Machine Learning for NLP

Supervised Learning

Aurélie Herbelot

2021

Centre for Mind/Brain Sciences
University of Trento

Supervised learning

- *Supervised:* the system learns a function mapping an input to an output, using training examples.
- Supervised learning mostly falls into classification (week 1) and regression (today).

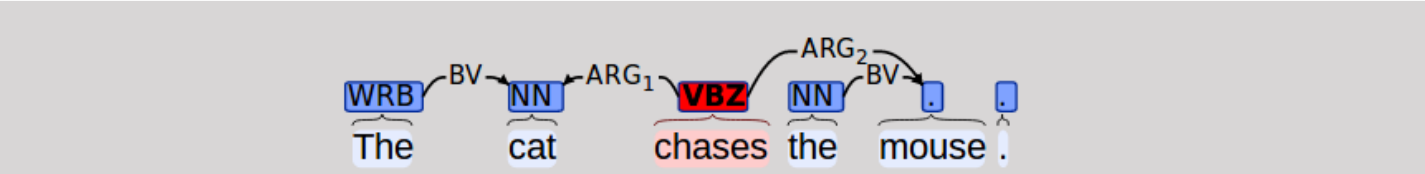
Linear Regression

Difference between regression and classification

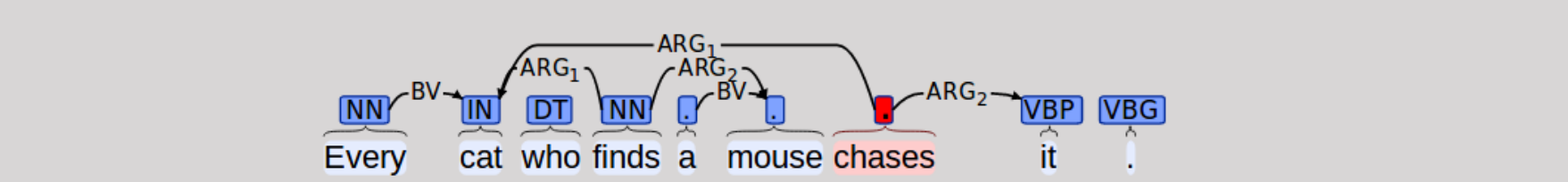
- Naive Bayes is a *classification* algorithm: given an input, we want to predict a discrete class, e.g.:
  - Austen vs Carroll vs Shakespeare;
  - bad vs good (movie review);
  - spam vs not spam (email)...
- In *regression*, given an input, we want to predict a continuous value.

Linear regression example

- Let's imagine that reading speed is a function of the structural complexity of a sentence.



58 edges



225 edges

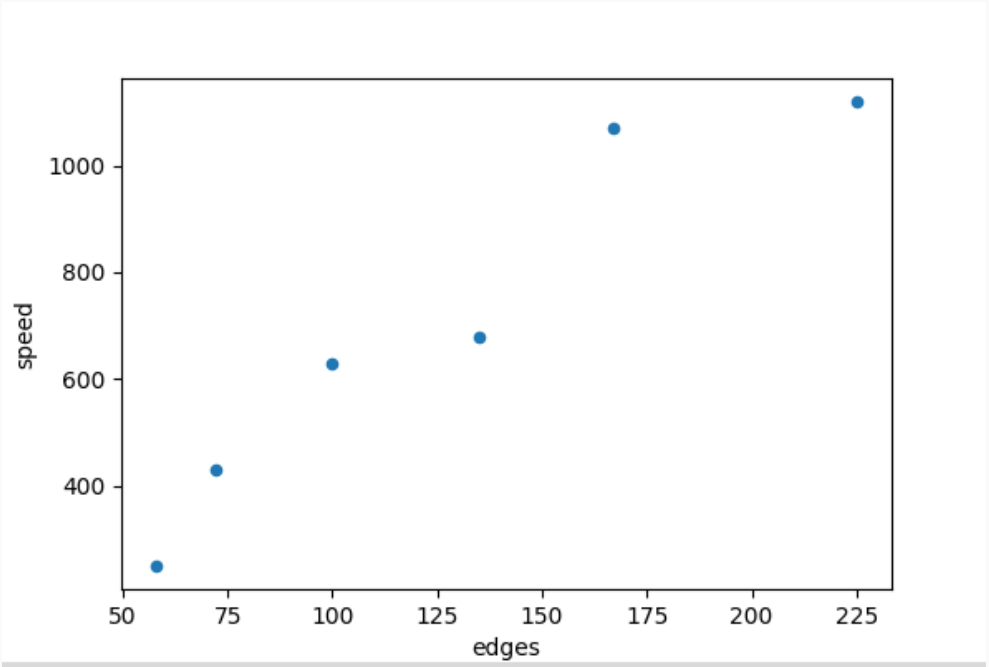
Parses from `http://erg.delph-in.net/logon`.
An 'edge' here is a *hypothetical* connection between two nodes of a parse tree.

Linear regression example

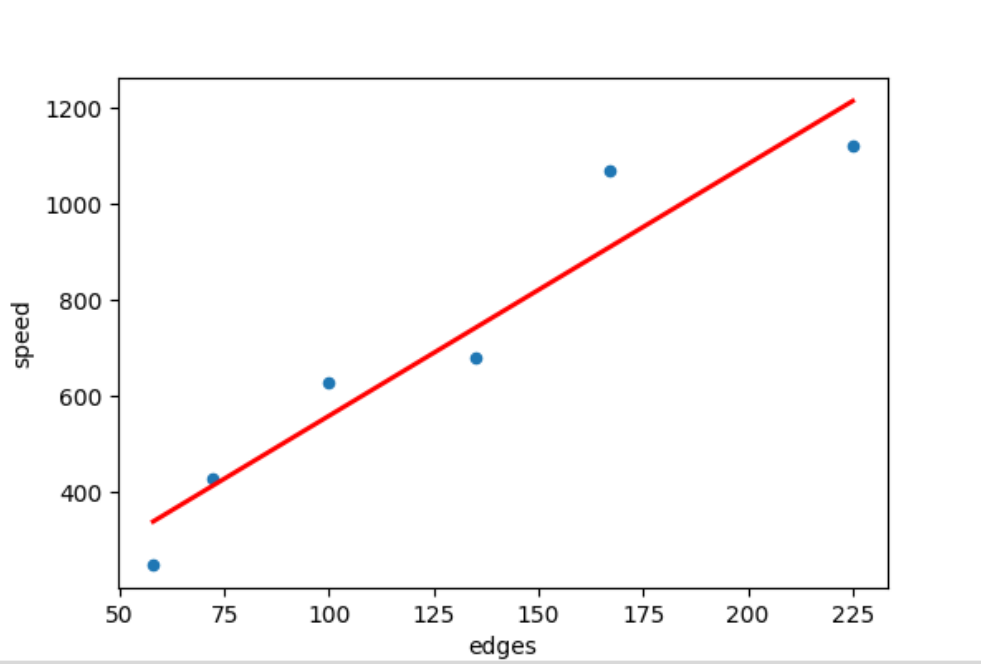
	#Edges	Speed (ms)
Sentence 1	58	250
Sentence 2	100	720
Sentence 3	72	430
Sentence 4	135	1120
Sentence 5	225	1290
Sentence 6	167	1270

- Let's call the edges feature x and the speed output y.
- We want to predict the *continuous* value y from x.
- Example: if a new sentence has 240 edges, can we predict the associated reading speed?

Linear regression example

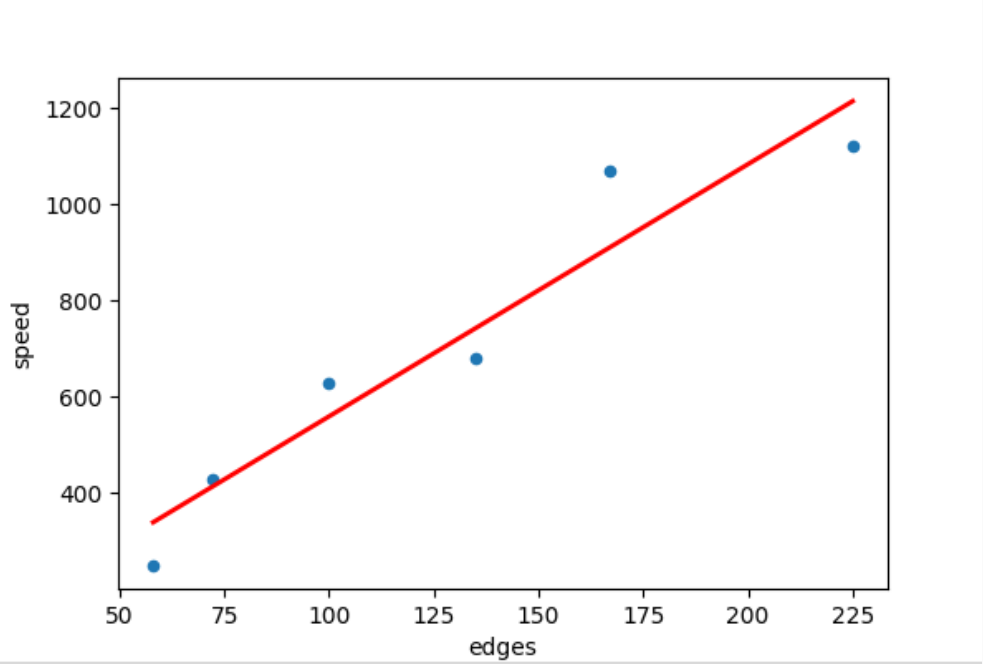


Linear regression example



We want to find a linear function that models the relationship between x and y.

Linear regression example

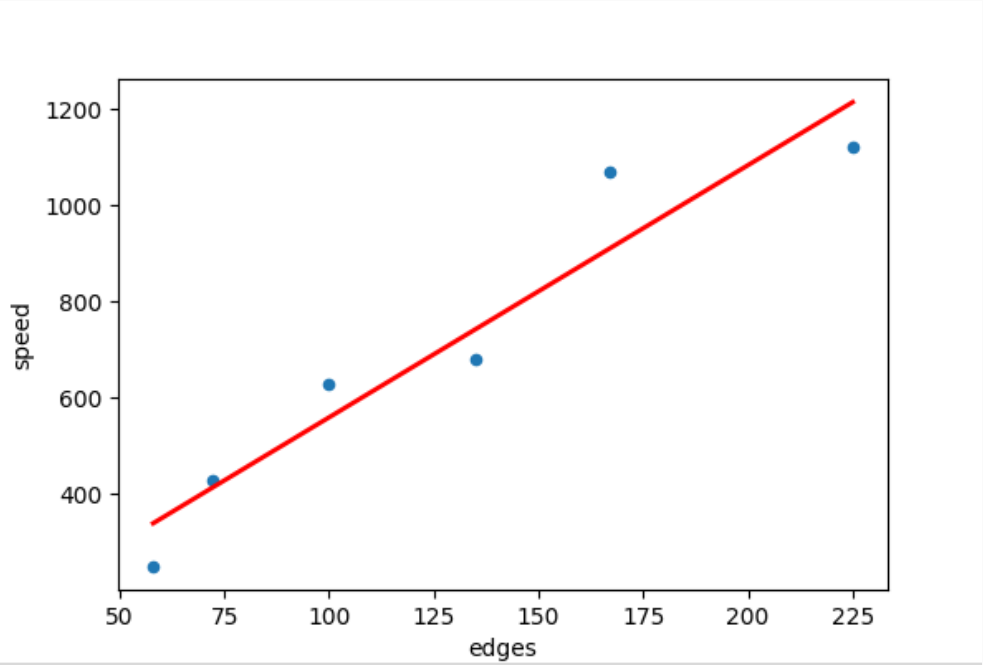


This linear function will have the following shape:

$$y = \theta_0 + \theta_1 x$$

θ_0 is the *intercept*, θ_1 is the *slope* of the line.

Linear regression example



Let's say our line can be described as $y = 36 + 5x$.

Now we can predict a reading speed for 240 edges:

$$speed = 36 + 5 \times 240 = 1236ms.$$

Evaluation: coefficient of determination r^2

How well did we do with our regression?
We can compute r^2 to find out:

$$r^2 = 1 - \frac{SSE}{SSTO}$$

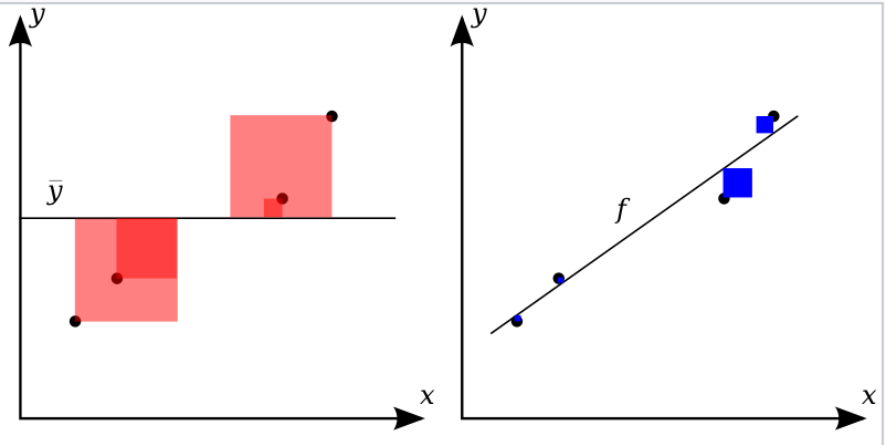
with SSTO the *total sum of squares*

$$SSTO = \sum_i (y_i - \bar{y})^2$$

(how much data points vary round their mean)

and SSE the *sum of squared errors*

$$SSE = \sum_i (y_i - \hat{y}_i)^2$$



By Orzetto - Own work, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=11398293

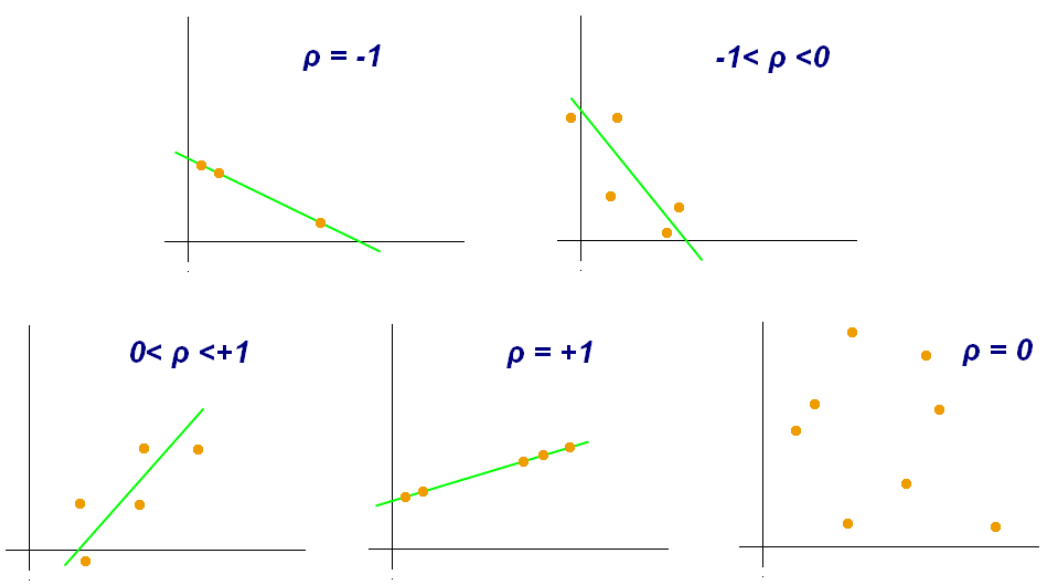
Evaluation: coefficient of determination r^2

- The result of r^2 can be interpreted as a percentage. This is how much of the variance is accounted for by our regression line.
- **Example:** a r^2 value of 0.52 means that 52% of the variation of y (e.g. reading speed) can be explained by x (e.g. syntactic complexity).

Evaluation: coefficient of determination r^2

r^2 is also simply the square of r,
the Pearson correlation coefficient.

Can you now see why r^2 is squared?



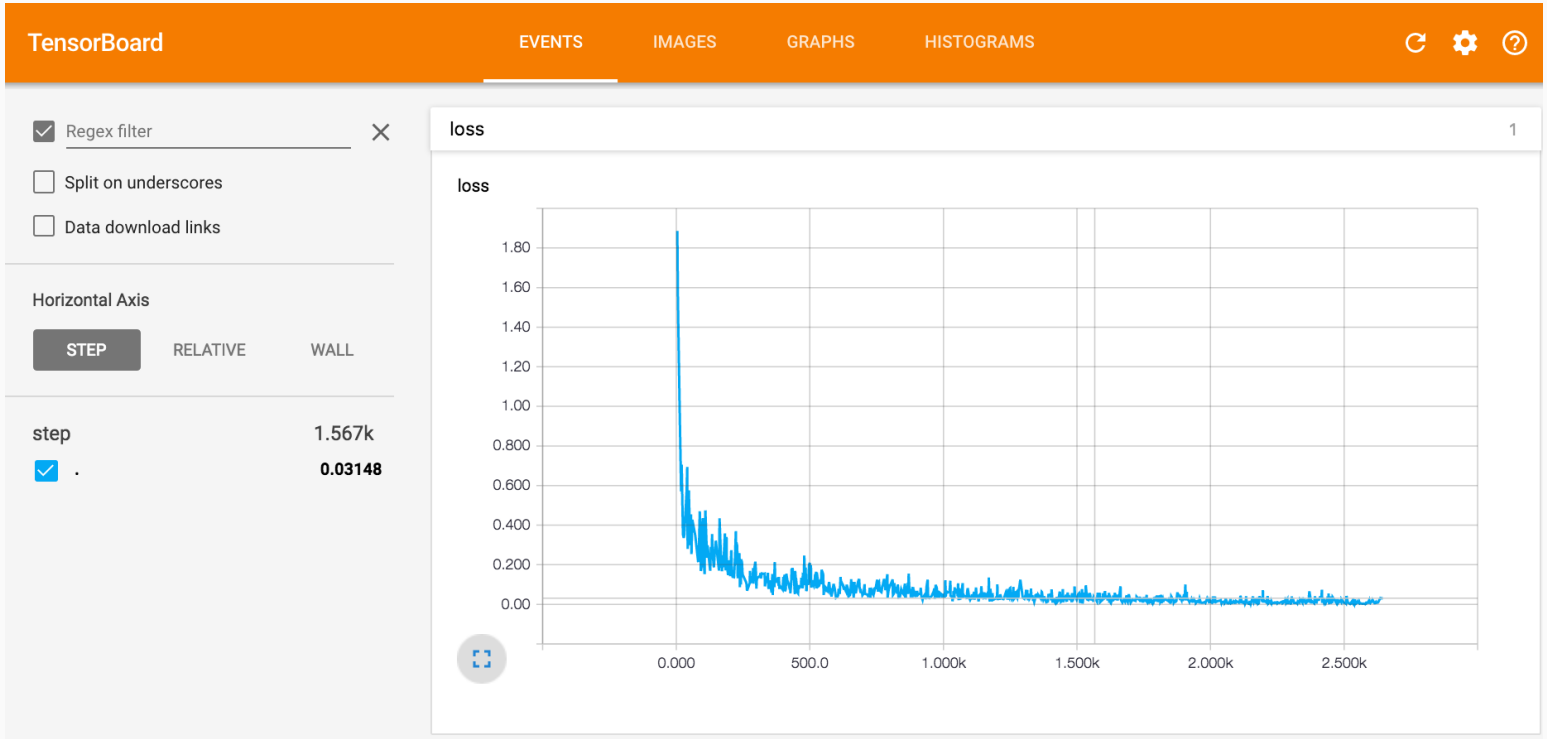
Evaluation: monitoring the loss in training

- How far is our line from the 'real' data points? This is the *loss / cost / error* of the function.
- Let's estimate θ_0 and θ_1 using the *least squares* criterion.
- This means our ideal line through the data will minimise the *sum of squared errors* (SSE). We will define our error as:

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

where N is our number of training datapoints, $\hat{y}_i$ is the model prediction for datapoint i, and y_i is the gold standard for i.

Evaluation: monitoring the loss in training



The Gradient Descent Algorithm

On determinism

- Machine Learning is not mathematics.
- We could get a solution to our regression problem by *deterministically* solving a system of linear equations.
- But often, solving things deterministically is very expensive computationally, so we hack things instead.
- The gradient descent algorithm is an efficient way to solve our regression problem. but it doesn't guarantee to find the best solution to the problem. It is *non-deterministic*.

Minimising the error function

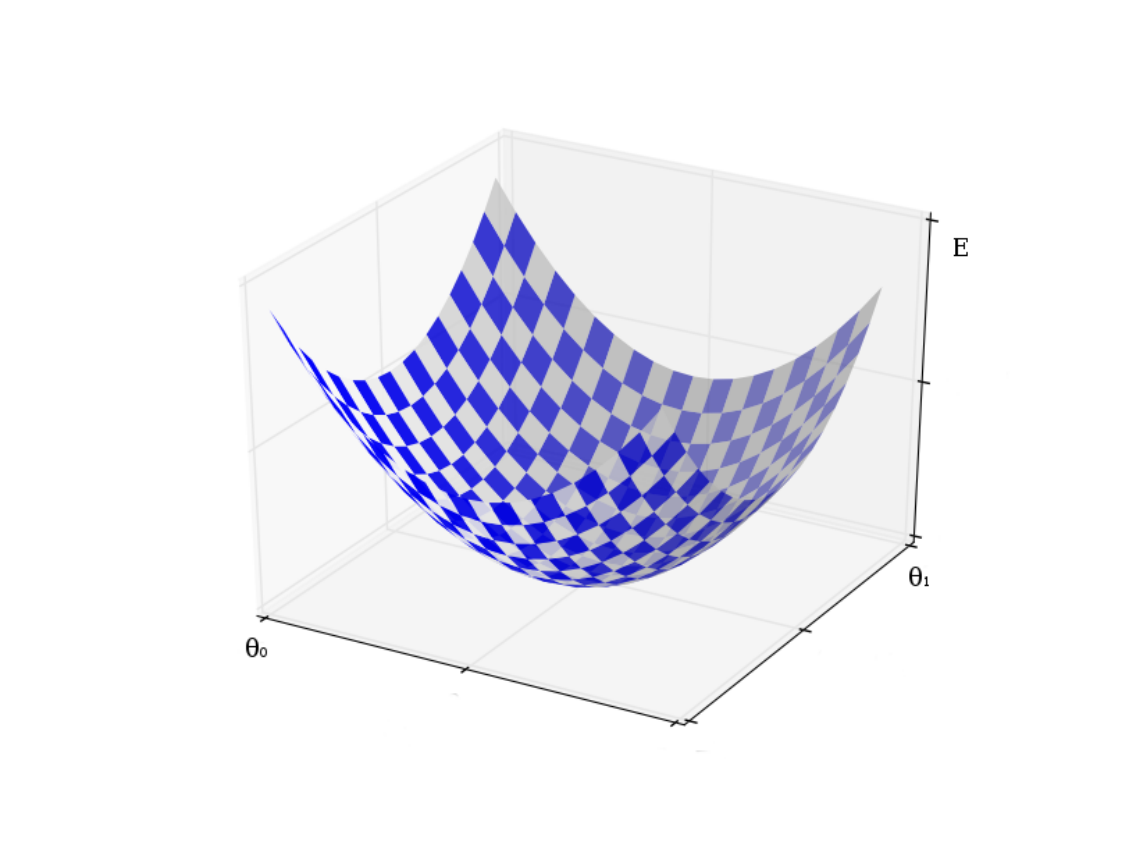
$$E = \frac{1}{2N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 = \frac{1}{2N}\sum_{i=1}^{N}(\theta_0 + \theta_1 x_i - y_i)^2$$

E is a function of θ_0 and θ_1.

It is calculated over *all* training examples in our data (see $\sum$).

How do we find its minimum *min* $E(\theta_0, \theta_1)$?

Gradient descent



In order to find *min* $E(\theta_0, \theta_1)$, we will randomly initialise our θ_0 and θ_1 and then 'move' them in what we think is the right direction to find the bottom of the plot.

What is the right direction?

To take each step towards our minimum, we are going to update θ_0 and θ_1 according to the following equation:

$$\theta_j := \theta_j - \alpha \frac{\delta}{\delta \theta_j} E(\theta_0, \theta_1)$$

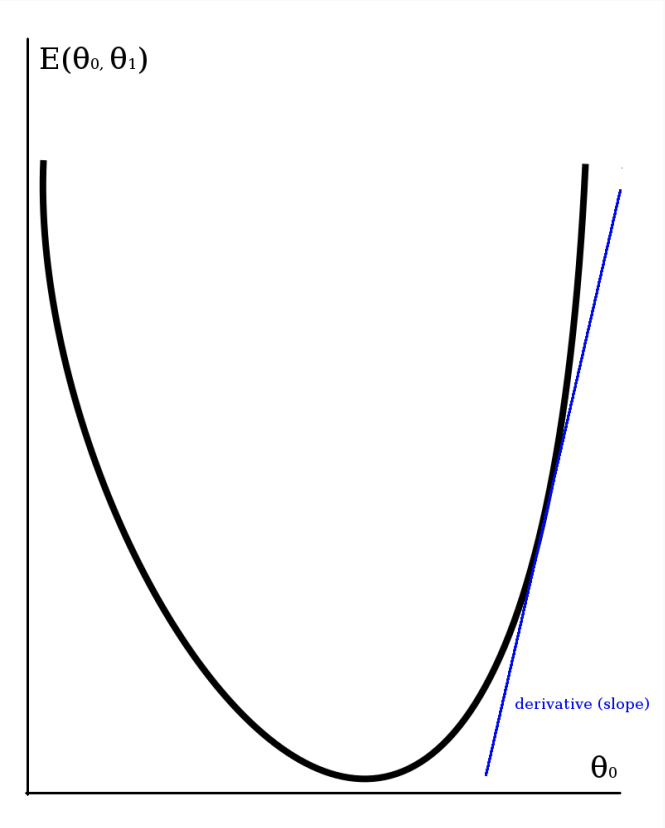
α is called the *learning rate*.

$\frac{\delta}{\delta \theta_j} E(\theta_0, \theta_1)$ is the *derivative* of E for a particular value of θ.

(j in the equation simply refers to either 0 or 1, depending on which θ we are updating.)

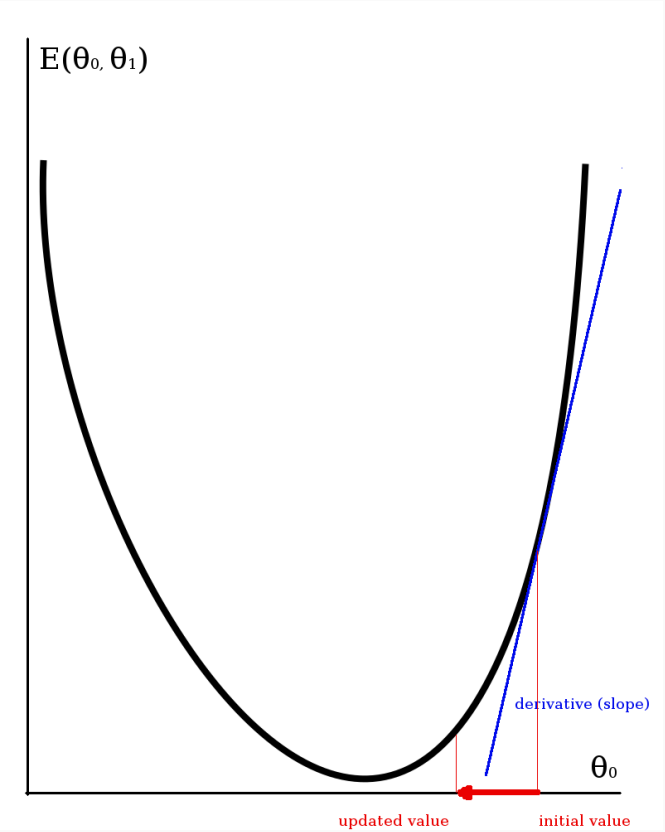
What does the derivative do?

- Imagine plotting just one θ, e.g. θ_0, against the error function.
- We have initialised θ_0 to some value on the horizontal axis.
- We now want to know whether to increase or decrease its value to make the error smaller.



What does the derivative do?

- The derivative of E at θ_0 tells us how steep the function curve is at this point, and whether it goes 'up or down'.
- Effect of positive derivative D_+ on the θ_0 update:
  $\theta_0 := \theta_0 - \alpha D_+$
  θ_0 decreases!

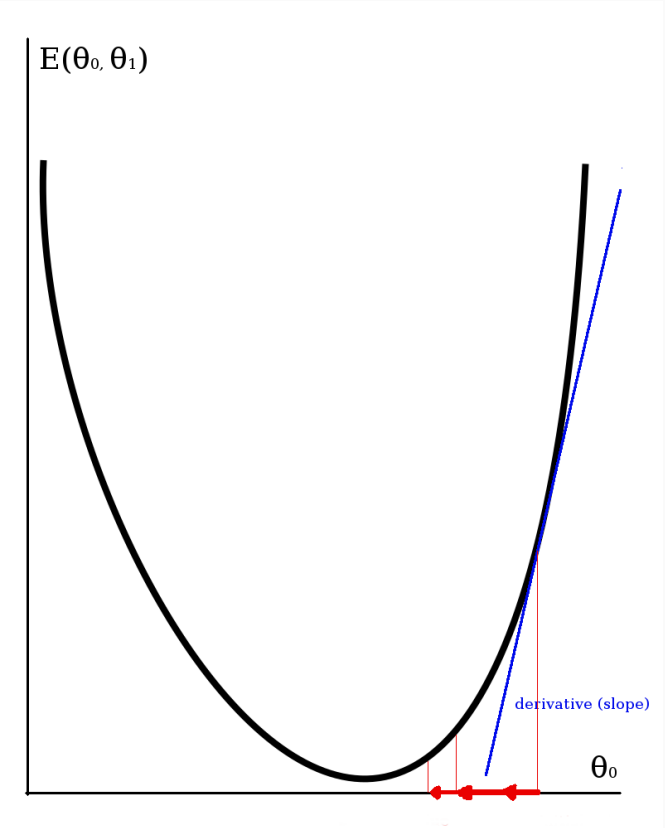


What does the learning rate do?

- α multiplies the value of the derivative, so the bigger it is, the bigger the update to θ:

$$\theta_j := \theta_j - \alpha \frac{\delta}{\delta \theta_j} E(\theta_0, \theta_1)$$

- A too small α will result in slow learning.

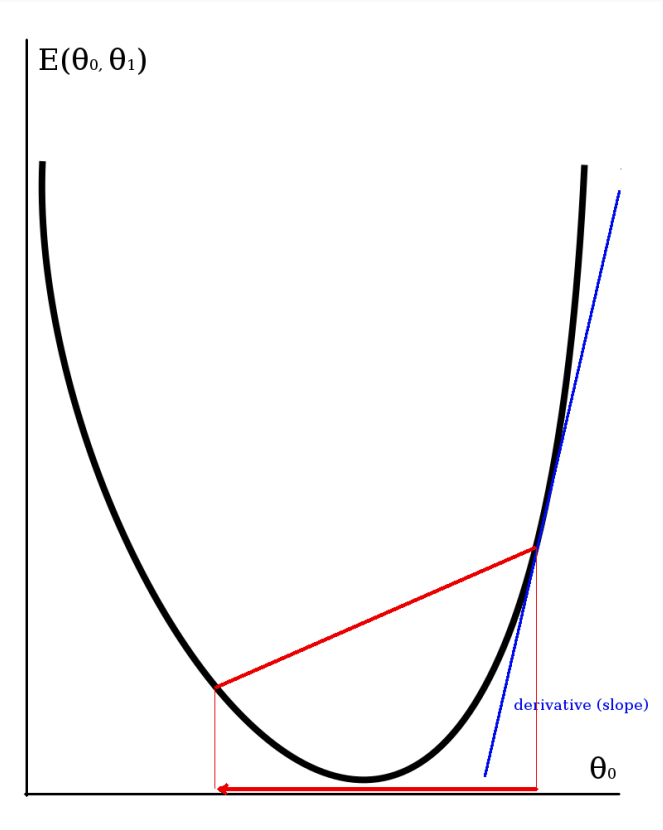


What does the learning rate do?

- α multiplies the value of the derivative, so the bigger it is, the bigger the update to θ:

$$\theta_j := \theta_j - \alpha \frac{\delta}{\delta \theta_j} E(\theta_0, \theta_1)$$

- A too large α may result in **not** learning.



Putting it all together

- The gradient descent algorithm finds the parameters θ of the linear function so that prediction errors are minimised with respect to the training instances.
- We do repeated updates of both θ_0 and θ_1 over our training data, until we *converge* (i.e. the error does not go down anymore).
- The final θ values after seeing all the training data should be the best possible ones.

To bear in mind...

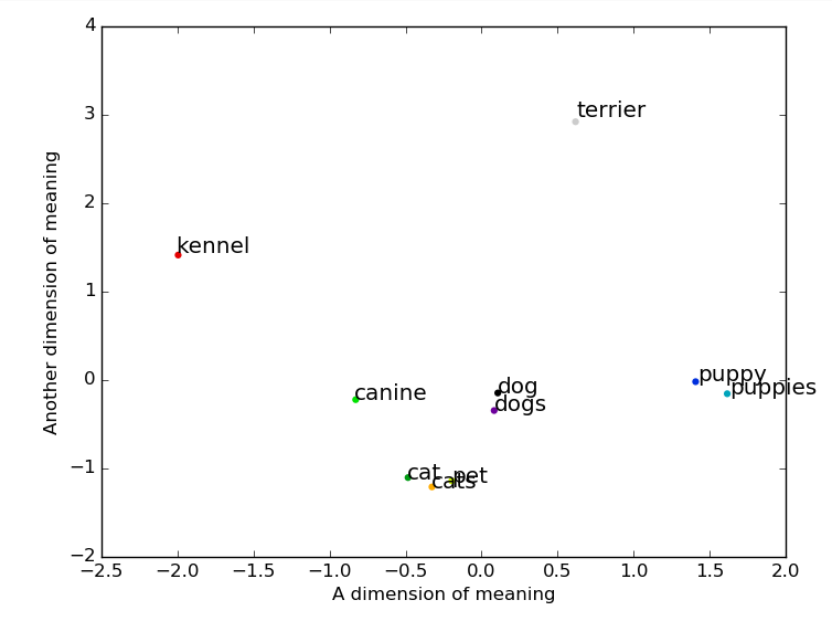
- How well and how fast gradient descent will train depends on how you initialise your parameters.
- Can you see why? (Hint: come back to the error curve and imagine a different starting value for θ_0.)

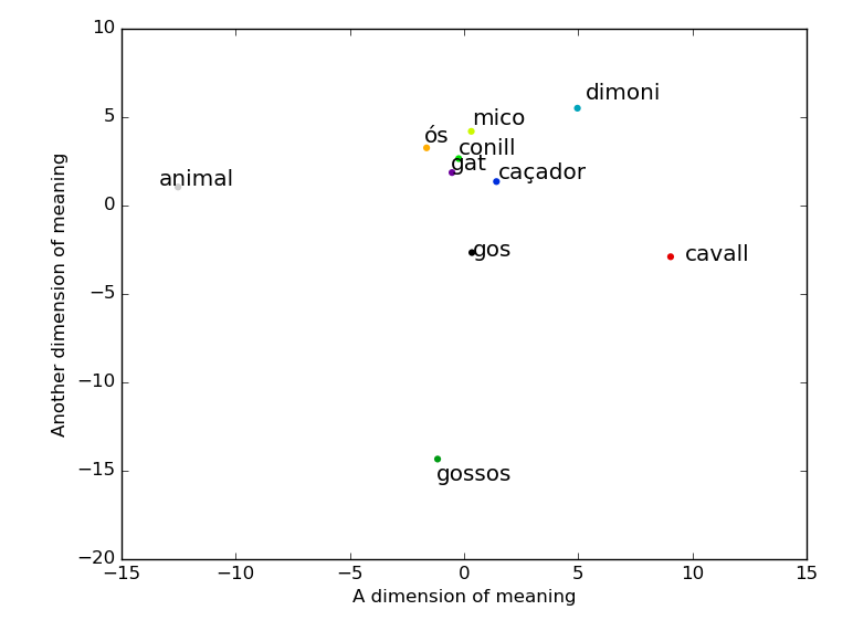
Partial Least Square Regression

Regression as mapping

- We can think of linear regression as directly *mapping* from a set of dimensions to another: e.g. from the values on the *x*-axis to the values on the *y*-axis.
- Partial Least Square Regression (PLSR) allows us to define such a mapping via a *latent common space*.
- Useful when we have more features than training datapoints, and when features are *colinear.*

Example matrix-to-matrix mapping

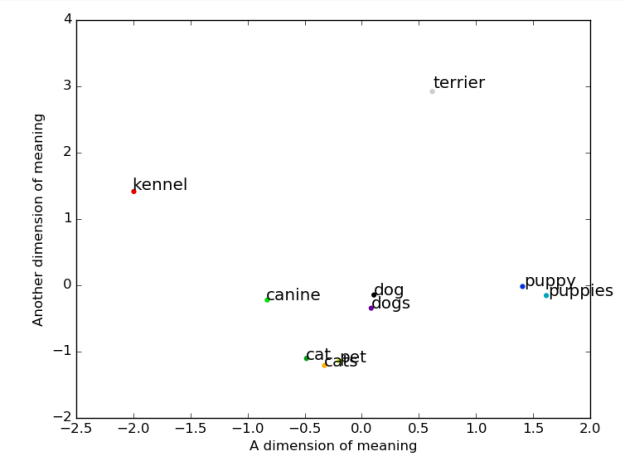


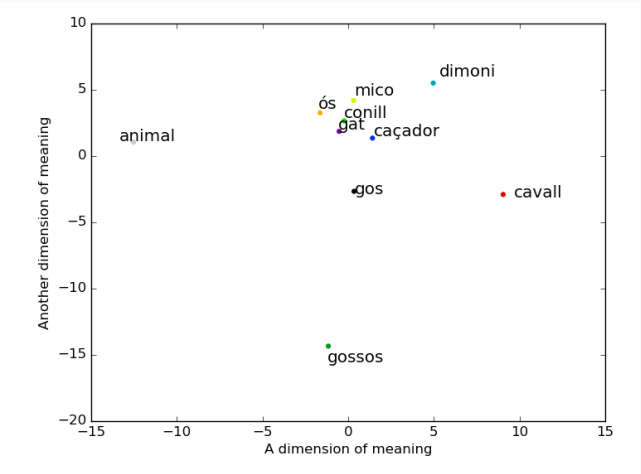


Can we map an English semantic space into a Catalan semantic space?

Example matrix-to-matrix mapping

- Here, each datapoint in both input and output is represented in hundreds of dimensions.
- The dimensions in space 1 are not the dimensions in space 2.
- Intuitively, translation involves a recourse to meta-linguistic concepts (some *interlingua*), but we don't know what those are.



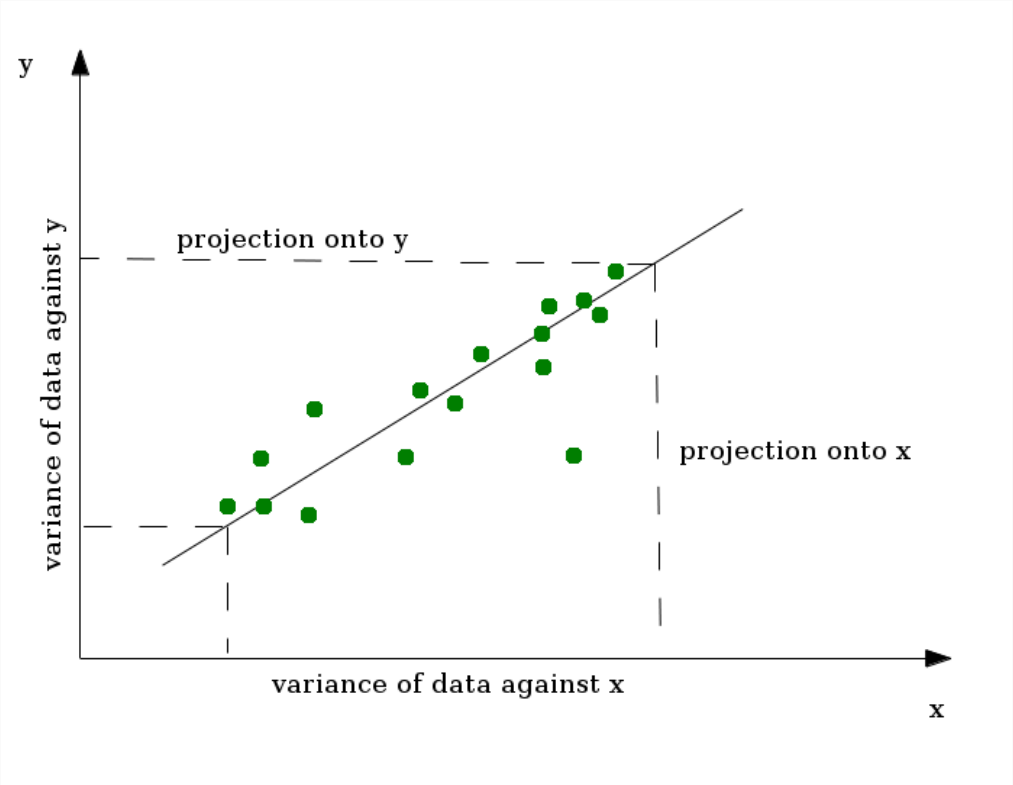


http://www.openmeaning.org/viz/ (it's slow!)

Principal Component Analysis

- Let's pause for a second and look at the notion of Principal Component Analysis (PCA).
- PCA refers to the general notion of finding the components of the data that maximise its variance.
- Let's now look at a graphical explanation of variance. It will be useful for our understanding of PLSR.

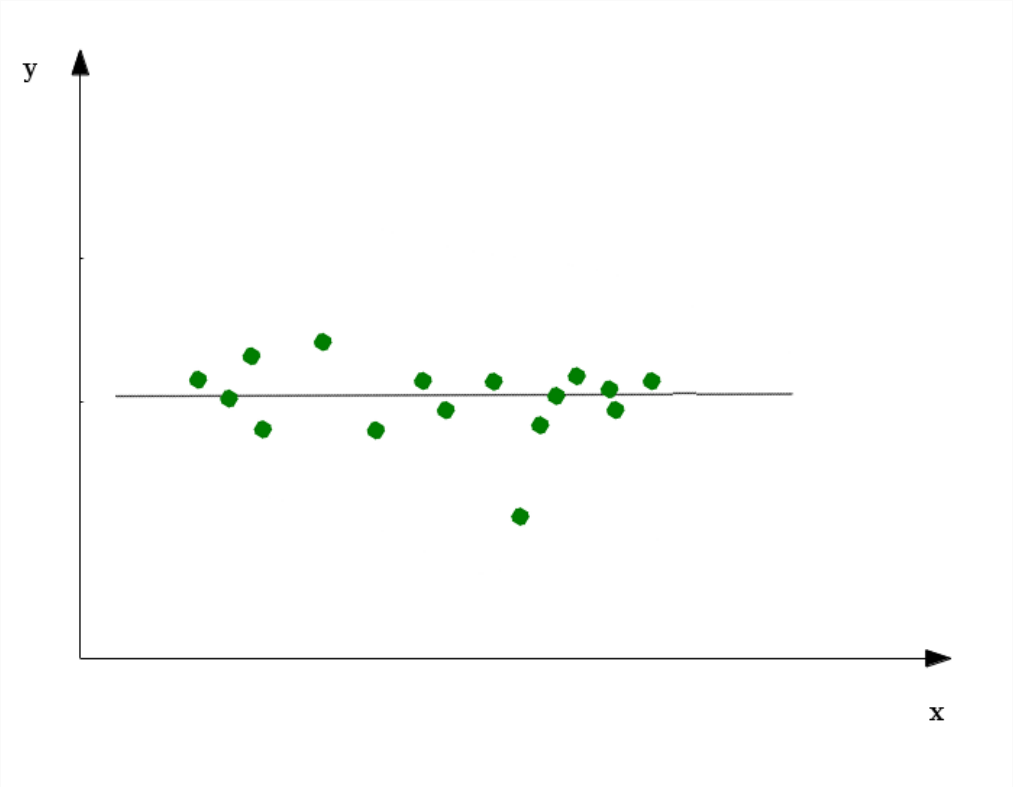
(Non-)explanatory dimensions



If we project these green datapoints on the *x* axis, we still explain a lot about the distribution of the data. A little less so with the *y* axis.

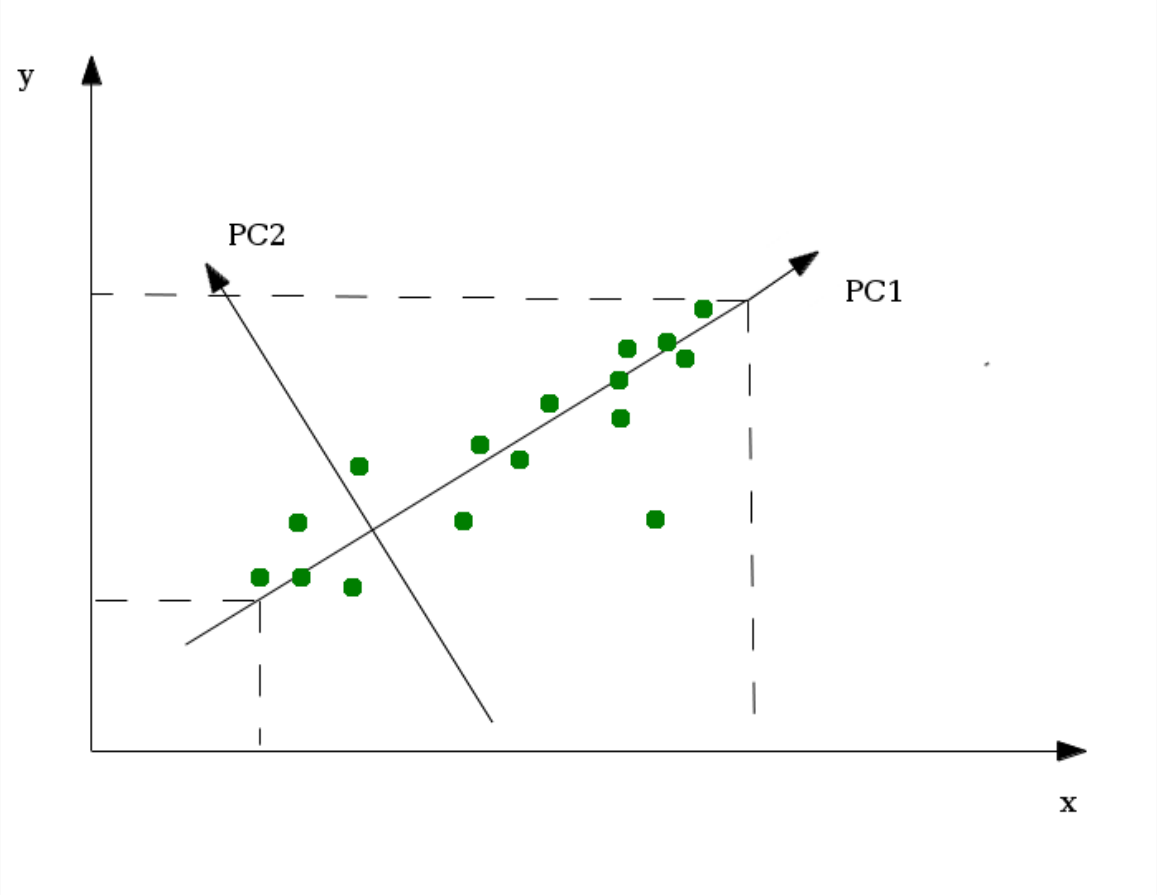
x* explains more of the variance than *y

(Non-)explanatory dimensions



Here, the *y* axis is rather uninformative. Get rid of it?

(Non-)explanatory dimensions



Actually, here are the most informative dimensions...
we'll call them PC1 and PC2.

We can find PC1 and PC2 by computing the *eigenvectors and eigenvalues* of the data.

Eigenvectors and eigenvalues

- Eigenvectors and eigenvalues live in pairs.
- An eigenvector is a vector and gives a *direction* through the data.
- The corresponding eigenvalue is a number and gives the amount of *variance* the data has along the direction of the eigenvector.
- Eigenvectors are perpendicular to each other. Their number corresponds to the dimensionality of the original data (number of features).
  2D = 2 eigenvectors, 3D = 3 eigenvectors, etc.

On the importance of normalisation

- Before performing PCA, the data should be *normalised*.
- Without normalisation, we may catch a lot of variance under a non-informative dimension (feature), just because it is expressed in terms of ‘bigger numbers’.

What is normalisation?

- Normalisation is the process of transferring values which were measured under different scales to a common scale.
- Examples:
  - Number of heartbeats a day: from 86,400 to 129,600.
  - Probability of airplane crash per airline: from 0,0000002 to 0,000000091.
  - Person's height: from 1m to 2m.
  - Person's height: from 1000mm to 2000mm.
- Can we put all this on a scale from 0 to 1? For instance, by doing min-max normalisation:

$$x := \frac{x - \min(x)}{\max(x) - \min(x)}$$

PCA without normalisation

- Let's say we are plotting people's height vs weight:
  - heights:
    1500mm, 1700mm, 1800mm, 2000mm (variance 32500)
  - weights:
    50kg, 70kg, 80kg, 120kg (variance 650)

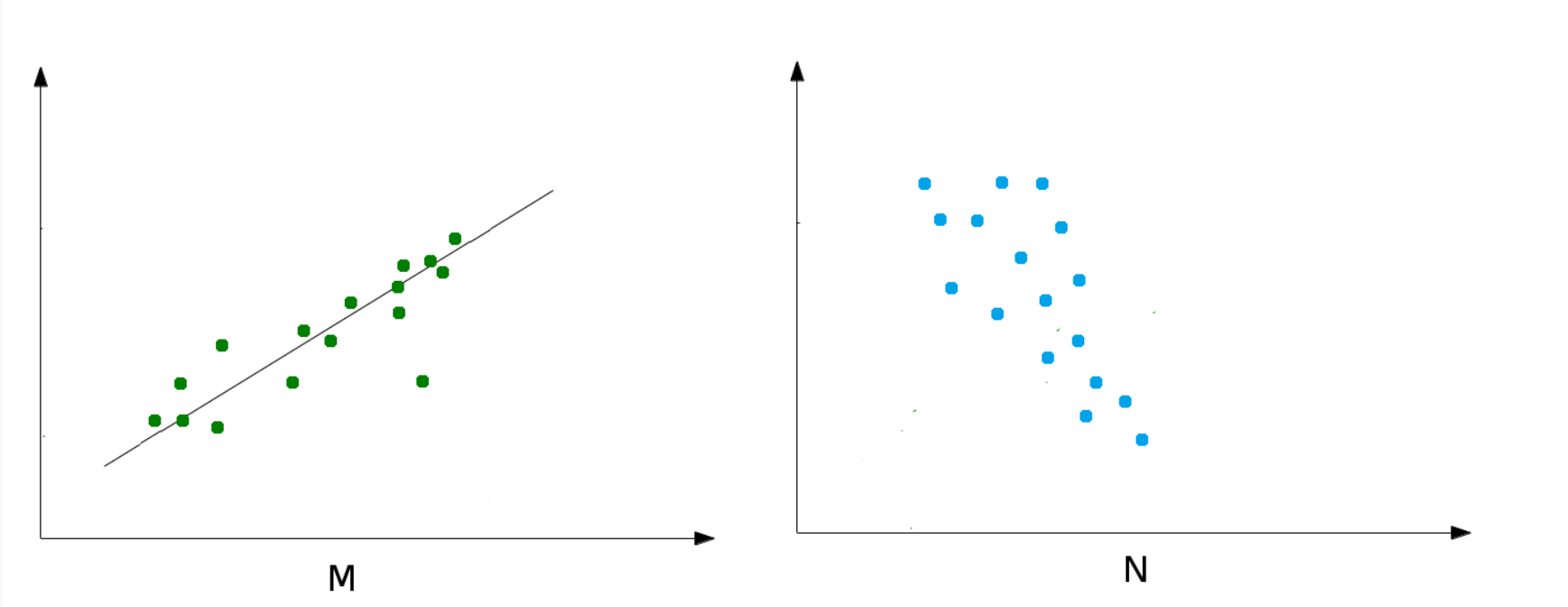
- With normalisation:
  - heights:
    0, 0.4, 0.6, 1 (variance 0.13)
  - weights:
    0, 0.29, 0.43, 1 (variance 0.13)

Back to PLSR...

- We use PLSR when we want to predict a set of features from another set of features.
- In effect, we want to predict a matrix N of size k x q from a matrix M of size k x p.
- For instance, we might have:
  - a matrix M with $k = 3$ English words (*dog, cat, eat*), expressed in $p = 300$ dimensions;
  - a matrix N with $k = 3$ Catalan words (*gos, gat, menjar*), expressed in $q = 400$ dimensions.

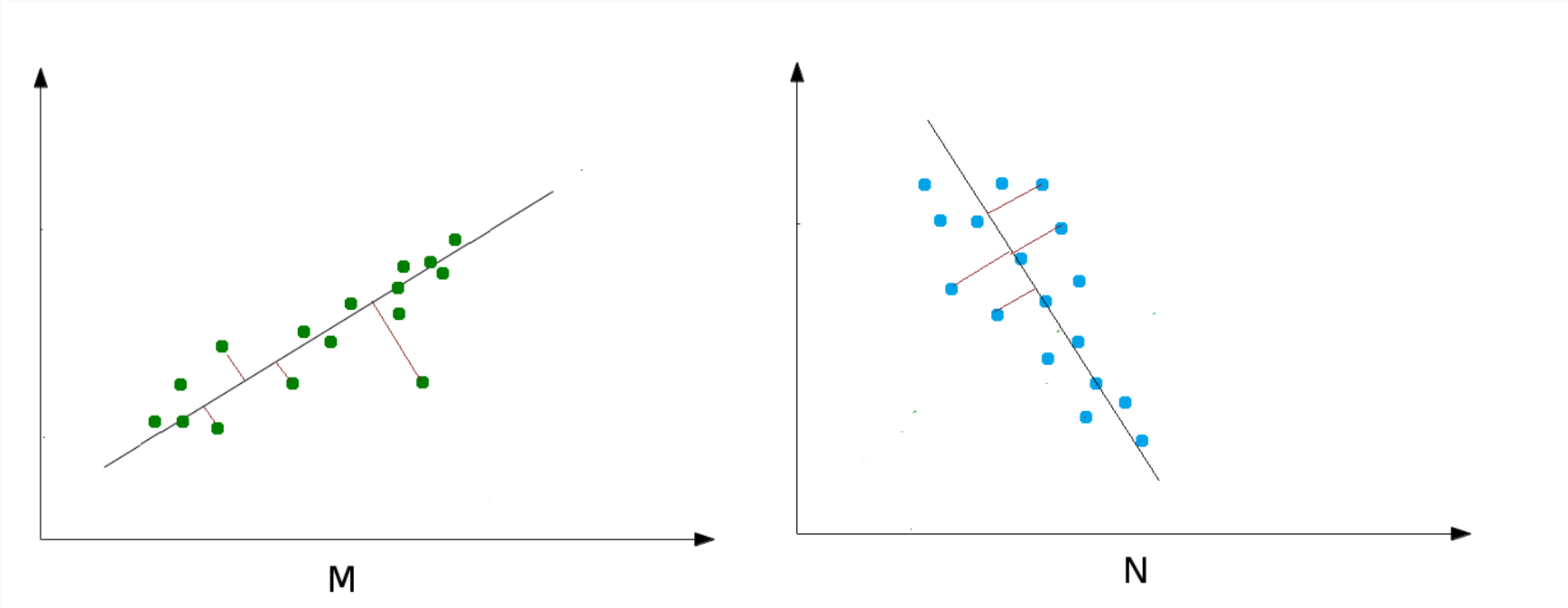
PLSR: intuition

- Let's assume we have two sets of data (the matrices *M* and *N*) and we perform PCA on both of them to find their principal components.
- Warning: it is not exactly what happens in PLSR, but it will make things easier to understand...



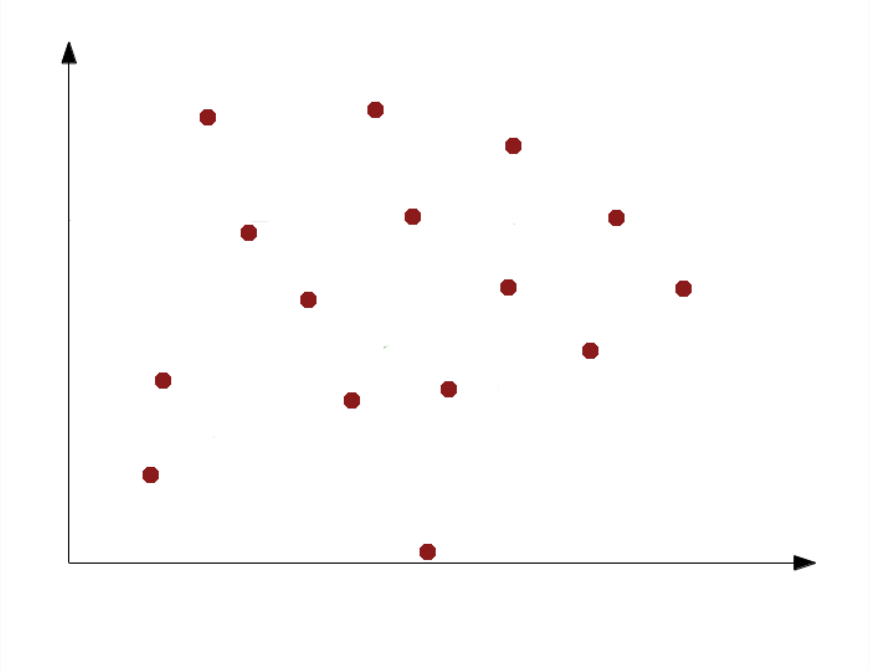
PLSR: intuition

- Now we have PC1 for each matrix, we can try to find a linear function that maps the data in PC1 of *M* into the data in PC1 of *N* (linear regression problem).
- We first need to replot our data onto PC1, using projection.



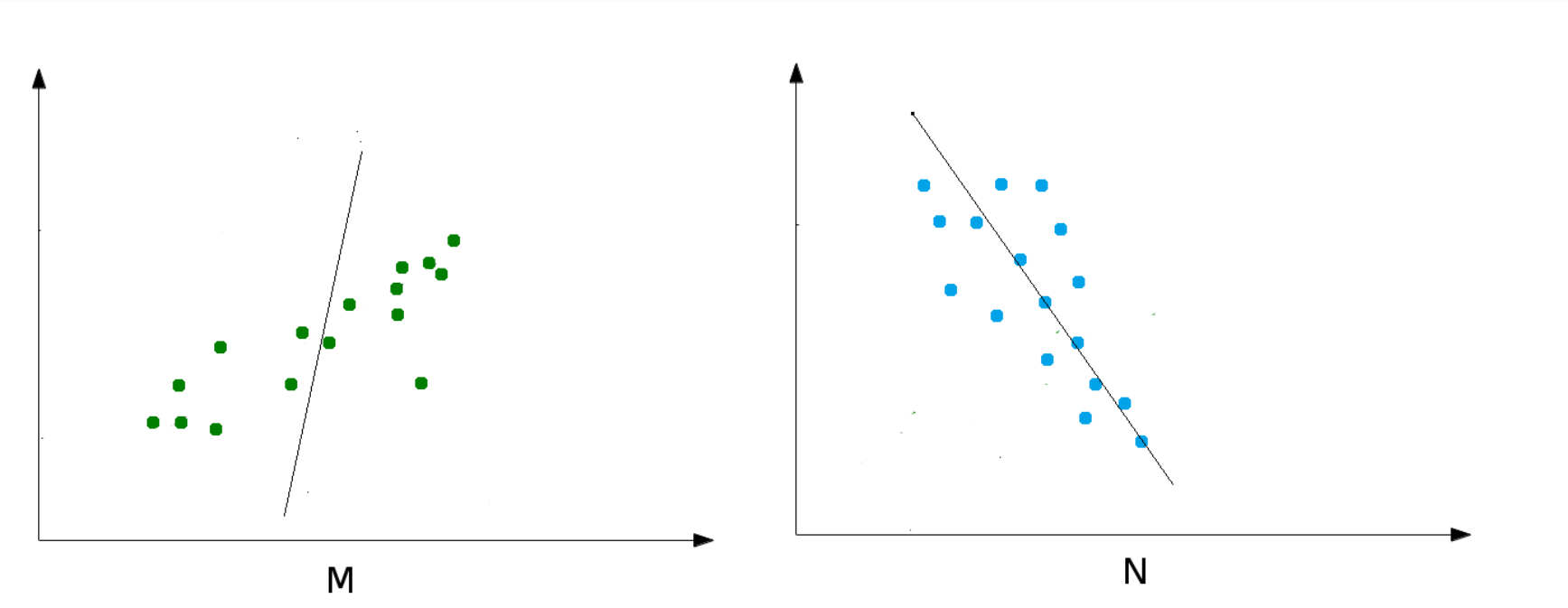
PLSR: intuition

- Once data has been replotted onto a component for both matrices, we can plot the result onto a 2D graph.
- On the resulting graph, the x dimension will receive the values of the points in *M* projected onto PC1 of *M*. The y dimension receives the values of the points in *N* projected onto PC1 of *N*.
- Problem: that plot might not show very much correlation...



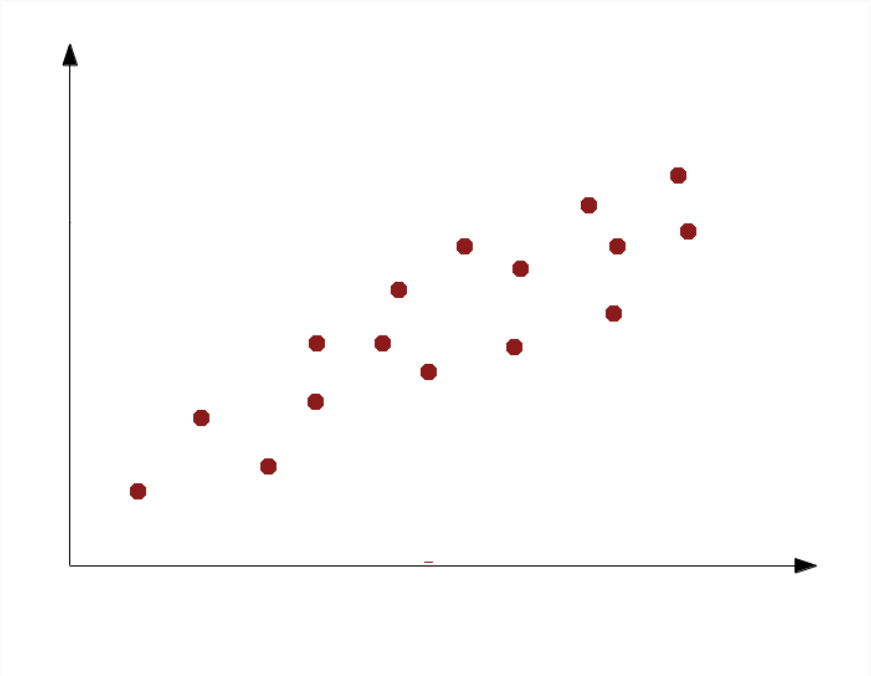
PLSR: what actually happens

- For a given component, we want to find the function that maximises the covariance (correlation) between the two matrices.
- To do this, we are going to keep PC1 of *N* and rotate PC1 of *M* until we find that maximum covariance.



PLSR: what actually happens

- For a given component, we want to find the function that maximises the covariance (correlation) between the two matrices.
- To do this, we are going to keep PC1 of *N* and rotate PC1 of *M* until we find that maximum covariance.



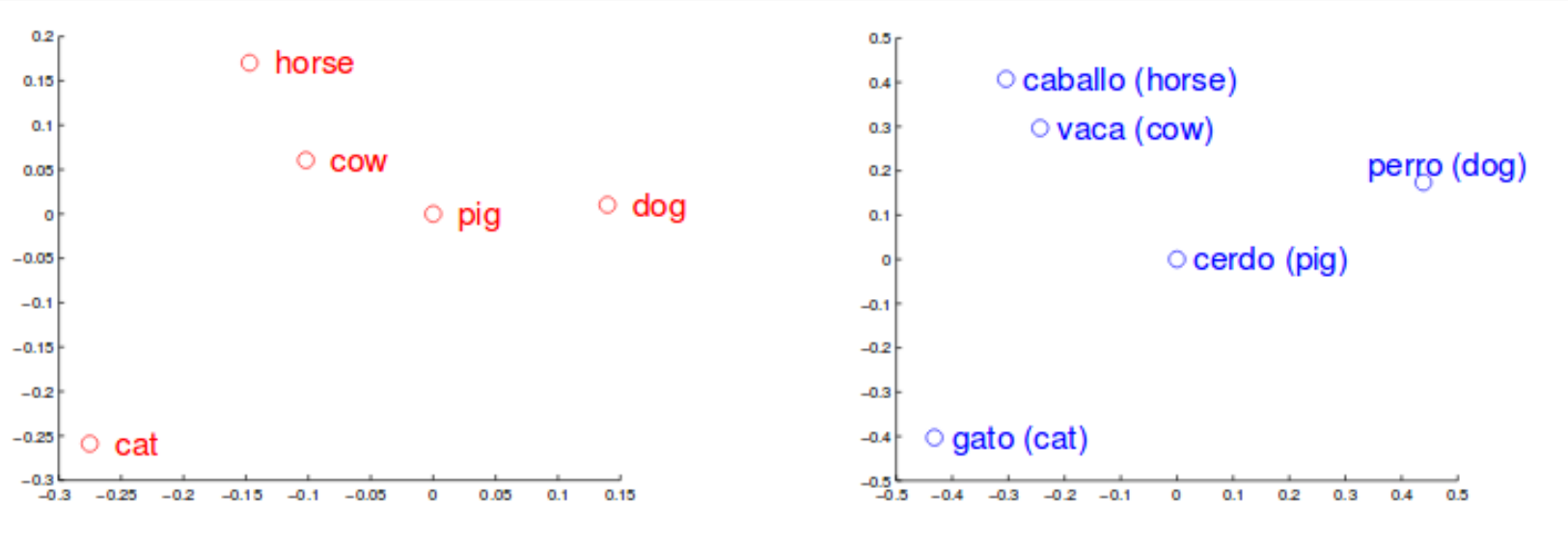
PLSR in real life

- In effect, running PLSR on two matrices M and N will give us a learnt projection matrix A which, when multiplied by M gives us and approximation of N.
- We have looked at what happens with just one component, but in practice, we'll choose a higher number of components.
- Is there a way to interpret the resulting, dimensionality-reduced representations of both original matrices? (More on this in the unsupervised learning session...)

Translating with regression

- Let's say we have two semantic spaces S_1 and S_2.
- We assume we also have a set of overlapping *seeds* $O = \{w_1, w_2 ... w_n\}$, for which vectors are available in both spaces.
- We can extract two matrices from S_1 and S_2 corresponding to the words in O:
  $O_1 \subset S_1$ and $O_2 \subset S_2$.
- Learn the regression.
- Apply to $S_1 - O_1$, the words for which we don't have a mapping.

Translating single words



Mikolov et al (2013). https://arxiv.org/pdf/1309.4168.pdf

Distance as confidence measure

- There may be some general regularity in translation, but some words' meanings will be idiosyncratic.
- How can we know whether a translation is good or not?
- **Use distance to nearest neighbour:**
  intuitively, if the translated vector y is quite far from every other word in the space's vocabulary V, it may not be a quality vector. So we impose the constraint:

$$max_{i \in V} cos(y, z_i) > t$$

  where t is a threshold.

Evaluation measure: P@k

- P@k is an evaluation measure that takes into account the rank of the correct answer.
- Say we have translated *horse* to vector $\hat{p} = [-0.2, 0.4]$ in our predicted Spanish space
  (see translated space in slide 43).
- Let's now compute the nearest neighbours of $\hat{p}$ amongst the gold standard vectors. We find
  1:cava (cow), 2:cerdo (pig), 3:caballo (horse).
- So the 'true' vector for *horse* is at rank 3 in the nearest neighbours of $\hat{p}$.

Evaluation measure: P@k

- P@k is the precision of the system at rank k:
  - How many times do I find the right answer as the 1st nearest neighbour of my prediction?
  - How many times at rank 2, 5, 10...?
- Obviously, P@1 is much harder than P@10.

Distance as confidence measure

- Evaluation: for a predicted vector, see whether its nearest neighbour in the gold data is the correct translation.
- Effect of threshold vs precision @ N for English –> Spanish.
- Huge effect on recall.

Threshold	Coverage	P@1	P@5
0.0	92.5%	53%	75%
0.5	78.4%	59%	82%
0.6	54.0%	71%	90%
0.7	17.0%	78%	91%

Problems with aligning concepts

- Concepts are not universal: different languages/cultures associate different things with a word.
- The mapping is actually highly non-linear!

money	Distance
money	0
cash	0.36
earn	0.44
much	0.46
pay	0.47
buy	0.47
you	0.47
me	0.48
bank	0.48
winnings	0.49

geld	Distance	
geld	0	money
blut	0.35	broke
dollar	0.36	dollar
contanten	0.37	cash
platzak	0.38	broke
contant	0.38	in cash
verdienen	0.4	earn
betaald	0.41	paid
aanbetaling	0.41	deposit
betalen	0.42	pay

http://meshugga.ugent.be/snaut-dutch/

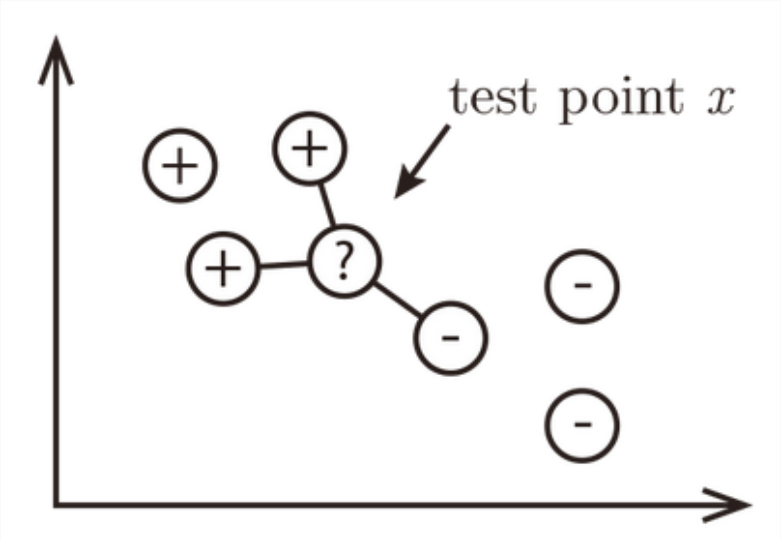
k-NN algorithm

The k-NN algorithm

- A simple but powerful algorithm using the position of training points in space.
- Let's assume some training data $\{(x_1, y_1), (x_2, y_2)...(x_n, y_n)\}$ where x_i is the input and y_i the output.
- y_i can take two forms:
  - $y_i \in \{1, ..., C\}$ in classification problems;
  - $y_i \in \mathbb{R}$ in regression problems (i.e y_i is a real value).
- We want to predict y_U from x_U, a new and unknown input.

The k-NN algorithm

The idea behind k-NN is that for a given x_i, we use its nearest neighbours for prediction.



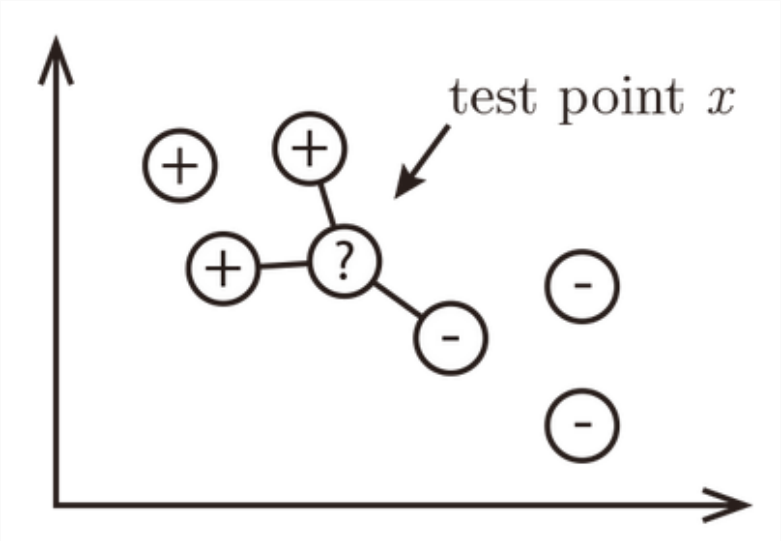
https://www.cs.cornell.edu/courses/cs4780/2017sp/lectures/lecturenote02_kNN.html

The k-NN algorithm

- The algorithm requires to set k, the number of neighbours, and a distance function d between points in the space.
- Four simple steps:
  - calculate $d(x_U, x_i)$ for all x_i in the data;
  - sort distances in descending order;
  - return top k distances;
  - y_U is given by *majority voting* for classification and *averaging* for regression.

Majority voting (classification)

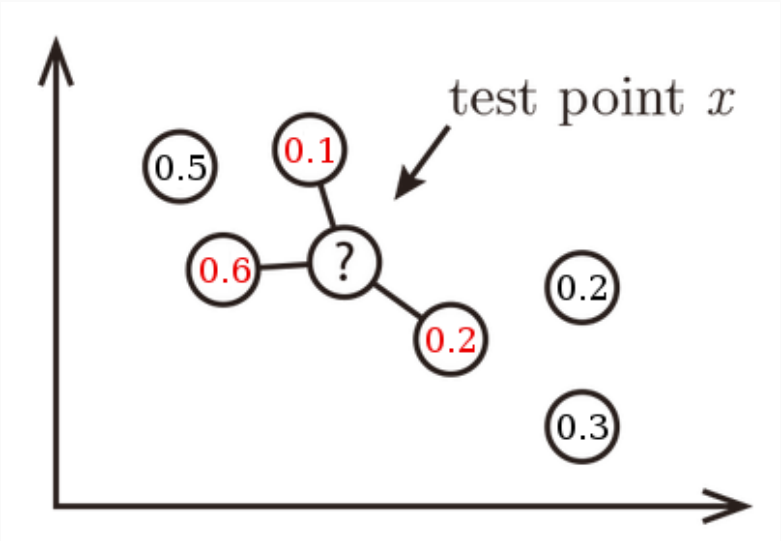
Here, with $k = 3$ and two discrete classes, $+$ and $-$, we will return $y_U = +$ (majority class for the 3 nearest neighbours).



https://www.cs.cornell.edu/courses/cs4780/2017sp/lectures/lecturenote02_kNN.html

Averaging (regression)

Here, we simply average the y values of the nearest neighbours (shown inside each point). So $y_U = \frac{0.6+0.1+0.2}{3} = 0.3$.



https://www.cs.cornell.edu/courses/cs4780/2017sp/lectures/lecturenote02_kNN.html

The distance function

- Traditionally, the Euclidian distance is used:

$$d(\mathbf{A}, \mathbf{B}) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2}$$

- In distributional semantics, cosine is the measure of choice:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Evaluation of k-NN

- Since k-NN can be used for both classification and regression, we will use the appropriate measures for the task:
  - Classification: accuracy, precision, recall...
  - Regression: R^2, P@k...

How near is a nearest neighbour?

- A feature space has a certain ‘landscape’.
- The configuration of the space can affect results or their interpretation.

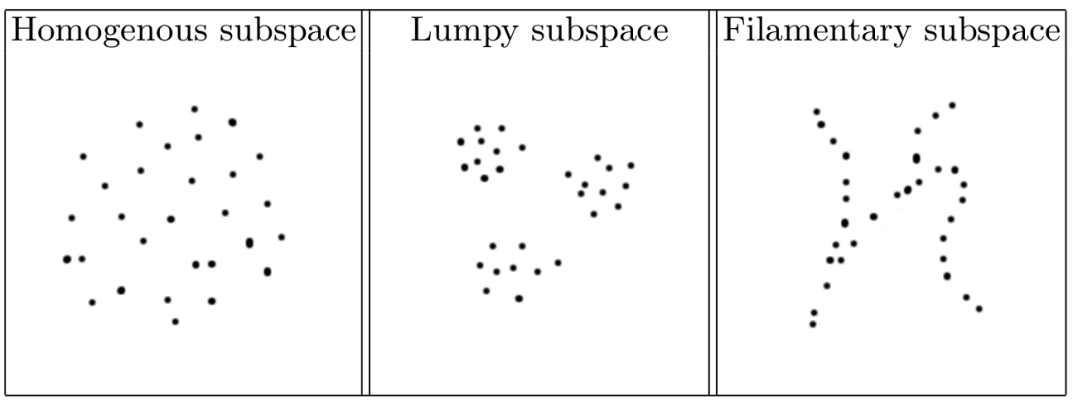


Fig. 3. Neighbourhoods of different character.

Karlgren et al, 2008