Linear and Logistic Regression

(the SQL way)

What is the purpose of this presentation?

- To show if linear or logistic regression is possible with SQL and to provide their implementation
- To provide enough arguments whether an SQL implementation of these regressions is worth or not
- When is it worth to perform any numerical calculations on the database side.

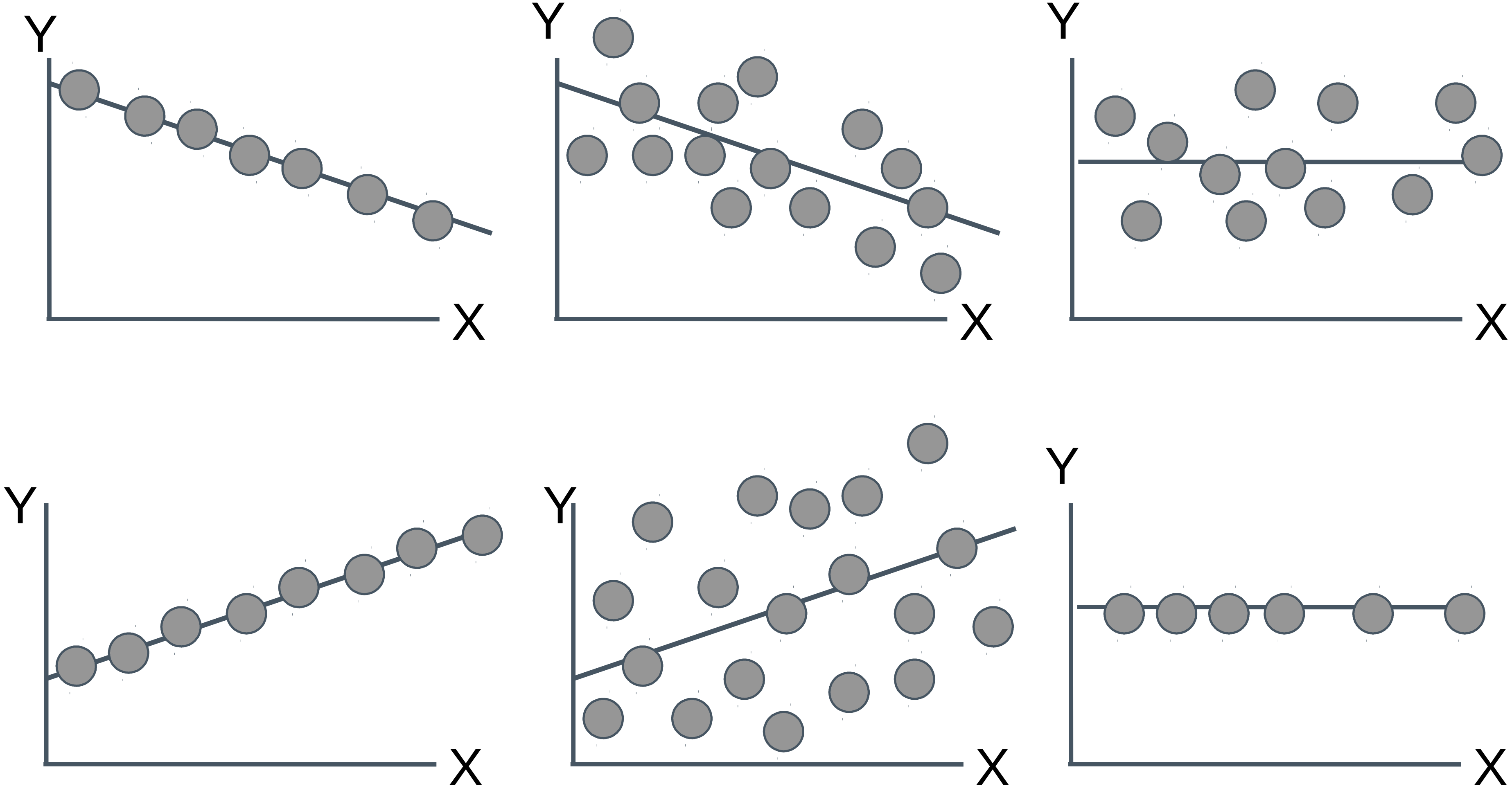
Presentation Structure

1. Linear Regression
   - What is linear regression ?
   - Use cases
   - Solving for coefficients in SQL (with demo)
2. Logistic Regression
   - What is logistic regression ?
   - Use cases
   - Logistic Regression vs Linear Regression comparison
   - Gradient Descent
   - Solving for coefficients in C++ demo
3. Discussion whether SQL implementation of the above is worth it or not?

What is Simple Linear Regression ?

- **Simple linear regression** is a statistical method that allows us to summarize and study relationships between two continuous (quantitative) variables:
- One variable, denoted *x*, is regarded as the **predictor**, **explanatory**, or **independent** variable.
- The other variable, denoted *y*, is regarded as the **response**, **outcome**, or **dependent** variable.

Scatter Plots of Data with Various Correlation Coefficients



Why do we need Linear Regression ?

- What dose is required for a particular treatment ?
- How many days will take for a full recovery ?
- How many cars will be parked here tomorrow ?
- How many insurances will be sold next month ?

Simple Linear Regression

- Given the training set $\{y_i, x_i\}_{i=1}^{n}$ we try to find the "best" coefficients which would give a $\hat{y}_i \approx y_i$
- $y_i = \beta_0 + \beta_1 \mathrm{x_i}$ where $i = 1 \ldots n$

Simple linear regression (**SLR**)

- There are multiple ways in which we could obtain the coefficients :
  - **Ordinary least squares** (OLS) : conceptually the simplest and computationally straightforward (advantageous for SQL)
  - **Generalized least squares** (GLS)
  - **Iteratively reweighted least squares** (IRWLS)
  - **Percentage least squares** (PLS)

- There are multiple ways in which we could obtain the β coefficients :
  - **Ordinary least squares** (OLS) : conceptually the simplest and computationally straightforward (advantageous for SQL)
  - **Generalized least squares** (GLS)
  - **Iteratively reweighted least squares** (IRWLS)
  - **Percentage least squares** (PLS)

SLR - OLS Minimize $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

SLR - OLS Minimize

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 \mathrm{x_i}))^2$$

= 0

$$\frac{d}{d\beta_0}\sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 \mathrm{x_i}))^2 = 0$$

$$\beta_0 = \frac{\left(\sum_{i=1}^{n} y_i - \beta_1 \sum_{i=1}^{n} x_i\right)}{n}$$

$$\boldsymbol{\beta_0 = \bar{y} - \beta_1 \bar{x}}$$

SLR - OLS Minimize $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

SLR - OLS Minimize

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 \mathrm{x_i}))^2$$

= 0

$$\frac{d}{d\beta_1}\sum_{i=1}^{n}(y_i - (\bar{y} - \beta_1\bar{x} + \beta_1 \mathrm{x_i}))^2 = 0$$

= 0

$$\frac{d}{d\beta_1}\sum_{i=1}^{n}(y_i - (\bar{y} + \beta_1(x_i - \bar{x})))^2 = 0$$

= 0

$$\sum_{i=1}^{n}(y_i - \bar{y})(\mathrm{x_i} - \bar{x}) + \beta_1(x_i - \bar{x})^2 = 0$$

$$\boldsymbol{\beta_1} = \frac{\sum_{i=1}^{n}(\boldsymbol{y_i} - \overline{\boldsymbol{y}})(\boldsymbol{x_i} - \overline{\boldsymbol{x}})}{\sum_{i=1}^{n}(\boldsymbol{x_i} - \overline{\boldsymbol{x}})^2}$$

SLR

- With only 2 parameters to estimate : β_0, β_1 computationally it is not a challenge for any DBMS
- But increasing the #parameters it will slow by noticeable constant factor if we solve all β in our ”scalar” fashion.
- However we could use a some matrix tricks to solve for any size of β.

Multilinear Linear Regression with LS

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad X = \begin{pmatrix} 1 & x_{11} & \dots & x_{1p} \\ 1 & x_{21} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \dots & x_{np} \end{pmatrix} \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix}$$

$$y = X\beta$$

$$\mathbf{LS}(\boldsymbol{\beta}) = (\boldsymbol{y} - \boldsymbol{X\beta})^{\boldsymbol{T}}(\boldsymbol{y} - \boldsymbol{X\beta})$$

Minimize

$$\text{Minimize } \mathrm{LS}(\beta) = (y - X\beta)^T (y - X\beta)$$

$$[\frac{d}{d\beta} \ \mathrm{LS}(\beta) = (y - X\beta)^T (y - X\beta)] = 0$$

$$[\frac{d}{d\beta} \ \mathrm{LS}(\beta) = -2(X)^T (y - X\beta)] = 0$$

$$X^T (y - X\beta) = 0$$

$$\widehat{\boldsymbol{\beta}} = \left(\boldsymbol{X}^T \boldsymbol{X}\right)^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

$$\widehat{\boldsymbol{y}} = \boldsymbol{X}\widehat{\boldsymbol{\beta}} = \boldsymbol{X}\left(\boldsymbol{X}^T \boldsymbol{X}\right)^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

Solving $\widehat{\boldsymbol{y}} = \boldsymbol{X}\widehat{\boldsymbol{\beta}} = \boldsymbol{X}\left(\boldsymbol{X}^{T}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{T}\boldsymbol{y}$ with QR

Solving with QR

$$\widehat{\beta} = (X^TX)^{-1}X^Ty$$

No need to compute $(X^TX)^{-1}$, *bring it to a simpler form :*

No need to compute , bring it to a simpler form :

$$X^TX\widehat{\beta} = X^Ty$$

Decompose X into $X = QR$ *where Q is orthogonal*$(Q^T = Q^{-1})$ *and R upper triangular*

where Q is orthogonal() and R upper triangular

$$(QR)^TQR\widehat{\beta} = (QR)^Ty$$
$$Q^TR^TQR\widehat{\beta} = Q^TR^Ty$$
$$QR\widehat{\beta} = y$$
$$\boldsymbol{R}\widehat{\boldsymbol{\beta}} = \boldsymbol{Q}^{T}\boldsymbol{y}$$

$$R\hat{\beta} = Q^T y$$

- At this point it is trivial to solve for $\hat{\beta}$ because R is an upper triangular matrix.

$$\begin{bmatrix} r_{1,1} & \cdots & r_{1,p} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & r_{p,p} \end{bmatrix} * \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} = \begin{bmatrix} const_0 \\ const_1 \\ \vdots \\ const_p \end{bmatrix}$$

- **QR Factorization** simplified the process but it’s still tedious (this is how the “**lm**” routine is implemented in R with C and Fortran calls underneath)

- At this point it is trivial to solve for because R is an upper triangular matrix.
- **QR Factorization** simplified the process but it’s still tedious (this is how the “**lm**” routine is implemented in R with C and Fortran calls underneath)

Problems with Multiple linear regression ?

- Operations for linear algebra must be implemented :
  - Matrix/vector multiplication
  - Matrix inverse/pseudo-inverse
  - Matrix factorization like SVD, QR, Cholesky, Gauss-Jordan
  - Too much number crunching for an engine which has different purpose, it’s far away from FORTRAN!
- Even C++ Boost’s library for basic linear algebra (BLAS) does a poor job in comparison to MATLAB.

What is Logistic Regression ?

- To predict an outcome variable that is categorical from predictor variables that are continuous and/or categorical
- Used because having a categorical outcome variable violates the assumption of linearity in normal regression
- The only “real” limitation for logistic regression is that the outcome variable must be discrete
- Logistic regression deals with this problem by using a logarithmic transformation on the outcome variable which allow us to model a nonlinear association in a linear way
- It expresses the linear regression equation in logarithmic terms (called the *logit*)

Logistic Regression Use Cases

- Google uses it to classify spam or not spam email
- Is a loan good for you or bad?
- Will my political candidate win the election?
- Will this user buy a monthly Netflix subscription?
- Will the prescribed drugs have the desired effect?
- Should this bank transaction be classified as a fraud?

Why not Linear Regression instead of Logistic Regression ?

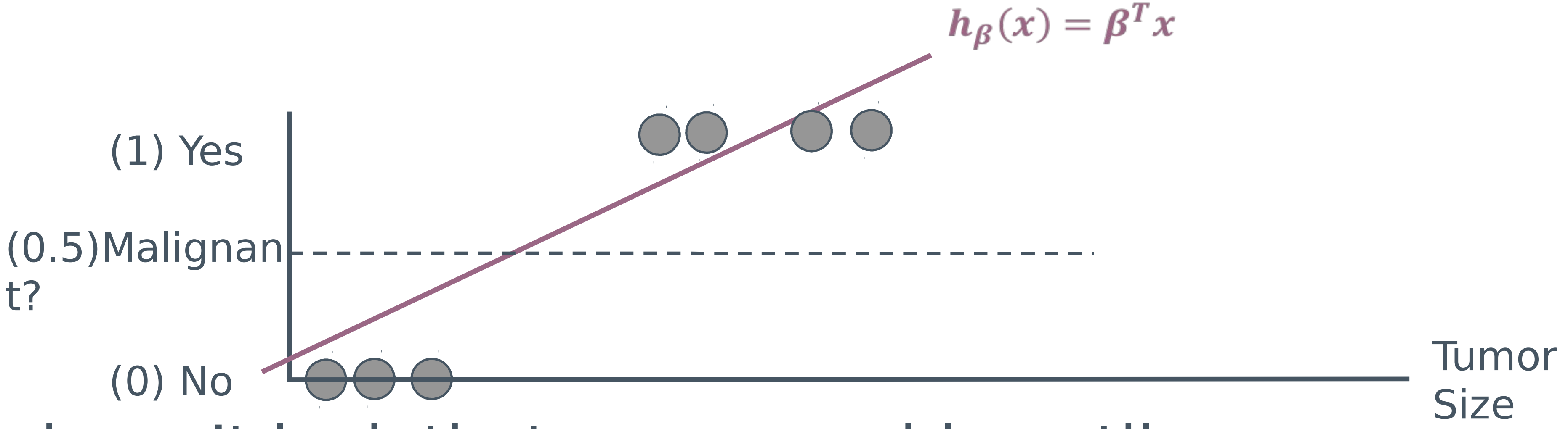
- Maybe we could solve the problem mathematically only using Linear Regression for classification ? and that will spare us a lot of complexity.
- We would like to classify our input into 2 categories : either a 0 or 1 (ex: 0 No, 1 Yes)

Why not Linear Regression instead of Logistic Regression ?

- Assume if $\begin{cases} h_\beta(x) \geq 0.5 \rightarrow y = 1 \\ h_\beta(x) < 0.5 \rightarrow y = 0 \end{cases}$

- Assume if

Real Data,
-> Training
set

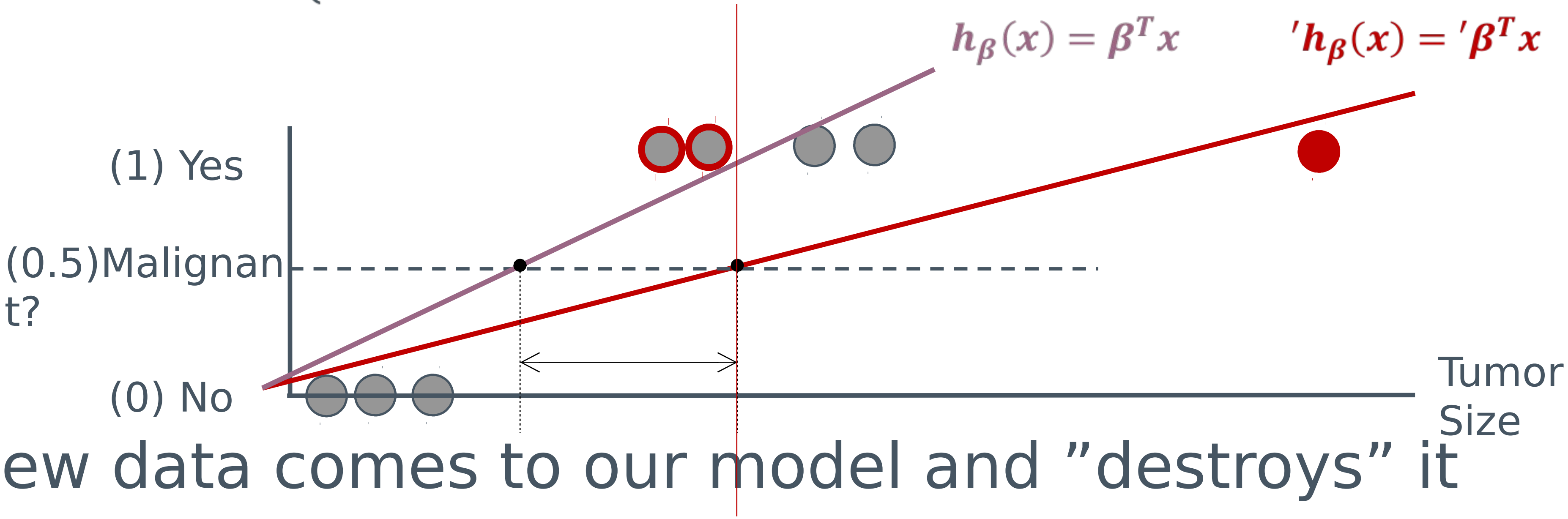


- It doesn't look that unreasonable until…

- It doesn't look that unreasonable until…

Why not Linear Regression instead of Logistic Regression ?

- Assume if $\begin{cases} h_\beta(x) \geq 0.5 \rightarrow y = 1 \\ h_\beta(x) < 0.5 \rightarrow y = 0 \end{cases}$
- Assume if

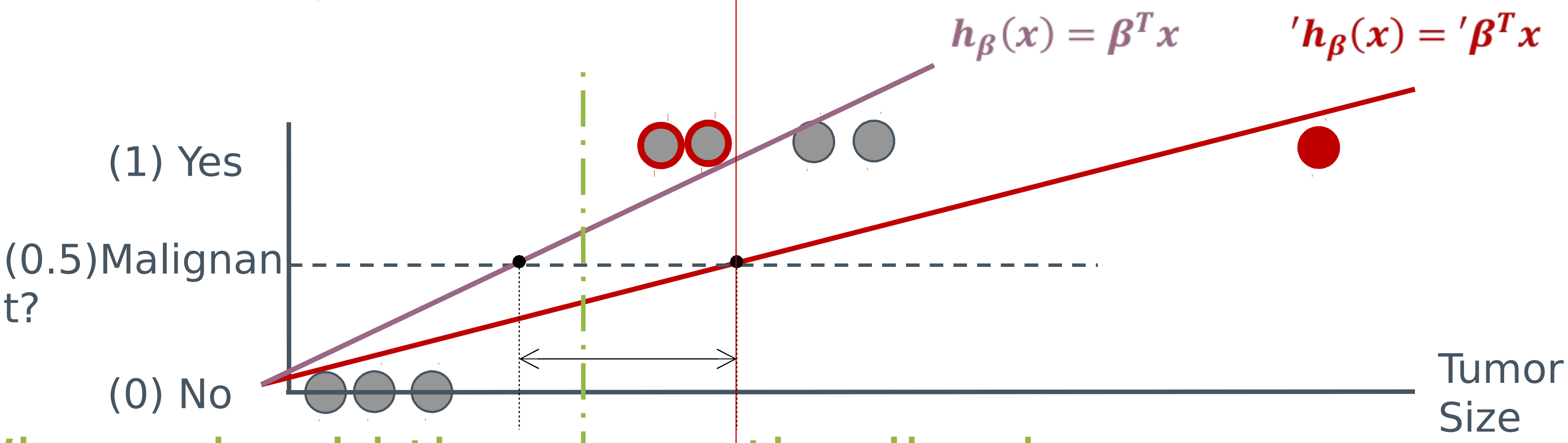


- New data comes to our model and ”destroys” it
- New data comes to our model and ”destroys” it

Why not Linear Regression instead of Logistic Regression ?

- Assume if $\begin{cases} h_\beta(x) \geq 0.5 \rightarrow y = 1 \\ h_\beta(x) < 0.5 \rightarrow y = 0 \end{cases}$

- Assume if



- Where should the separation line be

- Where should the separation line be

Why not Linear Regression instead of Logistic Regression ?

- Currently : $h_\beta(x) > 1 \; or \; h_\beta(x) < 0$
- For classification we need $y = 0 \; or \; y = 1$
- Logistic Regression : $0 \le h_\beta(x) \le 1$

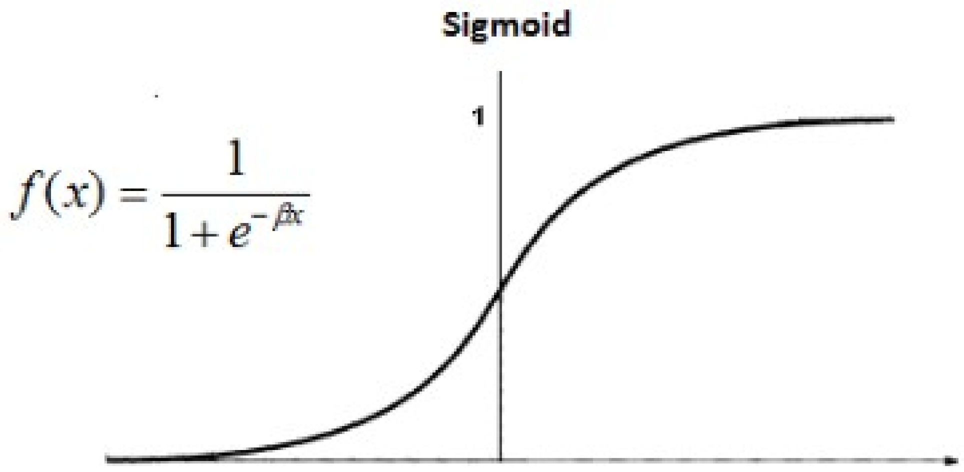
- Let's create a new function which will satisfy our conditions.
- $h_\beta(x) = \beta^T x$ wrap it to $h_\beta(x) = \boldsymbol{g}(\beta^T x)$

Logistic Regression

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$h_\beta(x) = \frac{1}{1 + e^{-x}}$$

$$h_\beta(x) = \frac{1}{1 + e^{-\beta^T x}} = \mathrm{P}(\mathrm{y} = 1 | x; \beta)$$



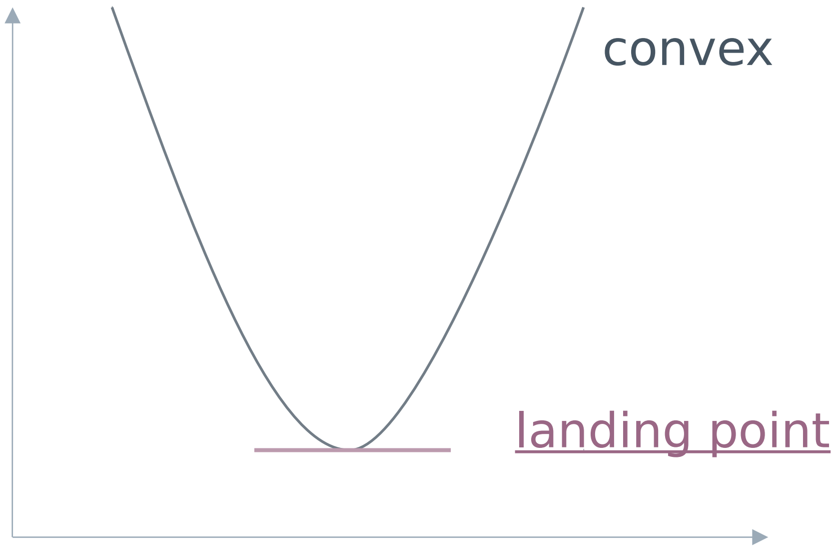
$$\mathrm{P}(\mathrm{y} = 1 | x; \beta) + \mathrm{P}(\mathrm{y} = 0 | x; \beta) = 1$$

Gradient Descent

- A technique to find minimum of a function
- With other words “for which input parameters of the function will I get a minimum output”
- Not the best algorithm but computationally and conceptually simple

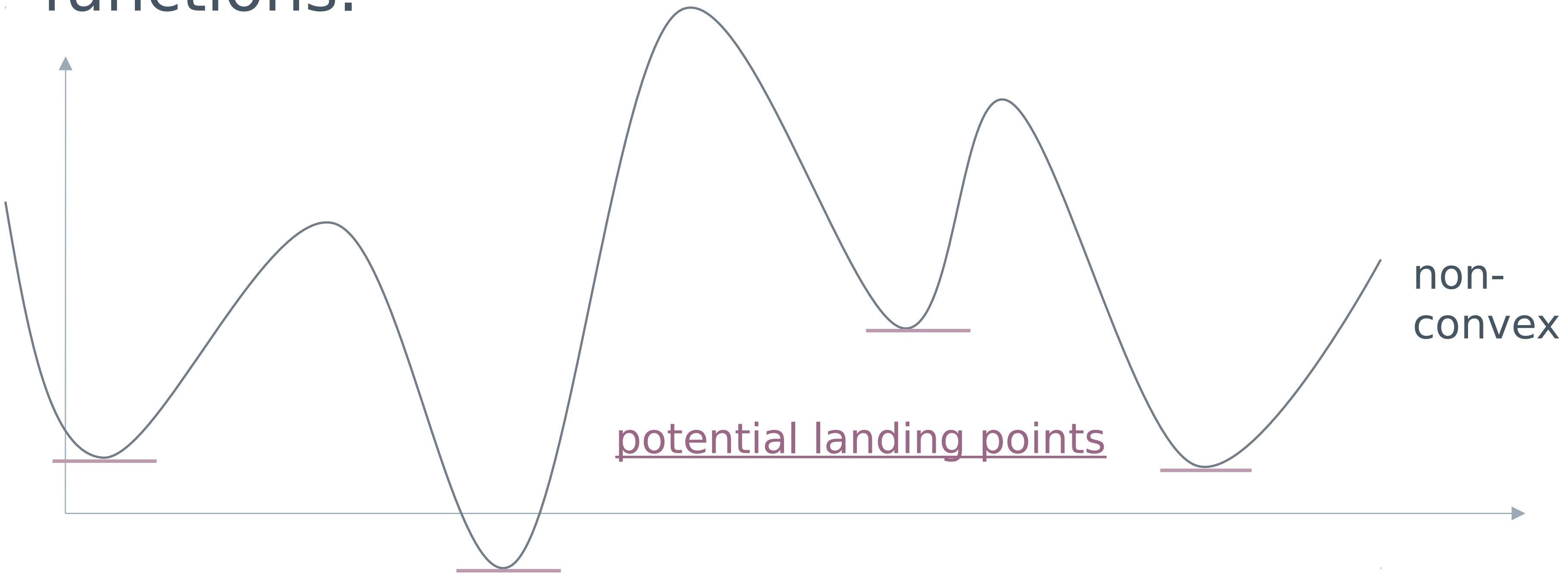
Gradient Descent Intuition

- This technique works well for convex functions.



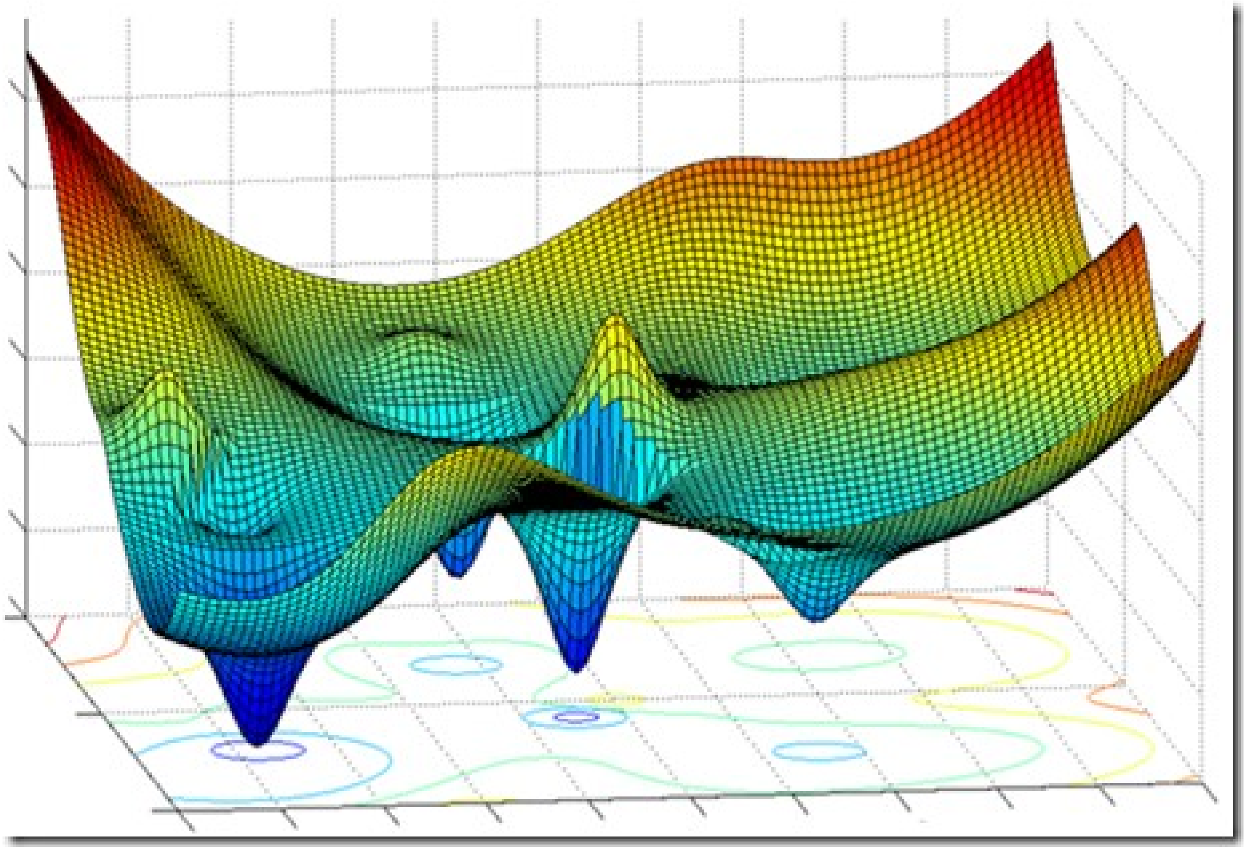
Gradient Descent Intuition

- This technique is not the best for non-convex functions.



- It can potentially find the global min but also local

Gradient Descent Intuition



Gradient Descent

Given some function $f(\beta_0, \beta_1 \ldots)$

We want $\min_{\beta_0, \beta_1 \ldots} f(\beta_0, \beta_1 \ldots)$

High level steps:

- Start with some $\beta_0, \beta_1 (e.g. \beta_0 = 0, \beta_1 = 0)$
- β_0, β_1 will keep changing to reduce $f(\beta_0, \beta_1 \ldots)$ until hopefully we come to the $\min_{\beta_0, \beta_1 \ldots} f(\beta_0, \beta_1 \ldots)$, that is : we converge.

Gradient Descent Algorithm

$$Repeat\ until\ convergence\ \{$$

$$tmp_0 := \beta_0 - \alpha * \frac{d}{d\beta_0} f(\beta_0, \beta_1)$$

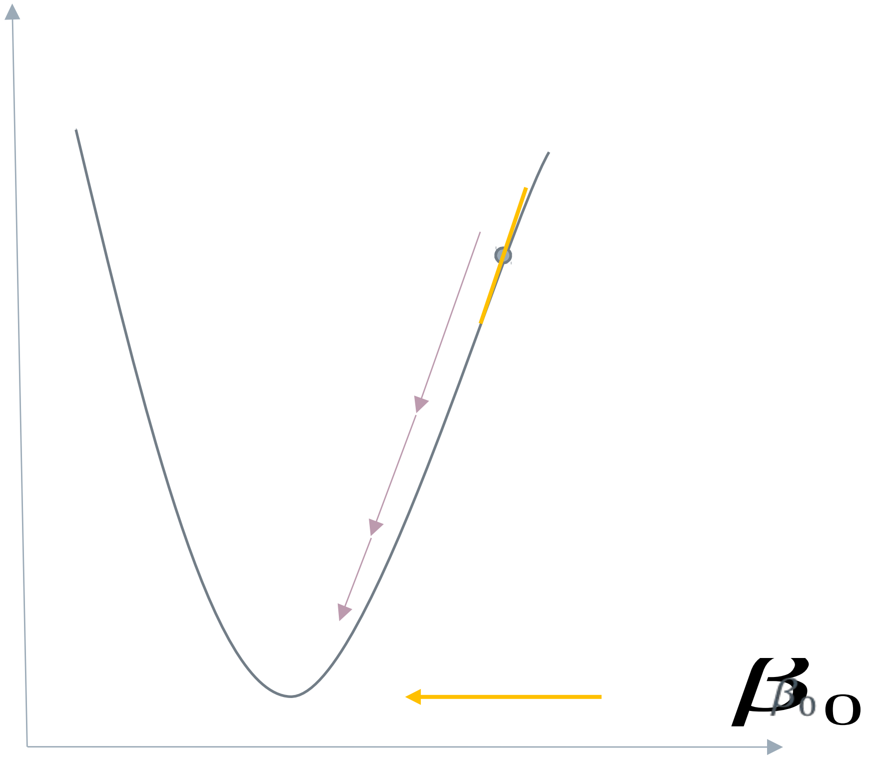
$$tmp_1 := \beta_1 - \alpha * \frac{d}{d\beta_1} f(\beta_0, \beta_1)$$

$$\beta_0 := tmp_0$$

$$\beta_1 := tmp_1$$

$$\}$$

Gradient Descent Intuition



Positive Tangent Case

Positive Tangent Case

$repeat\ until\ convergence$

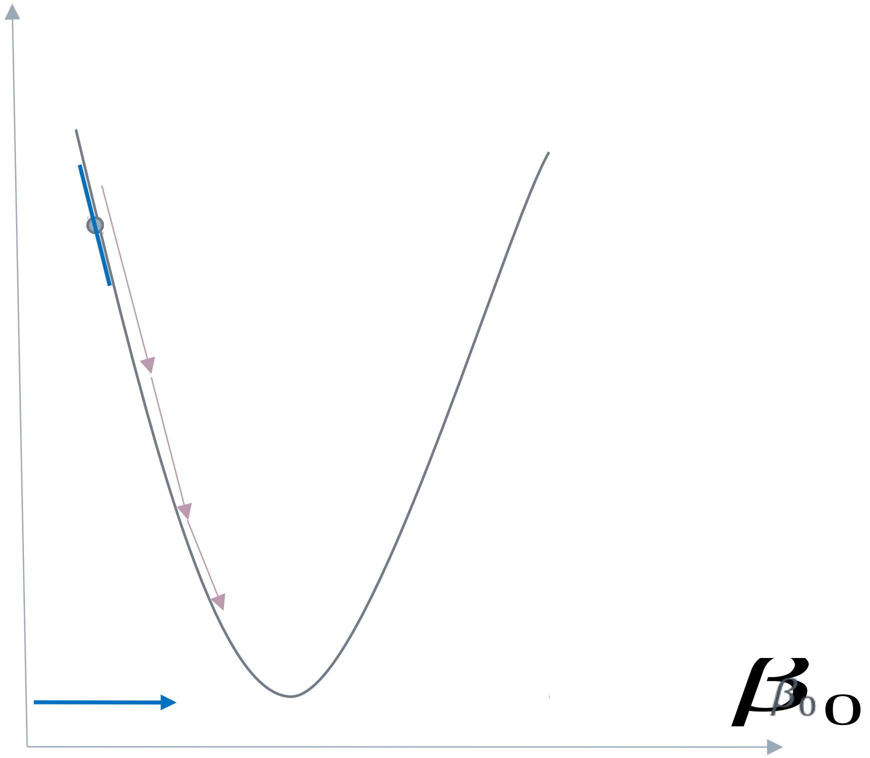
$repeat\ until\ convergence$

$$\{\beta_0 := \beta_0 - \alpha * \frac{d}{d\beta_0} f(\beta_0)\}$$

The learning rate will adjust accordingly.

The learning rate α will adjust accordingly.

Gradient Descent Intuition



Negative Tangent Case

Negative Tangent Case

repeat until convergence

repeat until convergence

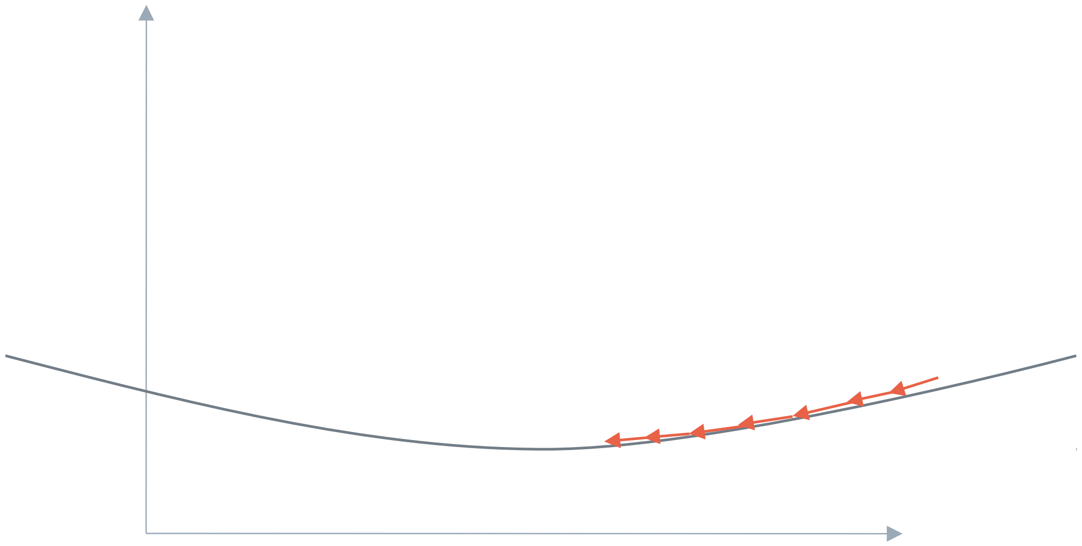
$$\{\beta_0 := \beta_0 - \alpha * \frac{d}{d\beta_0} f(\beta_0)\}$$

The learning rate will adjust accordingly.

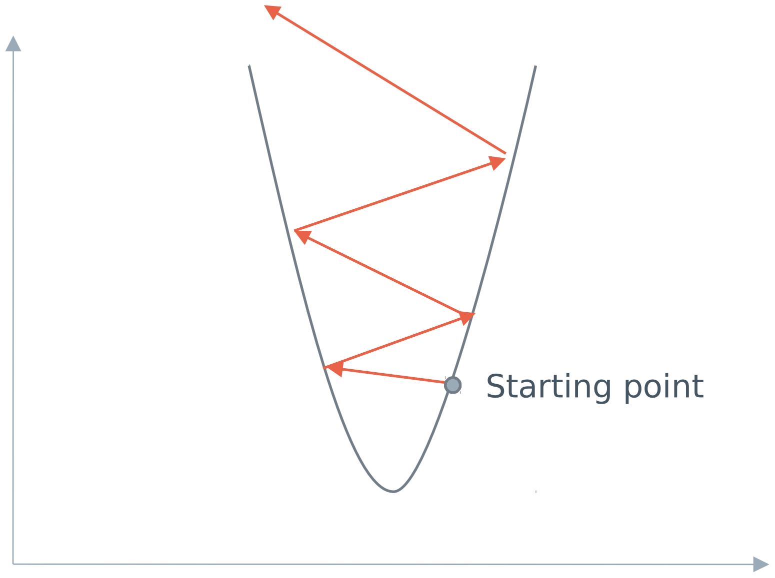
The learning rate α will adjust accordingly.

Some Ugly Cases for Gradient Descent

- It could converge slow, taking micro steps (almost flat surfaces)



- It may not converge at all (large α, may overshoot)



Gradient Descent notation

- $\nabla f(x,y) = \begin{pmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{pmatrix}$
- Can be generalized for any dimension

- Can be generalized for any dimension

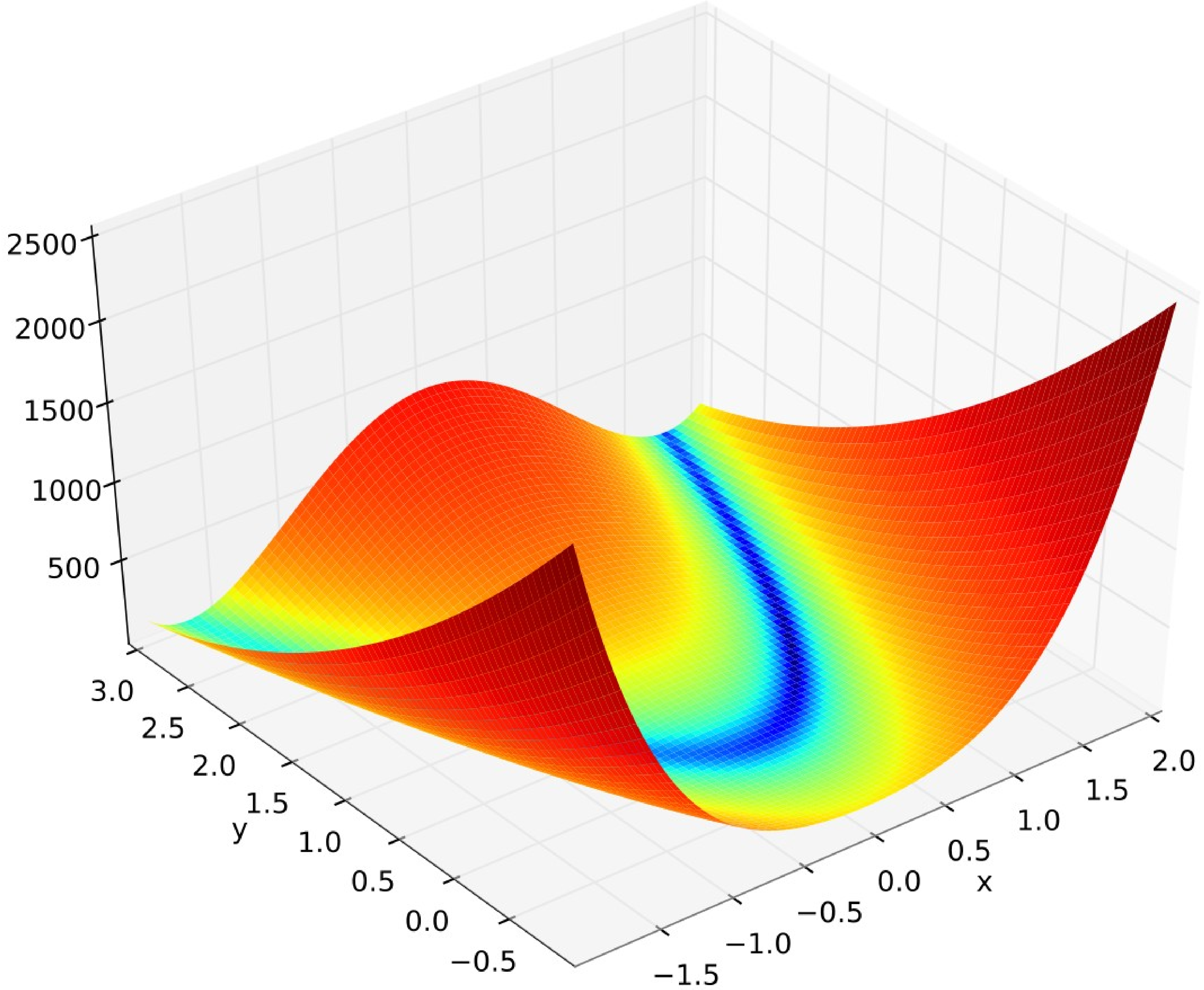
Rosenbrock function

- A special non-convex function used as a performance test problem for optimization algorithms.
- Also known as Rosenbrock's valley function or Rosenbrock's banana function.

$$f(x,y) = (a-x)^2 + b(y-x^2)^2$$

Global min at $(x,y) = (a,a^2)$

Global min at


2500
2000
1500
1000
500
3.0
2.5
2.0
1.5
y
1.0
0.5
0.0
−0.5
−1.5
−1.0
−0.5
0.0
0.5
x
1.0
1.5
2.0

Logistic Regression with Gradient Descent

- $h_\beta(x) = \frac{1}{1+e^{-\beta^T x}}$

- $\frac{d}{dx} h_\beta(x) = \frac{1}{1+e^{-\beta^T x}}\left(1 - \frac{1}{1+e^{-\beta^T x}}\right)$

$Repeat\ until\ convergence\ \{$

$Repeat\ until\ convergence\ \{$

$$\beta := \beta - \alpha * \frac{d}{dx} h_\beta(x)$$

$\}$

Is pure SQL Regression worth it?

MAYBE

- Simple linear regression

DEFINETELY NOT

- Multiple linear regression
- Multiple logistic regression
- “Numerical” Algorithms

Stackoverflow friendliness

Should every database provide statistical analysis functionality? [on hold]

-3

I am aware that neither MySQL nor Postgres provide such kind of functions out of the box.

Example for `regression modelling` , instead of linking a tool like `R` , would it be better if the `SQL interface` supported functions like lm ? if so , then why no SQL interface offers me such functionality yet ?

sql r database statistics

share edit reopen **undelete** flag

asked 2 hours ago
Oleg
463 ● 1 ● 5 ● 19

deleted by Oleg 51 mins ago

put on hold as primarily opinion-based by Gordon Linoff, bummi, John Cappelletti, hrbrmstr, Rui Barradas 1 hour ago

Many good questions generate some degree of opinion based on expert experience, but answers to this question will tend to be almost entirely based on opinions, rather than facts, references, or specific expertise.

If this question can be reworded to fit the rules in the help center, please edit your question.

2 And cars should cook your breakfast too. – Gordon Linoff 2 hours ago

comments disabled on deleted / locked posts / reviews

References

https://en.wikipedia.org/wiki/Linear_regression

https://en.wikipedia.org/wiki/Logistic_regression

https://en.wikipedia.org/wiki/Gradient_descent

https://stackoverflow.com/questions/6449072/doing-calculations-in-mysql-vs-php