Classification, Decoding & Clustering

Lucas Pinto
Assistant Professor
Department of Neuroscience
Northwestern University

NUIN 443, 04/03/2024

Outline

- Classification
  - LDA, SVM
  - Primer: Deep nets
- Decoding (esp. Bayesian decoders)
- Clustering
  - K-means, Gaussian Mixtures, Hierarchical
  - Primer: Spectral clustering

Classification

Introduction

A class *G* is a set of discrete categories that cannot be meaningfully ordered

$$G = \{\text{blueberry muffin}, \text{chihuahua}\}$$

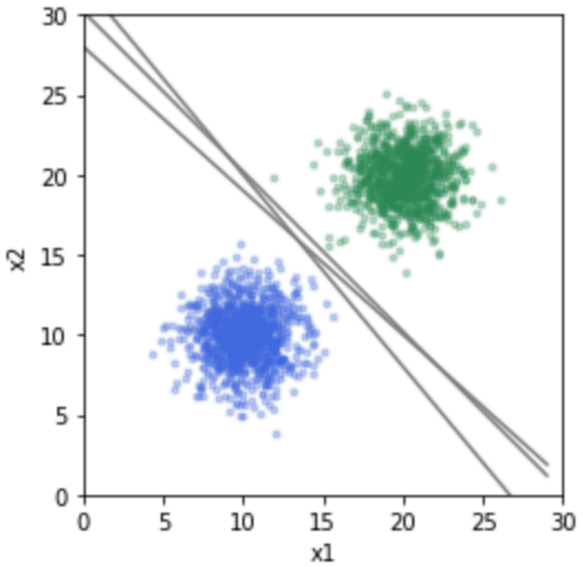
Throughout this lecture, we will refer to *G* as a set of *K* categories, which we might number for convenience

$$k = 1, 2, \ldots, K$$

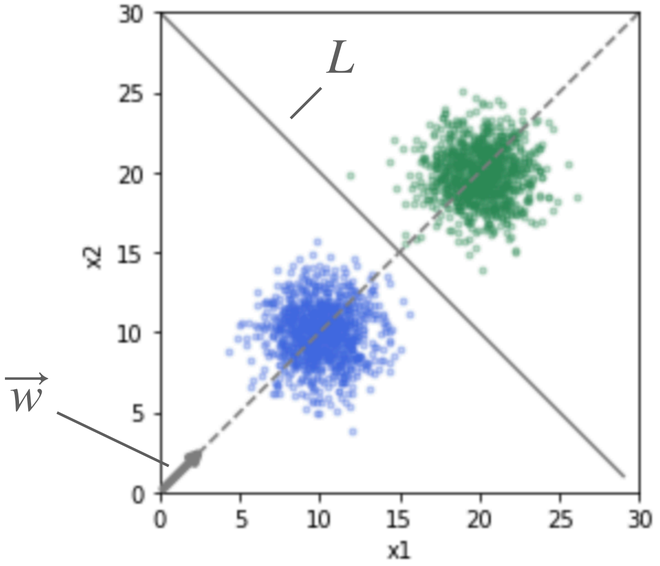
In classification problems, given a training set of labeled observations, we try to assign a label to a new observation *i* defined by a vector of of features, $\vec{x}_i$

Separating hyperplanes

Many classification problems involve estimating the hyperplane *L* that best separates the data categories



In this 2D case, *L* is a line. For these two categories *B* and *G*, there is an orthogonal vector to *L*, $\vec{w}$, such that



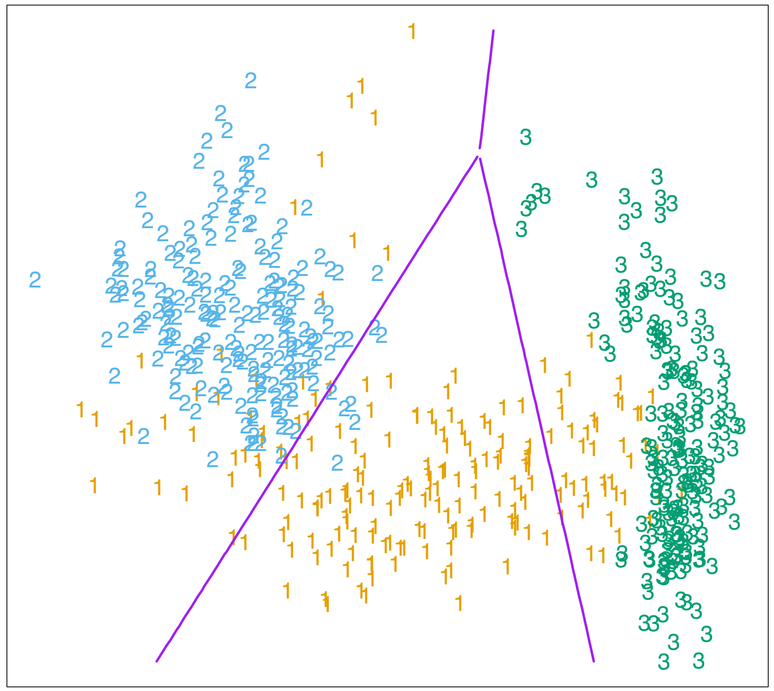
$$\vec{x}_B^T \vec{w} + w_0 \leq 0 \qquad \vec{x}_G^T \vec{w} + w_0 \geq 0$$

And for any point x_L lying in *L*:

$$\vec{x}_L^T \vec{w} + w_0 = 0$$

Linear discriminant analysis (LDA): intro

LDA finds a linear combination of features that best separates *K* classes. As such, it's related to regression and PCA



Hastie et al., 2017

Some definitions / reminders:

$P(G = k \mid X)$: conditional probability of the class given the data

$f_k(X) = P(X \mid G = k)$: class-conditional probability density function for *G* = *k*

π_k: prior probability of *G* = *k*, with $\sum_{j=1}^{K} \pi_j = 1$

Bayes rule: $P(A \mid B) = \dfrac{P(B \mid A)P(A)}{P(B)}$

Applying Bayes rule we get: $P(G = k \mid X) = \dfrac{f_k(X)\pi_k}{\sum_{j=1}^{K} f_j(X)\pi_j}$

- The choice of $f_k(X)$ determines the exact flavor of classification we use. For example, if we assume it is non-parametric and that *X* is conditionally independent in each class, we get the *Naïve Bayes classifier*

LDA: derivation

LDA assumes that $f_k(X)$ is a multivariate Gaussian of p dimensions, and that all classes have the same covariance matrix Σ.

$$f_k(X) = \frac{1}{(2\pi)^{p/2}\sqrt{\Sigma_k}} e^{-\frac{1}{2}(X-\mu_k)^T\Sigma_k^{-1}(X-\mu_k)} \ , \ \Sigma_k = \Sigma \ \forall k$$

In the simplest case where we want to compare two classes k and ℓ, it is sufficient to look at the log odds

$$\begin{aligned}
\log \frac{P(G=k\,|\,X)}{P(G=\ell\,|\,X)} &= \log \frac{f_k(X)\pi_k}{f_\ell(X)\pi_\ell} \\
&= \log \frac{f_k(X)}{f_\ell(X)} + \log \frac{\pi_k}{\pi_\ell} \\
&= \log \frac{\pi_k}{\pi_\ell} - \frac{1}{2}(\mu_k+\mu_\ell)^T\Sigma^{-1}(\mu_k-\mu_\ell) \\
&\quad + X^T\Sigma^{-1}(\mu_k-\mu_\ell)
\end{aligned}$$

(From this)

- Note that this is linear in X, and because the Σ's are equal a bunch of terms cancel out

We estimate the distribution parameters directly from the labeled data:

$\hat{\pi}_k = N_k/N$ N being the number of observations

$$\hat{\mu}_k = 1/N \sum_{g_i=k} x_i$$

$$\hat{\Sigma} = 1/(N-K) \sum_{j=1}^{K} \sum_{g_i=k} (x_i-\mu_j)(x_i-\mu_j)^T$$

The class boundary will be the line (hyperplane) where both classes are equally likely, i.e. the log-odds = 0.

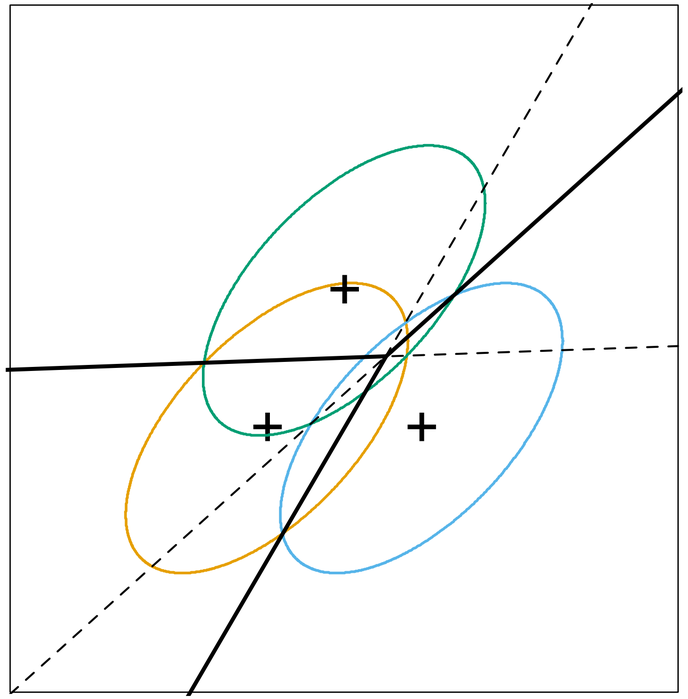
Thus, we classify a data point $\overrightarrow{x_i}$ as ℓ if

$$\overrightarrow{x_i}^T\hat{\Sigma}^{-1}(\hat{\mu}_k-\hat{\mu}_\ell) > \frac{1}{2}(\hat{\mu}_k+\hat{\mu}_\ell)^T\hat{\Sigma}^{-1}(\hat{\mu}_k-\hat{\mu}_\ell) - \log\frac{N_\ell}{N_k}$$

Equivalently, the linear discriminant function for k is

$$\delta_k(X) = \log P(G=k\,|\,X) = X^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log\pi_k$$

LDA: notes



Hastie et al., 2017

For more classes, we can compute all the pairwise linear thresholds, or do partitions to classify one *k* vs. all others, and then combine them.

- Note that, for spherical clusters, the threshold is just a line perpendicular to the bisecting lines between class centroids.

Quadratic discriminant analysis (QDA)

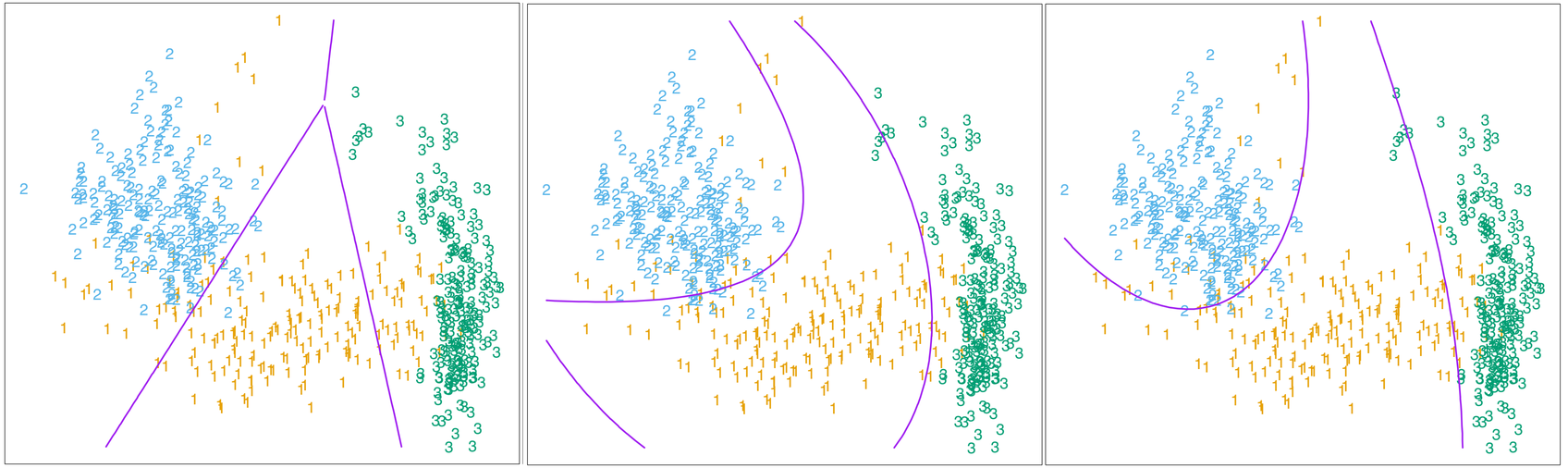
If we no longer assume that all class covariance matrices are equal, we can no longer cancel out some of the terms, and the discriminant function becomes

$$\delta_k(X) = -\frac{1}{2}\log|\Sigma_k| - \frac{1}{2}(X-\mu_k)^T\Sigma_k^{-1}(X-\mu_k) + \log\pi_k$$

And the boundary $\delta_k(X) = \delta_\ell(X)$ becomes quadratic. This is known as *QDA*.

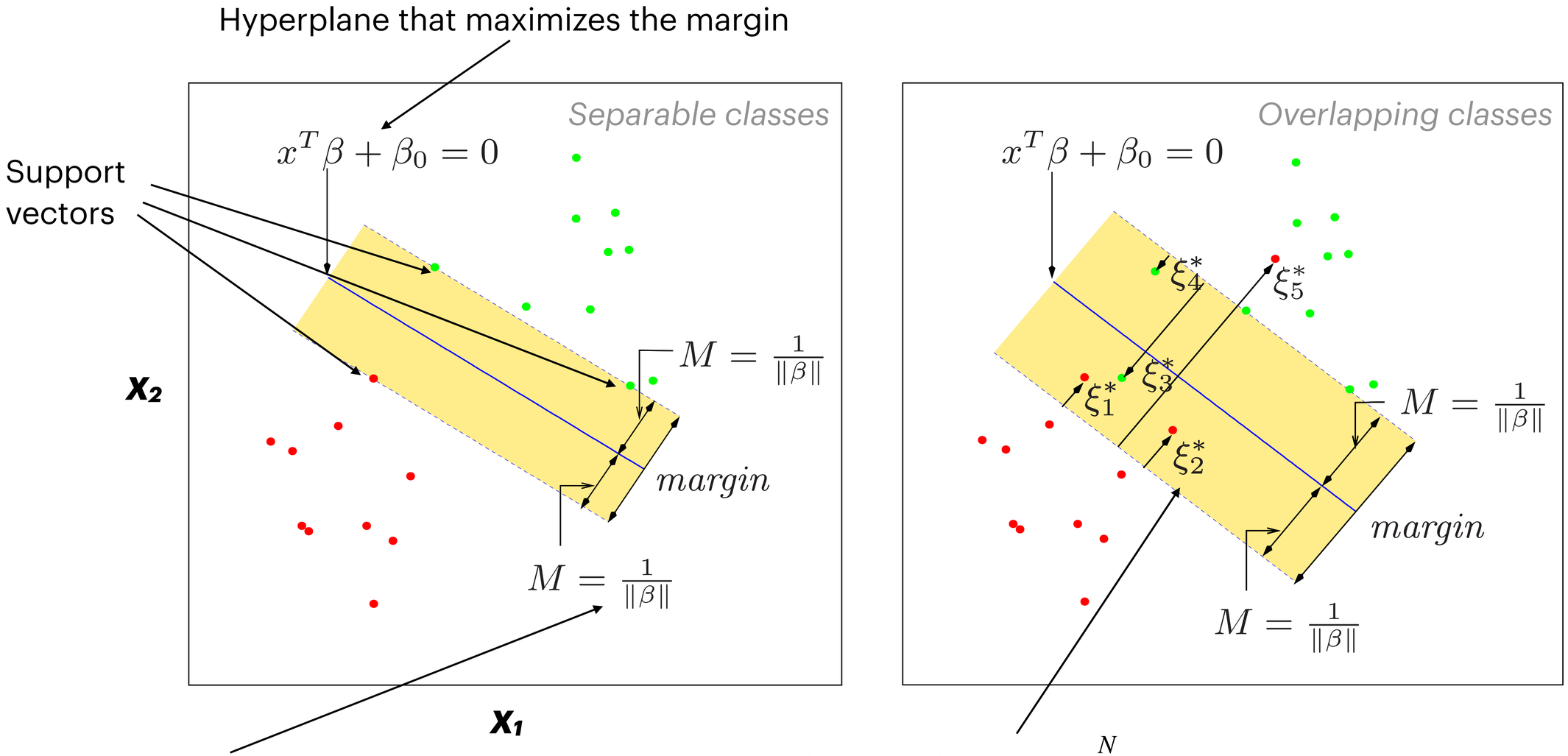
We can also use the same tricks from linear regression and do a non-linear approximation by running LDA on an expanded set of non-linear transformations of X

LDA on $[X_1, X_2]$ LDA on $[X_1, X_2, X_1X_2, X_1^2, X_2^2]$ QDA on $[X_1, X_2]$



Hastie et al., 2017

Support vector machines (SVMs): intro



Like before, β (or w) is perpendicular to the hyperplane

Margin is maximized up to a constraint of $\sum_{i=1}^{N} \xi_i \leq c$

Hastie et al., 2017

SVMs: formulation

Define a hyperplane

$$f(X) = Xw + w_0 = 0$$

The classification rule is

$$y = G(X) = \text{sign}[Xw + w_0] \ , \ y = \{-1,1\}$$

For the *separable case*, we want to maximize the margin (a.k.a. hard margin)

$$M = \frac{1}{||w||}$$

subject to $y_i(x_i w + w_0) \geq M \ , \ i = 1, 2, \ \ldots, N$

This is equivalent to minimizing $||w||^2/2$, which is a convex problem that can be solved with gradient descent. The solution to $f(X)$ can be shown to be a linear combination of the support vectors, i.e. data points that lie on the $f(X) \pm M$ hyperplanes

For the *non-separable* case, we pick additional slack variables

$$\xi = (\xi_1, \xi_2, \ldots, \xi_N)$$

And again maximize the margin (a.k.a. soft margin)

$$M = \frac{1}{||w||}$$

subject to $y_i(x_i w + w_0) \geq M - \xi_i \ , \ i = 1, 2, \ \ldots, N$

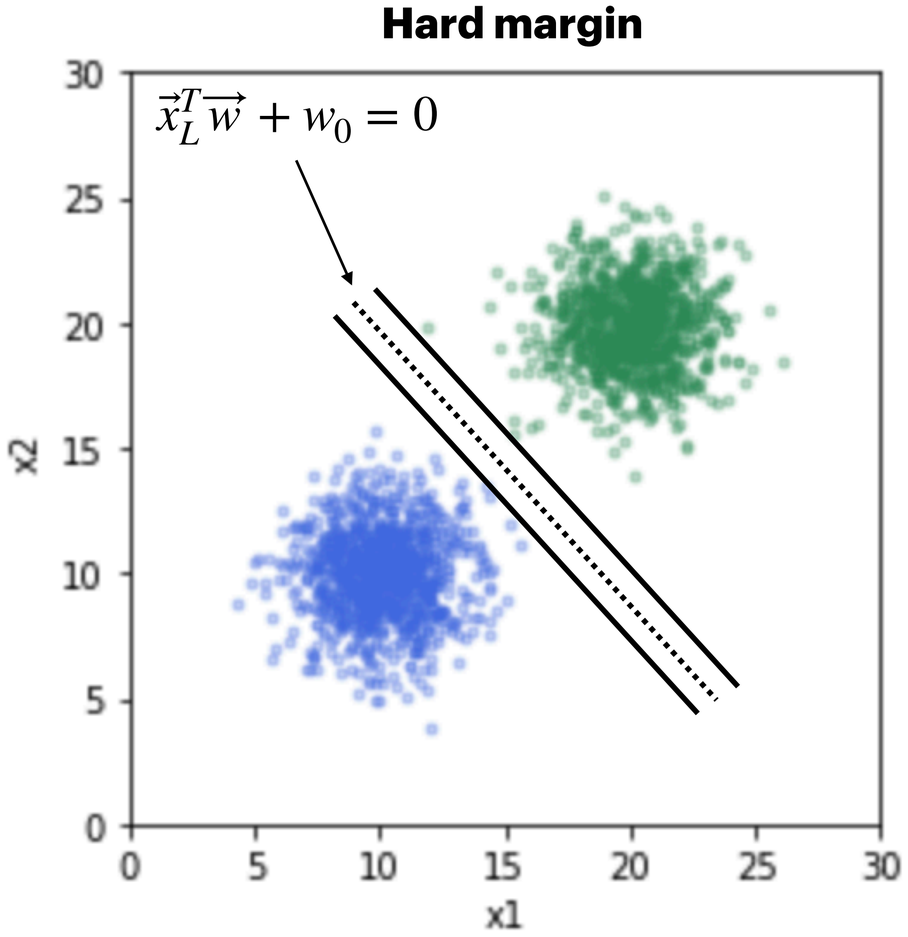
This can be posed as a convex problem with the following formulation

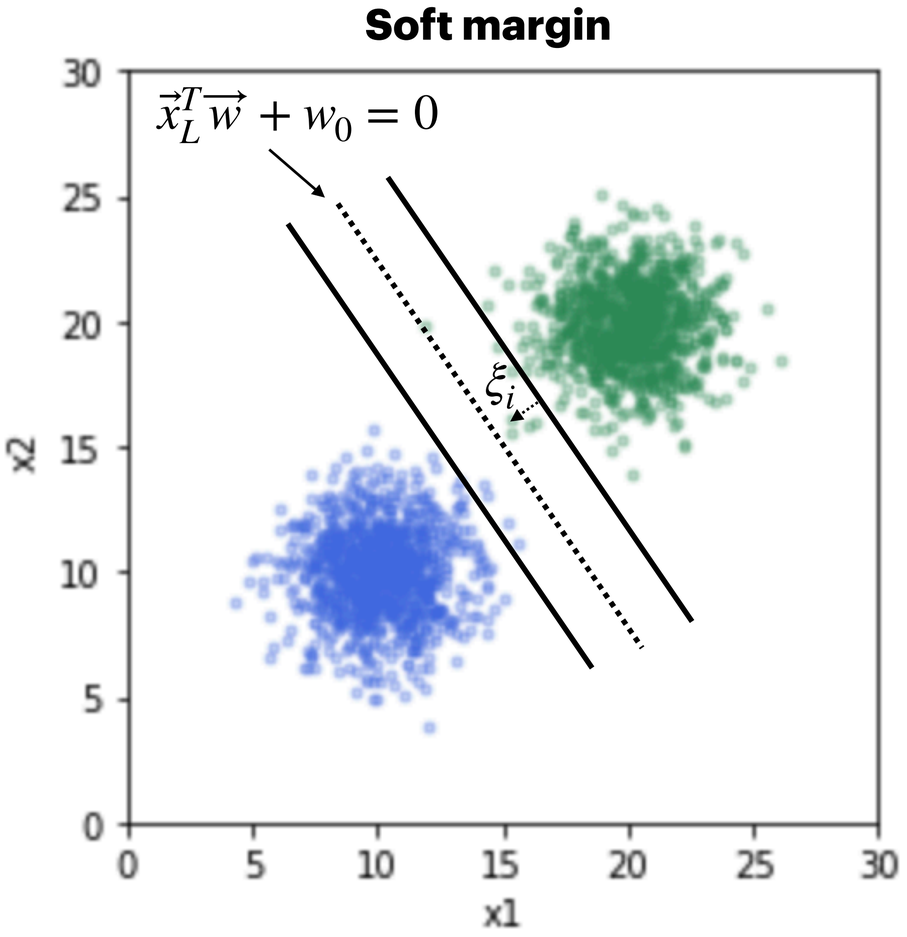
$$\min \ \frac{1}{2}||w||^2 + c\sum_{i=1}^{N} \xi_i$$

subject to $\begin{cases} y_i(x_i w + w_0) \geq 1 - \xi_i \ \forall x_i \\ \xi_i \geq 0 \end{cases}$

- Refer to Hastie chapters 4 and 12 for a full derivation of the cost function

Hard- vs. soft-margin SVM





Kernel SVMs

You may be starting to notice a pattern here, but as with other linear methods we can transform the feature space with non-linear functions, e.g. a set of basis functions

$$\Phi_n(X)\ , n = 1,\ 2,\ \ \ldots, N$$

Define a hyperplane

$$\hat{f}(X) = \Phi(X)w + w_0 = 0$$

The classification rule is

$$\hat{G}(X) = \text{sign}[f(\hat{X})]$$

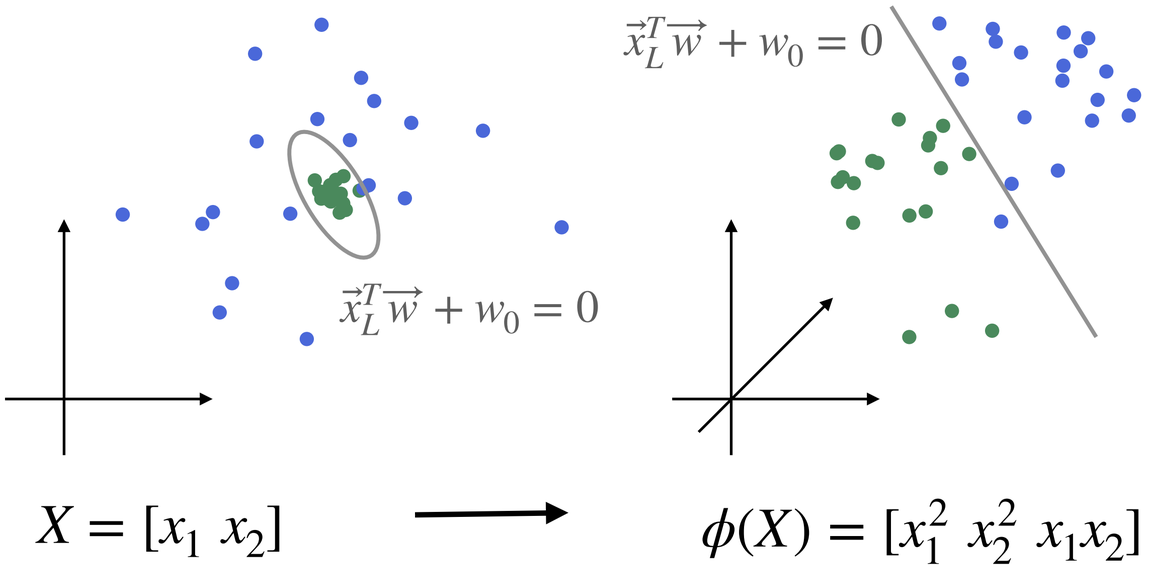
However, naïvely optimizing this set of transformed *X*, *[X,X']*, can be computationally very expensive, as it involves taking inner products X^TX' between all features

But we can find Kernel functions *K* s.t.

$$K(x, x') = \Phi(x)\Phi(x')$$

E.g., for a d-th degree polynomial

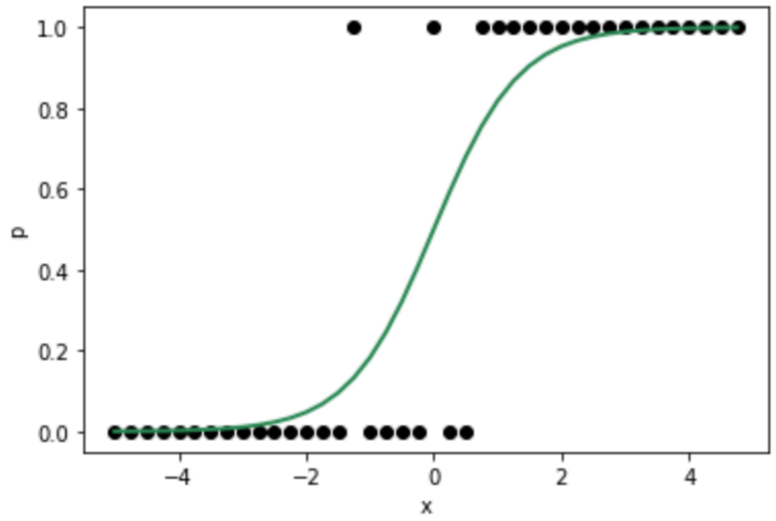
$$K(X, X') = (1 + X^T X')^d$$



Logistic regression

Logistic regression is a very common method to do binary classification in neuroscience, which we covered in our regression class

$$P(y = k \mid X) = \frac{1}{1 + e^{-Xw}}$$

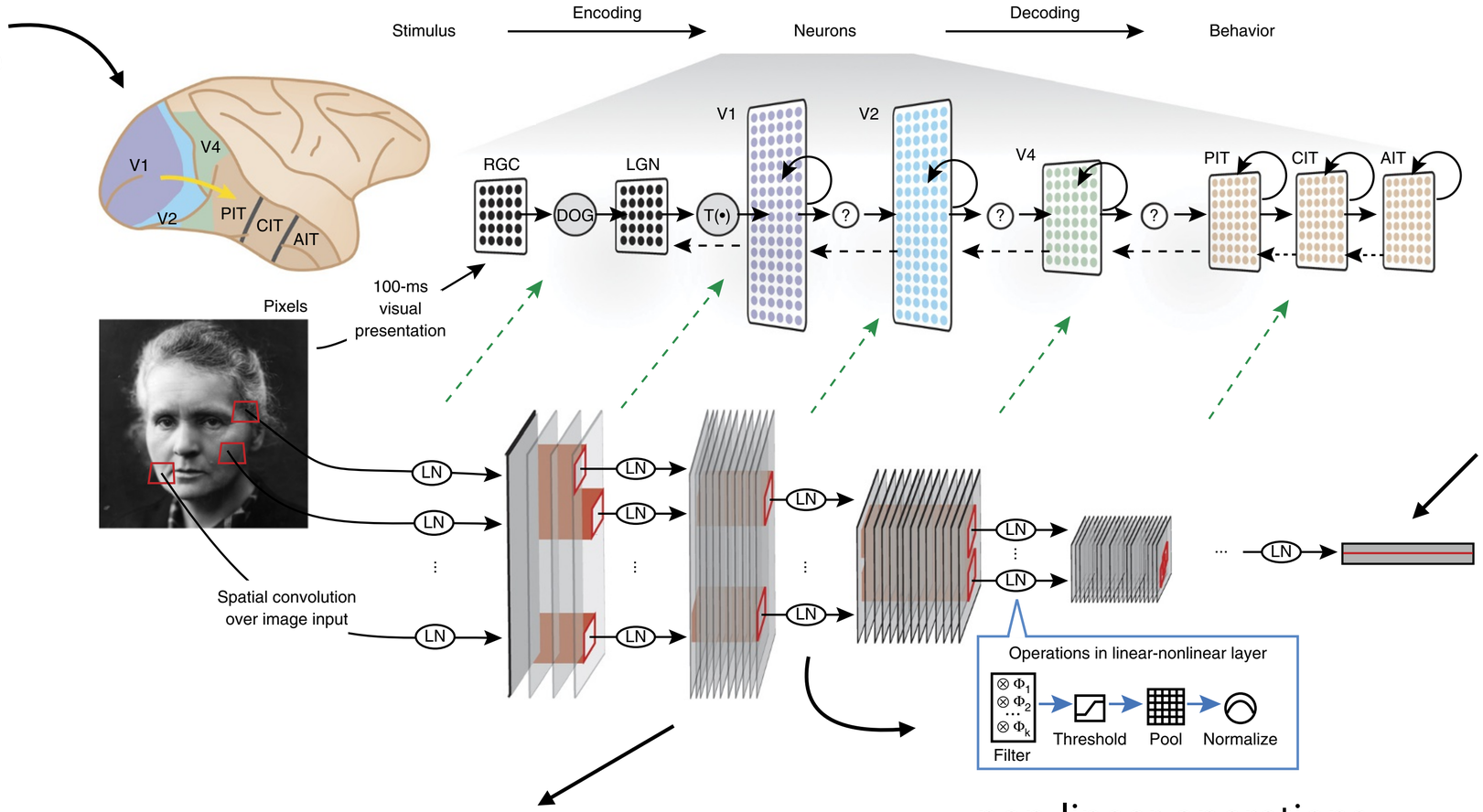


Unlike other classification methods, logistic regression by design assigns a probability that data point x_i belongs to class k, which can be useful

Some argue that this means it's not really classification, but that's beyond the point. The class prediction just includes an implicit threshold

Deep convolutional neural nets

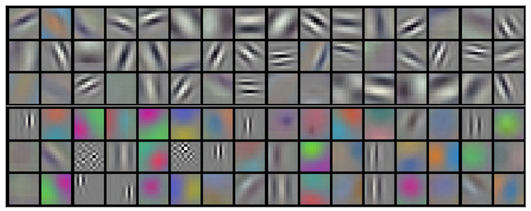
Inspired by the architecture of the visual system



Deep layers are usually fully recurrent

non-linear operations between layers

Successive convolutions with filter banks, typically resembling visual RFs



ImageNet
Khrizevsky et al., 2012, *NIPS* 2012

- Biggest success story is in computer vision (image classification)

Yamins & DiCarlo, 2016, *Nature Neurosci*

Brief overview of other classifiers

Perceptron

$$G(X) = \begin{cases} 1 \text{ if } Xw \geq 0 \\ 0 \text{ otherwise} \end{cases}$$

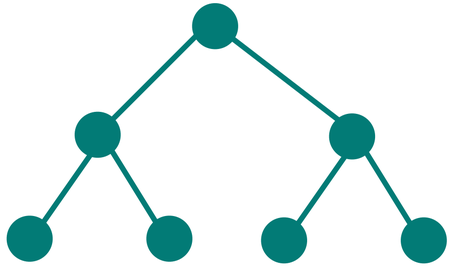
$$w^{t+1} = w^t + \eta(y_i - x_i w)x_i$$

K nearest neighbors

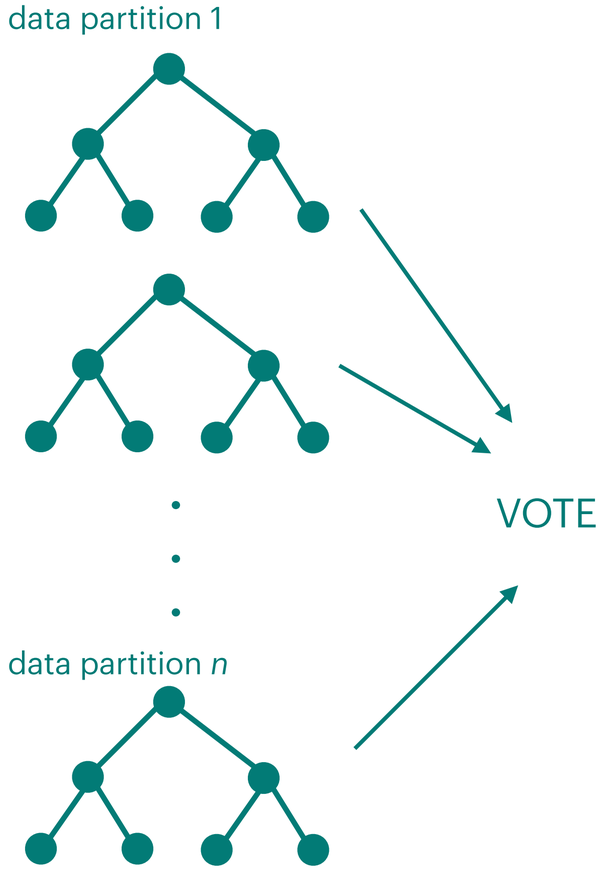
$G(x)$ = most frequent label within k nearest data points

Decision trees

Sequence of if-then's



Random forests



Multinomial logistic regression

generalization of logistic regression for K classes (bonus slide at the end of the deck)

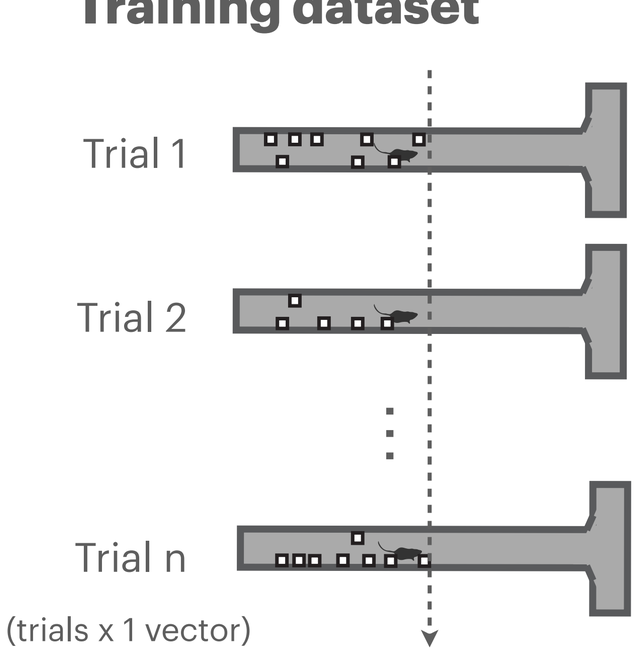
Decoding

Decoders: intro

- From a neuro perspective, decoders perform the reverse operation of an encoding model like a GLM
- I.e., a decoder asks how much information there is about some variable *x* (e.g. choice) in a response *y* (e.g. population neural activity), while an encoder ask what combination of features *x* (e.g., choice, movement, sensory stimuli) better explain the response *y* (neuronal activity)
- Complementary! Often helpful to do both.
- Decoders of categorical variables (e.g. choice) = classifiers. Very same methods as we just covered
- For continuous variables, it is common to just use regression or Bayesian decoders

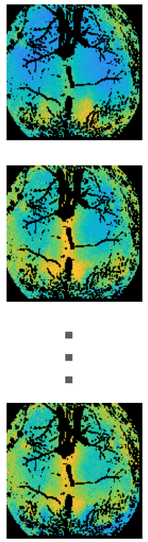
Brief example of linear regression for decoding

Training dataset



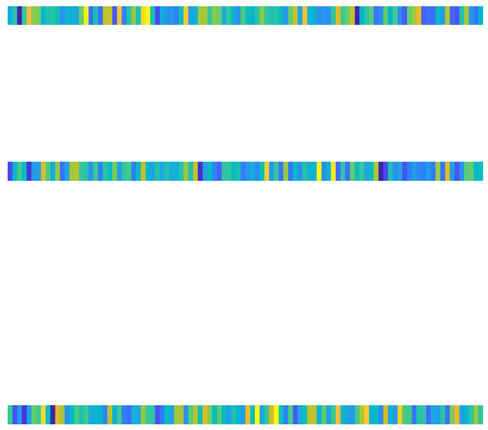
$E(y) = \Delta$
(cum. #R – #L towers)

Δ = –2
Δ = +3
Δ = +6



=

Vectorize pixels
(trials x pixels matrix)



*

Weights
(pixels x 1 vector)

$$E(y) = \sum_{i=1}^{Npixels} \Delta F/F_i(y)\,\boxed{\beta_i}$$

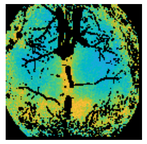
L2 penalty

$$+\boxed{\lambda}\,||\vec{\beta}||$$

Model parameters

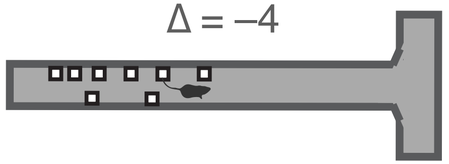
Test dataset

$$\widehat{E(y)} = \;*\; \vec{\beta}$$



How similar to actual evidence?
(linear correlation)

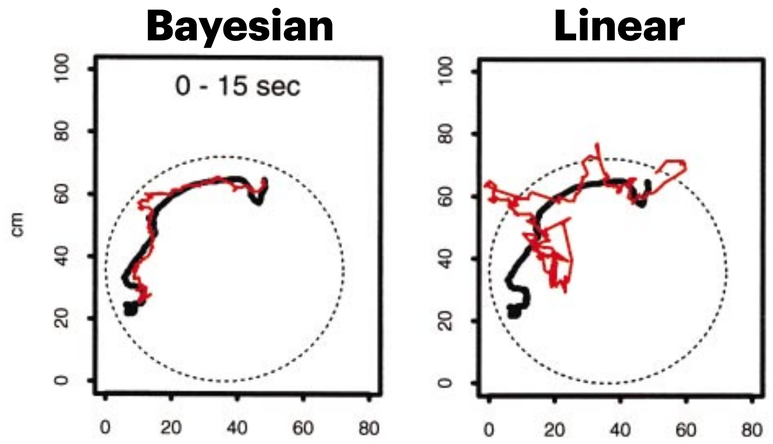
Δ = –4



Pinto et al., 2019, *Neuron*

Bayesian (population) decoders

Linear decoders are highly interpretable but may not yield the best estimates, e.g. because neural activity is messy and influenced by multiple variables beyond the one you are trying to decode



Brown et al., 1998, *J Neurosci*

Say you want to decode a continuous or discrete variable *C* and you have measured a set of other variables $\overrightarrow{C'} = (C'_1, C'_2, \ldots, C'_v)$

Say you have *N* neurons (pixel, voxel, etc). For each neuron, you first estimate how activity is influenced by each of these variables, typically using a (G)LM

Now, using the GLM weights, you can estimate the conditional probability of a response, *r*, given different values of *C*, by simply generating predictions. Assuming responses are conditionally independent,

$$P(\vec{r} \mid C, \overrightarrow{C'}) = \prod_{i=1}^{N} P(\overrightarrow{r_i} \mid C, \overrightarrow{C'})$$

Applying Bayes rule,

$$P_C = P(C \mid \vec{r}, \overrightarrow{C'}) = \frac{P(\vec{r} \mid C, \overrightarrow{C'})P(C \mid \overrightarrow{C'})}{P(\vec{r} \mid \overrightarrow{C'})}$$

The denominator is often ignored since P_C is still proportional to the numerator, and it may be reasonable to assume that *C* and *C′* are independent, s.t. *P*(*C*|*C′*) = P(*C*). Otherwise *P*(*C*|*C′*) can be estimated from the data.

For the continuous case, the decoded value can be taken as $\hat{C} = E[P_C] = \frac{\sum P_C C}{\sum P_C}$

see also, e.g., Pillow et al., 2008, Nature; Minderer et al., 2019, Neuron

Clustering

Unsupervised learning

Except for PCA, everything we've covered so far is a type of *supervised learning*: given set of features or predictors, *X*, and a set of labels or responses, *y*, learn the mapping function by minimizing the error between *y* and $\hat{y}$

Discrete data: classification
Continuous data: regression

In *unsupervised learning*, we try to directly estimate properties of *X* without a teacher or supervisor (*y*).

Unfortunately, that also means it's harder to judge success.

Clustering is one of the most used forms of unsupervised learning. The goal is to find subgroups (clusters) of datapoints within X s.t. they are more similar to each other within than across clusters.

Clustering algorithms always start with some dissimilarity or distance *N* x *N* matrix, which encodes pairwise distances between all data points

The most common ones are squared distance in *p* dimensions:

$$d(x_i, x_i') = \sum_{j=1}^{p} (x_{ij} - x_{ij}')^2 = ||x_i - x_i'||^2$$

Or its square root, the Euclidean distance

$$d(x_i, x_i') = \sqrt{||x_i - x_i'||^2}$$

Other metrics include absolute distance and 1 – corr(x,x')

Note that we can also differently weight different dimensions of the data when computing dissimilarity, if we think some features are more informative than others

K-means: formulation

K-means uses the squared distance

$$d(x_i, x_i') = ||x_i - x_i'||^2$$

We want to minimize the within-cluster scatter *W*(*C*), for a pre-selected number of clusters, *K*

$$W(C) = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}^{N_k}\sum_{C(i')=k}^{N_k} ||x_i - x_i'||^2$$

Which is equivalent to minimizing the distance between each point and the cluster average μ

$$W(C) = \sum_{k=1}^{K} N_k \sum_{C(i)=k}^{N_k} ||x_i - \mu_k||^2$$

(you can check by plugging in the average)

$$\mu = \frac{x_i + x_i'}{2}$$

Quick aside. The total distance between points is the sum of within- and between-cluster distances

$$T = W(C) + B(C)$$

$$B(C) = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}^{N_k}\sum_{C(i')\neq k}^{N_k} ||x_i - x_i'||^2$$

Thus, minimizing *W* is equivalent to maximizing *B*

$$W(C) = T - B(C)$$

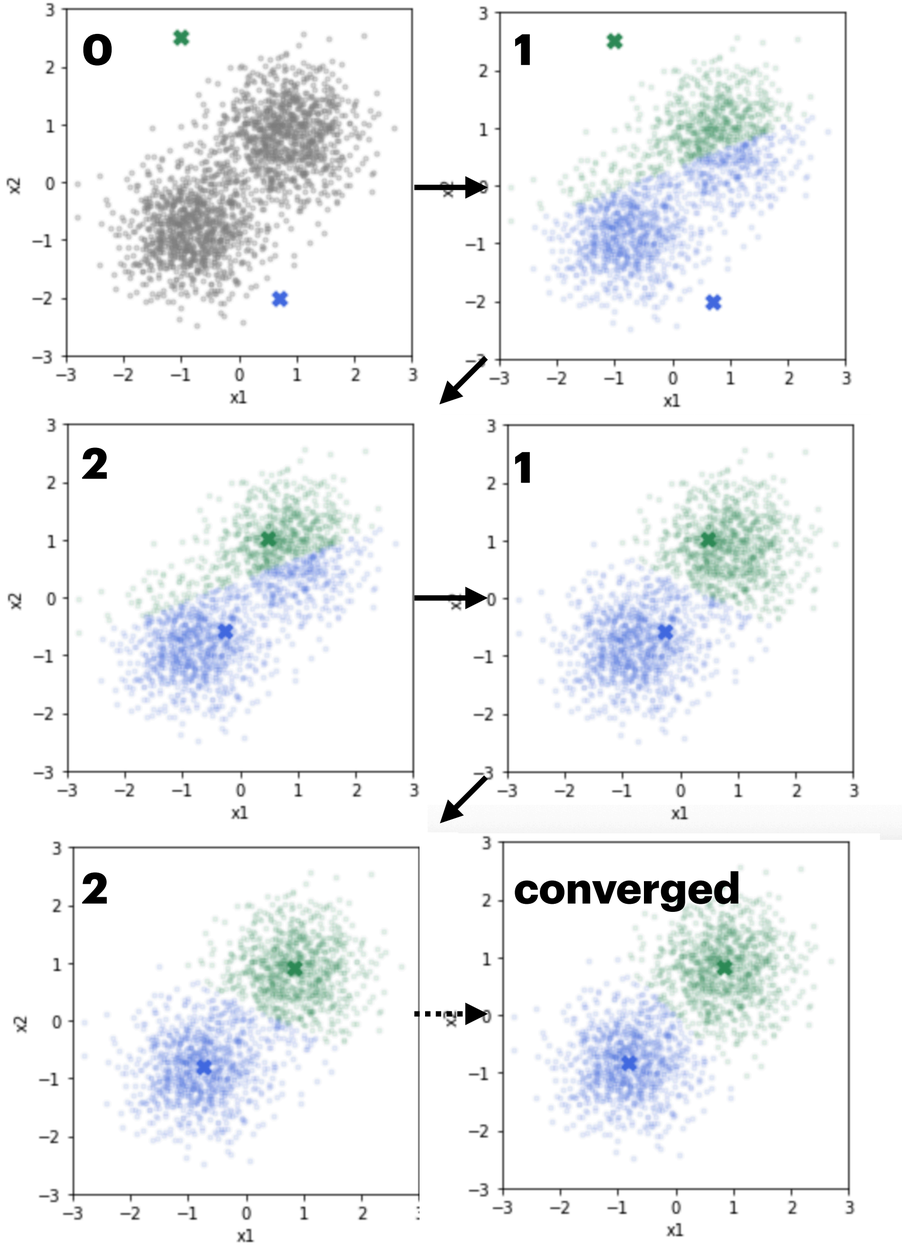
K-means: algorithm

0. Initialize μ's

Iterate until assignments no longer change:

1. Fix μ's. Minimize cost w.r.t. to cluster assignment *C(i)*. I.e., assign each data point to its nearest μ.
2. Fix assignment *C(i)*. Minimize cost w.r.t. to μ. I.e., compute new μ's.

- Guaranteed to converge, but could be local minimum
- Best to use different initializations and pick the best solution
- Note that the cost function is sensitive to the scale of the data. It is usually best to z-score
- Also note that the squared distance will give more weight to outliers, which may or may not be a good thing



The Expectation-**M**aximization **algorithm**

The K-means algorithm is closely related to E-M algorithms. They are used in a number of MLE optimization problems when it is not possible to compute analytical gradients for the cost function, e.g. because there are latent variables, like *C(i)*.

The likelihood function of the data *X* can be written as a function of the parameters θ and latent variables *Z*

$$L(\theta; X, Z) = \int p(X, Z \mid \theta) dZ = \int p(X \mid Z, \theta) p(Z \mid \theta) dZ$$

A bit of a catch-22. If we know θ, we can find *Z* by maximizing the likelihood over all possible values of *Z*.

If we know Z, we can find θ by empirically estimating them by e.g. grouping them according to the latents in Z and then computing some function, like the mean of the data.

But we can solve it by iterating over these *Z* to θ steps

1. **E**xpectation step. Estimate the expected value of the likelihood of given the current conditional distribution of *Z* on *X* and current θ estimate $\theta^{(t)}$.

   $$Q(\theta \mid \theta^{(t)}) = E[L(\theta; X, Z) \mid X, \theta^{(t)}]$$

   ~ in *K*-means: given $\mu^{(t)}$, assign *C(i)*

2. **M**aximization step. Maximize Q over θ to update its value.

   $$\theta^{(t+1)} = \arg\max_{\theta} \; Q(\theta \mid \theta^{(t)})$$

   ~ in *K*-means: given new *C(i)*, recompute μ

Gaussian mixture models (GMM)

K-means has limitations:

- hard cluster assignments make it unstable (a small perturbation to a datapoint can change *C(i)*)
- Assumes fully separable, spherical clusters of equal probability

GMMs solve this via a probabilistic picture: assume the data come from *K* multivariate Gaussian distributions that can be overlapping and have any covariance structure, each with an overall probability π

$$P(x) = \sum_{j=1}^{K} \pi_j \mathcal{N}(x \mid \mu_j, \Sigma_j) \ , \ \sum_{j=1}^{K} \pi_j = 1$$

The parameters are

$$\theta = (\pi_1, \ldots, \pi_j, \mu_1, \ldots, \mu_j, \Sigma_1, \ldots, \Sigma_j)$$

The loss function is the negative log likelihood

$$-\ell(X \mid \theta) = -\sum_{i=1}^{N} \log \left\{ \sum_{j=1}^{K} \pi_j \mathcal{N}(x_i \mid \mu_j, \Sigma_j) \right\}$$

The sum inside the log makes it really difficult to analytically compute ℓ

Now let's assign to each observation x_i an associated latent variable z_i that is categorical and indicates which cluster x_i is affiliated with cluster *j*

$$z_i = (z_{i1}, z_{i2}, \ldots, z_{iK}) \ , \ z_{ij} = \{0,1\}$$

The loss function now becomes

$$-\ell(X, Z \mid \theta) = -\sum_{i=1}^{N} \sum_{j=1}^{K} z_{ij} [\log \pi_j + \log \mathcal{N}(x_i \mid \mu_j, \Sigma_j)]$$

(See *L* in previous slide)

EM algorithm for GMMs

0. Initialize θ

$$\theta^{(t)} = \hat{\theta} = (\hat{\pi}_1, \ldots, \hat{\pi}_j, \hat{\mu}_1, \ldots, \hat{\mu}_j, \hat{\Sigma}_1, \ldots, \hat{\Sigma}_j)$$

1. **E**xpectation step. Compute the expected value of the the likelihood that latent variables z_i belong to cluster j (a.k.a. the responsibilities)

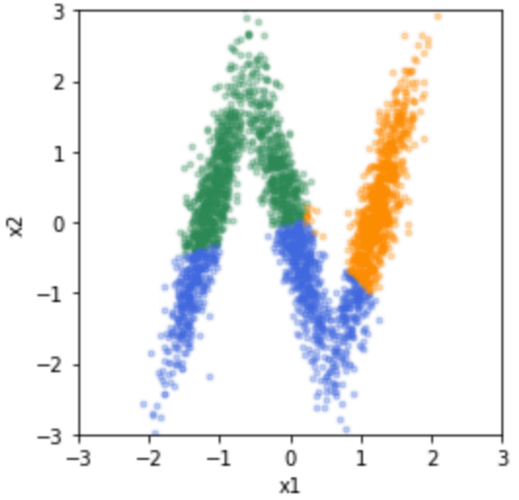
$$r_{ij} = E[z_{ij}] = \frac{\hat{\pi}_j \ \mathcal{N}(x_i \,|\, \hat{\mu}_j, \hat{\Sigma}_j)}{\sum_{j=1}^{K} \hat{\pi}_j \mathcal{N}(x_i \,|\, \hat{\mu}_j, \hat{\Sigma}_j)}$$

2. **M**aximization step. Maximize the likelihood by re-estimating θ given new responsibilities

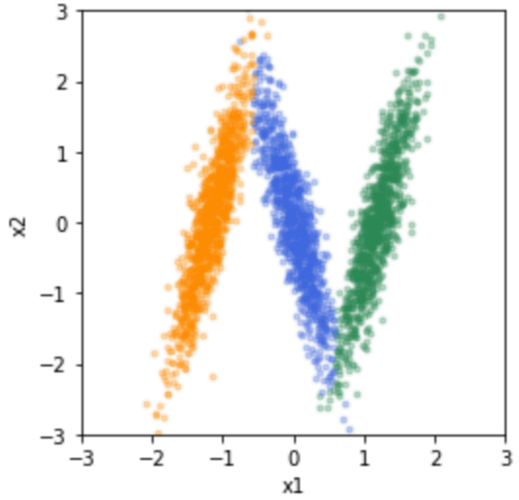
$$\theta_j^{(t+1)} = \left(\hat{\pi}_j = \frac{1}{N} \sum_{i=1}^{N} r_{ij} \ , \ \hat{\mu}_j = \frac{\sum_{i=1}^{N} r_{ij} x_i}{\sum_{i=1}^{N} r_{ij}} \ , \ \hat{\Sigma}_k = \frac{\sum_{i=1}^{N} r_{ik} (x_i - \hat{\mu}_j)(x_i - \hat{\mu}_j)^T}{\sum_{i=1}^{N} r_{ij}} \right)$$

3. Iterate 1 and 2 until convergence

K-means



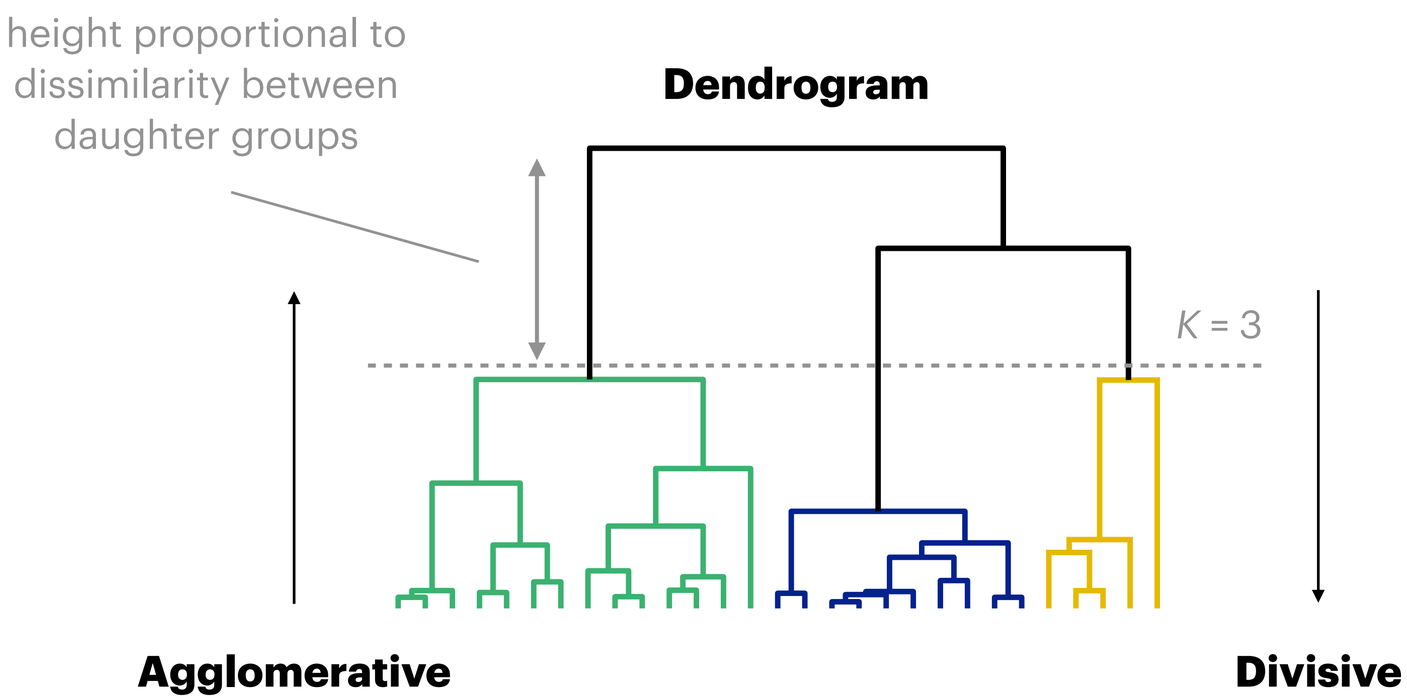
GMM



Hierarchical clustering

In *K*-means and GMMs, changing *K* leads to arbitrary changes in cluster assignment

Hierarchical clustering splits the data in order of dissimilarity, creating a highly interpretable hierarchical structure with *N* – 1 levels (from whole dataset to individual data points) that is invariant to *K*



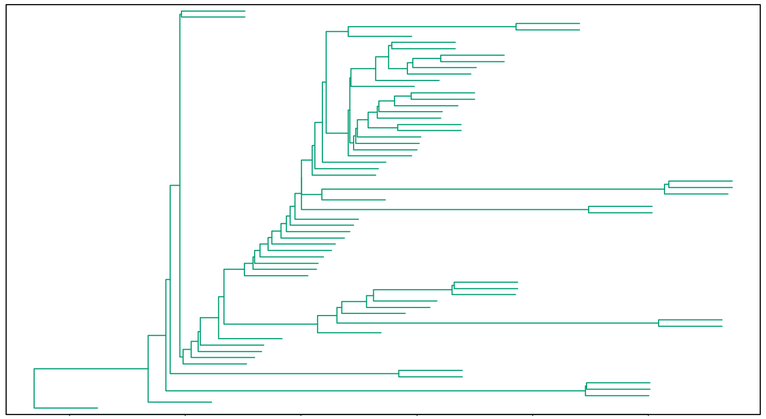
Recursively groups pairs of data points based on dissimilarity

Recursively splits data points based on dissimilarity from data cluster

Agglomerative clustering methods

Single linkage
distance between two groups *G* and *H* is the distance between least dissimilar pair of datapoints

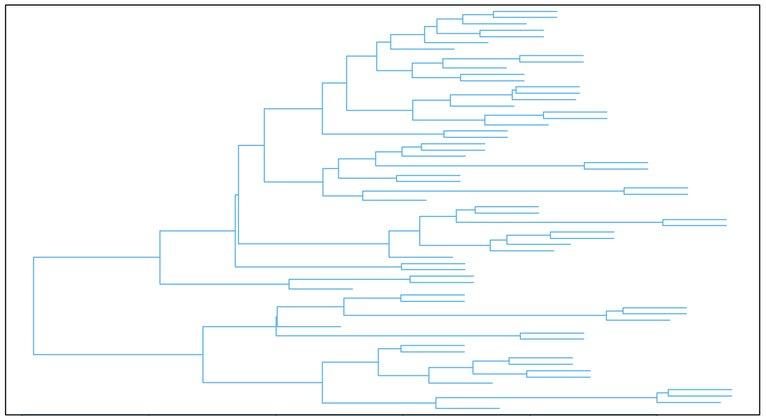
$$d_{SL}(G,H) = \min_{i \in G, j \in H} d_{ij}$$



Tends to yield extended clusters, i.e. large within-cluster distances

Complete linkage
distance between most dissimilar pair of datapoints

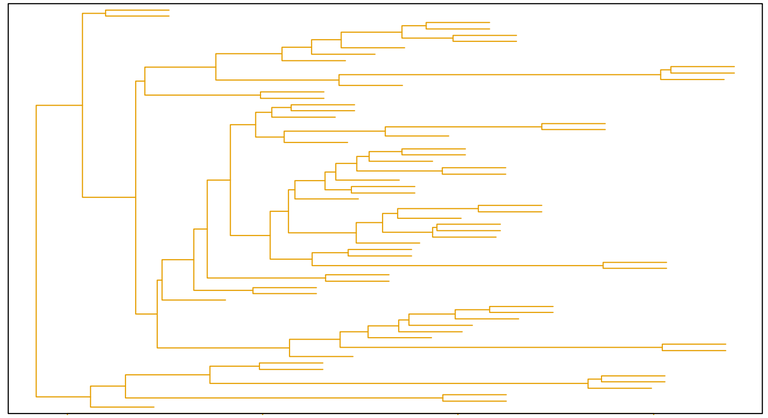
$$d_{CL}(G,H) = \max_{i \in G, j \in H} d_{ij}$$



Tends to yield rounder, compact clusters, but members of a cluster can sometimes be closer to other clusters than its own

Average linkage
average distance between all pairs of datapoints

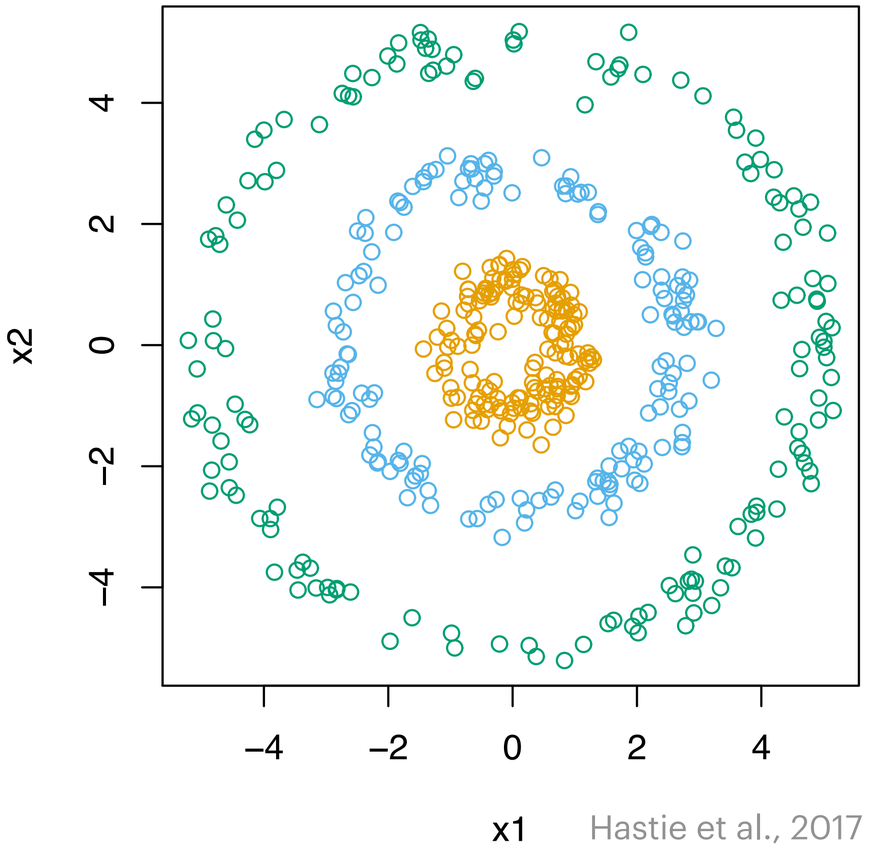
$$d_{CL}(G,H) = \frac{1}{N_G N_H} \sum_{i \in G} \sum_{j \in H} d_{ij}$$



Good compromise between the two, but not invariant to monotone transformations of the data (e.g. scaling)

Hastie et al., 2017

Primer: Spectral clustering



Hastie et al., 2017

Previous methods do not work well for non-convex problems

Spectral clustering is a graph-based generalization of the previous methods.

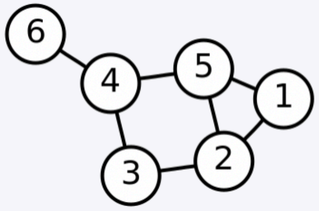
Preview of the algorithm

1. Build a similarity matrix *S* between data points, e.g.

$$d_{ii'} = ||x_i - x_i'||^2 \qquad s_{ii'} = e^{-d_{ii'}/c}, s_{ii'} \geq 0$$

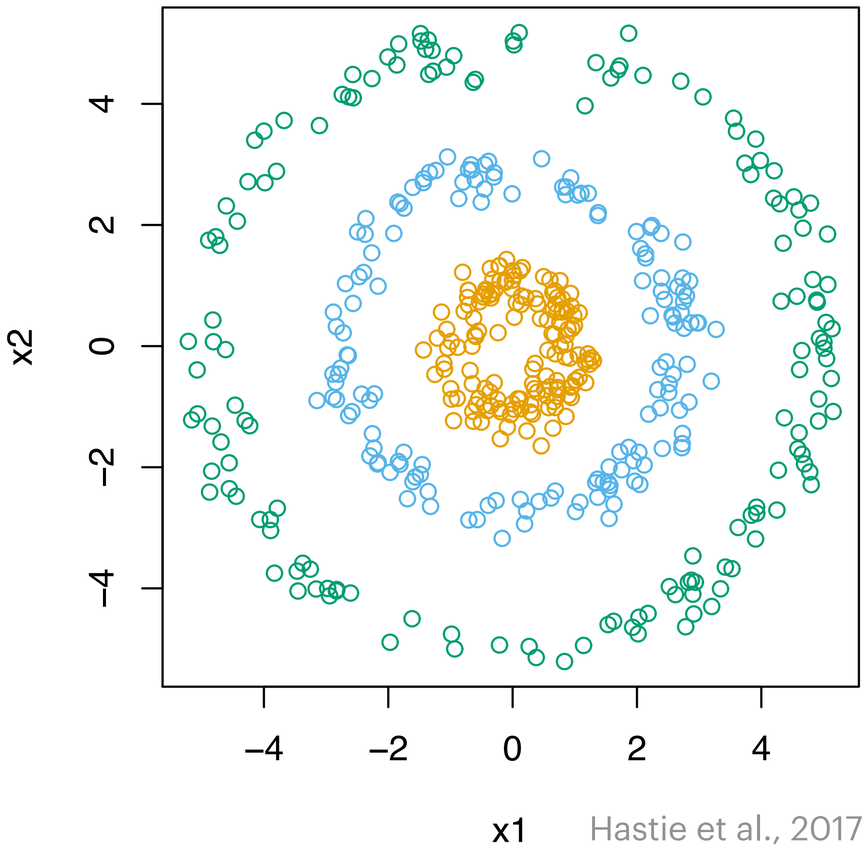
2. Build an undirected similarity graph matrix of *S*, *A*. This is the adjacency matrix, where the vertices represent data points, and vertex pairs are connected by an edge if $s_{ii'}$ exceeds some threshold. Edges are weighted by $s_{ii'}$.

3. Compute the Laplacian of the graph as *L* = *G* – *A*. *G* is a diagonal matrix with the degree of each node (the sum of its connection weights, in *A*. L is a compact description of the graph

Labelled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$

wikipedia

Primer: Spectral clustering



Hastie et al., 2017

Previous methods do not work well for non-convex problems

Spectral clustering is a graph-based generalization of the previous methods.

Preview of the algorithm

1. Build a similarity matrix *S* between data points, e.g.

$$d_{ii'} = ||x_i - x_i'||^2 \qquad s_{ii'} = e^{-d_{ii'}/c}, s_{ii'} \geq 0$$

2. Build an undirected similarity graph matrix of *S*, *A*. This is the adjacency matrix, where the vertices represent data points, and vertex pairs are connected by an edge if $s_{ii'}$ exceeds some threshold. Edges are weighted by $s_{ii'}$.

3. Compute the Laplacian of the graph as $L = G - A$. *G* is a diagonal matrix with the degree of each node (the sum of its connection weights, in *A*. L is a compact description of the graph

4. Find the Z_{Nxm} eigenvectors corresponding to the *m* smallest eigenvectors of *L* (thus the name *spectral*).

- This is because, in the ideal case, if a graph has *m* groups of connected components, *L* has *m* eigenvalues = 0.

5. Use a standard clustering methods, e.g. *K*-means, to cluster the rows of *Z*.

Picking *K*

- The trickiest part of clustering is picking the right *K*
- Particularly so for algorithms where the solution is not invariant to the choice of *K*
- The solution is typically to perform clustering for different *K*'s, and come up with some metric that is maximized for a particular *K*
- In practice, these metrics often still increase monotonically with *K*, so we instead pick an “elbow” in the function (as we do with dimensionality reduction)
- Some examples:

Cross-validation

Partition the data into two sets. For different *K*’s, estimate the parameters on one set (e.g., μ, σ) and then compute the loss (cost) function on the other set. Pick *K* that minimizes the loss.

- The others all want to minimize within-cluster dispersion

$$W_K(C) = \sum_{k=1}^{K} N_k \sum_{C(i)=k} ||x_i - \mu_k||^2$$

Elbow method

Plot *W*(*C*) as a function of K, pick “elbow” in function

Gap statistic

Maximize the distance between *W*(*C*) for the data and a null uniform distribution (*K* = 1) [min(data) max(data)], e.g. by averaging over Monte Carlo simulations

$$Gap(K) = \log W_k^{null} - \log W_k(C)$$

Calinski-Harabasz Index

Maximize the ratio between the between-cluster separation *B*(*C*) and *W*(*C*)

$$CH(K) = \frac{B_K(C)/(k-1)}{W_K(C)/(N-k)}$$

$$B_K(C) = \sum_{k=1}^{K} ||\mu_k - \mu||^2 \ , \ \mu = <\mu_k>$$

Summary

- Classifiers try to find the best separating hyperplane between categories
- Logistic regression and SVMs are common binary classifiers in neuroscience
- Kernel SVMs can deal with non-linear data
- Decoding is neuro jargon for predicting behavior / sensory stimuli based on (population) neural activity. Binary = classifiers or Bayesian; continuous = regression or Bayesian
- Clustering is a form of unsupervised learning in which we try to optimally group data into categories
- K-means finds spherical clusters with a form of EM algorithm; GMM are a probabilistic generalization that also take into account covariance matrices
- Hierarchical clustering is based on the distance between each data point; interpretable and invariant to choice of K.
- Spectral clustering is good for non-convex data but finicky
- Hard to choose K, requires both metrics and common sense

Suggested reading

- Hastie, Tibshirani & Friedman (2017). The Elements of Statistical Learning, 2nd edition. Chapters 4, 12, 14.
- Brown et al. (1998). A Statistical Paradigm for Neural Spike Train Decoding Applied to Position Prediction from Ensemble Firing Patterns of Rat Hippocampal Place Cells. *J Neurosci* 18:7411-7425
- Journal club papers on canvas

Bonus slide: Multinomial logistic regression

1. Multinomial logistic regression is a generalization for *K* classes, s.t. the probably of the *ith* observation belong to category *k i*s given by a function of a set regression coefficients *w*

$$f(k, i) = \vec{x}_i w \ , \ k \in G$$

2. An intuitive way to think about it is that we perform a set of *K – 1* binary classifications w.r.t. to a reference class *K*

$$\log\left[\frac{P(y = k|X)}{P(y = K|X)}\right] = Xw_k \ , \ k < K \ , \ \sum_{j=1}^{K} P(j) = 1$$

$$exp()$$

$$P(y = k|X) = P(y = K|X)e^{Xw_k}$$

3. Because probabilities add to 1

$$P(y = K|X) = 1 - \sum_{j=1}^{K-1} P(y = K|X)e^{Xw_j}$$

$$= 1 - P(y = K|X) \sum_{j=1}^{K-1} e^{Xw_j}$$

$$= \frac{1}{1 + \sum_{j=1}^{K-1} e^{Xw_j}}$$

Thus, $P(y = k|X) = \frac{e^{Xw_k}}{1 + \sum_{j=1}^{K-1} e^{Xw_j}}$

4. The negative log likelihood is known as cross-entropy and is differentiable, so we can fit with gradient descent

$$-\ell = -\sum_{i=1}^{n}\sum_{j=1}^{K} \delta_{j,y_i} \log(P(Y_i = j)) \ , \ \delta j, y_i = \begin{cases} 1 \text{ if } j = y_i \\ 0 \text{ otherwise} \end{cases}$$

The end