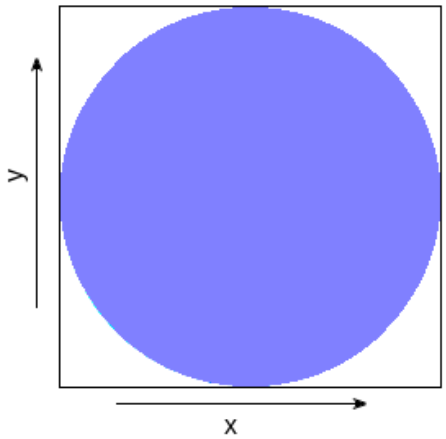
ASTR 3890 - Selected Topics: Data Science for Large Astronomical Surveys (Spring 2022)

Dimensionality Reduction & Density Estimation

Dr. Nina Hernitschek
April 11, 2022

Motivation: The *Curse of Dimensionality*

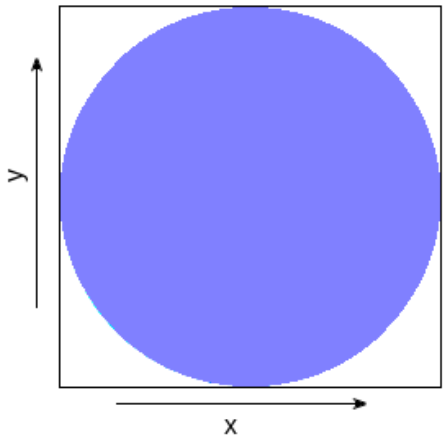
We inscribe a circle with radius r in a square.

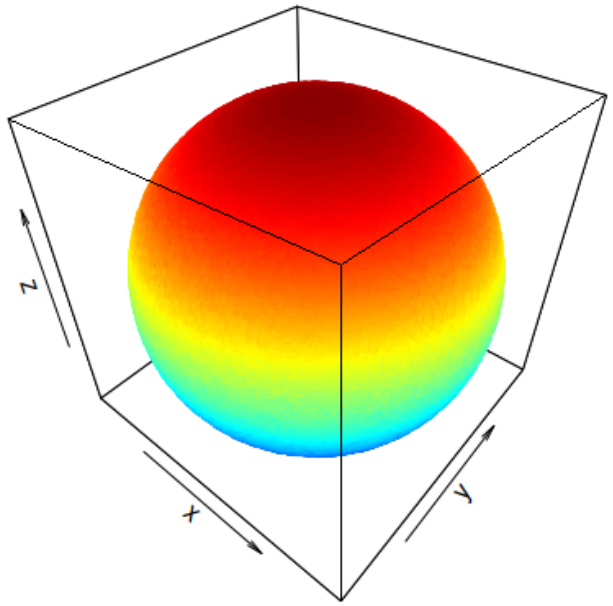


The fraction of area is: $\frac{V_{circle}}{V_{square}} = \frac{\pi r^2}{(2r)^2}$

Motivation: The *Curse of Dimensionality*

We inscribe a sphere with radius r in a cube.



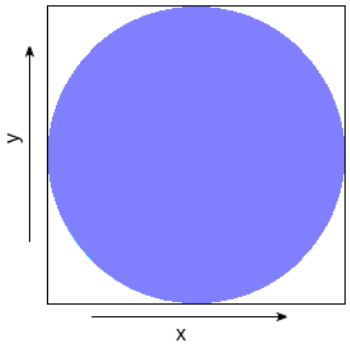


$$\frac{V_{circle}}{V_{square}} = \frac{\pi r^2}{(2r)^2}$$

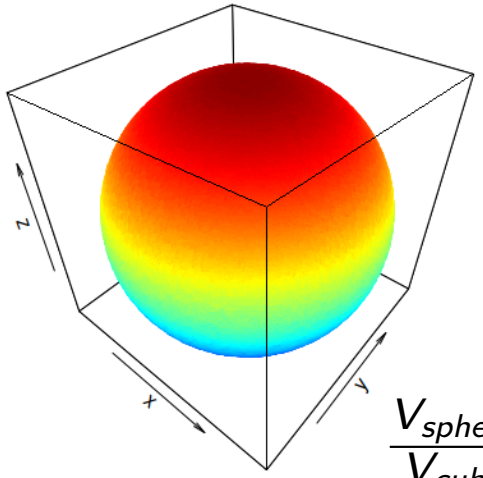
The fraction of volume is: $\frac{V_{sphere}}{V_{cube}} = \frac{4/3\pi r^3}{(2r)^3}$

Motivation: The *Curse of Dimensionality*

We **generalize** this:



$$\frac{V_{circle}}{V_{square}} = \frac{\pi r^2}{(2r)^2}$$



$$\frac{V_{sphere}}{V_{cube}} = \frac{4/3\pi r^3}{(2r)^3}$$

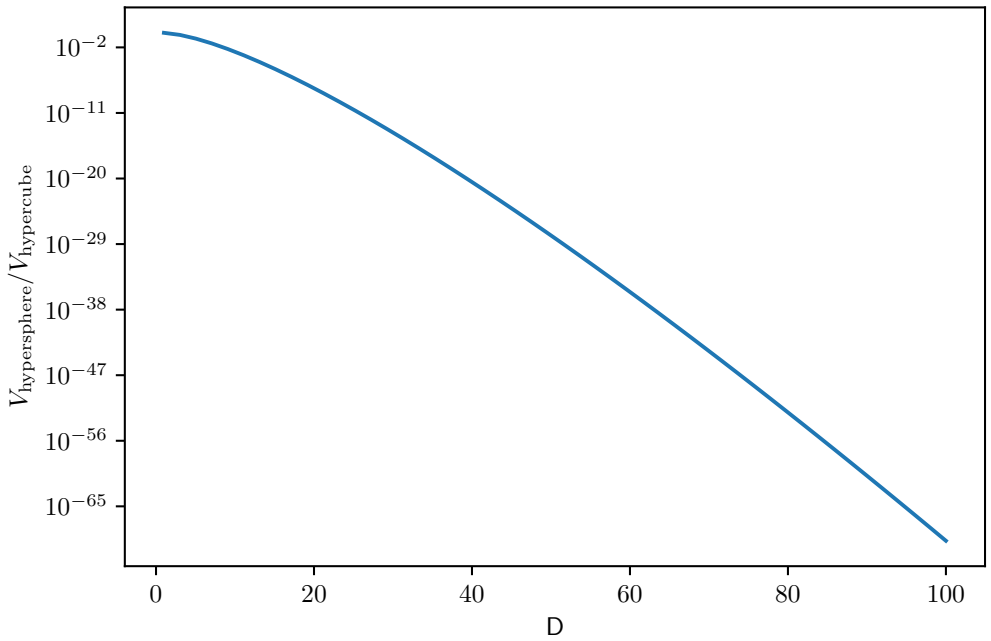
For D dimensions, the volume of hypersphere (with radius r) becomes $V_{hypersphere} = \frac{2r^D \pi^{D/2}}{D\,\Gamma(D/2)}$, with Γ being the Gamma function.

The volume of a hypercube becomes $V_{hypercube} = (2r)^D$. Thus the fraction becomes

$$\frac{V_{hypersphere}}{V_{hypercube}} = \frac{\pi^{D/2}}{D2^{D-1}\Gamma(d/2)}$$

Motivation: The *Curse of Dimensionality*

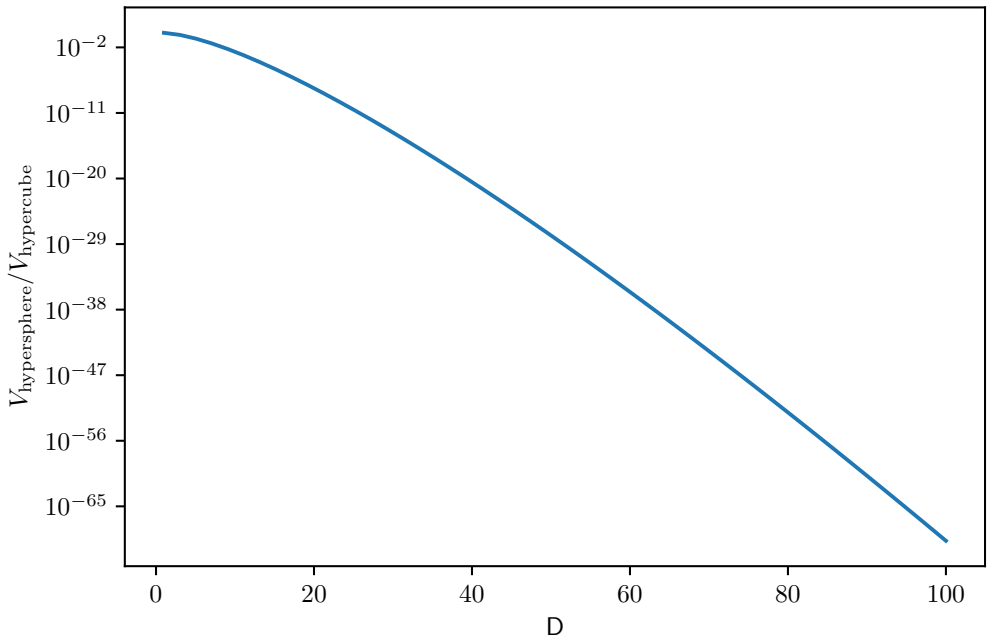
from numerical calculation:



$\frac{V_{hypersphere}}{V_{hypercube}}$ goes to 0 as D goes to infinity

Motivation: The *Curse of Dimensionality*

from numerical calculation:



$\frac{V_{hypersphere}}{V_{hypercube}}$ goes to 0 as D goes to infinity

The area outside of the circle (sphere, hypersphere...) grows larger and larger as the number of dimensions increases.

Motivation: The *Curse of Dimensionality*

Mathematically we can describe this as: the more **dimensions** that your data span, the **more points needed to uniformly sample the space**.

Motivation: The *Curse of Dimensionality*

Mathematically we can describe this as: the more **dimensions** that your data span, the **more points needed to uniformly sample the space**.

The curse of dimensionality refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces that do not occur in low-dimensional settings. The expression was coined by Richard E. Bellman when considering problems in dynamic programming.

Dimensionally cursed phenomena occur in domains such as numerical analysis, sampling, combinatorics, machine learning, data mining and databases.

Motivation: The *Curse of Dimensionality*

another ***example***:

The **Hughes Phenomenon** (also called the **peaking phenomenon**) states that for fixed number of training samples, the average (expected) predictive power of a classifier or regressor first increases as the number of dimensions or features used is increased but beyond a certain dimensionality it starts deteriorating instead of improving steadily.

Motivation: The *Curse of Dimensionality*

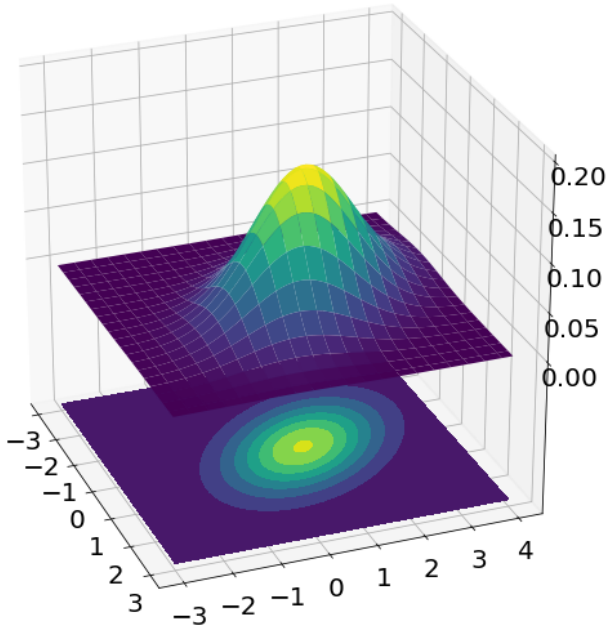
another ***example***:

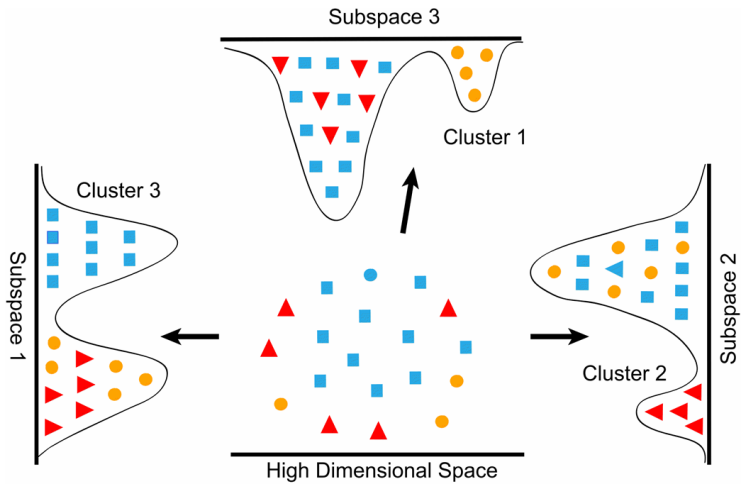
The **Hughes Phenomenon** (also called the **peaking phenomenon**) states that for fixed number of training samples, the average (expected) predictive power of a classifier or regressor first increases as the number of dimensions or features used is increased but beyond a certain dimensionality it starts deteriorating instead of improving steadily.

Issues that arise with high dimensional data are:

- Running a risk of overfitting the machine learning model.
- Difficulty in clustering similar features.
- Increased space and computational time complexity.

Dimensionality Reduction



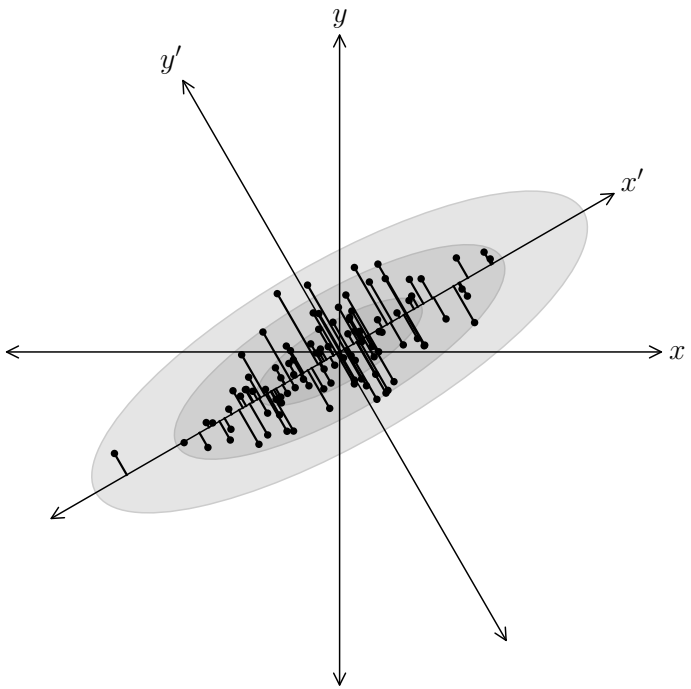


Dimensionality Reduction: Principal Component Analysis

Principal Component Analysis (PCA) is a an **unsupervised** method dimensionality reduction. A transformation is applied to the data such that the new coordinate axes are aligned with the maximal variance of the data.

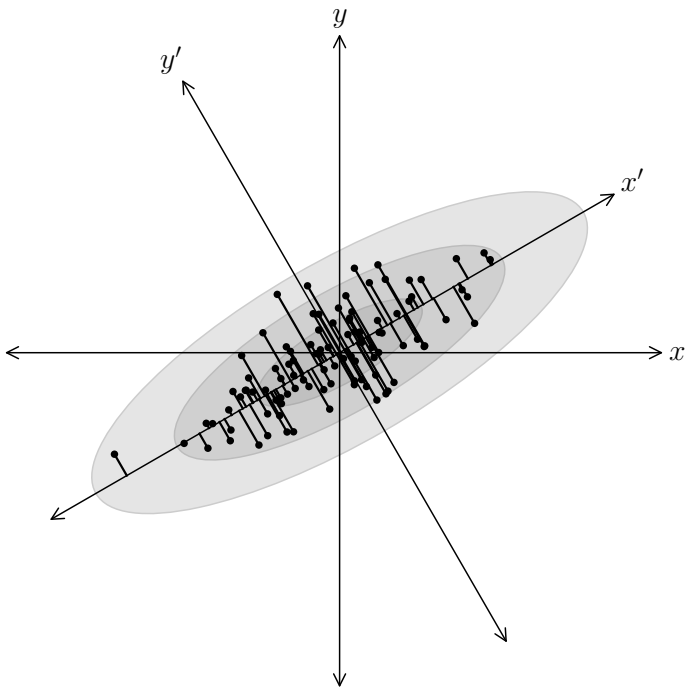
Dimensionality Reduction: Principal Component Analysis

Principal Component Analysis (PCA) is a an **unsupervised** method dimensionality reduction. A transformation is applied to the data such that the new coordinate axes are aligned with the maximal variance of the data.



Technically, it involves the process of finding the principal components, i.e. the decomposition of the feature matrix into eigenvectors. PCA transformations are linear transformations, thus it will not be effective with non-linear distributions.

Dimensionality Reduction: Principal Component Analysis



The rotation is chosen to maximize the ability to discriminate between the data points:

- **principal component** is the direction of maximal variance
- second principal component is orthogonal to the first component and maximizes the residual variance

Dimensionality Reduction: Principal Component Analysis

We can define the whole process of PCA into just four steps:

1. Standardization: The data has to be transformed to a common scale by taking the difference between the original dataset with the mean of the whole dataset. This will make the distribution 0 centered.

Dimensionality Reduction: Principal Component Analysis

We can define the whole process of PCA into just four steps:

1. Standardization: The data has to be transformed to a common scale by taking the difference between the original dataset with the mean of the whole dataset. This will make the distribution 0 centered.

2. Finding covariance: Covariance helps to understand the relationship between the mean and original data.

Dimensionality Reduction: Principal Component Analysis

We can define the whole process of PCA into just four steps:

1. Standardization: The data has to be transformed to a common scale by taking the difference between the original dataset with the mean of the whole dataset. This will make the distribution 0 centered.

2. Finding covariance: Covariance helps to understand the relationship between the mean and original data.

3. Determining the principal components: Principal components can be determined by calculating the eigenvectors and eigenvalues. Eigenvectors are a special set of vectors that help us to understand the structure and the property of the data that would be principal components. The eigenvalues on the other hand help us to determine the principal components. The highest eigenvalues and their corresponding eigenvectors make the most important principal components.

Dimensionality Reduction: Principal Component Analysis

We can define the whole process of PCA into just four steps:

1. Standardization: The data has to be transformed to a common scale by taking the difference between the original dataset with the mean of the whole dataset. This will make the distribution 0 centered.

2. Finding covariance: Covariance helps to understand the relationship between the mean and original data.

3. Determining the principal components: Principal components can be determined by calculating the eigenvectors and eigenvalues. Eigenvectors are a special set of vectors that help us to understand the structure and the property of the data that would be principal components. The eigenvalues on the other hand help us to determine the principal components. The highest eigenvalues and their corresponding eigenvectors make the most important principal components.

4. Final output: It is the dot product of the standardized matrix and the eigenvector. Note that the number of columns or features will be changed.

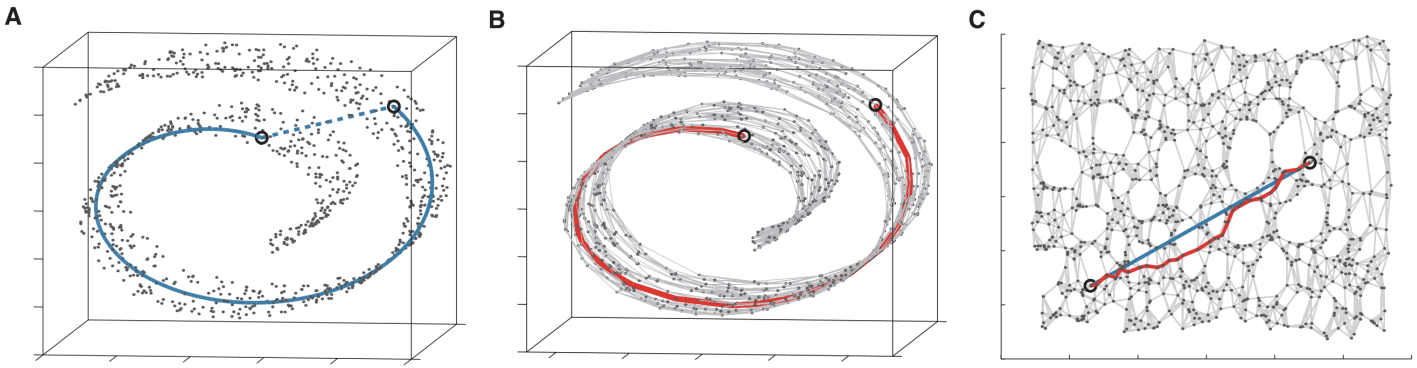
Dimensionality Reduction: Principal Component Analysis

Caveats:

- PCA is a linear process, whereas the variations in the data may not be. So it may not always be appropriate to use and/or may require a relatively large number of components to fully describe any non-linearity.
- PCA can be very impractical for large data sets which exceed the memory per core as the computational requirement goes as $\mathscr{O}(D^3)$ and the memory requirement goes as $\mathscr{O}(2D^2)$.

Dimensionality Reduction: Isometric Mapping

Isometric Mapping (IsoMap) is based on the multi-dimensional scaling (MDS) framework. Rather than using Euclidean distances between points, IsoMap approximates geodesic curves within in the embedded manifold to compute the distances between each point in the data set.



left: the Euclidean distance (dashed line) not truely reflects how far apart they are on the embedded manifold (solid line)

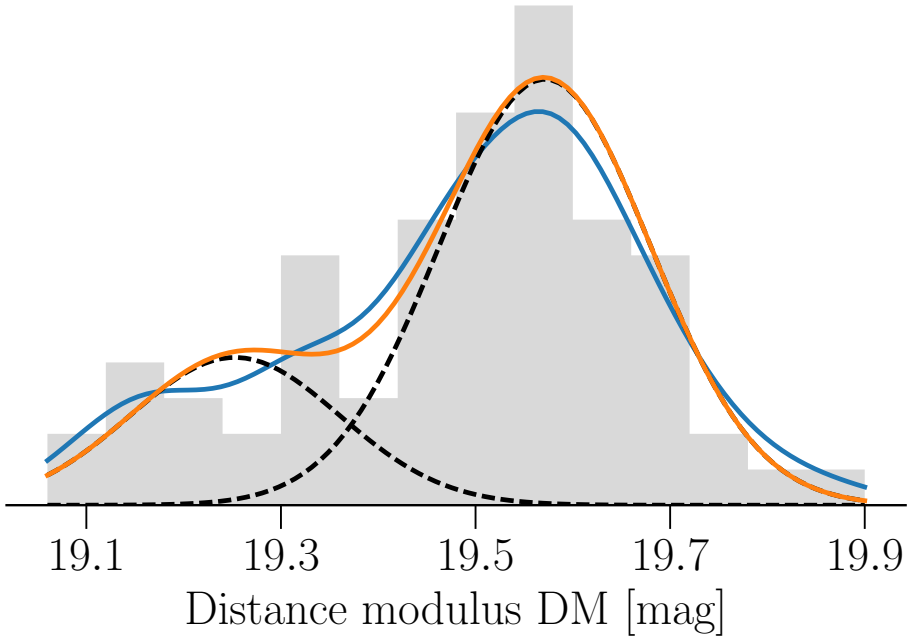
center: a neighboorhood graph is constructed, allowing the geodesic distance between points to be computed by finding the shortest path through the graph

right: a lower dimensional embedding of the manifold is recovered by IsoMap that preserves the relative geodesic distances between points

Density Estimation

Inferring the pdf of a sample of data is known as **density estimation**. Essentially we are smoothing the data to correct for the finiteness of our sample and to better recover the underlying distribution.

seen before: Gaussian Mixture models, which are a case of **Parametric Density Estimation**



Density Estimation

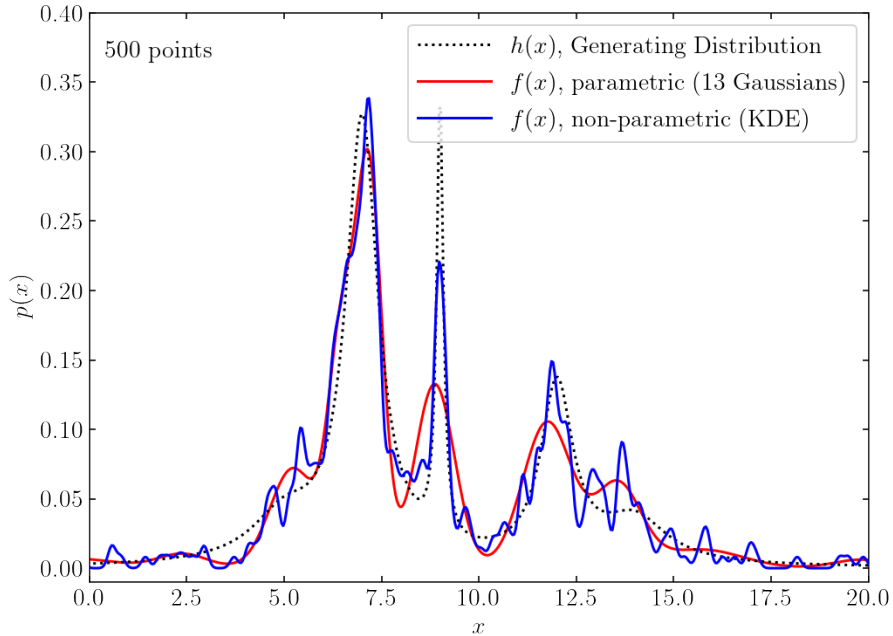
Here we cover in detail two other unsupervised learning processes for density estimation:

- Non-parametric Density Estimation, specifically Kernel Density Estimation (KDE)
- Nearest Neighbor Density Estimation

Non-parametric Density Estimation

Nonparametric density estimation is useful when we know nothing about the underlying distribution of the data, since we don't have to specify a functional form.

Kernel Density Estimation (KDE) is the standard approach for non-parametric density estimation.
We've already seen this briefly in Lecture 3:



Kernel Density Estimation

Kernel density estimators belong to a class of estimators called **non-parametric density estimators**. In comparison to **parametric estimators** where the estimator has a fixed functional form (structure) and the parameters of this function are the only information we need to store, Non-parametric estimators have no fixed structure and depend upon all the data points to reach an estimate.

Kernel density estimation estimates an unknown probability density function using a **kernel function** $K(u)$.

Kernel Density Estimation

Let $(x_1, x_2, ..., x_n)$ be independent and identically distributed samples drawn from some univariate distribution with an unknown density f. The **kernel density estimator** for f is

$$\widehat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\Big(\frac{x - x_i}{h}\Big)$$

where K is the **kernel function** and $h > 0$ is a smoothing parameter called the **bandwidth**.
More generally, to measure the distance using a metric different than the Euclidean one, replacing $(x - x_i)$ with $d(x, x_i)$, and generalizing to D dimensions:

$$\widehat{f}_h(x) = \frac{1}{nh^D}\sum_{i=1}^{n} K\left(\frac{d(x, x_i)}{h}\right)$$

Kernel Density Estimation

The **kernel function** typically exhibits the following properties:

- Symmetry such that $K(u) = K(-u)$.
- Normalization such that $\int_{-\infty}^{\infty} K(u)du = 1$
- Monotonically decreasing such that $K'(u) < 0$ when $u > 0$
- expectation value equals zero such that $E[K] = 0$.

Kernel Density Estimation

The **kernel function** typically exhibits the following properties:

- Symmetry such that $K(u) = K(-u)$.
- Normalization such that $\int_{-\infty}^{\infty} K(u)du = 1$
- Monotonically decreasing such that $K'(u) < 0$ when $u > 0$
- expectation value equals zero such that $E[K] = 0$.

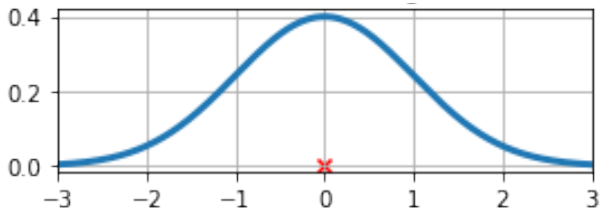
Choice of **bandwidth** h:

Intuitively one wants to choose h as small as the data will allow; however, there is always a trade-off between the bias of the estimator and its variance. The choice of bandwidth is discussed in more detail below.

Kernel Density Estimation

A common kernel is the **Gaussian kernel**:

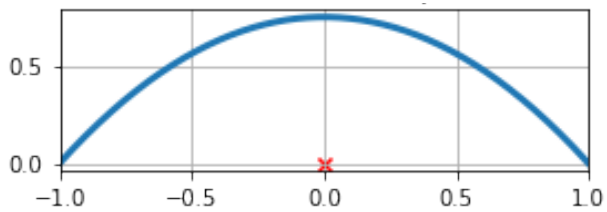
$$K(u) = \frac{1}{(2\pi)^{D/2}} \exp\left(-u^2/2\right)$$



where D denotes the dimensionality of the data.

The **Epanechnikov kernel** is 'optimal' because it minimizes the variance of the kernel density estimate:

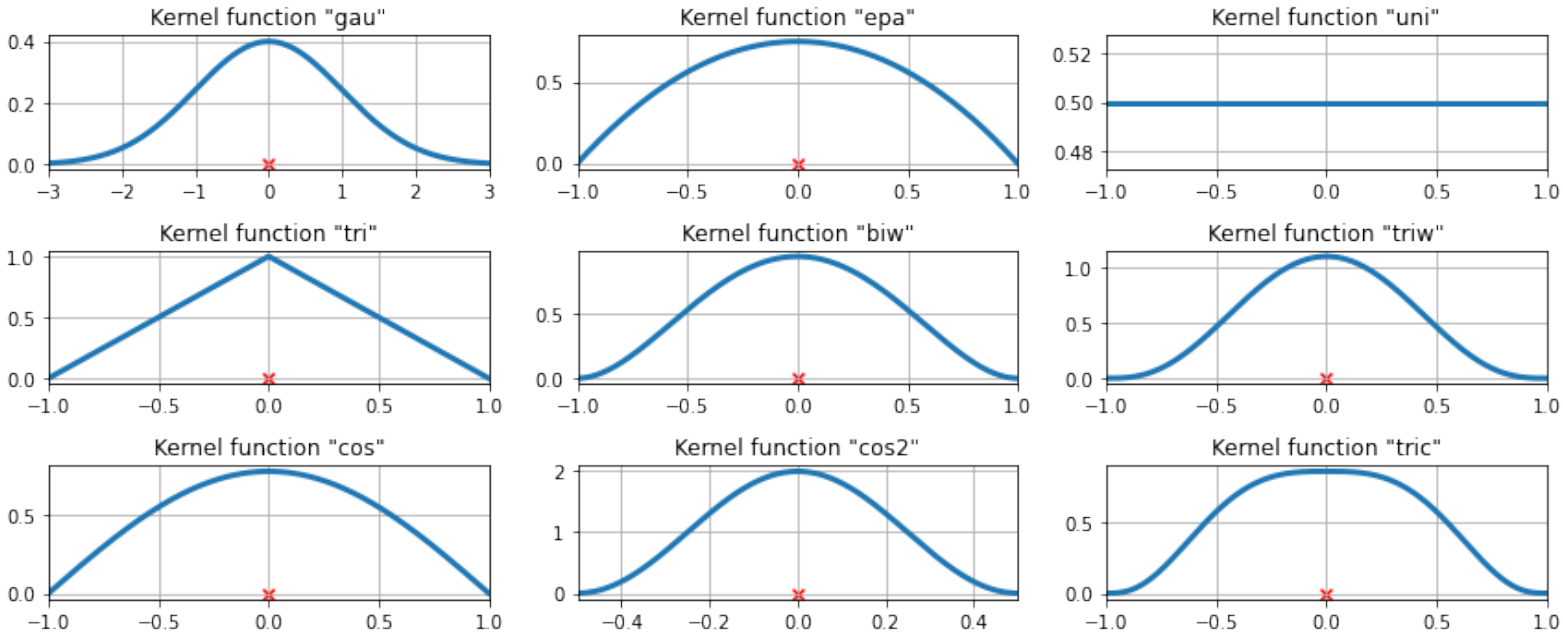
$$K(x) = \frac{3}{4}(1 - x^2),$$



for $|x| \leq 1$ and 0 otherwise.

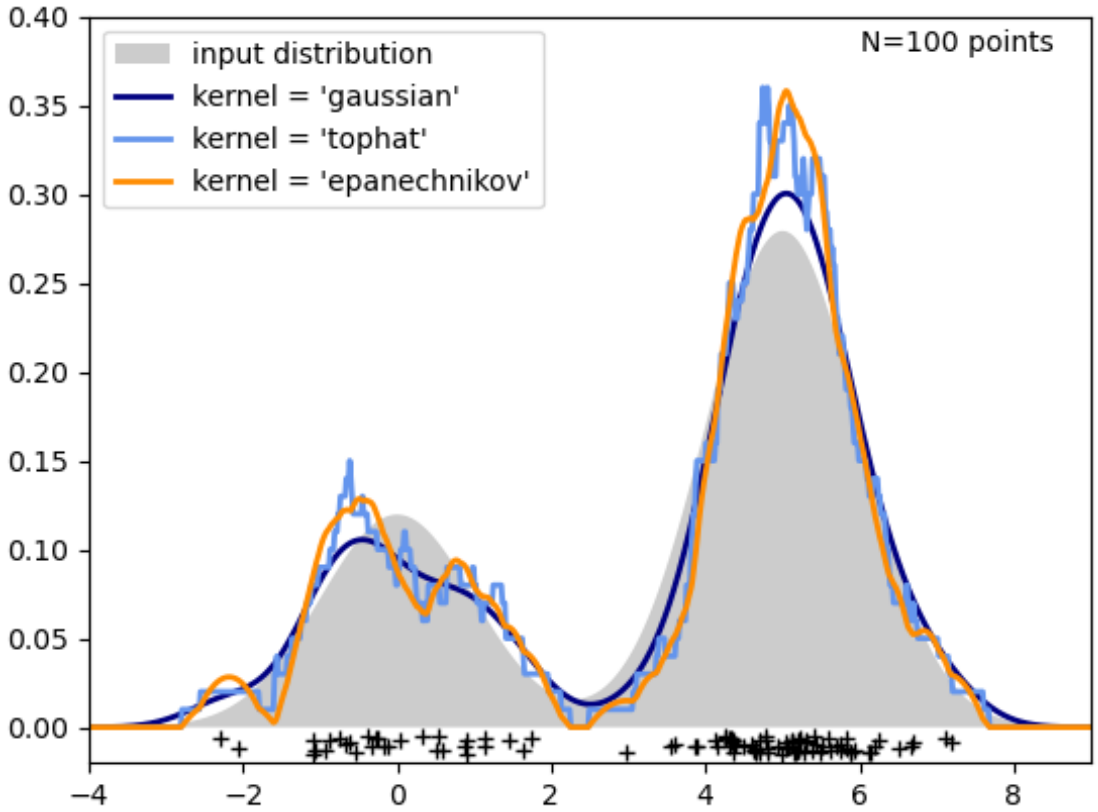
Kernel Density Estimation

several kernels are available from Python `statsmodels.nonparametric.kde`:



Kernel Density Estimation

In the following figure, 100 points are drawn from a bimodal distribution, and the kernel density estimates are shown for three choices of kernels:



Nearest-Neighbor Density Estimation

Another way to estimate the density of an N-dimensional distribution is to look to the K nearest objects and compute their distances, d_K. This is the K-**Nearest Neighbor algorithm**. The density at a given point, x is estimated as

$$\widehat{f}_K(x) = \frac{K}{V_D(d_K)},$$

where $V_D(d)$ is given generically by $\frac{2d^D\pi^{D/2}}{D\Gamma(D/2)}$ with Γ being the gamma function.

This estimator has some intrinsic bias, which can be reduced by considering all K nearest neighbors:

$$\hat{f}_K(x) = \frac{C}{\sum_{i=1}^{K} d_i^D}$$

Nearest-Neighbor Density Estimation

The KNN's steps are:

1. Receive an unclassified data.
2. Measure the distance from the new data to all others data that is already classified.
3. Get the K smaller distances.
4. Check the list of classes had the shortest distance and count the amount of each class that appears.
5. Take as correct class the class that appeared the most times.
6. Classify the new data with the class from step 5.

Break & Questions

afterwards we continue with **`lecture_11.ipynb`** from the `github` repository