Managing Big Data

Regression

Manolis Tzagarakis
Assistant Professor
Department of Economics
University of Patras

tzagara@upatras.gr
2610 969845
google:tzagara
Facebook: tzagara
SkypeID: tzagara
QuakeLive: DeusEx
Steam: xmachina1

Regression analysis

- Investigate and quantify the relationship between **one variable which takes continuous values** and **a set of other variables that may take any type of value**
  - In particular, investigate **the effects of one (or more) variable(s) onto the value of another variable in the dataset**.
    - E.g. How does the value of one variable change if other variable(s) change value?
- Goal: come up with a model (i.e. a function) that **predicts and/or explains** the value of one variable based on the values of other variables.

Regression analysis

- In regression, the relationship expressed is between one variable -called the **dependent variable**- and one or more **independent variables**
- **Very Important!** In regression, dependent variable **takes continuous values**
  - Independent variables can be of any type
- Relationship between variables take the **form of a function/equation**: Aims at expressing the value of the dependent variable as a function of the values of other independent variables.
  - This function also referred to as "**regression model**", "**regression equation**" or plain "**regression**".

Regression analysis

- Regression equations can take many different forms
  - But does not imply a deterministic relationship
- Examples of regression equations/models
  - **FoodConsumption = 0.78 Income + 1459**
    - e.g. for quantifying the relationship between annual **FoodConsumption (dependent variable)** of families and their annual **income (independent)**
  - **CarValue = PurchaseValue - $e^{(0.88*age)}$ e.g.**
    - E.g for quantifying the relationship between the **present value of a car (dependent variable)** and the variables **purchase value and age (independent variables)**.

Regression analysis

- Purpose of regression models
  - **Explain the variance** in the dependent variable based on the values of the independent variables(s) of the existing dataset
  - **Predict the value** of the dependent variable based on the values of the independent variable(s)

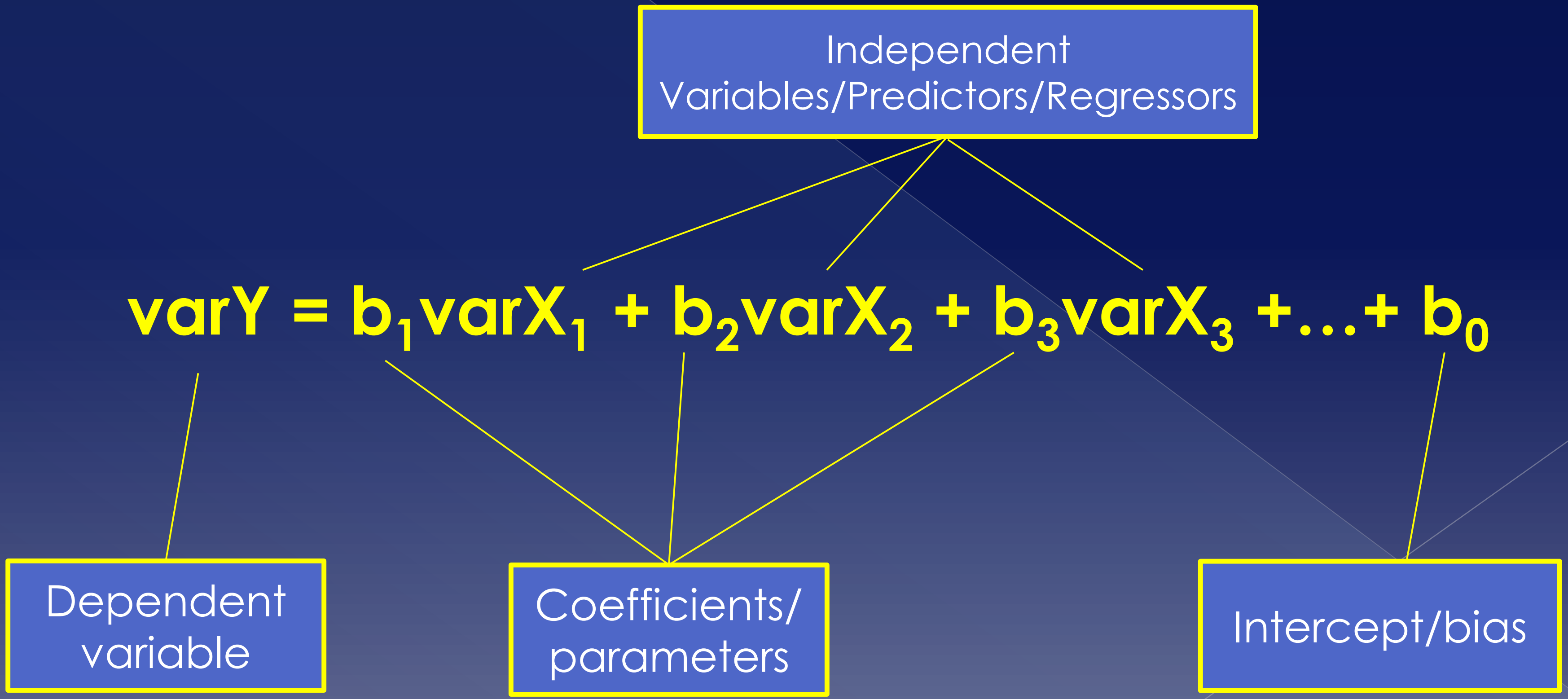
Regression analysis

- Regression analysis **requires a training set with observations on these variables** from which the relationship between the interested variables will be quantified.
- A regression model tries to come up with an **equation that best "fits"** the data in the training set.
  - There can be many regression equations that fit the data, but we look for the one that fit the best
  - This "fit" can assessed and the usefulness of the model can be determined.

Regression analysis

Terminology

General form of a regression model capturing the relationship between variable varY and variables $varX_1$, $varX_2$,….



Independent Variables/Predictors/Regressors

$$varY = b_1varX_1 + b_2varX_2 + b_3varX_3 +\ldots+ b_0$$

Dependent variable

Coefficients/ parameters

Intercept/bias

Regression analysis

- Examples of regression models (note the coefficients and independent variables):

$$\mathbf{Καταναλωση} = \boldsymbol{\beta_1 Εισόδημα^2} + \boldsymbol{\beta_0}$$

$$\mathbf{Αρτηριακη\ πιεση} = \boldsymbol{\beta_1 Φύλλο} + \boldsymbol{\beta_2 \sqrt{Ηλικια}} + \boldsymbol{\beta_0}$$

$$\boldsymbol{ln(Εισόδημα)} \\ = \boldsymbol{\beta_1}\mathbf{Εμπειρια} + \boldsymbol{\beta_2}\mathbf{Εμπειρια} + \boldsymbol{\beta_3}\mathbf{Εκπαιδευση} + \boldsymbol{\beta_0}$$

$$\boldsymbol{Crop\ yield} = \frac{\mathbf{1}}{\boldsymbol{(b_1 + b_2 CropSpacing)^{b_3}}} + \boldsymbol{\beta_0}$$

Regression analysis

- In a regression model, **the unknowns are the coefficients of the independent variables (usually represented as betas b_i) and the goal is to estimate them** from the training set and assess their importance
  - Estimation of coefficients is done using the existing training dataset
  - The coefficients bi for each independent variable tell us how that independent variable influences the dependent.
  - Values of the independent variables are known from the training data

Regression analysis

- Types of regression models
  - Based on **how the value of the dependent variable changes** when the values of the independent variables or coefficients change (That's very, very important and always to keep in mind – determines the form of the regression model)
    - i.e. how a change in the coefficients/parameters and independent variables affect the dependent variable.
    - Expressed as rate of change: $\frac{\Delta Y}{\Delta X}, \frac{\Delta Y}{\Delta b}$
  - Two types
    - **Linear** regression models
    - **Nonlinear** regression models

Regression analysis

- Types of regression models
  - Linear regression models
    - In linear regression models the **dependent variable depends linearly on all the coefficients/parameters (and only the coefficients!)**
    - **"Depends linearly"** means that the **rate of change of the dependent variable** -if the coefficient changes- is **independent of the value of the coefficient (i.e. constant wrt coefficient)**. I.e. **rate of change constant**
    - Linear regression models **do not need to depend linearly on the independent variables!**

Regression analysis

- Types of regression models
  - Linear regression models
    - The model **Consumption = b_1Income + b_0** is a linear model because the value of consumption **depends linearly on all coefficients b** of the model: (e.g. assuming coefficient b_1 increases by some ε):

$$\frac{\Delta \mathbf{Consumption}}{\Delta b_1} = \frac{(b_1 + \varepsilon)Income + \; b_0 - b_1 Income \; - b_0}{\varepsilon} = Income$$

Independent of the value of coefficient b_1 that changed. Hence the model is linear. Model happens to be also linear with respect to independent variable Income.

Regression analysis

- Types of regression models
  - Linear regression models
    - Dependent variable does **not need to be linearly dependent on the independent variables** (can be but its not required). **This means linear regression models –when plotted- can form curves**.
    - Hence that all the following regression models are also linear although they are not linear with respect to the independent variables.

All these are considered linear models because they depend linearly on the coefficients. Not on the independent variables. You may substitute e.g. $FamilySize^3$ with a new variable say Z

$$Consumption = b_1 Income + b_2 FamiltySize^3 + b_0$$

$$BloodPresure = b_1 Sex + b_2 \sqrt{Age} + b_0$$

$$\ln(Income) = b_1 Experience + b_2 Experience^2 + b_3 YearsEducation + b_0$$

Regression analysis

- Types of regression models
  - Nonlinear regression models
    - In nonlinear regression models the **dependent variable does <u>not depend linearly</u> on all the coefficients (and only the coefficients!)**
    - **"Does not depend linearly"** means that the **rate of change of the dependent variable** -if one of the coefficient changes- is **dependent of the value of the coefficient** i.e. definitely not constant!

Regression analysis

- Types of regression models
  - Nonlinear regression models
    - The following are examples of nonlinear regression models

$$\boldsymbol{Rate\ of\ reaction = \frac{b_1 Concentration}{b_2 + Concentration}}$$

Rate of a chemical reaction and the concentration of substance

$$\boldsymbol{Crop\ yield = \frac{1}{(b_1 +\ b_2 CropSpacing)^{b_3}}}$$

Regression analysis

- The notion of error in regression models
  - Regression models are **approximations** that **try to fit in the best way possible the real data and values** of the dependent variable in the training set.
    - You always have a sample – the training set. Never the population.
    - Because regression models approximate the value of the dependent variable, they **never succeed in capturing/predicting/estimating the real value** of the dependent variable.
      - But what is the real value of the dependent variable?

Regression analysis

- The notion of error in regression models
  - Two types of errors in regression models
    - Errors/disturbance
      - **The difference between the <u>(unobserved) real value</u> of the dependent variable <u>in the population</u> and the observed value in the training set.** <u>This error can never be observed or measured</u> because we are unaware of the real value of the dependent variable in the population.
    - Residuals/fitting deviations
      - **The difference between the dependent value in the training set and the predicted/estimated value by the regression model**. <u>This can be observed and measured</u>

Regression analysis

- The notion of error in regression models
  - Errors and residuals are included in the regression models.
    - Adding term **ε** (for error) when showing the general model or **ε_i** (for residuals) when
  - Full specification of a regression model includes error term e.g.

$$Consumption = b_1 Income + b_2 FamiltySize^3 + b_0 + \varepsilon$$

  - If the error term is not explicitly included in the regression model, it's implied.
    - This means, there always is an error term in the model!

Regression analysis

- More types of linear regression models
  - **Simple** linear regression models
    - When the regression model includes only 2 variables: one dependent and one independent variable
      - E.g. **Income = b_1Education + b_0**
  - **Multiple** linear regression models
    - When the regression model includes more than 2 variables
      - E.g. **Income = b_1Education + b_2Experience + b_0**

Regression analysis

- Who comes up with regression models and how?
  - Domain experts (economist, statisticians, engineers, etc)
    - Theory
      - **Read the relevant literature and identify factors** that affect the value of the dependent variable
    - Determine the purpose of the model
      - Explain the variance or predict the dependent variable?
        - These influence how the model will be evaluated.
    - Look at the data and how it changes
      - From existing data, see **how** the dependent variable changes when the independent variables change
    - Trial and error
      - Begin by **trying simple regression models and assess the results.** Continue by modifying the model if results are not appropriate.

Regression analysis

- Who comes up with regression models?
  - **Don't forget:** The coefficients of a model are calculated from the existing dataset (the training set) and these capture the relationships between the variables. Hence, regression models are approximations that **try to fit** the best way possible **the available data in the training set**.

Regression analysis

- In regression analysis you
  - First estimate the parameters from the training set
  - Then assess and evaluate the model to see if it meets the objectives, is useful and conclusions can be drawn
    - Evaluation methods depend on the model’s purpose (explanation or prediction?)

Estimating parameters in regression models

Estimating parameters

- In a regression model, the problem is **estimating the coefficients/parameters** that will **indicate/quantify the relationships** between the variables
  - Coefficients/parameters are estimated from an existing dataset (the **training set**) which is required.

Training set.

Tid	House Price	Marital Status	Income	m^2House
1	190K	Single	125K	180
2	145K	Married	100K	154
3	101K	Single	70K	110
4	187K	Married	120K	167
5	109K	Divorced	95K	110
6	96K	Married	60K	90
7	200K	Divorced	220K	190

$$\text{Income} = b_1 m^2\text{House} + b_0$$

Unknowns are the parameters b (independent variables known from training set). The parameters b of this regression model are estimated using the training set. The goal: find the best values of b which best fit the values of the dependent variable in the training set.

Estimating parameters

- **Different methods** to estimate parameters based on the type of the regression model
  - E.g. Linear vs Nonlinear
- The general idea: Estimation of parameters in regression model (linear or nonlinear) **involves a Cost function (also called “Loss function”)** that needs to be **minimized**.

Estimating parameters

- **Cost function** tries to measure **how big the error of the regression model is** when estimating the value of the dependent variable
  - The model with smallest error fits the data well i.e. is the best one.
  - Essentially, error is the sum of residuals which is to be minimized
  - Cost functions can have many different forms
    - Depending on the purpose
    - The **form of the cost function** determines the **type of regression**: Ordinary Least Squares (OLS), LASSO, Quantile etc

Estimating parameters in linear regression models

Estimating parameters

- **Linear regression models** have the following general form:

$$Y = b_1X_1 + b_2X_2 + b_3X_3 + \cdots b_kX_k + b_0 + \varepsilon$$

Where:
Y: Dependent variable
X_i : Independent variable i
b_i: Parameter to be estimated
ε: Error term

Estimating parameters

- Since linear regression models try to **fit the available training data**, the linear regression model can also be written in the form:

$$\boldsymbol{Y_i = b_1X_{1i} + b_2X_{2i} + b_3X_{3i} + \cdots + b_kX_{ki} + b_0 + e_i}$$

Where:

Y_i : Value of dependent variable in observation i in training set

X_{ki} : Value of independent variable k in observation i of the training set

b_i: Parameter to be estimated

e_i : Residual of the i-th observation in the training set

Estimating parameters

- If there are n observations in the training set, then there will be n equations, one for each observation, of the form:

$$\boldsymbol{Y_i = \ b_1X_{1i} + b_2X_{2i} + b_3X_{3i} + \cdots + b_kX_{ki} + \ b_0 + e_i}$$

Estimating parameters

- Because **parameters are estimated from the training set** and not the **true real values of the variables (remember: the training set is just a sample)**, the estimates are mentioned in the regression model by adding a hat (^)

$$\widehat{Y}_i = \widehat{b}_1 X_{1i} + \widehat{b}_2 X_{2i} + \widehat{b}_3 X_{3i} + \cdots + \widehat{b}_k X_{ki} + \widehat{b}_0 + e_i$$

Where:

$\widehat{Y}_i$: The **estimated** value of the dependent variable

$\widehat{b}_i$: The **estimated** value of the parameter i.

Estimating parameters

- Regression model in matrix notation
  - It’s customary to represent these n regression equation in matrix notation. If we define:

Matrix of values of independent variables in training set.

$$\hat{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \dots \\ Y_n \end{bmatrix} \quad X = \begin{bmatrix} 1 & X_{11} & X_{21} & \dots & X_{k1} \\ 1 & X_{12} & X_{22} & \dots & X_{k2} \\ 1 & X_{13} & X_{23} & \dots & X_{k3} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & X_{1n} & X_{2n} & \dots & X_{kn} \end{bmatrix} \quad \hat{b} = \begin{bmatrix} \hat{b}_0 \\ \hat{b}_1 \\ \hat{b}_2 \\ \dots \\ \hat{b}_k \end{bmatrix} \quad e = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ \dots \\ e_n \end{bmatrix}$$

  - Then the n linear regression equations, derived from the training set, can be written in matrix form:

$$\hat{Y} = X\hat{b} + e$$

If you carry out the operations, you’ll get the n linear regression equations as vectors.
The matrix form makes it easier to calculate the parameters using machines (i.e. computers).

Estimating parameters

- Two methods for estimating the parameters of linear regression models
  - **Ordinary Least Squares (OLS)**
  - **Gradient Descent and its variations**
- Each of the above method appropriate in specific situations.

Ordinary Least Squares - OLS

Estimating parameters

- Ordinary Least Square (OLS) Regression
  - In OLS the cost function is the **Sum of Squared Errors (SSE) i.e. sum of residuals** which must be minimized:

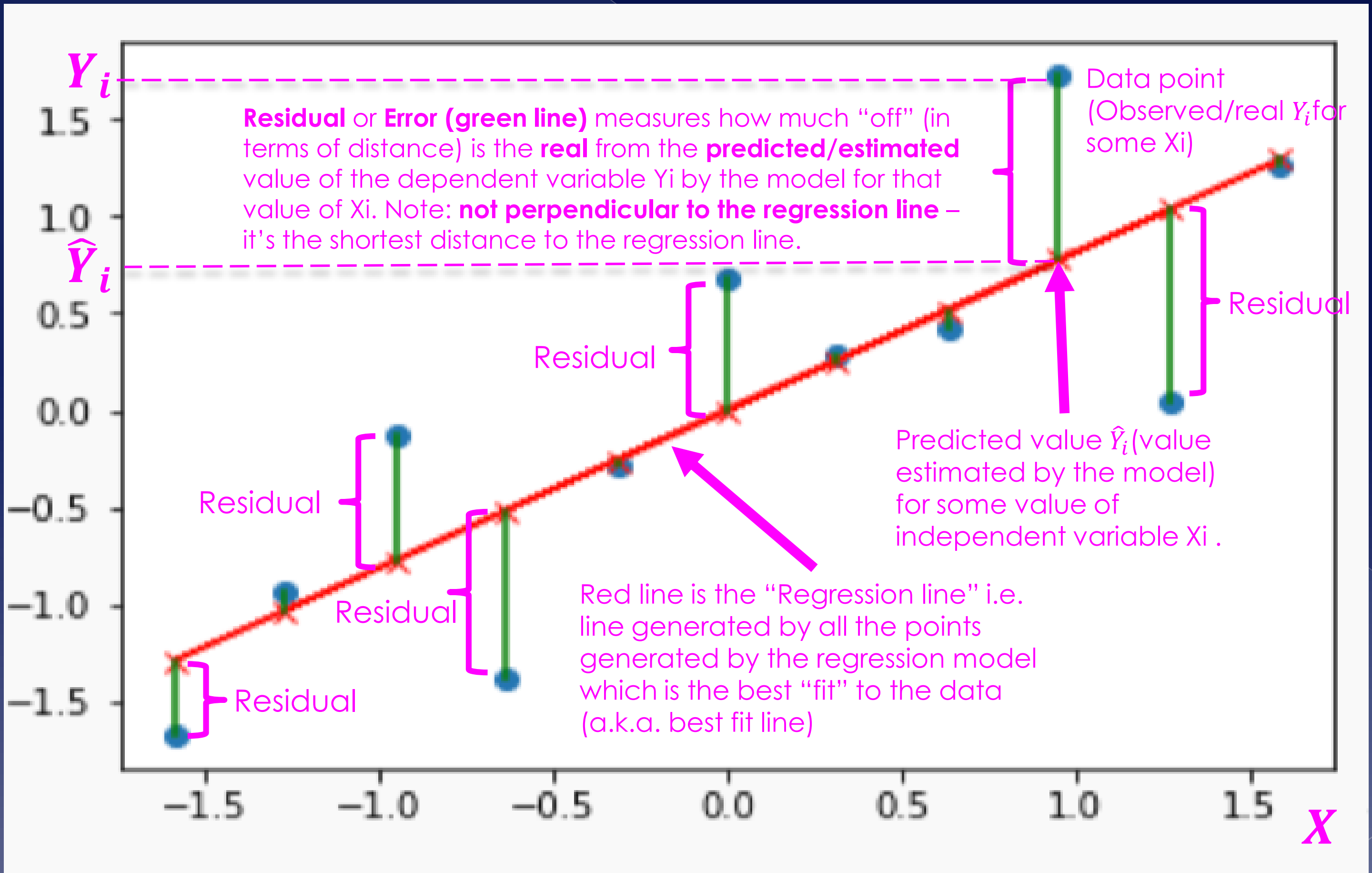
$$SSE = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$$

Y_i = Value of the dependent variable in observation i of the training set
$\hat{Y}_i$ = Estimated value by the linear regression model for the values of the independent variables in observation i in the training set.

  - The **parameters b which minimize** the above SSE **are the parameter estimates of the linear regression model that best fit the training data.**

Estimating parameters

- Geometric interpretation of SSE?
  - i.e. **Draw it for me** with crayons pls…



Residual or **Error (green line)** measures how much "off" (in terms of distance) is the **real** from the **predicted/estimated** value of the dependent variable Yi by the model for that value of Xi. Note: **not perpendicular to the regression line** – it's the shortest distance to the regression line.

Data point (Observed/real Y_i for some Xi)

Predicted value $\hat{Y}_i$ (value estimated by the model) for some value of independent variable Xi .

Red line is the "Regression line" i.e. line generated by all the points generated by the regression model which is the best "fit" to the data (a.k.a. best fit line)

The basic idea:
Sum of all squared residuals is a measure of how good a regression line (i.e. the predicted or estimated values by a regression model) is – how good it fits the data.
The **best regression line**, that fits best to the data, has of course the **smallest sum of squared residuals or errors.**

Estimating parameters

- But why Sum of **SQUARED** errors?
  - i.e. why **SQUARE** the errors?

$$SSE = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2$$

Why this SQUARE in residuals ????

1st reason: You need positive residuals or errors because they are in essence **distances** from the regression line.
So, why not use absolute measures instead e.g. $|Y_i - \hat{Y}_i|$? You can actually and that's used in many situations –it's called L1 norm of residuals! If you use such way of measuring residuals, the regression in not called OLS anymore. It's called Least Absolute Deviations – LAD – regression and used in Robust regression.
However, L1 norm of residuals has some other issues (see 2nd reason)

2nd reason: You want to penalize large residuals. I.e. **PUNISH large residuals**! This means that if you have big residuals, **YOU WANT TO MAKE THEM EVEN BIGGER** so that **SSE gets EVEN BIGGER when large residuals are present** !!! Why is that? Ask yourself: What's a better model: many small errors or 1 huge error? Now assume that Sum of small errors = 1 huge error. (L1 norm of residuals cannot differentiate. Squares CAN! I.e. "*missing by a little lots of times is better than missing by a lot a few times*".
But, if so, why not raise to the power of say 6 or 8 or 10 to punish large residuals more? **You can do this also!** But this adds little and makes calculations more difficult. Hence, raising to the power of 2 is cheap and does the job…
(Note: you do understand that you CANNOT raise it to an odd power?)

Estimating parameters

- Ordinary Least Square (OLS) Regression
  - How is the Cost function (SSE) minimized in OLS?
  - First write **SSE in matrix notation** as a function of the **vector b**

$$\boldsymbol{SSE\left(\hat{b}\right) = e^{T} e = \left(Y - X\hat{b}\right)^{T}\left(Y - X\hat{b}\right)}$$

  - And then minimize the Cost function (SSE) by solving the equation of partial derivatives:

$$\boldsymbol{\frac{\partial SSE\left(\hat{b}\right)}{\partial \hat{b}} = 0}$$

This equation **has a closed form solution** due to the form of the linear regression. Solving this will calculate the vector b that minimizes the SSE and hence finds the parameters we are looking for.

Estimating parameters

- Ordinary Least Square (OLS) Regression
  - The **closed form solution** derived from the previous equation for estimating the parameters b of a linear regression model in matrix form is:

$$\widehat{b} = \left(X^T X\right)^{-1} X^T Y$$

  - The above **closed form formula – <u>called normal equation</u> - gives you the vector of parameters estimates b**, based on the matrix of values of the independent variables X and the matrix of the values of the dependent variable Y in the training set, **which minimize SSE** and hence fit the best way possible the training data

Gradient descent

GD: Estimating parameters

- Gradient descent
  - While OLS minimizes the SSE in a very specific way (by finding the values of b who yield the partial derivative to zero) leading to a **closed form formula (the normal equation)** for estimating the parameters, Gradient descent **minimizes the cost function in a very different way.**
    - It also has a cost/loss function but the way it minimizes it is different
  - **Gradient descent is an iterative, numerical optimization method for minimizing the cost function and thus finding the parameter estimates**.
    - i.e. Gadient descent does not offer a closed form formula like the normal equation in OLS for calculating the parameters.
    - “iterative” ? Tries to **guess the proper coefficient values** of the parameters that lead to minimizing the cost function

GD: Estimating parameters

- Gradient descent
  - Why Gradient descent?
    - OLS has **three main concerns**
      - **1)** The normal equation requires inversion of a matrix:

$$\hat{b} = \underbrace{\left(X^T X\right)^{-1}}_{\text{Matrix inversion}} X^T Y$$

- **Matrix inversion <u>is a very expensive operation on computers</u>**. I.e. it takes a lot of time to calculate the inverse. If the **X^TX matrix has 100 variables** (i.e. is an 100x100 matrix), since an inversion requires n^3 operations (n=dimension of matrix) on average, it would require **~1000000 operations** to invert the matrix.
- **OLS performs poorly in big data settings!**
- **Gradient descent is much faster in estimating the parameters in such Big Data settings.**

GD: Estimating parameters

- Gradient descent
  - Why Gradient descent?
    - **2)** OLS makes the assumption that the matrix of independent variables X fits into the computer's main memory (RAM).
    - You cannot make this assumption when working with Big Data.
      - In big data settings, these matrices of the normal equation may not fit into the main memory (RAM). In such cases **OLS does not work AT ALL** because the normal equation can't be calculated.
      - How can you in such situations estimate the coefficients?
      - **Gradient descent offers versions that can estimate the parameters even when data does not fit into RAM.**

GD: Estimating parameters

- Gradient descent
  - Why Gradient descent?
    - **3)** OLS makes the assumption that you have all the data available when you start your parameter estimation.
    - In Big Data environments you cannot make this assumption.
      - What happens when you don't have all your data available because it arrives at some interval?
      - How can you in such situations estimate the coefficients?
      - You can't use the normal equation in such settings!
      - **There are versions of Gradient descent that can be used in such circumstances.**

GD: Estimating parameters

- Gradient descent
  - Why Gradient descent?
    - **4)** OLS and the normal equation **DOES NOT WORK AT ALL** when the number of observations is smaller than the dimension of the dataset i.e. the number of variables!
      - E.g. when you have 700 variables and 650 observations.
      - Why? Because in such cases $(X^T X)^{-1}$ is not computable! I.e. $(X^T X)$ cannot be inverted.
        - Leaving the proof to you ☺
      - **Gradient descent can be used in such situations**.
        - Offers .e.g. regularization

GD: Estimating parameters

- Gradient descent
  - Why Gradient descent?
    - **Gradient descent performs much better** –in terms of execution times/number of operations- **in big data contexts** than OLS and in such situations it's exclusively used.
    - Gradient descent works even when the data does not fit into main memory or when entire data not available at the beginning – different versions of GD
  - **Warning!** Gradient descent uses a different notation for the multiple linear regression model:

$$h_\theta\left(x^{(i)}\right) = \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_k x_k^{(i)} + \theta_0 + \varepsilon$$

θ_j = Parameter j (to be estimated)

$x_j^{(i)}$ = Value of independent variable j in observation i in training set

GD: Estimating parameters

- Cost function in Gradient descent
  - In Gradient descent the **cost function is called the mean squared error, J(θ)**

$$J(\theta_0, \theta_1, \dots, \theta_\kappa) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Where:
θ_i = (Unknown) parameter i of the linear regression model from a total of k+1 parameters
m = Number of observation in training set
$h_\theta()$ = The estimated value of the linear regression model for the values of the independent variables at observation i in training set.
$x^{(i)}$ = The values of the independent variables of observation i in training set
$y^{(i)}$ = The value of the dependent variable of observation i in training set

GD: Estimating parameters

- Cost function in Gradient descent
  - Gradient descent **attempts to minimize the cost function J(θ)** by finding/estimating the proper values of parameters θ.

$$\underbrace{J(\theta_0, \theta_1, \dots, \theta_\kappa)}_{} = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)^2$$

Often abbreviated simply as J(θ).

GD: Estimating parameters

- Cost function in Gradient descent
  - Cost function has things in common with the cost function (i.e. SSE) in OLS but differs a little for some reasons

$$J(\theta_0,\theta_1,\dots,\theta_\kappa) = \frac{1}{2m}\underbrace{\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2}_{\text{Sum of squared errors in OLS}}$$

Why divide by 2m? m because of two reasons: i) it's the **mean** squared error and ii) it yields to **smaller numbers** which is important **due to the numerical nature of the method.** Also, include the constant 2 in denominator to make things simpler as it's shown later on (hint: it will be eliminated). However, these terms do not affect the minimization process.

Form of the linear regression model, with θ the unknown parameters, is:

$$h_\theta\left(x^{(i)}\right) = \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_k x_k^{(i)} + \theta_0 + \varepsilon$$

GD: Estimating parameters

- Cost function in Gradient descent
  - Cost function in matrix form

Note: square each element of vector

$$\boldsymbol{J(\theta) = \frac{1}{2m}(X\theta - y)^T(X\theta - y) = \frac{1}{2m}(X\theta - y)^2}$$

Sum of all elements to get pure number of J(Θ).

Where:

$$\boldsymbol{y = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} 1 & X_{11} & X_{21} & \dots & X_{k1} \\ 1 & X_{12} & X_{22} & \dots & X_{k2} \\ 1 & X_{13} & X_{23} & \dots & X_{k3} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & X_{1n} & X_{2n} & \dots & X_{kn} \end{bmatrix} \quad \theta = \begin{bmatrix} \hat{\theta}_0 \\ \hat{\theta}_1 \\ \hat{\theta}_2 \\ \dots \\ \hat{\theta}_k \end{bmatrix}}$$

Vector of dependent variables

Matrix of independent variables with first columns all 1s for constant term.

Vector of estimated parameters

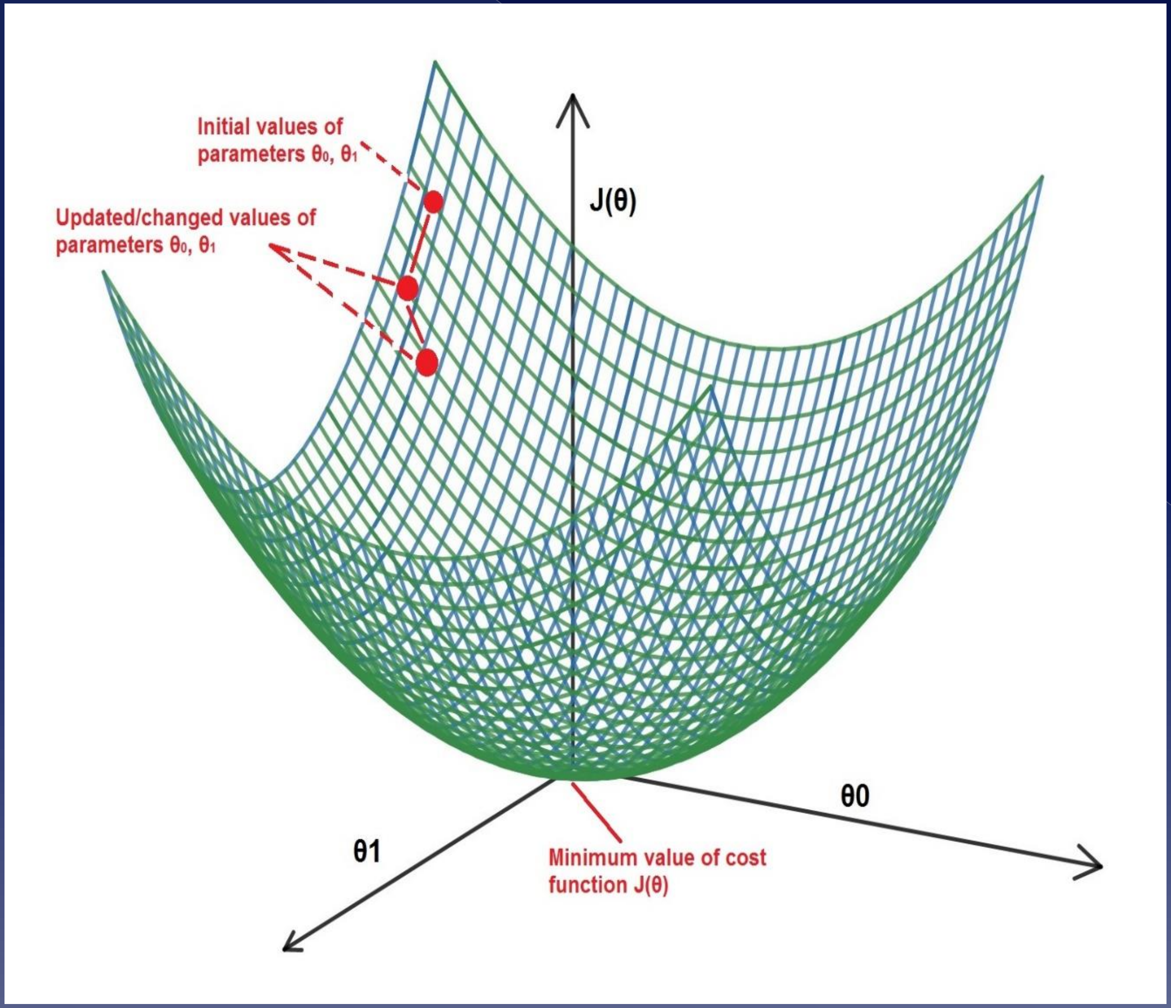
GD: Estimating parameters

- Cost function
  - A note on **notation**: cost function in **gradient descent uses different notation** (**θ** instead of **b** for parameters, **h_θ()** for linear regression model, **J(θ)** for cost function)
  - This is because Gradient descent originated from a different field. One of the **first algorithms which founded the area of machine learning** in applied mathematics
  - We use the same notation used by contemporary literature.

GD: Estimating parameters

- General idea of estimating the parameters θ with Gradient descent which **minimize the cost function J(θ) – it's a process i.e. no closed formula:**
  - Start with **initial, random values for the parameters θ**
  - **Update/Change the values of the parameters θ** in a way that **yield to smaller value of the cost function J(θ)**
  - Continue changing values of θ **iteratively until the smallest value of J(θ)** is attained.

GD: Estimating parameters



The general idea of Gradient descent.

Assume a simple linear regression model $h_\theta(x) = \theta_0 + \theta_1 x$

The cost function of such linear regression model, J(θ), will be convex and an example cost function is depicted on the left.

Gradient descent tries to modify the values of all the parameters θ iteratively, towards the smallest value of J(θ).

GD: Estimating parameters

- How to change the values of parameters θ in order to **minimize J(θ)**?
  - The **value of the first partial derivative** of the cost function J(θ) with respect to a parameter θ_j will tell us **how we need to modify the parameter θ_j** (leaving all other parameters constant) to achieve a smaller value of J(θ).
- Remember from your math classes:
  - If the **value of the first derivative of a function f(x)** with respect to x is at some point x_0
    - **positive (>0)**, an increase of x_0 leads to an increase of f(x). A decrease of x_0 leads to decrease of f(x)
    - **negative (<0)**, an increase of x_0 leads to an decrease of f(x). A decrease of x_0 leads to increase of f(x)
    - **is equal to zero (=0)**, an increase of x_0 leads to an increase or decrease of f(x) (has a point of deflection at x_0).

GD: Estimating parameters

- How to apply this rule on J(θ)?
  - In Gradient Descent J(θ) (cost function) is the f(x), with θs as our X variables.
    - Note here: we have many θs hence J(θ) has many unknowns. We address this later.
  - Thus, the value of the first derivative of J(θ) with respect to θ_j for some value θ of θ_j tells us if θ needs to increase or decrease in order to achieve an even smaller value of J(θ).
    - Value of the first derivative of J(θ) **tells us the direction of change of variable θ (increase or decrease).**

GD: Estimating parameters

- How to change the values of parameters θ in order to minimize J(θ)?
  - **J(θ) is a multivariate function**, where the parameters θ_j are the unknown variables.
  - To apply the derivative technique, we will use the **first partial derivative wrt to one θ_j** parameter and leaving other θs constant i.e. calculate the value of $\frac{\partial J(\theta)}{\partial \theta_j}$. If this value is positive, a decrease of θ_j will decrease the cost function, if it's negative, an increase of θ_j will decrease the cost function J(θ).
    - **Do this for all parameters θ** to see how they need to change **i.e. calculate $\nabla J(\theta)$**
    - **Do this iteratively** to get an even smaller value J(θ)

GD: Estimating parameters

- How to change the values of parameters θ in order to minimize J(θ)?
  - A more **clear example**
    - if initial parameters of θ = ($θ_0$, $θ_1$, $θ_2$, ..., $θ_k$) and at that point the cost function is J(θ), then if the value of $\frac{\partial J(\theta)}{\partial \theta_2}$ is negative, this means that **a small increase** (update/change) of parameter $θ_2$ leading to parameters θ' = ($θ_0$, $θ_1$, **$θ_2$+ε** ..., $θ_k$) (leaving all other θs the same) **will decrease J(θ)**. If it's positive, decrease $θ_2$ to get θ' = ($θ_0$, $θ_1$, **$θ_2$-ε** ..., $θ_k$) , to get a smaller J(θ).
      - i.e. J(θ') < J(θ)
    - **Do the same for each and all θs** in the linear regression model and update their values accordingly.
    - **Do such update for each θ iteratively** (i.e. many times over).

GD: Estimating parameters

- How to change the values of parameters θ in order to minimize J(θ)?
  - In Gradient Descent, **each parameter** is updated/changed, at each iteration, using the following formula:

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

a is a real value > 0, called the learning rate. It is a constant **given as input to gradient descent**. While the partial derivative will give us the direction in which the cost function will decrease, it does not specify how big the increase of θ should be. This is specified by the value of the learning rate a. Can be imagined as the step by which the θ will change. **Setting the appropriate value for a is very important and affects the significantly the algorithm.**

GD: Estimating parameters

- How to change the values of parameters θ in order to minimize J(θ)?
  - For a **multiple linear regression model**, you can actually calculate $\frac{\partial J(\theta)}{\partial \theta}$ for all θs, resulting in the following update formulas for the parameters θ:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)$$

Update for the constant term/intercept

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right) x_j^{(i)}$$

Update for all other parameters in the linear regression model

Where
m: number of observations in training set
$h_\theta(x^{(i)})$: value of linear regression model for the values of independent variables in observation i of the training set
$y^{(i)}$: value of the dependent variable in observation i of the training set
a : learning rate
$x_j^{(i)}$: value of independent variable x_j in observation i of the training set

Form of linear regression model: $h_\theta\left(x^{(i)}\right) = \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_k x_k^{(i)} + \theta_0 + \varepsilon$

GD: Estimating parameters

- How to change the values of parameters θ in order to minimize J(θ)?
  - In **matrix form**, the previous update formulas for parameters θ can be written as

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \boldsymbol{\alpha}\frac{\mathbf{1}}{\boldsymbol{m}}\boldsymbol{X}^{\boldsymbol{T}}\,(\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{Y})$$

Where:
m: Number of observations in training set
$\boldsymbol{\theta}$ **:** The vector of (k+1) parameters of the linear regression model
$\boldsymbol{X}$ **:** The mx(k+1) matrix of values of independent variables in the linear regression model, with the first column all 1 (ones).
$\boldsymbol{Y}$: The vector of m values of the dependent variables in the training set
a : the learning rate, given as input

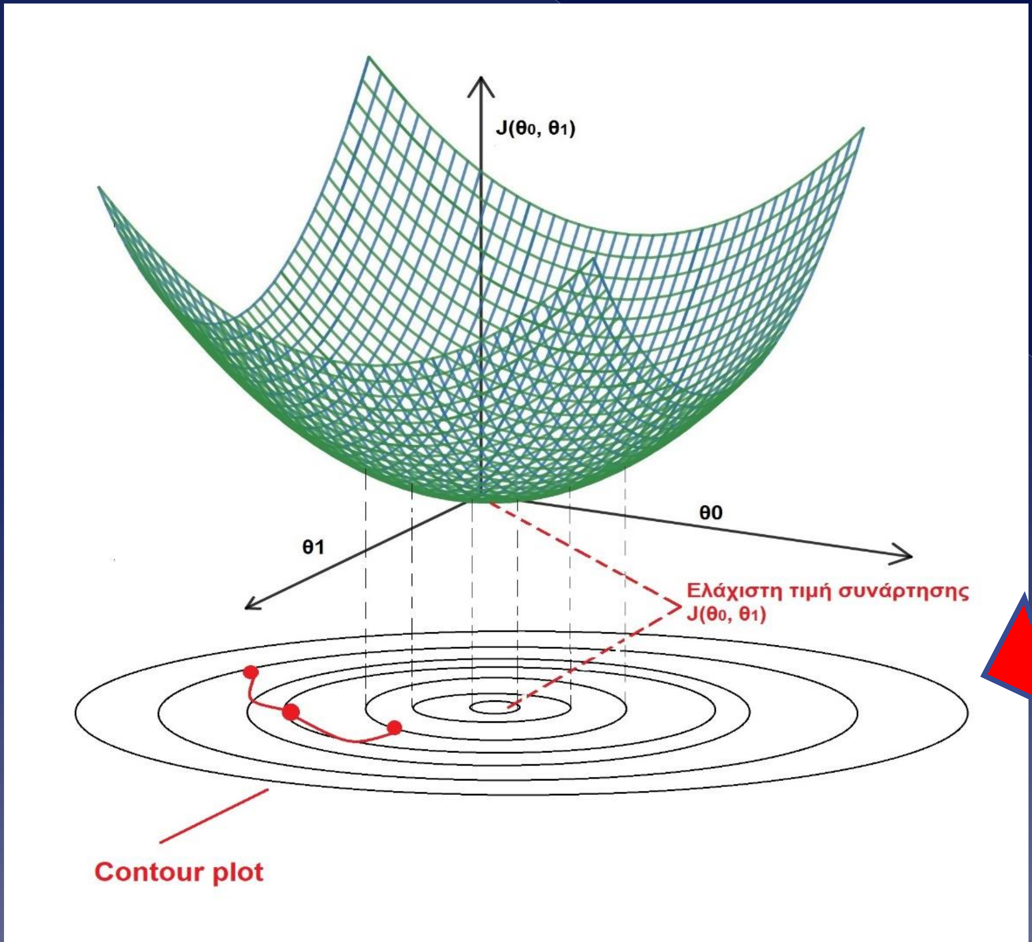
GD: Estimating parameters

- Gradient descent algorithm
  - Pseudocode

```
Initialize vector of parameters θ with random values
Initialize costVector # We will store the value of the cost function for each
                        iteration in this vector.
α = 0.01 # Set learning rate. See later how to come up with an appropriate value.

# Start iterations of Gradient descent
while termination conditions not met {

      update θ vector with  $\theta := \theta - \alpha \frac{1}{m} X^T (X\theta - Y)$
      calculate value of cost function J(θ) for the newly calculated values of θ
      Store value of cost function into vector costVector
}

# Vector θ will contain the estimated parameters which minimize J(θ)
print θ vector # Print the estimated parameters
plot costVector # Plot the costVector
```

GD: Estimating parameters

- Gradient descent usually depicted as a **contour plot**



Convergence of θs to the values which minimize J(θ) is usually depicted as Contour plot.

In a Contour plot, each circle represents the values of θ that lead to the same value of J(θ).

GD: Estimating parameters

- When does Gradient descent terminate?
  - 3 possible termination conditions
    - **When a predefined number of iterations have been completed.** Typical number of iterations are n=50, 20000 or greater depending how fast the algorithm converges
    - **When the improvement of the cost function is smaller than a predefined value**
    - **Early stopping**. With the current “version” of the cost function, **calculate the cost on a validation set (different from training set)** at each iteration. Compare the two consecutive values of J(θ) and if J(θ) starts to increase, terminate the algorithm. Used to **address overfitting**.

GD: Estimating parameters

- Gradient descent algorithm with predefined number of iterations as termination condition

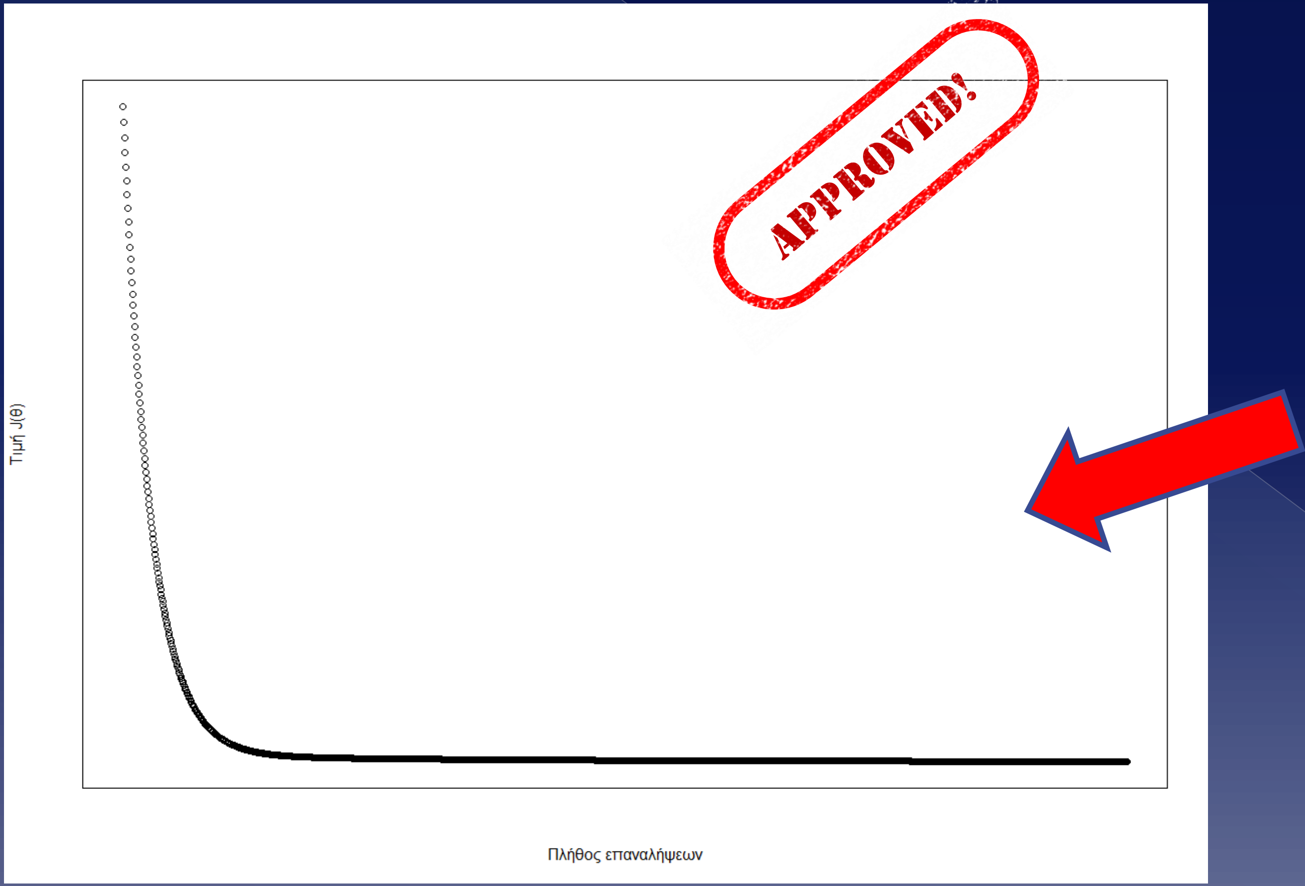
```
Initialize vector of parameters θ with random values
Initialize costVector # We will store the value of the cost function J(θ) for each iteration here
α = 0.01 # Setting the learning rate
numIterations = 10000 # Number of iterations to carry out
n = 0 # How many iterations we have done
while n < numIterations {

      update θ vector with   $\theta := \theta - \alpha \frac{1}{m} X^T \left( X\theta - \Upsilon \right)$
      calculate value of cost function J(θ) for the newly estimated values of θ
      Store value of cost function into vector costVector
      n = n + 1   # Next iteration
}

# Vector θ will contain the estimated parameters which minimize J(θ)
print θ vector
plot costVector
```

GD: Estimating parameters

- The **learning rate a**
  - Setting the **learning rate a** to the proper value is critical!
    - Determines **if and how fast Gradient descent** converges to the minimum of the cost function.
    - If the **value of the learning parameter is too small**, Gradient descent may **converge very slowly**
    - If the **value of the learning parameter is too large**, Gradient descent may **not converge at all** to the proper values of θ which minimize J(θ). May diverge!
  - How to check if learning parameter a is too small, too big or just appropriate?
    - Empirically, plot the cost function and see its shape

GD: Estimating parameters

- The learning rate a
  - **Appropriate value** of learning rate

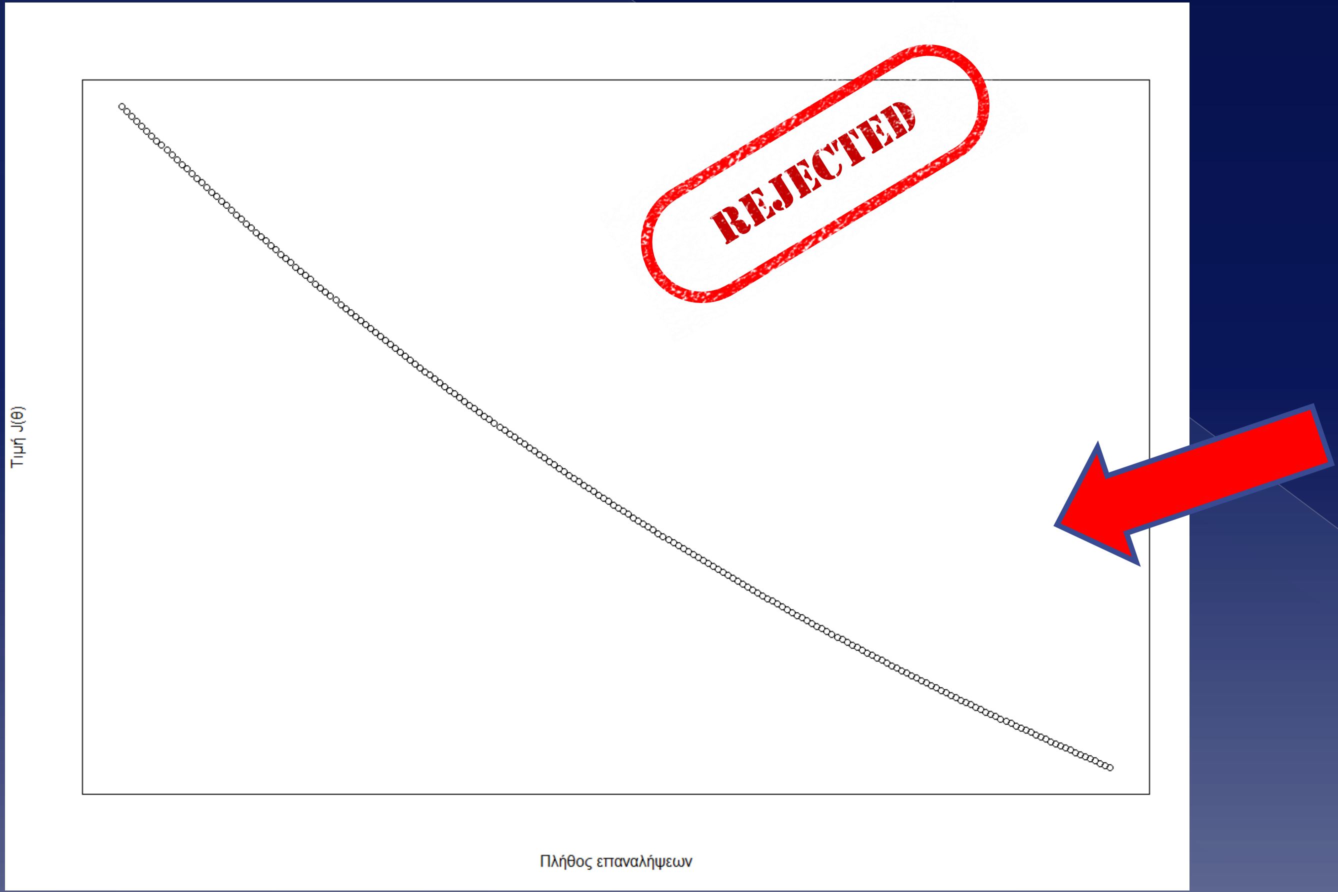


If the value of the learning rate is appropriate, the cost function J(θ) plotted against the number of iterations will have such shape. Cost function shows a steep drop and then a gradual improvement.

To check if the selected value is appropriate, run Gradient descent and plot the cost function.

GD: Estimating parameters

- The learning rate a
  - **Too small value** of the learning rate a

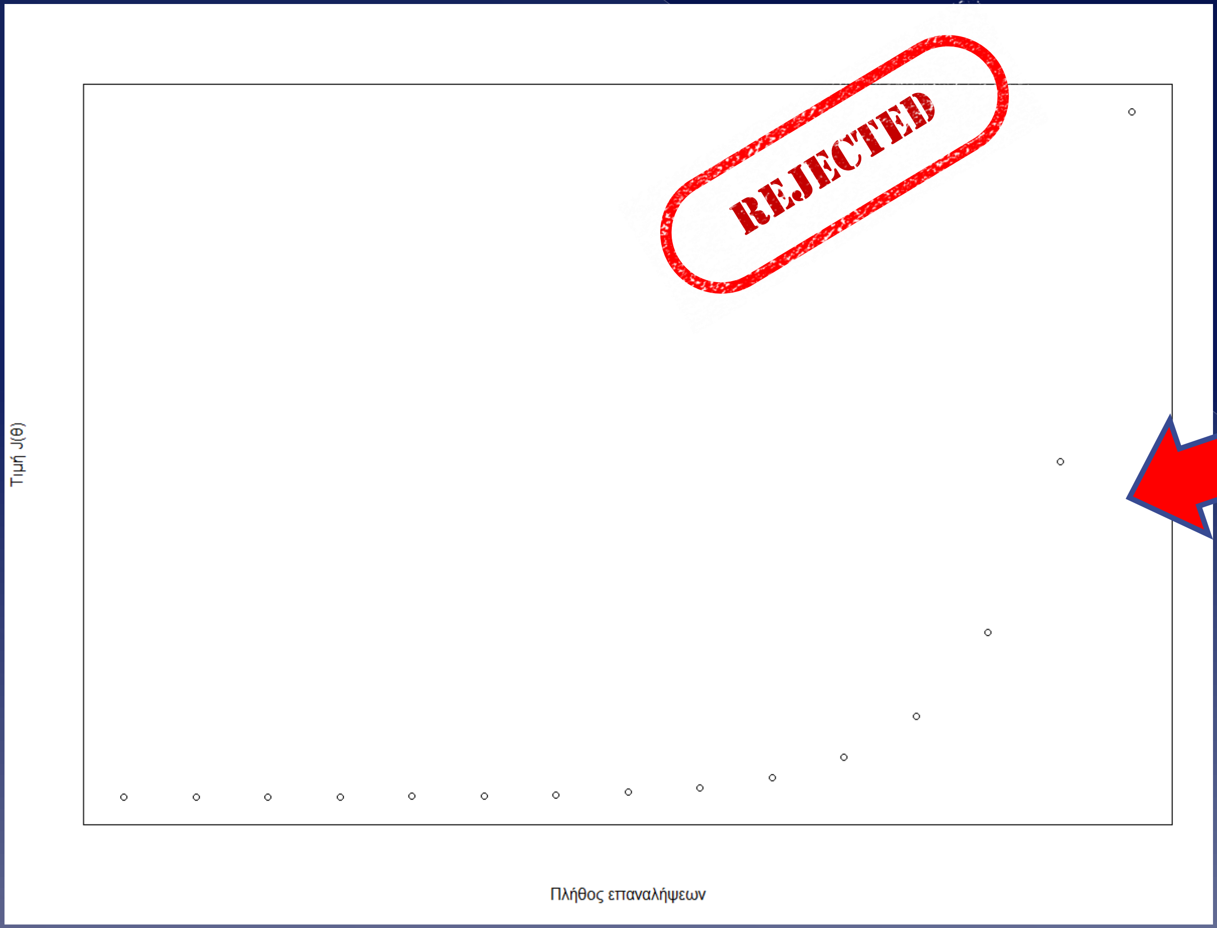


If the value of the learning rate is too small, the cost function J(θ) plotted against the number of iterations will have such shape.

This means Gradient descent will converge very, very slowly to the appropriate values of θs that minimize J(θ).

GD: Estimating parameters

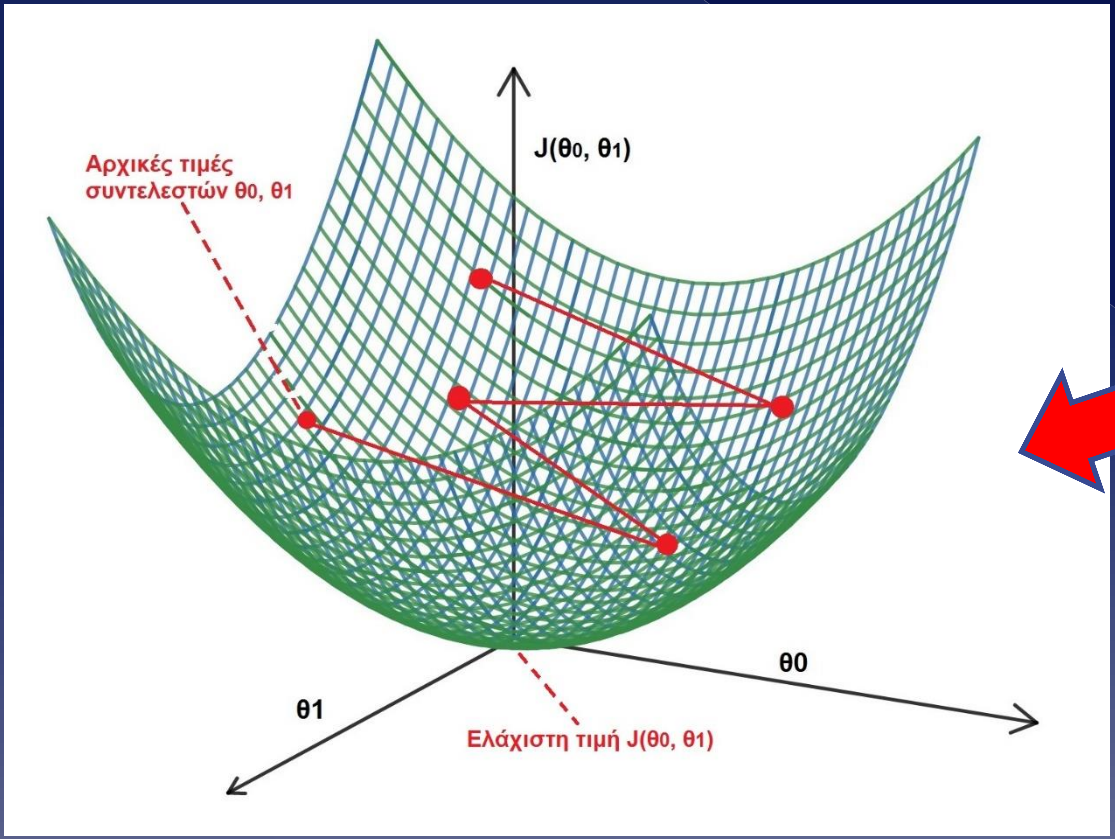
- The learning rate a
  - **Too big value** of the learning rate a



If the value of the learning rate is too big, the cost function J(θ) plotted against the number of iterations will have such a shape. The cost function increases with each iteration.

GD: Estimating parameters

- The learning rate a
  - **Too big value** of the learning rate a



If the value of the learning rate a is too big, Gradient descent may overshoot the proper values of θ that minimize the cost function.

Overshooting happens because the value of a is too big and hence the update $\theta_j = \theta_j - a \frac{\partial J(\theta)}{\partial j}$ the new values of θ_j may increase by too much, missing the values for which J(θ) is minimized. Gradient descent then diverges from the proper values of θs.

GD: Estimating parameters

- The learning rate a
  - Appropriate values for learning rate?
    - Typical values of the **learning rate a are 0.001, 0.01, 0.1**
    - Execute Gradient descent **with such values of the learning rate a** and **plot the cost function J(θ)** as a function of the number of iterations. Compare the shape of the plot with the plots shown previously.
    - If learning rate is too small, increase it by some amount e.g. from 0.01 to 0.03. Execute Gradient descent again and plot the cost function. Stop if the plot of the cost function has the appropriate shape.

GD: Estimating parameters

- The version of Gradient descent discussed previously is the “plain vanilla” style of the algorithm also known as **“Batch Gradient Descent”**
- Two other **versions of Gradient Descent** available that improve performance dramatically in Big data settings:
  - **Stochastic gradient descent - SGD**
  - **Mini-Batch gradient descent - MBGD**

Versions of Gradient descent

- Why the need to improve the performance of Gradient descent?
  - If **number of observations in training set is large** (e.g. **100000000 observations/records** or more), there are two main concerns with Batch Gradient descent:
    - Entire **training set** must be **stored into memory (RAM)**
    - **Update formulas** must **iterate over the entire training set** to calculate on step for all parameters **in each iteration**.
    - In such settings, **Batch Gradient descent is computationally expensive!**

Versions of Gradient descent

- **Concern**: Entire training set into memory
  - Looking at the matrix form of the update formula: **Does is fit into memory?**

$$\boldsymbol{\theta := \theta - \alpha\frac{1}{m}X^{T}\,(X\theta - Y)}$$

To execute this calculation, the entire matrix of the values of the independent variables X must be loaded into the main memory (RAM). What if it does not fit into RAM? E.g. if there are 10000000 observations and 50 numeric variables, you'll need to store 10000000 * 50 = 500000000 numbers and since each number requires at least 4 bytes you need 500000000 * 4= 2000000000 bytes of data in RAM or ~1.8GB of RAM. Do you have it?

Versions of Gradient descent

- **Concern**: Iterate over the entire training set at each iteration
  - Looking at the analytic formula indicates better the problem (Note: the same argument holds for the matrix form, but it’s clearer in the analytic form of the update formula): **Can be slow in big data contexts**

If the training set has m=10000000 observations, we iterate over all 10000000 observations just make one (1) update to one (1) parameter at one (1) iteration! Considering that we have many parameters, we traverse the 10000000 observations many times at each iteration. This makes Batch Gradient descent slow.

$$\boldsymbol{\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) {x_j}^{(i)}}$$

Versions of Gradient descent

- Why the need to improve the performance of Gradient descent?
  - The solution in such big data environments is simply **not to iterate over the entire training set at each iteration!**
  - The two other versions of Batch Gradient descent treat/scan the training set differently and address the above concerns

Versions of Gradient descent

- Stochastic Gradient Descent - SGD
  - At **each iteration, SGD uses only one observation** of the training set to update the parameters (instead of the entire training set in GD)

```
Initialize all parameters $\theta_j$ with random values
Initialize costVector # We will store the value of the cost function J(θ) for each iteration here
α = 0.01 # Setting the learning rate
Randomly shuffle the training set # To ensure that the observations do not have some kind of order
while termination criteria not met{
     Calculate cost function and store its value in costVector
     for each observation i in training set {
       for each parameter $\theta_j$ {
           Set new value of parameter $new_{\theta_j} := \theta_j - \alpha\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$
      }
      calculate value of cost function J(θ) for the newly estimated values of θ
      Store value of cost function into vector costVector
    }
     $\theta_j$ := $new_{\theta j}$
     n = n + 1    # Next iteration
}

# Vector θ will contain the estimated parameters which minimize J(θ)
print $\theta_j$
plot costVector
```

Versions of Gradient descent

- Pros/Cons of SGD
  - Pros
    - It's a so-called **online algorithm** – you **see the update of parameters immediately**, in a sequential fashion, during their estimation i.e. in real time. That's not possible with Batch GD
    - Does **not require entire training set in memory**
    - **Avoids local minima of J(θ)**
  - Cons
    - Can be **noisy** i.e. parameters jump around at each epoch with greater variance between epochs (epoch = one update of all parameters)

Versions of Gradient descent

- Mini-Batch Gradient Descent - MBGD
  - **MBGD** does not use one single observation of the training set to update the parameters. **It uses a "small batch"** of training set observations – typically between 2 and 100 observation in each batch.
    - To do this, we cut the large training set into smaller training subsets, and use these to update the parameters at each step
    - It's a method **"between" the extremes** of Batch Gradient descent (which uses entire training set for each parameter update) and Stochastic Gradient descent which uses only one observation to update the parameters.
    - Can be used when the dataset does not fit into computer's memory

Versions of Gradient descent

Mini-Batch Gradient descent

```
Initialize all parameters $\theta_j$ with random values
Initialize costVector # We will store the value of the cost function J(θ) for each iteration here
α = 0.01 # Setting the learning rate
Randomly shuffle the training set # To ensure that the observations do not have some kind of order
Cut the training set into batches/subsets $b_i$ each of size $n_b$ such that $b_i$*$n_b$ =m # Note: last batch might be smaller than $n_b$
while termination criteria not met{
     Calculate cost function and store its value in costVector
     for each batch $b_i$ {
       for each parameter $\theta_j$ {
           Set new value of parameter $new_{\theta_j} := \theta_j - \alpha \sum_{\iota=1}^{n_b}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$
      }
      calculate value of cost function J(θ) for the newly estimated values of θ
      Store value of cost function into vector costVector
    }
     $\theta_j$ := $new_{\Theta j}$ # update all parameters with new values
     n = n + 1   # Next iteration
}

# Vector θ will contain the estimated parameters which minimize J(θ)
print print $\theta_j$
plot costVector
```

OLS vs Gradient descent

OLS	Gradient descent
Estimates the **same, unbiased parameters** for the same training set if the linear regression assumptions hold (No or little multicollinearity, No of auto-correlation of residuals, Homoscedasticity etc)	Estimates of **parameters are** approximations and biased. May result in different parameter estimates for the same training set and regression model.
Computationally **expensive in big data contexts**.	**Suitable in big data contexts** where number of variables and number of observation are very large
Can be used to estimate parameters **only for linear regression models**	Can be **used (and is used) to estimate parameters in nonlinear regression models.**
Offers **closed formulas (the normal equation)** for calculating the parameters	Does **not offer closed formula**. Parameter estimates are iteratively calculated
Requires **entire training set in RAM**	Versions of Gradient descent **do not require entire training set in RAM** (e.g. Stochastic Gradient Descent, Mini-Batch Gradient Descent)
Taught and used mainly in **social sciences** to explain variance of dependent variable.	Taught and used mainly in **computer science and engineering** to predict the dependent variable.

Evaluating regression models

Evaluating regression models

- Evaluation?
  - Checking to see if the estimated model is justified and useful i.e. achieves its goal.
- Evaluation method depends strongly on the regression model's goal
  - Whether the goal is to explain the variance of the dependent variable
  - Whether the goal is to predict the value of the dependent variable

Explaining variance

- When the model's goal is to explain the variance of the dependent variable
  - The aim is to exactly determine how each individual independent variable influences the dependent.
    - Out of such observations you may validate or inform existing theories.
    - You don't want to mask any influence (capture their pure effect) and make sure that linearity is the proper way to capture the relationship between the variables.

Explaining variance

- Main tests you have to do for a linear regression model, if your goal is to explain the variance (many checks, but 4 most important):
  - Check the linearity hypothesis
  - Check the homoscedasticity of the residuals
  - Check if the residuals are normally distributed
  - Check (multi) collinearity of the independent variables
    - You don't want multicollinearity!

Predicting

- When the **model's goal is to predict** with high accuracy the value of the dependent variable
  - The aim is to see how well the estimated model performs on **new/unknown/unseen data** i.e. data that is **not in the training set**
    - When the aim is prediction, you don't care **how exactly** each individual independent variable influences the dependent.
      - You care about the accuracy of the prediction, not how each variable influences the dependent

Predicting

- Also, you don't really care which variable to include into the model.
  - If it improves prediction it's a good addition. You don't need to justify inclusion of independent variables because you don't care about a theory.
    - In general, adding variables improve prediction
    - **There are problems however** when adding variables that need to be addressed

Predicting

- How to evaluate a model when the goal it prediction?
  - You compare the model's predicted values of the dependent variable for unknown data with the actual, observed, real value of the dependent variable for unknown data. Their difference is the prediction error
    - Large prediction error => **bad**
    - Small prediction error => **good**
  - The error can be estimated with error metrics
    - There are many of them, each one appropriate for specific situations i.e. emphasizes different aspects

Predicting

- Most common error metrics used to evaluate prediction accuracy (n=number of observations, y_i real observation of dependent var, $\hat{y}_i$= predicted value of dependent variable)
  - Mean Absolute Error (MAE)
    - $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
  - Mean Squared Error (MSE)
    - $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
  - Root Mean Squared Error (RMSE) – **most popular**
    - $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$
  - Mean Absolute Percentage Error (MAPE)
    - $MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$

Predicting

- How to calculate prediction error of a linear regression model?
- Where to get the data from and check accuracy against?
  - In general, two approaches to do this
    - **In-Sample-Testing**: get a subset from the data that was used to estimate the parameters and check prediction errors. The error calculated from such dataset is called training error
      - The idea here is that the calculated training error is a good approximation of the error on unknown data.
    - **Out-of-Sample-Testing:** get a different, new, unseen subset of data –not in the training set- to check prediction error. Error calculated in such way is called generalization error.

Predicting

- **Out-of-Sample-Testing**
  - How/where to get new, unseen data?
    - 1) Collect new data from the source again i.e. apart from training data
      - Not always possible, costly, time consuming
    - 2) Divide the existing dataset randomly into two non overlapping subsets:
      - One for estimating the parameters thus becoming the **training set**
      - One for evaluating the prediction error called the **testing set/test set.** The test set is not used for estimating the parameters: this set is excluded and hence functions as an new, unseen data set.
      - Such way of testing/evaluating a model is called **cross validation**.

Predicting

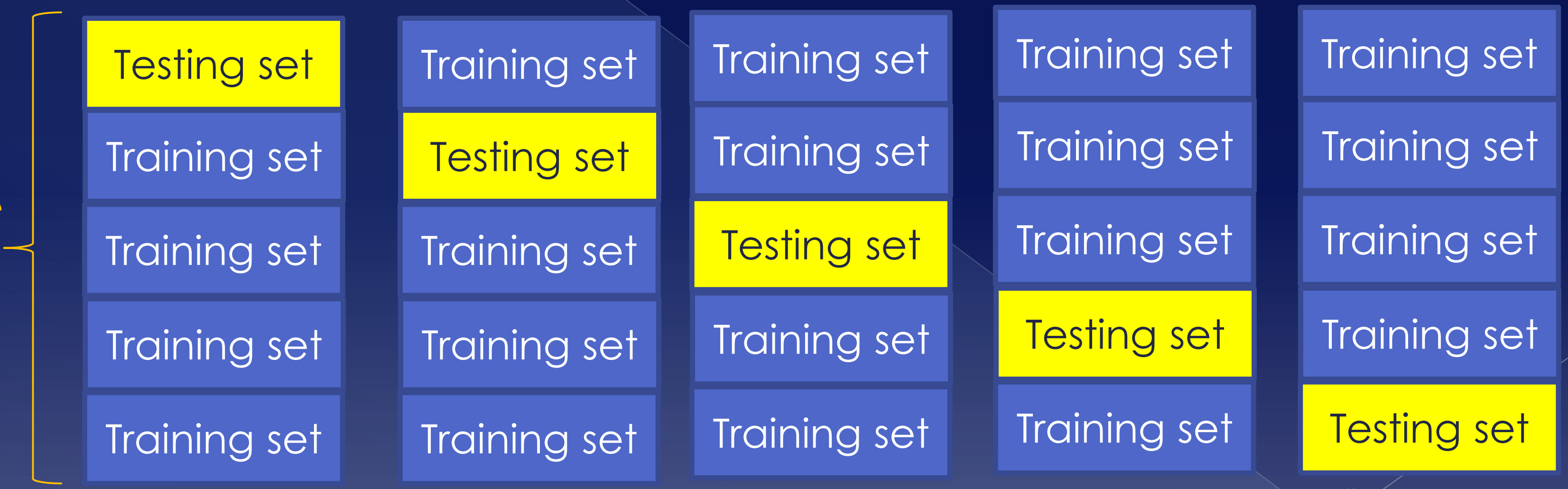
- **Cross validation**
  - Different versions available to get better estimate of the error
    - 1) **Holdout method:** Divide (once) randomly initial dataset into only two non-overlapping data subsets: one for training (training set) and one for testing (testing set)
      - In general **70-80%** of the initial dataset will be the training set, **30-20%** of the initial dataset will be the **testing set**.
      - Use the training set to estimate parameters
      - Use the testing set (which is a new/unseen dataset) to calculate error which is considered an approximation for the generalization error.

Predicting

- **Cross validation**
  - Different versions available to get better estimate of the error
    - 2) **k-Fold Cross validation method:** Divide the initial dataset into k non-overlapping subsets with approximately equal number of observations - not only into two as in the holdout method. k (number of subsets) is given by user (usually k=5 or k=10 or even greater).
    - Then, iterate over each of these k subsets, and use each one of the subsets as the testing set and the remaining k-1 subsets as the training set (merging them into one set). Process terminates when each one of the k subsets has been used as testing set.
    - Calculate error on each testing set. In total k errors will result out of which the average can be calculated. Much better estimate of the generalization error!

Predicting

- Why K-fold cross validation?
  - gives a much **better estimate for the generalization error**.



Cut original dataset into k approximately equally sized subsets. Here k=5

Testing set	Training set	Training set	Training set	Training set
Training set	Testing set	Training set	Training set	Training set
Training set	Training set	Testing set	Training set	Training set
Training set	Training set	Training set	Testing set	Training set
Training set	Training set	Training set	Training set	Testing set

1st iteration: use first subset as testing set and remaining as training set. Calculate error.

2nd iteration: use second subset as testing set and remaining as training set. Calculate error.

3rd iteration: use third subset as testing set and remaining as training set. Calculate error.

4th iteration: use fourth subset as testing set and remaining as training set. Calculate error.

5th iteration: use fifth subset as testing set and remaining as training set. Calculate error.

Predicting

- **Cross validation**
  - Used also in comparing the **accuracy of different linear regression models** to predict the same dependent variable.
    - E.g. which one of the following different linear regression models better predicts the mpg of cars?

$$mpg = \beta_1 Horsepower + \beta_2 Weight + \beta_0$$
$$mpg = \beta_1 Horsepower + \beta_2 Weight + \beta_3 \mathrm{Displacement} + \beta_0$$
$$mpg = \beta_1 Horsepower + \beta_2 Weight + \beta_3 Weight^2 + \beta_4 \mathrm{Displacement} + \beta_0$$

- Do a **k-fold cross validation** on the same dataset for each model. Compare errors.
  - Model with smallest error chosen

Predicting

- K-Fold Cross Validation in R
  - See file **R-k-FoldCrossValidation.rar** on eclass

Overfitting

- When the goal is prediction, the aim of a regression model is to have small generalization errors
  - Yet, real value of generalization error is never known – only estimations possible by using k-fold cross validation
- From the training error (which is always known), it’s not possible to get a reliable estimate of the generalization error
  - It’s a poor estimate for the generalization error

Overfitting

- However, from the relationship between the training error and generalization error (resulting from k-fold cross validation) useful conclusions about the regression model can be drawn
  - In particular these situations
    - Training error and generalization error small
    - Training error large, and generalization error large
    - Training error small and generalization error large

Overfitting

- Training error and generalization error small
  - Regression model fits the training data well. The ideal situation
- Training error and generalization error large
  - In this case the regression model is **underfitting the training data and the situation is called underfitting**: the model does not succeed in fitting the training data well.
  - Happens when the regression model is too simple i.e. has few independent variables.
  - Such models do not have the flexibility to adapt to the changes and variance of the dependent variable and hence has a poor fit to the data – hence the large training and generalization errors.

Overfitting

- Training error (very) small and generalization error large
  - In this case the regression model is **overfitting the training data and the situation is called overfitting**.
  - This happens when the regression model **fits the training data too well** – the model captures not only the (true) relationships between the variables in the training set but also random noise and fluctuations of training data
    - Hence, it is very good in predicting the value of the dependent variable for data in the training set (i.e. very small training error) but very bad in predicting the value of the dep. variable for new/unknown data (large generalization error).

Overfitting

- Overfitting is a generally associated with more complex regression models
  - A regression model is called complicated, the more independent variables it contains and higher degree of monomials.

$$mpg = \beta_1 Horsepower + \beta_2 Weight + \beta_0$$

Simpler regression model

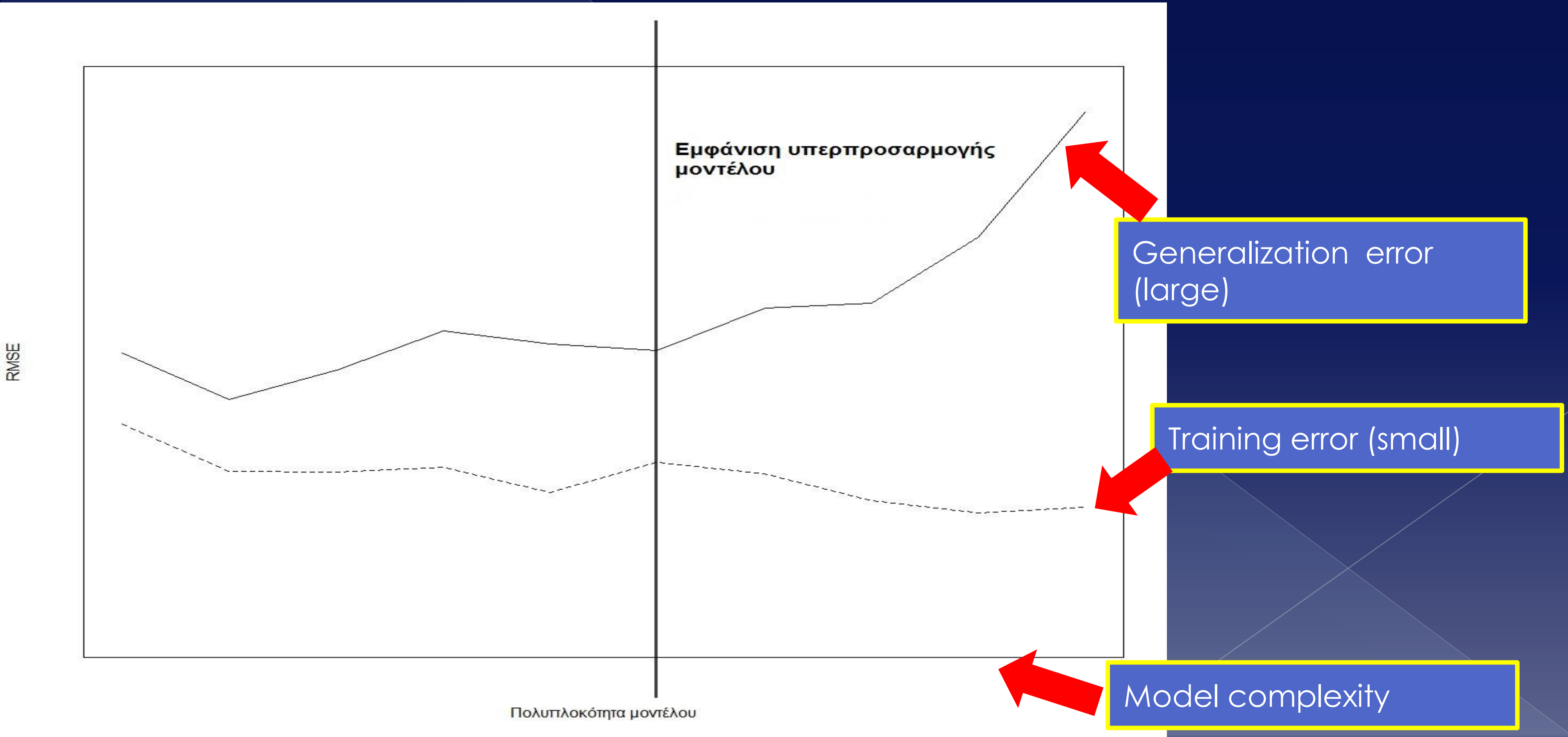
$$mpg = \beta_1 Horsepower + \beta_2 Weight + \beta_3 Displacement + \beta_0$$

$$mpg = \beta_1 Horsepower + \beta_2 Weight + \beta_3 Weight^2 + \beta_0$$

More complicated regression models than above: more independent variables and/or higher degree of monomials

Overfitting

- The more complex the regression model the more overfitting becomes a problem



Generalization error (large)

Training error (small)

Model complexity

Overfitting

- How to check if a model is prone to overfitting?
  - **Calculate training error** of model using in-sample-testing: take a subset of the training data – used to estimate the model's parameter- and calculate training error
  - **Calculate generalization error** using k-Fold Cross Validation
  - Compare these two errors (training, generalization)

Overfitting

- How to address a model that exhibits overfitting?
  - Two approaches
    - **Make the regression model simpler** by removing independent variables or reducing the degree of monomials
      - Not always easy: which variables to remove, model needs to be re-evaluated etc.
    - **Use regularization.**
      - What is regularization? A way/technique to overcome the problems of overfitting by limiting the size of the coefficients.

Overfitting

- Regularization
  - Two approaches
    - **Make the regression model simpler** by removing independent variables or reducing the degree of monomials
      - Not always easy: which variables to remove, model needs to be re-evaluated etc.
    - **Use regularization.**
      - What is regularization? A way/technique to overcome the problems of overfitting by limiting the size of the coefficients.

Overfitting

- Regularization
  - Size of coefficients
    - In general, overfitted regression models **are not biased**, but have a **huge variance** of the predicted dependent variable. The bias-variance tradeoff in predictive modeling!

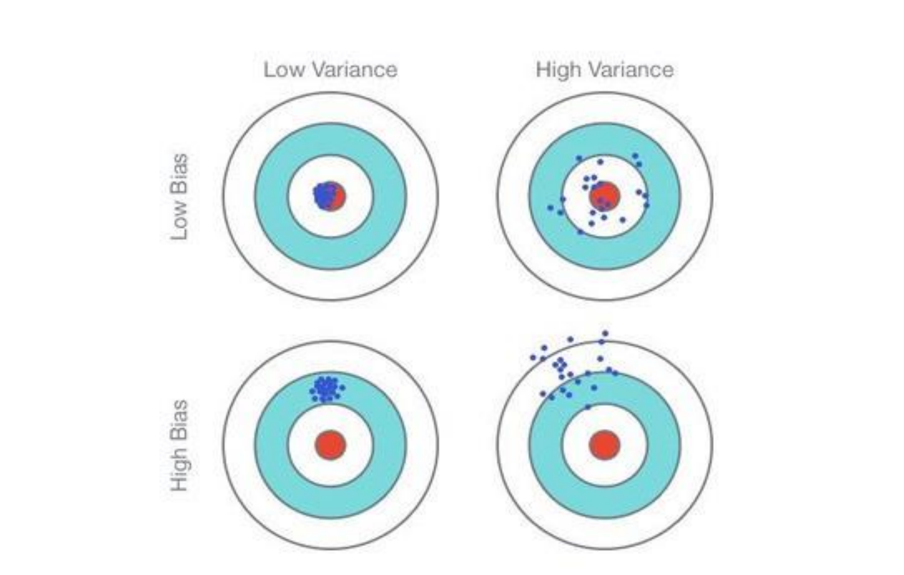


Fig. 1: Graphical Illustration of bias-variance trade-off , Source: Scott Fortmann-Roe., Understanding Bias-Variance Trade-off

Bias-variance tradeoff in predictive models: models having a low bias have large variance of the predicted value. Models with a low variance of the predicted value have large bias

Overfitting

- Regularization
  - Size of coefficients
    - This high variance of the predicted value is caused by the size of the estimated coefficients: they are large. As a result, even small changes of the dependent variable results in large variances of the dependent variable due to the large coefficients
  - Since bias in an overfitted regression model is low, high variance due to the large coefficients can be addressed by limiting their size
    - I.e. making them technically smaller.
    - This is what regularization achieves.

Overfitting

- What regularization does
  - Attempts to limit the size of coefficients in an overfitted regression model in order to control the variance of the predicted values of the dependent variable.
  - Regularization achieves this by modifying the loss function and placing also constraints on the magnitude of coefficients so that they don't grow to large values
    - I.e. loss function also constraints the magnitude of the coefficients.

Overfitting

- Modified Loss function
  - If gradient descent is the method used to estimate the coefficients, to achieve regularization an additional term is added: the regularization term:

$$J(\beta_0,\beta_1,\beta_2,\dots,\beta_k\,)=\underbrace{\frac{1}{2m}\sum_{i=1}^{m}\left(h_\beta\left(x^{(i)}\right)-\;y^{(i)}\right)^2}_{}+\lambda\underbrace{\sum_{j=1}^{n}\left|\beta_j\right|^q}_{}$$

(Traditional) Loss function of Gradient Descent

Regularization term: new term added to achieve regularization of coefficients. **λ** is a **regularization parameter** given as input. q is a parameter and determines the type of regularization

Overfitting

$$J(\beta_0,\beta_1,\beta_2,\dots,\beta_k\ ) = \underbrace{\frac{1}{2m}\sum_{i=1}^{m}\left(h_\beta\left(x^{(i)}\right) - \ y^{(i)}\right)^2}_{\text{(Traditional) Loss function of Gradient Descent}} + \underbrace{\lambda\sum_{j=1}^{n}\left|\beta_j\right|^q}_{\textbf{Regularization term}\text{: new term added to achieve regularization of coefficients}}$$

- Regularization term: why it works
  - The term added is the sum of estimated coefficients. Minimizing this new loss function means minimizing both terms i.e. minimize $\lambda\ \sum_{j=1}^{n}\left|\beta_j\right|^q$ also! That means that constraints on their size are placed and smaller values are preferred.

Overfitting

- q: parameter that determines the type of regularization
  - If q=1 the loss function becomes

$$\frac{1}{2m}\sum_{i=1}^{m}\left(h_\beta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{n}\left|\beta_j\right|$$

  - and the regression is called Lasso or L1 regression.
  - Lasso regression not only limits the size of coefficients but can also be used for feature selection
    - For large values of λ, Lasso regression will result in coefficients to become 0. This means that the respective independent variables not statistically significant and hence can be dropped (= feature/variable selection)!

Overfitting

- q: parameter that determines the type of regularization
  - If q=2 the loss function becomes

$$\frac{1}{2m}\sum_{i=1}^{m}\left(h_\beta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{n}\left|\beta_j\right|^2$$

  - and the regression is called Ridge or L2 regression or Thikonov regularization.
  - This is the most popular form of regularization with good results. Coefficients get smaller but not zero hence Ridge regression cannot be used for feature selection.

Overfitting

- q: parameter that determines the type of regularization
  - Third option also available called Elasticnet
    - Adds two terms to the Loss function, one similar to Lasso regression and one similar to Ridge regression with weights.

Overfitting

- λ: regularization parameter
  - Given as input to the regression analysis i.e. must be determined beforehand by the user
  - **If λ=0**, regularization term cancels out and the loss function takes its traditional form (i.e. without regularization)
  - The proper value for λ can be determined by doing k-Fold Cross Validation for different values of λ and selecting the most appropriate one based on the prediction error.
    - Typical values for λ are: 0.01, 0.02… 0.4… 0.7…

Overfitting

- Regularization in R
  - When using Gradient Descent, two things must change the cost function changes (here Ridge regression):
    - The partial derivative which becomes

$$\frac{\partial}{\partial \theta_i} J(\theta) = \frac{1}{m} \left[\sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right) x_j^{(i)} + \lambda \Theta_j \right]$$

    - The Loss/cost function which becomes:

```
calculateCostRidge<-function(X, y, theta, lambda=0){
  # Number of observations
  m <- length(y)

  return( sum((X%*%theta- y)^2) / (2*m) + lambda*sum(theta^2))

} # calculateCostRidge
```

Overfitting

- Regularization
  - Regularization does not apply only to Gradient Descent (here just used as an example)
  - Can also do regularization with any method of estimating coefficients for a regression model
    - E.g. OLS with regularization is possible.

References

- Waugh, F. V.: Choice of the Dependent Variable in Regression Analysis, Journal of the American Statistical Association, Vol. 38, No. 222, June., 1943, pp. 210-216
- Shmueli, G.: To Explain or to Predict? Statistical Science 25(3), January 2011
- Peng, R. D. *R Programming for Data Science*. Lean Publishing. Ανακτήθηκε στις 19 Νοεμβρίου 2018, από: https://leanpub.com/rprogramming
- Draper, N. R. and Smith, H.: Applied Regression Analysis, Wiley-Interscience; Third edition, 1998
- Brian, C.: *Regression Models for Data Science in R*. Lean Publishing, 2015, Ανακτήθηκε στις Ιούνιο 2019, από: https://leanpub.com/regmods
- Nilsson, N. J.: *Introduction to Machine Learning*, 1998, Ανακτήθηκε στις Ιούνιο 2019, από: http://robotics.stanford.edu/~nilsson/MLBOOK.pdf
- Wikipedia, *Classification*. Ανακτήθηκε Ιούνιο 2019, από: https://en.wikipedia.org/wiki/Classification
- Yanchang, Z.: *Regression and Classification with R*, 2015, Ανακτήθηκε στις Ιούλιο 2018, από: http://www.rdatamining.com/docs/regression-and-classification-with-r

- Kiefer, J. and Wolfowitz, J.: Stochastic Estimation of the Maximum of a Regression Function, Annals of Mathematical Statistics, Volume 23, Number 3, 1952, pp 462-466.
- Bottou, L., Curtis, F. E., Nocedal, J.: Optimization Methods for Large-Scale Machine Learning, SIAM Review, Volume 60 (2), 2018. Διαθέσιμο από http://arxiv.org/abs/1606.04838
- Tibshirani, R.: Regression Shrinkage and Selection via the Lasso, Journal Royal Statistical Society, Volume 58 (1), 1996, pp267-288
- Hoerl, A.E. and Kennard, R.: Ridge regression: Biased estimation for non orthogonal problems. Technometrics, 12, 1970, pp55-67
- Zou, H. and Hastie, T.: Regularization and variable selection via the elastic net, Journal of the Royal Statistical Society, Series B.67, 2005, pp. 301–320.
- Stanton, J. M.: Galton, Pearson, and the Peas: A Brief History of Linear Regression for Statistics Instructors, Journal of Statistics Education, Volume 9, Issue 3, 2001
- Poole, M. A. and O'Farrell, P. N.: The Assumptions of the Linear Regression Model, Transactions of the Institute of British Geographers, No. 52, 1971, pp. 145-158 .

Appendices

Deriving update formulas for parameters in linear regression models in Gradient Descent

- We will derive as an example the update for parameter θ_1 (parameter for an independent variable) - the same analysis holds for all other parameters θ_j

$$\frac{\partial J(\theta)}{\partial \theta_1} = \frac{\partial \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2}{\partial \theta_1} = \frac{\frac{1}{2m} \partial \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right)^2 - 2h_\theta\left(x^{(i)}\right)y^{(i)} + {y^{(i)}}^2\right)}{\partial \theta_1}$$

$$= \frac{1}{2m}\left[\frac{\partial \sum_{i=1}^{m} h_\theta\left(x^{(i)}\right)^2}{\partial \theta_1} - 2\frac{\partial \sum_{\iota=1}^{\mu} h_\theta\left(x^{(i)}\right)y^{(i)}}{\partial \theta_1} + \frac{\partial \sum_{\iota=1}^{\mu} {y^{(i)}}^2}{\partial \theta_1}\right] = < \textit{see next slide} >$$

Deriving update formulas for parameters in linear regression models in Gradient Descent

$$= \frac{1}{2m}\left[\frac{\partial \sum_{i=1}^{m} h_\theta\left(x^{(i)}\right)^2}{\partial \theta_1} - 2\frac{\partial \sum_{\iota=1}^{\mu} h_\theta\left(x^{(i)}\right) y^{(i)}}{\partial \theta_1}\right]$$

$$= \frac{1}{2m}\left[2\sum_{i=1}^{m} h_\theta\left(x^{(i)}\right)\frac{\partial h_\theta\left(x^{(i)}\right)}{\partial \theta_1} - 2\frac{\partial \sum_{\iota=1}^{\mu} h_\theta\left(x^{(i)}\right) y^{(i)}}{\partial \theta_1}\right]$$

$$= \frac{1}{2m}\left[2\sum_{i=1}^{m} h_\theta\left(x^{(i)}\right) x_1^{(i)} - 2\sum_{i=1}^{m} x_1^{(i)} y^{(i)}\right] = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_1^{(i)}$$

Thus the update formula for parameter θ1 becomes thus:

$$\boxed{\theta_1 := \theta_1 - \alpha\,\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_1{}^{(i)}}$$

Now do the same for all other parameters θ_0, θ_2, θ_3,…. and from this we get the closed form formulas for the updates of all parameters θ.

Batch Gradient Descent vs Stochastic Gradient Descent vs Mini-Batch Gradient Descent: An explanation of how they differ in updating the θs

Batch Gradient Descent vs Stochastic Gradient Descent vs Mini-Batch Gradient Descent

- It’s all in how the training set is scanned/traversed
  - Assume that training set has **10000000 observations** i.e. **m=10000000**. i.e. training set very large.

```
# How θs are updated in Batch Gradient descent
while termination conditions not met {
      update each θs (i.e. coefficients) as follows:

                $\theta_0 := \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)$

                $\theta_1 := \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right) {x_1}^{(i)}$

                $\theta_1 := \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right) {x_1}^{(i)}$

                … <do this for all thetas>
      calculate value of cost function J(θ) for the newly calculated values of θs
      Store value of cost function into vector costVector
}
```

Batch Gradient Descent

The problem is in this calculation of the sum: **For each coefficient θj** we iterate through m=10000000 observations to make one small adjustment of each θj. I.e. iterate through the same 10000000 observation for updating each and every θj. And this happens only during one iteration (see outer while termination condition not met). This is slow and inefficient for large datasets

Batch Gradient Descent vs Stochastic Gradient Descent vs Mini-Batch Gradient Descent

- It's all in how the training set is scanned/traversed
  - Assume that training set has 10000000 observations i.e. m=10000000. i.e. training set very large.

```
while termination criteria not met{
     Calculate cost function and store its value in costVector
     for each observation i in training set (i.e. for each 10000000 obs) {
       for each parameter θ_j {
           Set new value of parameter $new_{\theta_j} := \theta_j - \alpha\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$
      }
      calculate value of cost function J(θ)
      Store value of cost function into vector costVector
    }
     θ_j := new_θj
     n = n + 1    # Next iteration
}
```

Stochastic Gradient Descent (SDC)

Look at the update function of the θs: **there is no sum** – compare this to the update function in Batch Gradient Descent. Hence in SDG we don't iterate over 10000000 to make one update. In Stochastic Gradient Descent only one observation is used at any step not all 10000000. However, Stochastic Gradient Descent iterates through all 10000000 only once during each cycle (outer while)– see "for each 10000000 obs in training set"

Batch Gradient Descent vs Stochastic Gradient Descent vs Mini-Batch Gradient Descent

- It's all in how the training set is scanned/traversed
  - Assume that training set has 10000000 observations i.e. m=10000000. i.e. training set very large.

```
Cut the training set into batches/subsets $b_i$ each of size $n_b$ such that $b_i*n_b$ =m
while termination criteria not met{
     Calculate cost function and store its value in costVector
     for each batch $b_i$ {
       for each parameter $θ_j$ {
             Set new value of parameter $new_{\theta j} := \theta_j - \alpha \sum_{i=1}^{n_b}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$
      }
      calculate value of cost function J(θ) for the newly estimated values of θ
      Store value of cost function into vector costVector
    }
     $θ_j$ := $new_{θj}$ # update all parameters with new values
     n = n + 1   # Next iteration
}
```

Mini-Batch Gradient Descent

Look at the update function of the θs: **There is a sum but it does not iterate over all 10000000 observations** –it iterates over a lot smaller number of observations n_b because the training set of 10000000 has been cut into batches of smaller size (e.g. 3000 observations per batch). Mini-batch Gradient Descent will process all batches and hence will iterate through all 10000000 observations, but not all at once and for a single update.