ARTIFICIAL INTELLIGENCE 2

Introduction to Linear Models

Francesco Spinnato

* francesco.spinnato@unipi.it
University of Pisa

Regression

Regression predicts a **continuous** variable for each instance in a dataset.

$$f(\mathbf{X}) = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)] = \hat{\mathbf{y}}$$
$$\hat{\mathbf{y}} \in \mathbb{R}^N$$

Here f is a model learned from data, and $\hat{\mathbf{y}}$ are predictions on the observed dataset. We learn f from training data and later use it to predict on new data.

Linear Regression

Simple (Univariate)

Simple Univariate Regression

Let's start simple and suppose our dataset is composed of only:

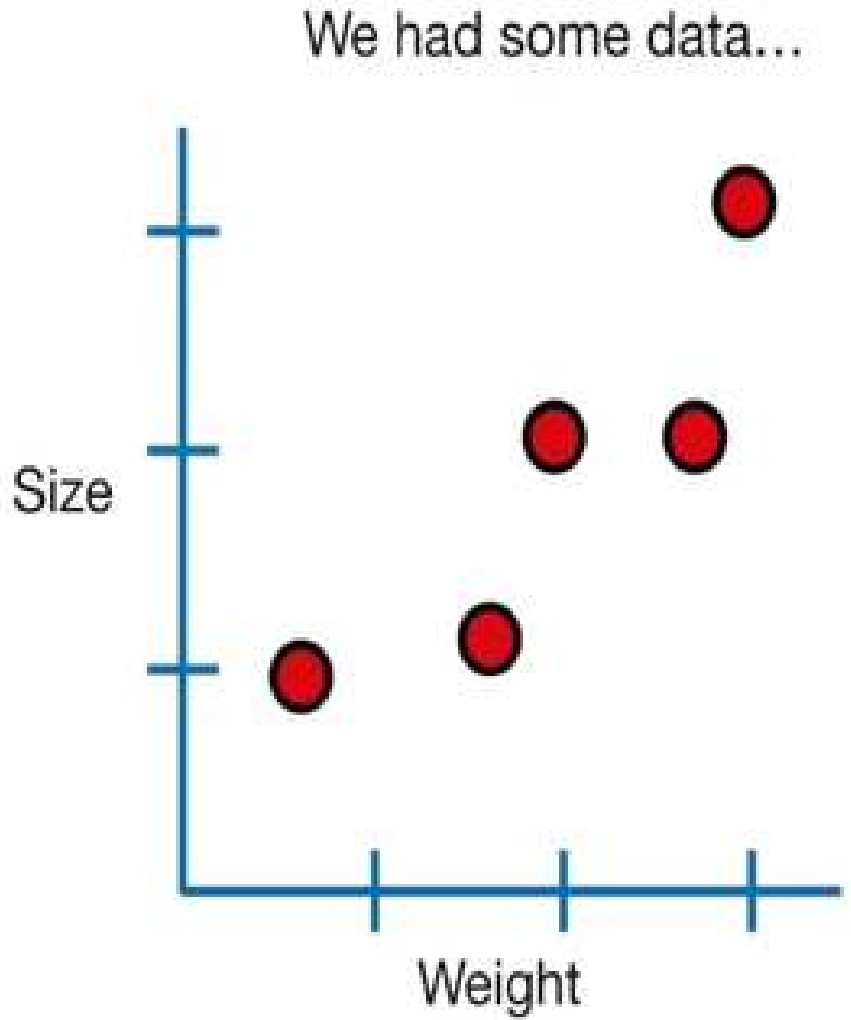
- one feature column, $\mathbf{x}$ (independent variable)
- one target column, $\mathbf{y}$ (dependent variable)

$\mathbf{x}$	$\mathbf{y}$
x_1	y_1
x_2	y_2
$\dots$	$\dots$
x_N	y_N

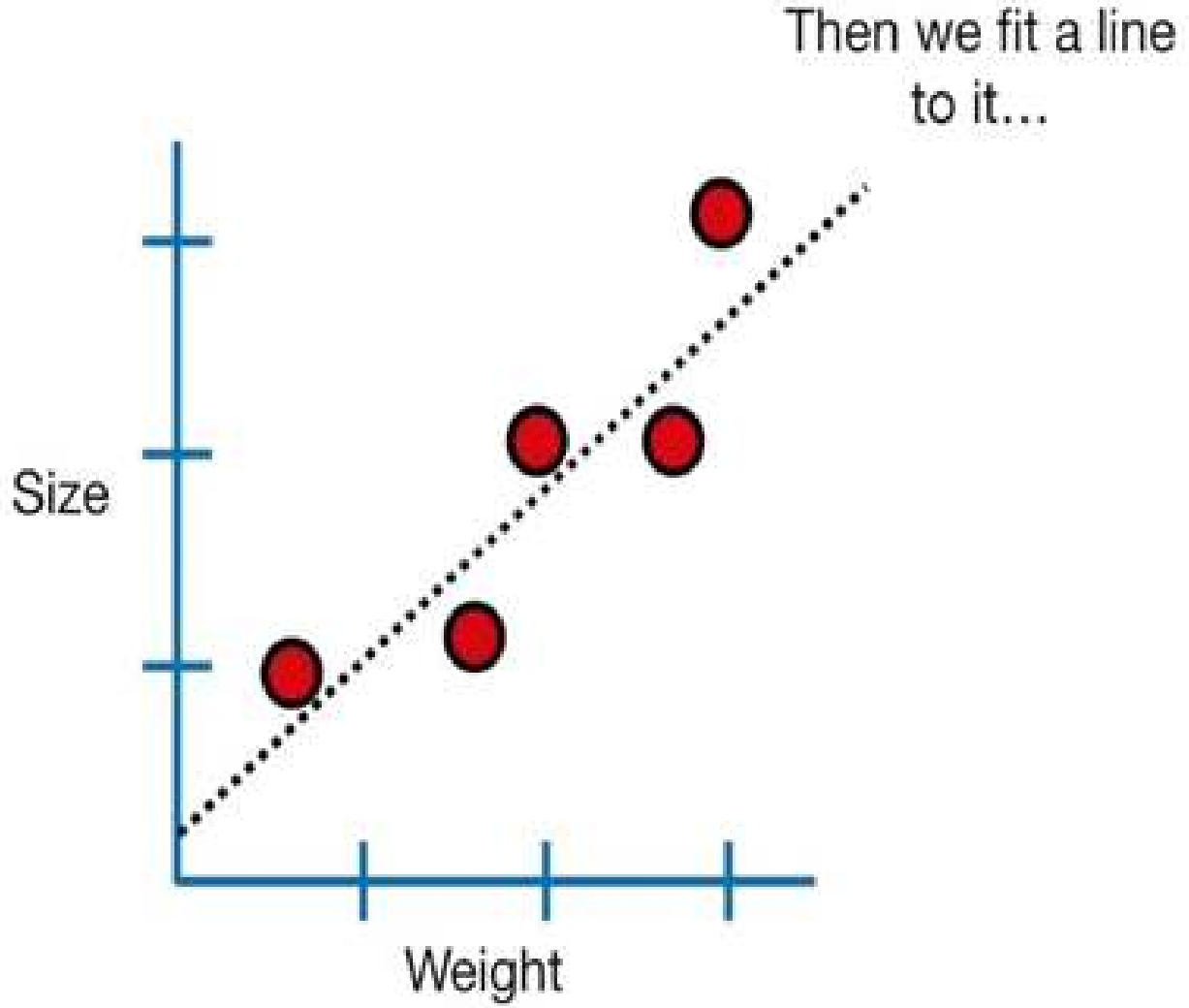
We want to predict each y given the corresponding x.

Applied Linear Regression, Chapter 2
https://www.stat.purdue.edu/~qfsong/teaching/525/book/Weisberg-Applied-Linear-Regression-Wiley.pdf

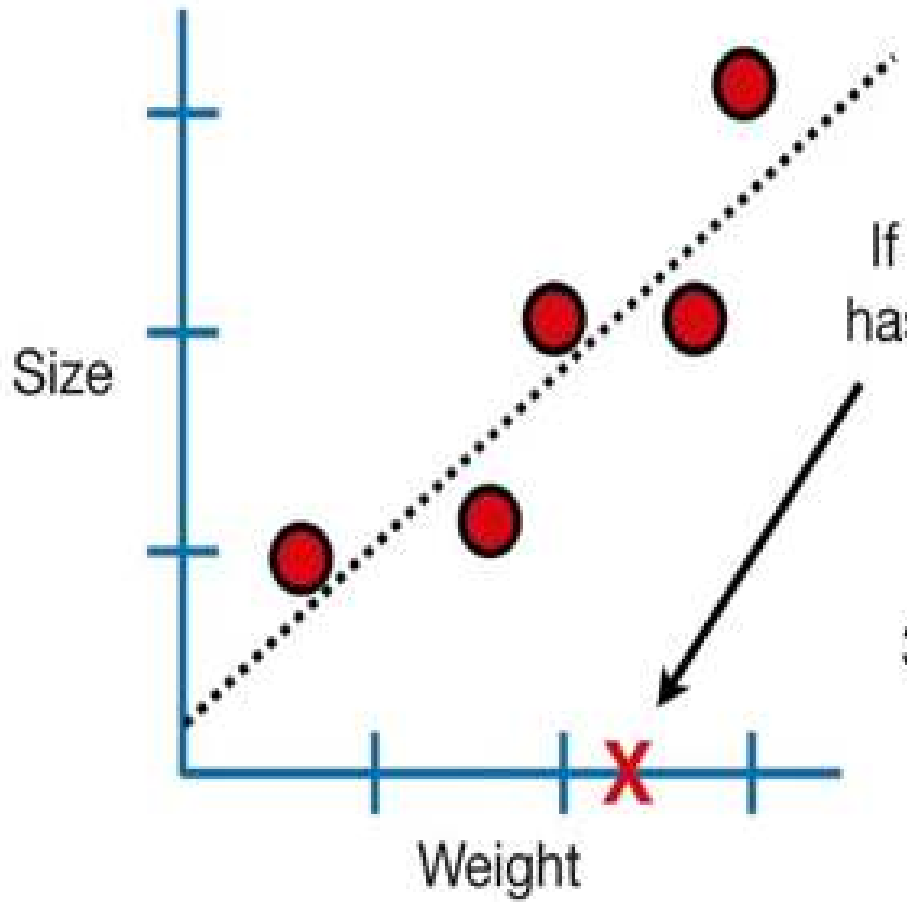
What does it mean to linearly predict y?



What does it mean to linearly predict y?



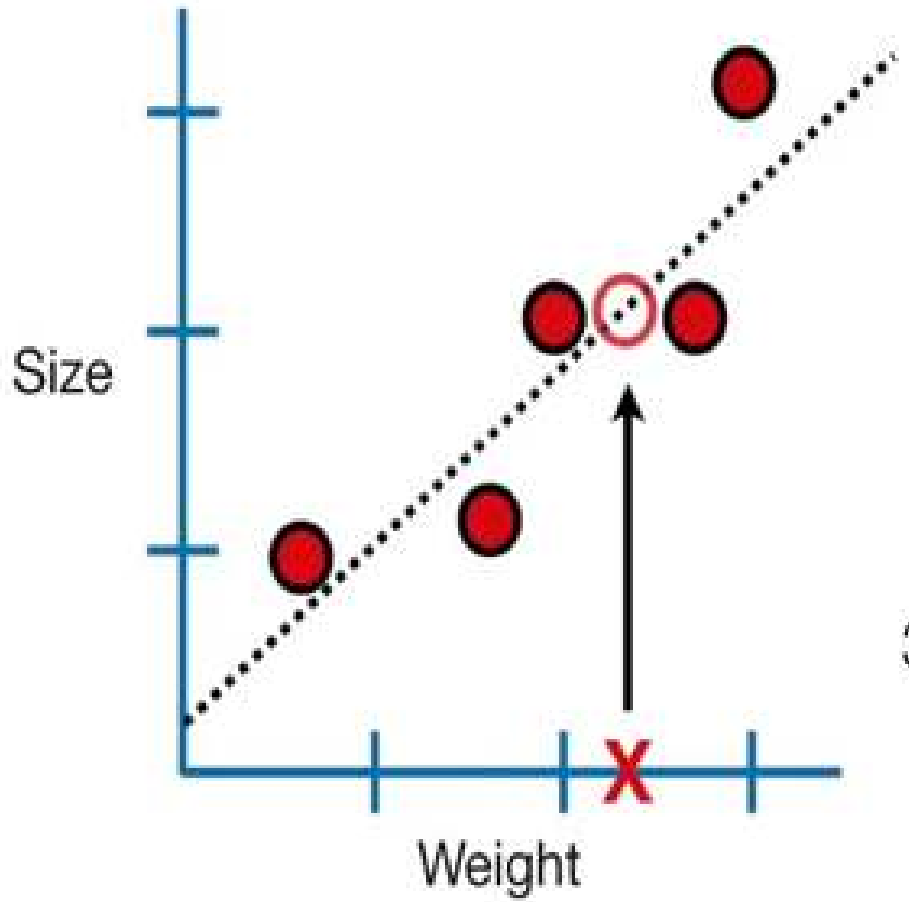
What does it mean to linearly predict y?



If a new mouse has this weight…

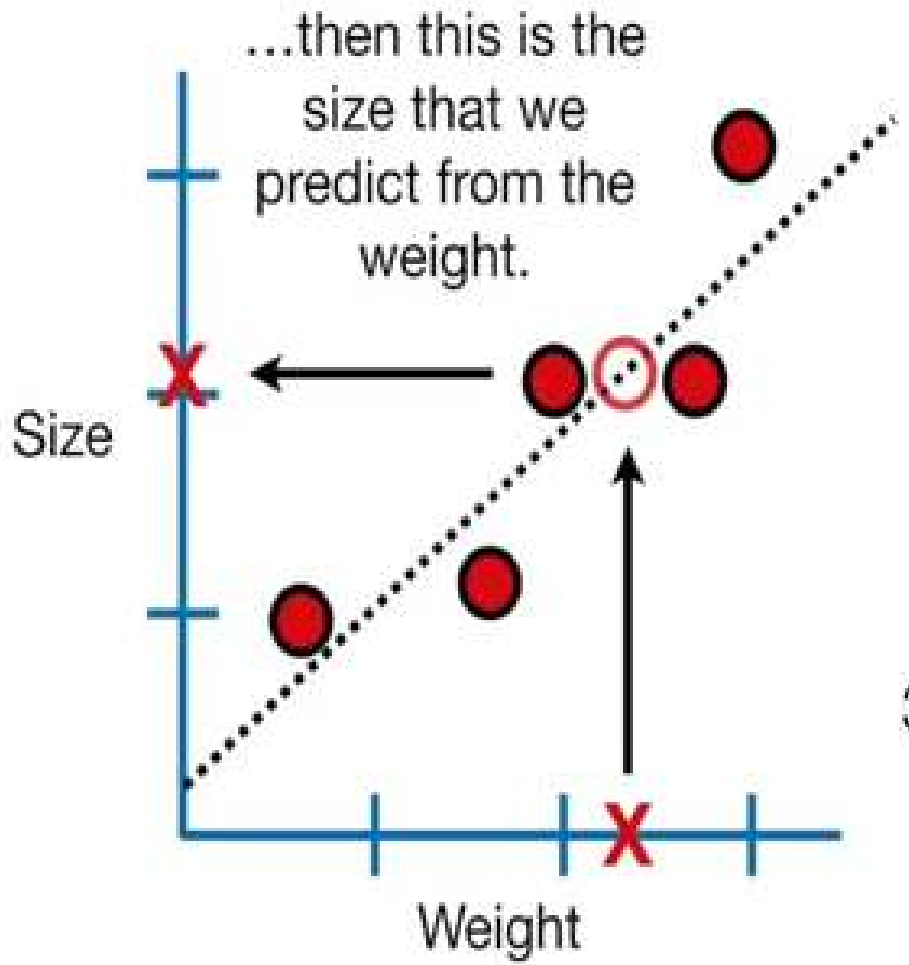
3) Use the line to predict ***size*** given ***weight***.

What does it mean to linearly predict y?



3) Use the line to predict ***size*** given ***weight***.

What does it mean to linearly predict y?



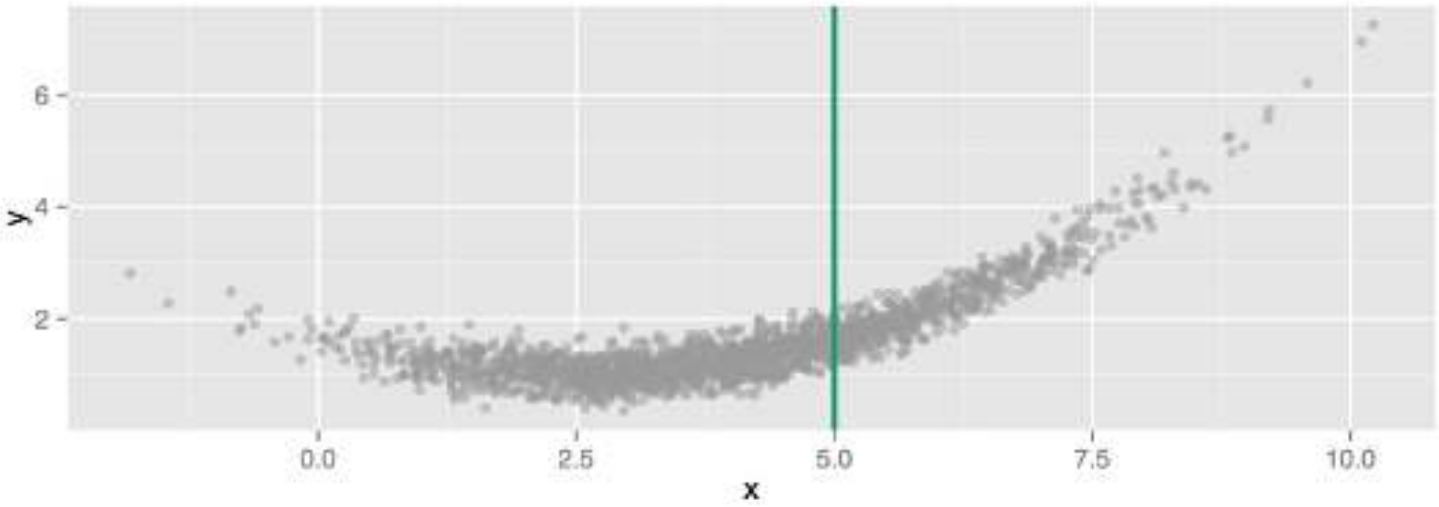
3) Use the line to predict ***size*** given ***weight***.

What does it mean to linearly predict y?

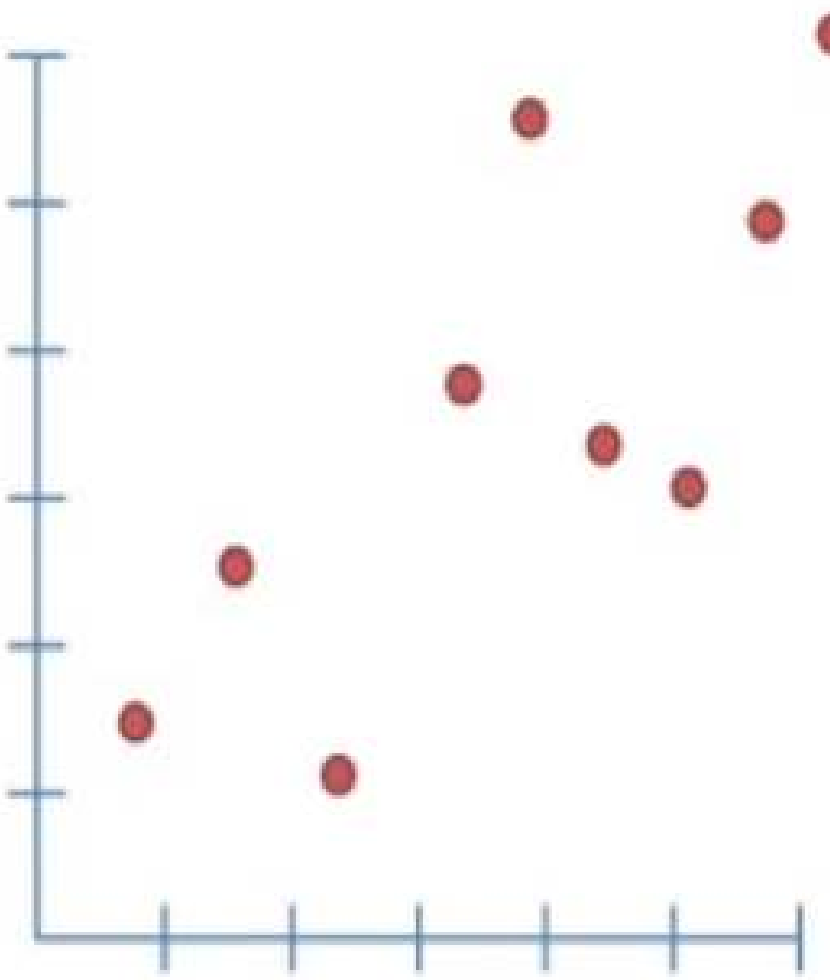
Look at $x = 5$; there are many different y values.

When we say predict y at $x = 5$, we are really asking:

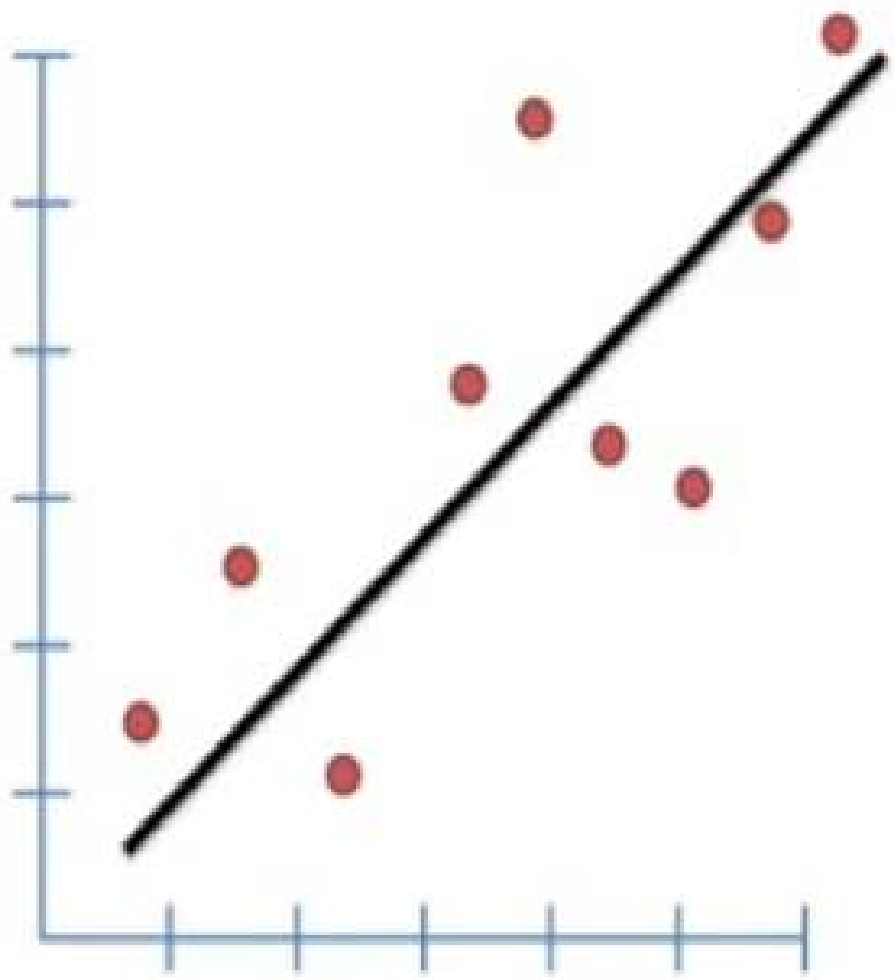
What is the expected value (average) of y at $x = 5$?



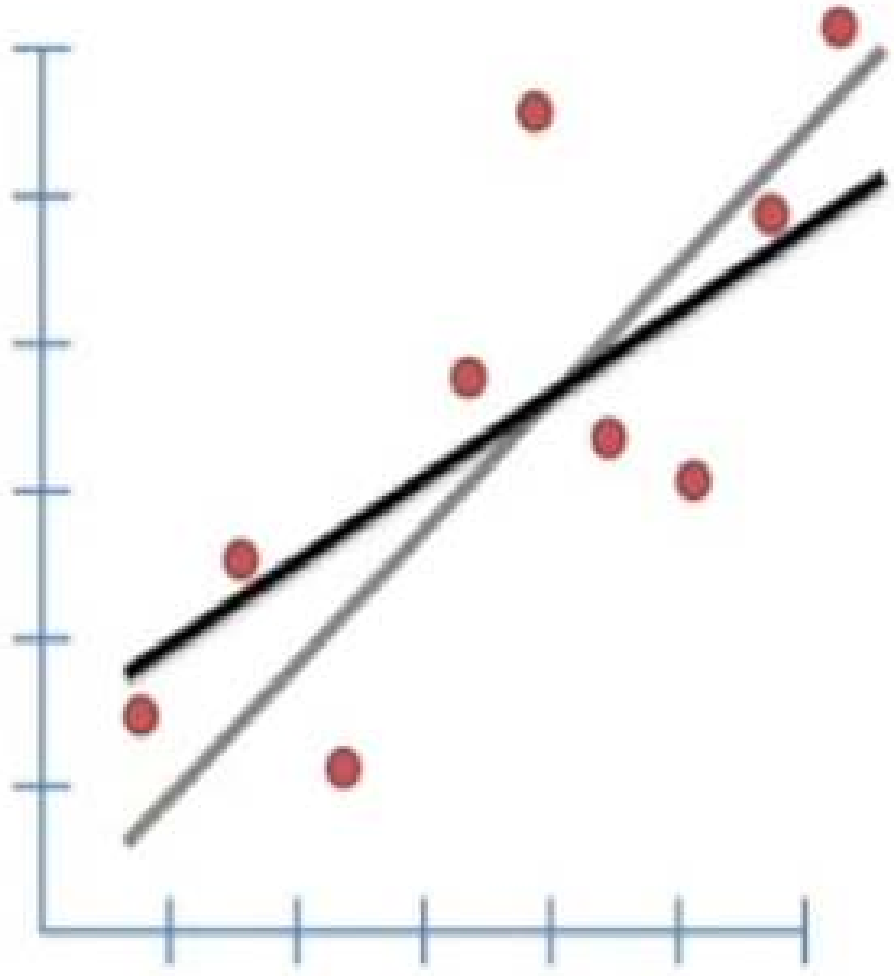
Fitting a Linear Regression Model



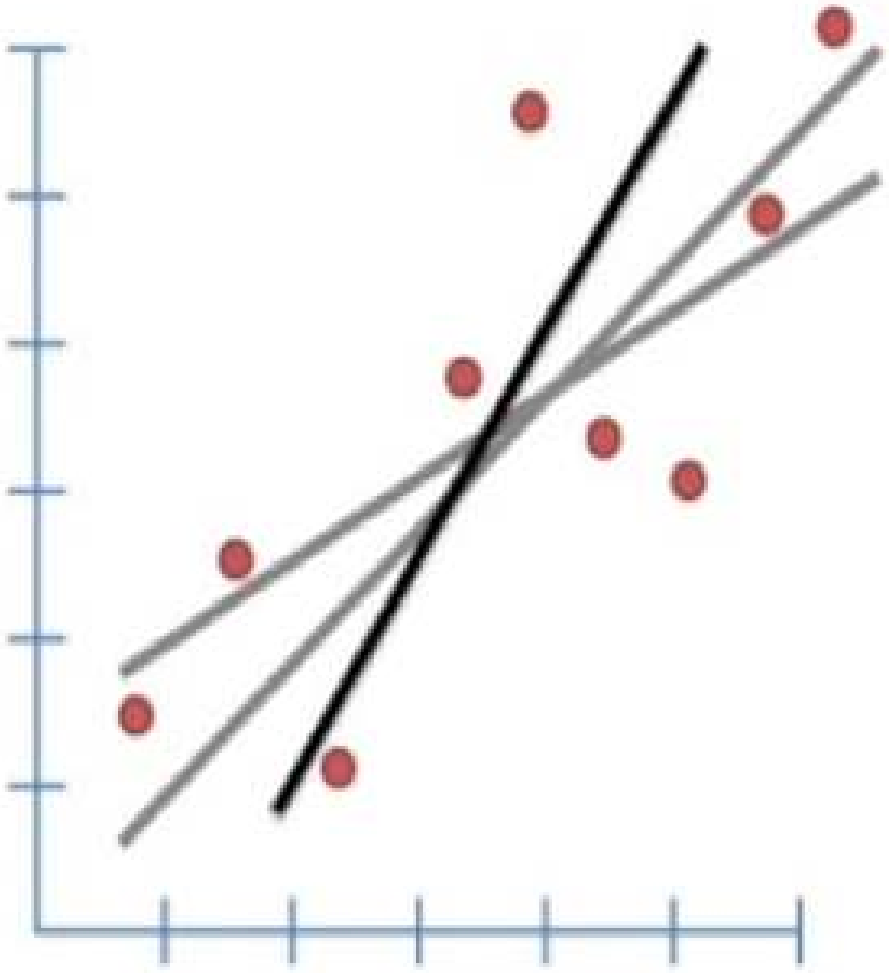
Fitting a Linear Regression Model



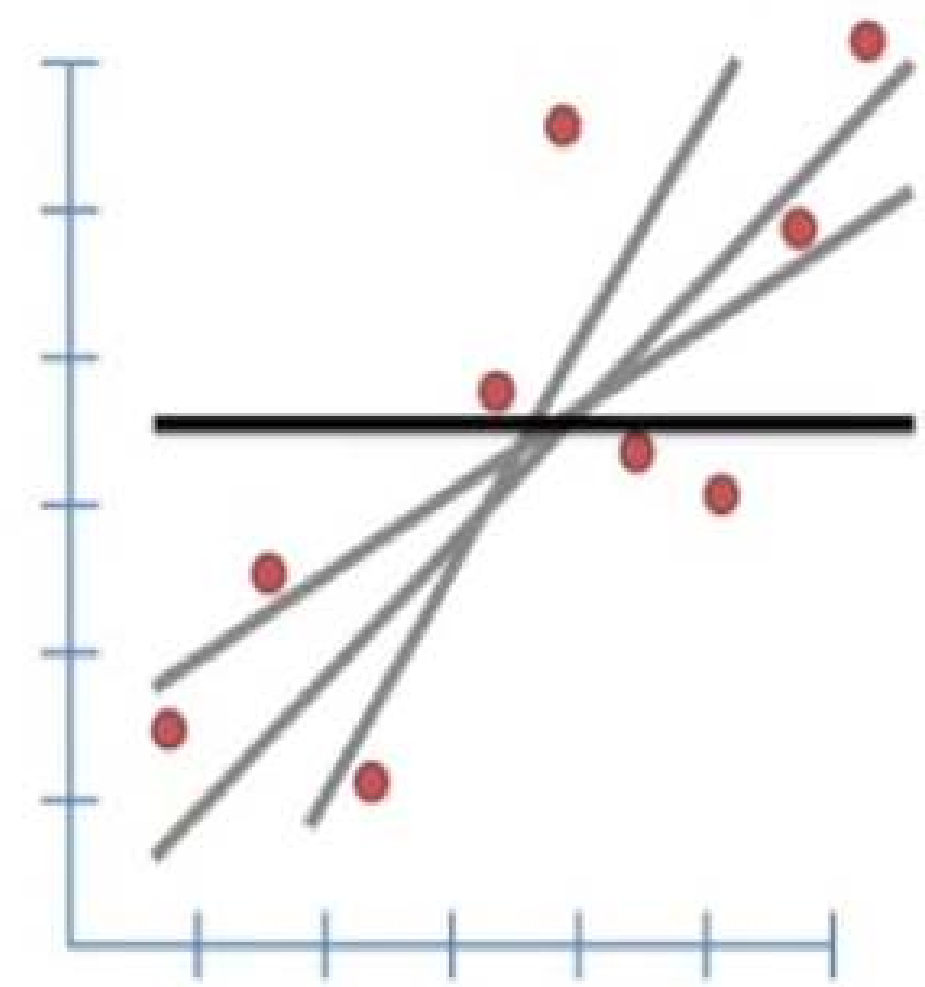
Fitting a Linear Regression Model



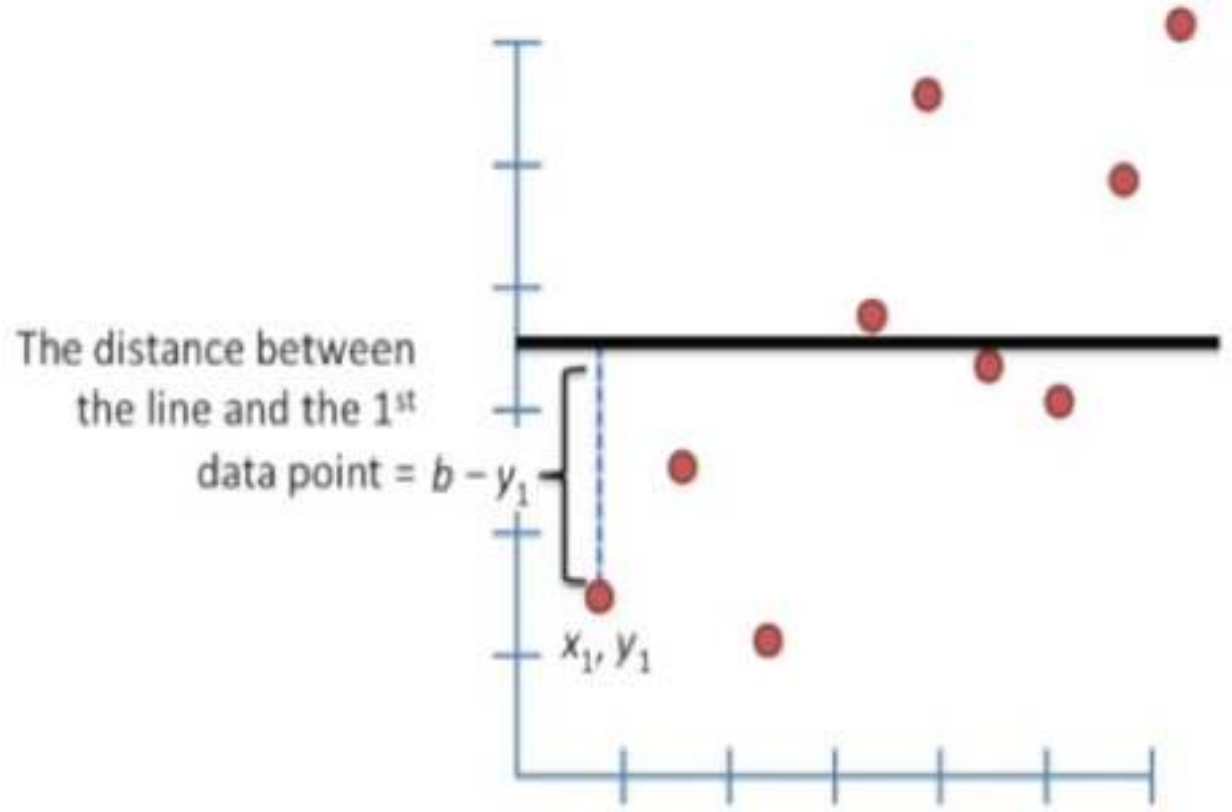
Fitting a Linear Regression Model



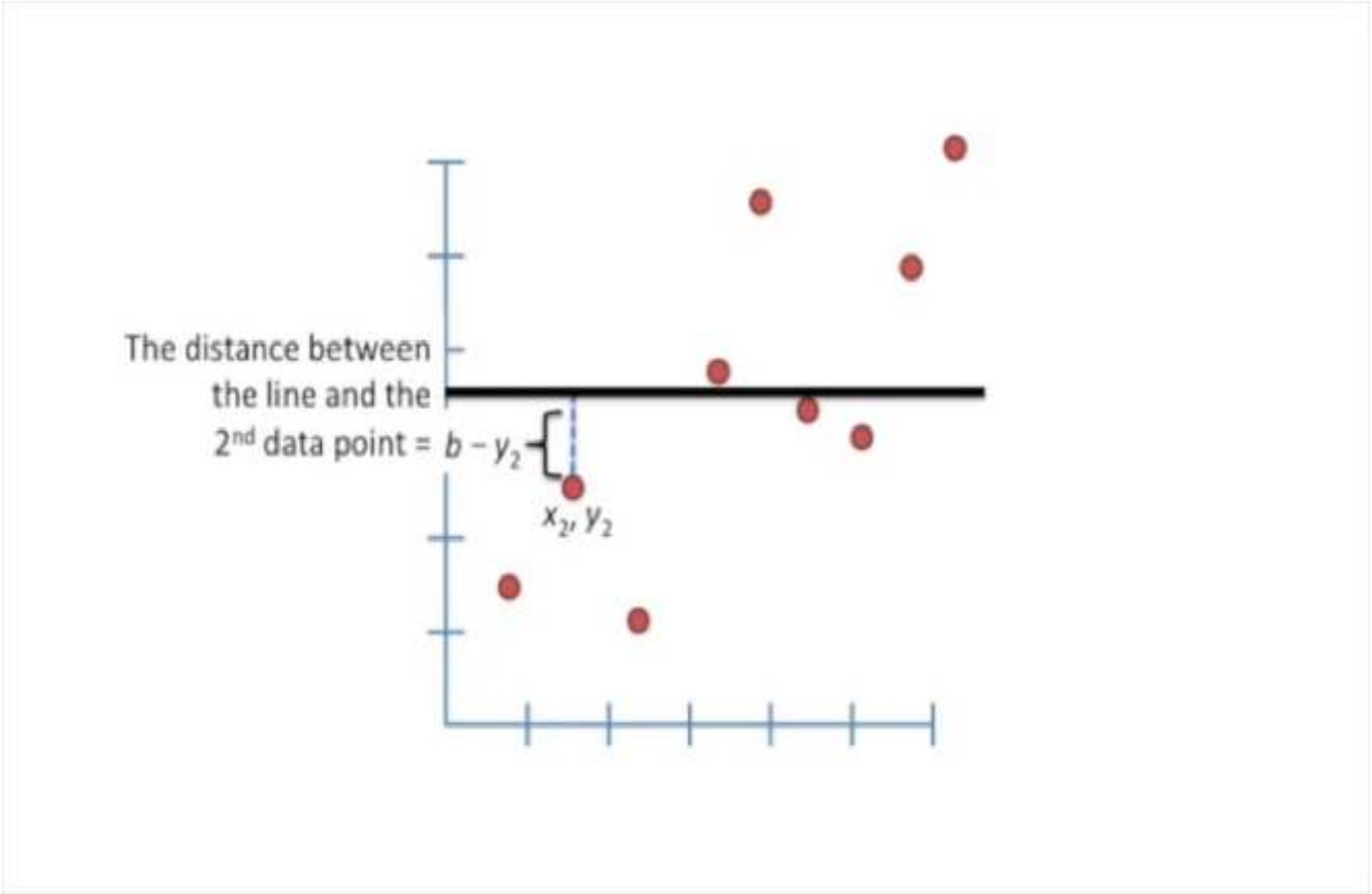
Fitting a Linear Regression Model



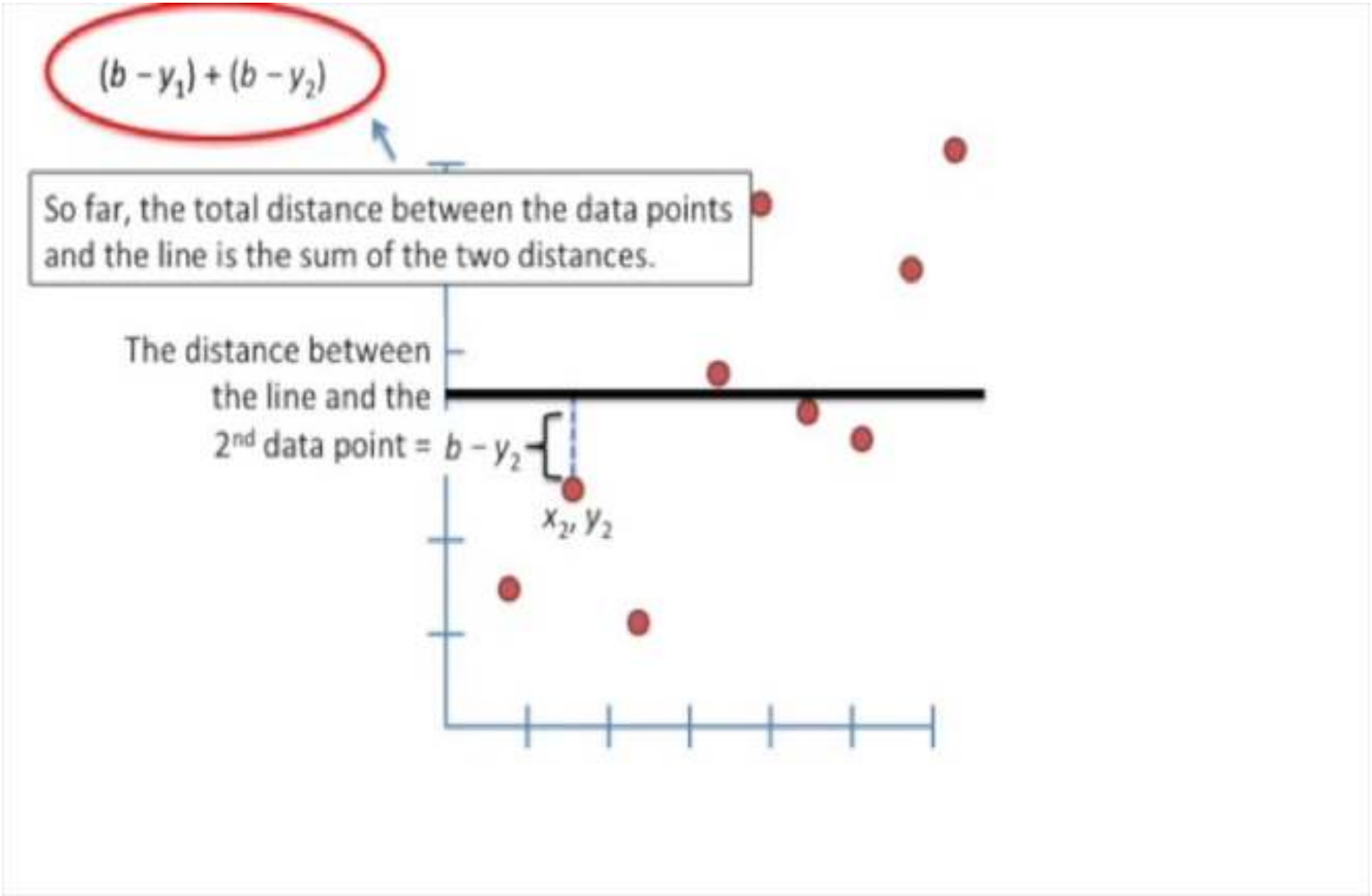
How to choose the best line?



How to choose the best line?

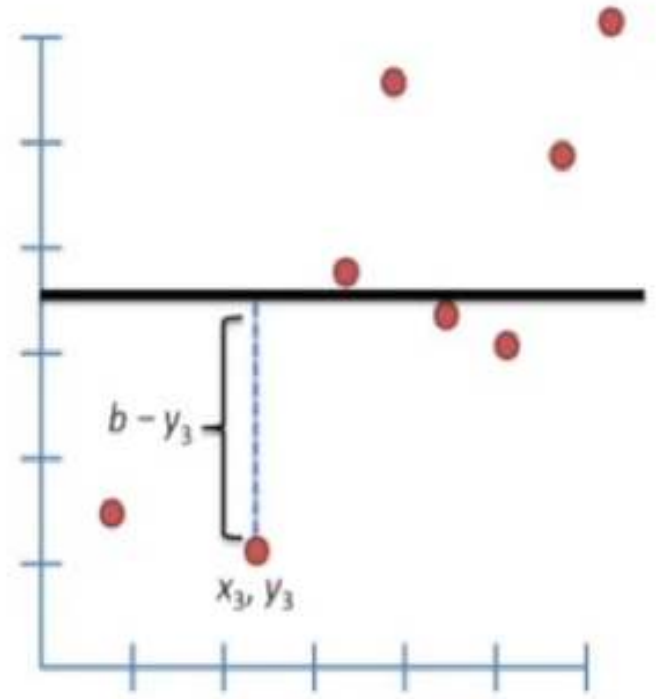


How to choose the best line?

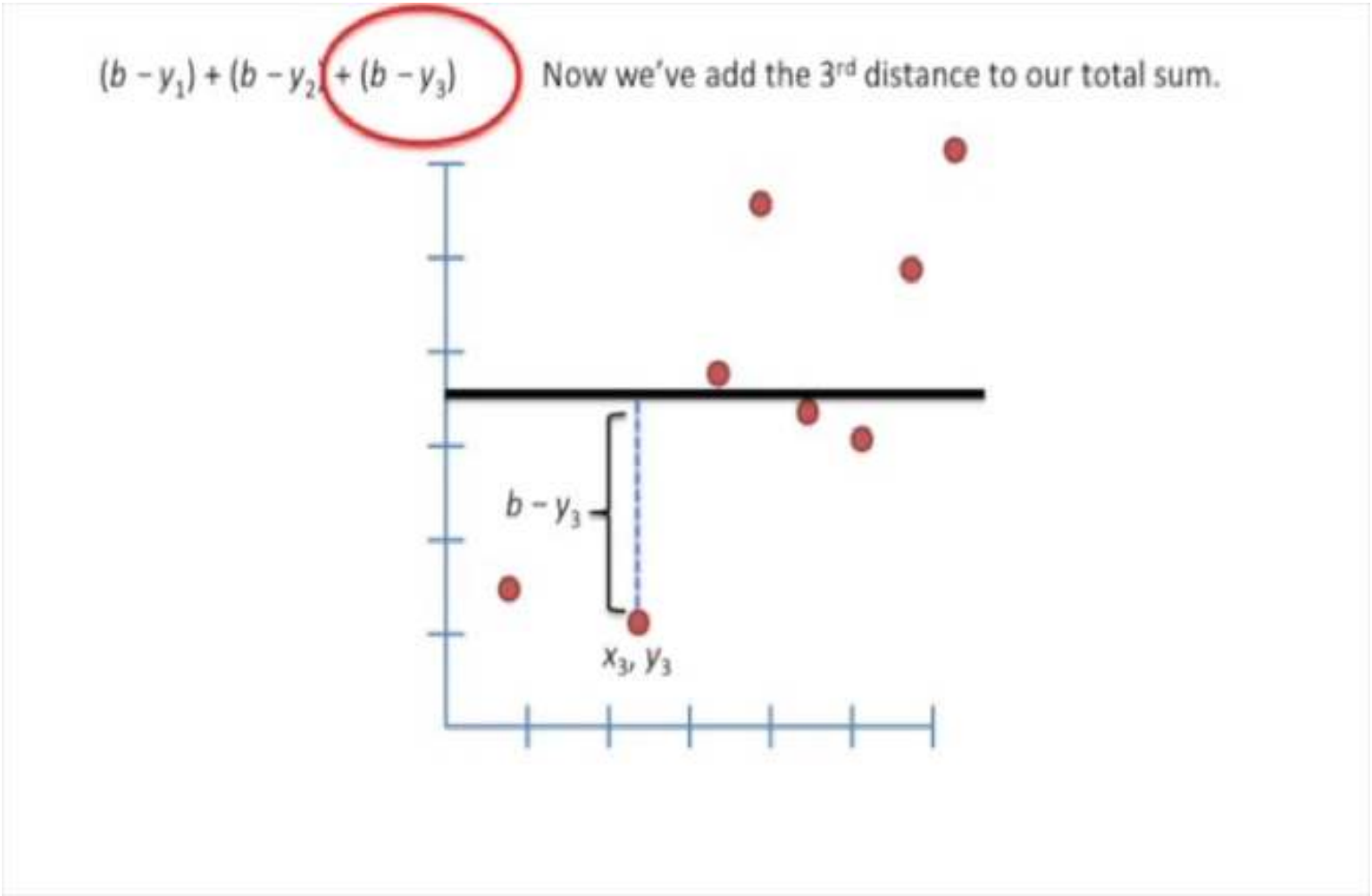


How to choose the best line?

$(b - y_1) + (b - y_2)$

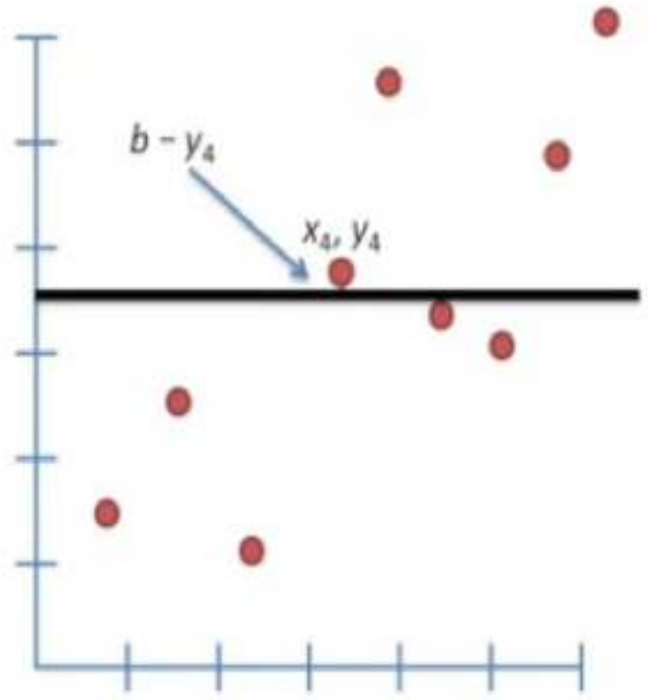


How to choose the best line?



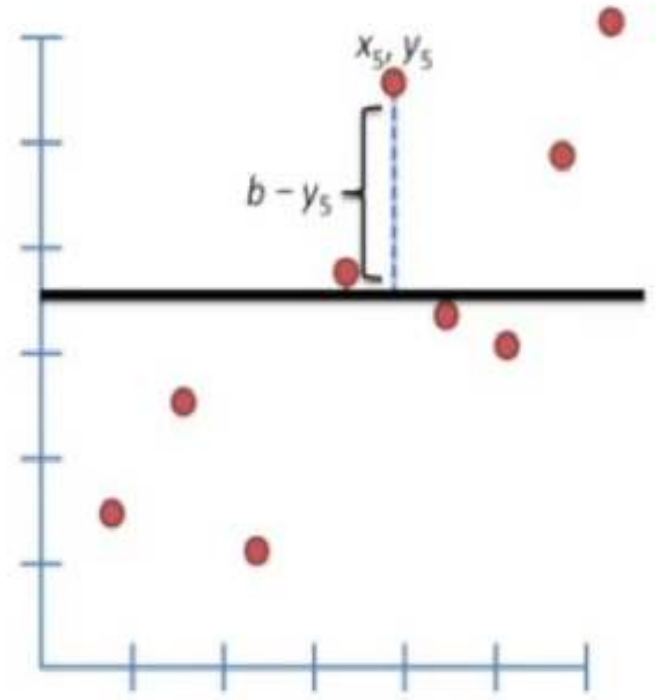
How to choose the best line?

$(b - y_1) + (b - y_2) + (b - y_3)$



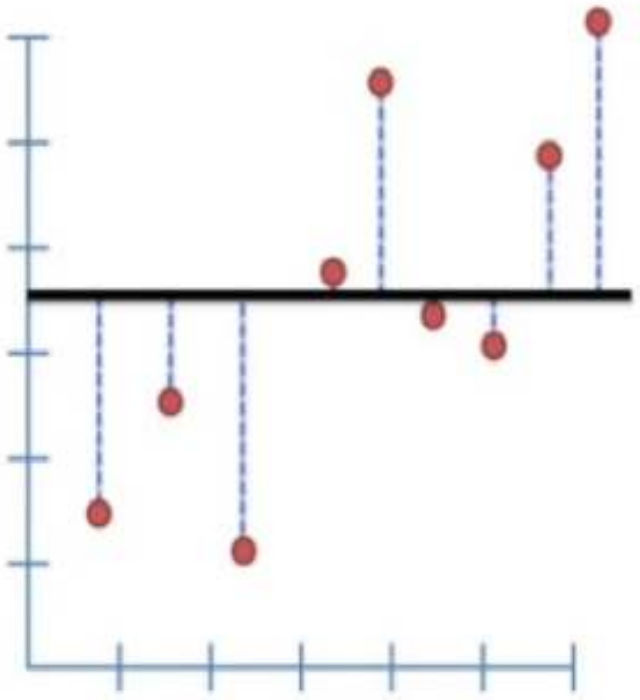
How to choose the best line?

$$(b - y_1) + (b - y_2) + (b - y_3) + (b - y_4)$$



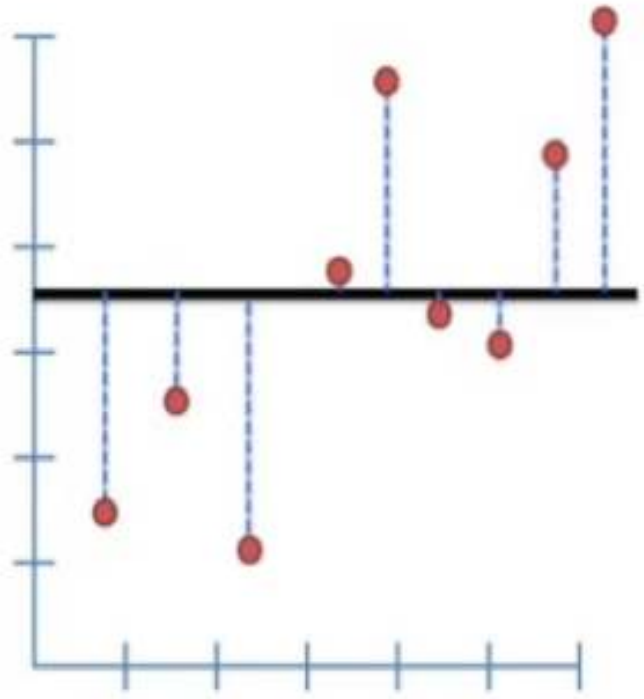
How to choose the best line?

$$(b - y_1)^2 + (b - y_2)^2 + (b - y_3)^2 + (b - y_4)^2 + (b - y_5)^2 + (b - y_6)^2 + (b - y_7)^2 + (b - y_8)^2 + (b - y_9)^2$$



How to choose the best line?

$$(b - y_1)^2 + (b - y_2)^2 + (b - y_3)^2 + (b - y_4)^2 + (b - y_5)^2 + (b - y_6)^2 + (b - y_7)^2 + (b - y_8)^2 + (b - y_9)^2$$

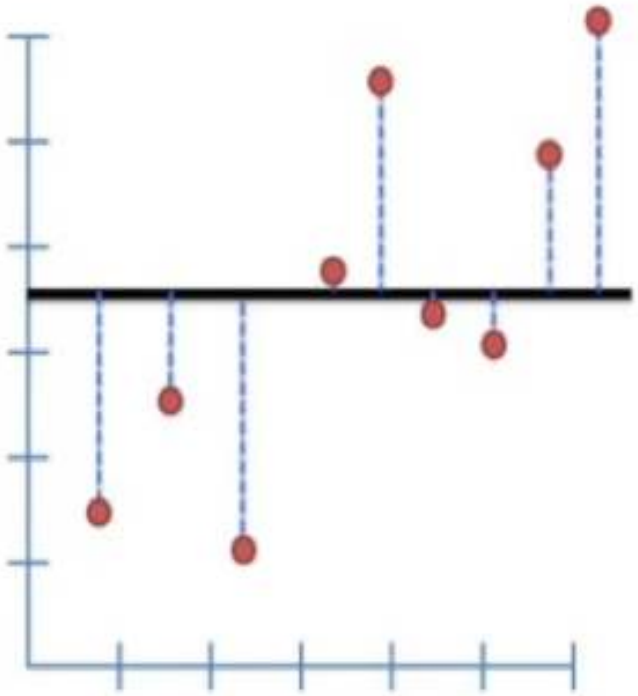


= 24.62

This is our measure of how well this line fits the data.

How to choose the best line?

$$(b - y_1)^2 + (b - y_2)^2 + (b - y_3)^2 + (b - y_4)^2 + (b - y_5)^2 + (b - y_6)^2 + (b - y_7)^2 + (b - y_8)^2 + (b - y_9)^2$$



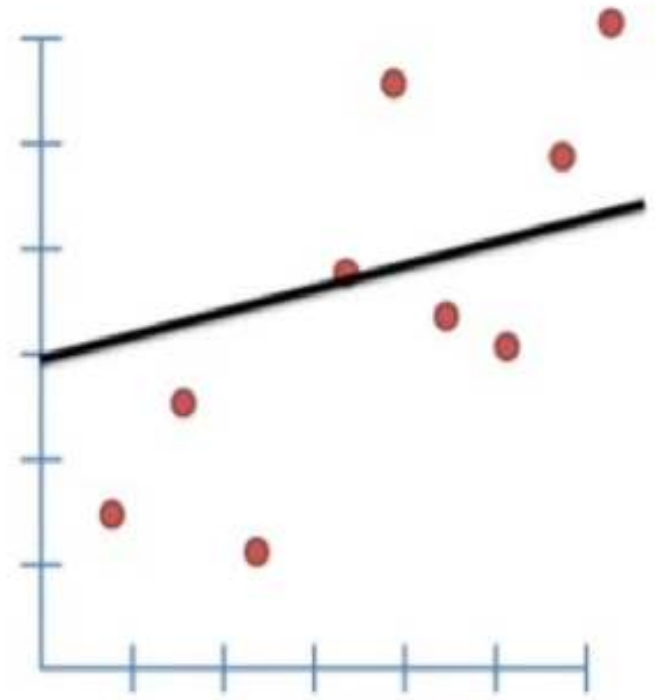
= 24.62

This is our measure of how well this line fits the data.

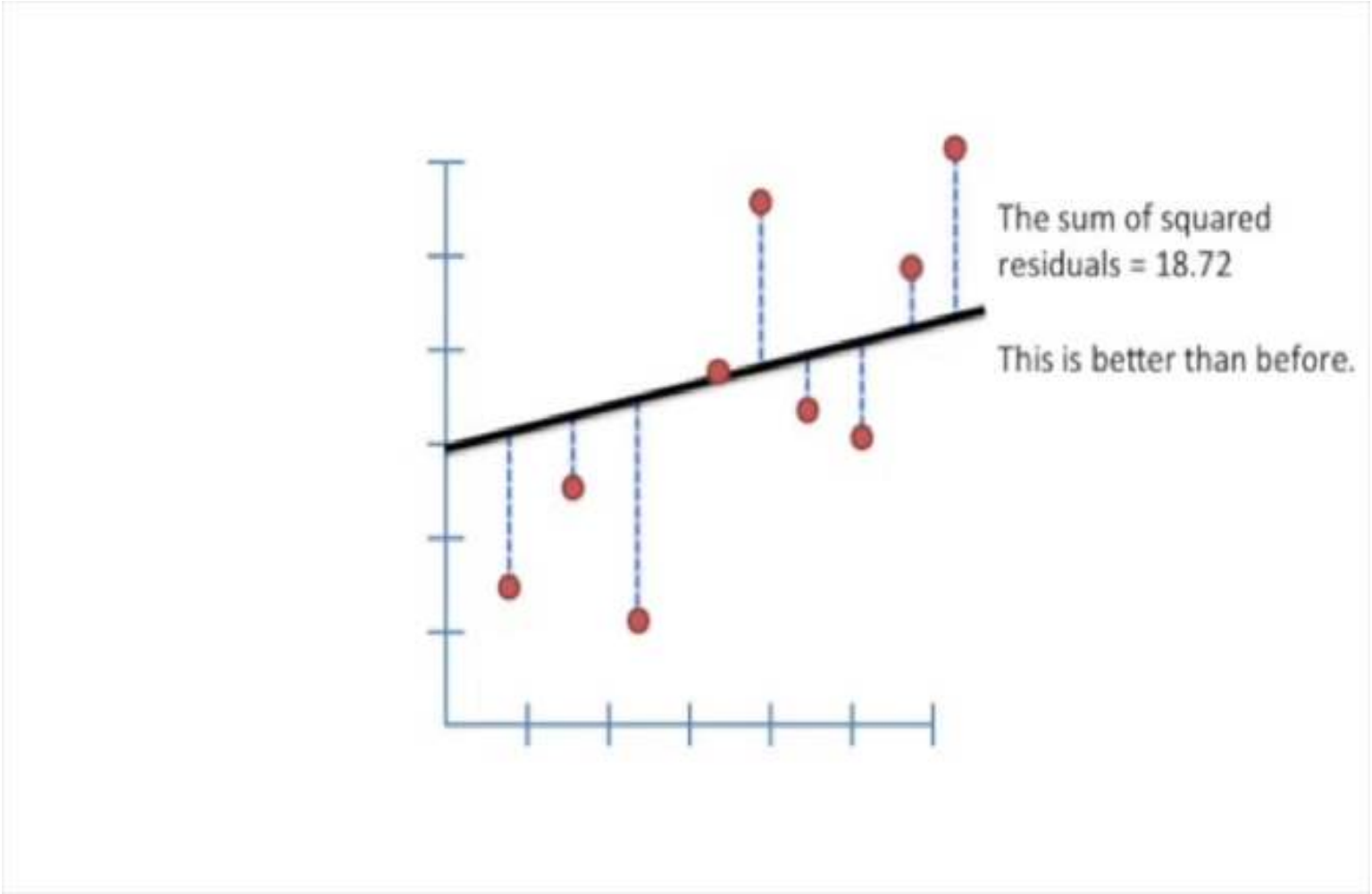
It's called the "sum of squared residuals, because the residuals are the differences between the real data and the line, and we are summing the square of these values.

How to choose the best line?

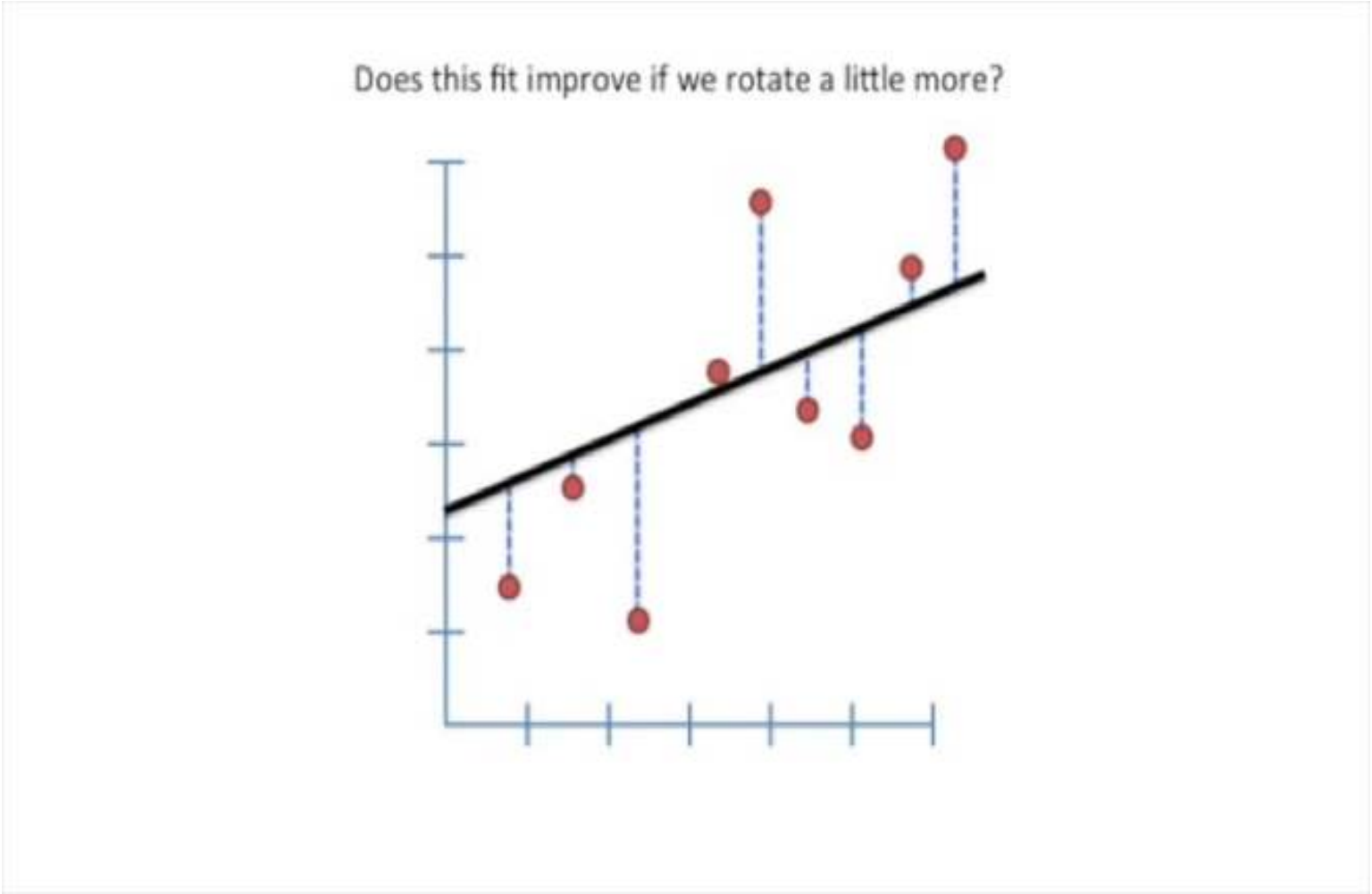
Now let's see how good the fit is if we rotate the line a little bit.



How to choose the best line?



How to choose the best line?



The Line Equation

The line equation can be denoted in a few equivalent ways:

$$y = mx + b$$

$$y = ax + b$$

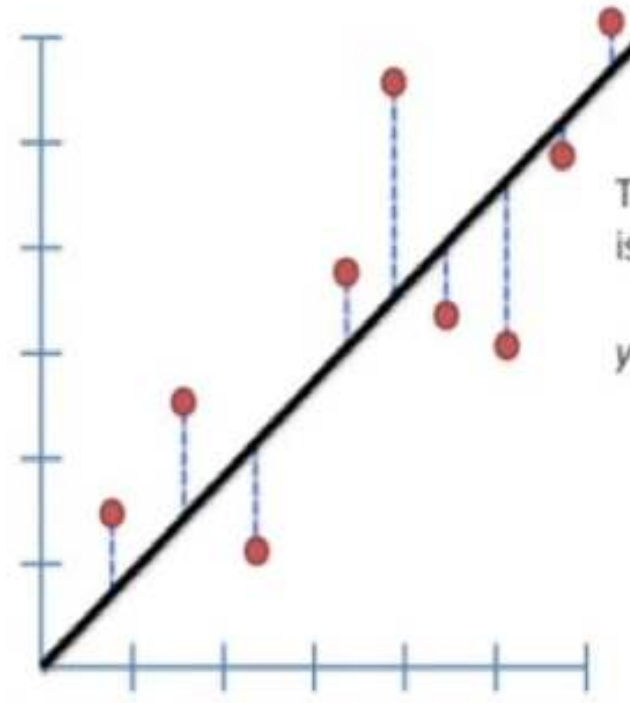
$$y = \beta_1 x + \beta_0$$

$$y = w_1 x + w_0$$

The Line Equation

$$\overbrace{y}^{\text{dependent variable}} = \underbrace{m}_{\text{slope}} \overbrace{x}^{\text{independent variable}} + \underbrace{b}_{\text{bias}}$$

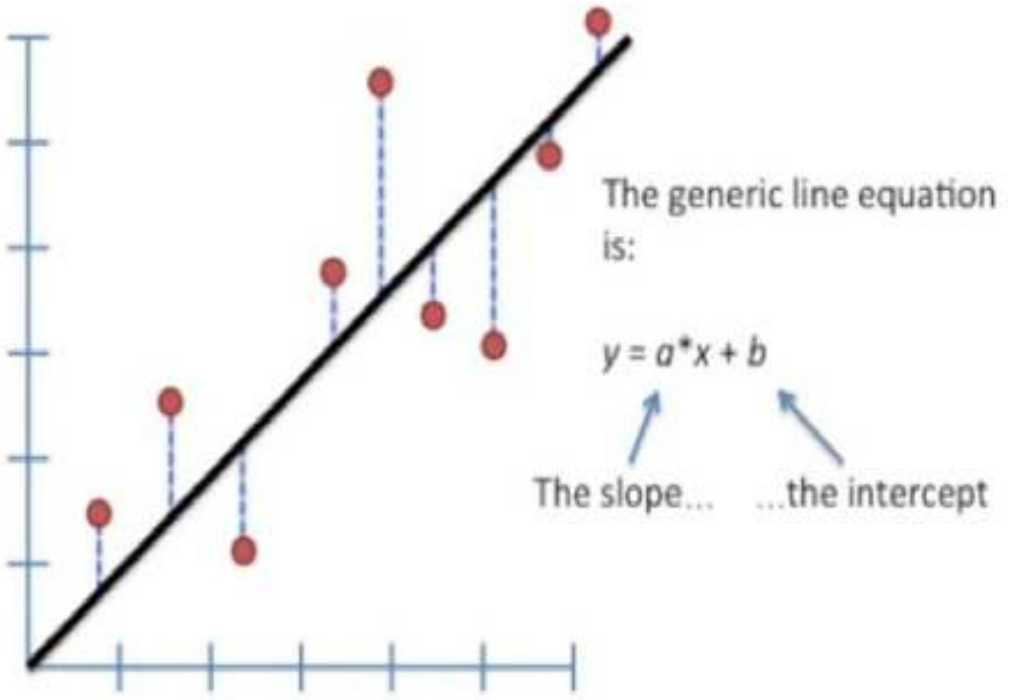
The Line Equation



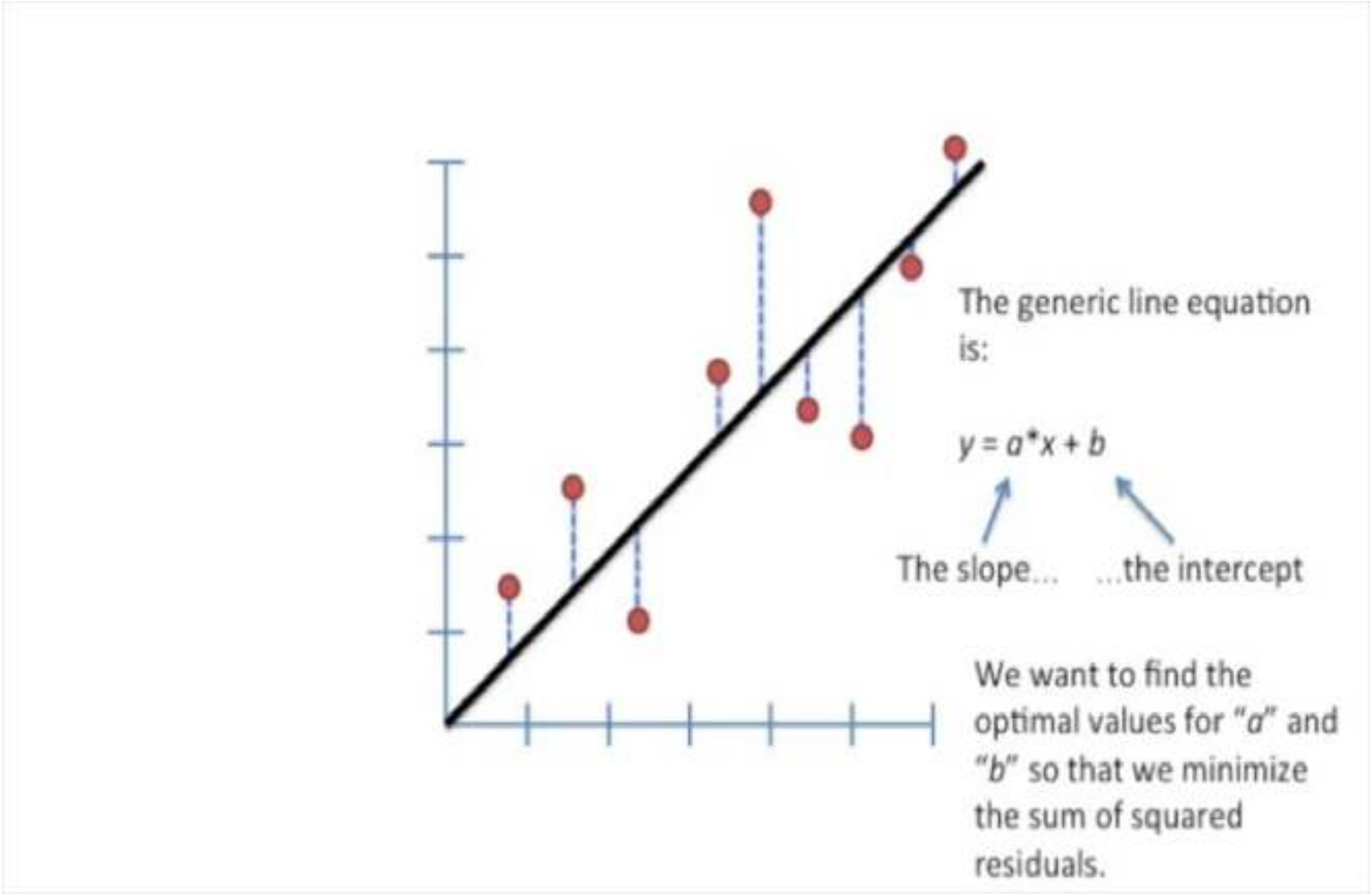
The generic line equation is:

$y = a^*x + b$

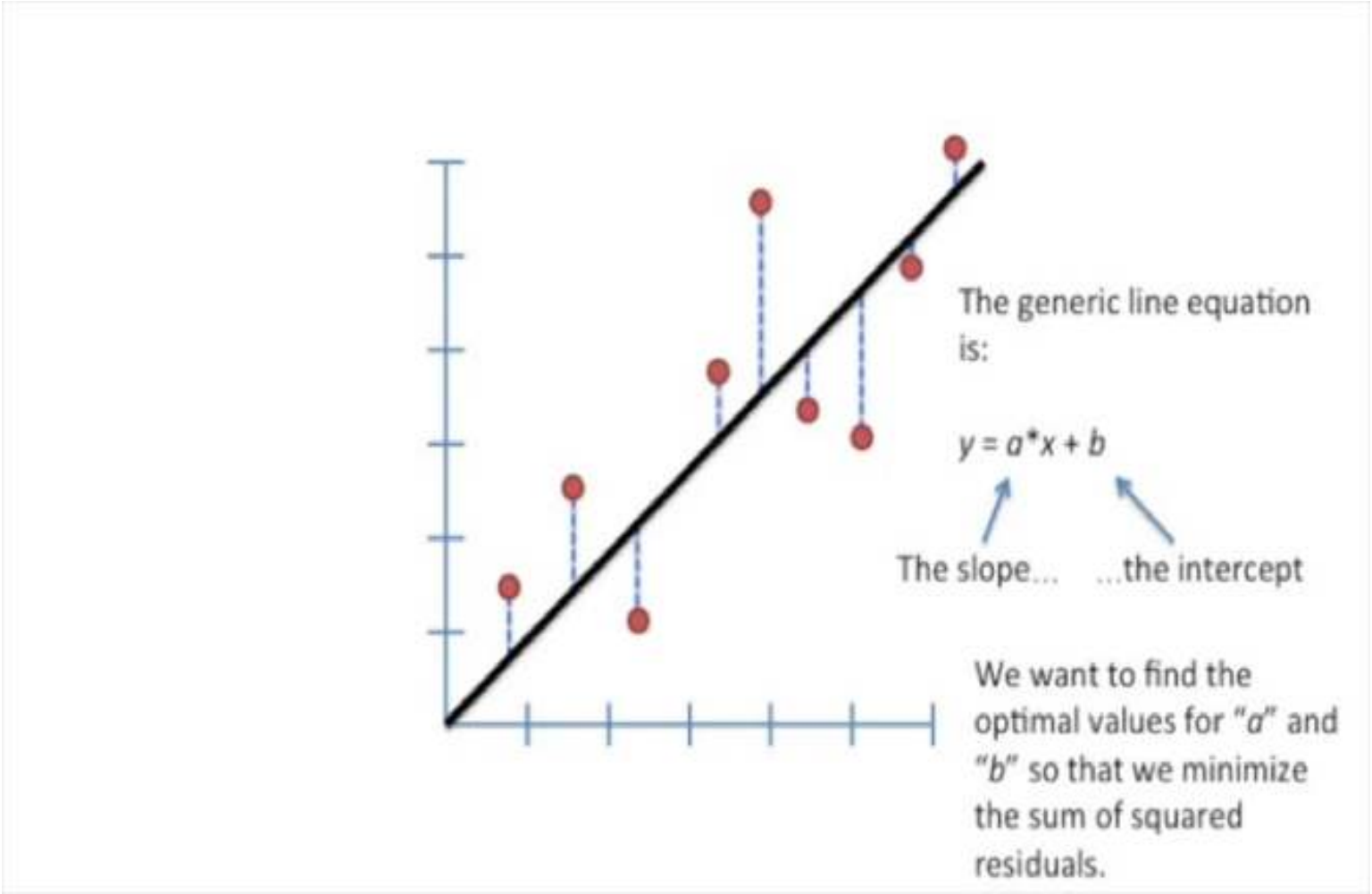
How to choose the best line?



How to choose the best line?



How to choose the best line?

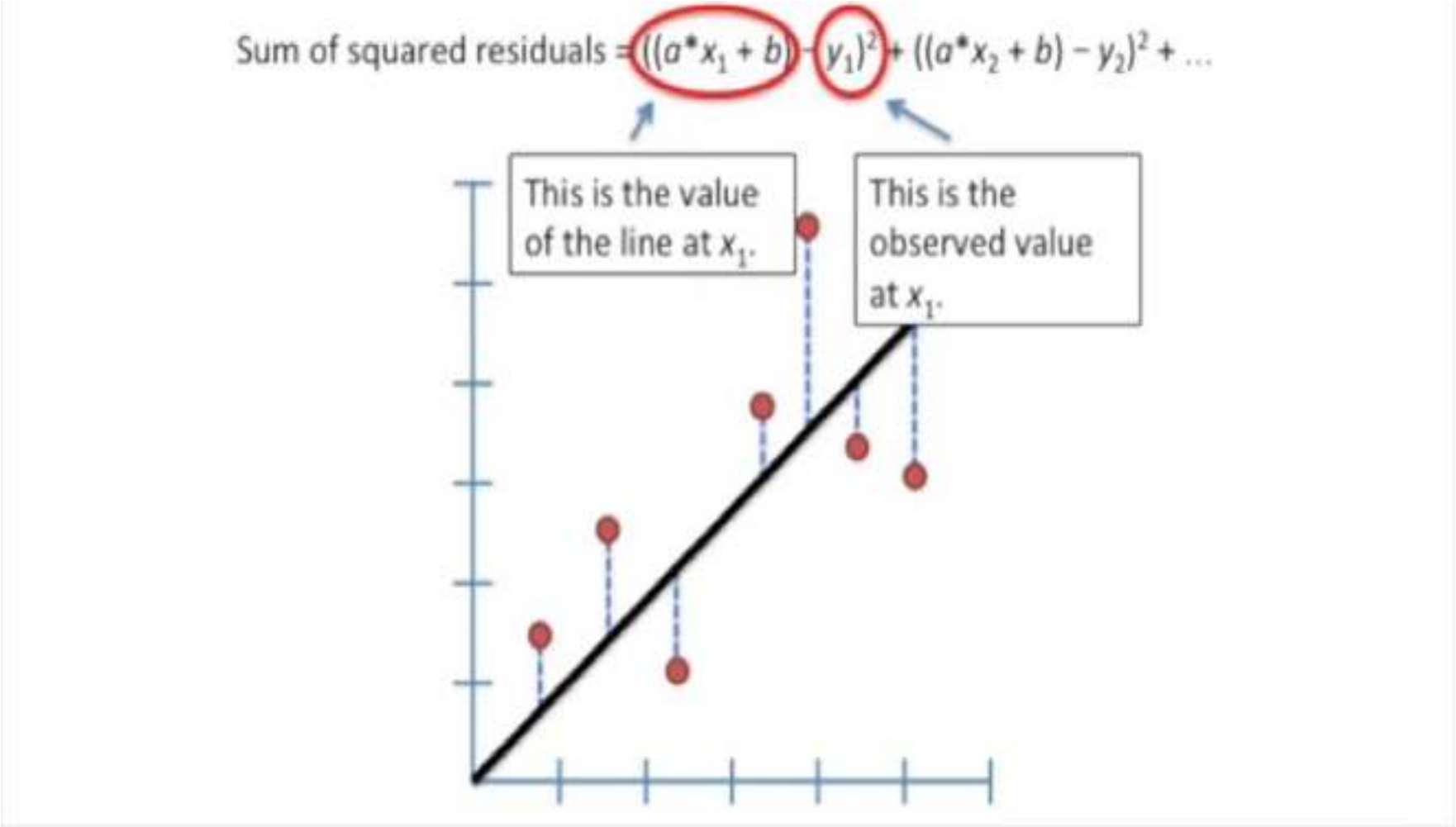


Least Squares

Given a linear model f, the formula for computing the sum of squared errors is:

$$\text{SSE} = \sum_{i=1}^{N} (y_i - f(x_i))^2$$

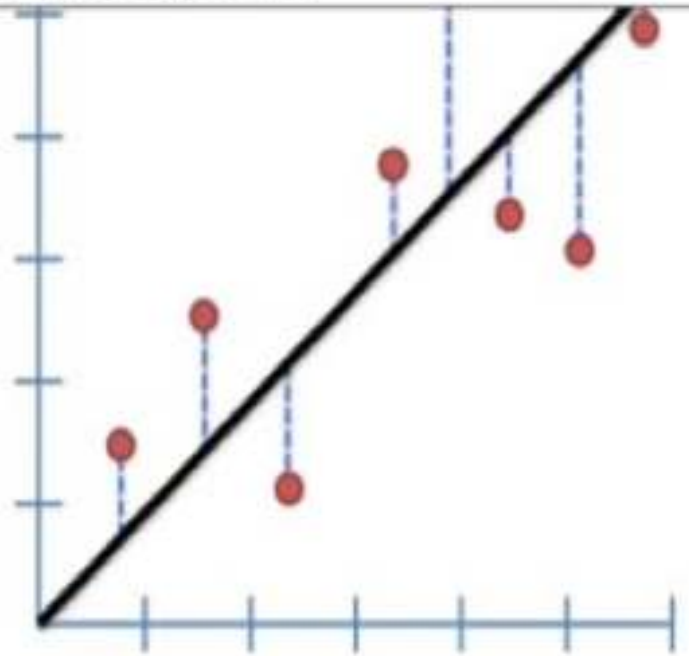
Least Squares



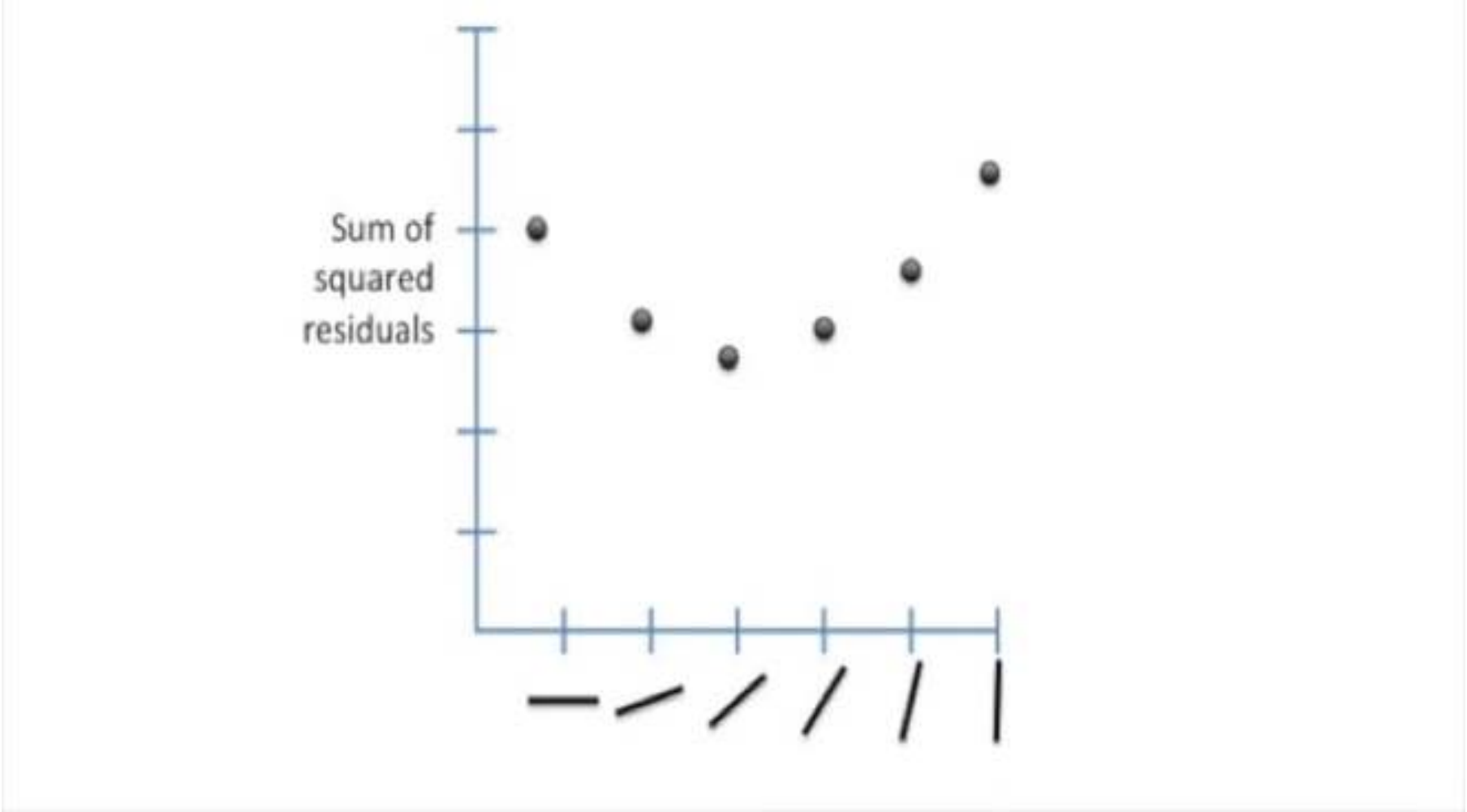
Least Squares

Sum of squared residuals = $((a*x_1 + b) - y_1)^2 + ((a*x_2 + b) - y_2)^2 + \ldots$

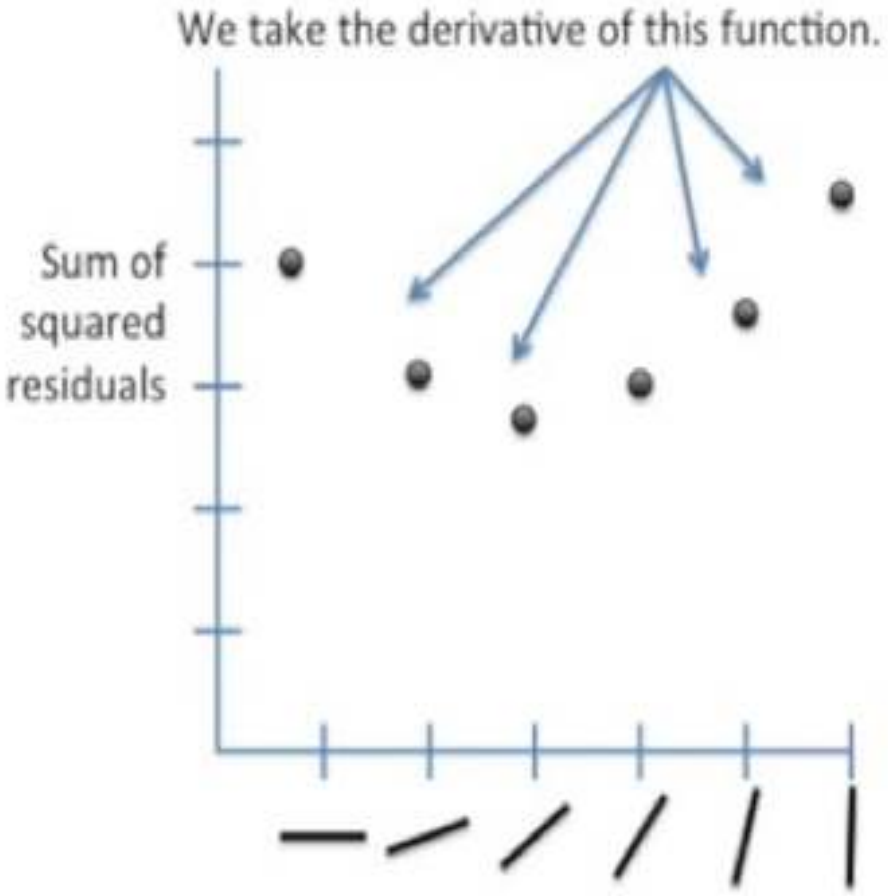
Since we want the line that will give us the smallest sum of squares, this method for finding the best values for "*a*" and "*b*" is called "Least Squares".



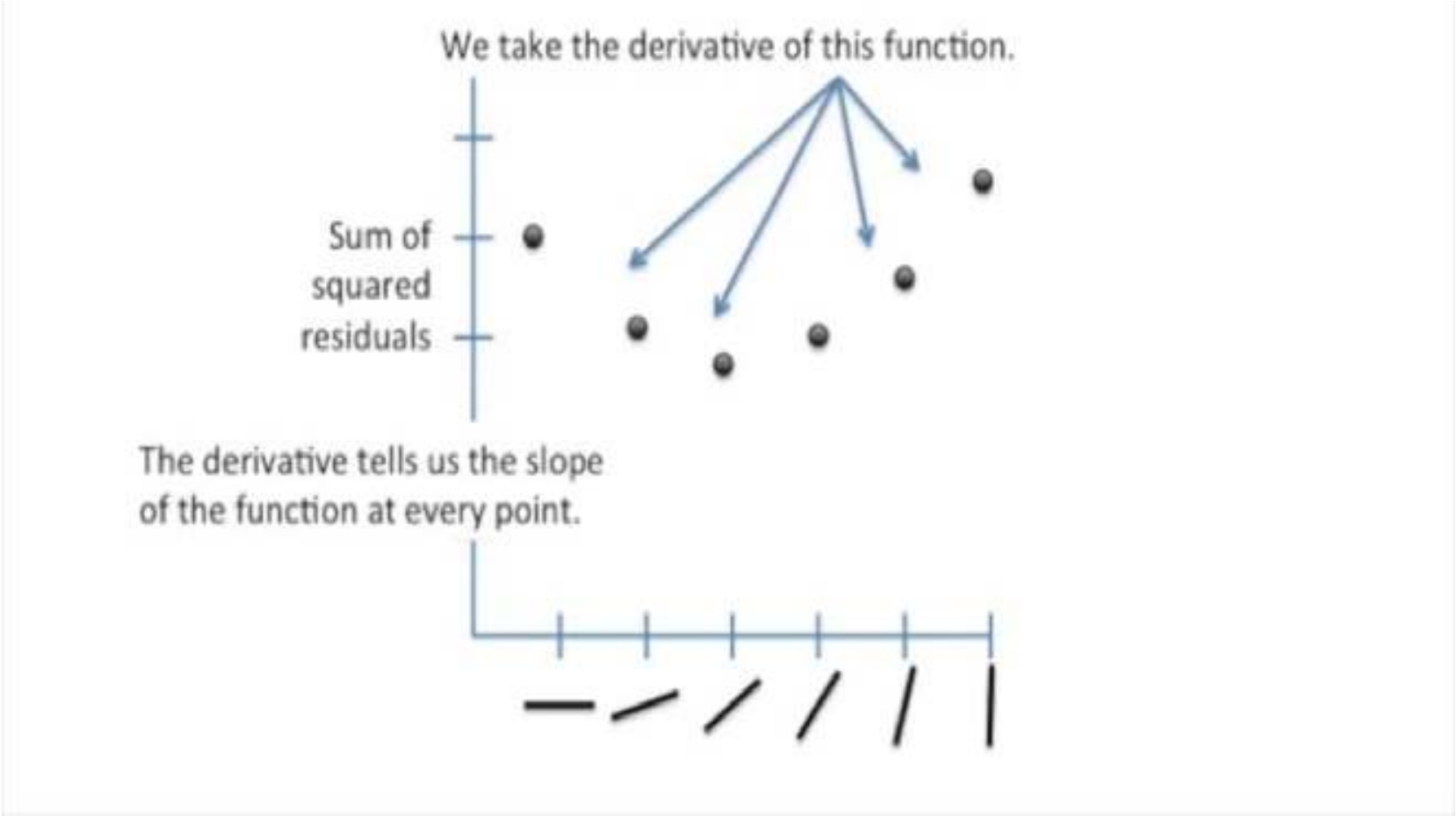
Least Squares



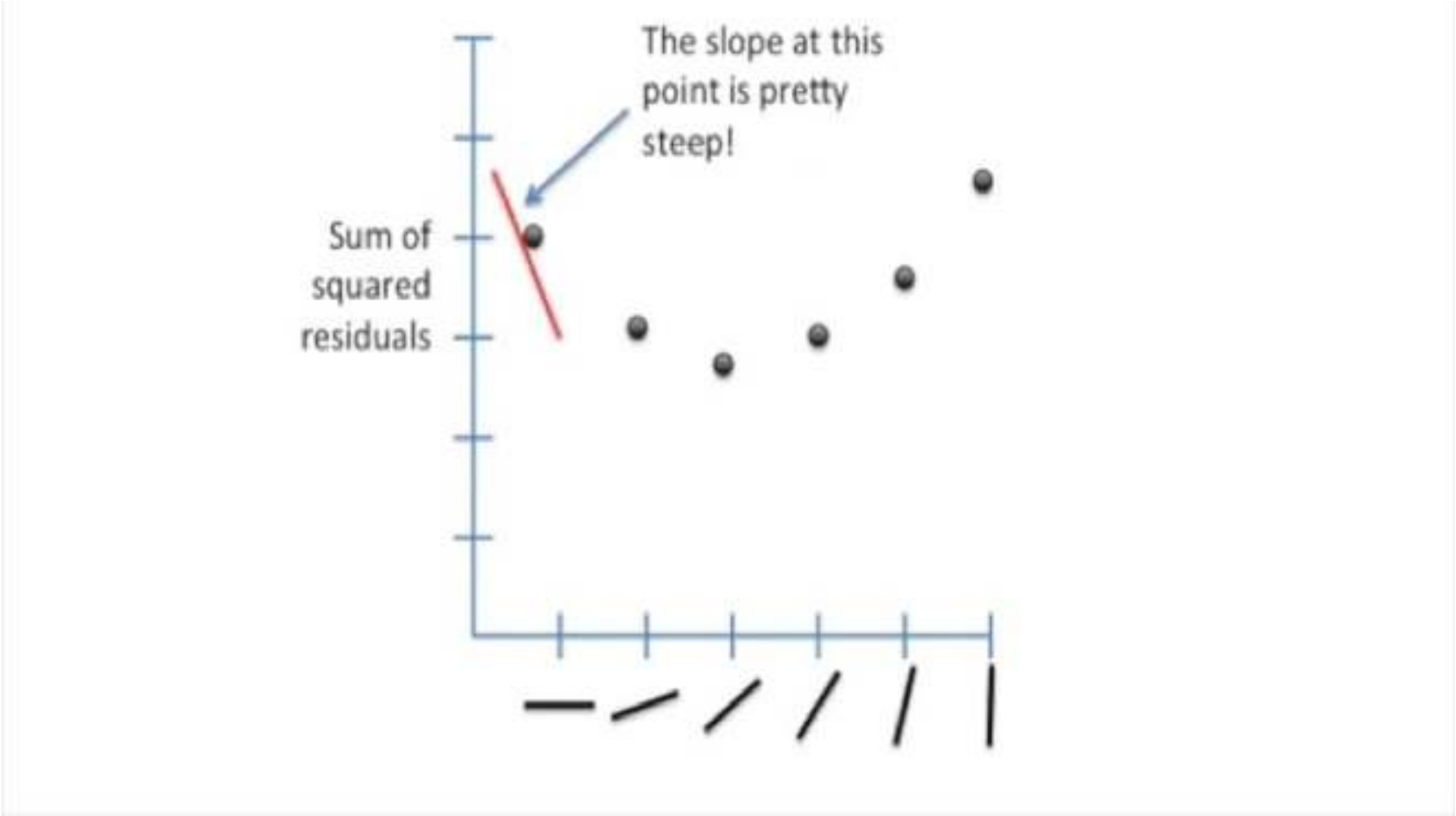
Least Squares



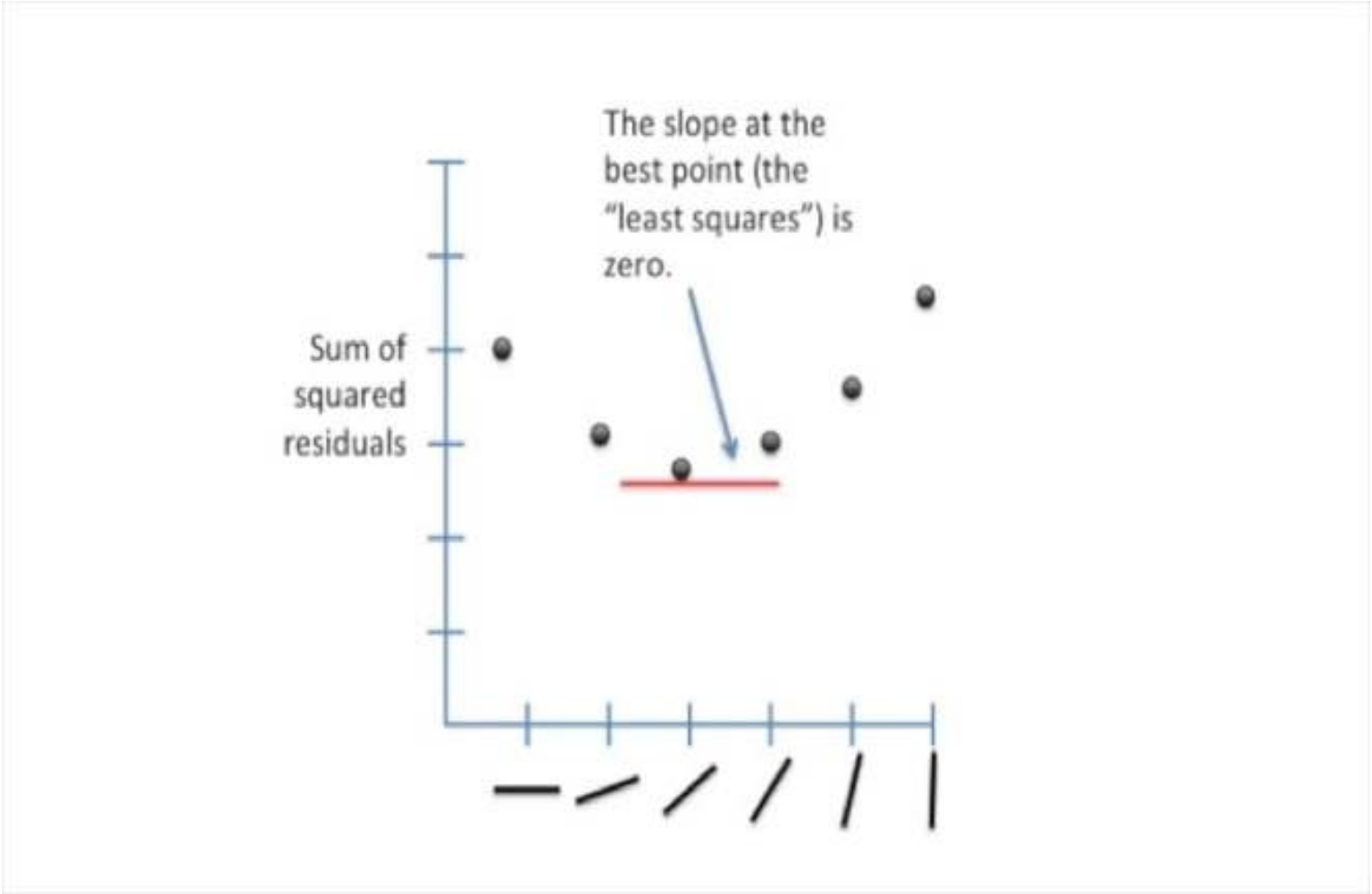
Least Squares



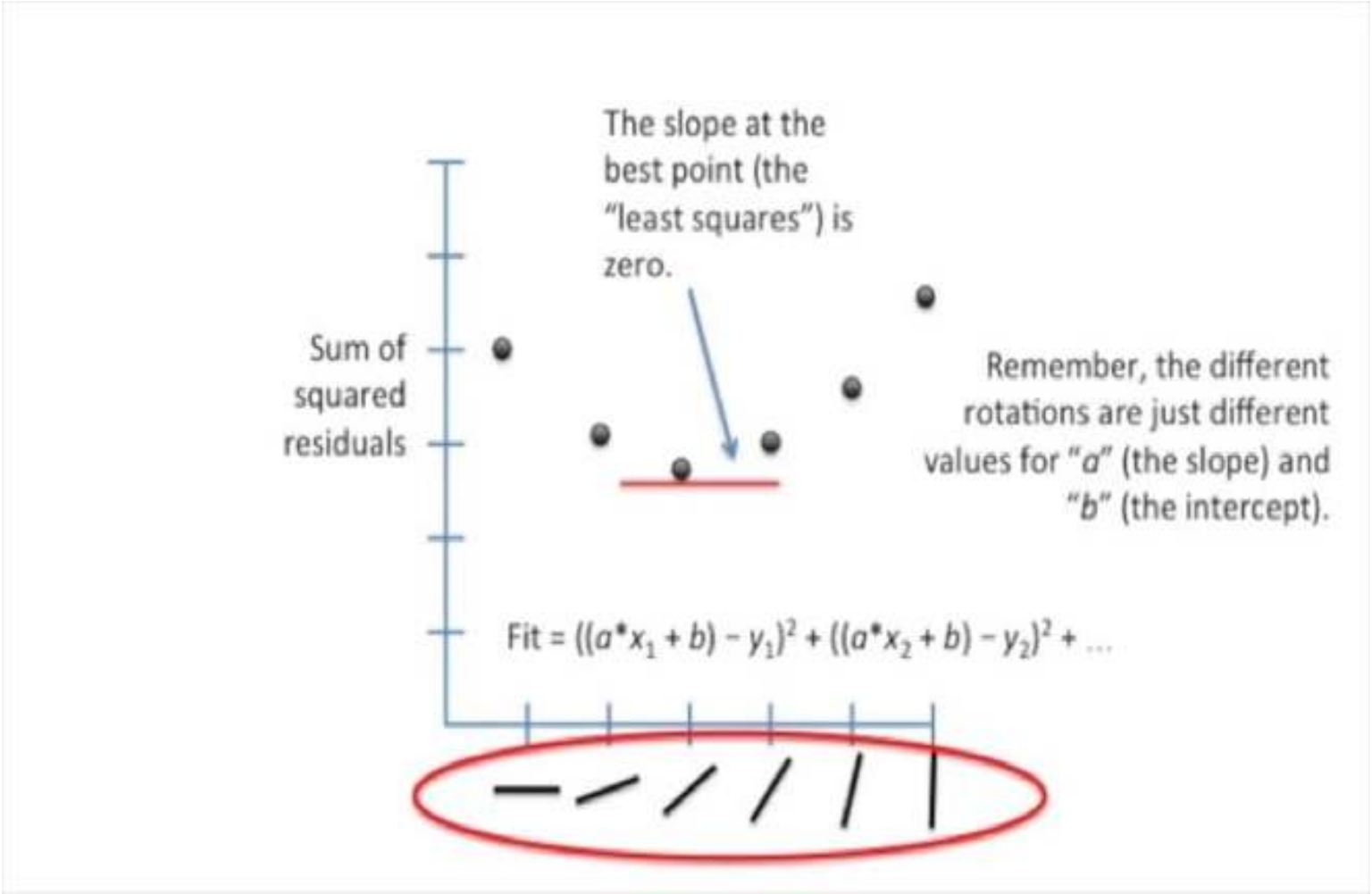
Least Squares



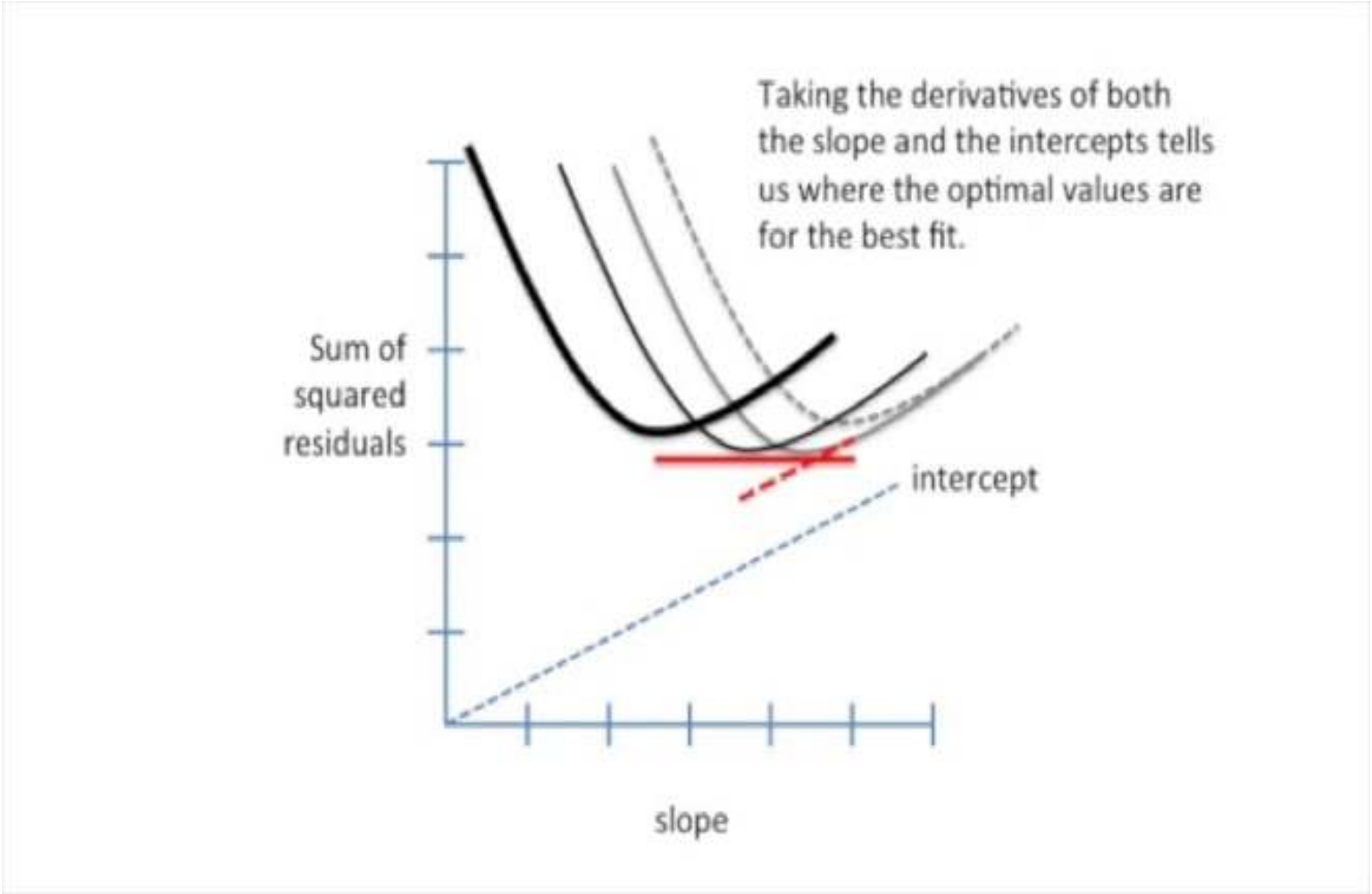
Least Squares



Least Squares



Least Squares



Solving Least Squares

We want to find optimal m, b such that the SSE is minimized:

$$(m^*, b^*) = \arg\min_{m,b} \sum_{i=1}^{N} (y_i - f(x_i))^2$$

This has a closed form solution obtained by setting the partial derivatives of SSE w.r.t. m and b to zero and solving:

$$m = \frac{N\sum(xy) - \sum x \sum y}{N\sum(x^2) - (\sum x)^2}, \quad b = \frac{\sum y - m\sum x}{N}$$

The solution is unique as long as the variance of x is not zero.

For large datasets, other optimization approaches like gradient descent can be preferred (see future lectures)

Why squared error?

Squared error:

- Penalizes **large errors more**
- Is **smooth and differentiable**
- Leads to a **closed-form solution** for linear regression

This is why Least Squares is so widely used.

Mean Squared Error

Mean Squared Error (MSE):

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

So $\text{MSE} = \frac{\text{SSE}}{N}$

Minimizing MSE or SSE gives the **same solution.**

Mean Absolute Error

Mean Absolute Error (MAE):

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|$$

- MAE is more robust to outliers
- But it is not differentiable at 0
- Therefore it does not yield a closed-form solution like Least Squares

Although MAE has no closed-form solution, it can still be optimized efficiently using **numerical methods**.

The R^2 score

MSE, SSE and MAE depend on the **scale** of y. To measure **relative** quality of a model, we use the **coefficient of determination** R^2:

$$R^2 = 1 - \frac{\text{SSE}}{\text{SST}}$$

Where, the **model error** is the SSE, the baseline error (predicting the mean only) is: $\text{SST} = \sum_{i=1}^{N}(y_i - \bar{y})^2$.

- $R^2 = 1$ → perfect predictions
- $R^2 = 0$ → no better than predicting the mean
- $R^2 < 0$ → worse than predicting the mean

Adding more predictors never decreases R2, even if they are irrelevant. To compensate, adjusted R2 penalizes model complexity. R2 can be misleading for general nonlinear models.

Linear Regression

Multiple

Multiple Regression (Two Variables)

Now suppose our dataset is composed of:

- two feature columns: x_1, x_2 (independent variables)
- one target column: y (dependent variable)

$\mathbf{x}_1$	$\mathbf{x}_2$	y
x_{11}	x_{12}	y_1
x_{21}	x_{22}	y_2
$\dots$	$\dots$	$\dots$
x_{N1}	x_{N2}	y_N

We want to predict each y_i given the corresponding pair $\left(x_{i1}, x_{i2}\right)$.

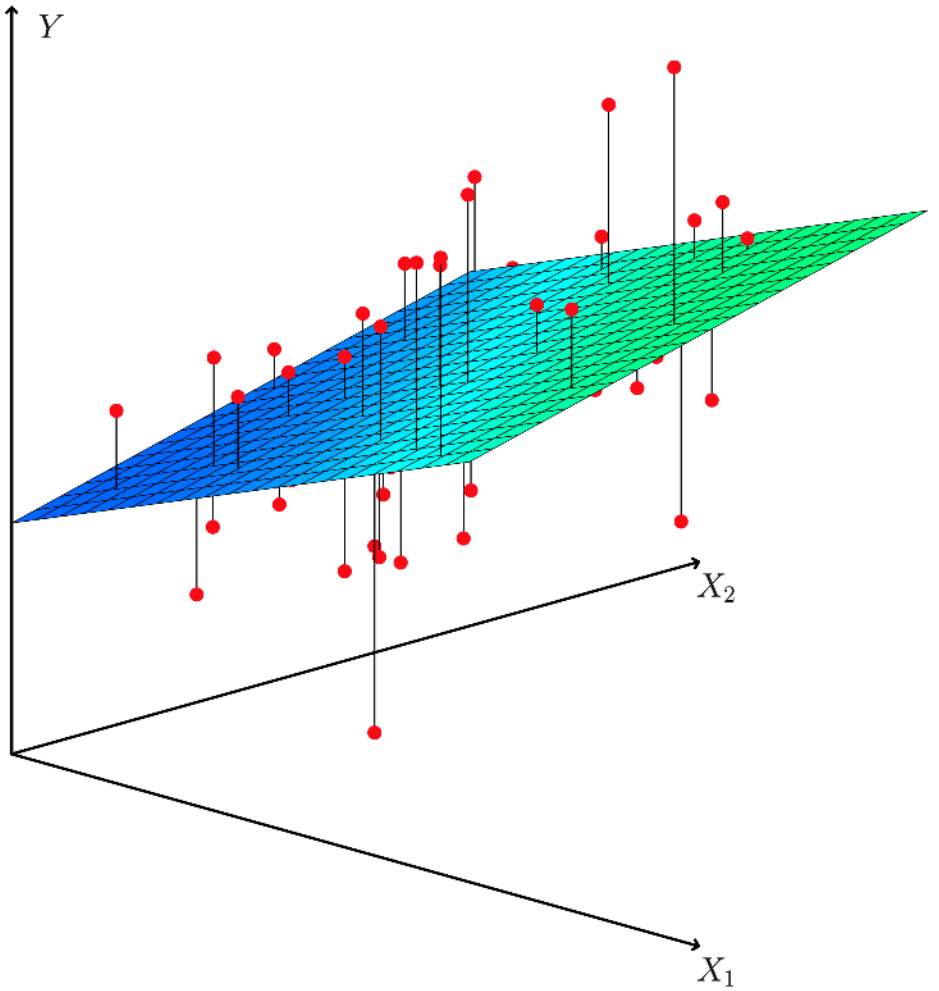
Multiple Regression (Two Variables)

For each sample:

$$\hat{y}_i = w_1 x_{i1} + w_2 x_{i2} + b$$

where w_1, w_2 are the weights and b is the bias.

We want to find the parameters w_1, w_2, b such that $y_i \approx \hat{y}_i$.



Multiple Regression (General Case)

In the general case, with a dataset containing M features, each row is a vector $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{iM}) \in \mathbb{R}^M$

We predict:

$$\hat{y}_i = w_1 x_{i1} + w_2 x_{i2} + \cdots + w_M x_{iM} + b = \boxed{\sum_{j=1}^{M} w_j x_{ij} + b}$$

where w_j are the weights and b is the bias.

Least squares extends directly to multiple regression

Interpreting Coefficients

For multiple regression, w_j indicates the change in y per unit change in x_j, holding all other variables constant.

Input scale matters, so feature-wise standardization allows a better comparison.

$$x'_j = \frac{x_j - \mu_j}{\sigma_j}$$

Benefits:

- Faster convergence
- Comparable coefficients
- Necessary for Ridge/Lasso

Assumptions of Linear Regression

A linear regression model assumes:

- **Linearity:** linear in parameters
- **Independent errors**: The error terms are statistically independent across observations
- **Homoscedasticity:** constant variance of errors
- **No strong multicollinearity**: predictors are not nearly exact linear combinations of each other
- **Normal residuals*** (for statistical inference): the errors are normally distributed

Normality is required for valid hypothesis tests and confidence intervals, but not for point prediction.

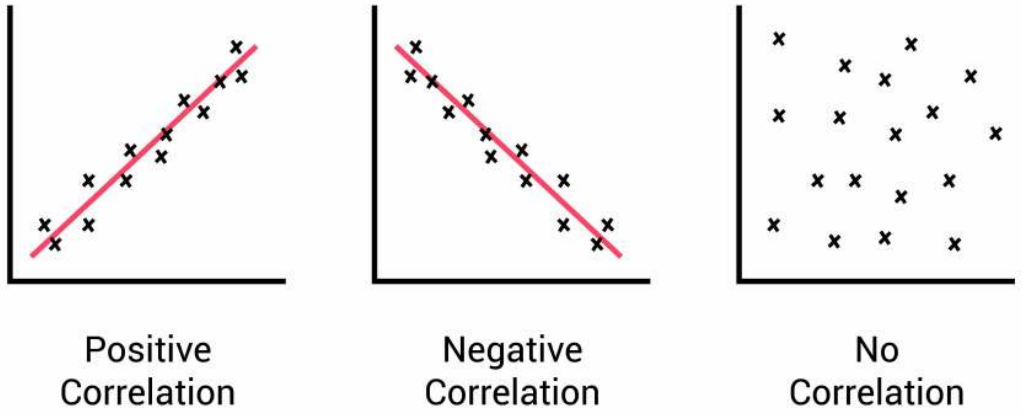
Multicollinearity

When predictors are highly correlated:

- Coefficients become unstable
- Large variance in estimates
- Difficult interpretation

Mitigation:

- Ridge regression
- Feature selection
- PCA



Regularizations

Some linear regression problems can benefit from **regularization**, i.e., discouraging large coefficient values.

- **Ridge** regression is a regularization method for ill-posed problems, particularly useful for mitigating multicollinearity.
- **Lasso** (Least Absolute Shrinkage and Selection Operator) performs both variable selection and regularization to enhance the prediction performance and interpretability of the resulting statistical model.

Regularizations

For the single variables case:

$$\text{Ridge: } (m^*, b^*) = \arg\min_{m,b} \sum_{i=1}^{N} (y_i - f(x_i))^2 + \lambda m^2$$

$$\text{Lasso: } (m^*, b^*) = \arg\min_{m,b} \sum_{i=1}^{N} (y_i - f(x_i))^2 + \lambda |m|$$

(This is mainly useful when you have many independent variables.)

Regularization

For multiple variables:

$$\text{Ridge: } \arg\min_{\mathbf{w},b} \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{M} w_j^2$$

$$\text{Lasso: } \arg\min_{\mathbf{w},b} \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{M} |w_j|$$

Regularization shrinks coefficients and reduces overfitting.

Classification

Via Regression

Classification

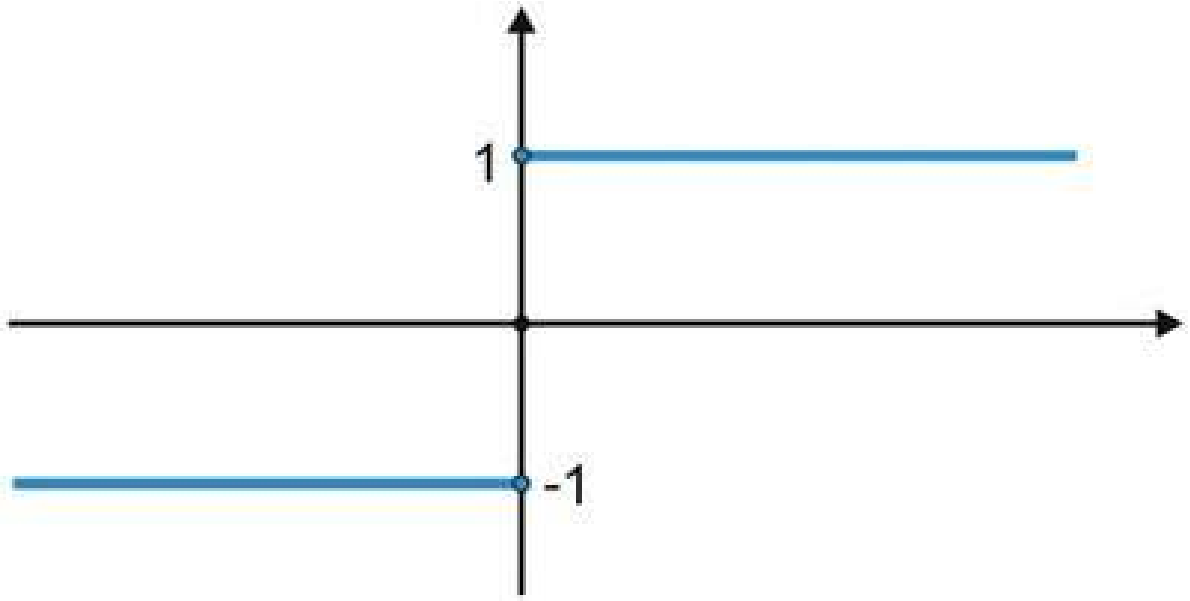
The aim of classification is to predict a **categorical** variable for each instance in a dataset.

$$f(\mathbf{X}) = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)] = [\hat{y}_1, \ldots, \hat{y}_N] = \hat{\mathbf{y}}$$
$$\hat{\mathbf{y}} \in \{1, \ldots, C\}^N$$

Binary Classification

The goal: assign a label $y \in \{-1, +1\}$ (or $\{0, 1\}$) to an input $\mathbf{x}$

Given a linear regression model $\sum_{j=1}^{M} w_j x_j + b$, the output can be transformed into a binary one, by applying a function to the output.



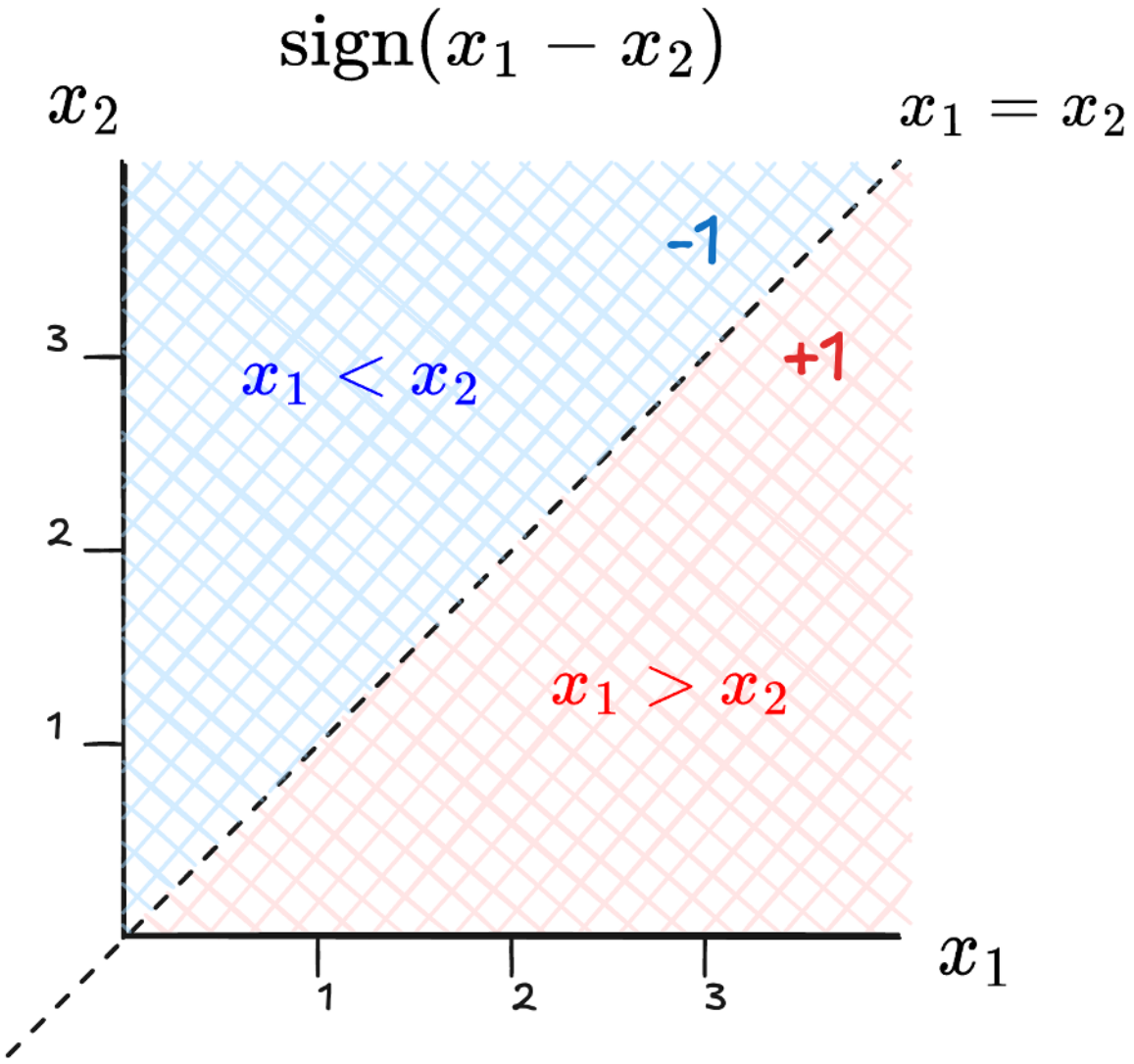
Binary Classification in 2D

In $\mathbb{R}^2$, $w_1 x_1 + w_2 x_2 + b$ defines a line that separates space into two regions.

As a simple classification rule, we can use the sign function, which returns $+1$ if $w_1 x_1 + w_2 x_2 + b \geq 0$; otherwise -1:

$$\hat{y} = \text{sign}(w_1 x_1 + w_2 x_2 + b)$$

For example, $\text{sign}(x_1 - x_2)$ is visualized on the right.



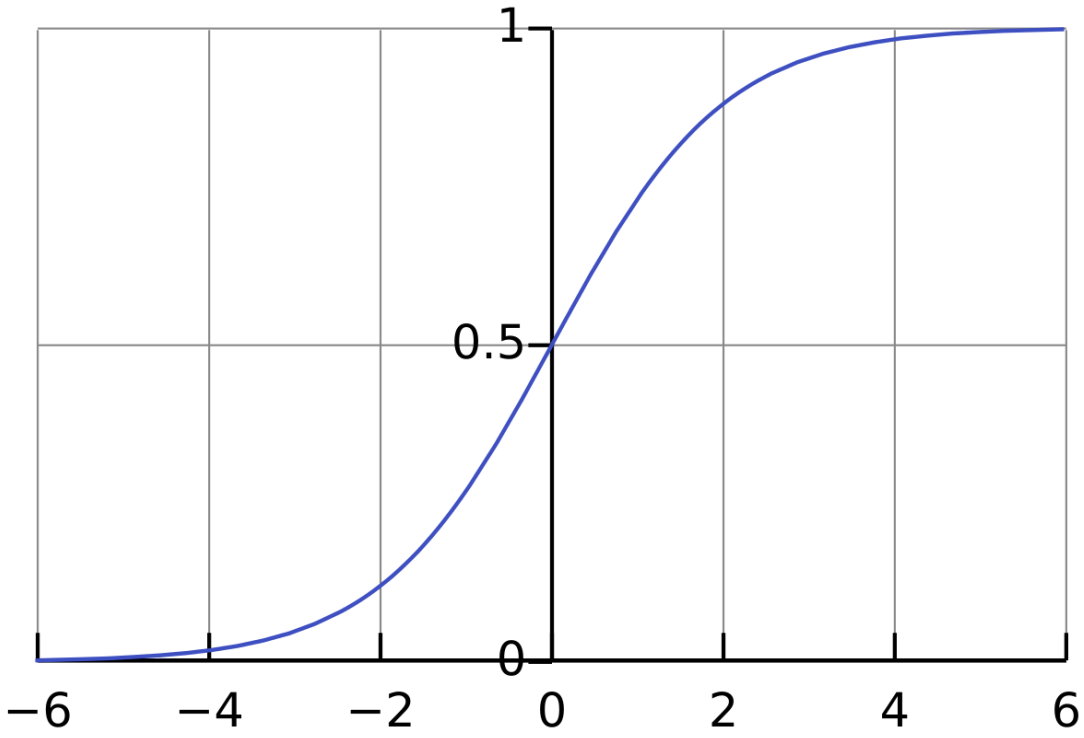
Logistic Regression

- Instead of a step function, use a smooth function
- Logistic (sigmoid) function:

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

- Interpreted as $P(y = 1 \mid x)$
- A threshold is used for classification:

$$\hat{y} = \begin{cases} 1 & \text{if } \sigma(z) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$



Logistic Regression

Logistic regression models probability using **log-odds:**

$$\log \frac{p(y=1|\mathbf{x})}{1-p(y=1|\mathbf{x})} = \sum_{j=1}^{M} w_j x_j + b \implies p(y=1|\mathbf{x}) = \sigma\left(\sum_{j=1}^{M} w_j x_j + b\right)$$

Parameters are estimated by maximizing the likelihood, equivalently minimizing the (binary) **cross-entropy** loss:

$$\ell(\hat{\mathbf{p}}, y) = -\log \hat{p}_y = -\left[y \log \hat{p} + (1-y)\log(1-\hat{p})\right]$$

Limitations

Binary Linear Classifiers can split the space only with a single straight line

Multiclass Classification

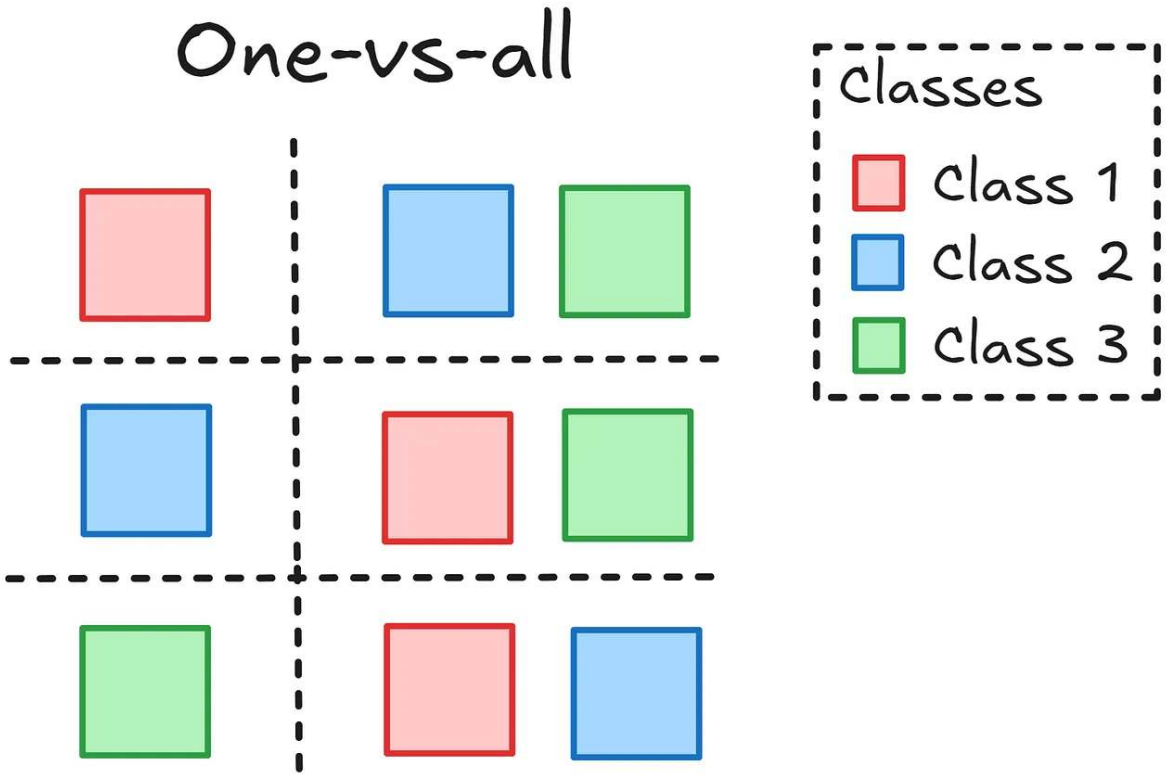
To build a multiclass classifier for $C > 2$ classes, there are several strategies:

- **Softmax Regression** (Multinomial logistic regression): use a regression model that outputs a vector of scores, one per class. These scores are converted into class probabilities with the **softmax** function (see future lectures for more details!).
- **One-vs-Rest**: Train one binary classifier for each class vs all the others
- **One-vs-One**: Train one binary classifier for each class vs each other class

One-vs-Rest (OvR)

One-vs-Rest (or One-vs-All) is a strategy where:

- we train a binary classifier for each unique class (C classifiers) against the rest in the multi-class dataset.
- at inference time we predict the output with each classifier, and choose the prediction with the highest probability

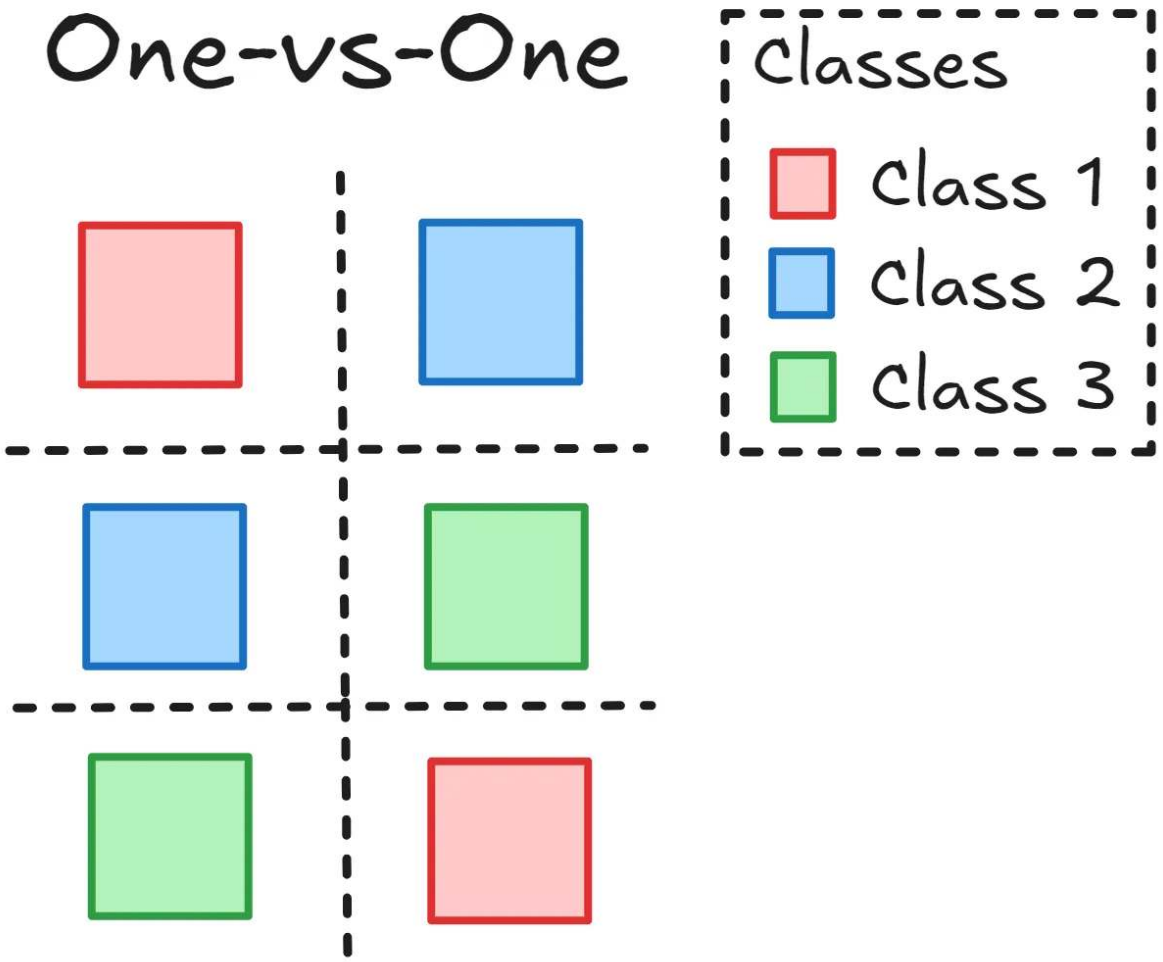


https://www.nb-data.com/p/one-vs-all-vs-one-vs-one-which-multi

One-vs-One (OvO)

One-vs-One is a strategy where:

- we train a binary classifier for each combination of classes ($\frac{C(C-1)}{2}$ classifiers)
- at inference time:
  - we predict the output with all classifiers (each classifier predicts one of its two classes);
  - each predicted class gets one vote;
  - we choose the class with the most votes



OvR vs OvO

OvR

Strengths

- Only C classifiers: faster training/inference
- Scales well to many classes
- Simple implementation
- Easy probability interpretation

Weaknesses

- Class imbalance (1 vs rest)
- Harder decision boundaries
- Often lower accuracy when classes overlap

OvO

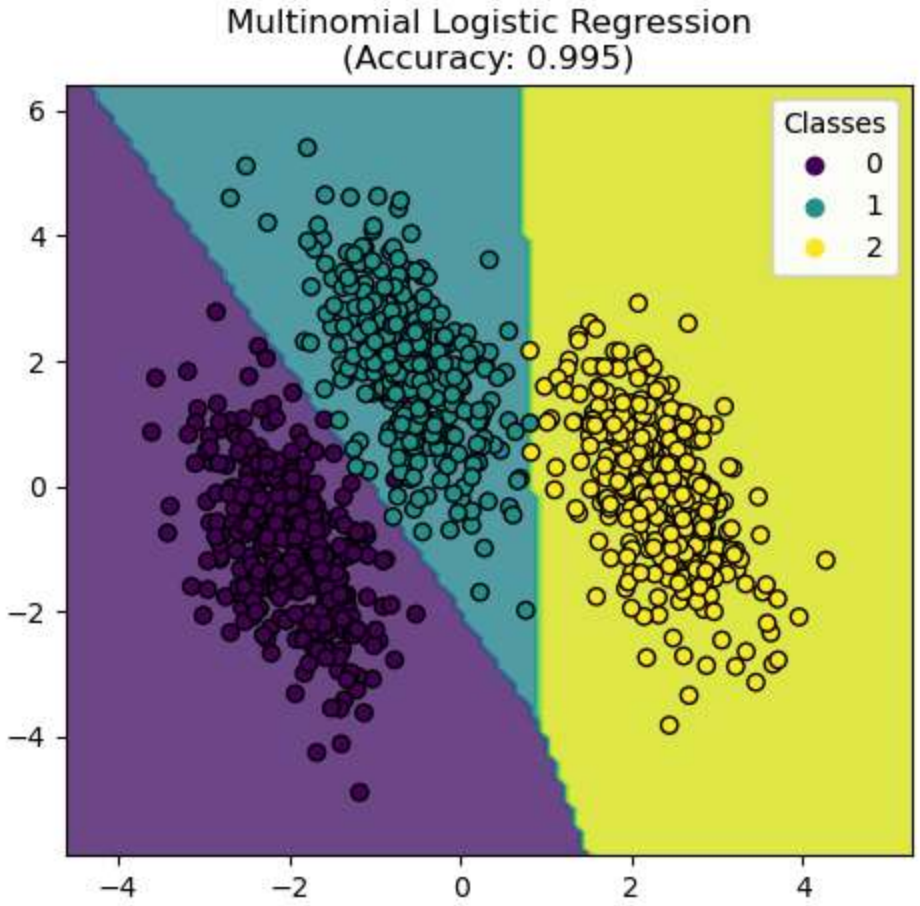
Strengths

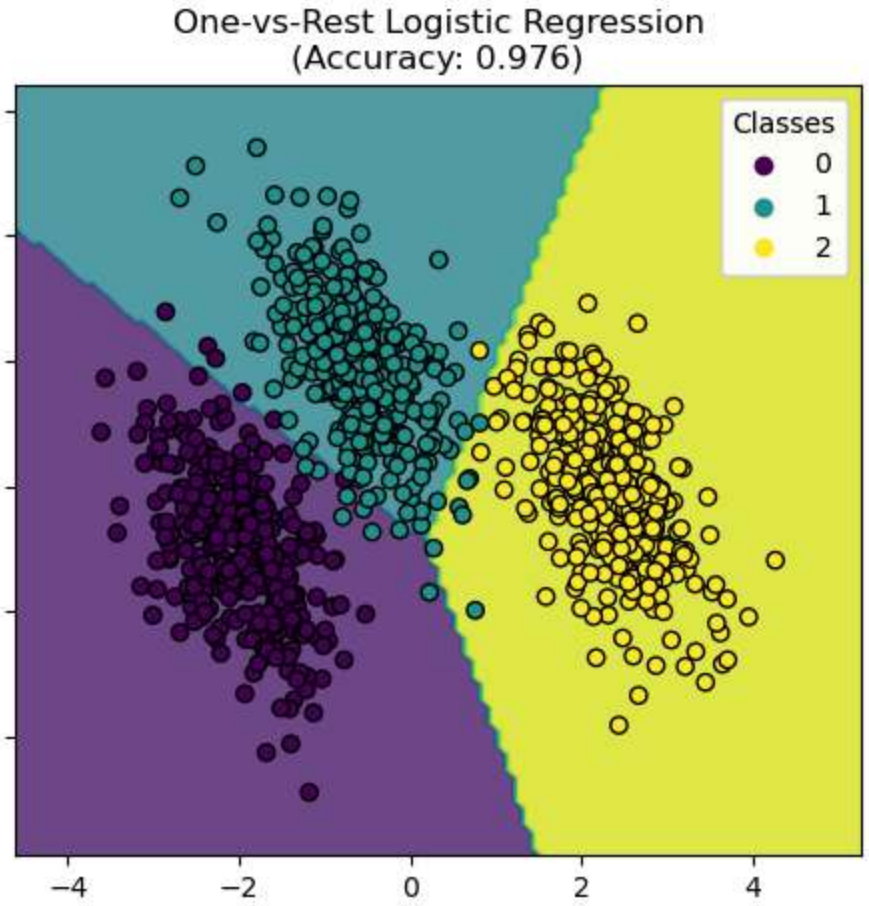
- Balanced binary problems
- Often higher accuracy with overlapping classes
- Good for complex boundaries

Weaknesses

- C(C-1)/2 classifiers: slower, more memory
- Slower inference
- Needs voting / tie-breaking

Decision Boundary





Linear Models for Unstructured Data

Using Feature Extraction

Motivation

Linear models for classification or regression (and the majority of the basic machine learning models) assume a **structured**, tabular, feature matrix:

$$\mathbf{X} \in \mathbb{R}^{N \times M}$$

Unstructured data such as images, audio, raw signals, or text do not naturally conform to this representation.

Direct application of linear models on raw representations often leads to:

- Extremely high dimensional feature spaces
- Poor inductive bias
- Weak generalization performance

Why Raw Representations Fail

Examples:

- Images: using raw pixels as features ignores spatial structure
- Time series: using raw timestamps ignores temporal dynamics
- Audio: raw waveform values are highly redundant
- Text: raw strings are not numerical

Linear models do not capture local correlations, invariances, or hierarchical structure inherent in unstructured data.

Core Idea

The solution is **feature extraction**.

We transform unstructured data into structured numerical summaries:

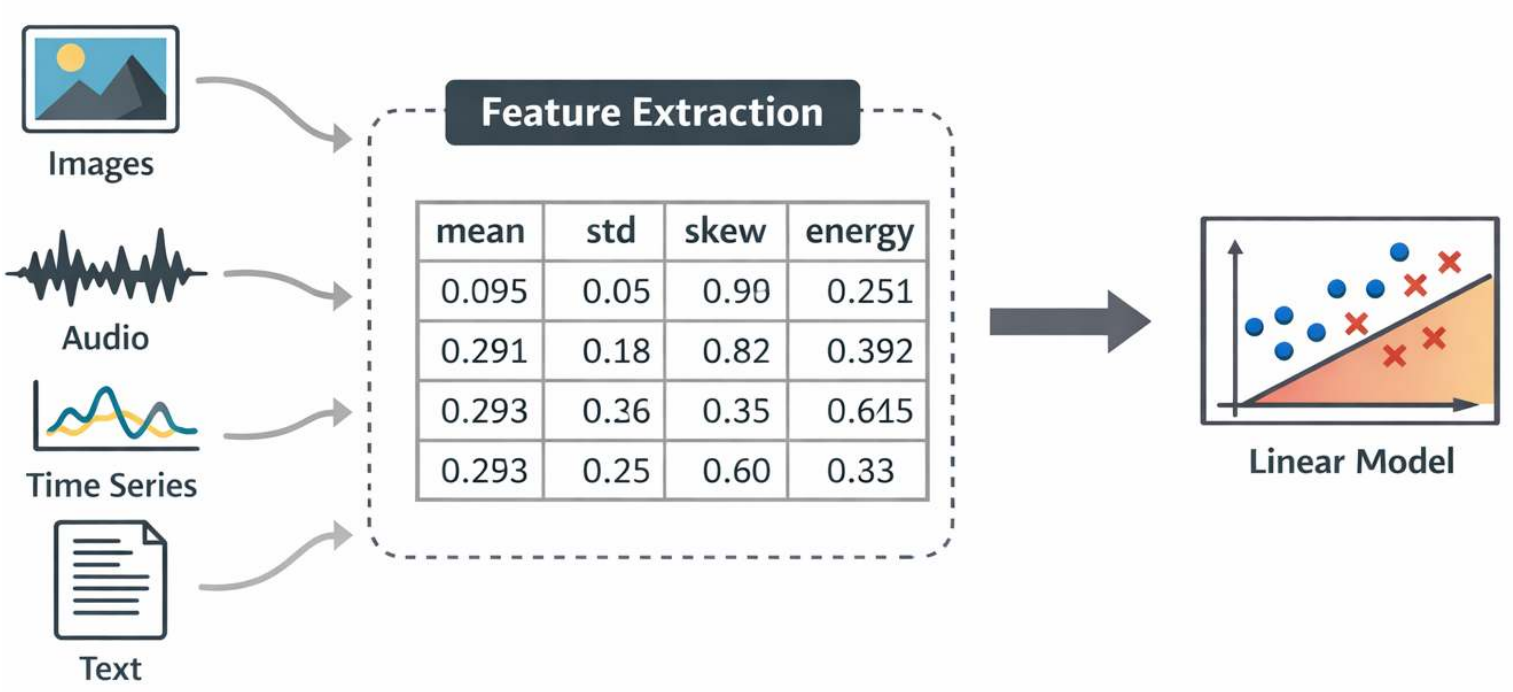
Unstructured data → Feature extraction → Structured feature matrix → Linear model

This allows:

- Dimensionality reduction
- Noise reduction
- Improved interpretability
- Compatibility with classical ML methods

Statistical Features (General)

Generic statistical features summarize distributional properties of the input instance: Mean, Standard deviation or variance, Median, Minimum and maximum, Quantiles, Skewness, Kurtosis, Energy or norm.



Data Specific Features

Depending on the kind of data, features may include:

- **Time series**: Autocorrelation coefficients, Peak statistics, Trend coefficients, Fourier or spectral components
- **Audio**: MFCCs, Spectral centroid, Zero crossing rate
- **Images**: Intensity or color histograms, Edge density, Texture descriptors such as Local Binary Patterns
- **Text**: Bag of Words, TF-IDF, Word or sentence embeddings

Some of these features can be extracted also from subparts of the data (e.g., a certain time series window, or a certain image area)

The objective is not to use raw unstructured data directly, but to encode informative structure in a compact numerical form.