Overview

Source:
`https://slds-lmu.github.io/i2ml/chapters/14_cod/`

- How our intuitions miserably fails as the dimensionality of the data increases
- Properties of high dimensional data
- Implications for distance based algorithms (e. g. KMeans, kNN)
- How to deal with those implications
- Most importantly - what happens if we peel a high dimensional mandarin (we will also have a practical session for the 3D case)

CURSE OF DIMENSIONALITY

- The phenomenon of data becoming sparse in high-dimensional spaces is one effect of the **curse of dimensionality**.
- The **curse of dimensionality** refers to various phenomena that arise when analyzing data in high-dimensional spaces that do not occur in low-dimensional spaces.
- Our intuition about the geometry of a space is formed in two and three dimensions.
- We will see: This intuition is often misleading in high-dimensional spaces.

CURSE OF DIMENSIONALITY: EXAMPLE

To illustrate one of the problematic phenomena of data in high dimensional data, we look at an introductory example:

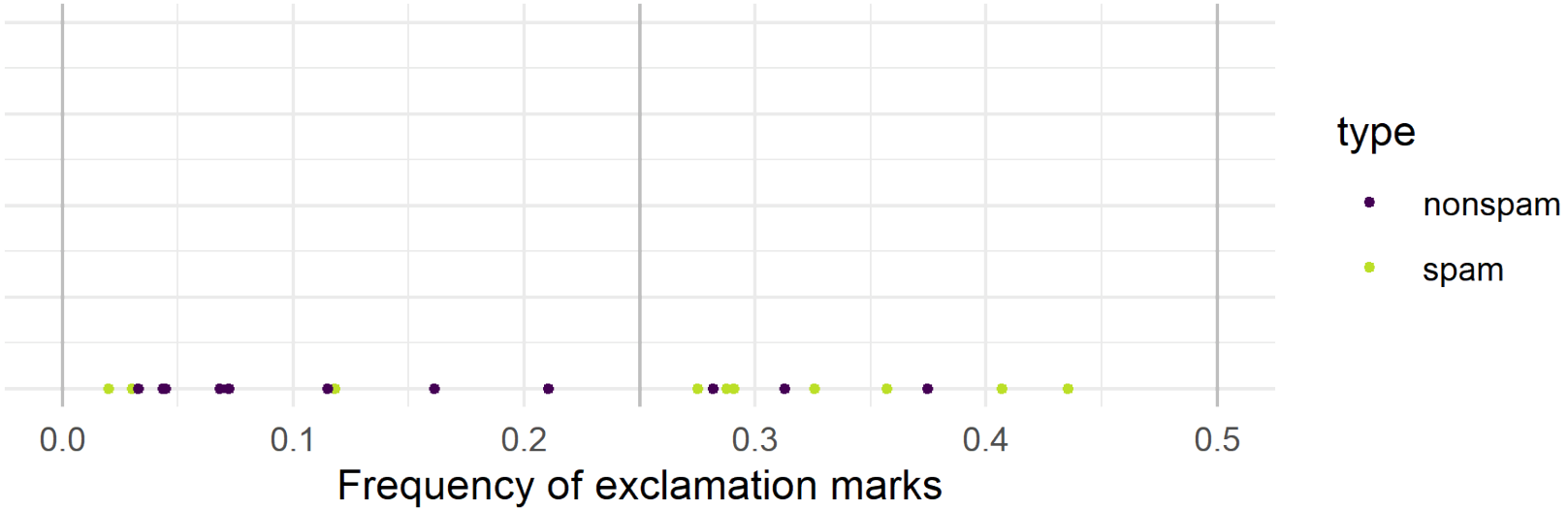
We are given 20 emails, 10 of them are spam and 10 are not.
Our goal is to predict if a new incoming mail is spam or not.

For each email, we extract the following features:

- frequency of exclamation marks (in %)
- the length of the longest sequence of capital letters
- the frequency of certain words, e.g., “free” (in %)
- ...

... and we could extract many more features!

CURSE OF DIMENSIONALITY: EXAMPLE

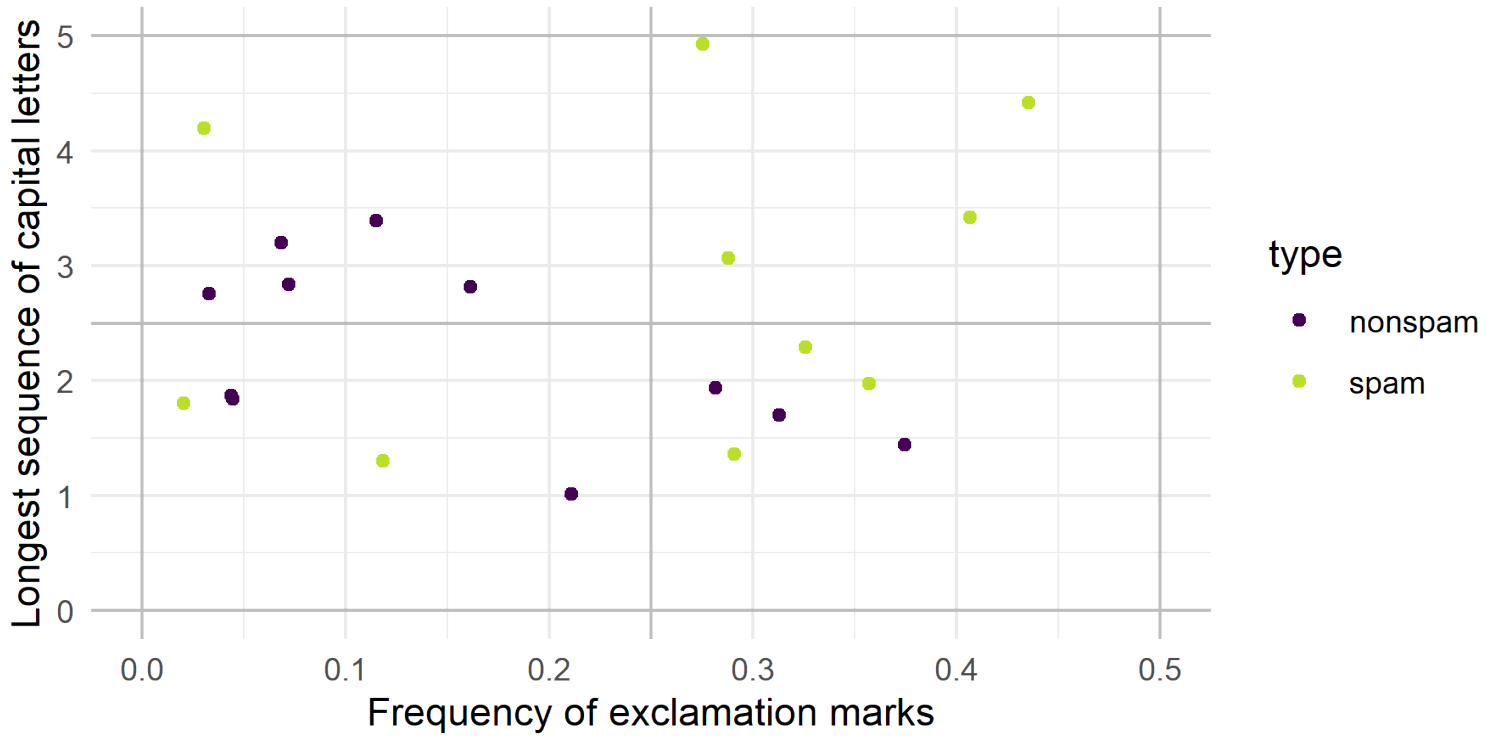


Based on the frequency of exclamation marks, we train a very simple classifier (a decision stump with split point $\mathbf{x} = 0.25$):

- We divide the input space into 2 equally sized regions.
- In the second region $[0.25, 0.5]$, 7 out of 10 are spam.
- Given that at least 0.25% of all letters are exclamation marks, an email is spam with a probability of $\frac{7}{10} = 0.7$.

CURSE OF DIMENSIONALITY: EXAMPLE

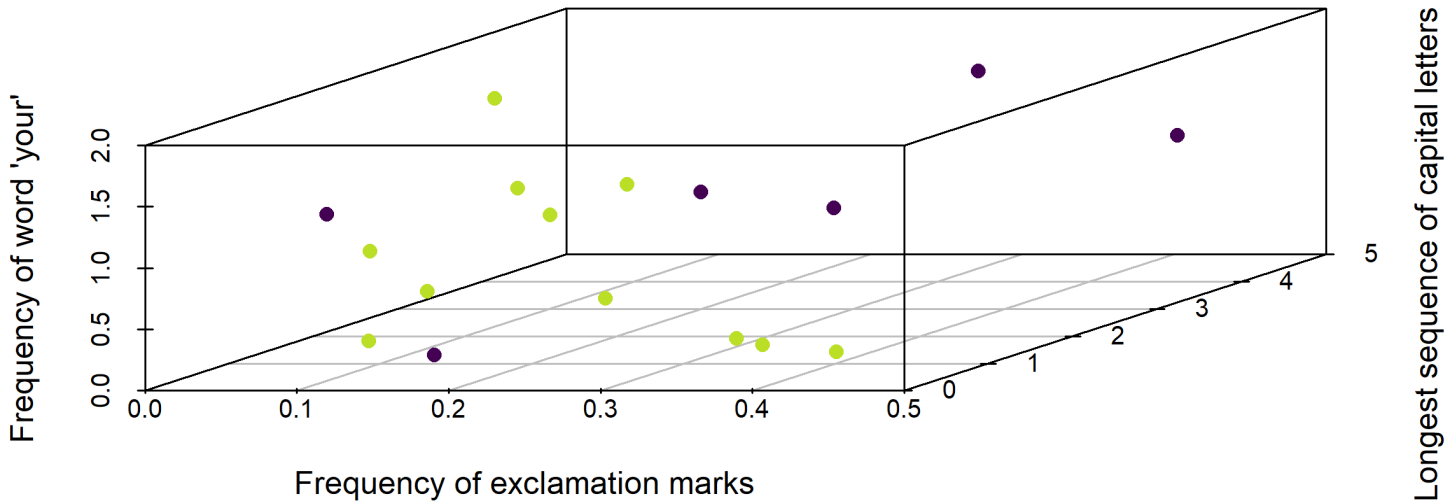
Let us feed more information into our classifier. We include a feature that contains the length of the longest sequence of capital letters.



- In the 1D case we had 20 observations across 2 regions.
- The same number is now spread across 4 regions.

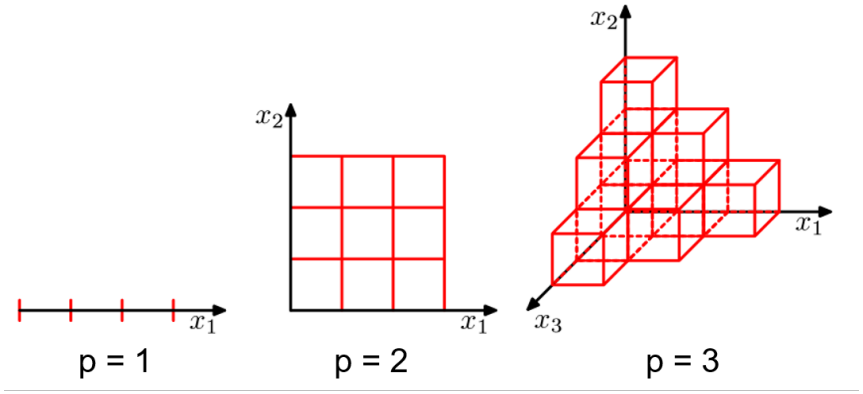
CURSE OF DIMENSIONALITY: EXAMPLE

Let us further increase the dimensionality to 3 by using the frequency of the word “your” in an email.



CURSE OF DIMENSIONALITY: EXAMPLE

- When adding a third dimension, the same number of observations is spread across 8 regions.
- In 4 dimensions the data points are spread across 16 cells, in 5 dimensions across 32 cells and so on ...
- As dimensionality increases, the data become **sparse**; some of the cells become empty.
- There might be too few data in each of the blocks to understand the distribution of the data and to model it.

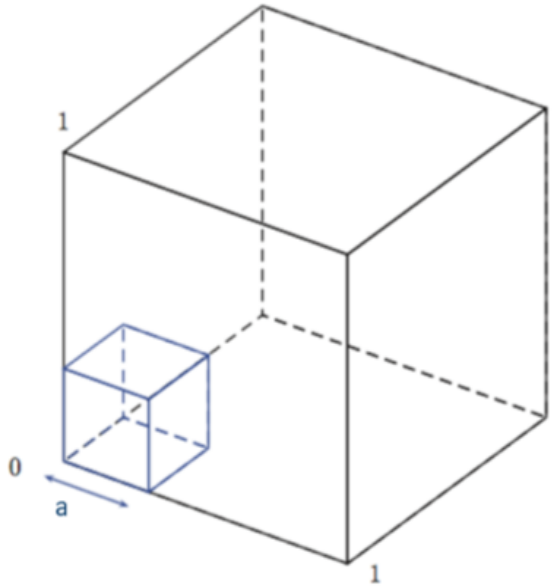


Bishop, Pattern Recognition and Machine Learning, 2006

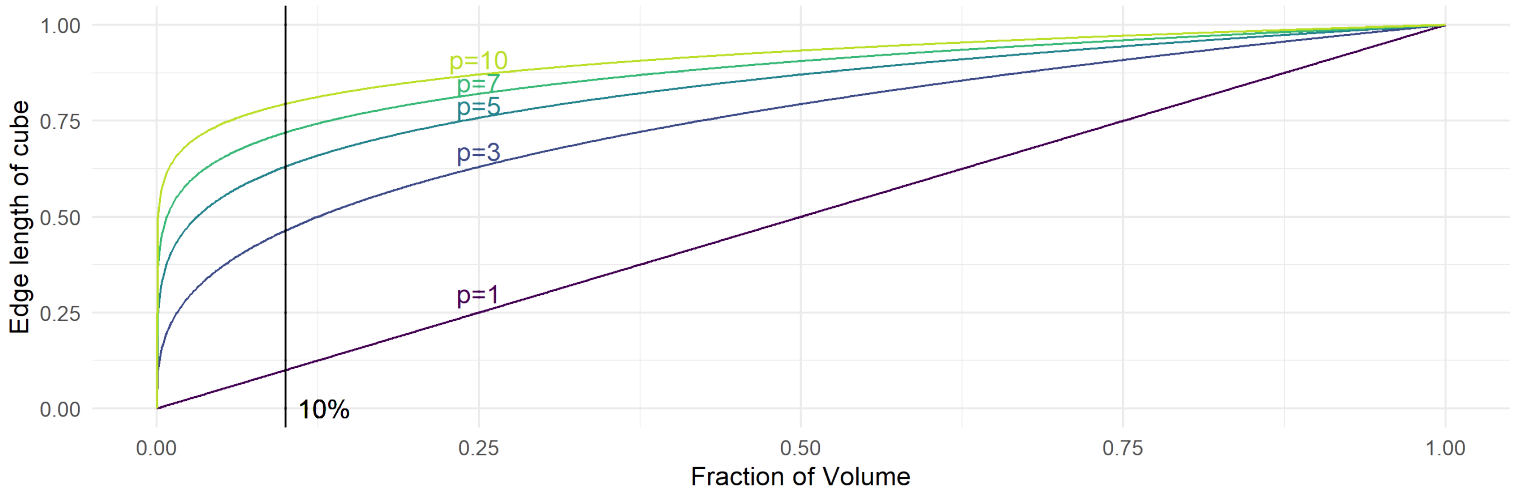
Geometry of High-Dimensional Spaces

THE HIGH-DIMENSIONAL CUBE

- We embed a small cube with edge length *a* inside a unit cube.
- How long does the edge length *a* of this small hypercube have to be so that the hypercube covers $10\%, 20\%, \ldots$ of the volume of the unit cube (volume 1)?



THE HIGH-DIMENSIONAL CUBE



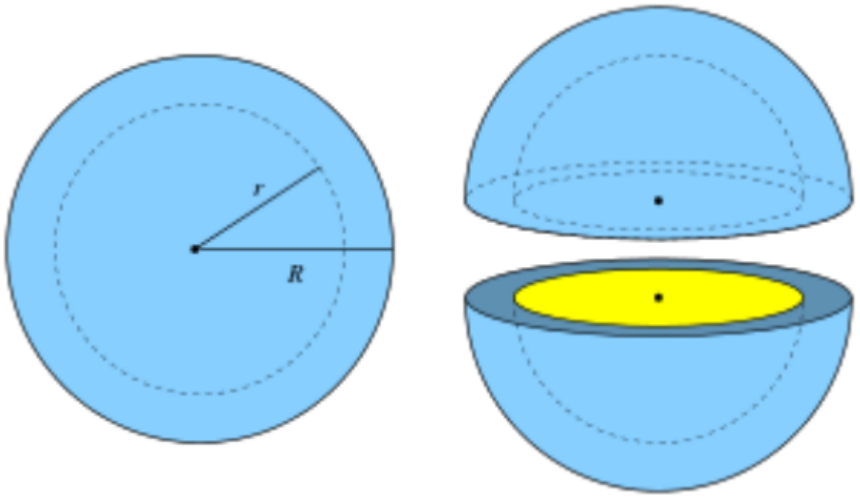
$$a^p = \frac{1}{10} \Leftrightarrow a = \frac{1}{\sqrt[p]{10}}$$

- So: covering 10% of total volume in a cell requires cells with almost 50% of the entire range in 3 dimensions, 80% in 10 dimensions.

THE HIGH-DIMENSIONAL SPHERE

Another manifestation of the **curse of dimensionality** is that the majority of data points are close to the outer edges of the sample. Consider a hypersphere of radius 1. The fraction of volume that lies in the ϵ-"edge", $\epsilon := R - r$, of this hypersphere can be calculated by the formula

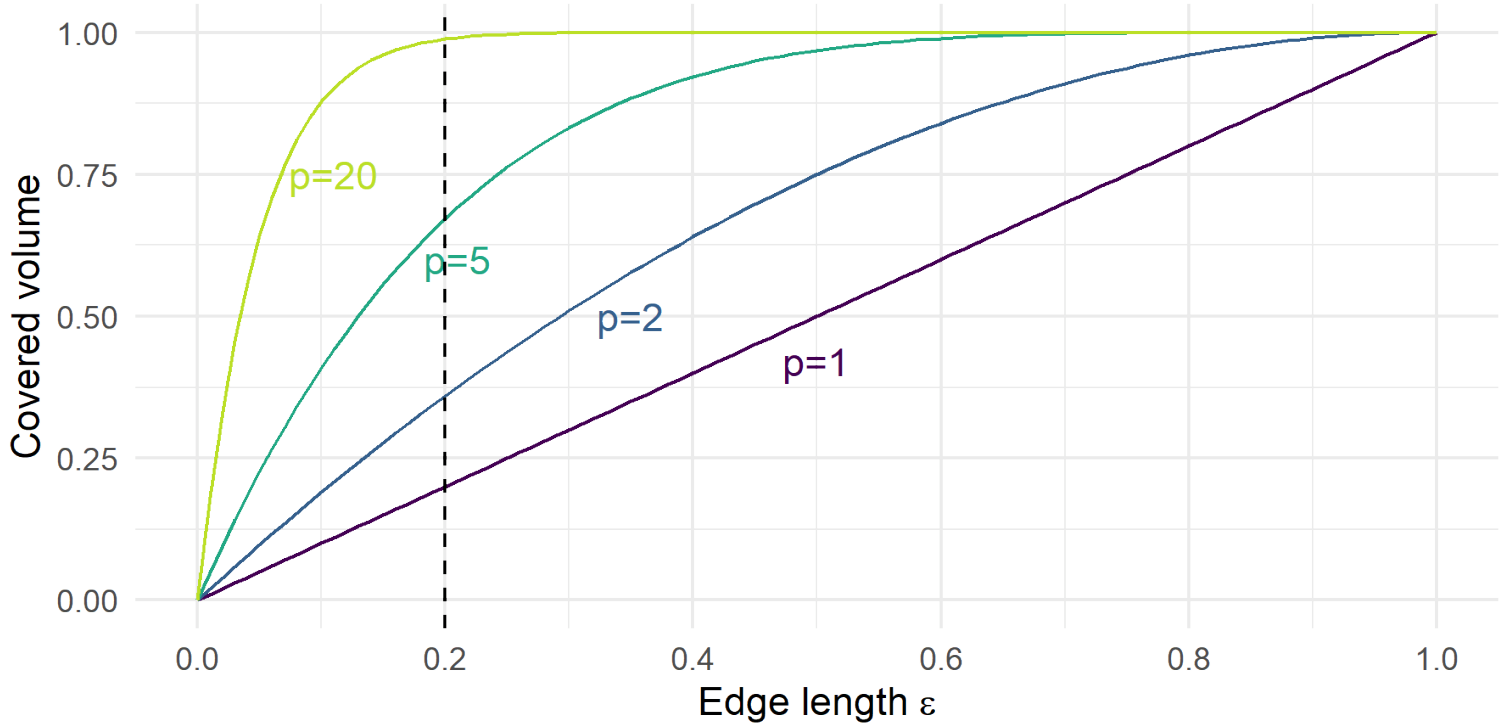
$$1 - \left(1 - \frac{\epsilon}{R}\right)^p.$$



If we peel a high-dimensional orange, there is almost nothing left.

THE HIGH-DIMENSIONAL SPHERE

Consider a 20-dimensional sphere. Nearly all of the volume lies in its outer shell of thickness 0.2:

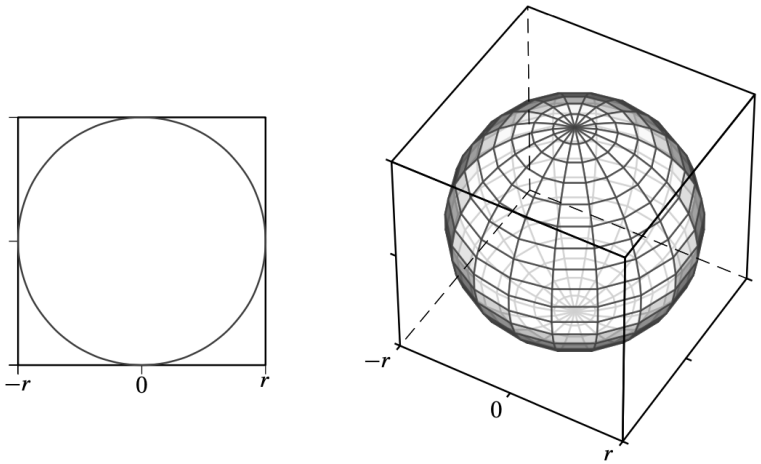


HYPHERSPHERE WITHIN HYPERCUBE

Consider a p-dimensional hypersphere of radius r and volume $S_p(r)$ inscribed in a p-dimensional hypercube with sides of length $2r$ and volume $C_p(r)$. Then it holds that

$$\lim_{p\to\infty} \frac{S_p(r)}{C_p(r)} = \lim_{p\to\infty} \frac{\left(\frac{\pi^{\frac{p}{2}}}{\Gamma(\frac{p}{2}+1)}\right) r^p}{(2r)^p} = \lim_{p\to\infty} \frac{\pi^{\frac{p}{2}}}{2^p \Gamma(\frac{p}{2}+1)} = 0,$$

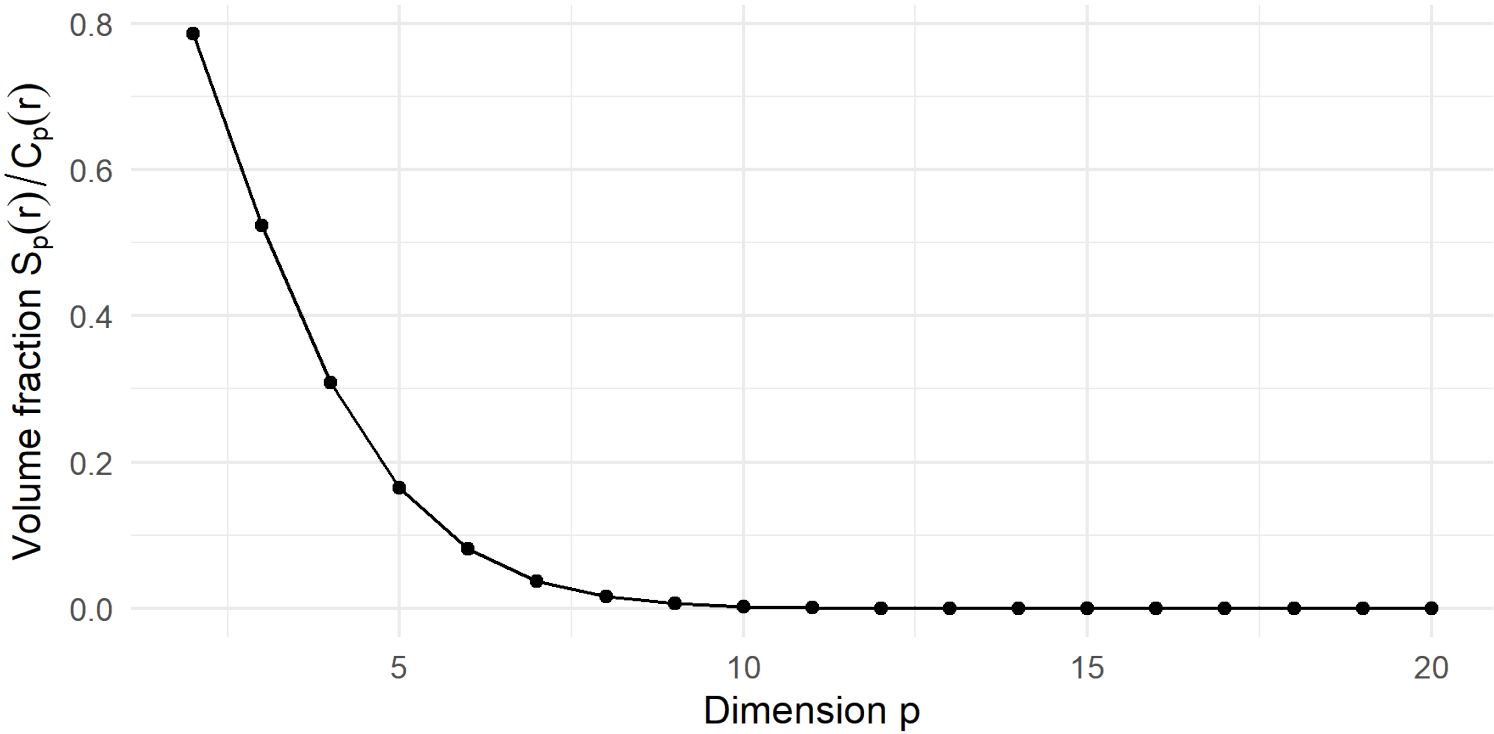
i.e., as the dimensionality increases, most of the volume of the hypercube can be found in its corners.



Mohammed J. Zaki, Wagner Meira, Jr., Data Mining and Analysis: Fundamental Concepts and Algorithms, 2014

HYPHERSPHERE WITHIN HYPERCUBE

Consider a 10-dimensional sphere inscribed in a 10-dimensional cube. Nearly all of the volume lies in the corners of the cube:



Note: For $r > 0$, the volume fraction $\frac{S_p(r)}{C_p(r)}$ is independent of r.

UNIFORMLY DISTRIBUTED DATA

The consequences of the previous results for uniformly distributed data in the high-dimensional hypercube are:

- Most of the data points will lie on the boundary of the space.
- The points will be mainly scattered on the large number of corners of the hypercube, which themselves will become very long spikes.
- Hence the higher the dimensionality, the more similar the minimum and maximum distances between points will become.
- This degrades the effectiveness of most distance functions.
- Neighborhoods of points will not be local anymore.

GAUSSIANS IN HIGH DIMENSIONS

A further manifestation of the **curse of dimensionality** appears if we consider a standard Gaussian $N_p(\mathbf{0}, \boldsymbol{I}_p)$ in p dimensions.

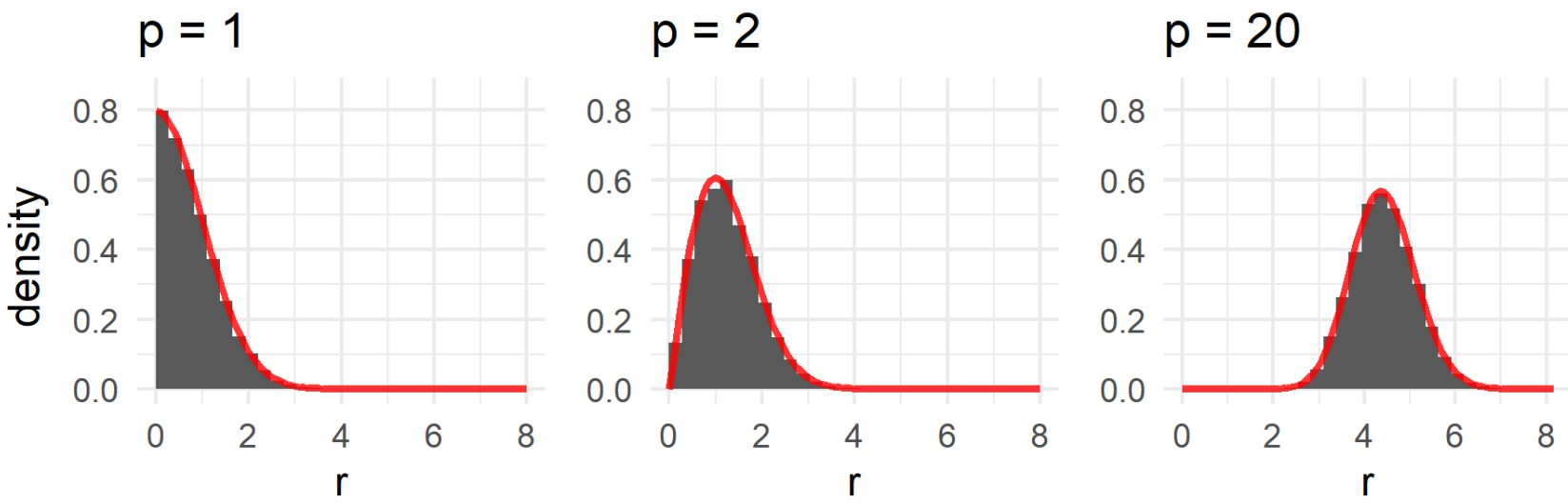
- After transforming from Cartesian to polar coordinates and integrating out the directional variables, we obtain an expression for the density $p(r)$ as a function of the radius r (i.e., the point's distance from the origin), s.t.

$$p(r) = \frac{S_p r^{p-1}}{(2\pi\sigma^2)^{p/2}} \exp\left(-\frac{r^2}{2\sigma^2}\right),$$

  where S_p is the surface area of the p-dimensional unit hypersphere.
- Thus $p(r)\delta r$ is the approximate probability mass inside a thin shell of thickness δr located at radius r.

GAUSSIANS IN HIGH DIMENSIONS

- To verify this functional relationship empirically, we draw 10^4 points from the p-dimensional standard normal distribution and plot $p(r)$ over the histogram of the points' distances to the origin:



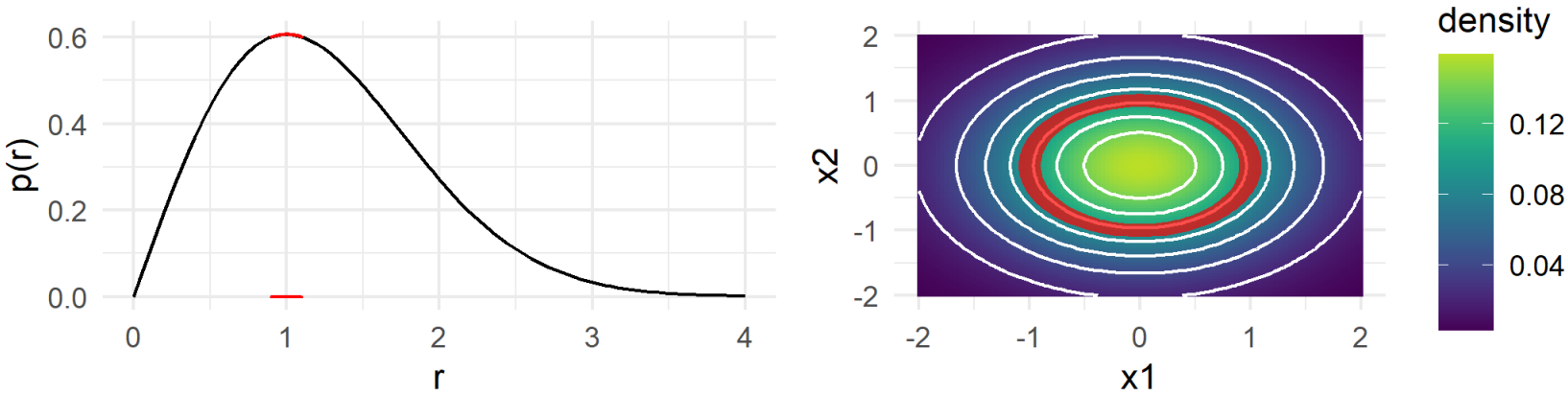
- We can see that for large p the probability mass of the Gaussian is concentrated in a fairly thin "shell" rather far away from the origin. This may seem counterintuitive, but:

GAUSSIANS IN HIGH DIMENSIONS

- For the probability mass of a hyperspherical shell it follows that

$$\int_{r-\frac{\delta r}{2}}^{r+\frac{\delta r}{2}} p(\tilde{r})d\tilde{r} = \int_{r-\frac{\delta r}{2} \leq ||\mathbf{x}||_2 \leq r+\frac{\delta r}{2}} f_p(\tilde{\mathbf{x}})d\tilde{\mathbf{x}},$$

where $f_p(\mathbf{x})$ is the density of the p-dimensional standard normal distribution and $p(r)$ the associated radial density.



Example: 2D normal distribution

- While f_p becomes smaller with increasing r, the region of the integral -the hyperspherical shell- becomes bigger.

Gaussian soap bubble: where is the typical sample?

Consider $X \sim \mathcal{N}(0, I_p)$ in p dimensions.

- The *mode* (densest point) is at the origin.
- **Question:** where do typical samples actually land?

Gaussian soap bubble: where is the typical sample?

Consider $X \sim \mathcal{N}(0, I_p)$ in p dimensions.

- The *mode* (densest point) is at the origin.
- **Question:** where do typical samples actually land?

The squared norm $\|X\|^2 \sim \chi^2_p$, so

$$\mathbb{E}\|X\| \approx \sqrt{p}, \qquad \text{std}(\|X\|) \to \text{const as } p \to \infty.$$

Typical samples sit on a thin shell at radius $\sqrt{p}$. The densest point is the *least likely* place to draw from.

Why this matters: foundation for MCMC initialization, diffusion-model noise schedules, and the "soap bubble" picture of high-dim Gaussians. The mode of a distribution is *not* where its mass lives.

EXAMPLE: K-NN

Let us look at the performance of algorithms for increasing dimensionality. First, we consider the k-NN algorithm:

- In a high dimensional space, data points are spread across a huge space.
- The distance to the **next neighbor** $d_{NN1}(\mathbf{x})$ becomes extremely large.
- The distance might even get so large that all points are **equally far** away - we cannot really determine the nearest neighbor anymore.

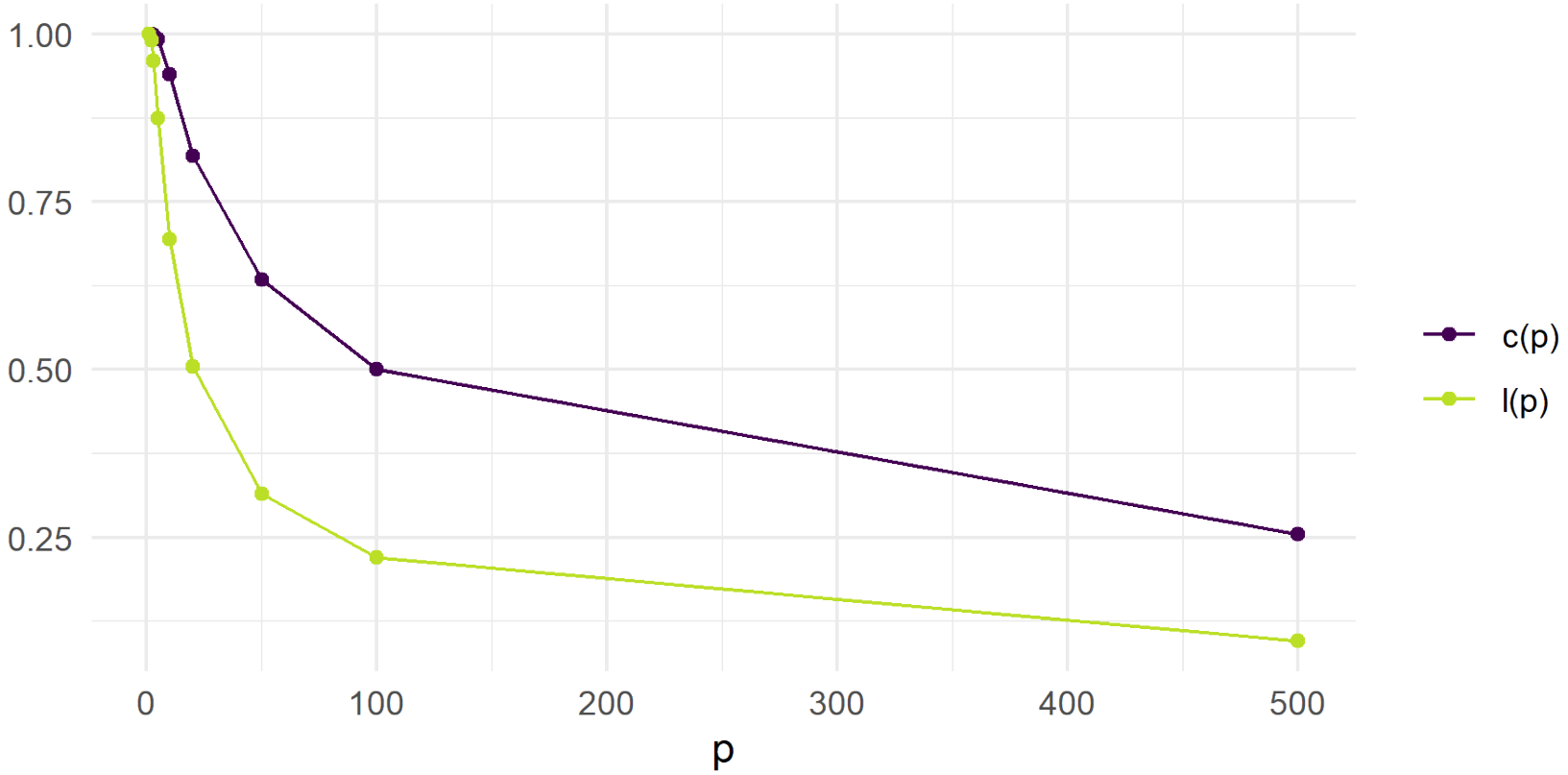
EXAMPLE: K-NN

Minimal, mean and maximal (NN)-distances of 10^4 points uniformly distributed in the hypercube $[0,1]^p$:

p	$\min d(\mathbf{x}, \tilde{\mathbf{x}})$	$\overline{d(\mathbf{x}, \tilde{\mathbf{x}})}$	$\max d(\mathbf{x}, \tilde{\mathbf{x}})$	$\overline{d_{NN1}(\mathbf{x})}$	$\max d_{NN1}(\mathbf{x})$
1	1.2e-08	0.33	1	5e-05	0.00042
2	0.00011	0.52	1.4	0.0051	0.02
3	0.0021	0.66	1.7	0.026	0.073
5	0.016	0.88	2	0.11	0.23
10	0.15	1.3	2.5	0.39	0.63
20	0.55	1.8	3	0.9	1.2
50	1.5	2.9	4.1	2	2.4
100	2.7	4.1	5.4	3.2	3.5
500	7.8	9.1	10	8.2	8.6

EXAMPLE: K-NN

We see a decrease of relative contrast[1] $c := \frac{\max(d(\mathbf{x},\tilde{\mathbf{x}})) - \min(d(\mathbf{x},\tilde{\mathbf{x}}))}{\max(d(\mathbf{x},\tilde{\mathbf{x}}))}$ and "locality"[2] $l := \frac{\overline{d(\mathbf{x},\tilde{\mathbf{x}})} - \overline{d_{NN1}(\mathbf{x})}}{\overline{d(\mathbf{x},\tilde{\mathbf{x}})}}$ with increasing number of dimensions p:



[1] [Aggarwal et al., 2001]

[2] our non-standard definition

EXAMPLE: K-NN

The consequences for the k-nearest neighbors approach can be summarized as follows:

- At constant sample size n and growing p, the distance between the observations increases
  $\rightarrow$ the coverage of the p-dimensional space decreases,
  $\rightarrow$ every point becomes isolated / far way from all other points.
- The size of the neighborhood $N_k(x)$ also "increases"
  (at constant k)
  $\rightarrow$ it is no longer a "local" method.
- Reducing k dramatically does not help much either, since the fewer observations we average, the higher the variance of our fit.

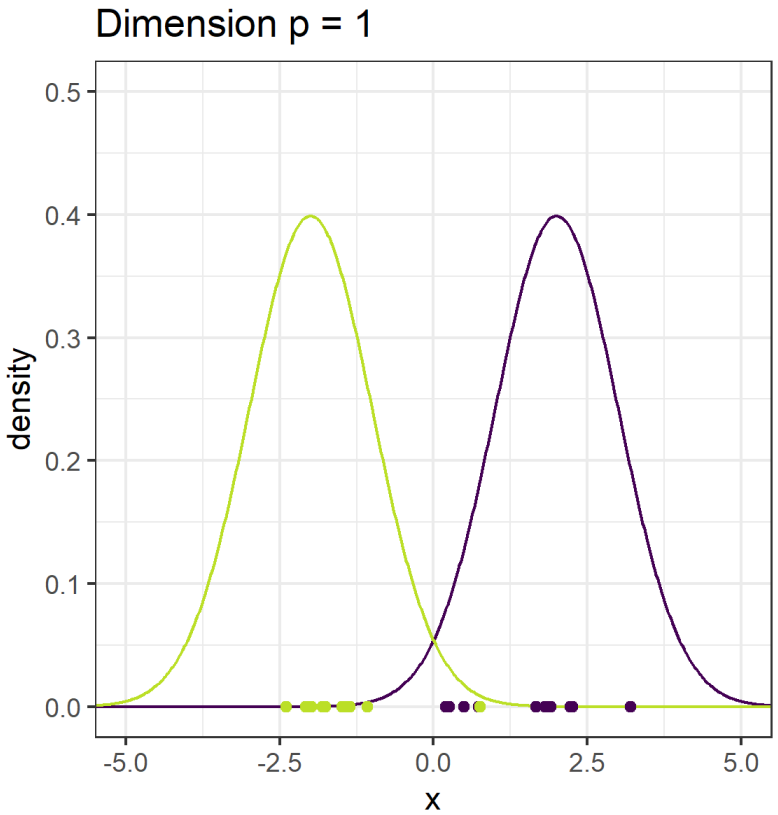
$\rightarrow$ k-NN estimates get more inaccurate with increasing dimensionality of the data.

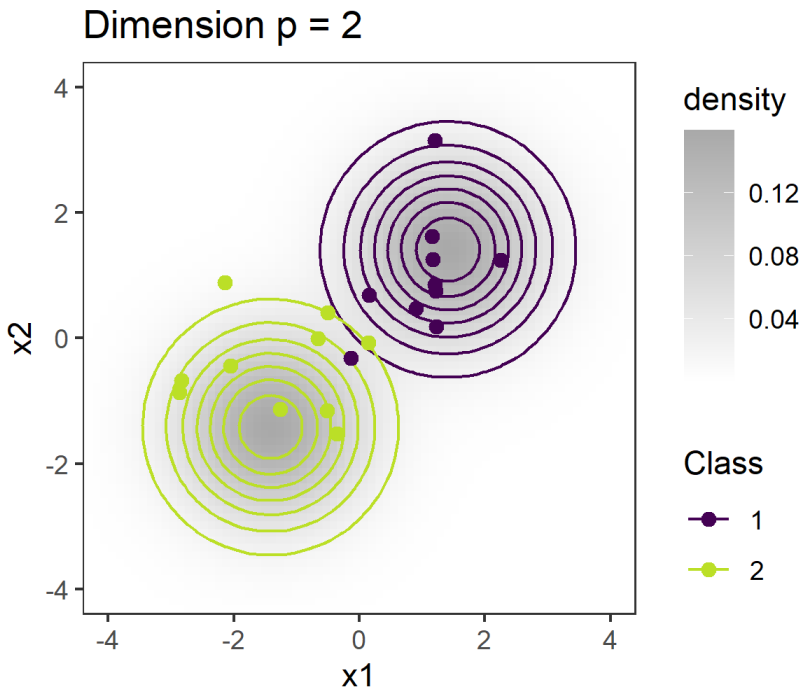
EXAMPLE: K-NN

To demonstrate this, we generate an artificial data set of dimension p as follows: We define $a = \frac{2}{\sqrt{p}}$ and

- with probability $\frac{1}{2}$ we generate a sample from class 1 by sampling from a Gaussian with mean $\boldsymbol{\mu} = (a, a, ..., a)$ and unit covariance matrix
- with probability $\frac{1}{2}$ we generate a sample from class 2 by sampling from a Gaussian with mean $-\boldsymbol{\mu} = (-a, -a, ..., -a)$ and unit covariance matrix

EXAMPLE: K-NN


Dimension p = 1
density
0.5
0.4
0.3
0.2
0.1
0.0
-5.0
-2.5
0.0
2.5
5.0
x


Dimension p = 2
x2
4
2
0
-2
-4
-4
-2
0
2
4
x1
density
0.12
0.08
0.04
Class
1
2

EXAMPLE: K-NN

This example is constructed such that the Bayes error is always constant and does not depend on the dimension p.

The Bayes optimal classifiers predicts $\hat{y} = 1$ iff

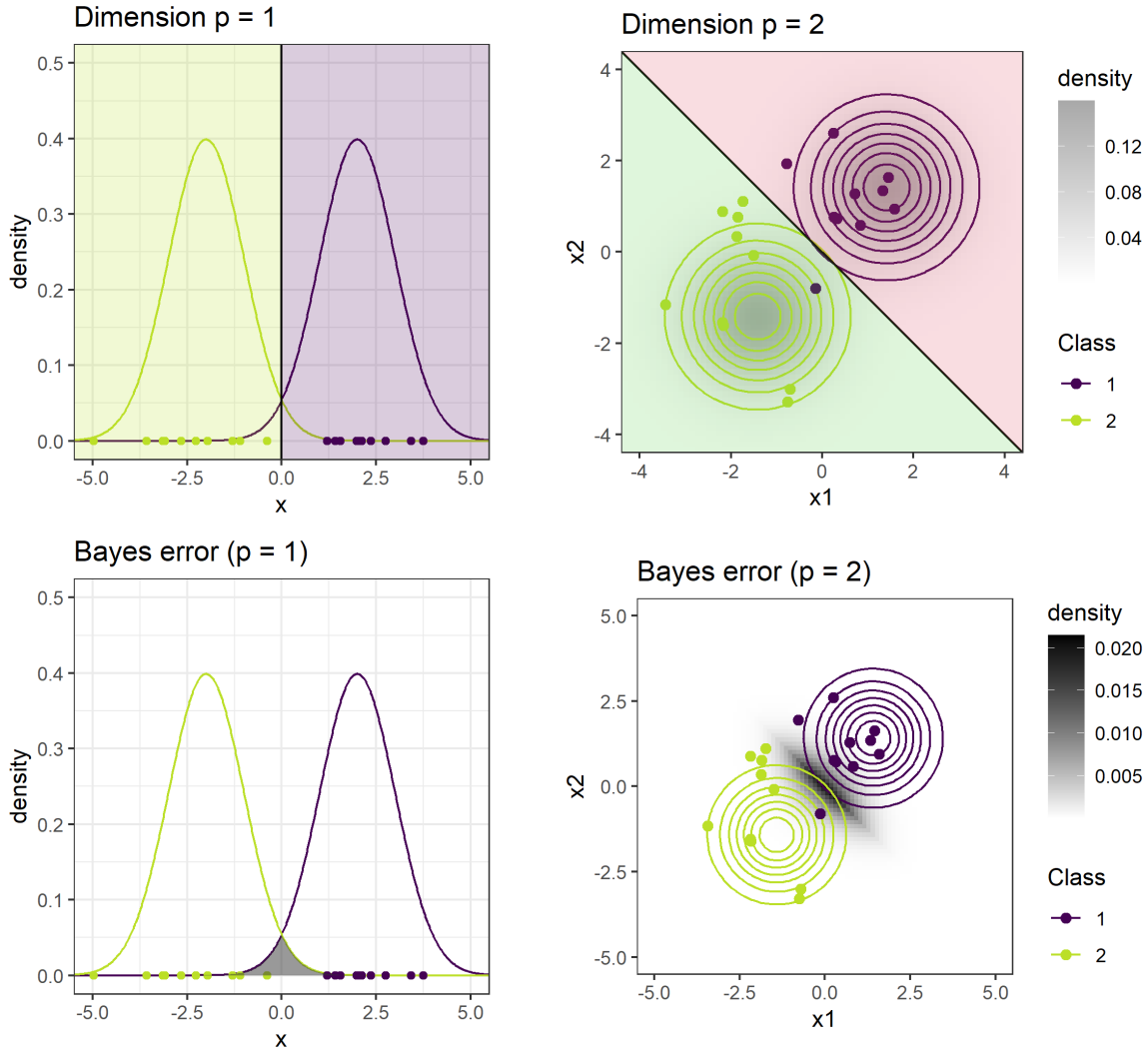
$$\begin{aligned}
\mathbb{P}(y = 1 \mid \mathbf{x}) &= \frac{p(\mathbf{x} \mid y = 1)\mathbb{P}(y = 1)}{p(\mathbf{x})} = \frac{1}{2} \cdot \frac{p(\mathbf{x} \mid y = 1)}{p(\mathbf{x})} \\
&\geq \frac{1}{2} \cdot \frac{p(\mathbf{x} \mid y = 2)}{p(\mathbf{x})} \\
&= \frac{p(\mathbf{x} \mid y = 2)\mathbb{P}(y = 2)}{p(\mathbf{x})} = \mathbb{P}(y = 2 \mid \mathbf{x}).
\end{aligned}$$

This is equivalent to

$$\begin{aligned}
\hat{y} = 1 &\Leftrightarrow \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top (\mathbf{x} - \boldsymbol{\mu})\right) \geq \exp\left(-\frac{1}{2}(\mathbf{x} + \boldsymbol{\mu})^\top (\mathbf{x} + \boldsymbol{\mu})\right) \\
&\Leftrightarrow \mathbf{x}^\top \mu \geq 0.
\end{aligned}$$

EXAMPLE: K-NN

Optimal Bayes classifier and Bayes error (shaded area):



EXAMPLE: K-NN

We can calculate the corresponding expected misclassification error (Bayes error)

$$
\begin{aligned}
& p(\hat{y} = 1 \mid y = 2)\mathbb{P}(y = 2) + p(\hat{y} = 2 \mid y = 1)\mathbb{P}(y = 1) \\
= \quad & \frac{1}{2} \cdot p(\mathbf{x}^\top \boldsymbol{\mu} \geq 0 \mid y = 2) + \frac{1}{2} \cdot p(\mathbf{x}^\top \boldsymbol{\mu} \leq 0 \mid y = 1) \\
\overset{\text{symm.}}{=} \quad & p(\mathbf{x}^\top \boldsymbol{\mu} \leq 0 \mid y = 1) = p\left(\sum_{i=1}^{p} a\mathbf{x}_i \leq 0 \mid y = 1\right) \\
= \quad & p\left(\sum_{i=1}^{p} \mathbf{x}_i \leq 0 \mid y = 1\right).
\end{aligned}
$$

$\sum_{i=1}^{p} \mathbf{x}_i \mid y = 1 \sim \mathcal{N}(p \cdot a,\ p)$, because it is the sum of independent normal random variables $\mathbf{x}_i \mid y = 1 \sim \mathcal{N}(a, 1)$ (the vector $\mathbf{x} \mid y = 1$ follows a $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{I})$ distribution with $\boldsymbol{\mu} = (a, ..., a)$).

EXAMPLE: K-NN

We get for the Bayes error:

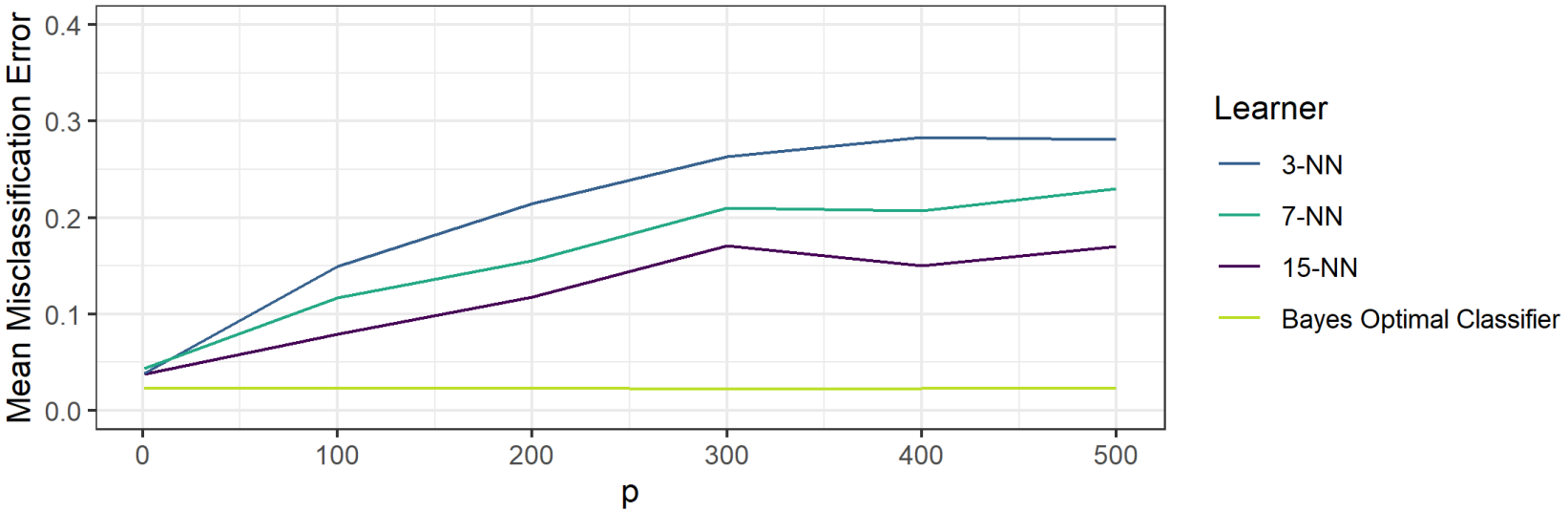
$$\begin{aligned} &= p\left(\frac{\sum_{i=1}^{p} \mathbf{x}_i - p \cdot a}{\sqrt{p}} \leq \frac{-p \cdot a}{\sqrt{p}} \mid y = 1\right) \\ &= \Phi(-\sqrt{p}a) \overset{a=\frac{2}{\sqrt{p}}}{=} \Phi(-2) \approx 0.0228, \end{aligned}$$

where Φ is the distribution function of a standard normal random variable.

We see that the Bayes error is independent of p.

EXAMPLE: K-NN

We also train a k-NN classifier for $k = 3, 7, 15$ for increasing dimensions and monitor its performance (evaluated by 10 times repeated 10-fold CV).



$\rightarrow$ k-NN deteriorates quickly with increasing dimension

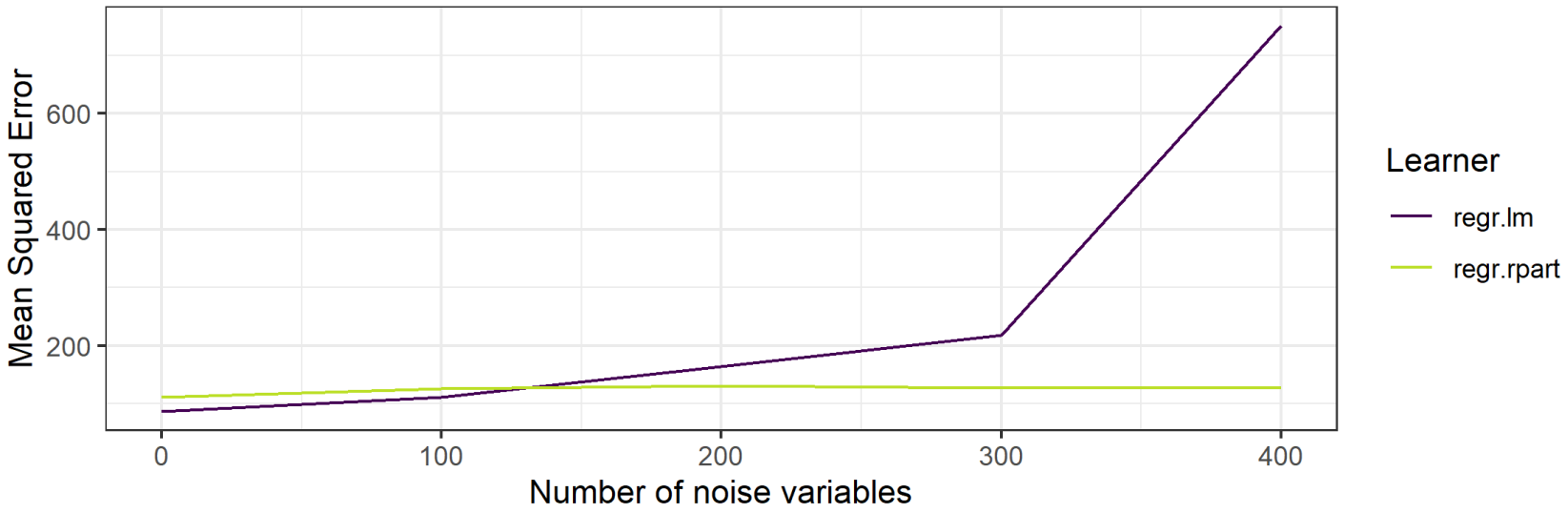
EXAMPLE: LINEAR MODEL

We also investigate how the linear model behaves in high dimensional spaces.

- We take the Boston Housing data set, where the value of houses in the area around Boston is predicted based on 13 features describing the region (e.g., crime rate, status of the population, etc.).
- We train a linear model on the data consisting of 506 observations.
- We artificially create a high-dimensional dataset by adding $100, 200, 300, \ldots$ noise variables (containing no information at all) and look at the performance of a linear model trained on this modified data (10 times repeated 10-fold CV).

EXAMPLE: LINEAR MODEL

We compare the performance of an LM to that of a regression tree.



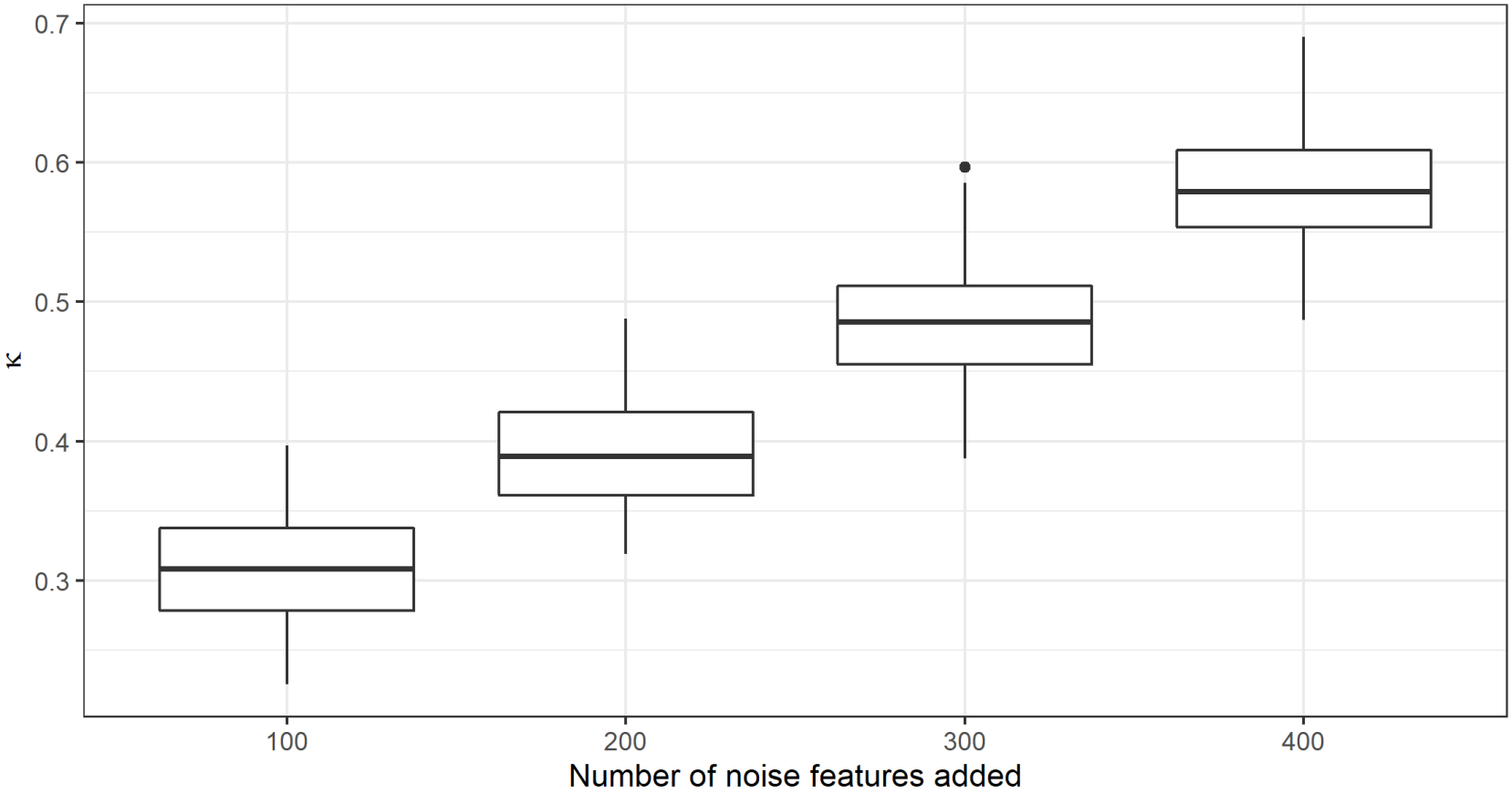
$\rightarrow$ The unregularized LM struggles with the added noise features, while our tree seems to nicely filter them out.

Note: Trees automatically perform feature selection as only one feature at a time is considered for splitting (the smaller the depth of the tree, the less features are selected). Thus, they often perform well in high-dimensional settings.

EXAMPLE: LINEAR MODEL

- The regression coefficients of the noise features can not be estimated precisely as zero in the unregularized LM due to small random correlations.
- With an increasing number of these noise features, the prediction error rises.
- To see this, we can quantify the influence of the noise features on the prediction of each observation.
  Therefore we decompose the response $\hat{y}^{(i)}$ of each iterations' test set into $\hat{y}^{(i)}_{\text{true}}$ (predicted with noise features set to 0) and $\hat{y}^{(i)}_{\text{noise}}$ (predicted with true features set to 0), s.t.
  $\hat{y}^{(i)} = \hat{y}^{(i)}_{\text{true}} + \hat{y}^{(i)}_{\text{noise}} + \text{intercept}$.
  With this, we can define the "average proportional influence of the noise features" $\kappa := \overline{\left(\frac{|\hat{y}^{(i)}_{\text{noise}}|}{|\hat{y}^{(i)}_{\text{true}}| + |\hat{y}^{(i)}_{\text{noise}}|} \right)}$.

EXAMPLE: LINEAR MODEL



When we add 400 noise features to the model, most of the time, on average, over 50% of the flexible part of the prediction ($\hat{y}^{(i)}$ − intercept) is determined by the noise features.

COD: WAYS OUT

Many methods besides k-NN struggle with the curse of dimensionality. A large part of ML is concerned with dealing with this problem and finding ways around it.

Possible approaches are:

- Increasing the space coverage by gathering more observations (not always viable in practice!)
- Reducing the number of dimensions before training (e.g. by using domain knowledge, PCA or feature selection)
- Regularization

Hughes peaking: more features can hurt

Setup: fix training set size n. Vary number of features p. Train a classifier and measure accuracy.

Predict: how does accuracy depend on p?

Hughes peaking: more features can hurt

Setup: fix training set size n. Vary number of features p. Train a classifier and measure accuracy.

Predict: how does accuracy depend on p?

- Accuracy rises with p (more information).
- Then it **peaks**.
- Then it *degrades* – even though we keep adding "information."

This is the *Hughes phenomenon* (1968). The Boston Housing + noise features experiment was exactly this.

Takeaway: throwing away features can improve accuracy. The optimal p depends on n and the signal-to-noise ratio of each feature. This is why feature selection is not optional.

Why do we use cosine similarity instead of the Euclidean distance?

Why do we use cosine similarity? (answer)

The puzzle: in high dimensions, Euclidean distances collapse together (we saw this for k-NN). So how can any distance-based method work?

Why do we use cosine similarity? (answer)

The puzzle: in high dimensions, Euclidean distances collapse together (we saw this for k-NN). So how can any distance-based method work?

Key insight: distances concentrate, but *angles* still discriminate.

- Radial coordinate $\|x\|$ concentrates around $\sqrt{p}$ – noisy, uninformative.
- Direction $x/\|x\|$ keeps the signal.

For unit vectors x, y:

$$\cos(\theta) = 1 - \tfrac{1}{2}\|x - y\|^2.$$

Same information in principle – but cosine ignores the concentrated radius. For TF-IDF and word/sentence embeddings: document length is irrelevant, word-direction carries the meaning.

Random vectors are (almost) orthogonal

Draw two vectors uniformly from the unit sphere S^{p-1}. What is the angle between them?

Random vectors are (almost) orthogonal

Draw two vectors uniformly from the unit sphere S^{p-1}. What is the angle between them?

The angle concentrates at 90°, with standard deviation $\approx 1/\sqrt{p}$.

Stronger statement: you can pack $e^{c \cdot p}$ *almost-orthogonal* vectors into $\mathbb{R}^p$ (pairwise $|\langle u, v \rangle| < \epsilon$). Compare to only p truly orthogonal vectors.

Why this matters – LLM superposition: GPT-style models use embeddings of dimension $d \approx 12{,}288$. They store millions of "concepts" by exploiting near-orthogonality – many more concepts than the embedding dimension, with minimal interference between them.

The curse becomes a *feature*.

The manifold hypothesis

A 256×256 RGB image lives in $\mathbb{R}^{196,608}$. But the set of "natural images" – photos of faces, of Yerevan streets, of cats – is a vanishingly thin *manifold* of effective dimension maybe a few thousand.

Same for:

- Sentences (vs. random character strings)
- Audio of speech (vs. white noise)
- Gene expression patterns of real cells (vs. arbitrary $\mathbb{R}^{20000}$ vectors)

The curse is real for arbitrary high-dim data. **Real data is structured.** Deep networks learn the manifold instead of the ambient space – that is why they work where k-NN dies.

Bridge to representation learning: the next chapter.