KNN Imputation

Christian Werner

(Quantitative geneticist and biostatistician) **EiB, CIMMYT**, Texcoco (Mexico)

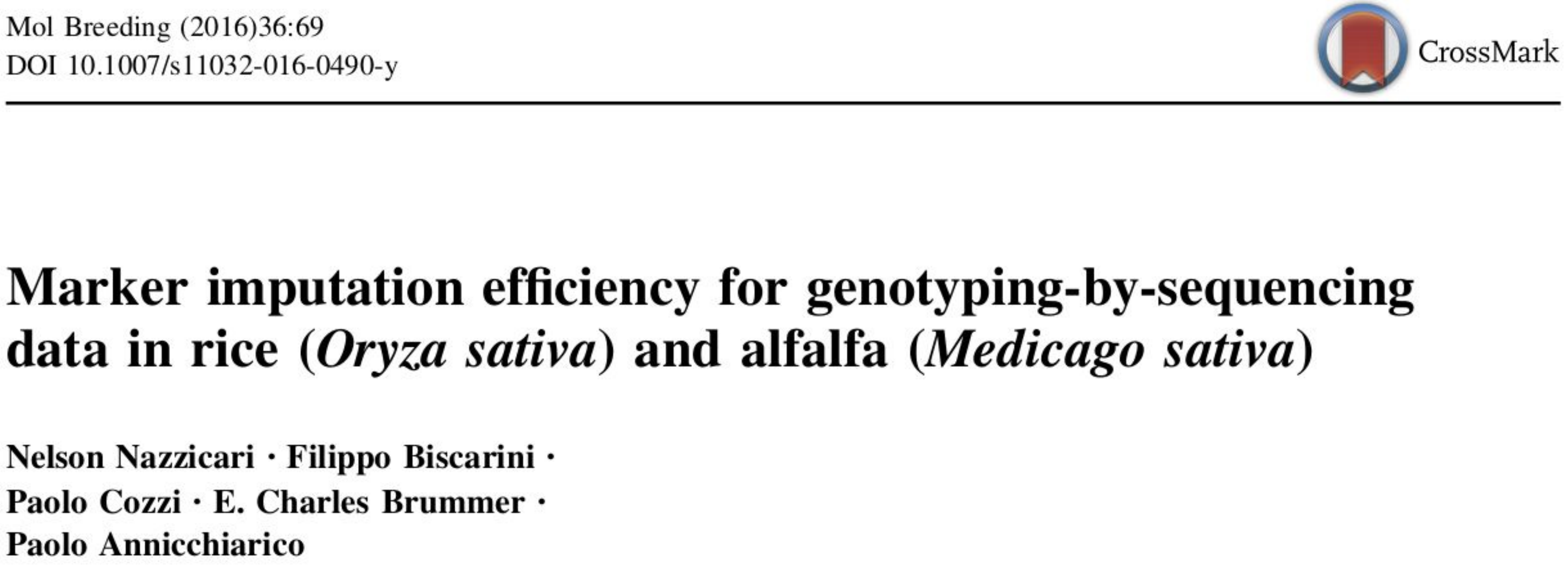
Filippo Biscarini HerrFalloppio

(Biostatistician, bioinformatician and quantitative geneticist) **CNR-IBBA**, Milan (Italy)

Oscar González-Recio OscarGenomics

(Computational biologist and quantitative geneticist) **INIA-UPM**, Madrid (Spain)

Why KNN imputation?

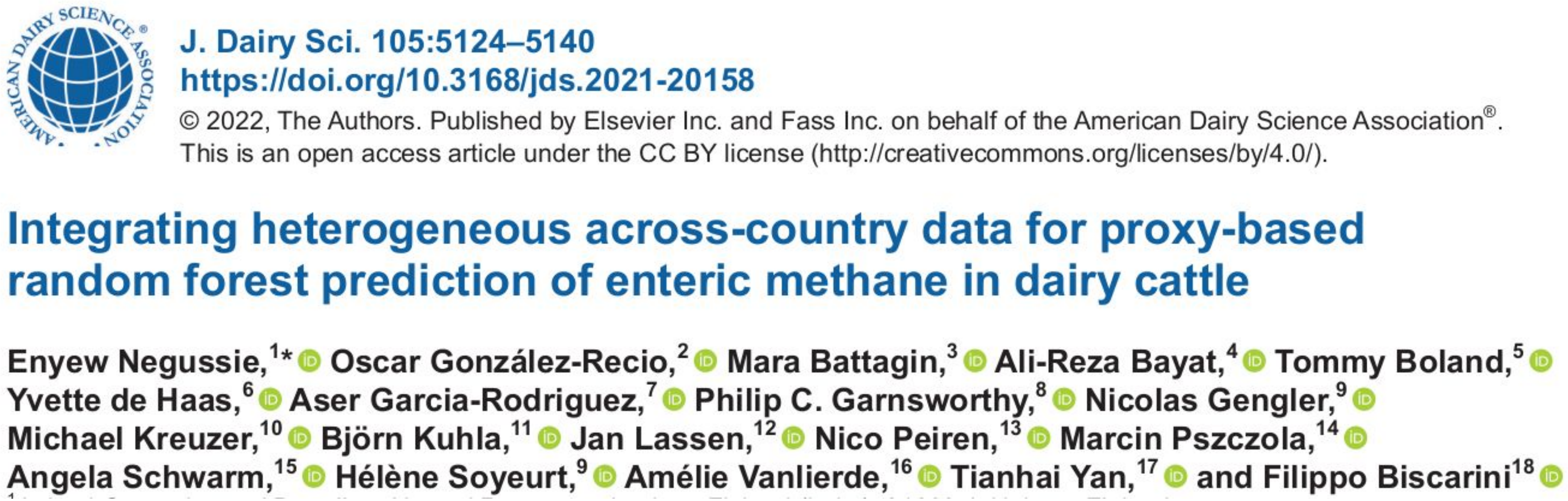


Mol Breeding (2016)36:69
DOI 10.1007/s11032-016-0490-y

Marker imputation efficiency for genotyping-by-sequencing data in rice (*Oryza sativa*) and alfalfa (*Medicago sativa*)

Nelson Nazzicari · Filippo Biscarini · Paolo Cozzi · E. Charles Brummer · Paolo Annicchiarico

Not only genotypes

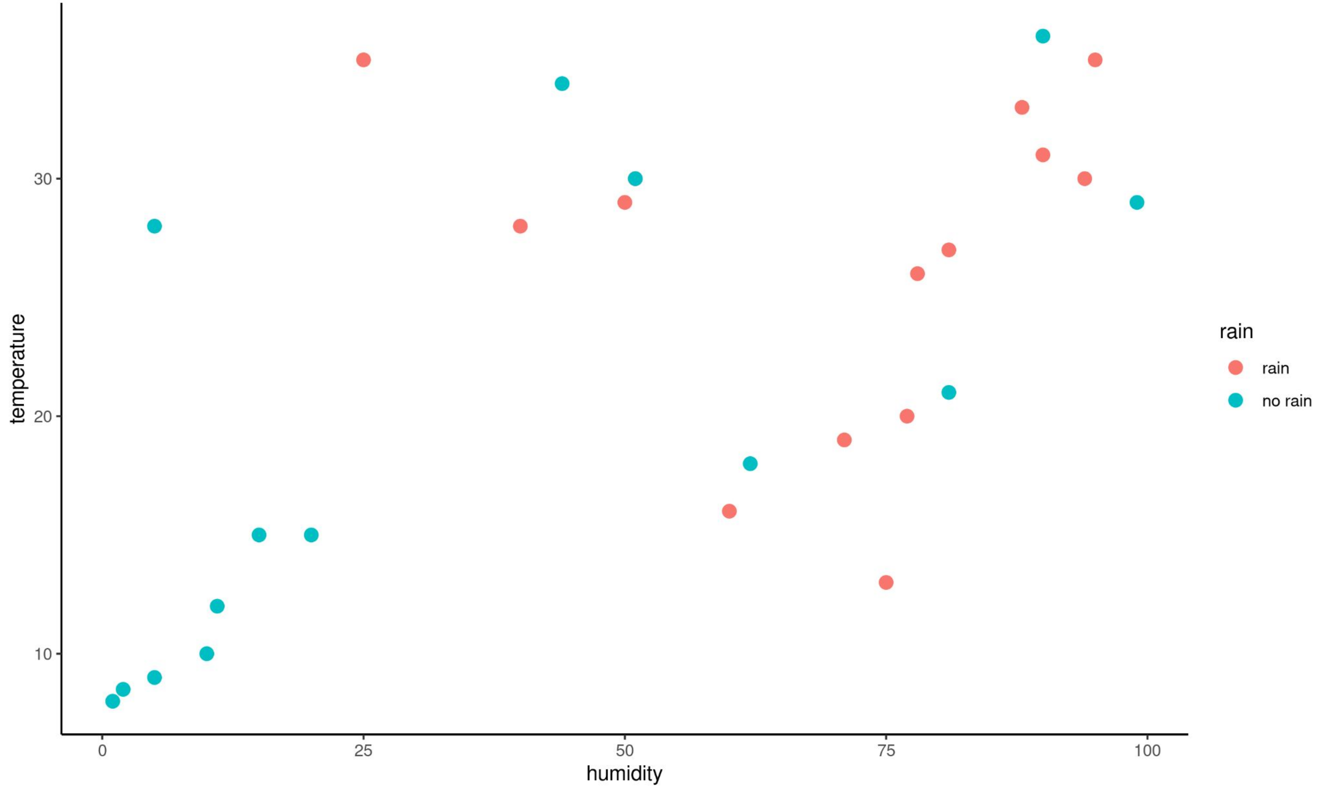


J. Dairy Sci. 105:5124–5140
https://doi.org/10.3168/jds.2021-20158

Integrating heterogeneous across-country data for proxy-based random forest prediction of enteric methane in dairy cattle

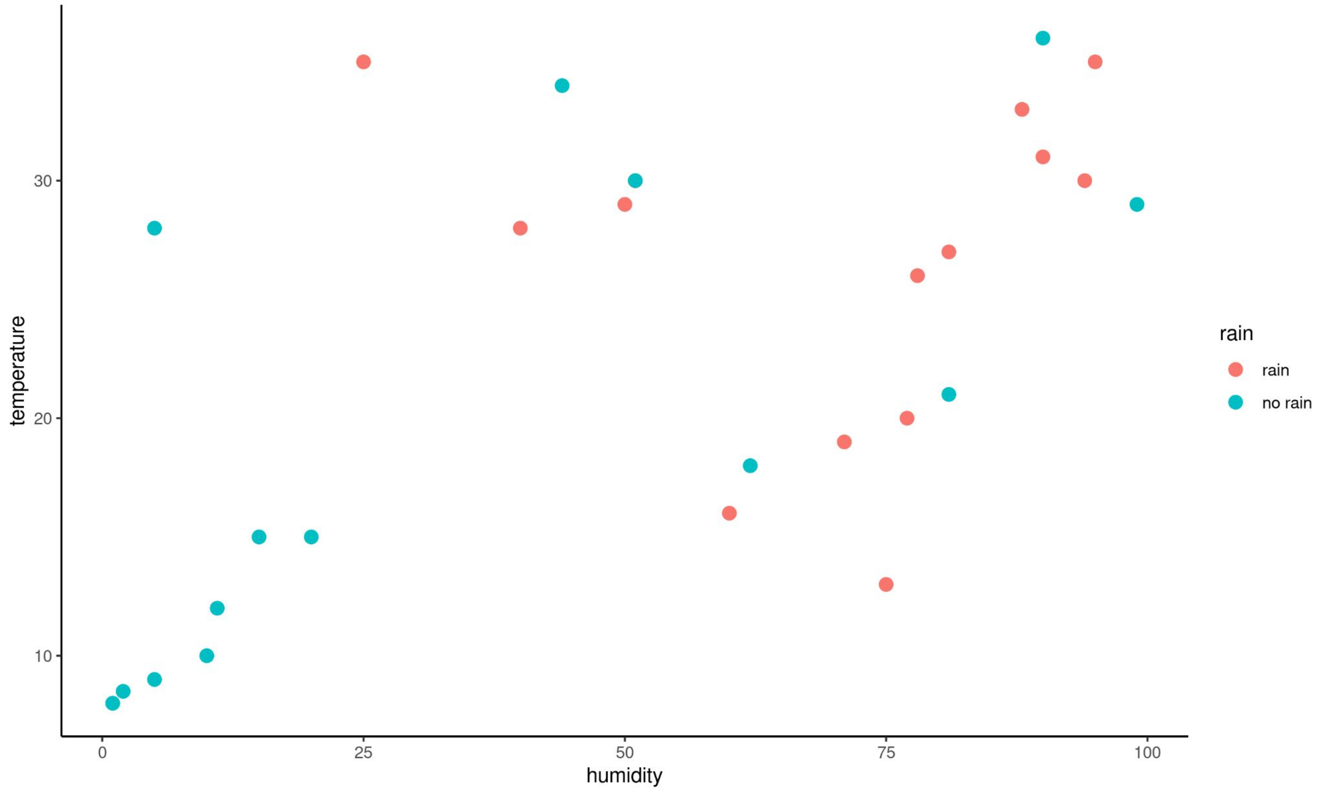
Enyew Negussie,[1*] Oscar González-Recio,[2] Mara Battagin,[3] Ali-Reza Bayat,[4] Tommy Boland,[5] Yvette de Haas,[6] Aser Garcia-Rodriguez,[7] Philip C. Garnsworthy,[8] Nicolas Gengler,[9] Michael Kreuzer,[10] Björn Kuhla,[11] Jan Lassen,[12] Nico Peiren,[13] Marcin Pszczola,[14] Angela Schwarm,[15] Hélène Soyeurt,[9] Amélie Vanlierde,[16] Tianhai Yan,[17] and Filippo Biscarini[18]

One-minute k-nearest neighbors (KNN)!



1. We collect data on temperature, humidity and rain, for a number of days
2. New day: will it rain?

k-nearest neighbors (KNN): a bit of math



1. the values for humidity and temperature define the new datapoint in input
2. measure the Euclidean distance (Pythagoras theorem inside!) with all other points
3. find the closest point (neighbor) to our new point
4. assign to the new point the label (colour) of this nearest neighbor (k=1)

One-minute k-nearest neighbors (KNN)!

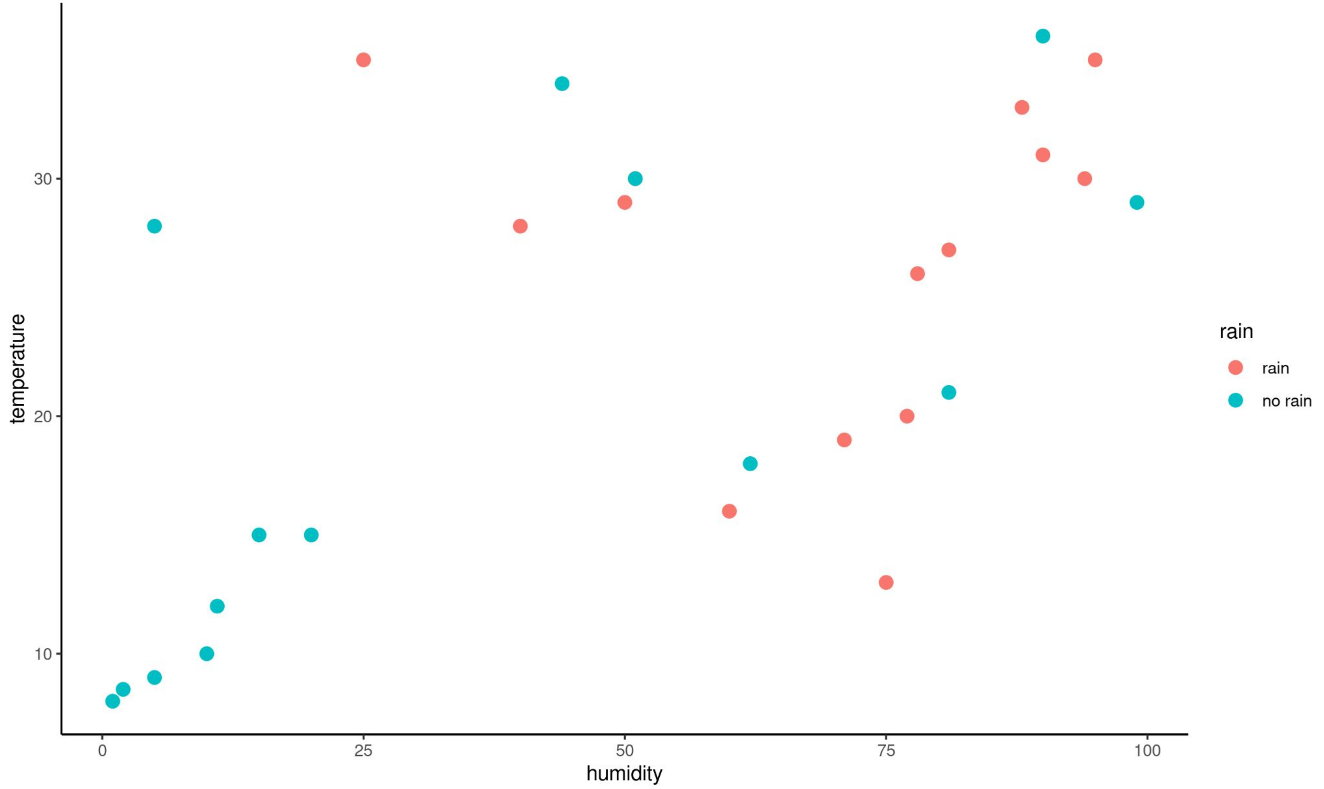
If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck.

$$Pr(Y = j | X = x_0)$$

KNN: more than 3-D

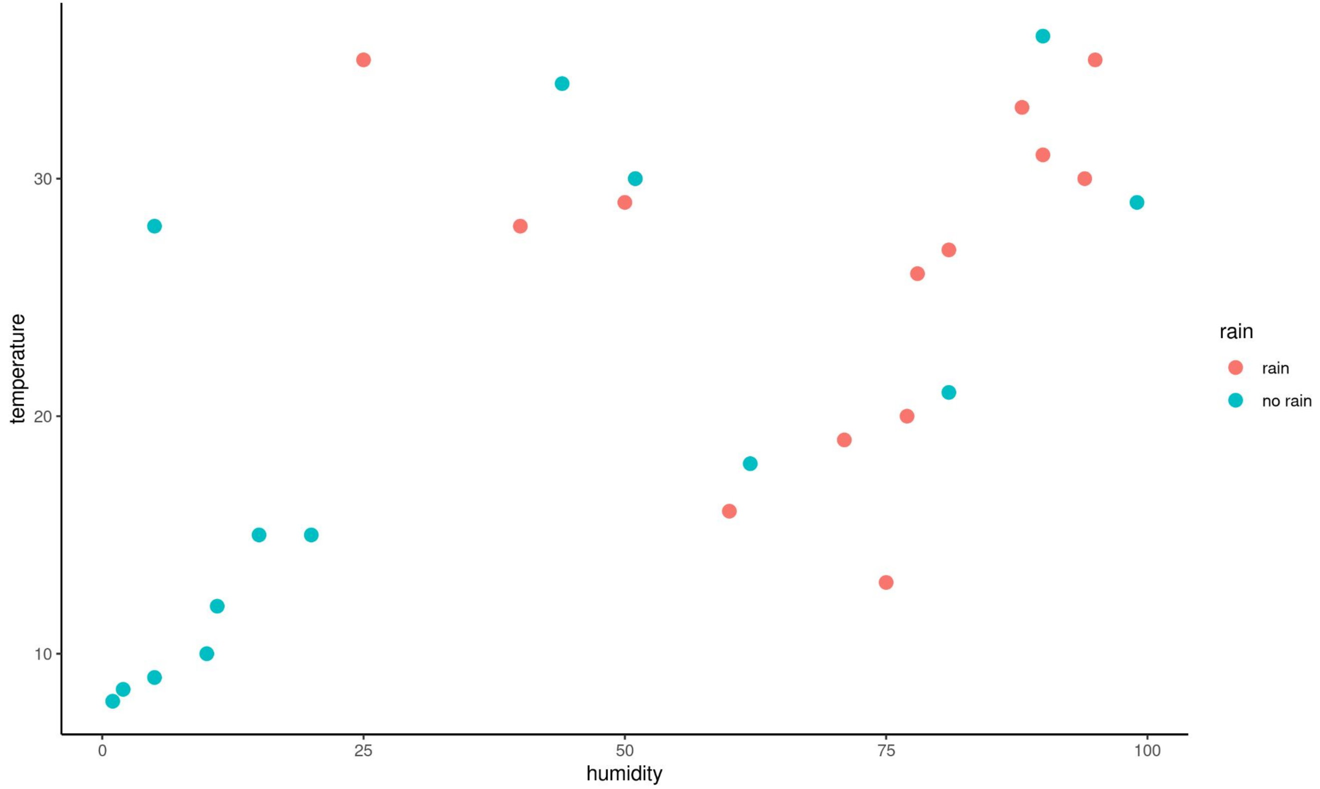
- We saw a simple example with few datapoints and only 2 dimensions
- When we have (typically) many dimensions (e.g. many SNPs) and many datapoints (e.g. many samples) we need the machine to do it! (and we can no longer visualize it)

KNN: K?



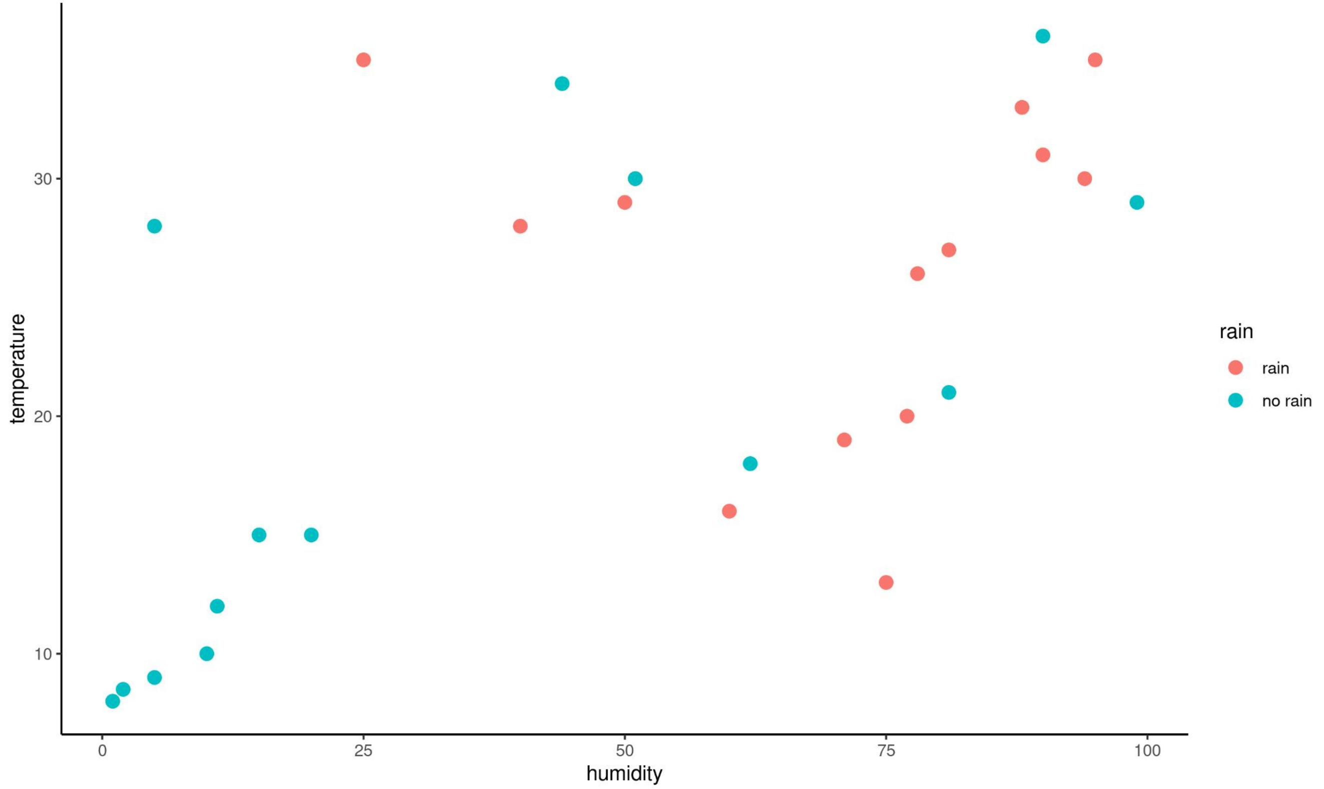
- K-neighborhood → majority!

KNN: K?



- K-neighborhood → majority!
- majority?

KNN: Euclidean distance?



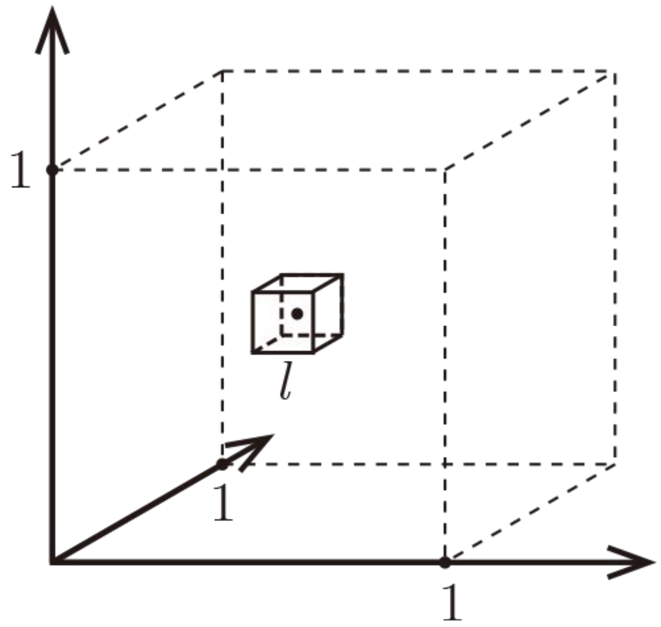
- similarity↔dissimilarity
- [0, +∞]
- many possible distance metrics (Hamming, Chebyshev, Jaccard etc.) → see here

KNN: a few things to tweak

- size of neighborhood (K)
- type of distance
- type of assignment metric (majority, average, weighted metrics, etc.)

The curse of dimensionality

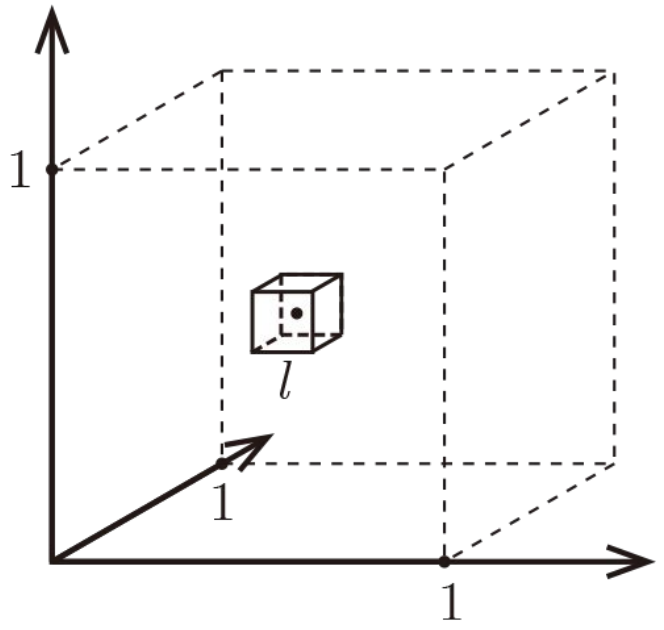
- KNN assumes that **similar (close) points share similar labels/target**
- unfortunately, **in high dimensional spaces points tend to never be close together**



From: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html

The curse of dimensionality

- increasing the number of dimensions (parameters) of the problem increases and complicates the identification of k neighbors which are close enough to the data point to be classified/predicted



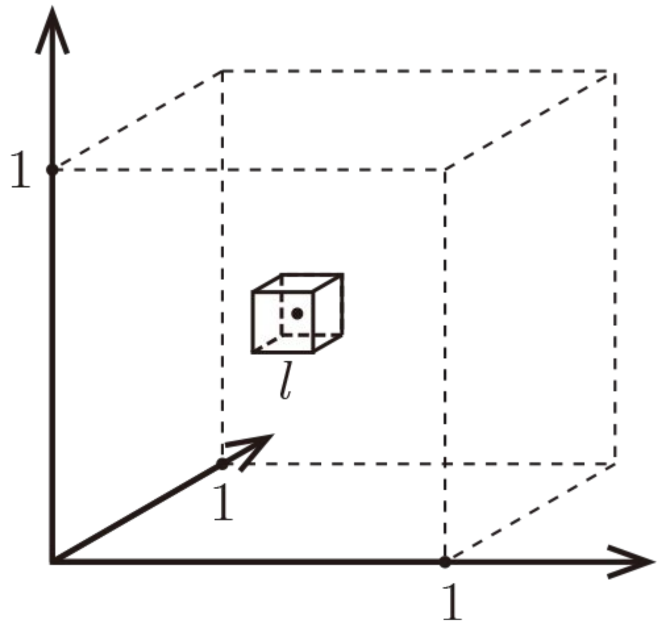
From: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html

The curse of dimensionality

$$l = \left(\frac{k}{n}\right)^{1/d}$$

- **l**: side of the hypercube that include the k neighbours
- **n**: sample size
- **d**: n. of dimensions

- increasing the number of dimensions (parameters) of the problem increases and complicates the identification of k neighbors which are close enough to the data point to be classified/predicted



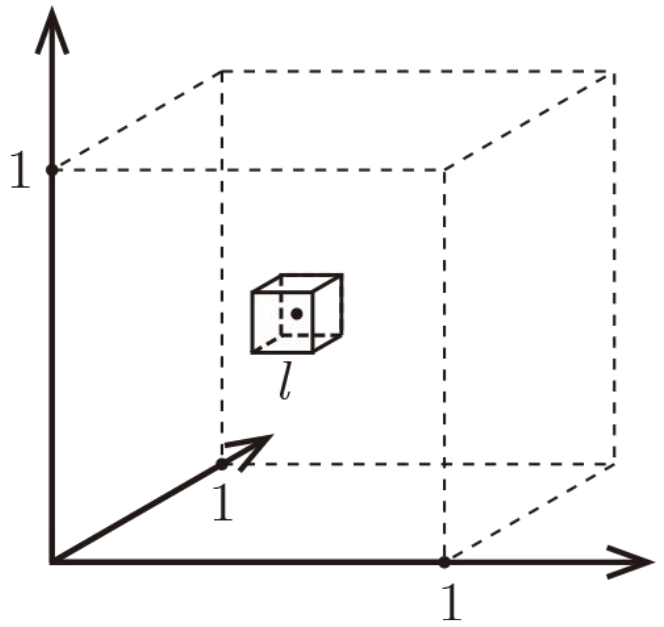
From: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html

The curse of dimensionality

$$l = \left(\frac{k}{n}\right)^{1/d}$$

- **l**: side of the hypercube that include the k neighbours
- **n**: sample size
- **d**: n. of dimensions

- with n constant (data size), **the hypercube in which the *k* neighbors lie gets bigger as *d* increases**



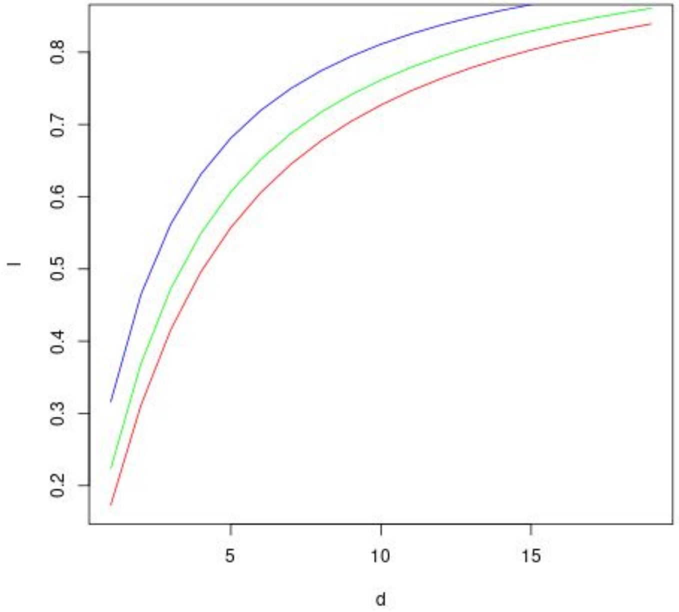
From: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html

The curse of dimensionality

$$l = \left(\frac{k}{n}\right)^{1/d}$$

- **l**: side of the hypercube that include the k neighbours
- **n**: sample size
- **d**: n. of dimensions

- with n constant (data size), **the hypercube in which the *k* neighbors lie gets bigger as *d* increases**

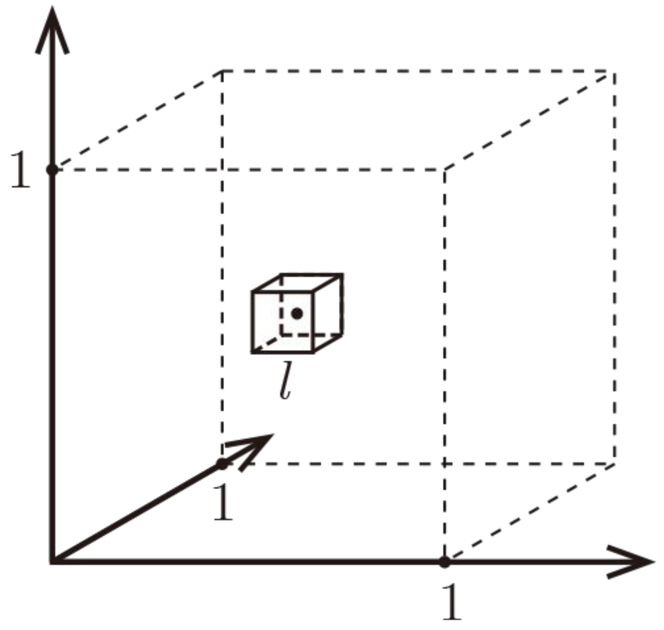


The curse of dimensionality

$$l = \left(\frac{k}{n}\right)^{1/d}$$

- **n** = 1000
- **k** = 5

d	l
2	0.071
10	0.588
50	0.899
100	0.948
1000	0.994

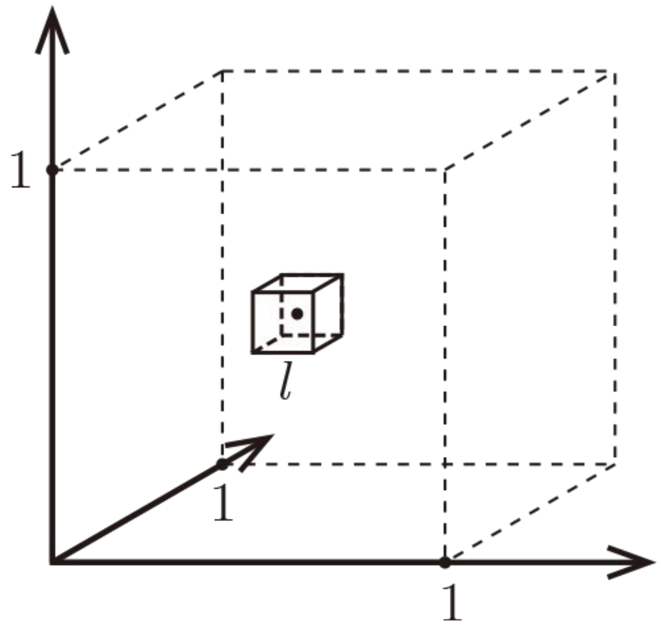


From: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html

The curse of dimensionality

- when d ≫ 2 **almost the entire space** is needed to find the k NN
- this breaks down the KNN assumptions, because **the k NN are not particularly closer (and therefore more similar) than any other data points** in the dataset
- why would the test point share the label with those k-nearest neighbors, if they are not actually similar to it?

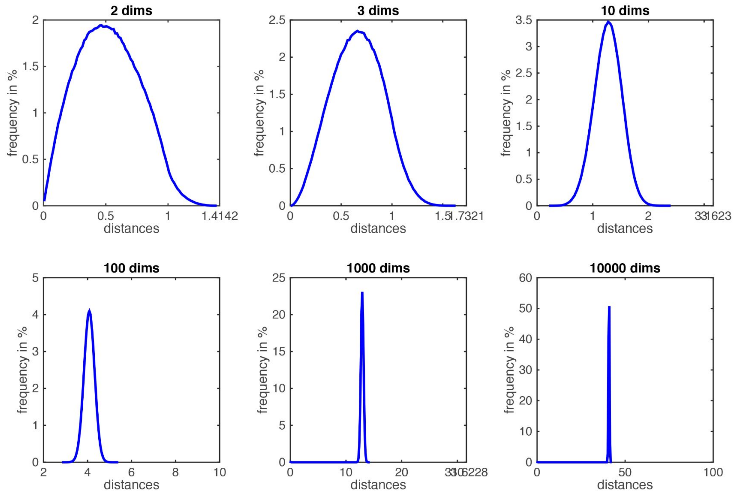
d	l
2	0.071
10	0.588
50	0.899
100	0.948
1000	0.994



From: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html

The curse of dimensionality

distributions of all pairwise distances between randomly drawn points within d-dimensional unit hypercubes: as the number of dimensions *d* grows, all distances concentrate within a very small range ("*the night where all cows are black*")



From: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html

The curse of dimensionality

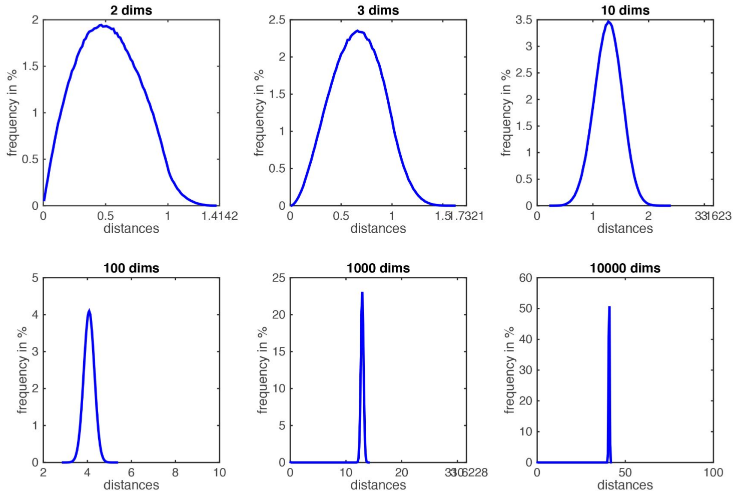
distributions of all pairwise distances between randomly drawn points within d-dimensional unit hypercubes: as the number of dimensions *d* grows, all distances concentrate within a very small range ("*the night where all cows are black*")

increase the number of samples *n* (data size)?

e.g. fix **l = 0.1**

$$n = \frac{k}{l^d} = k \cdot 10^d$$

- **exponential growth**
- for d > 100 we would need more data points than there are electrons in the universe!



From: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html

KNN: as lazy as it gets

- KNN is a **lazy algorithm**: each time new datapoints are added (e.g. to be predicted) pairwise distances with all existing datapoints (over all dimensions) must be calculated
- calculations are slow
- However:
    - when new data are available, there's no need to retrain the model (no parameters to estimate or fine-tune) → excellent for applications where data are added incrementally (e.g. on-line learning, update predictions)

What about imputation?

- Ok, we learnt about KNN, but imputation?
- The **imputation of missing SNP genotypes** is a type of prediction (slightly "*sui generis*"), where non-missing values can be considered as training observations and missing values as the test observations

NEXT LECTURE

Genotype imputation with KNN: a demonstration

(R code)