Machine Learning and Big Data Processing: Lab sessions

LAB2: PRESENTATION

Esther Rodrigo Bonet esther.rodrigo.bonet@vub.be (PL9.2.27)
Leandro Di Bella leandro.di.bella@vub.be (PL9.2.36)

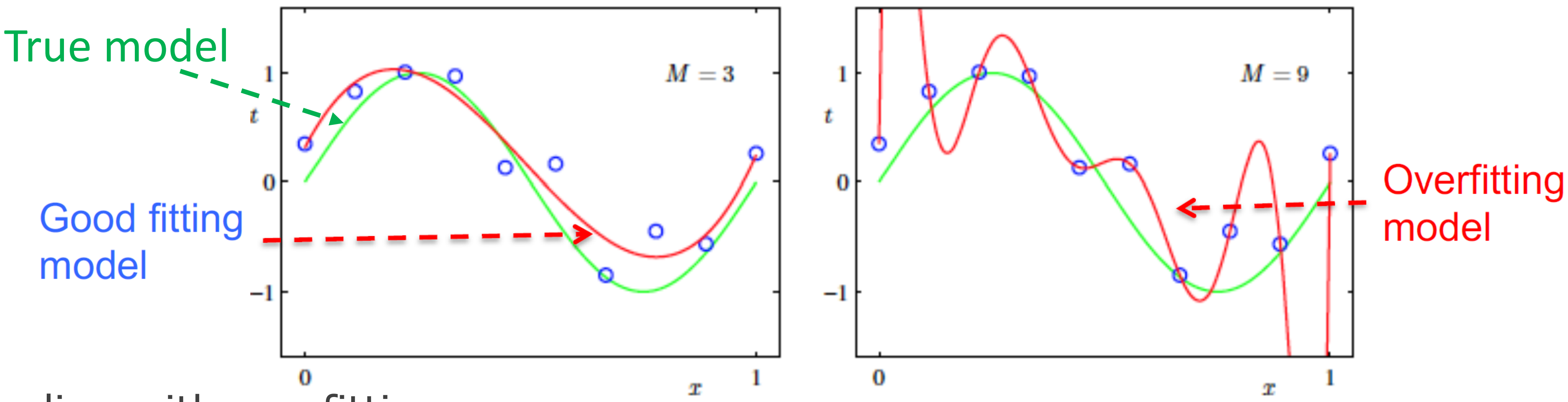
Content

- **Regression**
  - Regularized Regression
- **Gradient Descent**
  - Linear Regression using Gradient Descent
  - Regularized Linear Regression using Gradient Descent
  - Linear Regression using Stochastic and Mini-batch Gradient Descent

Regularized Regression

Regularized Regression

- The problem of overfitting



- Dealing with overfitting
    - ***Reduce number of features***
        - Disadvantage: lose some information
    - ***Regularization***
        - Advantage: keep all features

Regularized Regression

- Parameter estimation is done by minimizing the REGULARIZED sum of squared residuals (RSS).

Regularization Parameter

$$\min_{\beta_0,\beta_1,\ldots,\beta_{m-1}} \frac{1}{2n}\left[\sum_{i=1}^{n}(y_i-\hat{y}_i)^2+\boxed{\lambda}\sum_{j=0}^{m-1}\beta_j^2\right]$$

- In the L-2 norm form this minimization problem can be solved by:

$$g(\beta)=\|y-\mathbf{X}\beta\|_2^2+\lambda\|\beta\|_2^2=y^Ty-2y^T\mathbf{X}\beta+\beta^T\mathbf{X}^T\mathbf{X}\beta+\lambda\beta^T\beta$$

$$\frac{\partial}{\partial\beta}g(\beta)=0 \implies \beta=(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^Ty$$

Regularized Regression

- The optimal coefficients in regularized linear regression are given by:

Regularization parameter

Response vector

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$$\boldsymbol{\beta} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

(m+1)x(m+1) matrix

Regression coefficient

Data matrix

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_m \end{bmatrix}$$

$$\boldsymbol{X} = \begin{bmatrix} 1 & X_{11} & X_{12} & X_{13} \dots & X_{1m} \\ 1 & X_{21} & X_{22} & X_{23} \dots & X_{2m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & X_{n1} & X_{n2} & X_{n3} \dots & X_{nm} \end{bmatrix}$$

$$\boldsymbol{I} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & & & & \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

Regularized Regression

- The predicted value can be calculated by:

$$\hat{y} = X\beta$$

- Call the *Mean Square Error (MSE)* and *Mean Absolute Error (MAE)* functions to evaluate the learned model. (which has finished in *ex3*)

Gradient Descent

Linear Regression using Gradient Descent

- In linear regression, a hypothesis function need to be found that maps input features to target values:

$$\hat{y}_i = h(\underline{x}_i) = \underline{\theta}^T \underline{x}_i = \theta_0 + \sum_{j=1}^{m} \theta_j x_{i,j}$$

The number of features

i-th predicted target value

Parameter vector

i-th features

The bias

- The cost function we defined as:

$$\mathcal{J}_{\underline{\theta}} = \frac{1}{2n} \sum_{i=1}^{n} \left(h\left(\underline{x}_i\right) - y_i\right)^2 \xrightarrow{\textit{Matrix form}} \mathcal{J}_{\underline{\theta}} = \frac{1}{2n} \left(X\underline{\theta} - \underline{y}\right)^T \left(X\underline{\theta} - \underline{y}\right)$$

$$X \in \mathbb{R}^{n \times (m+1)} \qquad \underline{\theta} \in \mathbb{R}^{(m+1) \times 1} \qquad \underline{y} \in \mathbb{R}^{n \times 1}$$

Linear Regression using Gradient Descent

- In linear regression, a hypothesis function need to be found that maps input features to target values:

$$\hat{y}_i = h(\underline{x}_i) = \underline{\theta}^T \underline{x}_i = \theta_0 + \sum_{j=1}^{m} \theta_j x_{i,j}$$

The number of features

i-th predicted target value *Parameter vector* *i-th features* *The bias*

- The cost function we defined as:

$$\boxed{\mathcal{J}_{\underline{\theta}} = \frac{1}{2n} \sum_{i=1}^{n} \left(h\left(\underline{x}_i\right) - y_i\right)^2}$$

Matrix form

$$\mathcal{J}_{\underline{\theta}} = \frac{1}{2n} \left(X\underline{\theta} - \underline{y}\right)^T \left(X\underline{\theta} - \underline{y}\right)$$

$X \in \mathbb{R}^{n\times(m+1)}$ $\underline{\theta} \in \mathbb{R}^{(m+1)\times 1}$ $\underline{y} \in \mathbb{R}^{n\times 1}$

Linear Regression using Gradient Descent

- Cost Function Minimization:
  - Iterative algorithmic processes: Gradient Descent

$$\mathcal{J}_{\underline{\theta}} = \frac{1}{2n}\left(X\underline{\theta} - \underline{y}\right)^{\mathsf{T}}\left(X\underline{\theta} - \underline{y}\right)$$

- The gradient descent of the cost function with respect to its parameter is:

$$\frac{\partial \mathcal{J}}{\partial \underline{\theta}} = \frac{1}{n} X^{\mathsf{T}}(X\underline{\theta} - \underline{y}).$$

- Gradient Descent Algorithm:
  - **Step 1**: start with a random initialization of a parameter vector $\underline{\Theta}$
  - **Step 2**: update $\underline{\Theta}$ according to:

Learning rate

$$\underline{\theta}^{\mathrm{new}} = \underline{\theta}^{\mathrm{old}} - \alpha \frac{\partial \mathcal{J}}{\partial \underline{\theta}^{\mathrm{old}}}$$

  - **Step 3**: repeat step 2 for a predefined number of times or until a certain criterion is met.

Linear Regression using Gradient Descent

- Cost Function Minimization:
  - Iterative algorithmic processes: Gradient Descent

$$\mathcal{J}_{\underline{\theta}} = \frac{1}{2n}\left(X\underline{\theta} - \underline{y}\right)^{\mathsf{T}}\left(X\underline{\theta} - \underline{y}\right)$$

- The gradient descent of the cost function with respect to its parameter is:

$$\frac{\partial \mathcal{J}}{\partial \underline{\theta}} = \frac{1}{n} X^{\mathsf{T}}(X\underline{\theta} - \underline{y}).$$

- Gradient Descent Algorithm:
  - **Step 1**: start with a random initialization of a parameter vector $\underline{\Theta}$
  - **Step 2**: update $\underline{\Theta}$ according to:

Learning rate

$$\underline{\theta}^{\text{new}} = \underline{\theta}^{\text{old}} - \alpha \frac{\partial \mathcal{J}}{\partial \underline{\theta}^{\text{old}}}$$

  - **Step 3**: repeat step 2 for a predefined number of times or until a certain criterion is met.

Linear Regression using Gradient Descent

- Cost Function Minimization:
  - Iterative algorithmic processes: Gradient Descent

$$\mathcal{J}_{\underline{\theta}} = \frac{1}{2n}\left(X\underline{\theta} - \underline{y}\right)^{T}\left(X\underline{\theta} - \underline{y}\right)$$

- The gradient descent of the cost function with respect to its parameter is:

$$\boxed{\frac{\partial \mathcal{J}}{\partial \underline{\theta}} = \frac{1}{n} X^{T}(X\underline{\theta} - \underline{y}).}$$

- Gradient Descent Algorithm:
  - **Step 1**: start with a random initialization of a parameter vector $\underline{\Theta}$
  - **Step 2**: update $\underline{\Theta}$ according to:

Learning rate

$$\underline{\theta}^{\text{new}} = \underline{\theta}^{\text{old}} - \alpha \frac{\partial \mathcal{J}}{\partial \underline{\theta}^{\text{old}}}$$

  - **Step 3**: repeat step 2 for a predefined number of times or until a certain criterion is met.

Linear Regression using Gradient Descent

- Cost Function Minimization:
  - Iterative algorithmic processes: Gradient Descent

$$\mathcal{J}_{\underline{\theta}} = \frac{1}{2n}\left(X\underline{\theta} - \underline{y}\right)^{T}\left(X\underline{\theta} - \underline{y}\right)$$

- The gradient descent of the cost function with respect to its parameter is:

$$\frac{\partial \mathcal{J}}{\partial \underline{\theta}} = \frac{1}{n} X^{T}\left(X\underline{\theta} - \underline{y}\right).$$

- Gradient Descent Algorithm:
  - **Step 1**: start with a random initialization of a parameter vector $\underline{\theta}$
  - **Step 2**: update $\underline{\theta}$ according to:

Learning rate

$$\underline{\theta}^{\text{new}} = \underline{\theta}^{\text{old}} - \alpha \frac{\partial \mathcal{J}}{\partial \underline{\theta}^{\text{old}}}$$

  - **Step 3**: repeat step 2 for a predefined number of times or until a certain criterion is met.

Linear Regression using Gradient Descent

- The gradient estimation can be obtained following the definition of gradient, that is:

$$\frac{\partial \mathcal{J}_{\underline{\theta}}}{\partial \theta_i} \approx \frac{\mathcal{J}(\theta_1, \theta_2, \theta_i + \epsilon, \cdots, \theta_m) - \mathcal{J}(\theta_1, \theta_2, \theta_i - \epsilon, \cdots, \theta_m)}{2\epsilon},$$

- Correct implementations of the two methods should result in gradients with very small sum of squared errors (around 10^{-18})
- α can strongly affect the performance of the final model after training, we will do an experiment to see the effects of α on the GD algorithm.

Regularized Linear Regression using Gradient Descent

- Applying L-2 regularization technique, the cost function can de defined as:

$$\mathcal{J}_{\underline{\theta}} = \frac{1}{2n}\sum_{i=1}^{n}\left(h\left(\underline{x}_i\right) - y_i\right)^2 + \lambda\sum_{j=1}^{m}\theta_j^2,$$

Regularization parameter : control the effect of the regularization term

- The gradient of the cost function with respect to its parameter is:

$$\frac{\partial \mathcal{J}}{\partial \underline{\theta}} = \frac{1}{n}X^T(X\underline{\theta} - \underline{y}) + \lambda\underline{\theta}.$$

- Employ the GD algorithm to learn $\underline{\Theta}$

Stochastic Gradient Descent (SGD)

- Linear Regression Cost Function:

$$J(\theta) = \frac{1}{2n}\sum_{i=1}^{n}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}^{(i)} - y^{(i)}\right)x_j^{(i)}$$

- Vanilla GD and SGD

1) In SGD, before for-looping, you need to randomly shuffle the training examples.
2) In SGD, its path to the minima is noisier (more random) than that of the Vanilla gradient.

Conclusion: In **SGD**, the cost gradient of **1 example** at each iteration is used, In **Vanilla GD**, the sum of the cost gradient of **ALL** examples is used.

Vanilla GD algorithm

Vanilla (Batch) G.D.

$$\theta^{new} = \theta^{old} - \alpha \boxed{\frac{\partial}{\partial\theta_j}J(\theta)}$$

$$= \frac{1}{n}\boxed{\sum_{i=1}^{n}}\left(\hat{y}^{(i)} - y^{(i)}\right)x_j^{(i)}$$

(for every $j = 0, 1, \cdots, m$)

SGD algorithm

Stochastic G.D.

for i in range (n):

$$\theta^{new} = \theta^{old} - \alpha \cdot \boxed{\text{only one example}}$$

$$= \left(\hat{y}^{(i)} - y^{(i)}\right)x_j^{(i)}$$

(for every $j = 0, 1, \cdots, m$)

Mini-batch Gradient Descent (MBGD)

- Mini-batch Gradient Descent is a good alternative for both GD and SGD algorithms

Vanilla GD algorithm

Vanilla (Batch) G.D.

$$\theta^{new} = \theta^{old} - \alpha \boxed{\frac{\partial}{\partial \theta_j} J(\theta)}$$

$$= \frac{1}{n} \boxed{\sum_{i=1}^{n}} (\hat{y}^{(i)} - y^{(i)}) x_j^{(i)}$$

(for every $j = 0, 1, \ldots, m$)

MBGD algorithm

Mini-batch G.D.

b: batch size

for $i = 1, 1+b, 1+2b, \ldots, \boxed{1+Nb}$ $\leq m$

$$\theta^{new} = \theta^{old} - \alpha \cdot \frac{1}{b} \boxed{\sum_{k=i}^{i+b+1}} (\hat{y}^{(k)} - y^{(k)}) x_j^{(k)}$$

(for every $j = 0, 1, \ldots, m$)

SGD algorithm

Stochastic G.D.

for i in range(n):

$$\theta^{new} = \theta^{old} - \alpha \cdot \boxed{\text{only one example}}$$

$$= (\hat{y}^{(i)} - y^{(i)}) x_j^{(i)}$$

(for every $j = 0, 1, \ldots, m$)

- **Gradient Descent**: Use all m examples in each iteration
- **SGD**: Use 1 example in each iteration
- **MBGD**: Use b examples in each iteration ($b < m$)