpose. How are the results?

Task 6: Create new features

The linear model tries to estimate the output only as a linear combination of the inputs x_i. But what if the output depends, say, on the squared value x_i^2 of an input? The model cannot capture this. Instead, we can create new features manually, by performing transformations of the input data.

Augment the existing dataset by appending, for every column x_i, new columns with the squared values x_i^2, and also all possible products $x_i \cdot x_j$. Then repeat the training process on the augmented data. How are the results?

4. Final questions

1. What happens when μ is too large? What if it is too small?