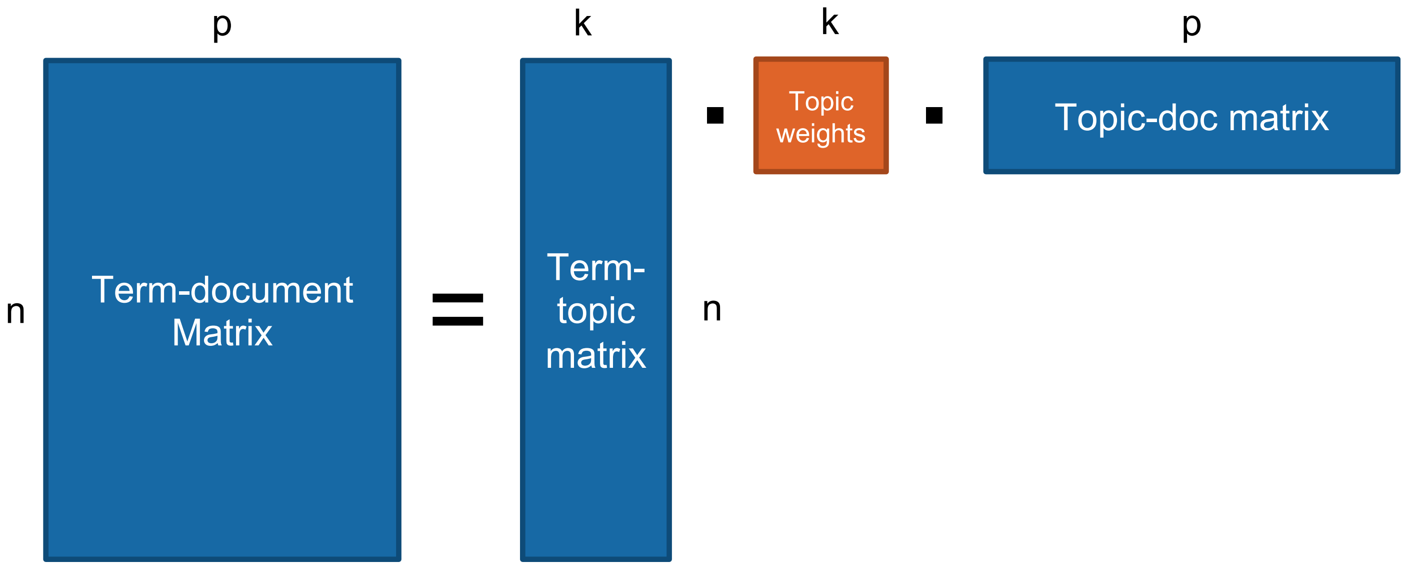
Example: Using the SVD to create an optimal* approximation of the original matrix X

- What if we allow ourselves *two* pairs of vectors?
- $X \approx U_1\Sigma_{11}V_1^T + U_2\Sigma_{22}V_2^T$ is the best rank 2 approximation*
- Let's try it on a 7x3 matrix…

* In the least-squares sense.

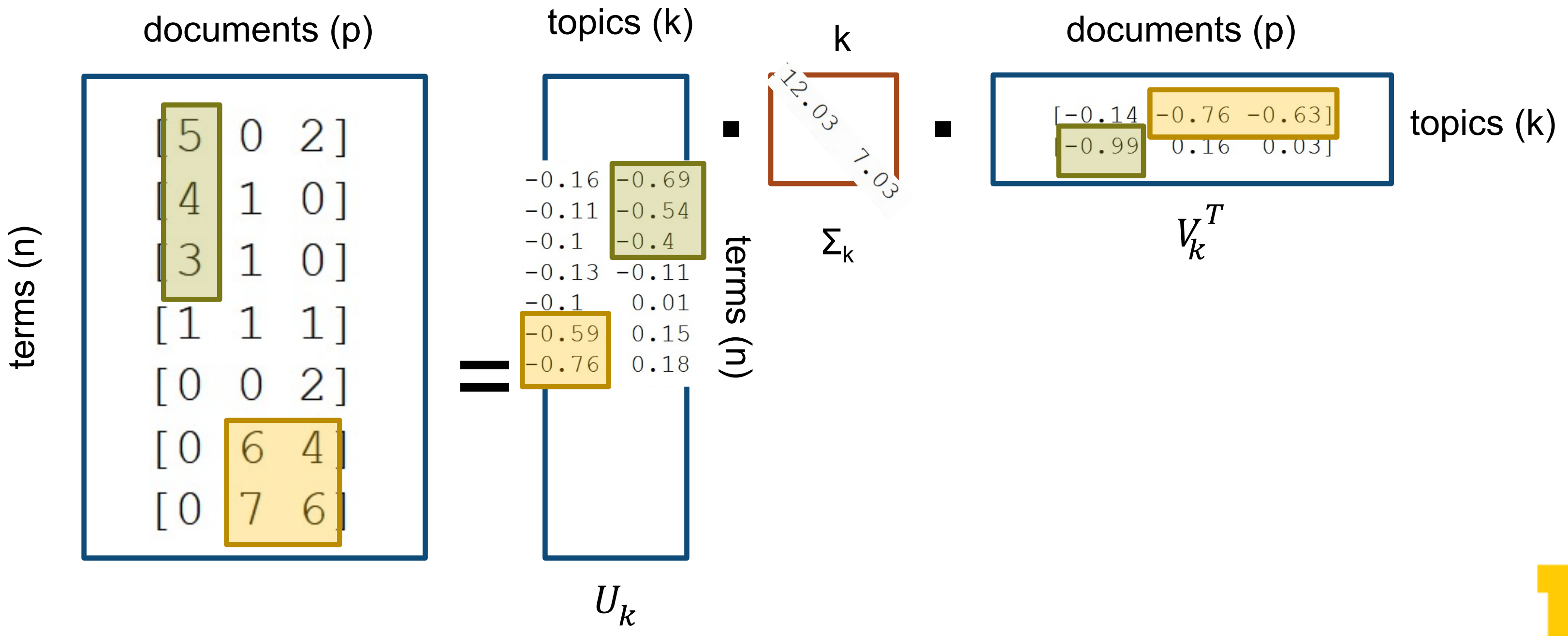
Preview: SVD for text data = Latent Semantic Indexing

Interpretation of SVD of term-document matrix for *k* topics



Preview: SVD for text data = Latent Semantic Indexing

Interpretation of Truncated SVD of term-document matrix for *k* topics



In general, the optimal* rank-k approximation of M is given by: (*with $k < min(n,p)$*)

$$M \approx \Sigma_{11} U_1 V_1^T + \Sigma_{22} U_2 V_2^T + \cdots + \Sigma_{kk} U_k V_k^T$$

- This is the Truncated SVD
- Why is matrix approximation useful?
- This is dimensionality reduction!
- Exactly what our goal was with PCA…

* In the least-squares sense.

Having an SVD-factored version of X makes other calculations much simpler!

- The SVD gives us $X = U\,\Sigma\,V^T$
- Orthonormal *U*, *V* imply these fun facts:

$$X^T X = (U\,\Sigma\,V^T)^T (U\,\Sigma\,V^T) = V\Sigma^T U^T U \Sigma V^T = V\Sigma^2 V^T$$

$$X X^T = (U\,\Sigma\,V^T)(U\,\Sigma\,V^T)^T = U\Sigma V^T V \Sigma U = U\Sigma^2 U$$

$$(X^T X)^{-1} = V\Sigma^{-2} V^T \qquad X(X^T X)^{-1} X^T = U U^T$$

$$\hat{y} = X \underbrace{\boxed{(X^T X)^{-1} X^T y}}_{\hat{\beta}} = U U^T y$$

for ordinary least-squares regression on (*X*,*y*)

Is $X^T X$ invertible? The SVD tells you.

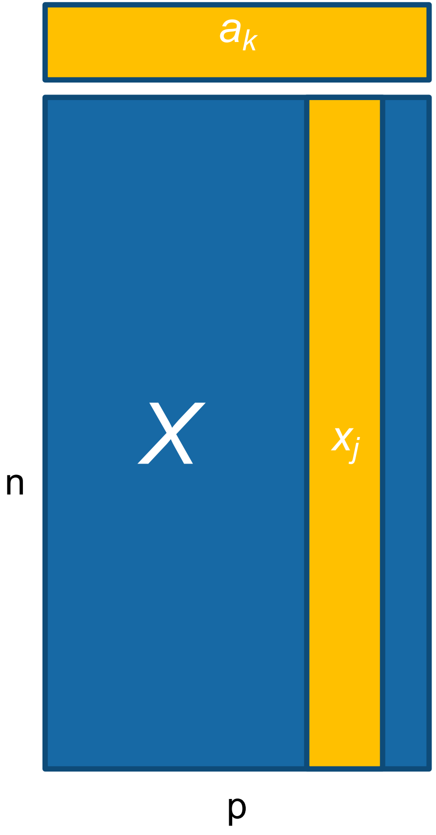
- To know if the matrix $X^T X$ is invertible, just check smallest singular value:
- Non-zero $\Rightarrow$ invertible
- X itself has no inverse if $n > p$.

- Rank of X? Just the number of non-zero singular values.

Principal components analysis (PCA)

- PCA(X) is just the SVD on centered X.
- Columns of V are **PC loading** vectors.
- Columns of U (up to scaling): **PC score** vectors.
- Recall that we were seeking loadings that maximized variance.

PCA is the SVD of the centered data matrix X*



PCA usually uses a centered version X* of matrix X as input.

$$x_{ij}^{*} = x_{ij} - x\_mean_j$$

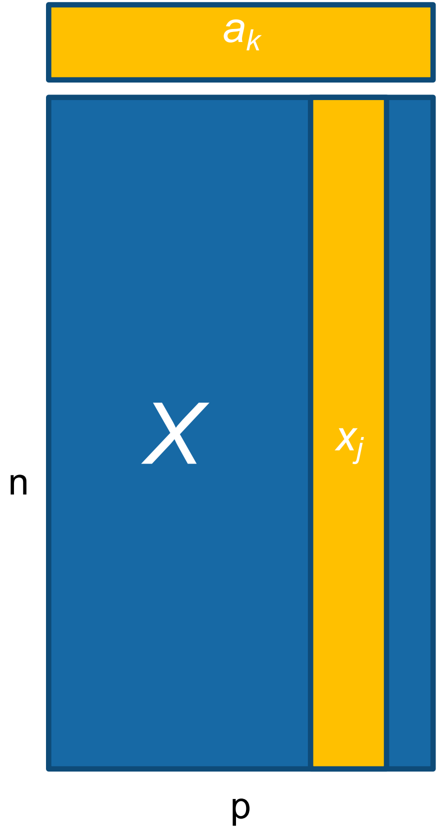
After centering, the covariance matrix *S* can be expressed in terms of X*

$$(n-1)S = {X^{*}}^{T} X^{*}$$

The Singular Value Decomposition of a matrix Y is obtained from the eigendecomposition of Y^TY.
Hence PCA is equivalent to the SVD of the centered data matrix X*

(More on this and other details of SVD in a separate lecture)

PCA is also done on correlation or cross-product matrices



There are versions of PCA that take SVD of correlation matrix $\hat{X}^T\hat{X}$

On occasion the raw crossproduct matrix X^TX is used (e.g. sparse data).

Keeping the first *k* components of PCA is equivalent to doing Truncated SVD.

Applications: using the SVD to impute missing values

- If elements of X are randomly missing the SVD gives you one effective and easy way to impute the missing values.
- Beware: Imputing missing values is a complex topic and actually an entire subarea within statistics.
- The best imputation method will depend on multiple factors:
    - Fraction of noise, covariance structure, etc.

Imputing missing values using SVD (simplified version!)

1. Fill in missing elements of X using (say) the column mean. Result: new matrix X^*
2. Compute SVD of X^* to get rank-k approximation X_k (say, k=4)
3. Replace the missing elements in the original X with the corresponding elements of the rank-k approximation X_k
4. Repeat until values converge.

Related work: https://web.stanford.edu/~hastie/Papers/mazumder10a.pdf

Other SVD applications in data analytics

- Say X represents a set of p-dimensional points to which you want to fit a linear model with least squares.
- But first, you want to quickly test if X is multicollinear, i.e. high correlations between feature(s)
  - Easy with SVD: look at the singular values. If $\Sigma_{1,1} \;/\; \Sigma_{p,p}$ is huge, LS is a bad idea.
  - You'll to apply a more sophisticated regression approach: Partial Least Squares regression, or PCA regression.

SVD routines in Python

- scikit-learn only has Truncated SVD
- “pure” SVD in scipy.linalg
- Dense data:
  - Column-centered SVD or Truncated SVD
- Sparse data:
  - Use Truncated SVD (does not center the data – efficient for sparse matrices)
- NOTE! SVD suffers from ‘sign indeterminacy’. The signs of the components_ depend on algorithm and random state. Fit instances of TruncatedSVD once, then keep it around to do transformations.

UNIVERSITY OF MICHIGAN

Kevyn Collins-Thompson

kevynct@umich.edu

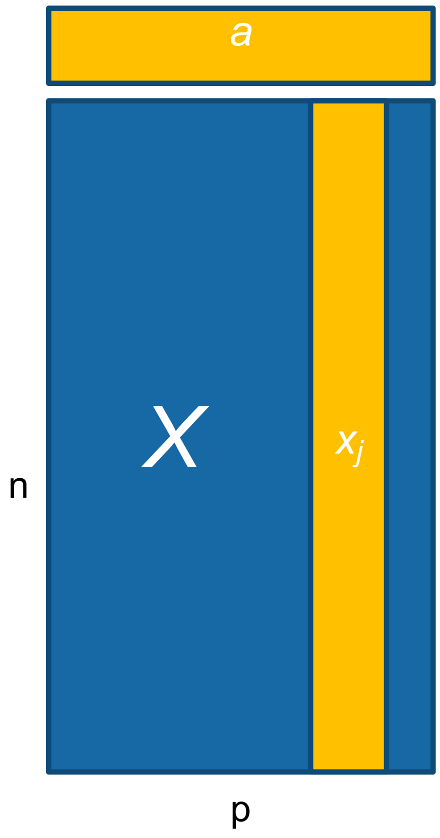
School of Information

PCA: Biplots and how to use them

Kevyn Collins-Thompson

Associate Professor of Information and Computer Science
School of Information, University of Michigan

Review: Linear algebra basis for PCA



We want to compute a new, smaller set of features that preserves information about variance in our original dataset.

Each new feature (column) should be obtained by taking a linear combination (vector) **a** of all the existing columns.

$$Xa = \sum_{j=1}^{p} a_j x_j$$

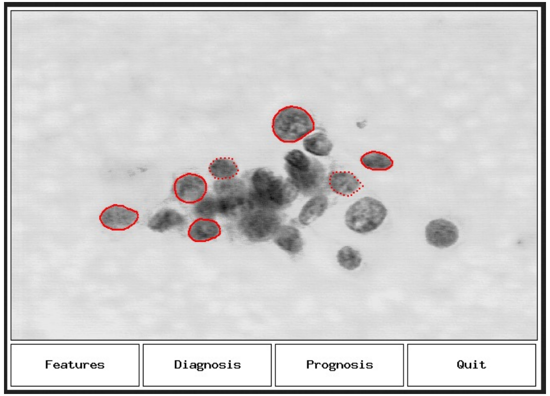
But what is the initial "best" linear combination vector **a**?

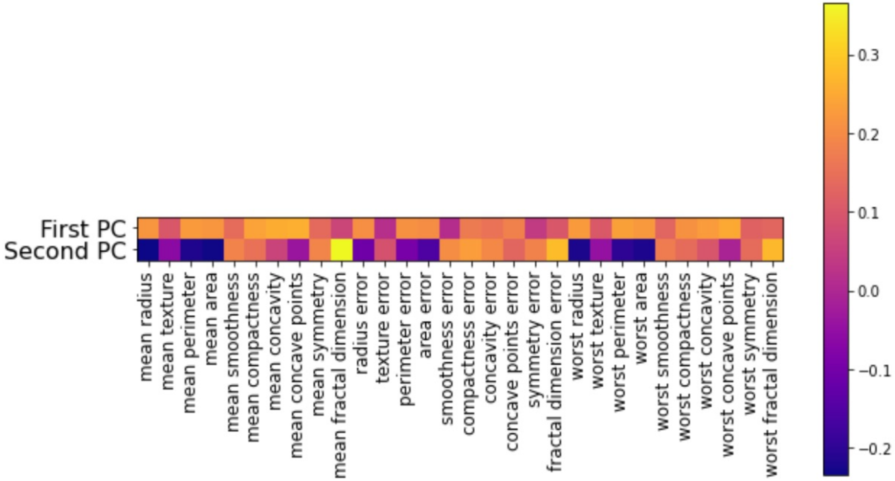
The ***vector a*** should ***maximize*** $var(Xa) = a^T S a$ *subject to* $a^T a = 1$.
Call this first solution $\mathbf{a_1}$. *Iterate to produce next solution* $\mathbf{a_2}$ *etc.*

The *p* values in each $\mathbf{a_i}$ are called loadings.
The result of multiplying $X\mathbf{a_i}$ are called the scores: X in the new feature space.

```
pca = PCA(n_components = 2).fit(X_cancer_normalized)
X_pca = pca.transform(X_cancer_normalized)
pca.components_    # columns contain a1 , a2 , ...
```

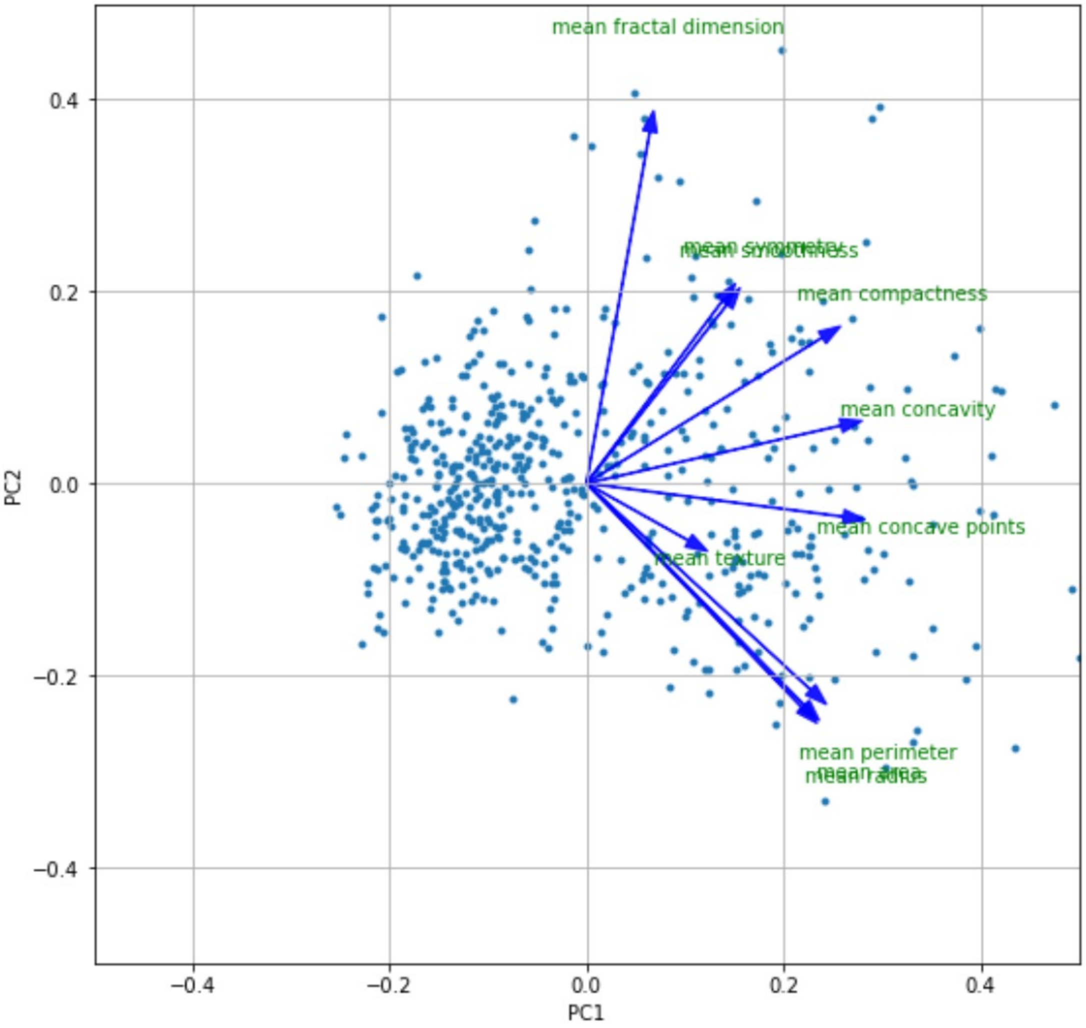
PCA on the Wisconsin Breast Cancer dataset






```
For a 10-component PCA:

pca.explained_variance_ratio_     = [0.44 0.19 0.09 0.07 0.05 0.04 0.02 0.02 0.01 0.01]
np.sum(explained_variance_ratio_) = 0.95
```

Image source: http://pages.cs.wisc.edu/~street/saves/xcyt1.gif



A <u>biplot</u> shows markers for

- Individuals (points)

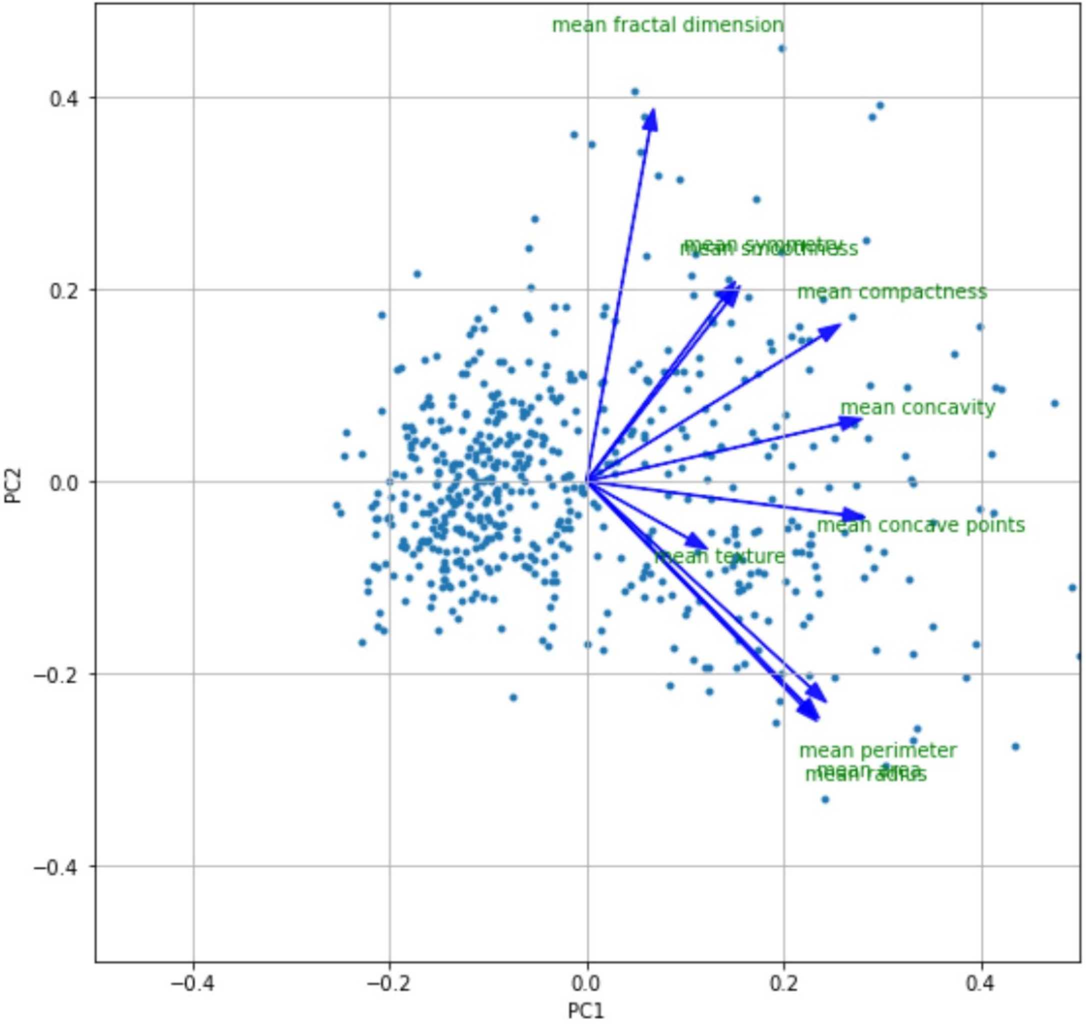
Scatterplot is generated from the **scores:**

```
X_pca =
    pca.transform(X_original)
```

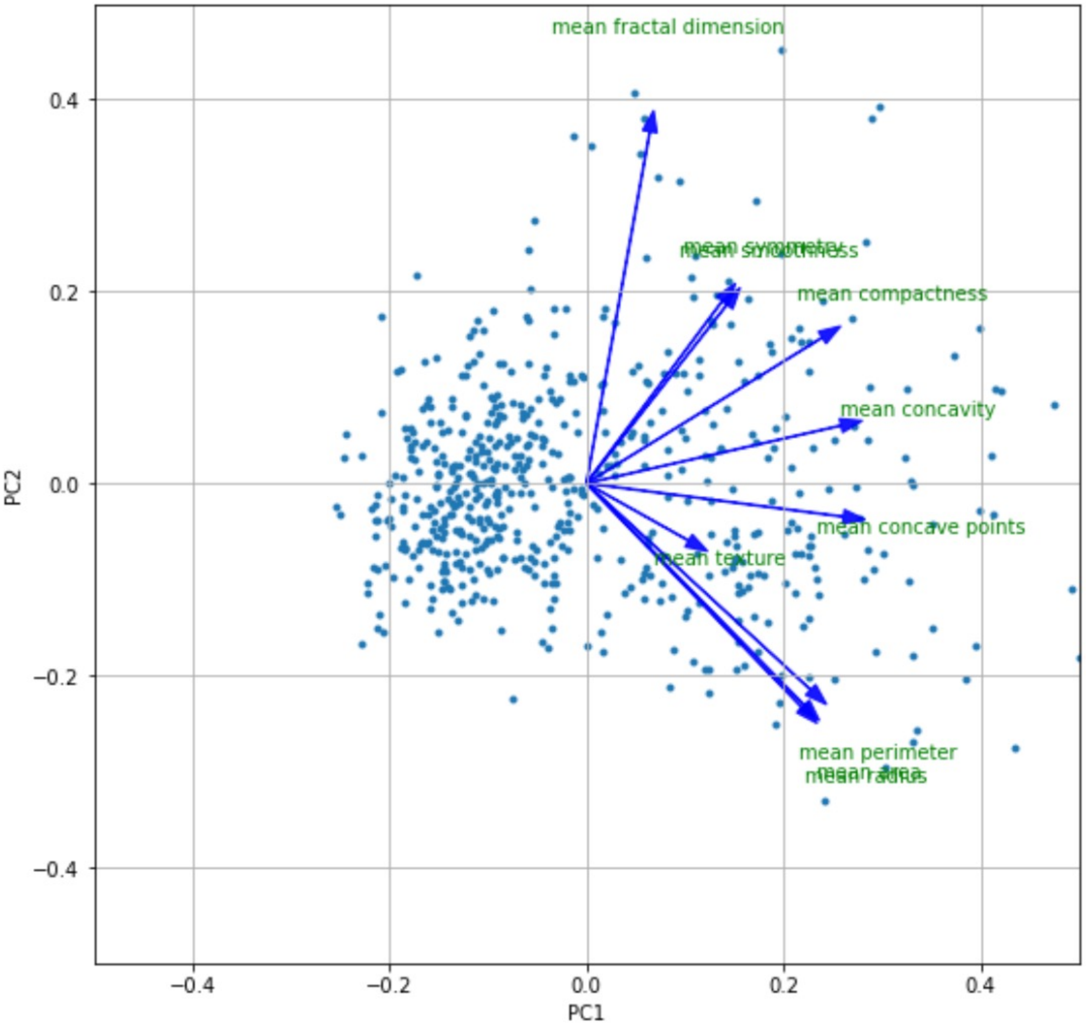
- Variables (vectors)

Plotted vectors v_i are from the **loadings** (PC 1, PC 2):

```
X= pca.components_[i,0]
Y =pca.components_[i,1]
```

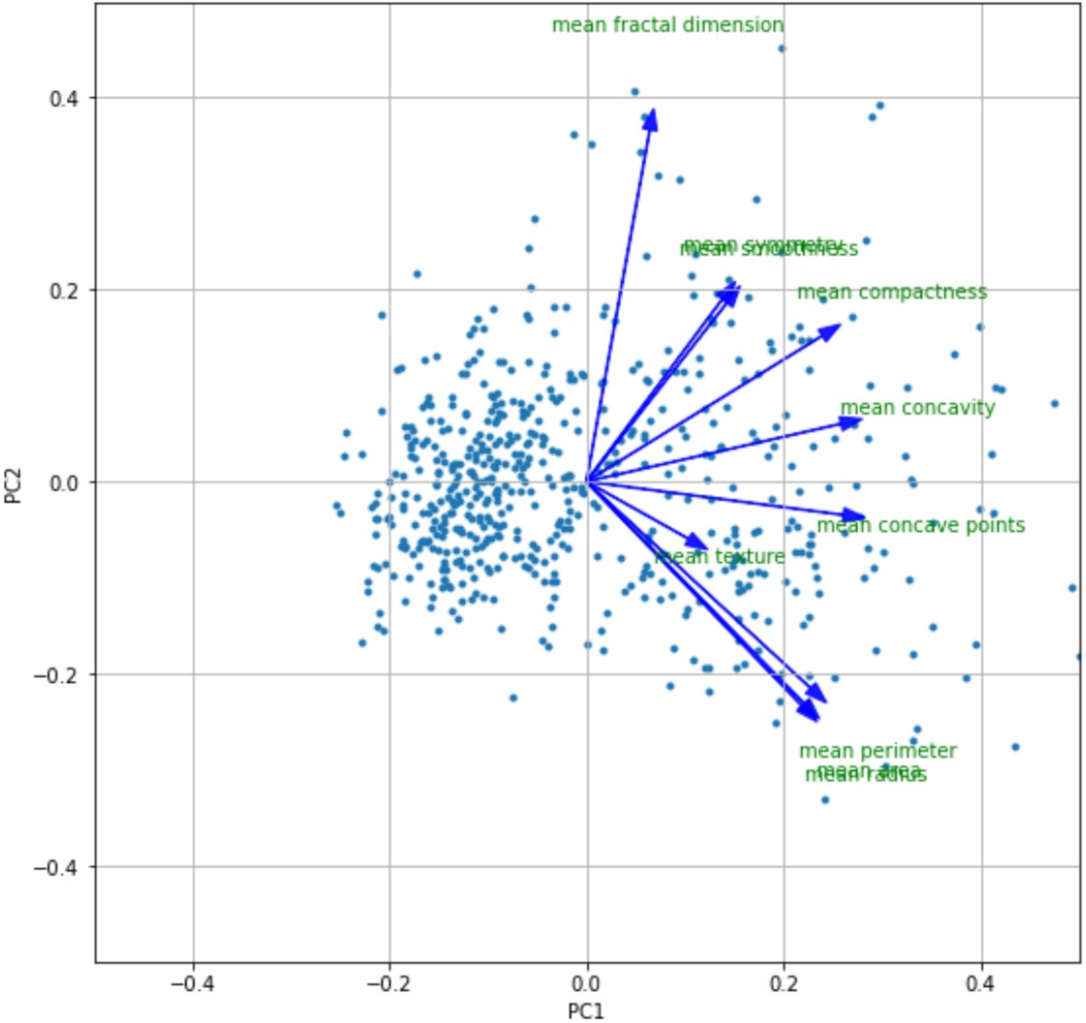


1. The <u>cosine of the angle</u> between any two variable markers (vectors) is the coefficient of correlation between those variables.

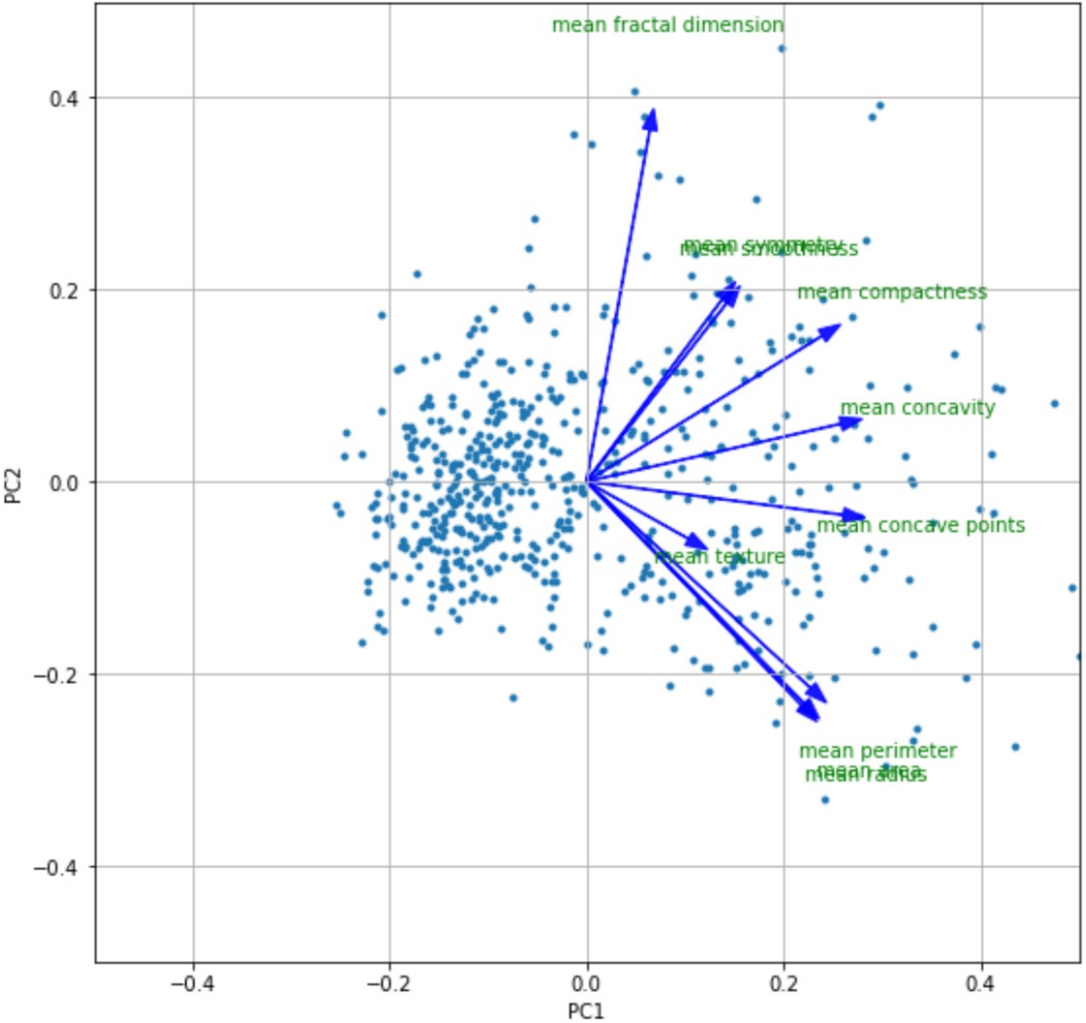


1. The <u>cosine of the angle</u> between any two variable markers (vectors) is the coefficient of correlation between those variables.

2. The cosine of the angle between a variable marker (vector) and the axis for a given PC is the coefficient of correlation between those two variables.



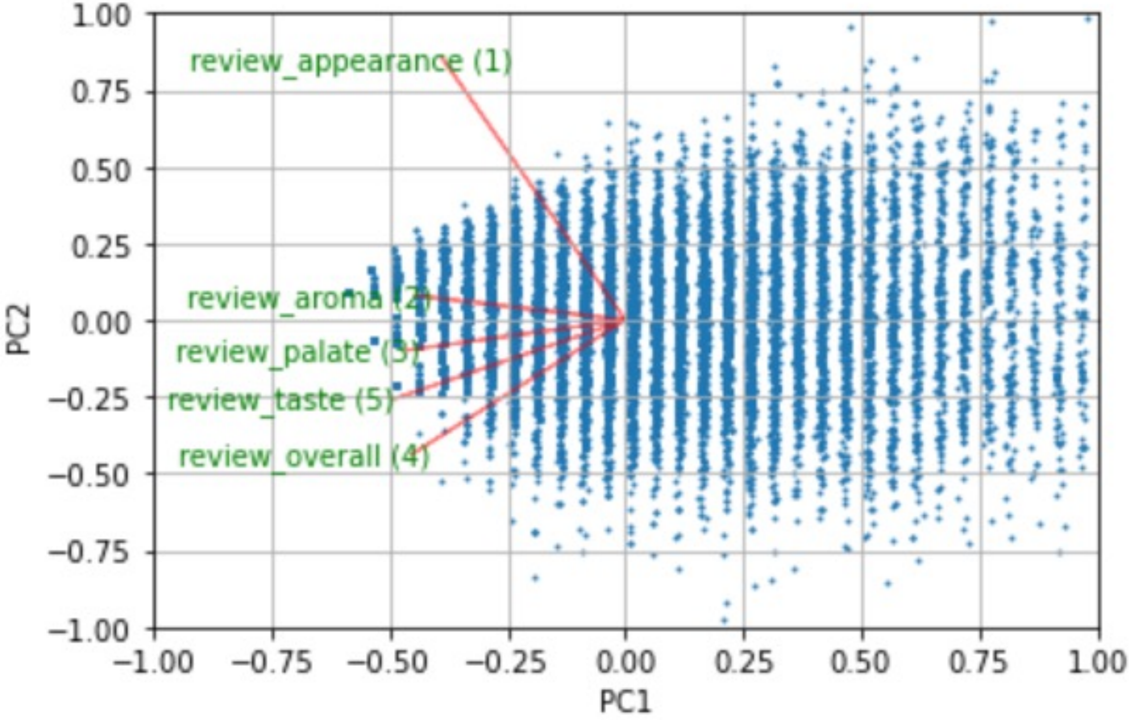
3. The dot product between the markers for data point *i* and variable *j* gives the reconstructed (centered) value of data point *i* for variable *j*.



4. The Euclidean distance in the biplot between the markers for individuals *i* and *j* is proportional to their variance-adjusted (Mahalanobis) distance in the original feature space.

Mahalanobis distance measures in 'units of standard deviation'.

Reduces to Euclidean distance for uncorrelated variables with unit variance.



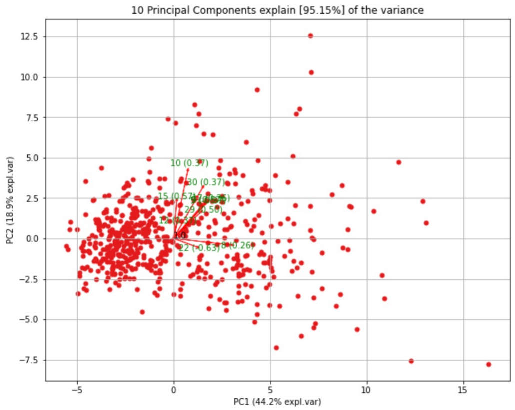
review_appearance (1)
review_aroma (2)
review_palate (3)
review_taste (5)
review_overall (4)
PC1
PC2
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

Creating biplots

- No direct method in scikit-learn. Feel free to use the provided notebook code.

- Alternative: ‘pca’ python package
  - Includes a biplot plotting routine
  - Installation: `pip install pca`

```
from pca import pca

model   = pca(n_components = 10)
results = model.fit_transform(X_cancer_normalized)
fig, ax = model.biplot(n_feat=10, legend = False
plt.figure()
```



UNIVERSITY OF MICHIGAN

Kevyn Collins-Thompson

kevynct@umich.edu

School of Information

UNIVERSITY OF MICHIGAN

Other useful resources:

https://blogs.sas.com/content/iml/2019/11/06/what-are-biplots.html

School of Information

Sparse, Rotated, Robust and Kernel PCA

Kevyn Collins-Thompson

Associate Professor of Information and Computer Science
School of Information, University of Michigan

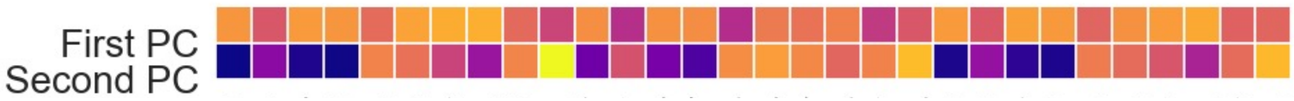
Some useful variants of PCA

- Sparse PCA: most influential PC variables
- Rotated PCA: easier interpretation
- Robust PCA: handling noisy input
- Kernel PCA: non-linear low-dim structure

- *Not covered: PCA with missing values*
  - *PCA with missing values: weighted LS with w_ij = 0 if X_ij missing, 1 otherwise.*

Sparse PCA

- Problem:
  - It can be hard to interpret a jumble of non-zero variable loadings

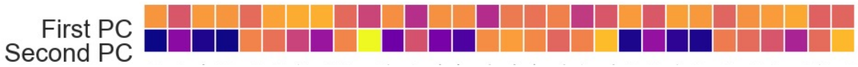


- Solution:
  - Add constraints to PCA so that the PC must be sparse (mostly zero, most important variables non-zero)
  - Adding L1 penalty is one way to do this.

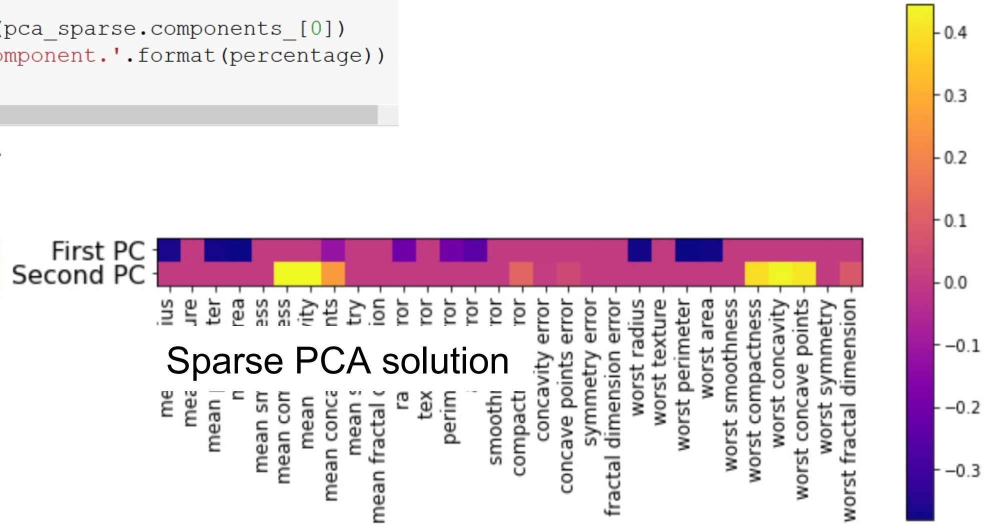
SparsePCA class in scikit-learn

```
from sklearn.decomposition import SparsePCA
pca_sparse = SparsePCA(n_components=3, alpha = 15, random_state=0)
pca_sparse.fit(X_cancer_normalized)
X_transformed = pca_sparse.transform(X_cancer_normalized)

# most values in the components_ are zero (sparsity)
percentage = np.sum(pca_sparse.components_[0] == 0)/len(pca_sparse.components_[0])
print('{:2.2%} of values are zero in Sparse PCA first component.'.format(percentage))
plot_cancer_pca(pca_sparse)
```

```
66.67% of values are zero in Sparse PCA first component.
```



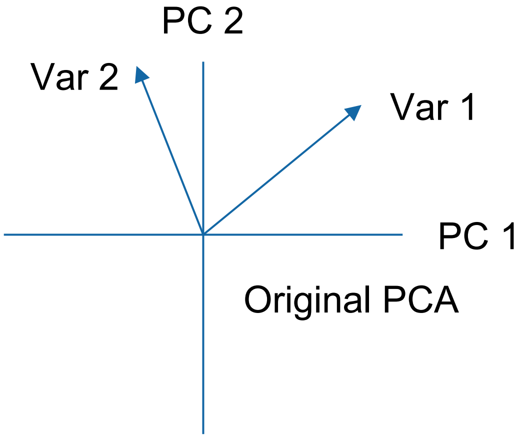
Ordinary PCA: non-sparse solution



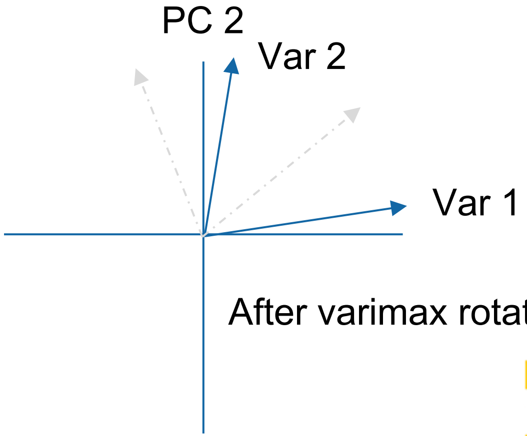
Sparse PCA solution

Rotated PCA

- Tries to find a rotation of all the PCs so that each variable is associated with at most one rotated PC.
- This rotated version is easier to interpret because each rotated PC has one clear associated variable.
- PCA + varimax is the most popular optimization criterion.
- Varimax = entries either large or near zero
- Is PCA + varimax still PCA? The rotated PC are not principal components any more, so not so much. (Complex question..)
- How is this different than Factor Analysis?



Original PCA



After varimax rotation

PCA + varimax sounds similar to..Factor Analysis. Is it?

- Factor Analysis (FA) also seeks a lower-dimensional ‘explanation’ of structure in a higher-dimensional dataset
- BUT… FA and PCA have very different optimization objectives:
  - PCA finds linear combinations of the input variables that are orthogonal.
  - FA is more flexible than PCA: it explores a broader set of possible latent variables and structures (‘constructs’). Factors don’t have to be orthogonal.
  - PCA could be considered a very special form of FA.
- The bottom line:
  - Run factor analysis if you assume or wish to test a theoretical model of latent factors that “explains” observed variables.
  - Run principal component analysis if you just need to reduce your correlated observed variables to a smaller set of important independent composite variables.

Robust PCA

- Problem:
  - Ordinary PCA tries hard to find PC that capture all variation in your data.
  - But if there are major outliers (sensor failures, malicious injection, occlusions etc.), they can add huge variation.
  - Results from ordinary PCA are sensitive to such huge outliers.
- Idea: try to automatically separate the outliers/noise from the low-rank ‘main’ data: Principal Component Pursuit. (Candès, Li, Ma, Wright, 2009).

Principal Component Pursuit

For original data matrix *M*, find low-rank approximation *L*.

- Ordinary PCA:

$$\begin{array}{ll} \text{minimize} & \|M - L\| \\ \text{subject to} & \operatorname{rank}(L) \le k. \end{array}$$

- Robust PCA via PCP estimate:

$$\begin{array}{ll} \text{minimize} & \|L\|_* + \lambda\|S\|_1 \\ \text{subject to} & L + S = M \end{array}$$

Sparse matrix *S*

$\|M\|_* := \sum_i \sigma_i(M)$

$\|M\|_1 = \sum_{ij} |M_{ij}|$

Robust PCA: Principal Component Pursuit

- If the low-rank non-noise data has PCs that are ‘reasonably’ spread, PCP can recover them with probability nearly one.
- Even with arbitrary and completely unknown corruption patterns (as long as these are randomly distributed).
- In fact, this works for large values of the rank, i.e. on the order of $n/(\log n)^2$ (see paper for detailed conditions)

Robust PCA for image recognition

- Simple objects lit with distant illumination can be represented by a low-rank model (9 dim subspace, Basri & Jacobs)
- Shadows, specular reflections, image acquisition glitches are spatially sparse, large magnitude outliers w.r.t main image.
- Shows how PCA-family preprocessing (unsupervised) can improve the accuracy of face recognition (supervised).
- Unfortunately still no Robust PCA implementation in scikit-learn (as of Aug. 2020)

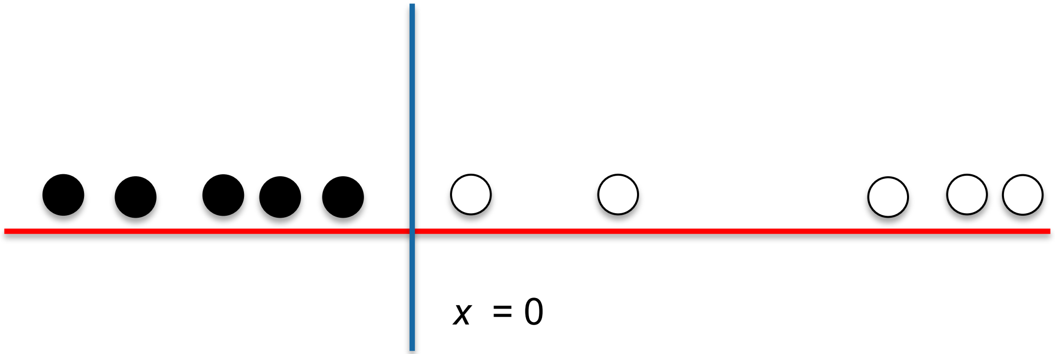
(a) Original frames (b) Low-rank $\hat{L}$ (c) Sparse $\hat{S}$

(above) Background detection
(below) Removing shadows/highlights

Source: https://arxiv.org/pdf/0912.3599.pdf

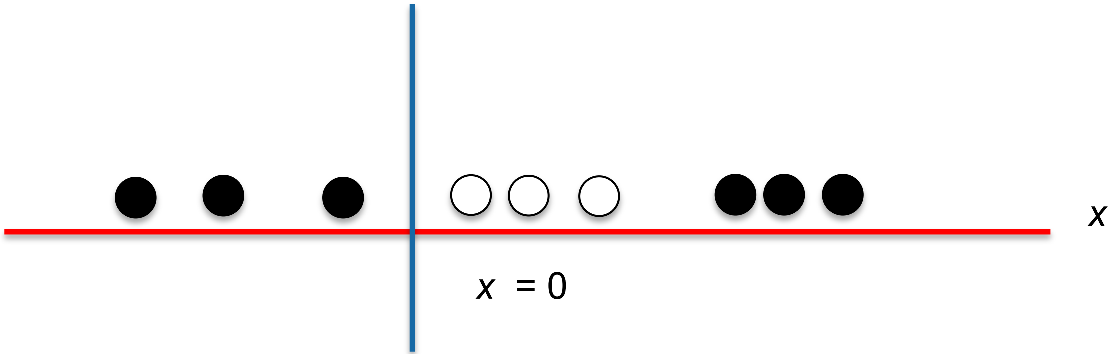
Kernel PCA: the 'kernel trick' resurfaces to help with more complex dimensionality reduction!

Remember this simple 1-dimensional classification problem for a linear classifier?

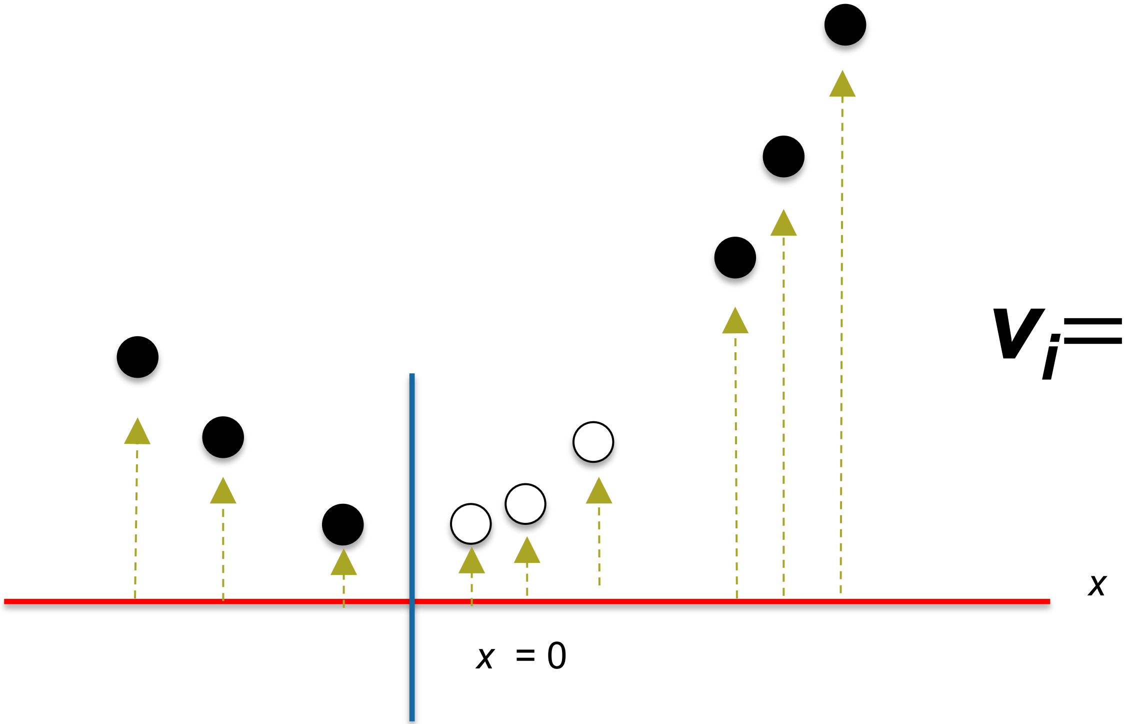


A more perplexing 1-d classification problem for a linear classifier..

A more perplexing 1-d classification problem for a linear classifier

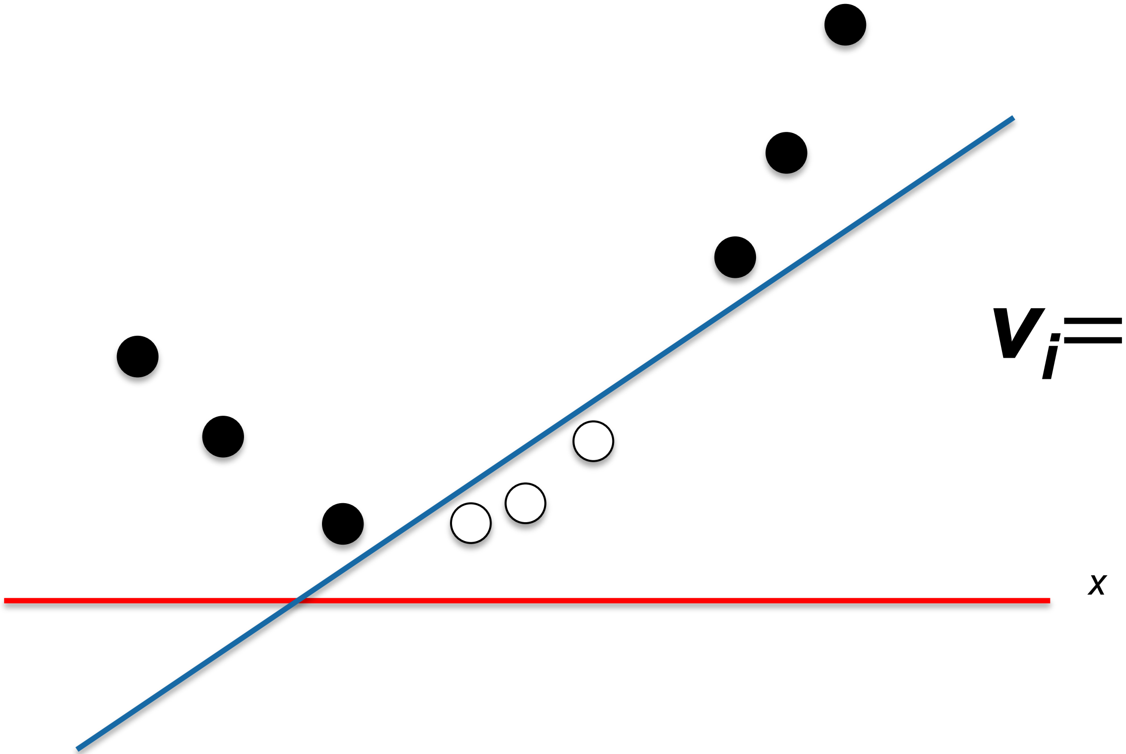


Let's transform the data by adding a second dimension/feature (set to the squared value of the first feature)



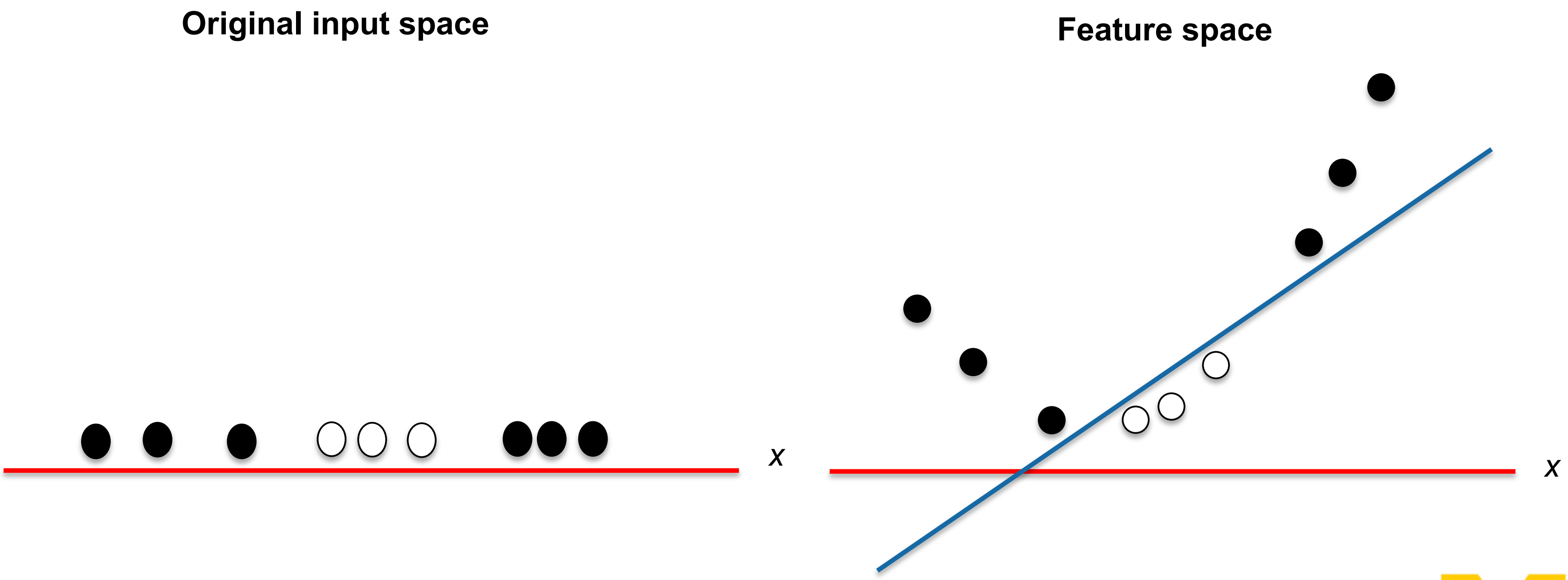
$$\mathbf{v}_i = (x_i, x_i^2)$$

The data transformation makes it possible to solve this with a linear classifier



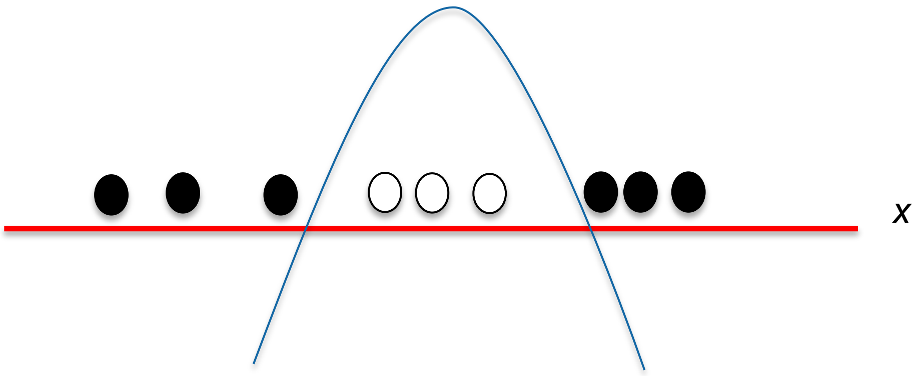
$$\mathbf{v}_i = (x_i, x_i^2)$$

What does the linear decision boundary in feature space correspond to in the original input space?

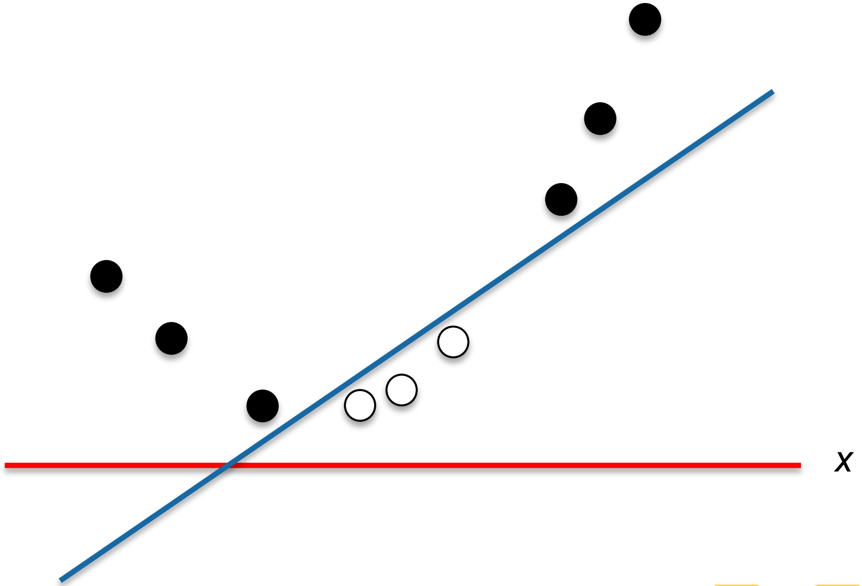


What does the linear decision boundary correspond to in the original input space?

Original input space

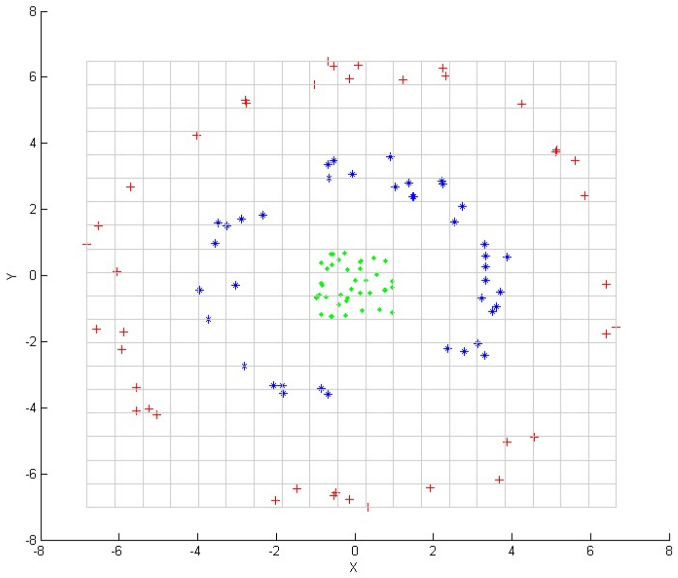


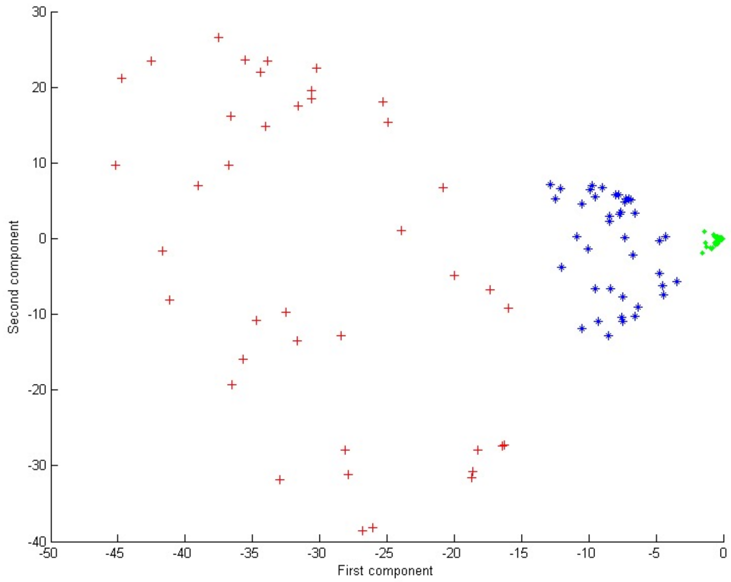
Feature space



Another kernel transform example

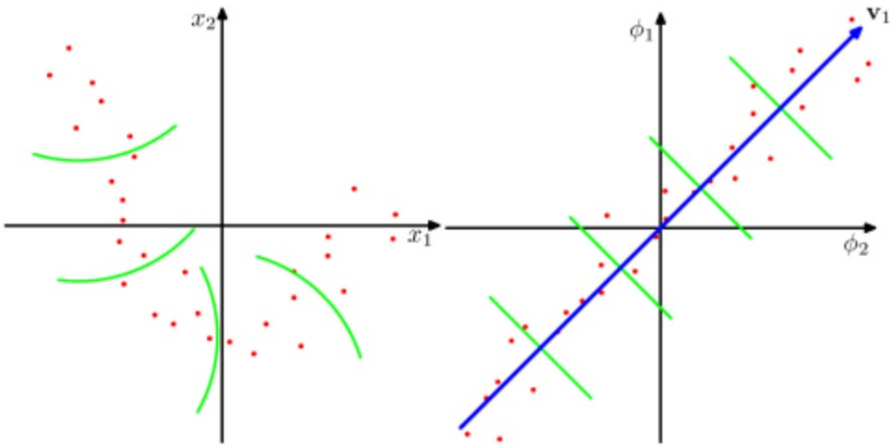
$$k(x, y) = (x^{\mathrm{T}} y + 1)^2$$





Source: Petter Strandmark - Own work, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=3936385

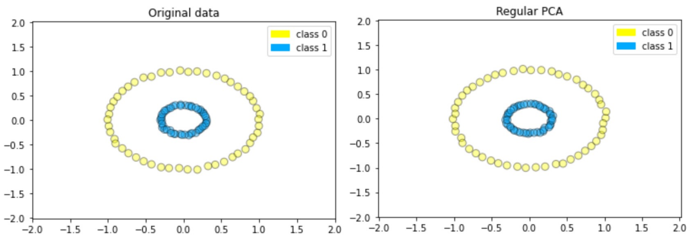
Using the ‘kernel trick’, Kernel PCA can find non-linear low-dimensional structure that is not ‘visible’ to ordinary PCA

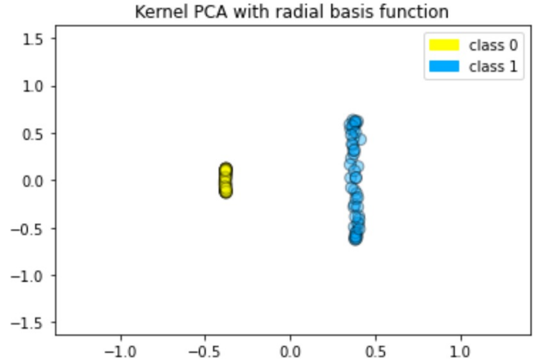


Source: C. Bishop, "Pattern Recognition and Machine Learning" (Figure 12.16)

Kernel PCA in scikit-learn

```
from adspy_shared_utilities import plot_labelled_scatter

from sklearn.datasets import make_circles
from sklearn.decomposition import PCA, KernelPCA

np.random.seed(0)

X, y = make_circles(n_samples = 100, noise = 0.01, factor = 0.3)

# from the adspy_shared_utilities module
plot_labelled_scatter(X, y, ['class 0', 'class 1'], title = "Original data")

kpca = KernelPCA(kernel = "rbf", gamma = 5)
X_kpca = kpca.fit_transform(X)

plot_labelled_scatter(X_kpca, y, ['class 0', 'class 1'],
                      title = "Kernel PCA with radial basis function")

pca = PCA()
X_pca = pca.fit_transform(X)

plot_labelled_scatter(X_pca, y, ['class 0', 'class 1'], title = "Regular PCA")
```





Kernel PCA: when to apply?

- As a pre-processing step, do task-based evaluation.
- Perform model tuning on a validation set to compare results with/without kernel, varying kernel parameters.
- Final accuracy evaluation on test set.
- Close connections to Spectral Clustering.

UNIVERSITY OF MICHIGAN

Kevyn Collins-Thompson

kevynct@umich.edu

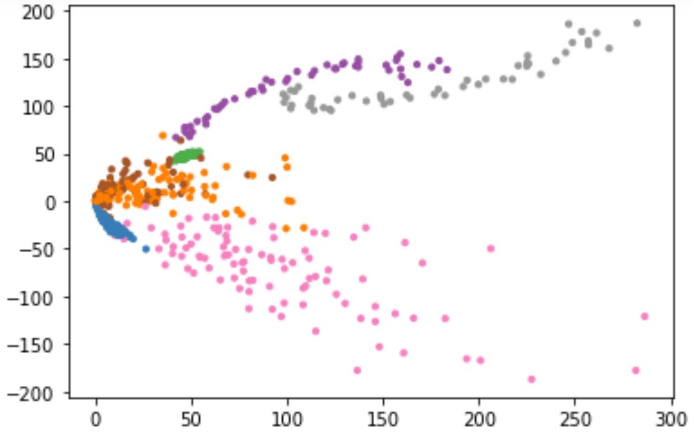
School of Information

Manifold Learning

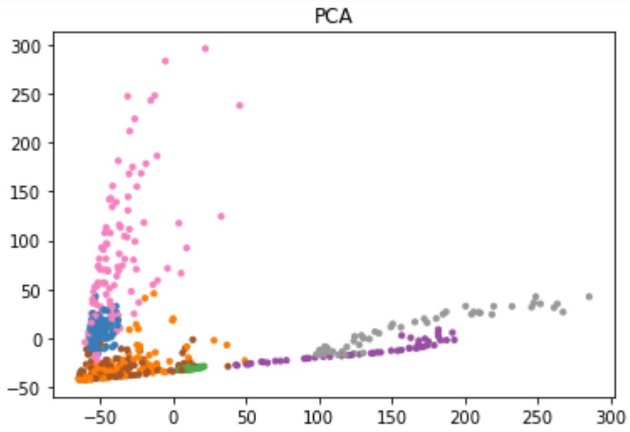
Kevyn Collins-Thompson

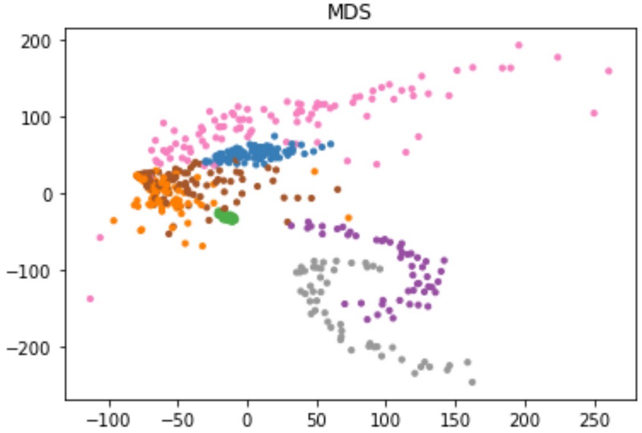
Associate Professor of Information and Computer Science
School of Information, University of Michigan

PCA can fail to find more complex manifolds: manifold learning methods like MDS are sometimes more effective.

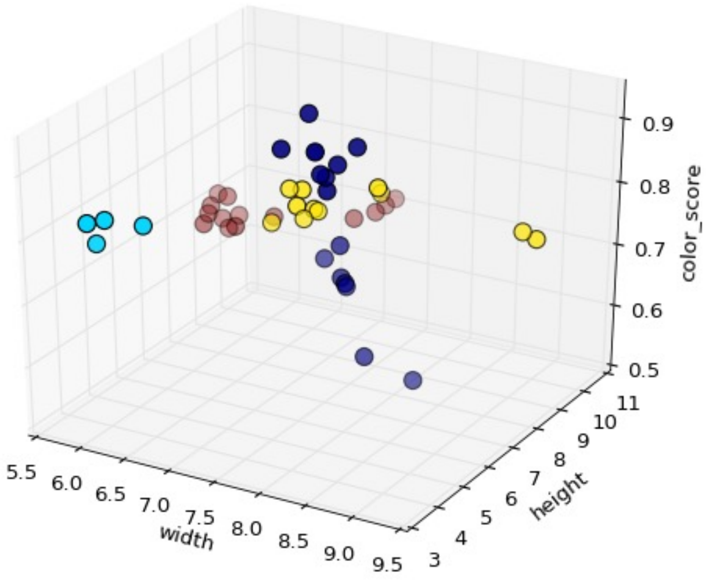


Original dataset

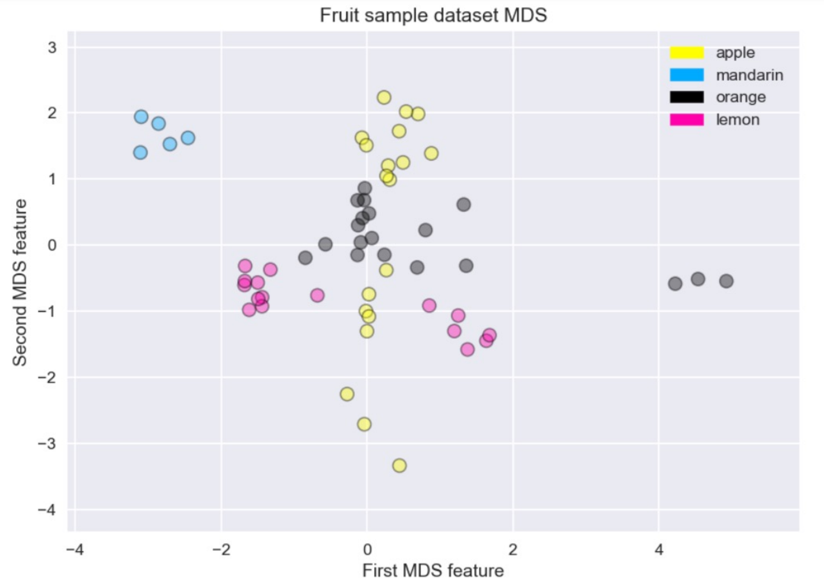




Multidimensional scaling (MDS) attempts to find a distance-preserving low-dimensional projection



High-dimensional dataset



Two-dimensional MDS projection

This lower-dimensional version is sometime referred to as an embedding.

Review: distance vs dissimilarity measures

- Distance measures d(x,y) satisfy:
  - $d(x, y) \geq 0$ and $d(x,y) = 0$ if and only if $x=y$
  - $d(x, y) = d(y, x)$
  - $d(a, b) \leq$
    $d(a, c) + d(c, b)$
    Triangle inequality
- Dissimilarity measures s(x,y) are more general and do not have to satisfy the above metric properties.

<u>Example</u>:

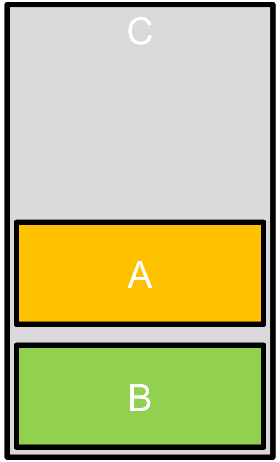
If dissimilarity measure s(x,y) is defined as:

s(x,y) = fraction of x NOT contained in y

then in the example below the triangle inequality is not satisfied, because

$s(a, b) = 100\%$
$s(a, c) = 0\ \%$
$s(c, b) = 80\%$

so $s(a, b) > s(a, c) + s(c, b)$



Note that s(x,y) is also not symmetric because
$s(a, c) = 100\%$ but
$s(c, a) = 20\%$

So $s(x,y) \geq 0$ is a valid dissimilarity measure, but not a distance measure.

Classical metric MDS algorithm

Given an n x n distance matrix D = $\{d_{ij}\}$ for n points in $\mathbb{R}^p$, MDS finds points

$x_1, x_2, \ldots, x_n \in \mathbb{R}^q$ such that a 'stress' function

$$\sum_{i \neq j = 1, \ldots, N} \left(d_{ij} - \|x_i - x_j\|\right)^2$$

that measures distortion is minimized (Euclidean norm).

Does this least-squares criterion look familiar?

Classical MDS with Euclidean distance D is equivalent to extracting two principal components from a PCA analysis. So we'll move on to general metric MDS…

General metric MDS

- Same ‘minimize stress’ goal as classical MDS
- But goes beyond Euclidean distance:
  - More general forms of stress functions.
  - Input matrix of known distances using weighted dimensions.
  - Linear or non-linear mapping $f(d_{ij})$ of the input dissimilarities d_{ij}.
  - Alternative non-Euclidean distance measures in the low-*d* output.

General metric MDS example

We introduce a monotonic (increasing) function $f_\theta(x)$ that transforms the input dissimilarities D = $\{d_{ij}\}$. In effect these create new "target" pseudo-distances in the low-d space. This function *f* has parameters θ. Optimal θ can be learned as part of the optimization. Lots of choices for *f*.

One popular one is ratio MDS: $\hat{d}_{ij} = f_\theta(d_{ij}) = \theta_0 \cdot d_{ij}$

Another is exponential metric MDS:

$$\hat{d}_{ij} = f_\theta(d_{ij}) = \theta_0 + \theta_1 \exp(d_{ij})$$

Given f_θ, the new optimization objective 'stress' function is:

$$\sigma(\hat{d}, X, \theta) = \sum_{i<j} \left(\boxed{f_\theta(d_{ij})} - \|x_i - x_j\|_2\right)^2$$

Handling missing data in MDS

Add weights to the optimization objective ‘stress’ function:

$$\sigma(\hat{d}, X, \theta) = \sum_{i<j} \boxed{w_{ij}} \left(f_\theta(d_{ij}) - \|x_i - x_j\|_2\right)^2$$

w_{ij} =1 if the dissimilarity between points x_i, x_j is defined

w_{ij} =0 if the dissimilarity between points x_i, x_j is undefined

Thus, any pairs with a missing data point will just be skipped by the optimization.

This scheme is used in many other methods beyond MDS!

Non-metric MDS

- Non-metric MDS is used when only the *ranks of the distances* are known, not the absolute dist.
- Now the elements d_{ij} of input matrix *D* are called 'disparities' because they are <u>ordinal</u> values for ranking, *not* real-valued dissimilarities.
- A form of regression called isotonic regression is used to find x_{ij} that respect the ranking condition:

  if $d_{ij} < d_{jk}$ then $\|x_i - x_j\|_2 < \|x_j - x_k\|_2$

Non-metric MDS usage

- Often used with survey data where:
  - Response items are ranked by preference or perceived similarity.
  - Products are defined in terms of attributes which are rated on a scale. Product ranking is then derived from weighted attribute ranking.
- Scenarios where objects can be ranked but where a numeric distance or dissimilarity measure is hard to define.

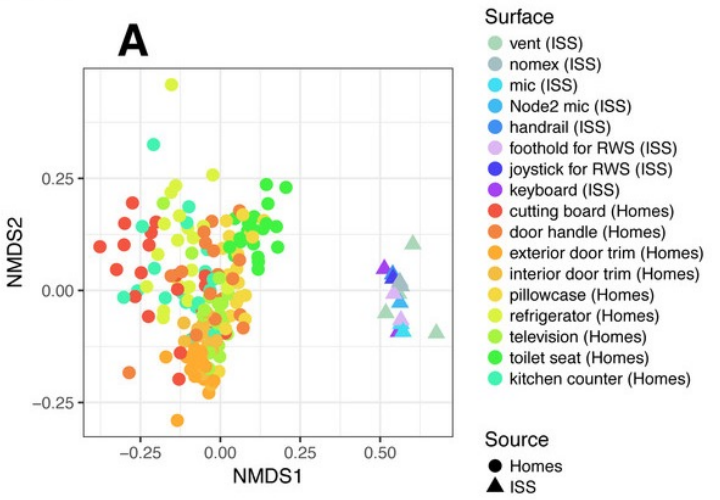


Figure 4: Non-metric multidimensional scaling (NMDS) ordination plots of ISS surface sample

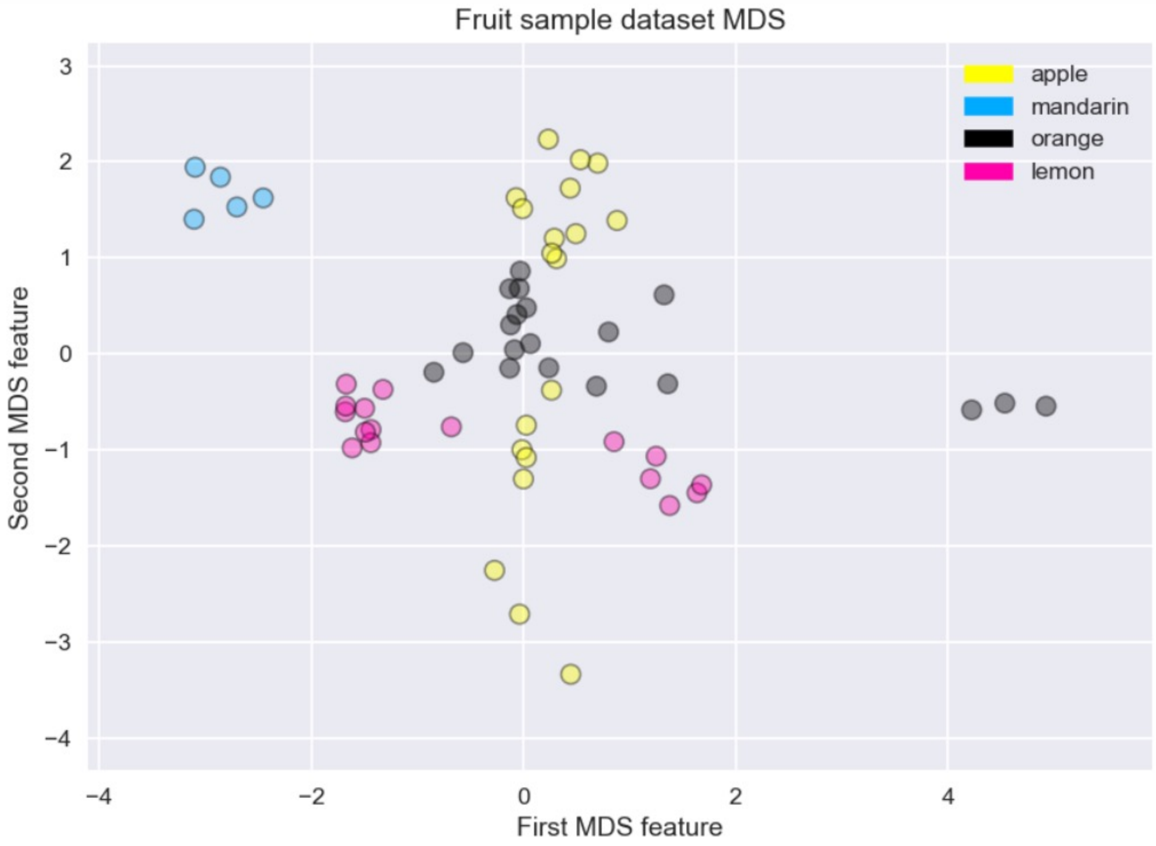
Similarity of microbial communities on ISS vs Earth home

Source: J.M. Lang et al. (2017) A microbial survey of the International Space Station.
https://peerj.com/articles/4029/

Notebook: MDS on the Fruit Dataset

MDS defaults to metric MDS (ratio mapping)

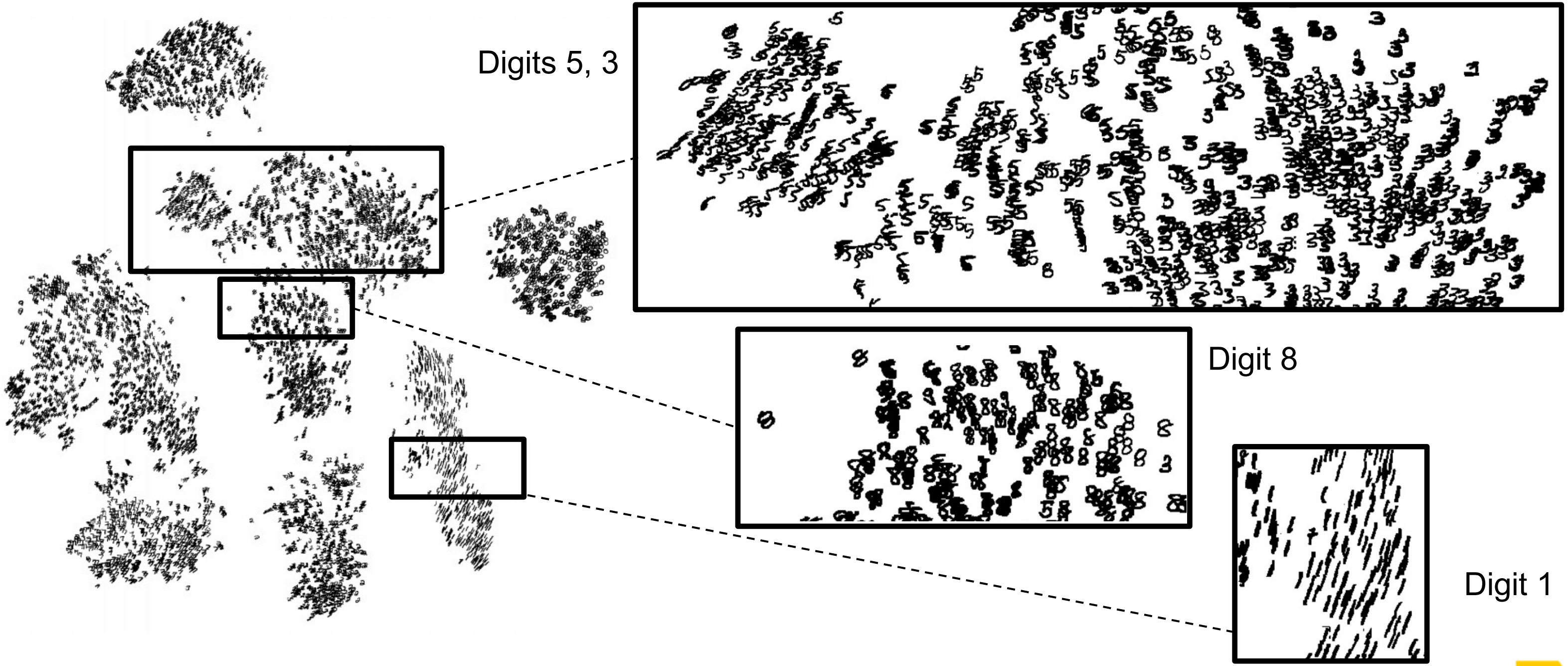
```
# Multidimensional scaling
from adspy_shared_utilities import plot_labelled_scatter
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import MDS

# each feature should be centered (zero mean) and with unit variance
X_fruits_normalized = StandardScaler().fit(X_fruits).transform(X_fruits)

mds = MDS(n_components = 2)

X_fruits_mds = mds.fit_transform(X_fruits_normalized)

plot_labelled_scatter(X_fruits_mds, y_fruits, ['apple', 'mandarin', 'orange', 'lemon'])
plt.xlabel('First MDS feature')
plt.ylabel('Second MDS feature')
plt.title("Fruit sample dataset MDS")
```


Fruit sample dataset MDS
apple
mandarin
orange
lemon
Second MDS feature
First MDS feature
3
2
1
0
−1
−2
−3
−4
−4
−2
0
2
4

t-SNE: t-Distributed Stochastic Neighbor Embedding (van der Maaten & Hinton, 2008)

- Like MDS, it's a manifold learning method that optimizes a set of low-dim output points to preserve similarities in high-dim data.
- But unlike MDS it is focused on preserving <u>local distances</u> between neighbors, not global distance structure.
- Thus it should be used for visualization, not clustering.

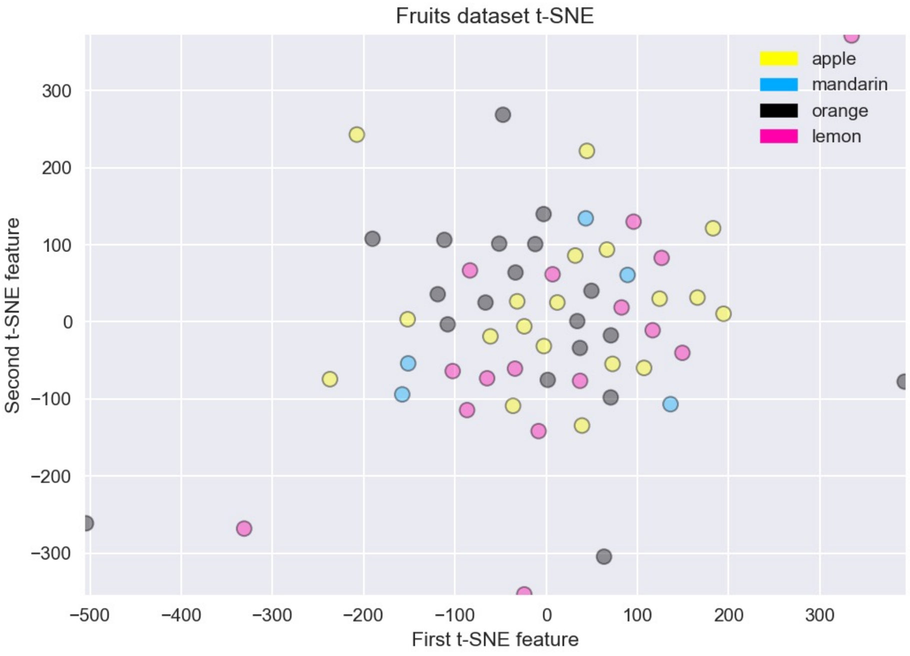
t-SNE: a powerful manifold learning method that finds a 2D projection preserving information about neighbors



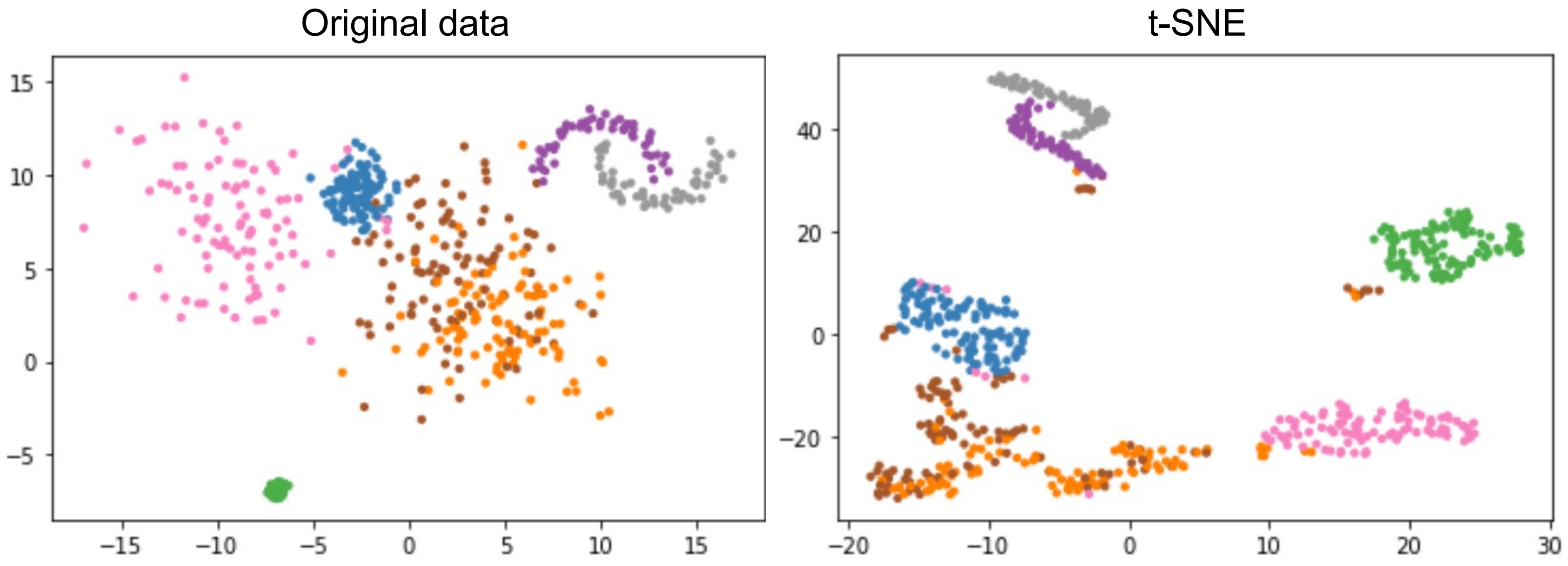
Source: van der Maaten & Hinton https://lvdmaaten.github.io/tsne/

t-SNE on the Fruit Dataset

```
from sklearn.manifold import TSNE

tsne = TSNE(random_state = 0)

X_tsne = tsne.fit_transform(X_fruits_normalized)

plot_labelled_scatter(X_tsne, y_fruits,
    ['apple', 'mandarin', 'orange', 'lemon'])
plt.xlabel('First t-SNE feature')
plt.ylabel('Second t-SNE feature')
plt.title("Fruits dataset t-SNE")
```



t-SNE does not preserve cluster separation or density! (by design)



- Use t-SNE for exploration and visualization, not clustering

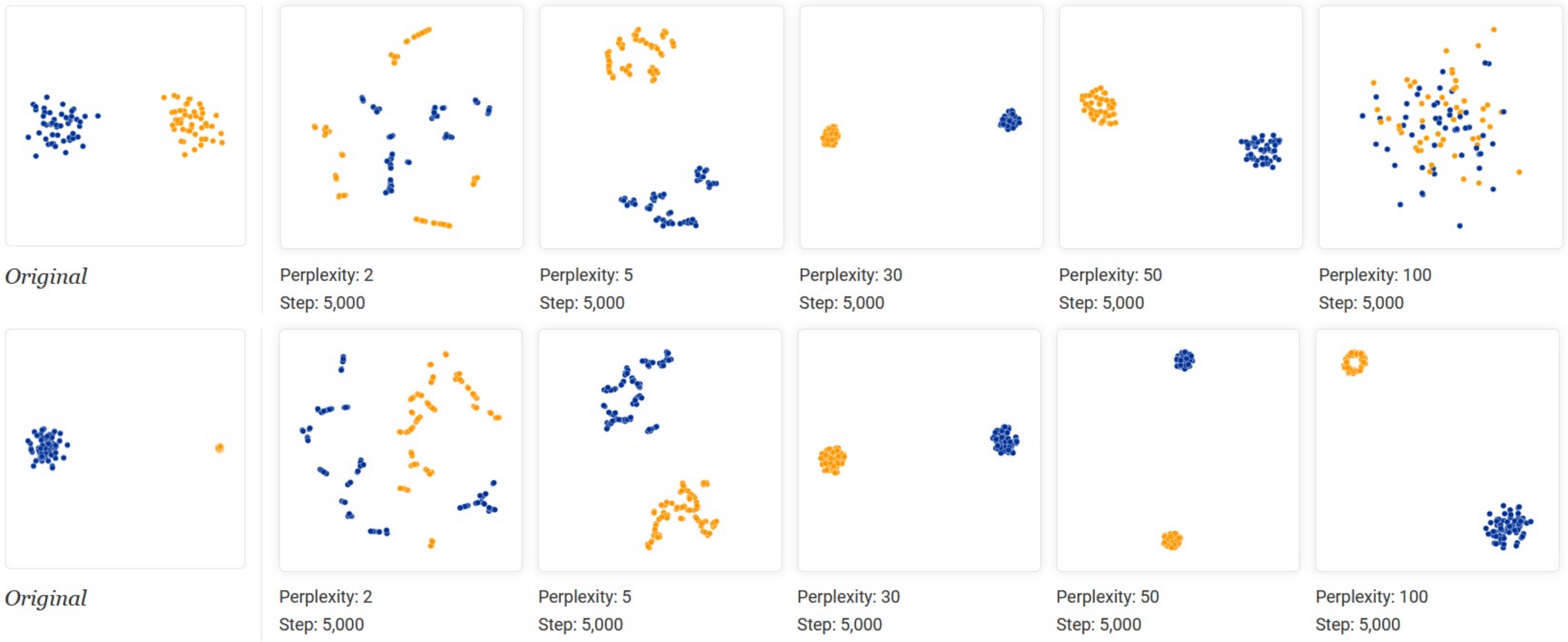
Using t-SNE effectively

- Perplexity is a key parameter of t-SNE.
- It’s roughly an estimate of how many neighbors are near each data point.
- You should experiment with multiple values of perplexity.
- Like other complex optimization methods, it can give different results every time you run it.

You can find an interactive guide and demonstration for t-SNE here. Experiment with t-SNE perplexity and other parameters:

[Wattenberg, et al., "How to Use t-SNE Effectively", Distill, 2016.]

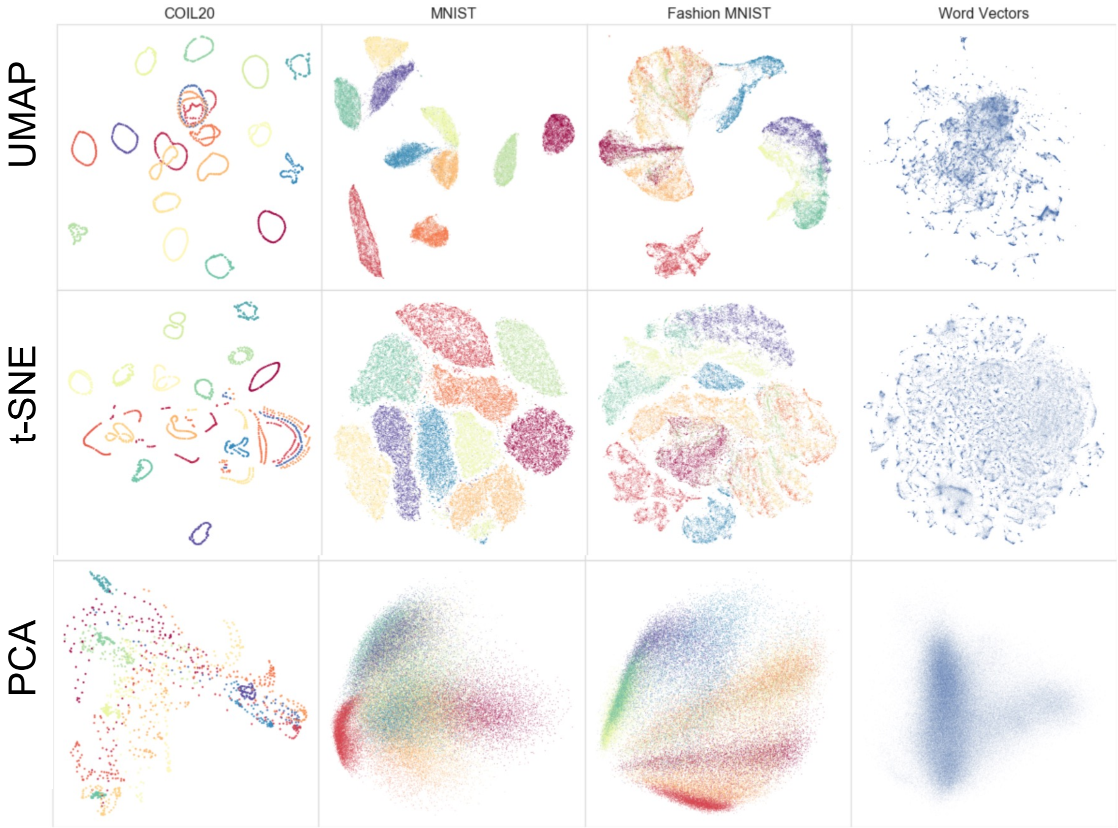
t-SNE results can be very sensitive to the value of perplexity parameter



Source: [Wattenberg, et al., "How to Use t-SNE Effectively", Distill, 2016.](#)

The UMAP algorithm: Uniform Manifold Approximation & Projection
(McInnes, Healy, Melville, 2018)

- Similar to t-SNE: oriented toward preserving local neighborhood structure.
- But also better at preserving some global structure than t-SNE.
- Scales to much larger datasets than t-SNE.
- No restrictions on output (embedding) dimension.
- So it's appropriate for both visualization and general dim reduction for use with ML (e.g. supervised learning, clustering)
- Not part of scikit-learn, but compatible with it.
- See https://umap-learn.readthedocs.io/en/latest/index.html#
- Paper on arXiv: https://arxiv.org/abs/1802.03426



Source: https://arxiv.org/abs/1802.03426

Summary of dimensional reduction algorithms

Two main categories of dimension reduction algorithms:

1. Preserve the distance structure within the data, globally
   - PCA, MDS (and others like Sammon mapping)
2. Preserve local distances (neighborhood structure) over global distance.
   - T-SNE, UMAP (and other like Isomap, LargeVis, …)

- For very large, high-dimensional datasets sometimes combining both approaches is appropriate.
  - First apply PCA, then t-SNE or UMAP

UNIVERSITY OF MICHIGAN

Kevyn Collins-Thompson

kevynct@umich.edu

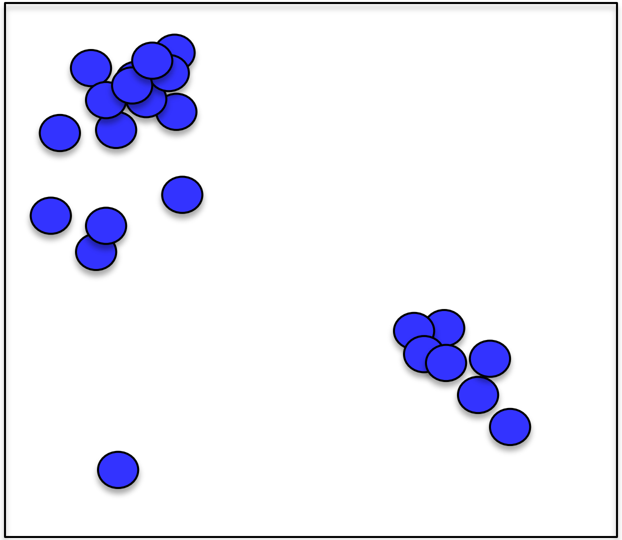
School of Information

Density Estimation

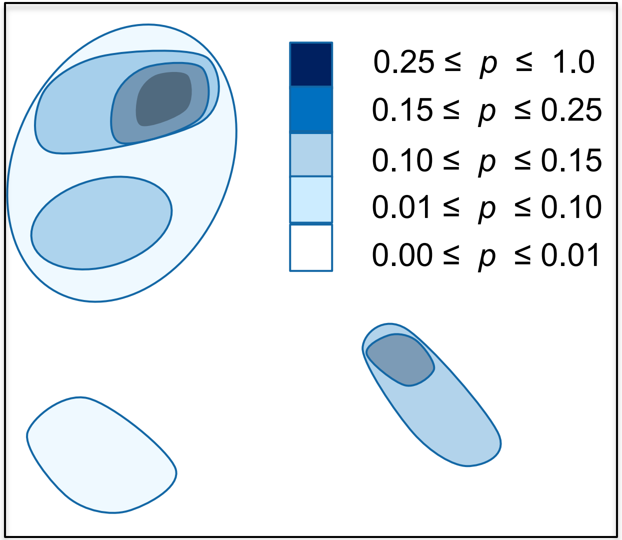
Kevyn Collins-Thompson

Associate Professor of Information and Computer Science
School of Information, University of Michigan

Transformations: Density Estimation



Individual measurements

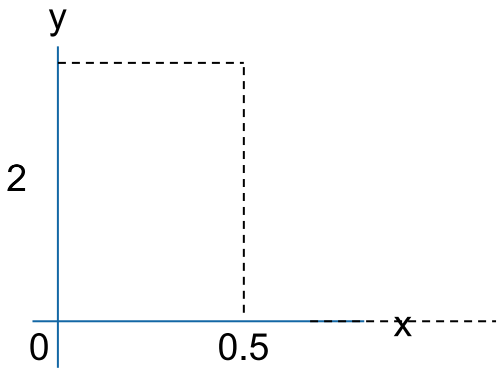


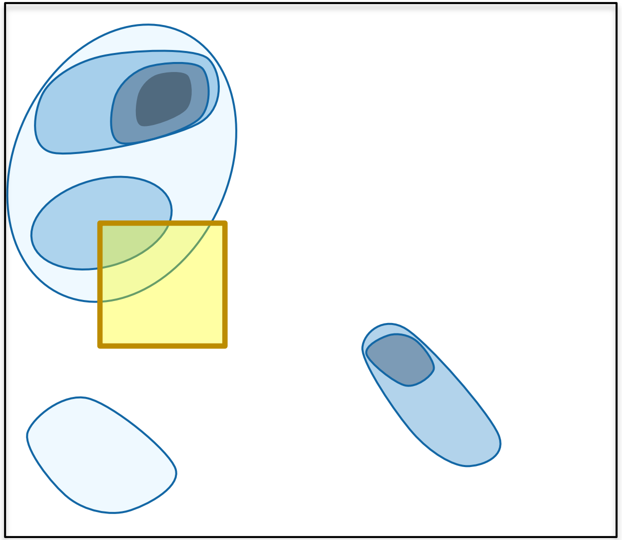
Density estimate

(Estimated probability p of observing a measurement at a given location)

Probability density functions

To compute the probability of seeing an instance/event in a particular region, integrate the density function over that region.





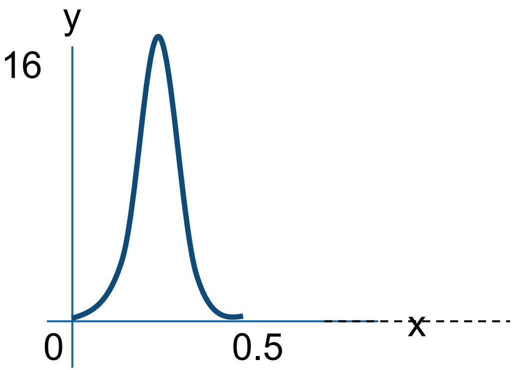
A density function $f(x)$ must integrate to 1 over the domain of X.. But its range over the domain is not restricted to being between 0 and 1.

For random variable X with density function $f_X(x)$

$$\Pr[a \leq X \leq b] = \int_a^b f_X(x)\,dx.$$

Probability density functions

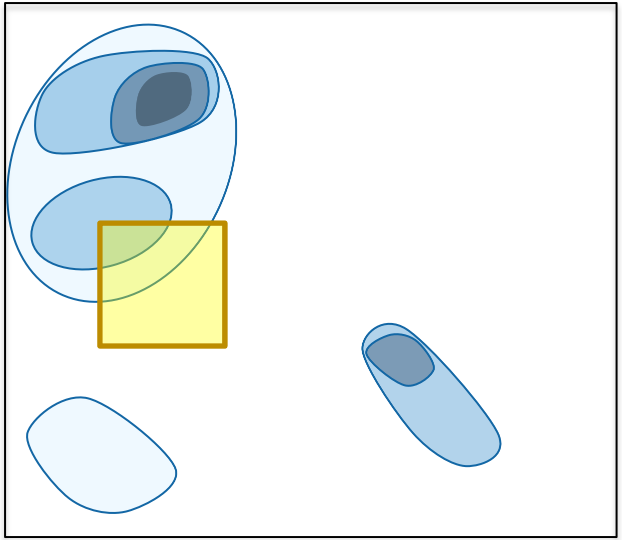
To compute the probability of seeing an instance/event in a particular region, integrate the density function over that region.



A density function $f(x)$ must integrate to 1 over the domain of X. But its range over the domain is not restricted to being between 0 and 1.

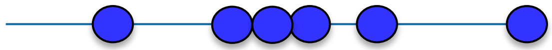
For random variable X with density function $f_X(x)$

$$\Pr[a \le X \le b] = \int_a^b f_X(x)\, dx.$$

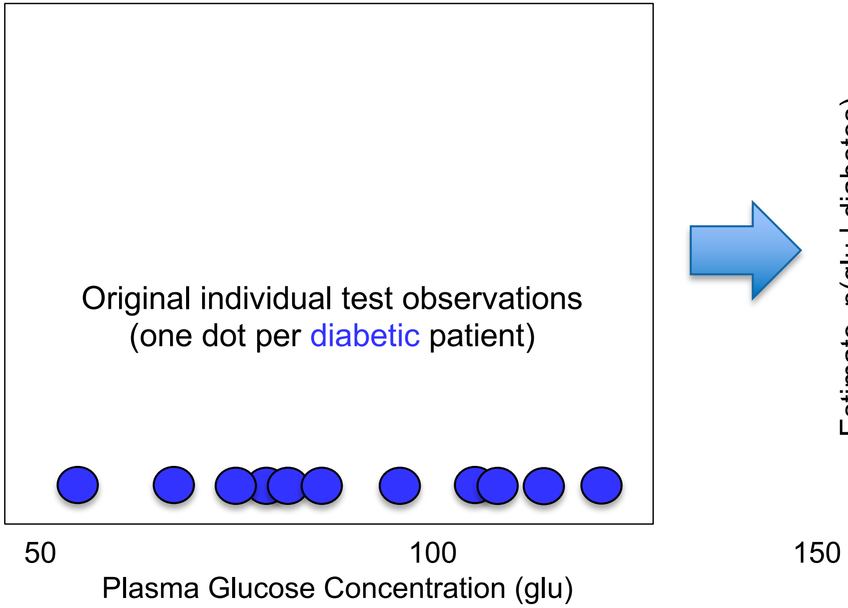


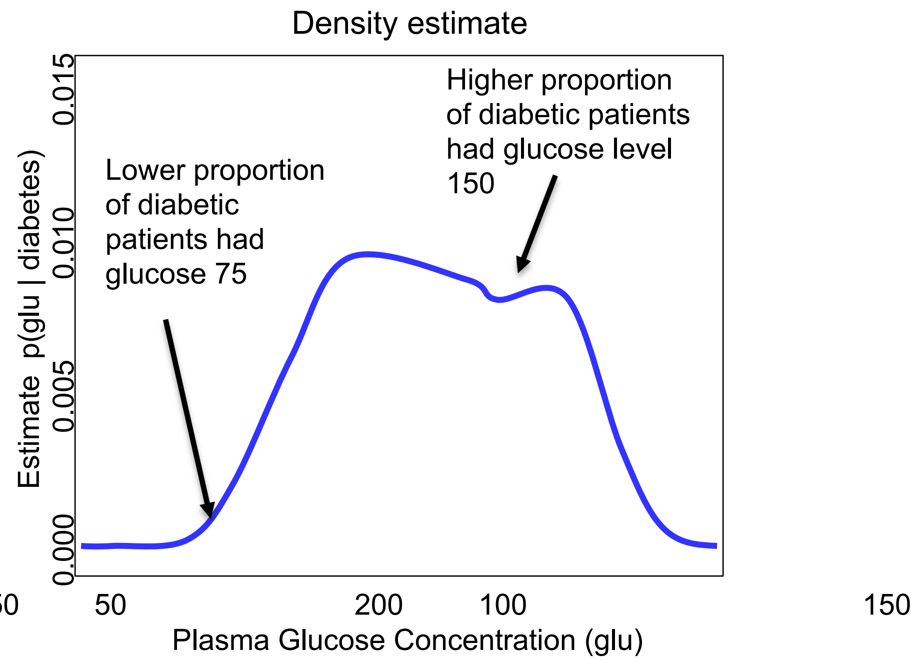
Two main families of density estimators

- Parametric
  - Assumes the data follow a specific, parametrized probability distribution
    - e.g. Gaussian, Exponential, Poisson, Beta, etc.
    - Maximum likelihood estimates for their parameters are estimated from the given data.
    - e.g. for Gaussian, the sample mean and std. dev.
- Non-parametric (our focus here)
  - Does not assume anything about the nature of the probability distribution of the data
    - e.g. Histograms, Kernel density estimation

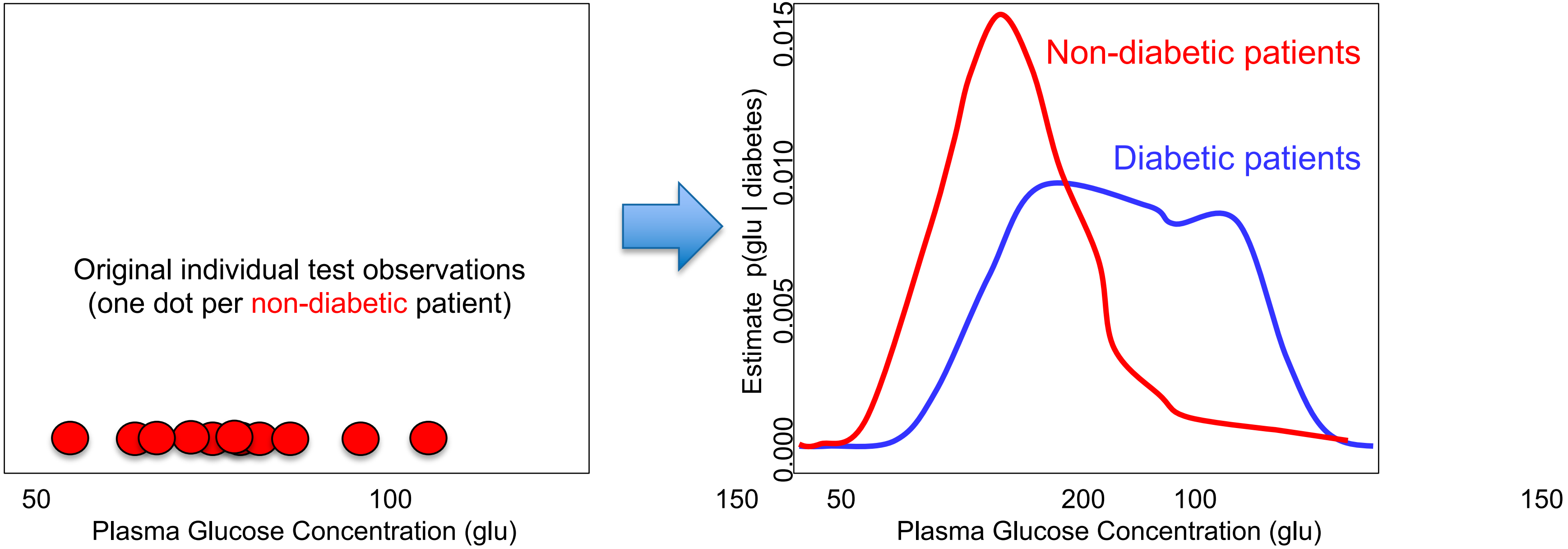


Density estimation example (1-d)





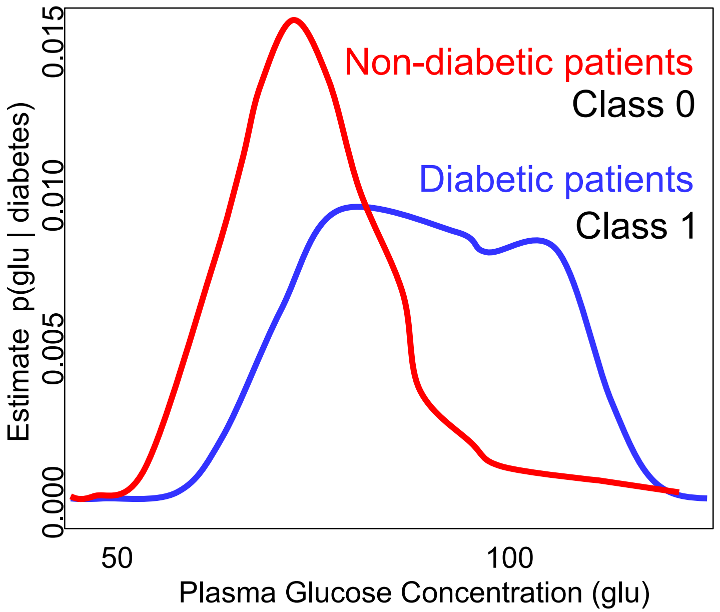
Density estimation example (1-d)



Application of density estimation: Probabilistic (Bayesian) classification

Suppose we have a binary class variable C we want to predict.
To classify a point x we need to compute $p(C|x)$

Since we computed the density function we now know $p(x|C)$.
But how do we get $p(C|x)$ from that?



Application of density estimation: Probabilistic (Bayesian) classification

Suppose we have a binary class variable C we want to predict.
To classify a point x we need to compute $p(C|x)$

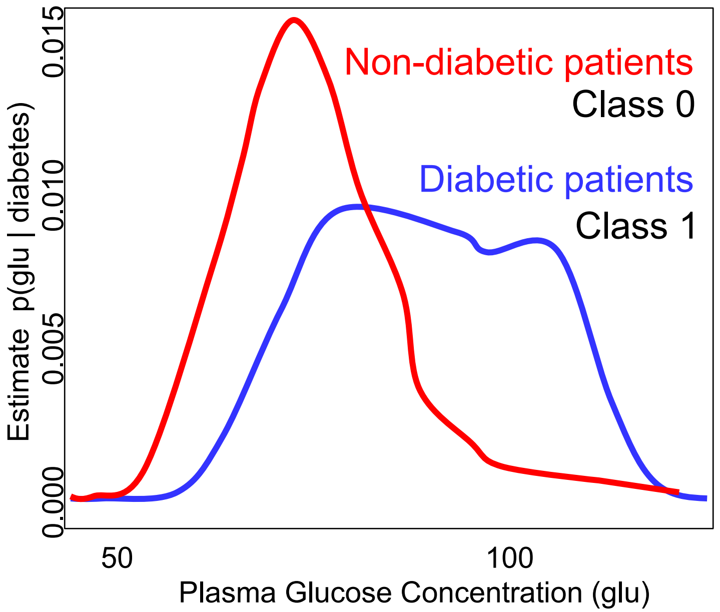
Since we computed the density function we now know $p(x|C)$.
But how do we get $p(C|x)$ from that?

<u>Bayes' Theorem</u>!

$$\boxed{p(C|x) = \frac{p(x|C)p(C)}{p(x)}}$$

The classifier will predict the most likely class,
i.e. whichever is greater: $p(C = 0|x)$ or $p(C = 1|x)$
We drop the $p(x)$, which is the same for both, to get:

$$\boxed{\begin{array}{l} p(C = 0|x) \propto p(x\,|C = 0)\; p(C = 0) \\ p(C = 1|x) \propto p(x\,|C = 1)\; p(C = 1) \end{array}}$$



Note! This requires us to specify a prior belief *p*(*C*).
This is an example of a classifier based on a <u>generative model</u>: it models each class explicitly (instead of just the boundary between them).

Local density estimators

- Estimate the density in a small region to be

Density: "normalized (fraction of total) points per unit volume"

$$p_i = \frac{n_i}{N \, \Delta_i}$$

points in region

volume of region

total points

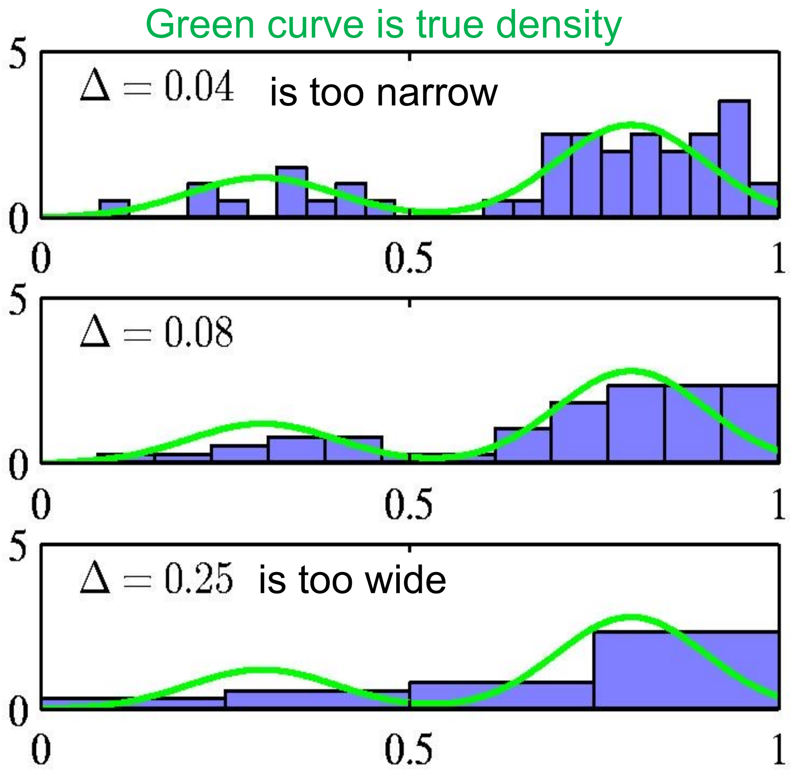
- Problem 1: High variance in estimate if "bin" volume is too small with too few points per "bin".
- Problem 2: Too-low variation across the region if "bin" volume is too big compared with the smoothness of the true density.

Histograms are density estimators

Low dimensional data:
use a histogram as a density estimator.

- How wide should the bins be? (width=regularizer)
- Do we want the same bin width everywhere?
- Do we believe the true density is actually zero for empty bins?

$$p_i = \frac{n_i}{N\,\Delta_i}$$



Some good and bad properties of histograms as density estimators

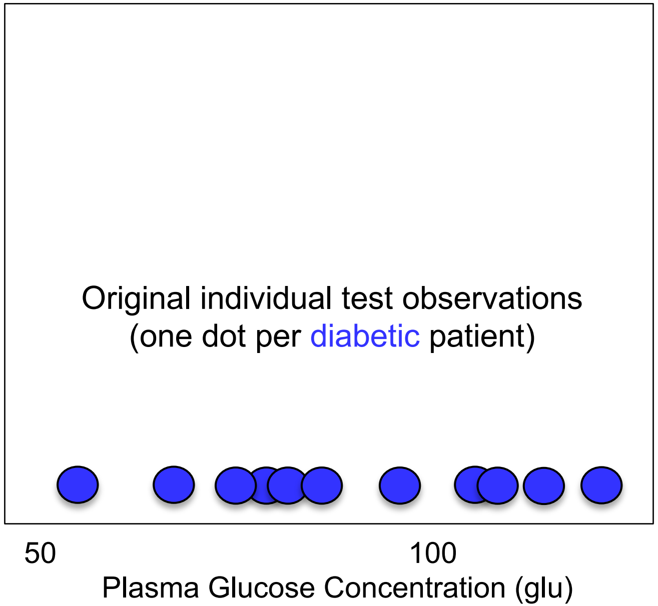
- No need to fit a particular model to the data.
    - We just compute some very simple statistics (the number of datapoints in each bin) and store them.
- The number of bins is exponential in the dimensionality of the dataspace. So high-dimensional data is tricky:
    - We must either use big bins or get lots of zero counts (or adapt the local bin width to the density)
- The density has annoying discontinuities at the bin boundaries.
    - We must do better by using some kind of smoothing.

Kernel density estimators

Use regions centered on the datapoints

- Allow the regions to overlap.
- Let each individual region contribute a total density of 1/N
- Use regions with soft edges to avoid discontinuities (e.g. Gaussians)
- For a sample of points $x_1 \dots x_n$

$$p(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N} \frac{1}{(2\pi\sigma^2)^{D/2}} \exp\left(\frac{||\mathbf{x}-\mathbf{x}_n||^2}{2\sigma^2}\right)$$

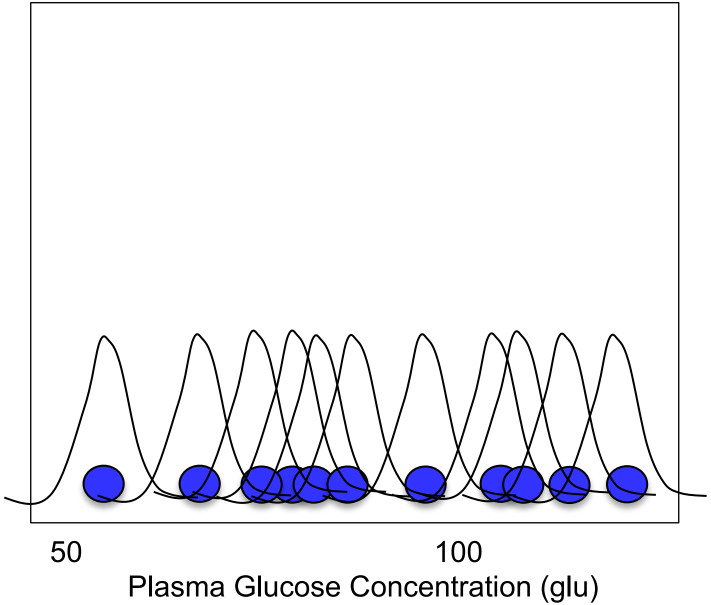


Kernel density estimators

Use regions centered on the datapoints

- Allow the regions to overlap.
- Let each individual region contribute a total density of 1/N
- Use regions with soft edges to avoid discontinuities (e.g. Gaussians)
- For a sample of points $x_1 \ldots x_n$

$$p(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N} \frac{1}{(2\pi\sigma^2)^{D/2}} \exp\left(\frac{||\mathbf{x}-\mathbf{x}_n||^2}{2\sigma^2}\right)$$

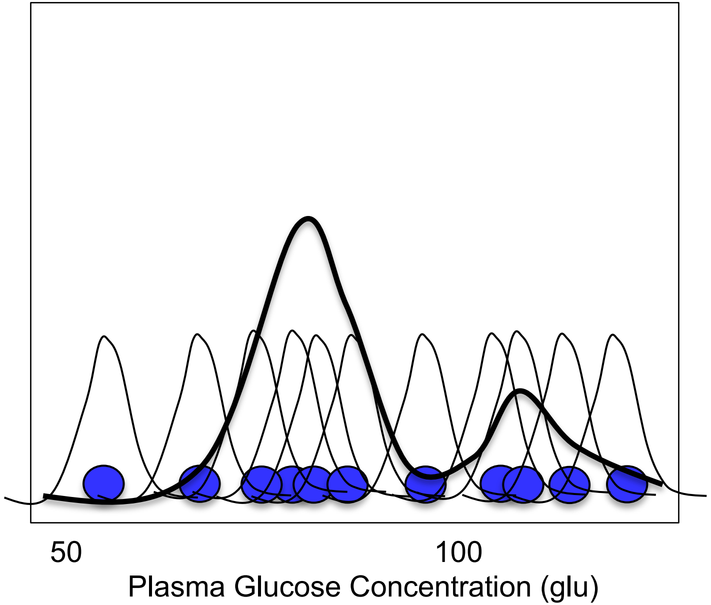


Kernel density estimators

Use regions centered on the datapoints

- Allow the regions to overlap.
- Let each individual region contribute a total density of 1/N
- Use regions with soft edges to avoid discontinuities (e.g. Gaussians)
- For a sample of points $x_1 \ldots x_n$

$$p(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N} \frac{1}{(2\pi\sigma^2)^{D/2}} \exp\left(\frac{||\mathbf{x}-\mathbf{x}_n||^2}{2\sigma^2}\right)$$



Kernel functions

- A kernel is a positive function K(*x*, *h*) controlled by the bandwidth parameter *h.*
- The density estimate $\boldsymbol{\rho_K(x)}$ at a point *y* within a given group of "training" points $\{\boldsymbol{x_i}\}$ is given by:

$$\boldsymbol{\rho_K(x) = \sum_{i=1}^{N} K((x - x_i)/h)}$$

- The bandwidth acts as a *smoothing* parameter
  - LARGE bandwidth -> very smooth (high-bias) density estimator
  - small bandwidth -> very bumpy (high-variance) density estimator

Kernel density estimation in scikit-learn

`KernelDensity` class in `sklearn.neighbors`

```
kde = KernelDensity(kernel='gaussian', bandwidth=0.75).fit(X)
```

How do I change the kernel being used? *Set the kernel parameter.*

```
kde = KernelDensity(kernel='tophat', bandwidth=0.75).fit(X)
```

How do I change the width of the kernel? *Set the bandwidth parameter.*

```
kde = KernelDensity(kernel='gaussian', bandwidth=0.75).fit(X)
```

How to get the probability of a new instance under the estimated density?

`kde.score_samples(X)` outputs log probability of X

Kernel density estimation in scikit-learn

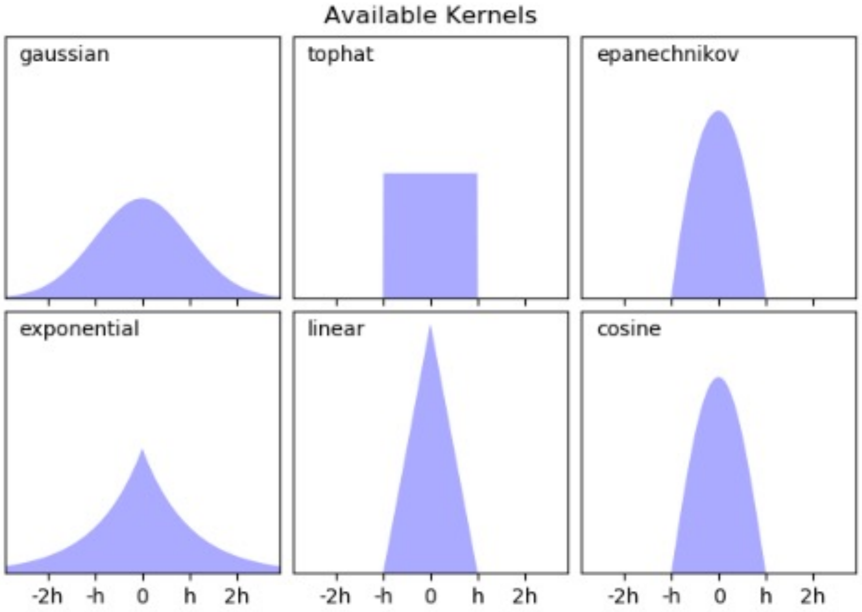
`KernelDensity` class in `sklearn.neighbors`

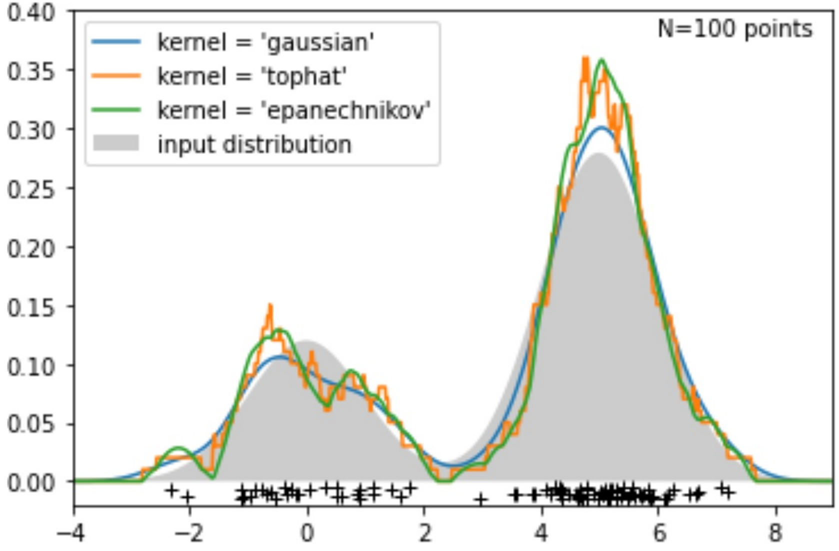
`kde = KernelDensity(kernel='gaussian’, bandwidth=0.75).fit(X)`

```
#### Get probabilities under the estimated density, using score_samples on a toy dataset
X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
outliers = np.array([[-5, -6], [7, 8]])

kde = KernelDensity(kernel='gaussian', bandwidth=0.3).fit(X)

# note that we call np.exp() since score_samples returns LOG probabilities
print("Probabilities of training points:", np.exp(kde.score_samples(X)))
print("Probabilities of new (outlier) points:", np.exp(kde.score_samples(outliers)))
```

```
Probabilities of training points: [0.3  0.3  0.29 0.3  0.3  0.29]
Probabilities of new (outlier) points: [1.64e-049 1.02e-126]
```

A variety of kernels are available in scikit-learn

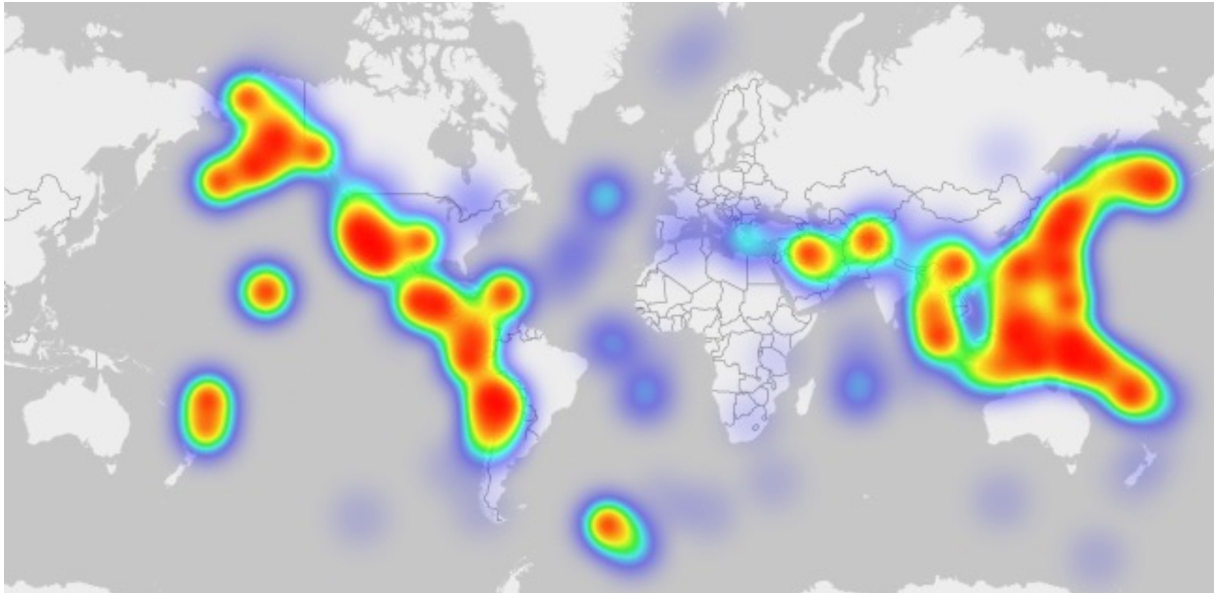




Source: https://scikit-learn.org/stable/modules/density.html

Kernel Density Example

`KernelDensity` is especially popular for creating heatmaps



Recent global earthquake activity (U.S. Geological Survey data)

Source: http://www.digital-geography.com/csv-heatmap-leaflet/

How can we evaluate density estimators?

Compute held-out log-likelihood:

Given a density estimator M and a held-out dataset X with instances x_i
You can compute how likely any given x_i is under the estimated density $\hat{P}(x_i|M)$

$$\hat{P}(data|M) = P(x_1 \& \, x_2 \& \, \ldots \& \, x_R|M) = \prod_i \hat{P}(x_i|M)$$

This is the held-out data likelihood. It assumes the dataset instances/records are 'generated' independently under the model.

$$\log \hat{P}(data|M) = \sum_i \log \hat{P}(x_i|M)$$

Finding the optimal bandwidth: using grid search and cross-validation to maximize held-out log-likelihood.

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import LeaveOneOut

## This uses the X random 1-d data computed in a previous cell.

## Note that the score (being optimized) by default is the total log-likehihood of the data
## which is what the KernelDensity score(...) methods returns.

bandwidths = 10 ** np.linspace(-1, 1, 20)
grid = GridSearchCV(KernelDensity(kernel='gaussian'),
                    {'bandwidth': bandwidths},
                    cv=LeaveOneOut())
grid.fit(X);
print("Optimal bandwidth:", grid.best_params_)
print("Data log-likelihood:", grid.best_score_)
```

```
Optimal bandwidth: {'bandwidth': 0.42813323987193935}
Data log-likelihood: -1.9649814109010284
```

Parametric density estimation methods

- Simple joint density (see e.g. Naïve Bayes)
- Gaussian mixture models (sklearn.mixture.GaussianMixture)

The Gaussian Mixture Model (GMM) is a widely-used density estimation algorithm

The probability distribution of *x* is modeled as a simple linear combination of Gaussian components:

$$\hat{f}_K(\boldsymbol{x}) = \sum_{k=1}^{K} w_k \cdot \phi(\boldsymbol{x}|\mu_k, \Sigma_k)$$

$$\boldsymbol{x} = (x_1, x_2, \dots, x_d)$$

Picking the # of components K is one challenge.
We'll revisit this!

In short, methods include: BIC criterion, Variational GMM.

Increasing K leads to large number of parameters
K-1 weight parameters (they sum to 1)
K x d parameters for the mean vectors
K x d(d+1)/2 in the Gaussian covariance matrices

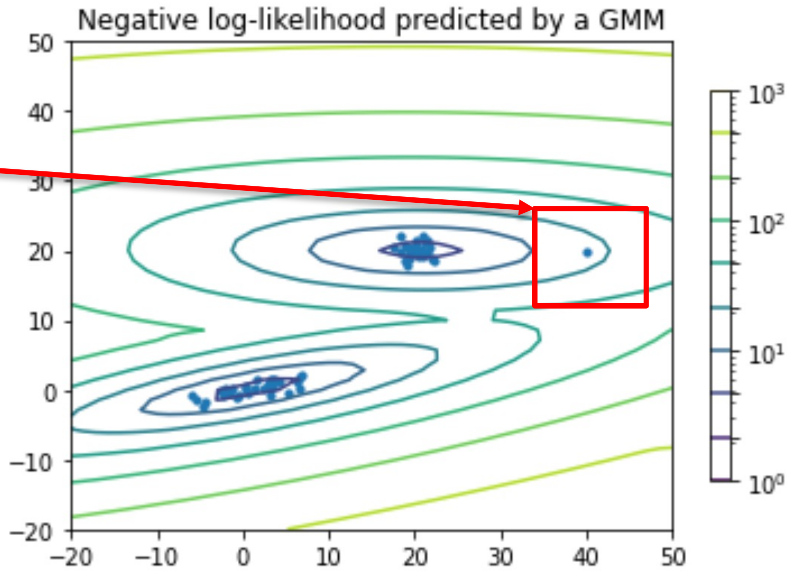
Could assume all cov matrices are equal for all k

Each mixture component (usually) represents a unique cluster with center μ_k and covariance Σ_k
Optimal parameters are estimated with the EM algorithm – more on this and EM in another lecture!

Example of GMM with two components, showing negative log-likelihood plot

Another application of density estimation: anomaly/outlier detection

```
# fit a Gaussian Mixture Model with two components
clf = mixture.GaussianMixture(n_components=2,covariance_type='full')
clf.fit(X_train)

# display predicted scores by the model as a contour plot
x = np.linspace(-20., 50.)
y = np.linspace(-20., 50.)
X, Y = np.meshgrid(x, y)
XX = np.array([X.ravel(), Y.ravel()]).T
Z = -clf.score_samples(XX)
Z = Z.reshape(X.shape)
```



Pros and cons of Gaussian mixture models

Advantages:

- Speed: fastest algorithm for learning mixture models.
- Agnostic: As this algorithm maximizes only the likelihood, it will not bias the
- means towards zero, or bias the cluster sizes to have specific structures that might or might not apply.

Disadvantages:

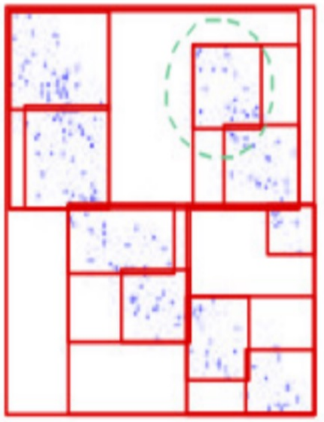
- Singularities When one has insufficiently many points per mixture, estimating the covariance
- matrices becomes difficult, and the algorithm is known to diverge and find solutions with infinite
- likelihood unless one regularizes the covariances artificially.

- Number of components This algorithm will always use all the components it has access to,
- needing held-out data or information theoretical criteria to decide how many components to use
- in the absence of external cues.

High-dimensional density estimation

- Bins become hypercubes in high-dim space
- The number of hypercubes increases exponentially!
- We can mitigate the curse of dimensionality and cost of searching a high-d space if we build an index on the training data first.
- Makes finding nearest neighbors fast so that we don't have to compare with all N points in the entire data set
- Two indexing methods: K-D trees, Ball trees (both supported in scikit-learn)
- Trade off computation time for accuracy with `atol` (abs tolerance) & `rtol` (rel tolerance) params.

K-D Trees



Ball trees

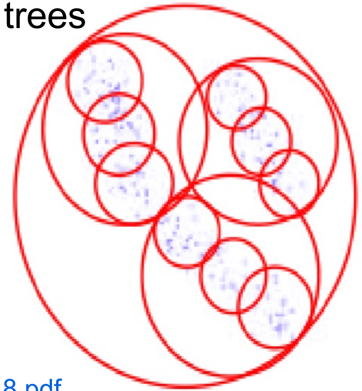
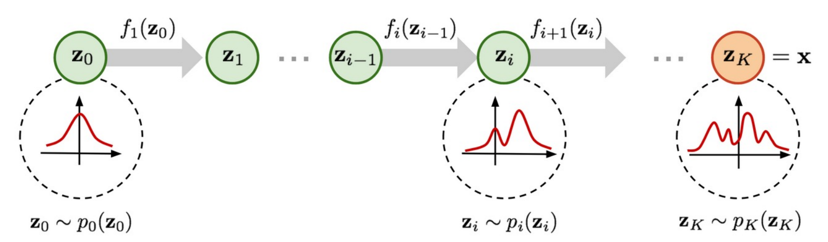


Image source: https://sites.stat.washington.edu/mmp/courses/stat534/spring19/Handouts/lecture-april18.pdf

Deep learning methods for density estimation

Normalizing flows: Intro & Review. Kobyzev, Prince, Brubaker (2020) https://arxiv.org/pdf/1908.09257.pdf
Density estimation using Real NVP. Dinh, Sohl-Dickstein, Bengio (2017) https://arxiv.org/pdf/1605.08803.pdf

- Normalizing flows / Real VAP
- Estimate arbitrarily complex transformation (density estimation) by:
  - Choosing an initial simple density (‘base density’).
  - Applying a sequence of ‘nice’ invertible smooth transformations.
  - To obtain a series of successively more complex densities.
  - Eventually obtaining a (complex) probability distribution for the target variable.
- See linked papers for details.



Source: https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

UNIVERSITY OF MICHIGAN

Kevyn Collins-Thompson

kevynct@umich.edu

School of Information