Tools and techniques for model building

A presentation by Russell Milne

Before I begin, a few notes on applied math

- By definition, applied math is applied, which means that it is used to solve real-world problems
- Because there's lots of data now, a lot of contemporary mathematical applications involve working with data
- If you're using math in the real world, it helps to have a large toolbox and know the right tool for the job
- Sufficiently applied math is often indistinguishable from engineering or computer science
- New math is constantly being developed to fit needs. Applied math is much quicker in this regard than pure math

Necessity is the mother of invention

- Wavelets: developed during the ‘80s (Morlet, Daubechies, Cazelles, etc.)
- Lasso method: Santosa and Symes 1986, Tibshirani 1996
- Boosting: Kearns and Valiant 1989, Freund and Schapire 1995
- Local Getis-Ord statistic: Getis and Ord 1992, 1995
- Random forest: Heath et al. 1993, Ho 1995
- Synthetic Minority Oversampling TEchnique: Chawla et al. 2002
- t-SNE: 2024 Nobel winner Geoffrey Hinton and collaborators (2002, 2008)
- U-Net: Ronneberger et al. 2015
- SINDy: University of Washington team 2016; refinement ongoing
- Shapley Additive exPlanations: Lundberg and Lee 2017
- Time series counterfactuals: development ongoing (currently active)

A quick background on Python, in case you're not already familiar with it

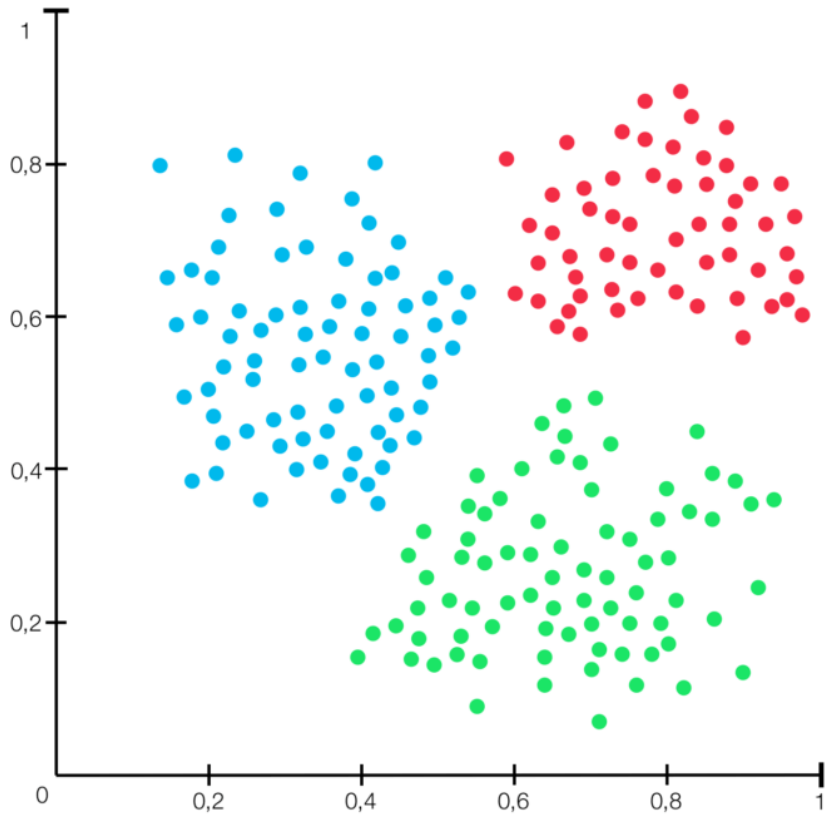
- Programming language often used for ML and data science applications
- Created by CS people rather than mathematicians; indices start with 0
- Free to use; Jupyter Lab is a good GUI implementation
- Many, many useful packages (e.g. SciPy, Pandas, Matplotlib)

What if you have a lot of different objects and want to find out which ones are similar?

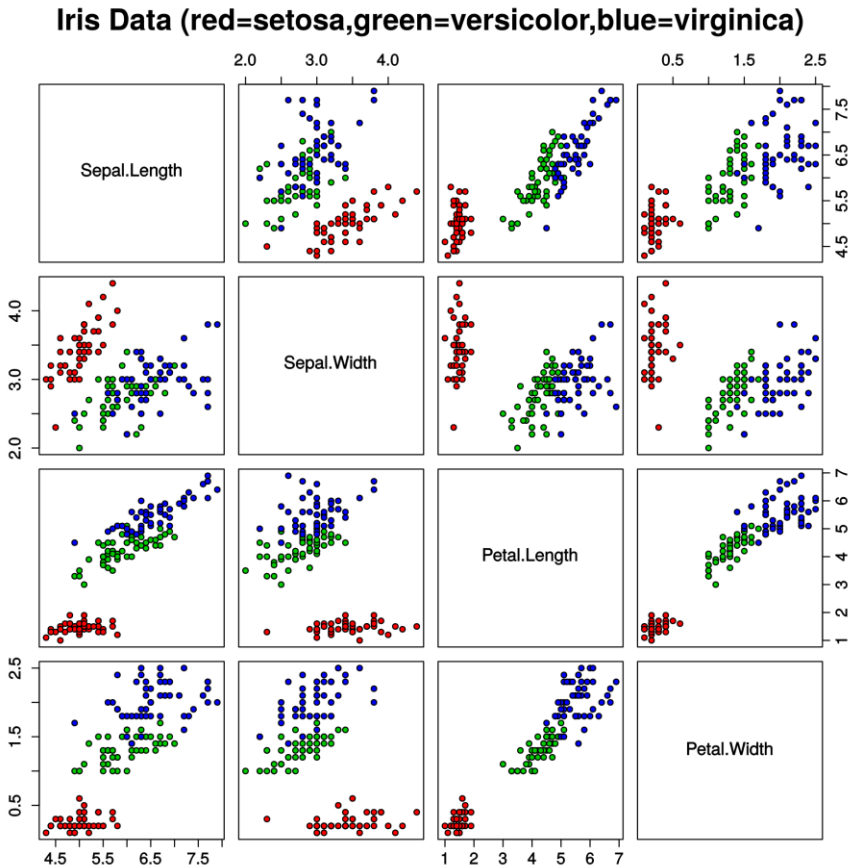
Single-linkage clustering, k-means, expectation-maximization

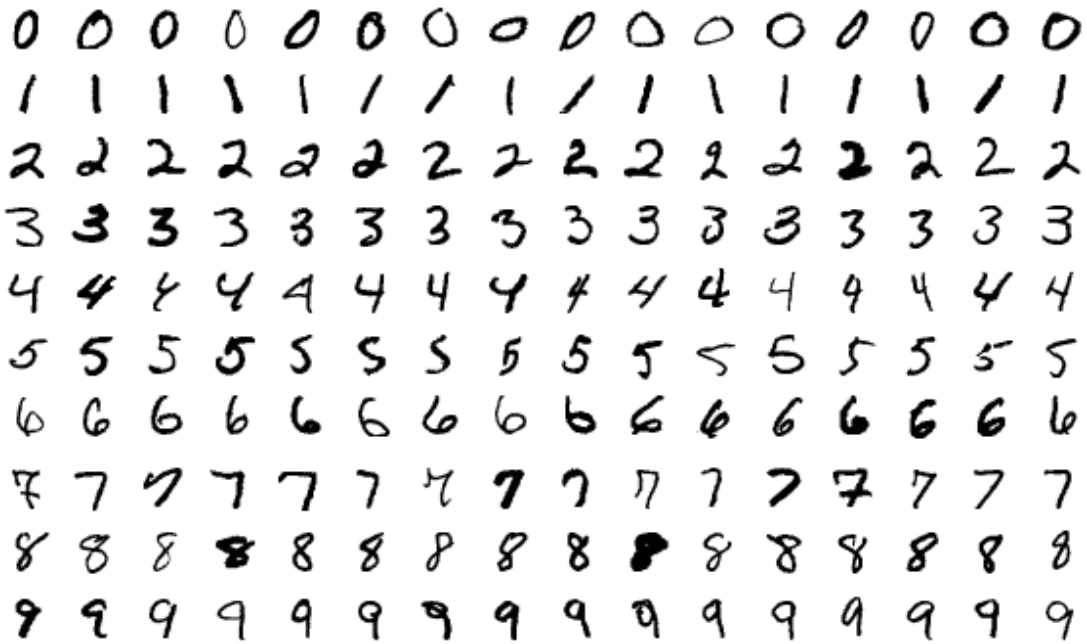
What is cluster analysis?

- Essentially a way of grouping similar objects together
- Generally works by initializing some number of clusters and then assigning objects to one (or more) of the clusters
- Good if you have a lot of objects and manually doing pairwise comparisons would be a hassle



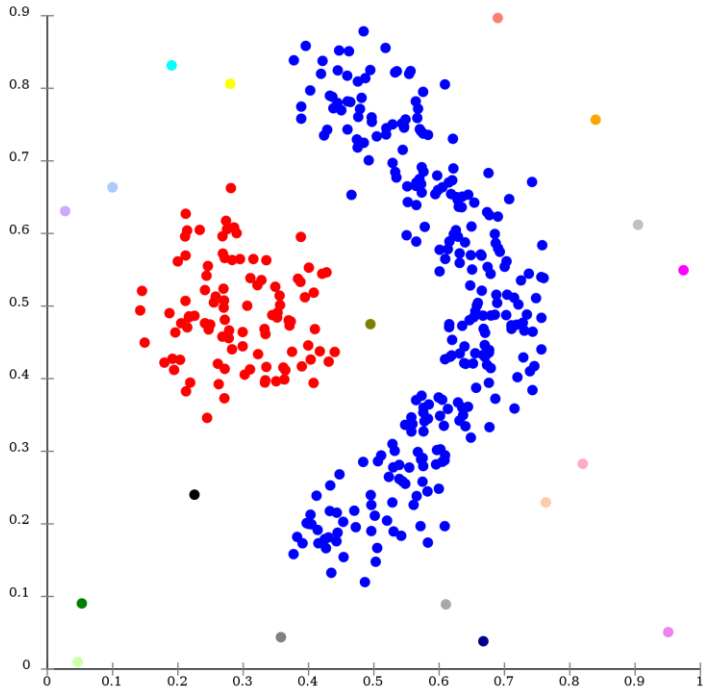
FYI: two classic datasets (Fisher Iris and MNIST) that will come up later





Single-linkage clustering

- Simple, intuitive clustering method based on repeatedly joining nearby clusters together
- Hierarchical clustering algorithm, meaning that you can use it to create dendrograms
- Sometimes called the “friends of friends” algorithm, in reference to its method of joining clusters



Simple SLC algorithm: setup

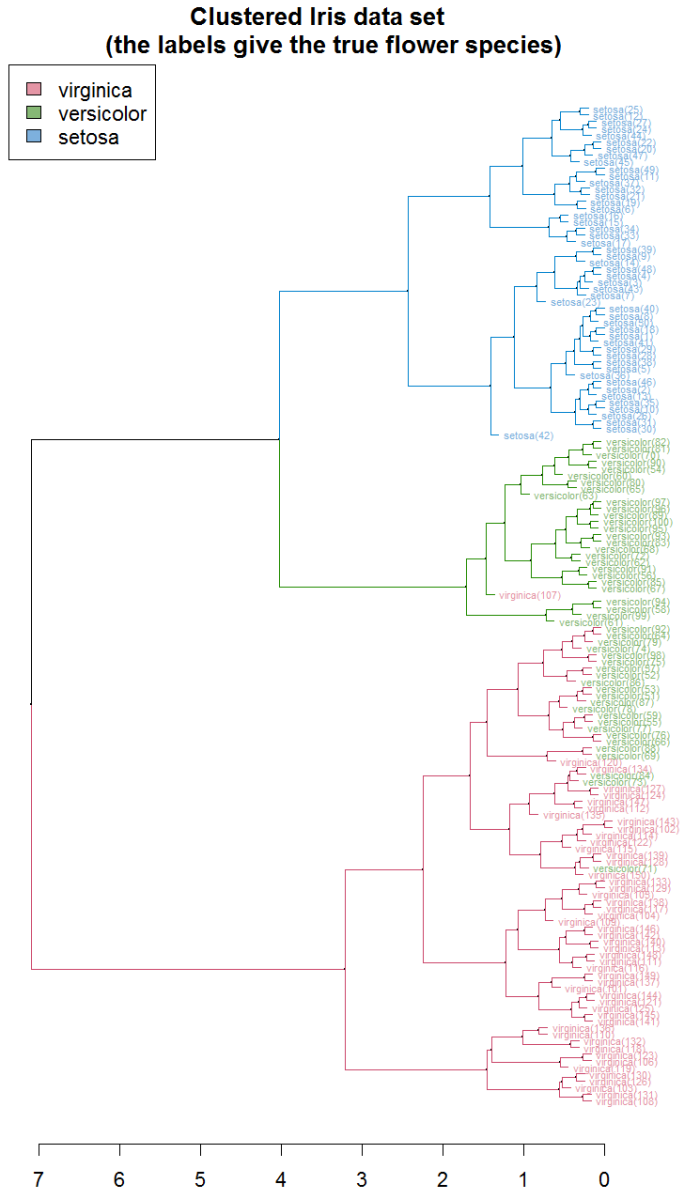
- Suppose we have n objects and know the pairwise distances between them, using a distance metric d.
- Start with n clusters, i.e. each cluster contains exactly one object. This means that at the beginning, the distances between clusters will be the same as the distances between objects.
- Store the distances between clusters in a square matrix M, and keep track of which cluster each object is in in a vector v. This can be done by assigning each initial cluster a number. (A convenient way to do this would be to use the clusters' row and column indices in M.)

Simple SLC algorithm: running the algorithm

- Find the lowest distance between two clusters. These two clusters, which we will call *a* and *b*, will be merged into one bigger cluster. Make note of the cluster numbers associated with *a* and *b*; we'll need these later.
- The merged clusters *a* and *b* are now considered to be one cluster. In the row of M containing the distances from *a* to the other clusters, if d(*a*,*x*) > d(*b*,*x*) for some cluster *x*, then set d(*a*,*x*) = d(*b*,*x*).
- Do the same thing for the column corresponding to distances from *a*.
- After all this, set d*(b,x*) to NaN (or some extremely high number) for all *x*, to reflect the fact that cluster *b* no longer exists.
- We still need to update the entries in v to reflect that all objects in cluster *b* are now part of cluster *a*. For all entries in v whose value is the cluster number of *b*, change this to the cluster number of *a*.
- Repeat these steps until you have your desired number of clusters.

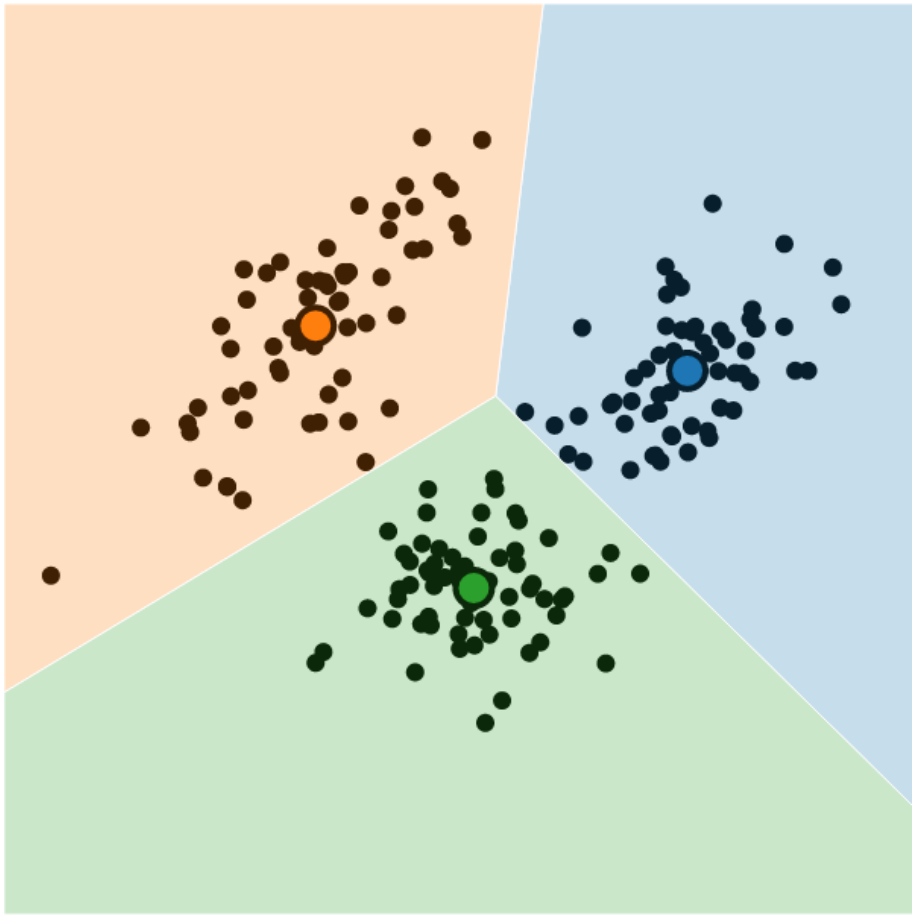
Dendrograms

- Single-linkage clustering can also be used to create dendrograms
- Here, each initial object can be considered a node of a graph
- In the previous algorithm, when *a* and *b* are merged, the distance between *a* (or *b*) and the node that joins the two is d(*a*,*b*)/2
- Continuing the algorithm until only one cluster remains, while joining nodes at each step, produces the dendrogram



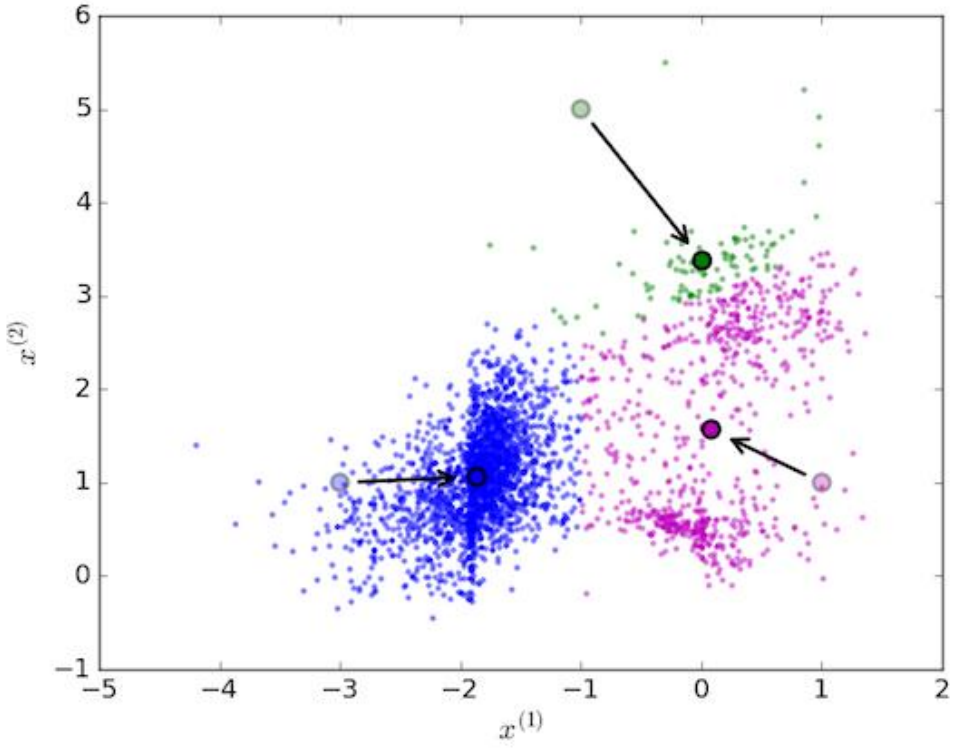
k-means clustering

- This is a method that clusters vectors by their distance to each of *k* different cluster means (not part of the original dataset)
- It works by creating a Voronoi diagram out of the vector space
- Hence, k-means splits along straight lines and generates convex clusters



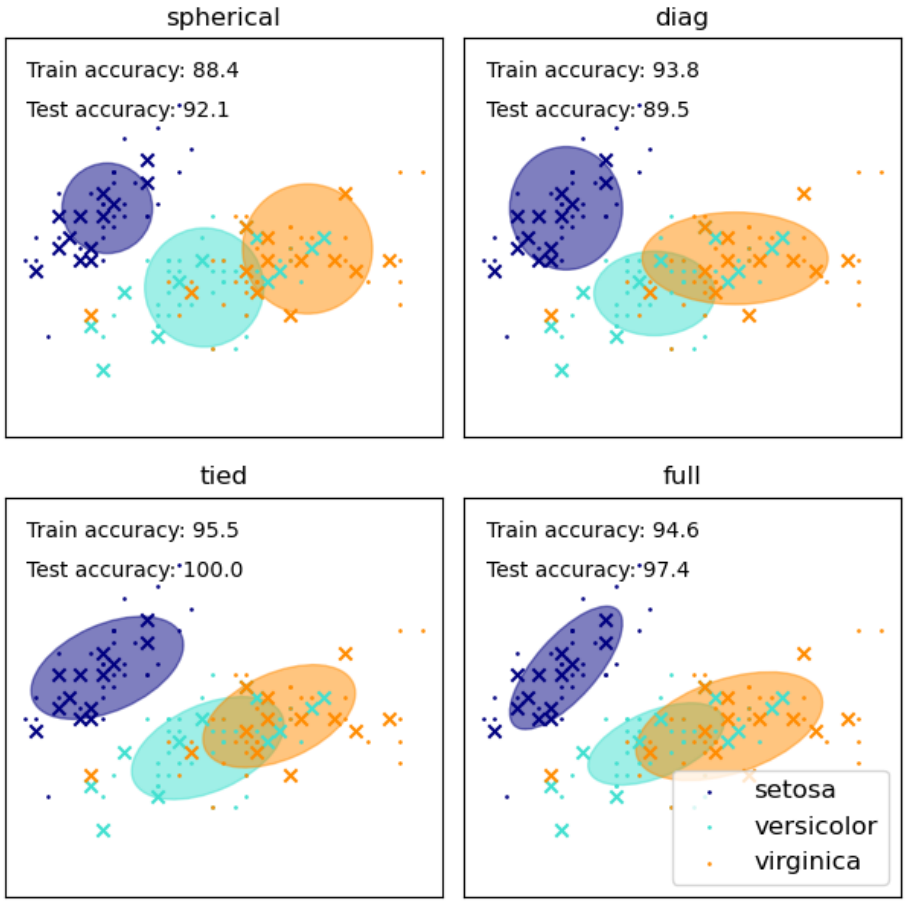
How k-means works

- Two steps: assignment, update
- To start, initialize *k* different cluster means as seed values (k-means++ is good for this)
- 1: Assign all vectors to a cluster based on which cluster mean they are closest to
- 2: Calculate the mean location of all vectors in each cluster; use these as the new cluster means. Repeat until convergence.



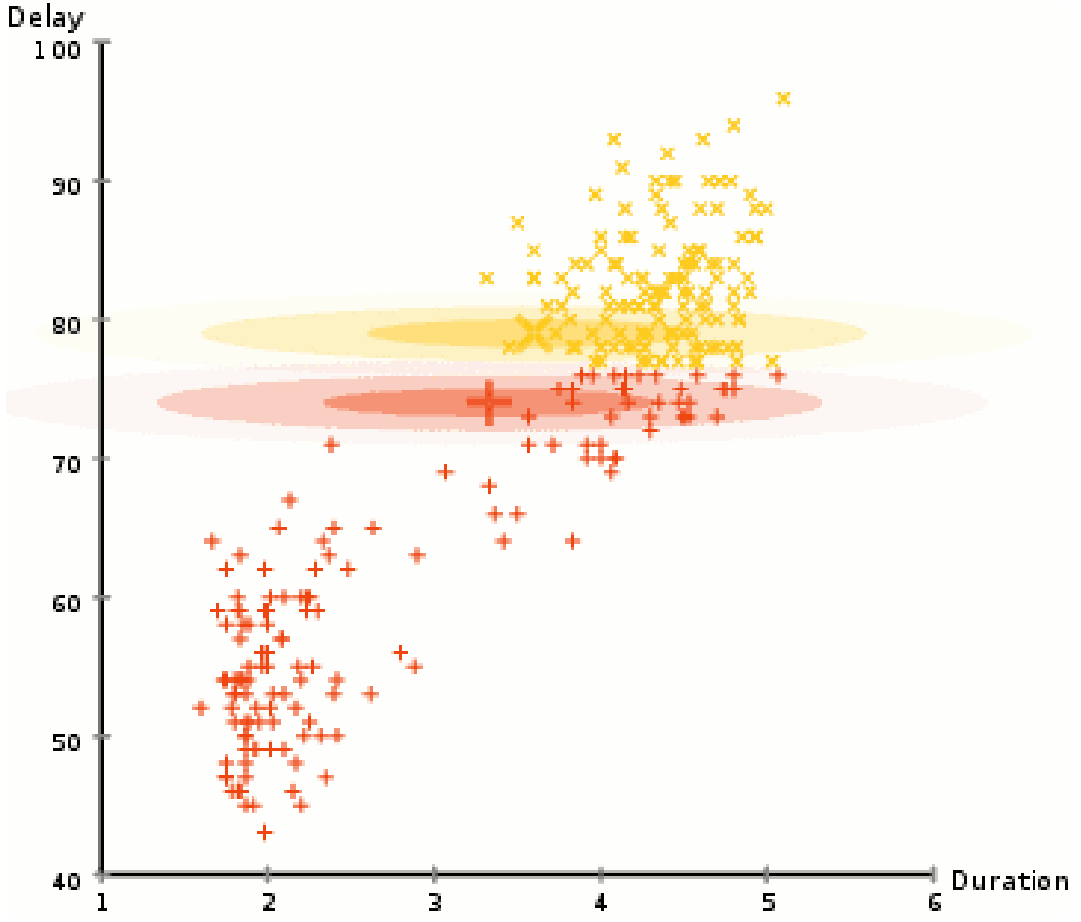
Expectation-maximization: a probabilistic clustering algorithm

- EM is a way to find properties of a distribution that some data is assumed to be drawn from
- This means that it can be used to create Gaussian mixture models, where each Gaussian can be considered as a cluster
- Using it this way makes it a version of k-means in which vectors are assigned to clusters probabilistically



How to use EM

- At the outset, initialize *k* different multivariate Gaussian distributions (these are your priors)
- Expectation step: For each vector, find the likelihood that that vector belongs to each distribution
- Maximization step: Given this distribution membership data, recalculate the mean and variances of each distribution

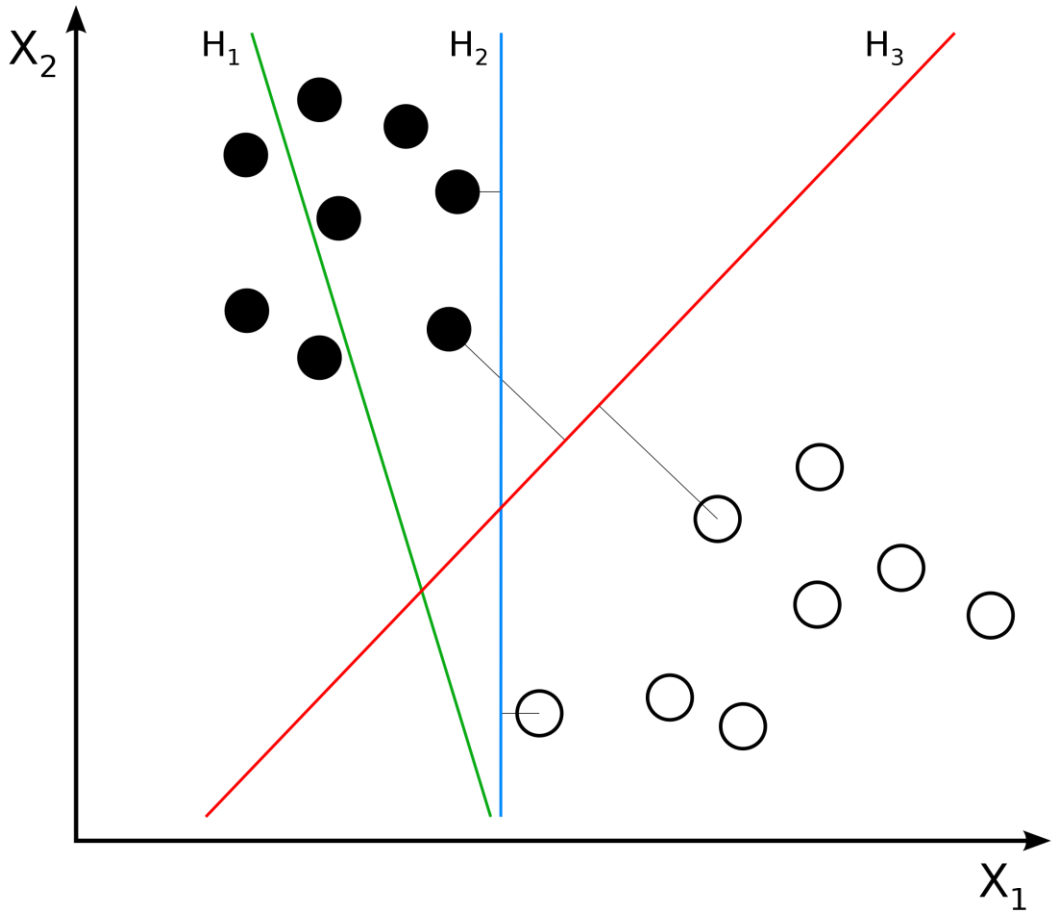


What if you already know which categories your objects fall into, but want to classify new objects?

Support vector machine, decision trees

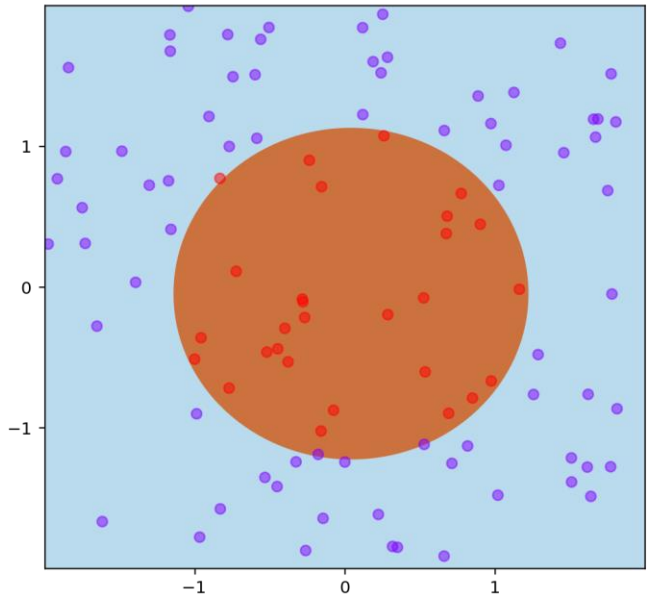
The support vector machine

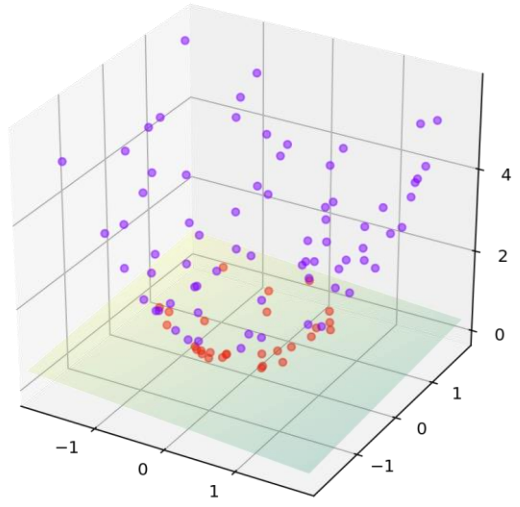
- Given some data that is classified into two categories, a SVM will produce an equation separating the categories
- Specifically, a SVM creates a hyperplane (a higher-dimensional analogue of a line or a plane)
- The chosen hyperplane maximizes the gap between instances of the two categories
- New objects can be classified based on which side of the hyperplane they fall on



Separating categories by using dimensionality to our advantage (the “kernel trick”)

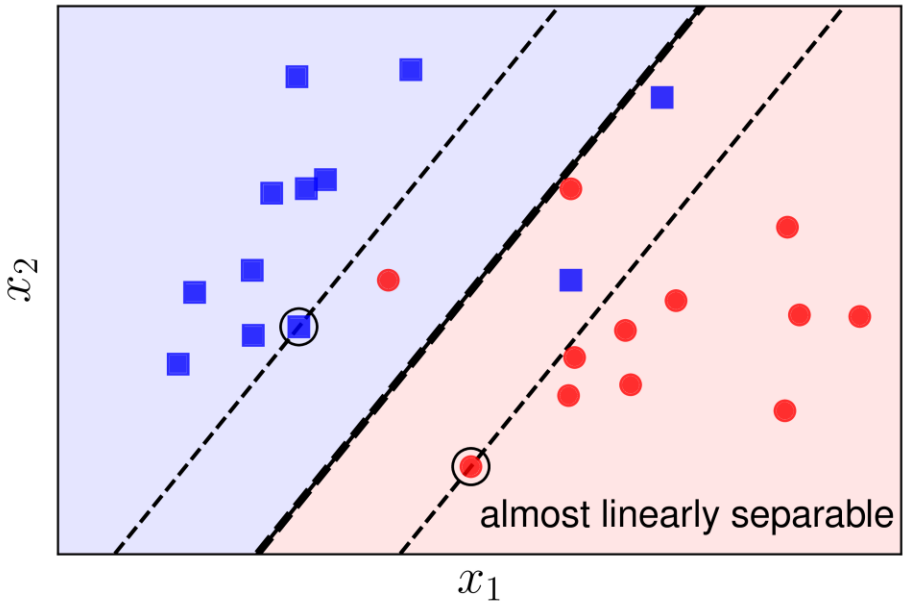
- SVMs can distinguish categories that are linearly separable, since hyperplanes are linear
- However, you can find nonlinear boundaries by implicitly calculating more dimensions
- This is done by defining an inner product on the vectors in your dataset





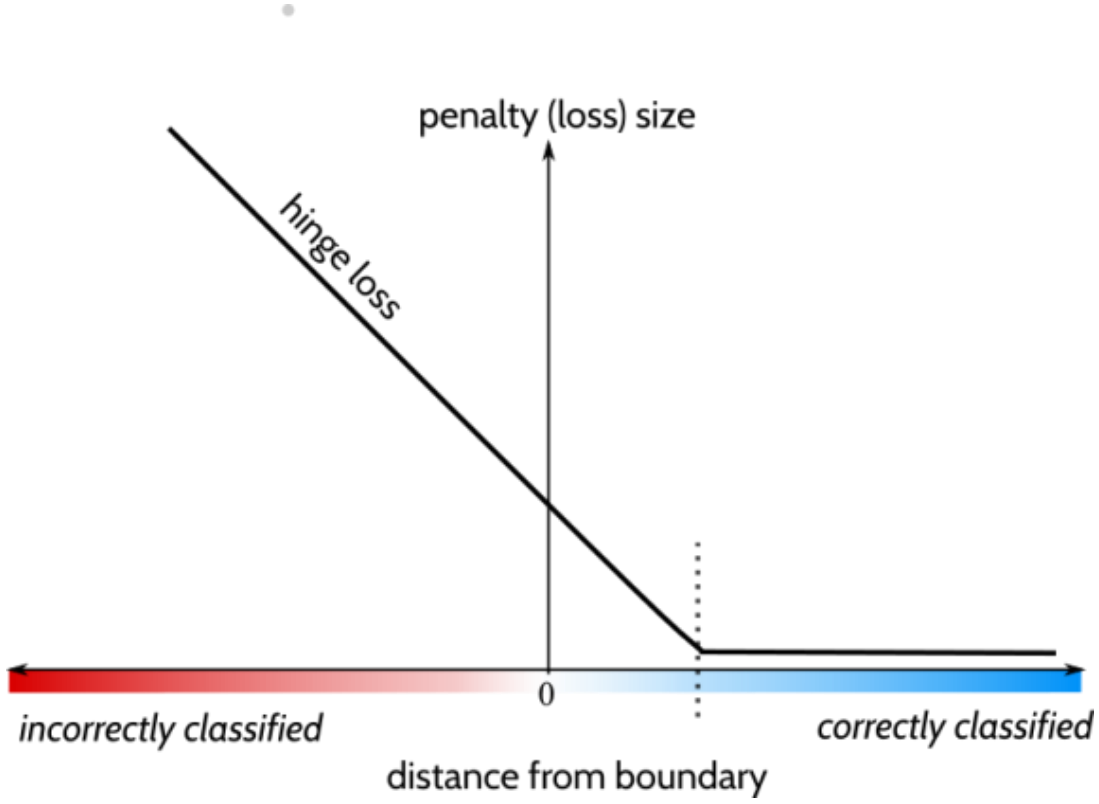
SVMs can be hard or soft

- Categories that are linearly separable (“hard margin”) can always be separated by a SVM
- If the categories cannot be linearly separated (“soft margin”), a loss function is required to select a hyperplane
- In such a case, we want a hyperplane that does the best job at classification rather than a perfect job



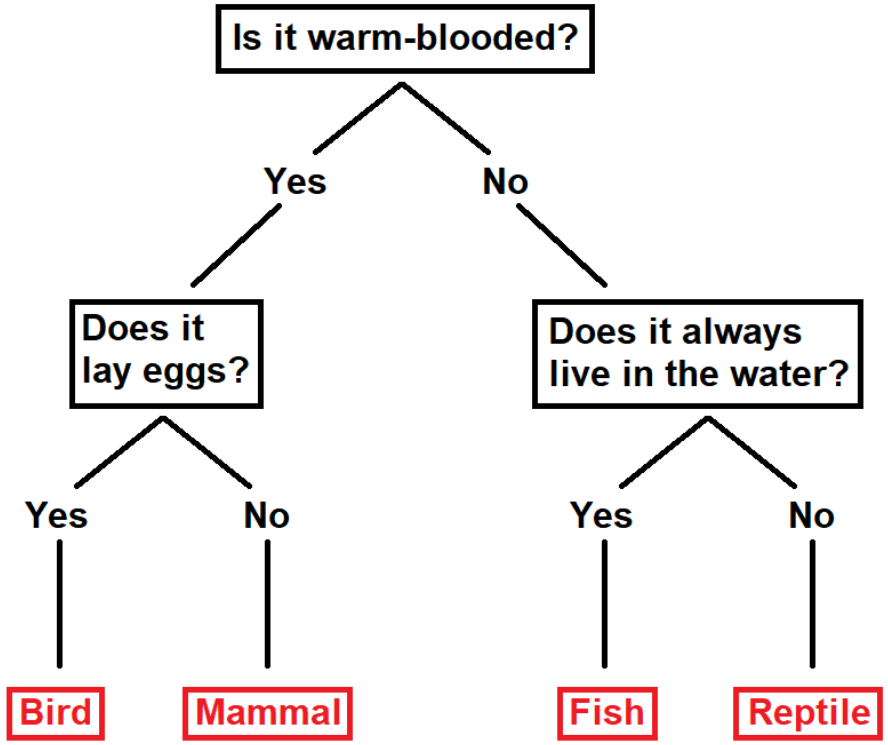
The hinge loss function for soft SVMs

- Hinge loss, i.e. f(x) = max(0, 1-x) or smoothed versions, is commonly used to quantify SVM error
- For points on the correct side of the hyperplane with enough margin, the hinge loss function takes the value 0
- For points on the wrong side of the hyperplane (i.e. separated from their category), the hinge loss function scales with distance to the hyperplane



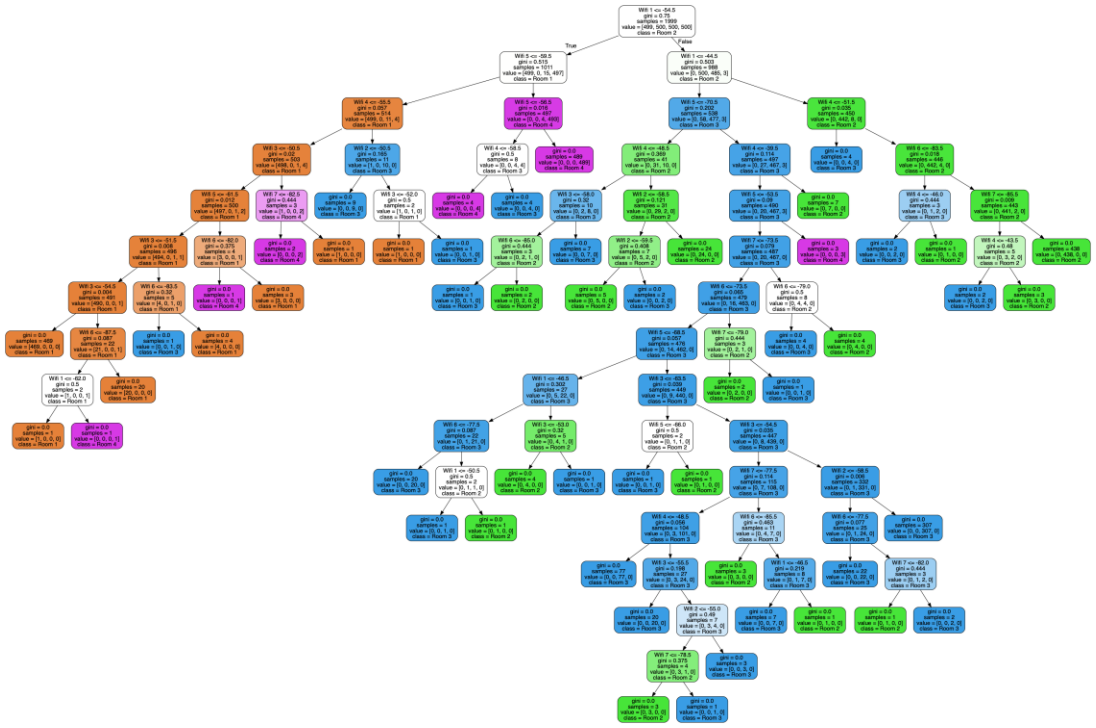
Decision trees

- A decision tree is a graph that takes an object as input, and has branching paths that ask questions about the object
- Following a path until the end yields a decision about the object
- Decision trees can use classification (where objects are assigned to discrete categories at the end) or regression (which instead have a continuous response variable)



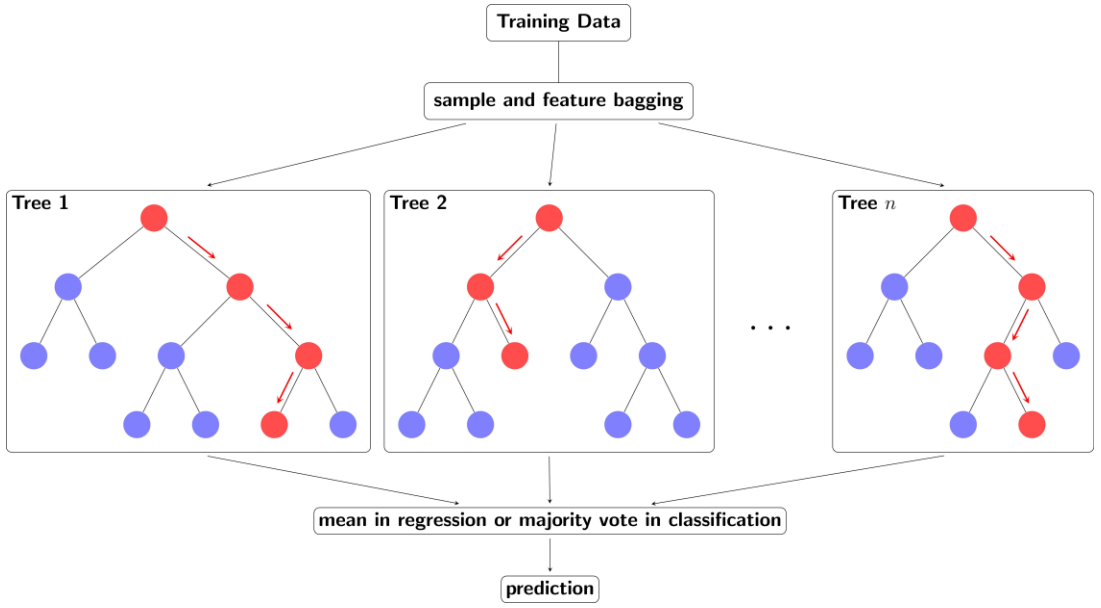
ML can be used to create decision trees

- This is done by repeatedly partitioning a dataset according to rules produced by a ML algorithm
- First the entire dataset is divided (thus making the first decision branches), then each subset created by this decision is itself divided, et cetera
- Trees that are too complex may induce overfitting; this can be addressed using ensemble methods like bagging or boosting



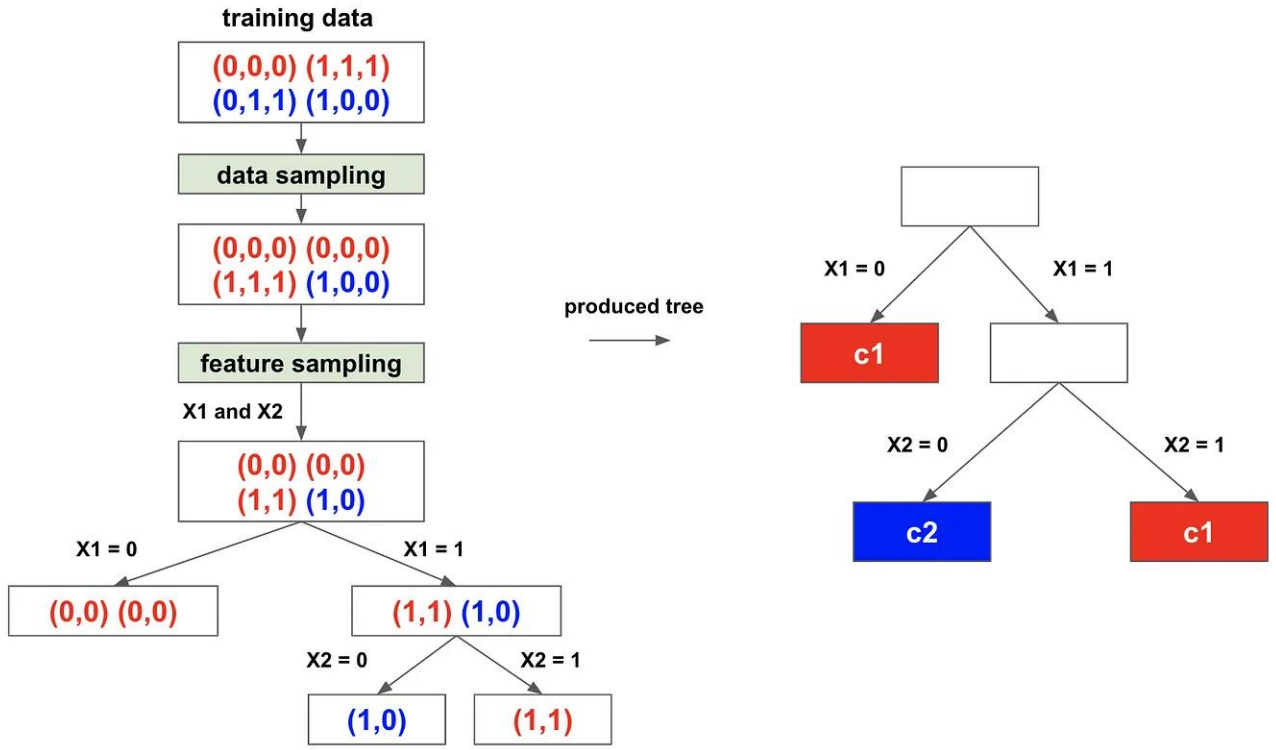
Having multiple trees can address the risk of overfitting

- Trees can be aggregated in a “random forest”, where the mean prediction made by all trees in the forest is adopted by the model
- The random forest algorithm uses bagging (bootstrap aggregation) to induce randomness at two different stages
- Each tree in the forest is trained on a random subset of the data, and the features that the tree uses for classification are chosen randomly



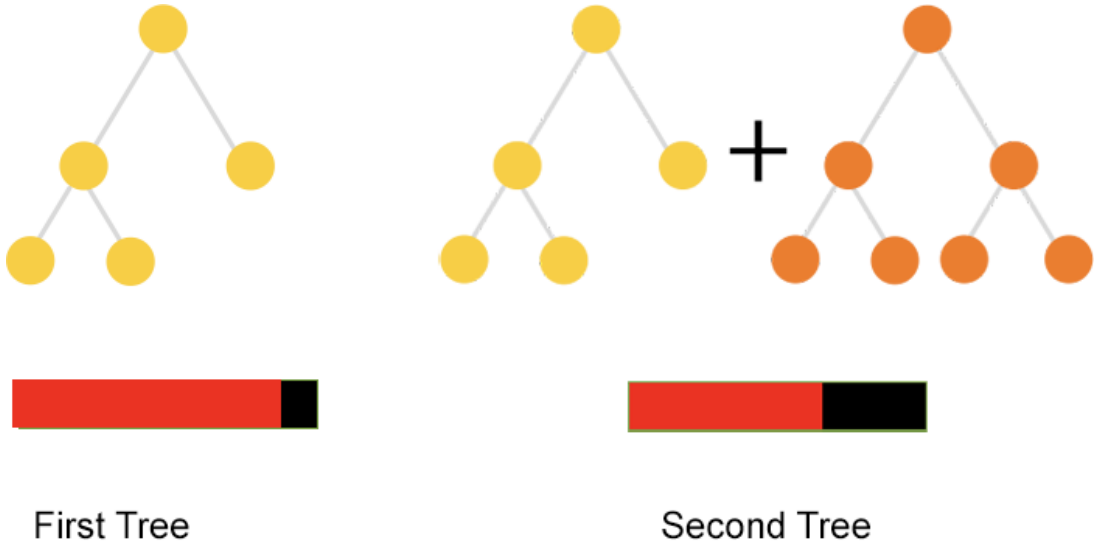
How a random forest is constructed

- Sample available training data with replacement to build a dataset for creating one tree
- Sample features with replacement to choose which feature each step in the tree will use to split the data
- Do this many times to produce lots of trees

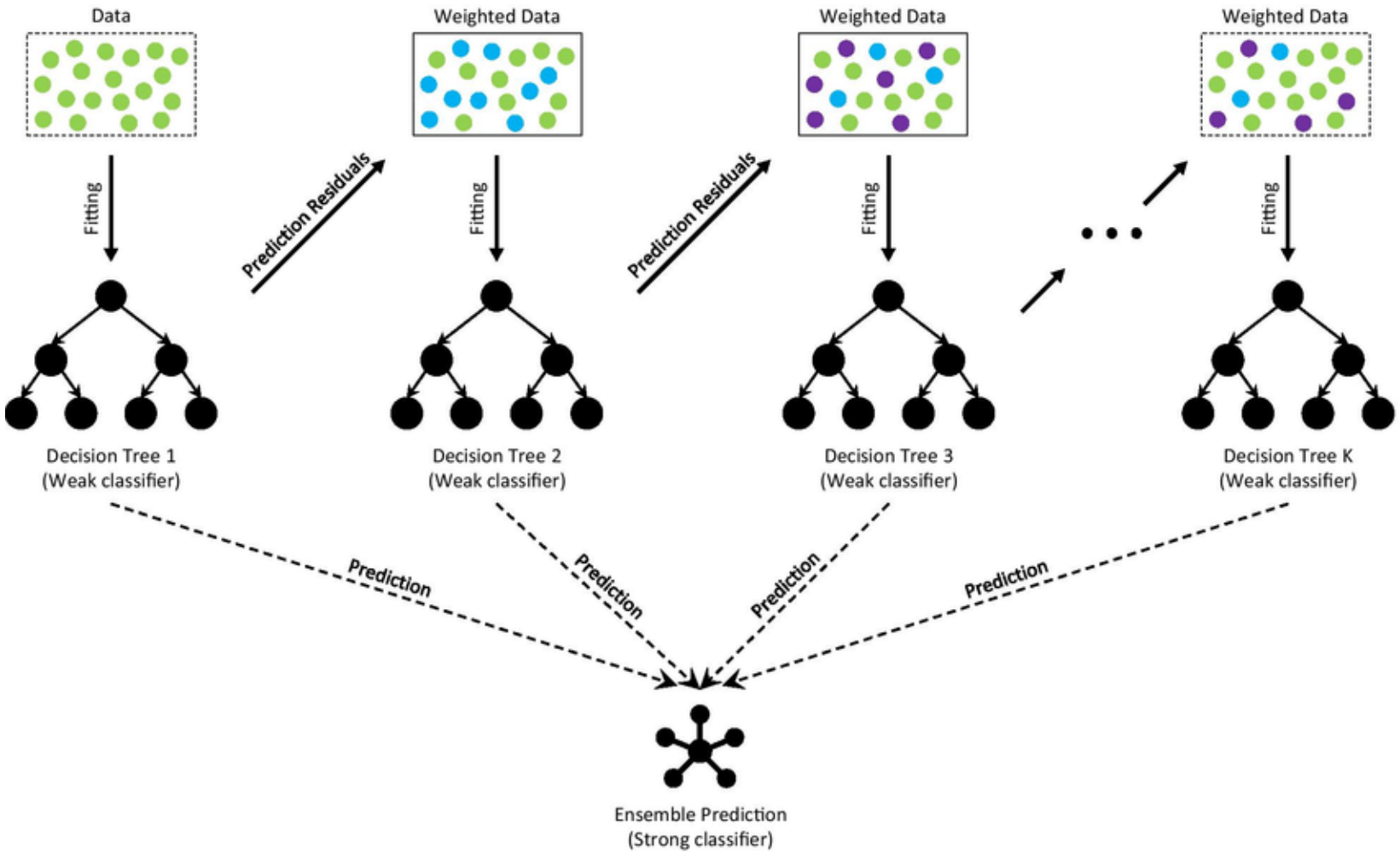


Boosting: another way to get more accurate predictions from decision trees

- Boosting assembles a strong classifier from many weak ones in a sequential training process
- To use boosting with decision trees, we first create a simple tree trained on the observations in our dataset (often with only one step)
- We then use the residuals or loss function gradients of this tree to train the next tree, and so on until we have enough trees to aggregate
- This process allows us to correct mistakes made by previous trees



How boosting works in schematic form

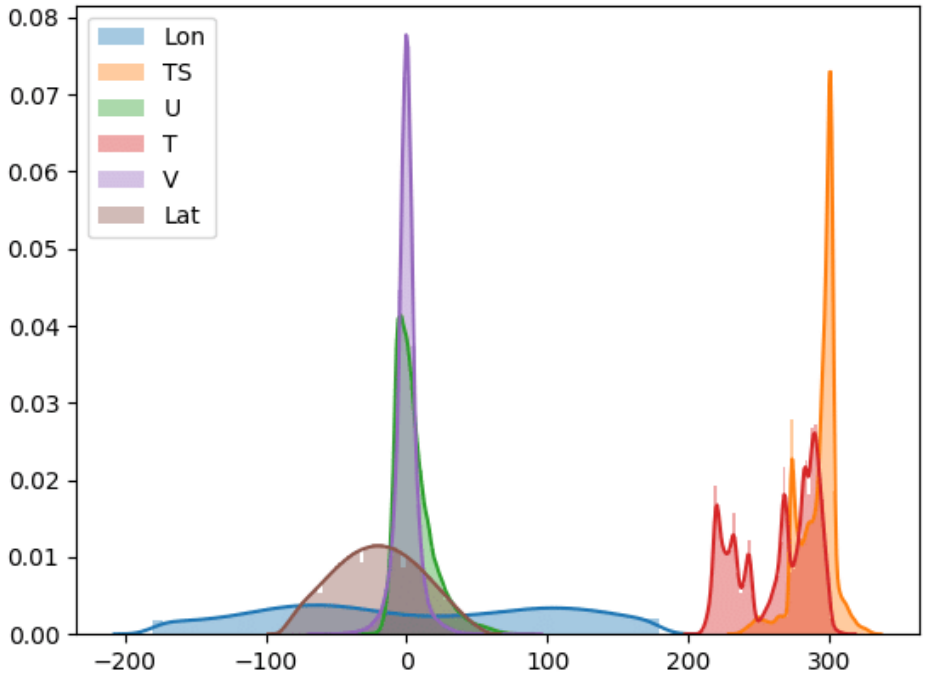


What if you have high-dimensional data and want to better visualize its structure?

Principal component analysis, t-distributed stochastic neighbour embedding

A brief word about scale in datasets

- If you have a lot of different features in a dataset, they might be on different scales
- Suppose you have two features that take values {1,2,3,4,5} and {100,200,300,400,500}
- Feature importance algorithms might place far more weight on the second of those two features, even though it has the same relative structure as the first



How can we fix this?

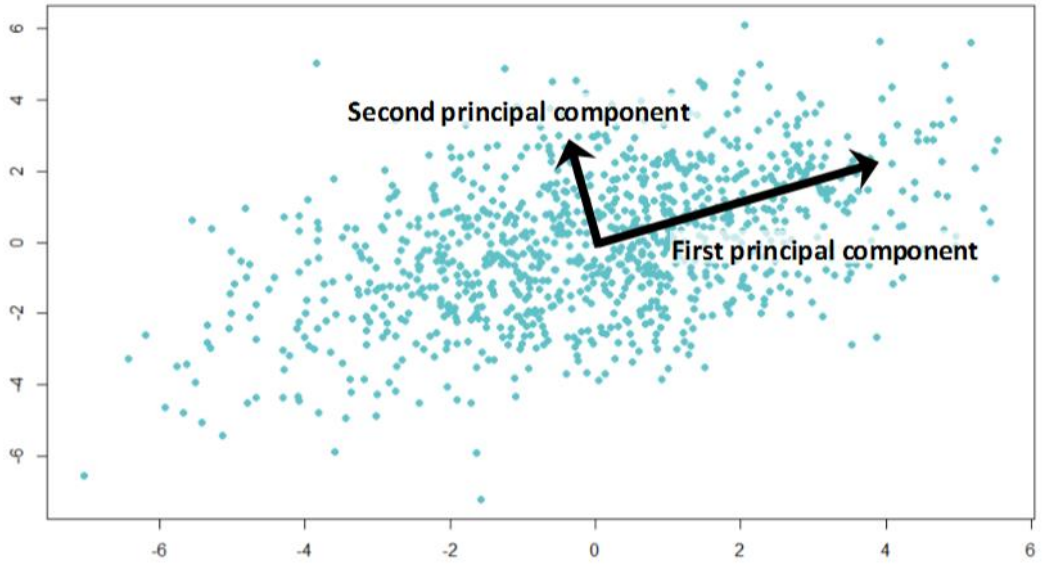
- Normalization scales a feature's minimum value to 0 and its maximum value to 1
- Another method is converting feature values into Z-scores
- The Z-score of a value in a distribution is how many standard deviations it is away from the mean

$$\tilde{x}_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

$$\tilde{x}_i = \frac{x_i - \mu}{\sigma}$$

Principal component analysis

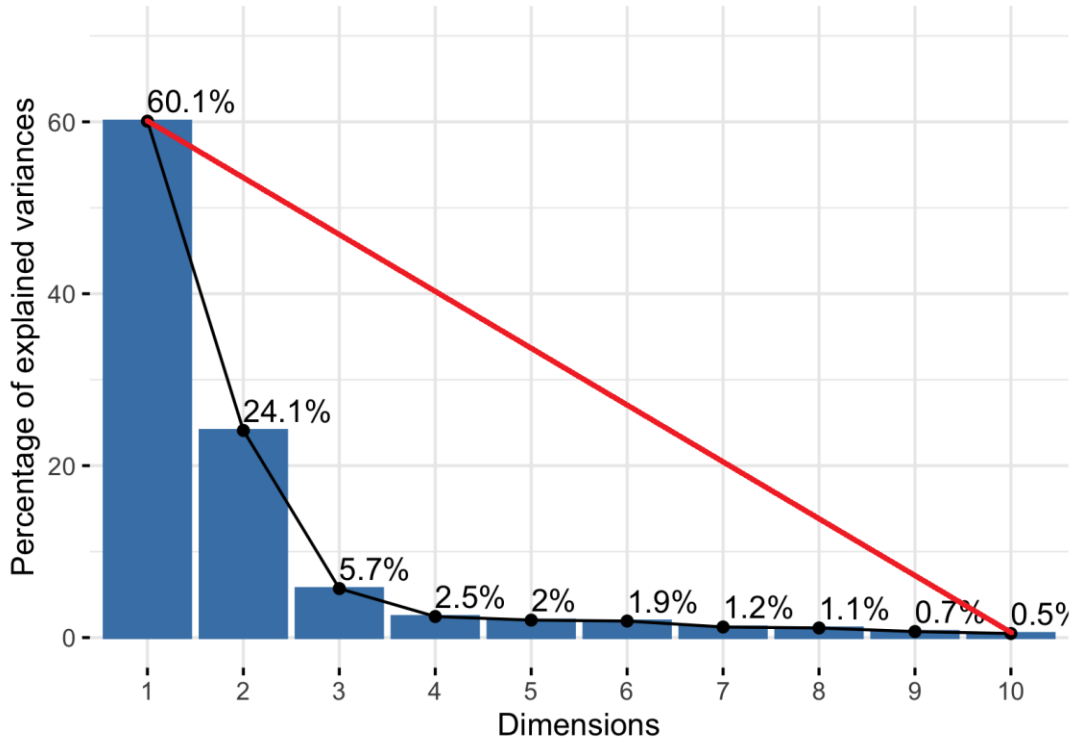
- If you have a really big dataset, some of its features might be correlated with each other
- PCA works by transforming data onto orthogonal axes (“principal components”)
- Each principal component is an eigenvector of the matrix of covariates between features
- Hence, the principal components are linear combinations of the original features



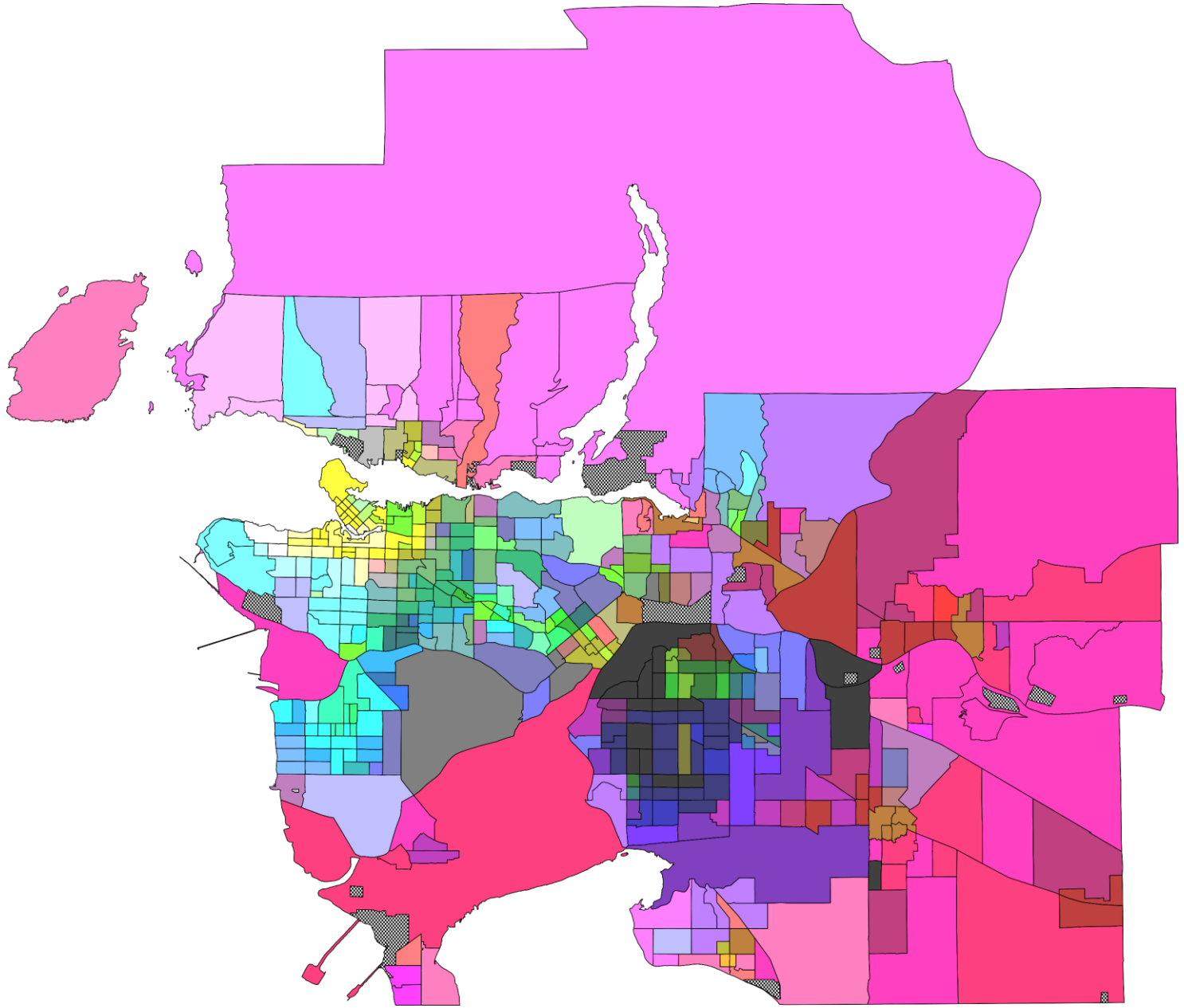
How many principal components are necessary to describe data?

- Usually, the first few principal components capture most of the variance in the data (so most can be dropped)
- One way to tell is by looking at a scree plot of explained variance and using the “elbow method” (finding a place where the scree plot bends, or has a sizable gap between successive components)
- Draw a line from the first point in the scree plot to the last one. The point farthest away from this line is a good guess

Figure 1

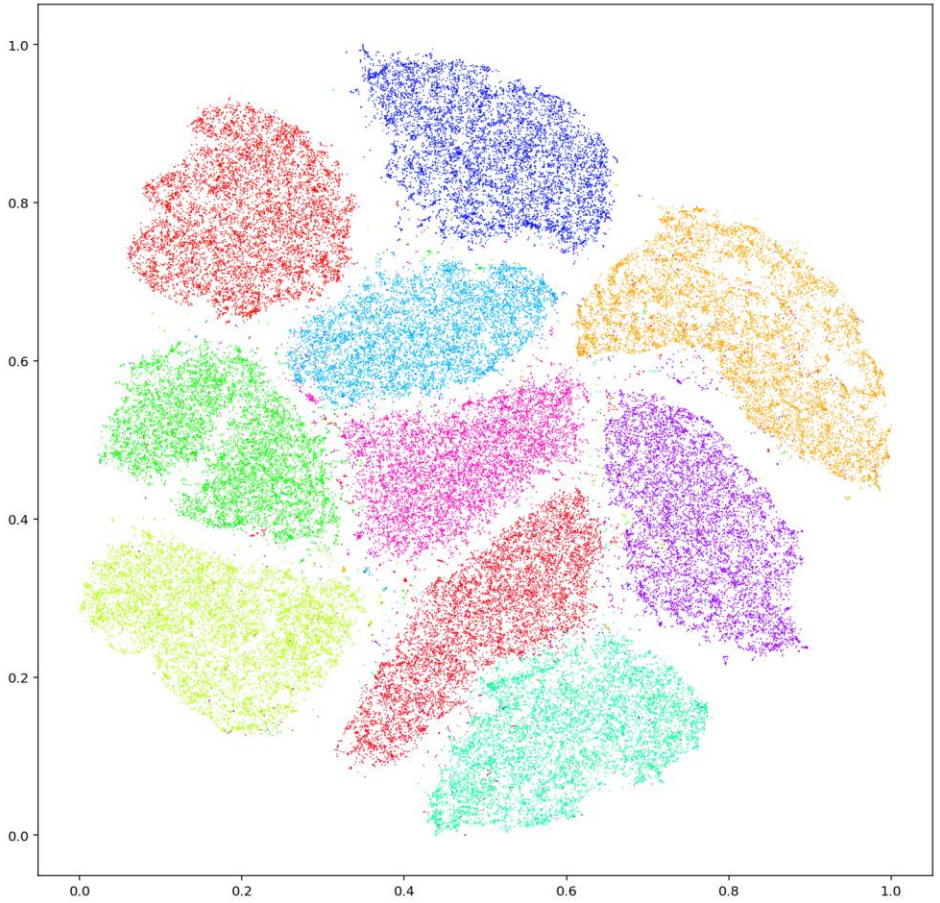


PCA can uncover big-picture patterns in data



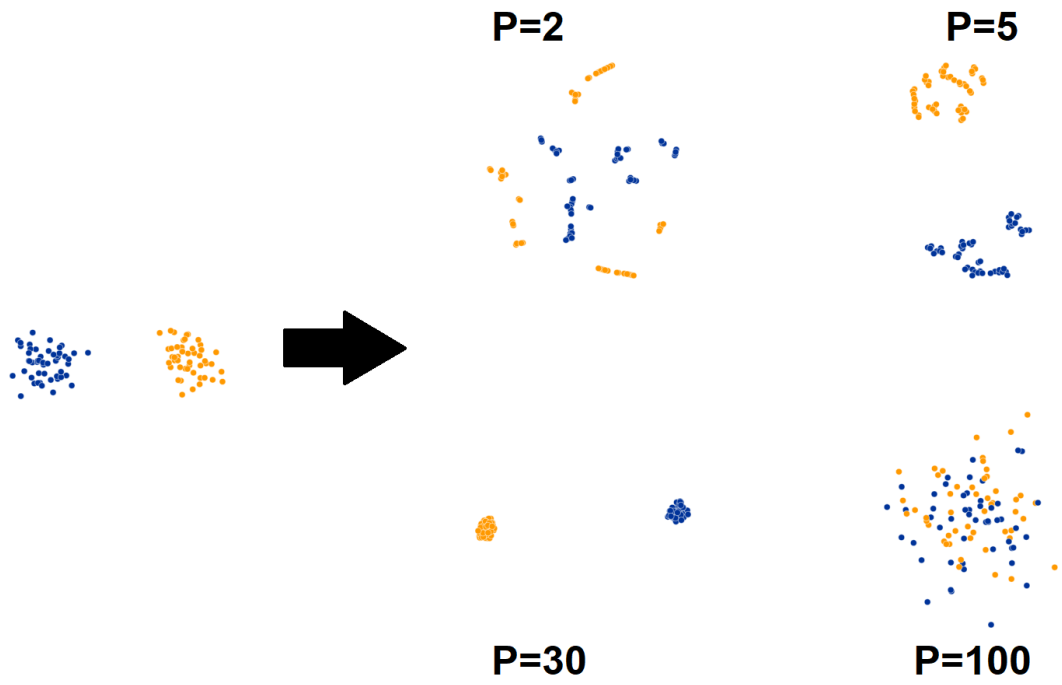
t-distributed stochastic neighbour embedding

- t-SNE is a method for visualizing high-dimensional data (e.g. MNIST), and a *de facto* clustering algorithm
- It works by finding the likelihood that each object would choose each other object as a neighbour, based on weighted distance functions
- There's an implementation of it in scikit-learn (in Python)



Quirks of t-SNE

- It can form clusters that are nonlinearly separated in high-dimensional space
- Its axes have no meaning, so neither do the sizes of the clusters or the distances between them
- It's non-deterministic (different runs give different results)
- Hyperparameters: "perplexity" is sort of like the desired number of neighbours for each point; epsilon is t-SNE's learning rate

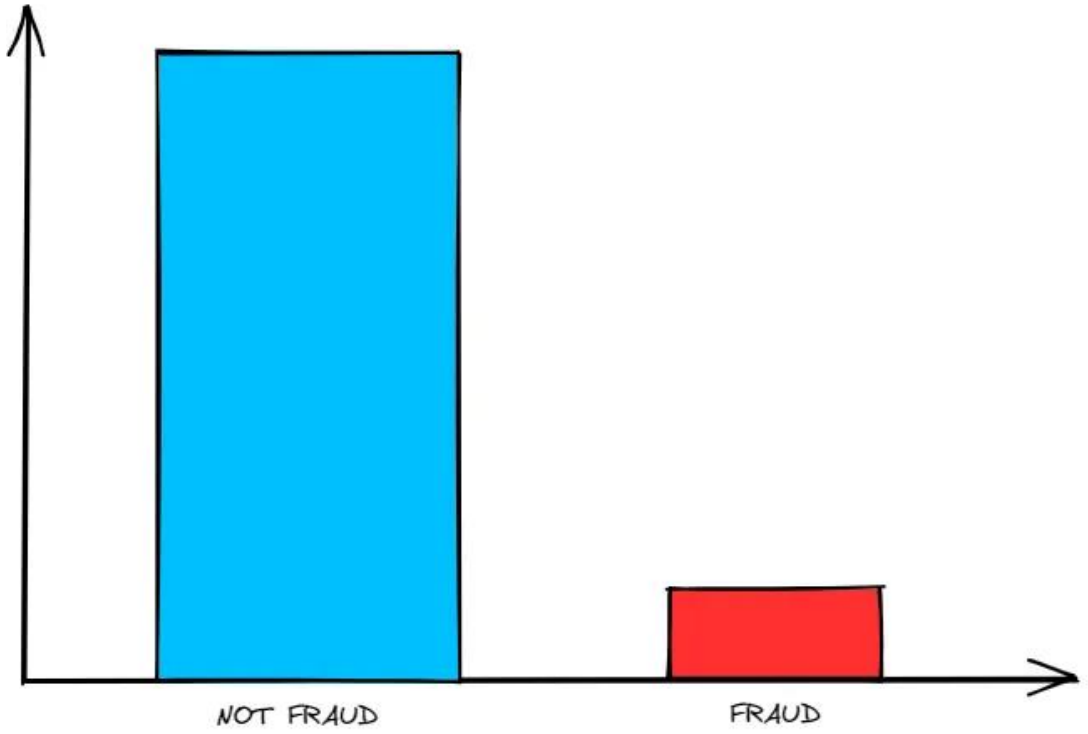


What if you want to do further analysis on your data, but it’s split into imbalanced categories?

SMOTE

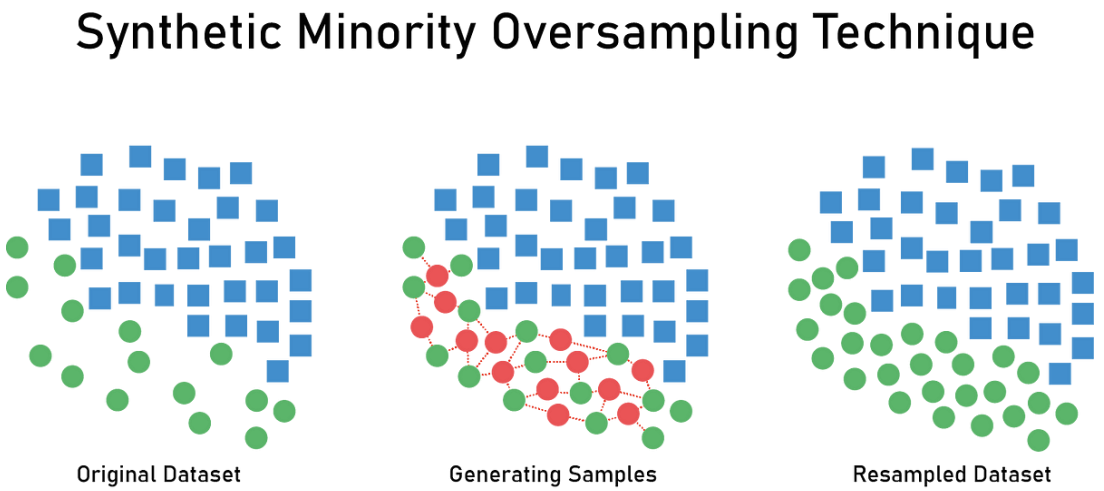
Why are imbalanced datasets a problem?

- Sometimes, an event of interest occurs very rarely, and training data for a model may have very few instances of it occurring but many of it not occurring
- In such cases, null models (e.g. “nobody ever commits credit card fraud”, “nobody ever gets this rare form of cancer”) may have high predictive power despite being useless or harmful



A solution: Synthetic Minority Oversampling TEchnique (SMOTE)

- SMOTE creates new observations by taking linear combinations of existing observations in the class with fewer members
- This is done to observations that are close enough together that the linear combinations of them will still be in the minority class
- The idea is to make perturbations to data points in the minority class, so we get new points rather than resampling the existing ones



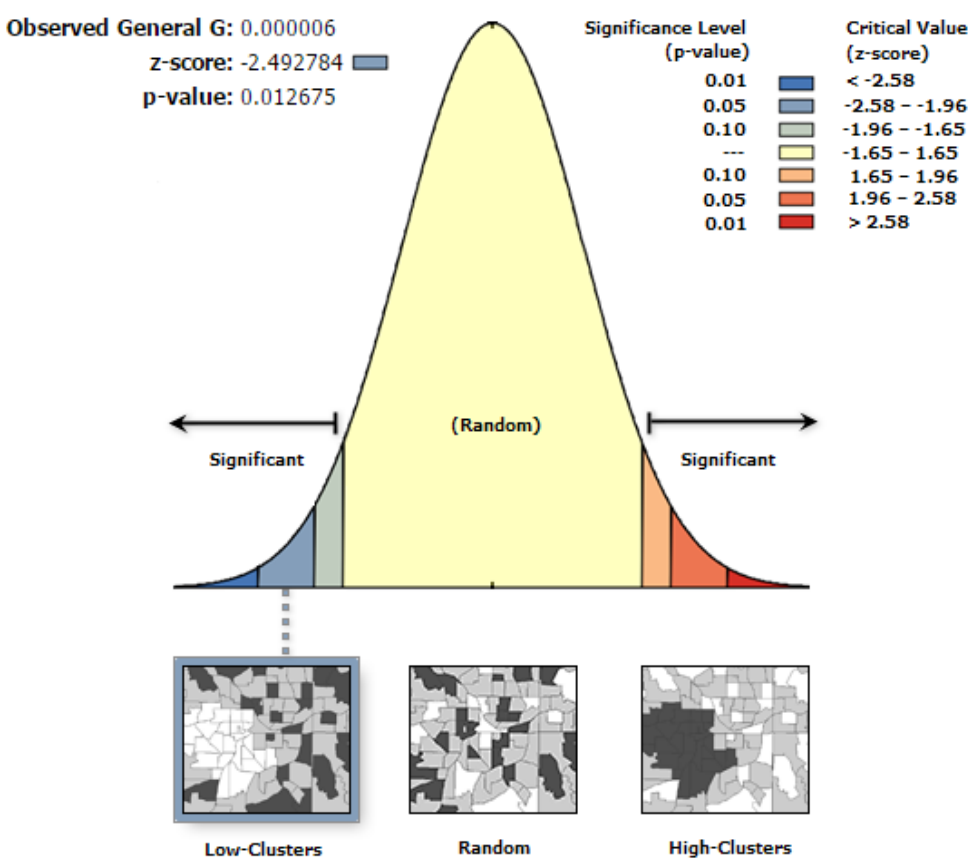
That’s all for today

What if you already have a cluster of points, and want to see which ones are the most prominent?

Local Getis-Ord statistic

Hot spots and cold spots

- Suppose you have some observations that are joined together in a network, for example an adjacency matrix (for spatial data)
- A hot spot is where several observations with high values of a variable of interest are close together in the network; likewise for cold spots and low values



The local Getis-Ord statistic: a method for quantifying hot and cold spots

The local Getis–Ord statistic is defined over graphs. Given a graph with vertices $i = 1, \ldots, n$ and edges given by an adjacency matrix ω, if each vertex i is associated with a value x_i, the z-value and G_i^* are defined by

$$G_i^* = \frac{\sum_j w_{ij} x_i - \bar{X} \sum_j w_{ij}}{s \sqrt{\frac{n \sum_j w_{ij}^2 - \left(\sum_j w_{ij}\right)^2}{n-1}}},$$

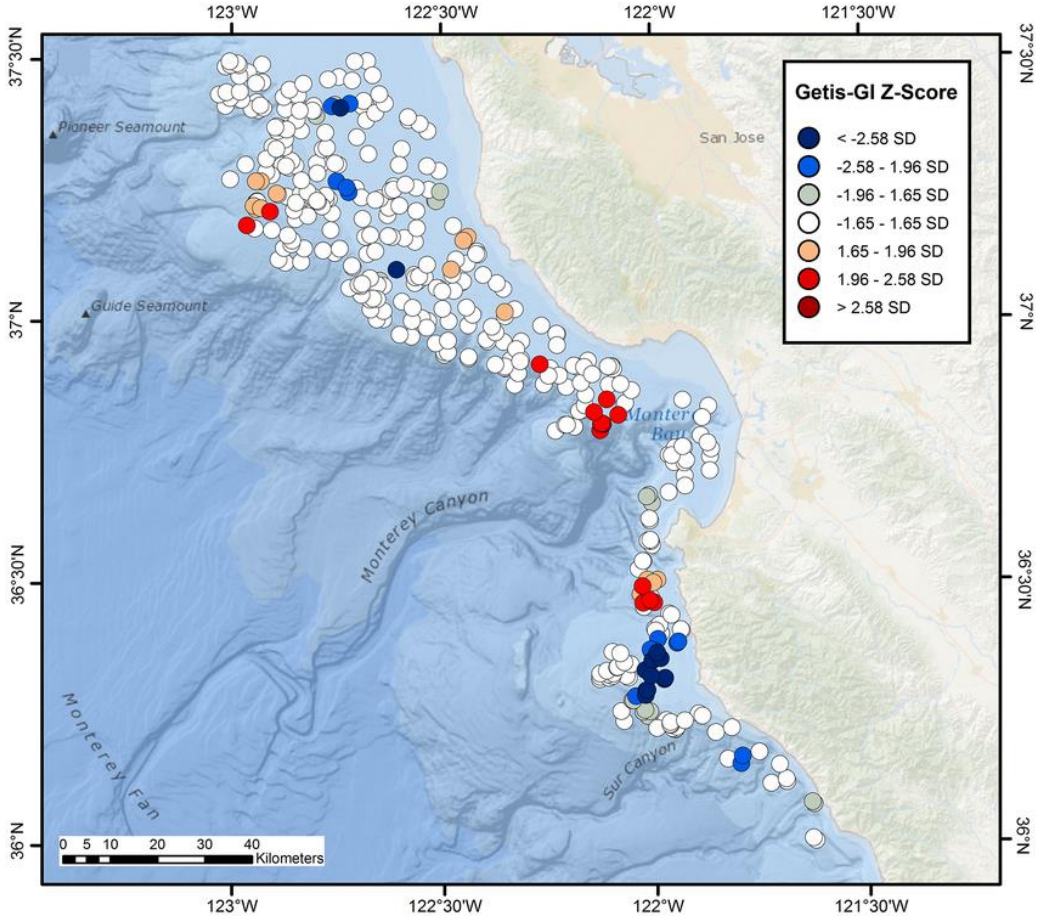
where

$$\bar{X} = \frac{\sum_j x_j}{n}, \quad s = \sqrt{\frac{\sum_j x_j^2}{n} - \bar{X}^2}, \quad z_i = \frac{G_i^* - E[G_i^*]}{\sqrt{VarG_i^*}}.$$

The p-value p_i for each vertex i is calculated by the probability that the z-value z_i occurs under the null hypothesis.

Hot spots often come up if you're doing anything with spatial data

- A common application of local Getis-Ord is to find out whether a variable of interest is abnormally high or low across some region
- This makes it very common in applications involving satellite imagery and/or GIS data
- It can be used on any kind of mesh connecting local observation points, even unusual ones such as a lakeshore (i.e. circle-shaped graph)

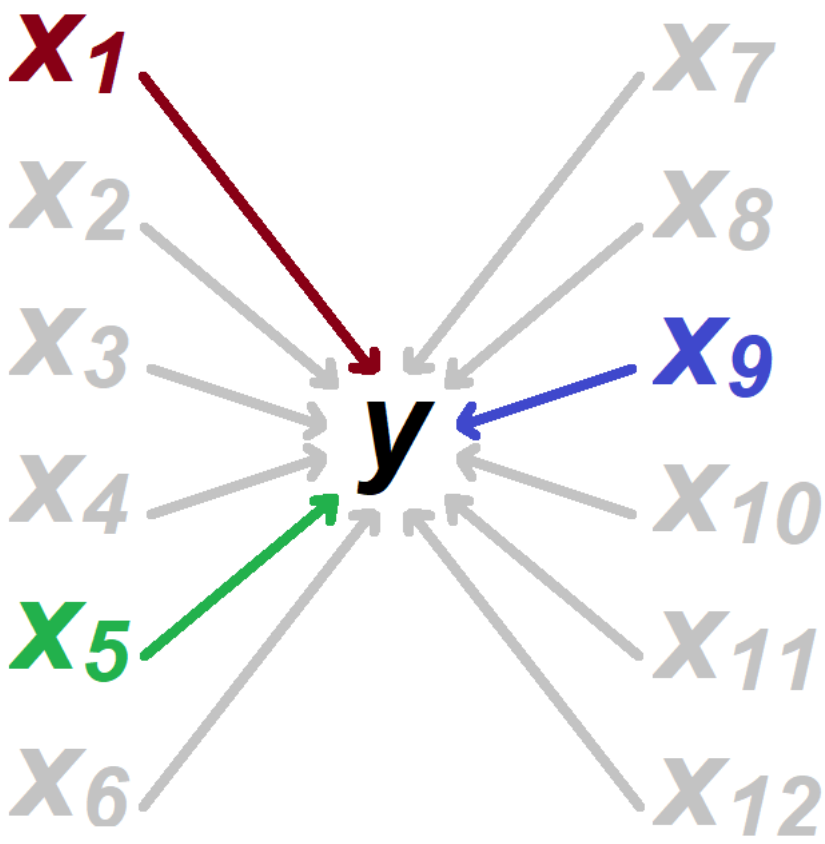


What if you have a few different features and want to see which ones are important?

Lasso method, Shapley values

Sparse regression

- Suppose we're doing a regression (linear or otherwise), and we have a lot of variables
- If we use too many, we run the risks of both overfitting and a very long computation time
- Instead, we may want to assume that the number of variables in the regression model is sparse compared to the number of observations



The lasso method: feature selection for a sparse regression

- Lasso ("least absolute shrinkage and selection operator") makes there be fewer regression features by setting some regression coefficients to zero
- It works by using the L^1 norm of the regression coefficients as a penalty term added to least squares error
- The geometric interpretations of these two terms are a constrained area around the origin with corners on the axes and an ellipse around the point that minimizes least squares
- The geometry of the L^1 norm means that the least squares error ellipse is likely to touch a corner of the L^1 area

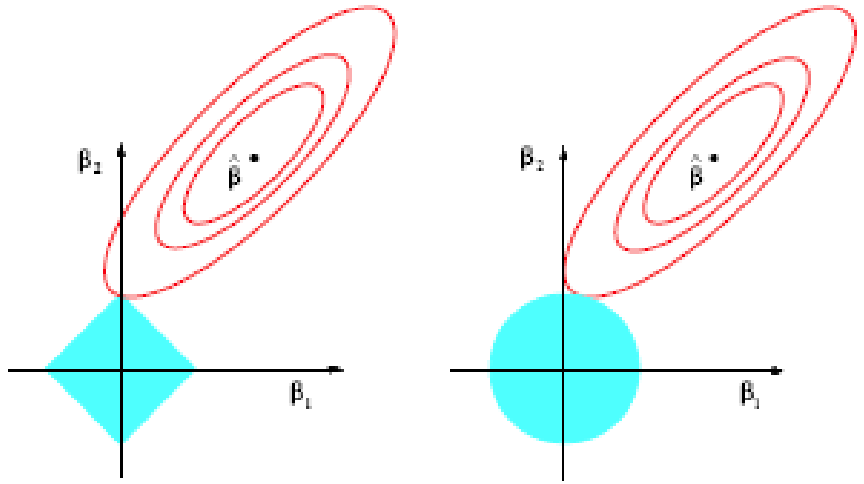


Figure 3.12: *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$, *respectively, while the red ellipses are the contours of the least squares error function.*

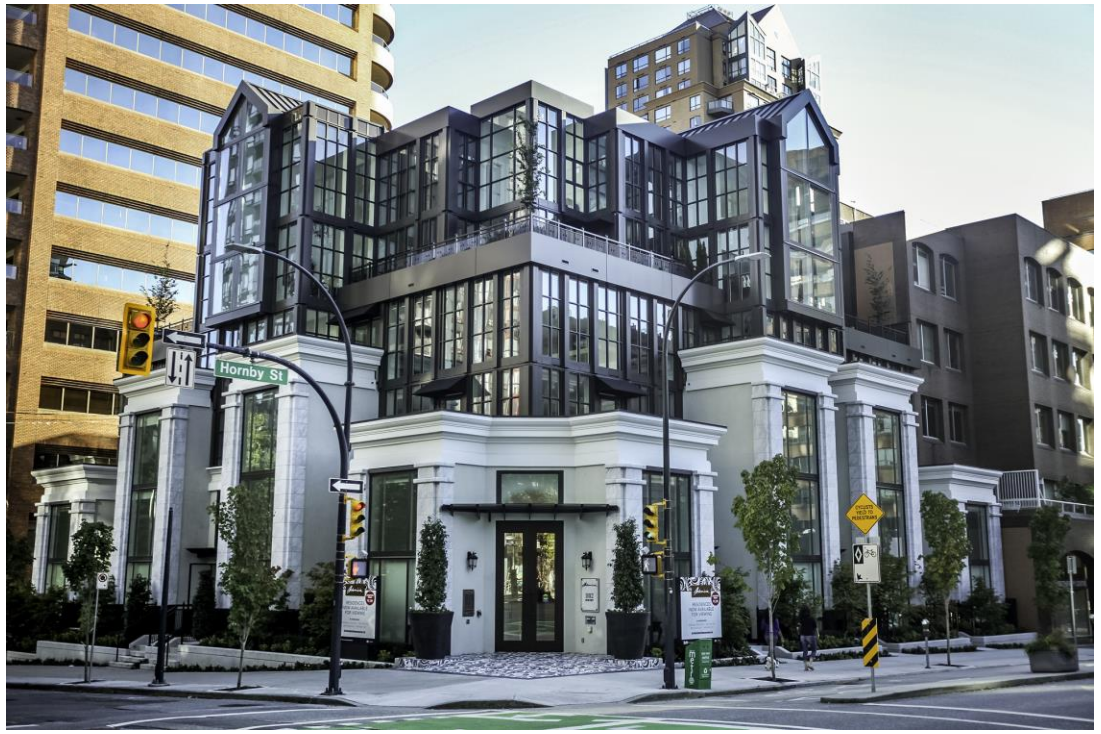
How to do a lasso

- Lasso minimizes a loss function made up of two parts added together
- The first of these parts is the residuals of a least squares regression
- The second part is some constant λ times the L^1 norm of the regression coefficients (i.e. the sum of their absolute values)
- λ is chosen beforehand (this prevents the λ=0 optimization condition), and can be tuned to balance sparsity with a good fit

$$\text{Minimize:} \sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

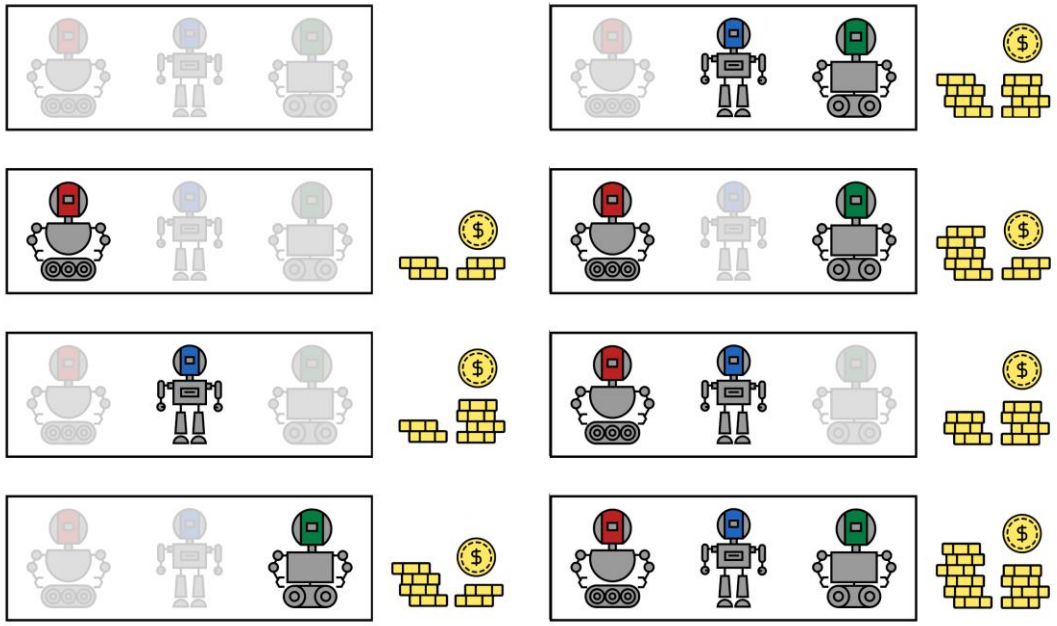
A machine learning-based hypothetical scenario

- Suppose you have a ML model that predicts apartment prices
- Suppose it predicts that an 80 m^2 apartment that's 30 minutes from downtown and in a building that's 15 years old would cost $1.4M
- You might wonder how much the 80 m^2 size, the half-hour commute, and the 15-year age each contribute to the predicted price
- However, ML models can be opaque and difficult to interpret



Shapley values

- These are based on a concept from game theory designed to distribute payout to players
- Each player in a game is assigned some portion of the payout depending on how much more a typical team with them earned than a typical team without them
- They can also be used in ML to assess how much a given prediction is due to some feature value being what it is

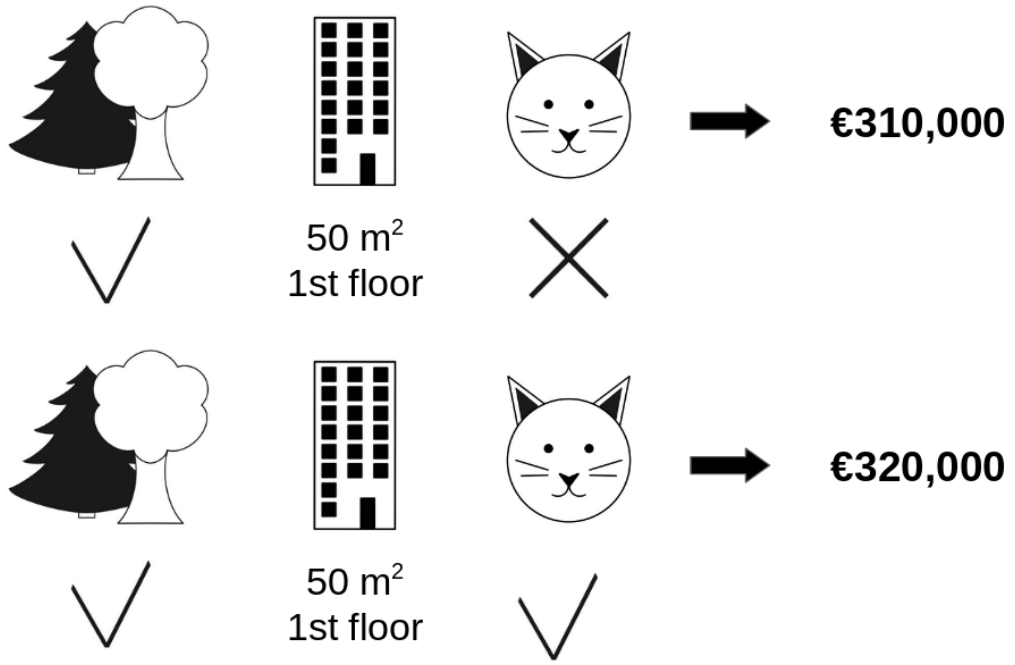


How can ML modelling be thought of as a game?

- Game theory considers games in which players can earn a payout. For instance, in the game Prisoner's Dilemma, if Player 1 cooperates and Player 2 defects, P1 gets a payout *S* and P2 gets a payout *T*, where T > S
- Since the original use of Shapley values was to apportion payout amongst a bunch of players depending on how much they contributed, we can use the same framework to "apportion" a predicted response variable value amongst features depending on how much they raised or lowered it
- This works by having features collaborate in "coalitions", and determining how much the response variable is predicted to change when a feature plays (is part of the coalition) vs. doesn't play (isn't part of the coalition)
- Here, a game is one specific prediction made by a model (i.e. with specific feature values), the players are the features, and the payout is the predicted value for the response variable in this specific case minus its expected value

How Shapley values can be computed (the simplified version)

- Suppose we predicted a response variable *R* to have value *R** using a model with *n* features total
- For a given feature *x*, there are 2^{n-1} possible coalitions without *x*
- Pick one such coalition, called *C*, and predict *R* using just the features in *C*
- For each feature in *C*, use its value from when you predicted *R**
- For each feature not in *C* (including *x*), use its expected value (integrate it out or do sampling)
- Now, add *x* to *C*, and predict *R* in the same way using the new coalition. Repeat for all coalitions

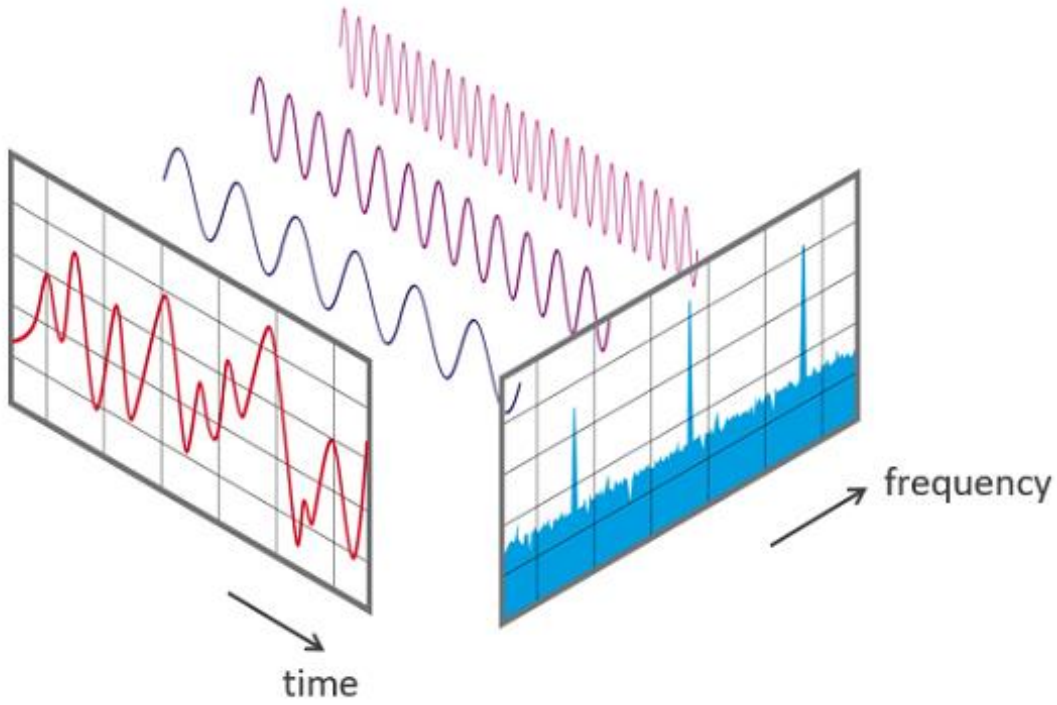


What if you have a recurrent signal and want to know when it will show up?

Short-time Fourier transform, wavelets

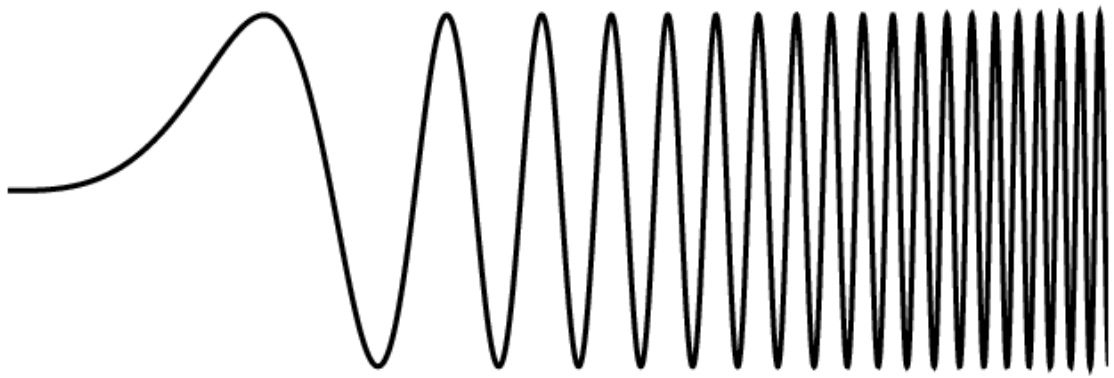
Background: Fourier series

- They’re a way of decomposing a function into sines and cosines, which means you can use them to extract periodic frequencies
- This is possible because sines and cosines form a basis for the space of all functions, so any function is a linear combination of them
- The Fourier transform thus converts between the time domain and the frequency domain



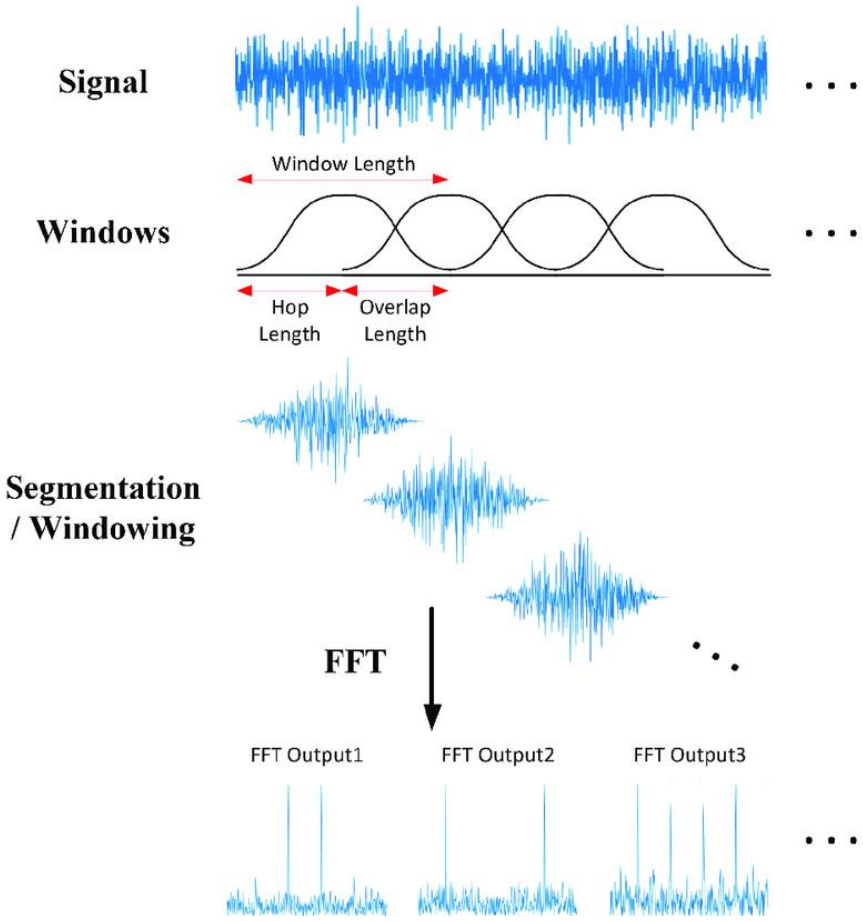
What if you have a wavefunction whose shape changes over time?

- Fourier transforms present a static picture of a function, as integrals used when constructing each term are over all time
- If you have a periodic signal that “speeds up” or “slows down” (i.e. its period changes), using a standard Fourier transform will result in missing or inaccurate information



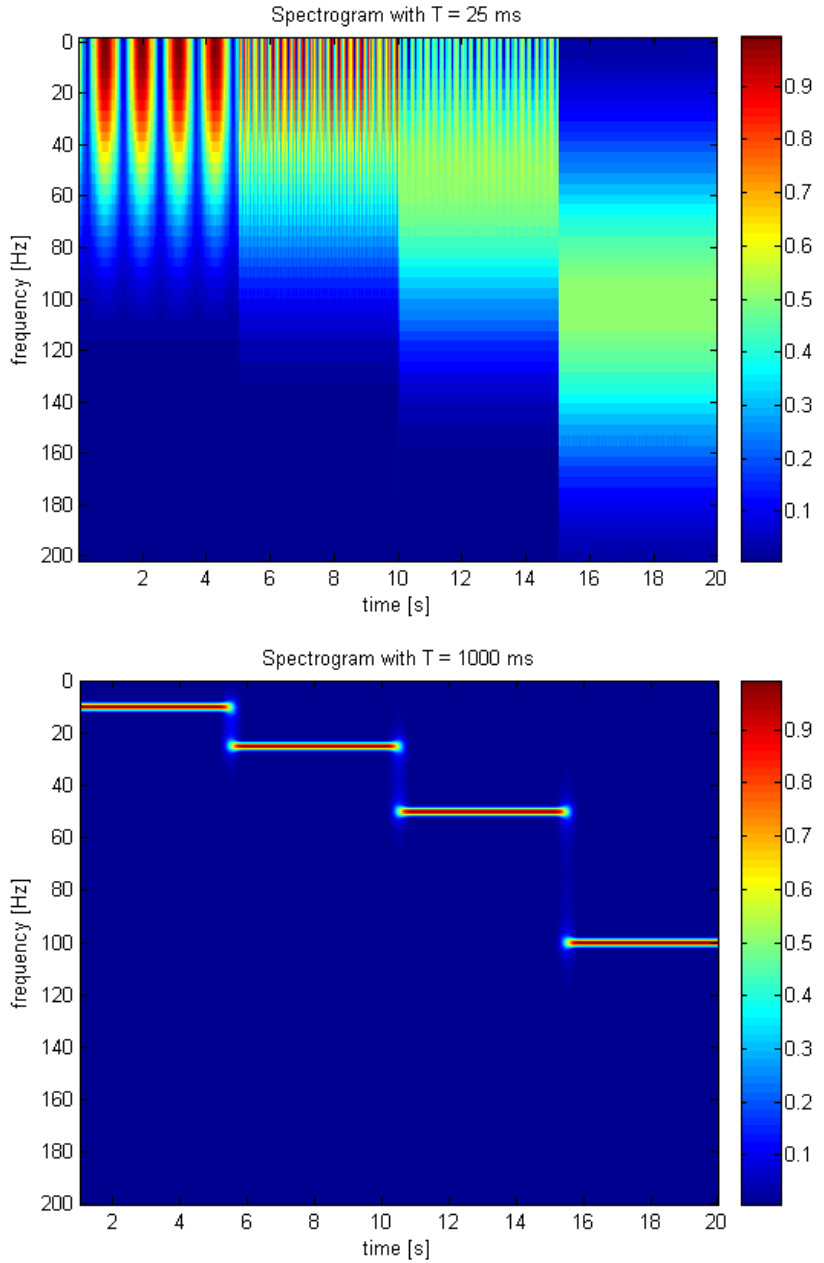
Short-time Fourier transform

- These pick up signals localized in time by repeatedly doing Fourier transforms on pieces of a function
- The discrete version breaks up the input function into blocks and does a Fourier transform on each block
- The continuous version does a Fourier transform on the input function multiplied by a “window function” with finite support, which slides along the axis (this is similar to convolution)



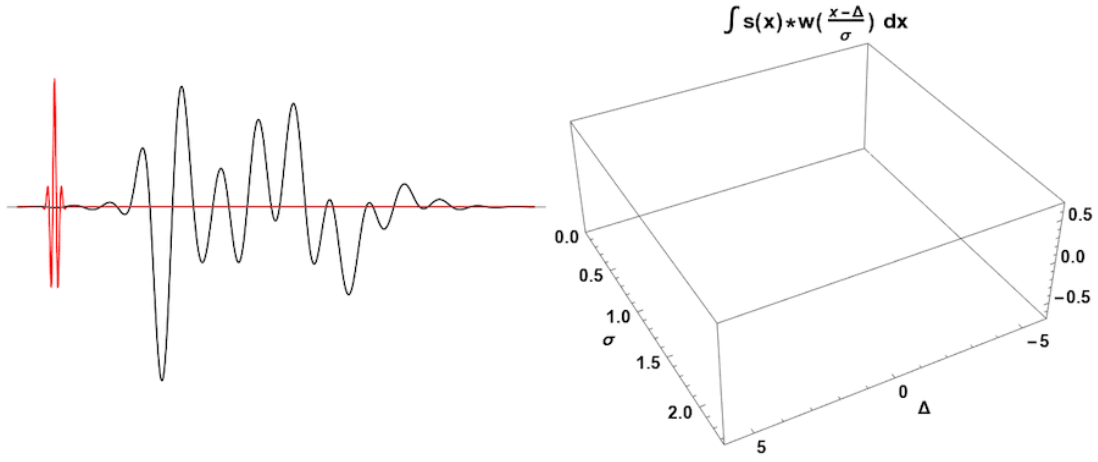
Window size in STFT

- Using a narrow window function enables accurate detection of when frequencies change, since you're finding frequencies over smaller areas of time
- Using a wide window function provides a good resolution in frequency, since many waves can fit within the window even if they have low frequencies
- However, with STFT, you can typically only get one or the other

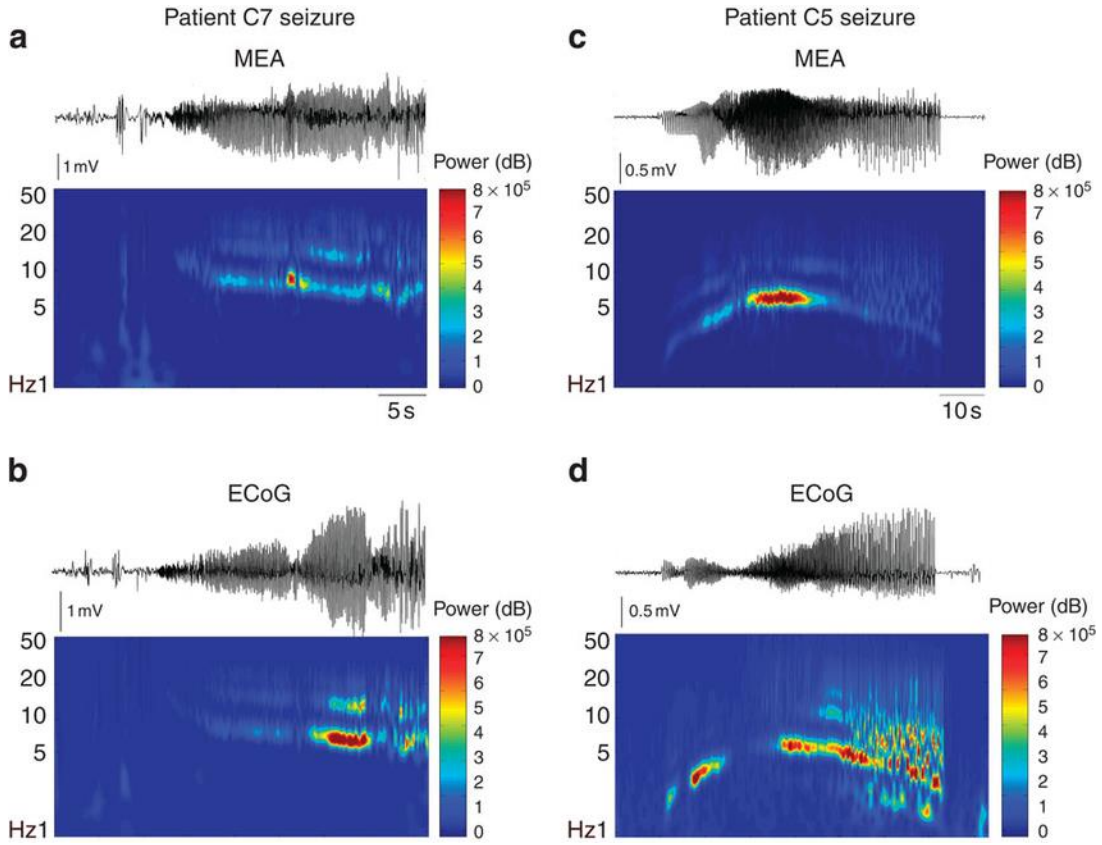


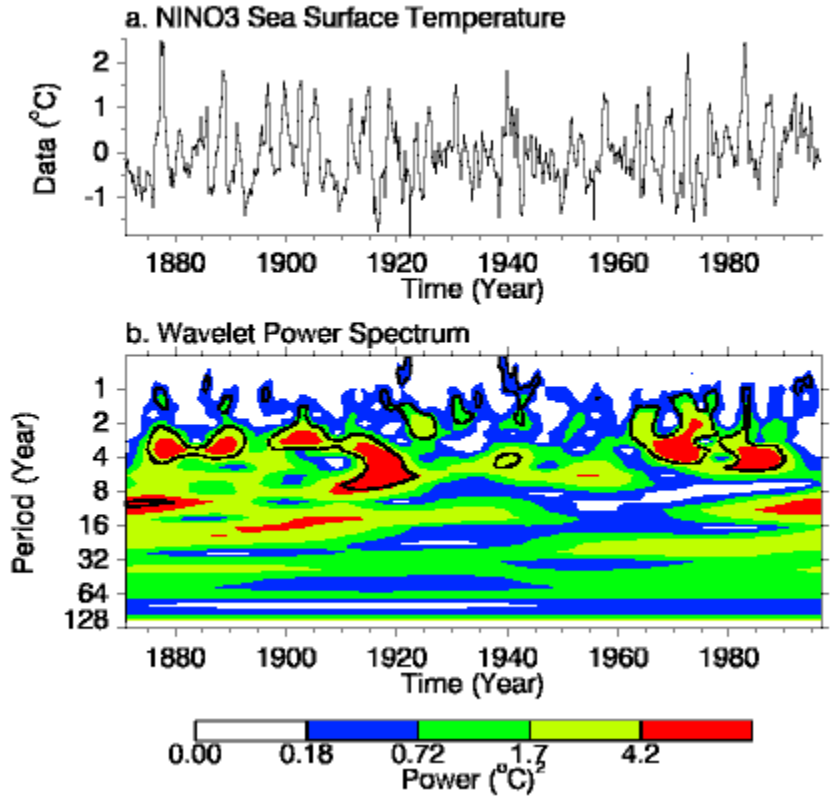
Wavelets: an introduction

- Wavelets are procedurally-generated functions with finite support, and are usually oscillatory
- A wavelet transform is similar to STFT in the sense that it features convolving an input function with a series of basis functions (a “family” of wavelets)
- Because the wavelets in a family are on different temporal scales (unlike the window in STFT), we can get good resolution in both time and frequency from them



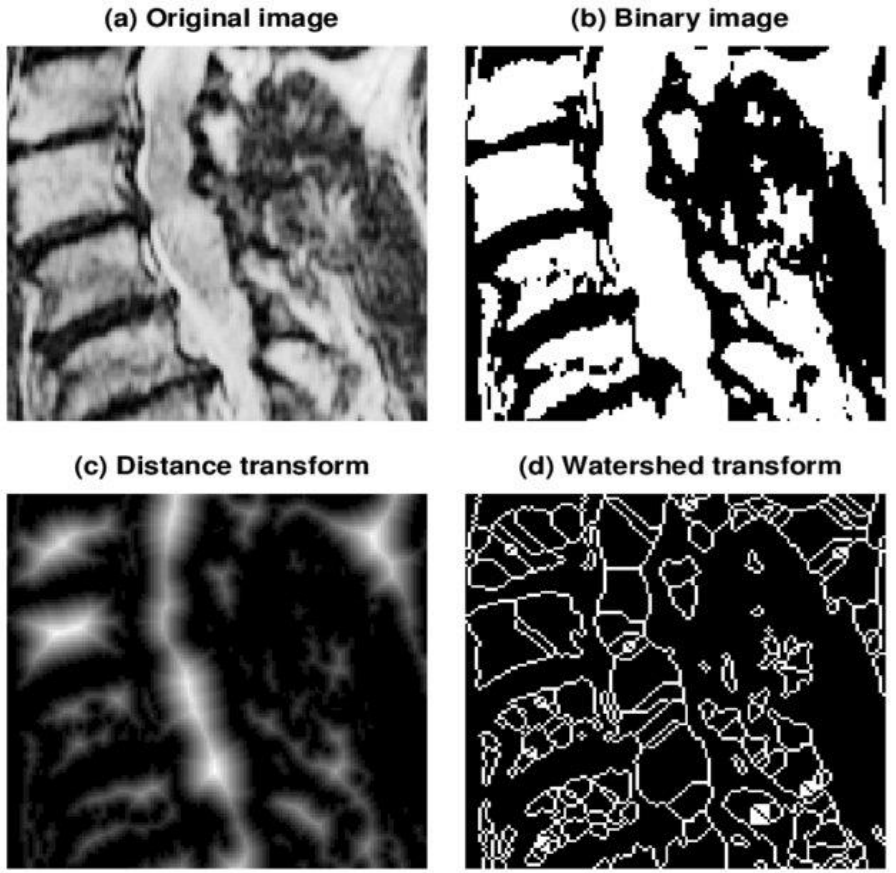
How wavelets can be used: epileptic seizures and sea surface temperature





Sidenote: how about finding patterns in 2D data?

- Wavelets have also been used for image segmentation and denoising; this often comes up in medical image processing
- Image segmentation can also be done using k-means
- Machine learning methods have recently been developed for this, such as U-Net (which is also part of the foundation of Midjourney)
- Building machine learning network architectures based on wavelets is an active research area

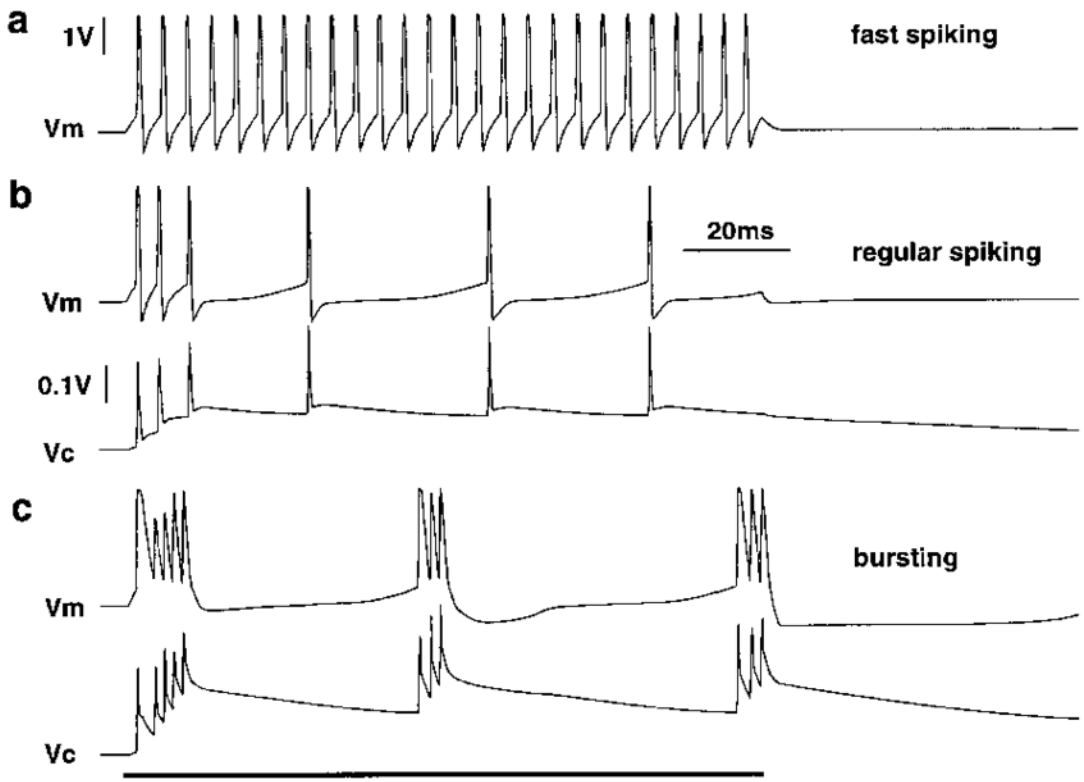


What if you have some time series data and want to build a dynamical system out of it?

Sparse identification of nonlinear dynamics

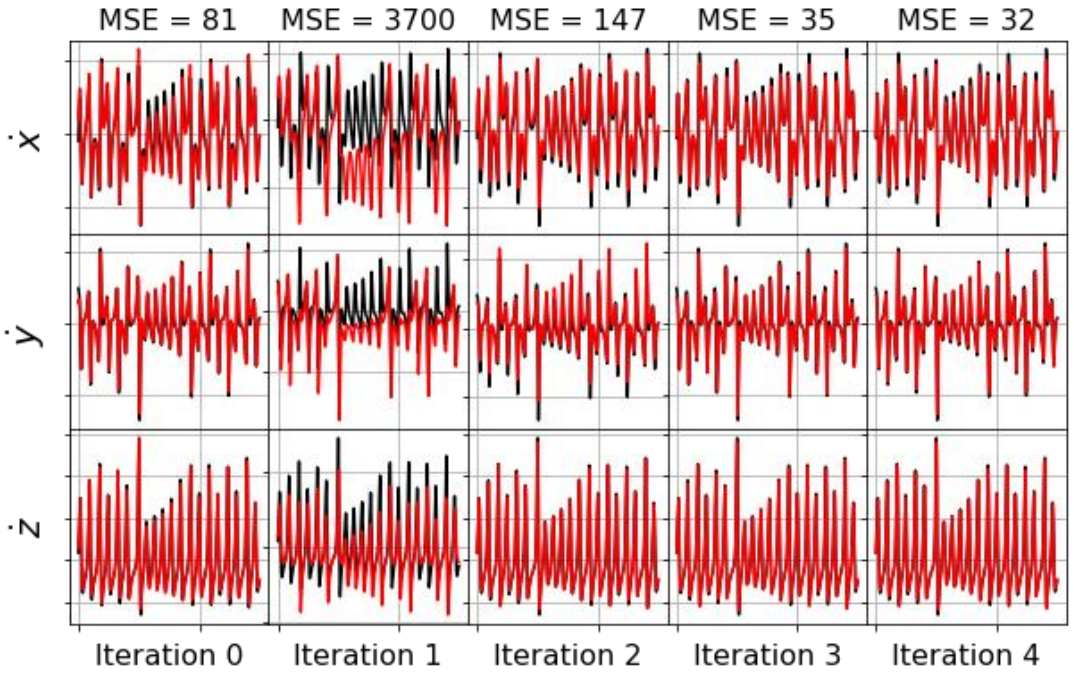
What does “sparse identification” mean in the context of dynamical systems?

- Complex systems with many interacting parts are all over the place, but deriving mechanistic models for them can be tough
- There potentially could be lots of different terms in a dynamical system model, e.g. x, x^2, xy, x/(y+k)
- However, most equations will only contain a few of these, making the terms in any given DE sparse in the overall space of terms

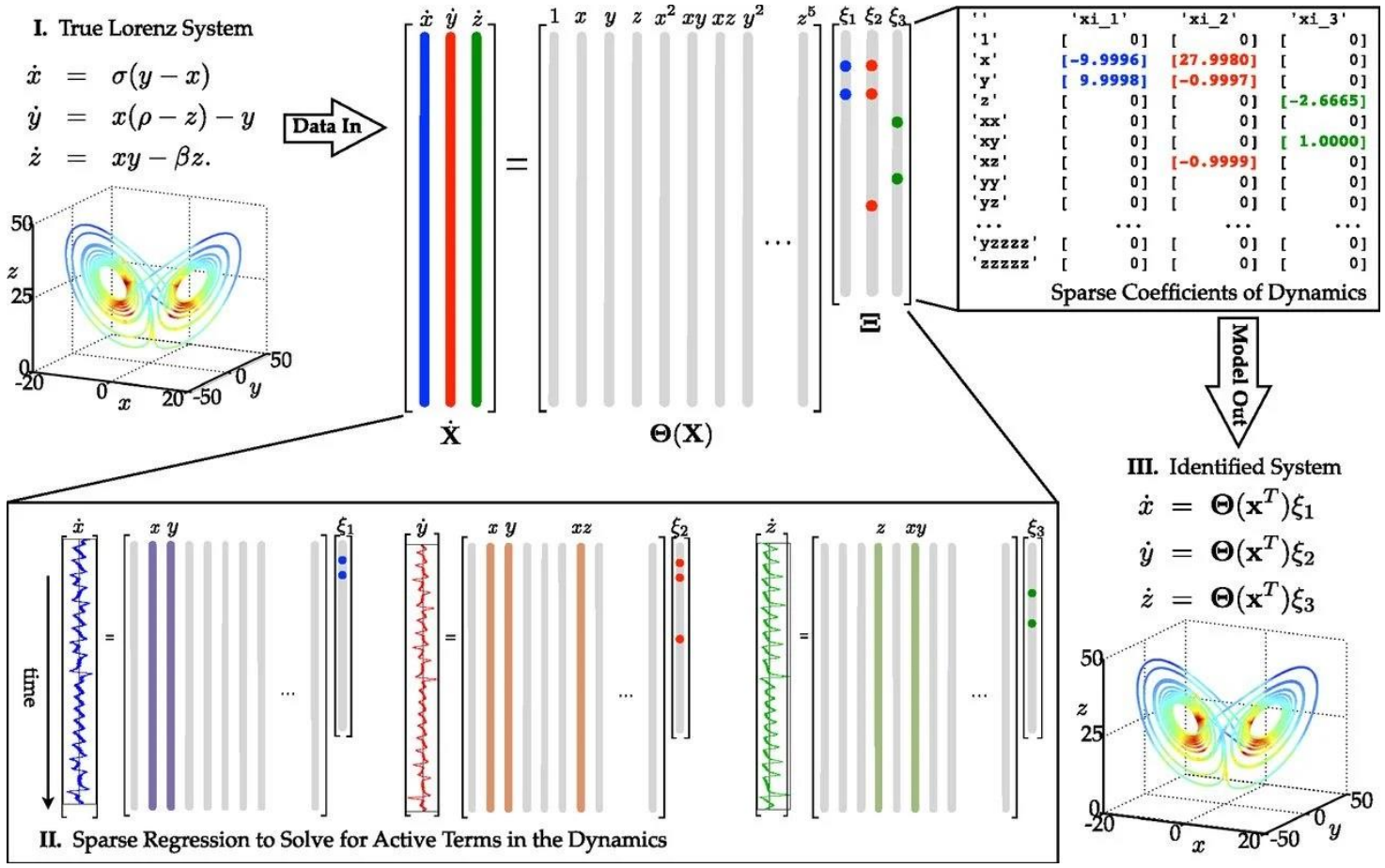


SINDy creates parsimonious dynamical systems from time series data

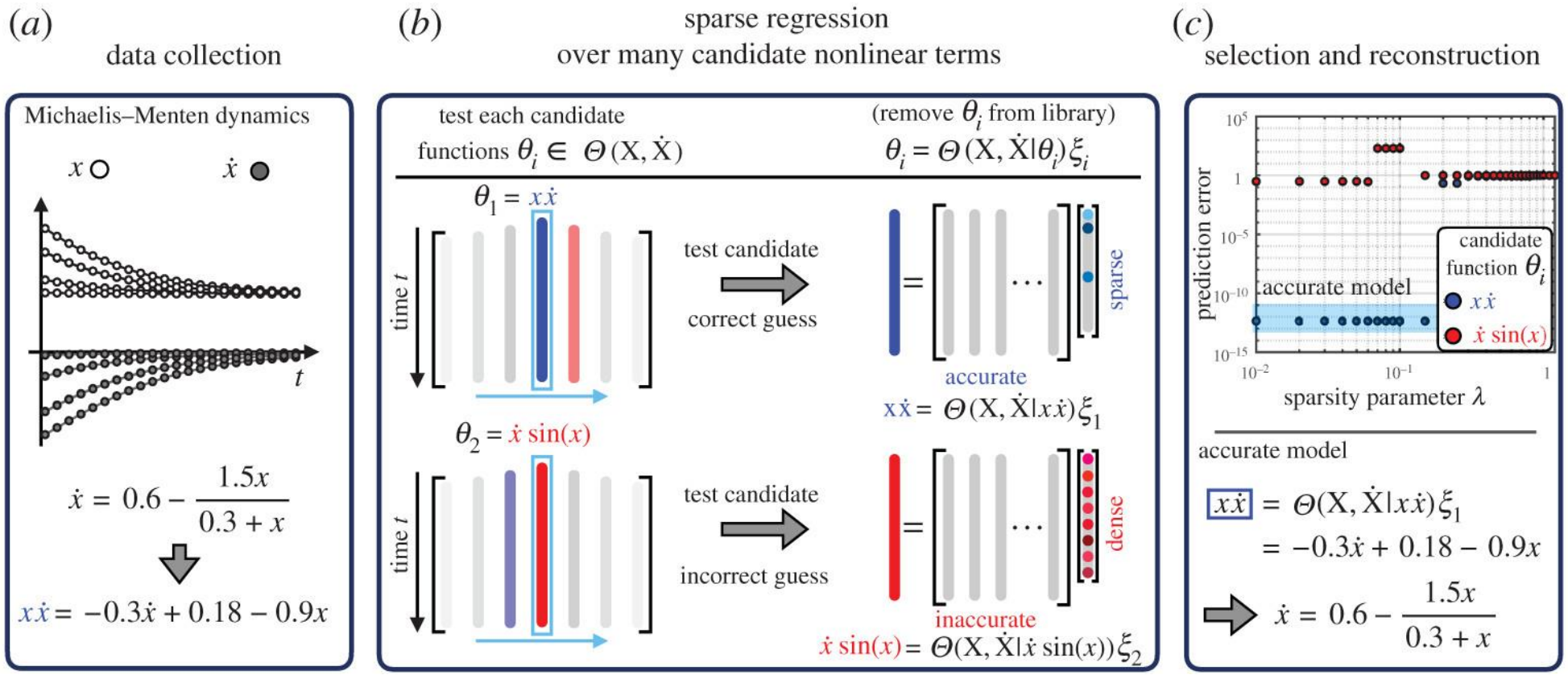
- To use SINDy, you need a multivariate time series and some terms that you think might be in the governing DEs for the time series
- SINDy uses Lasso to select a parsimonious set of terms to constitute the DEs, along with the terms' coefficients
- There's an implementation of it in Python (called PySINDy)



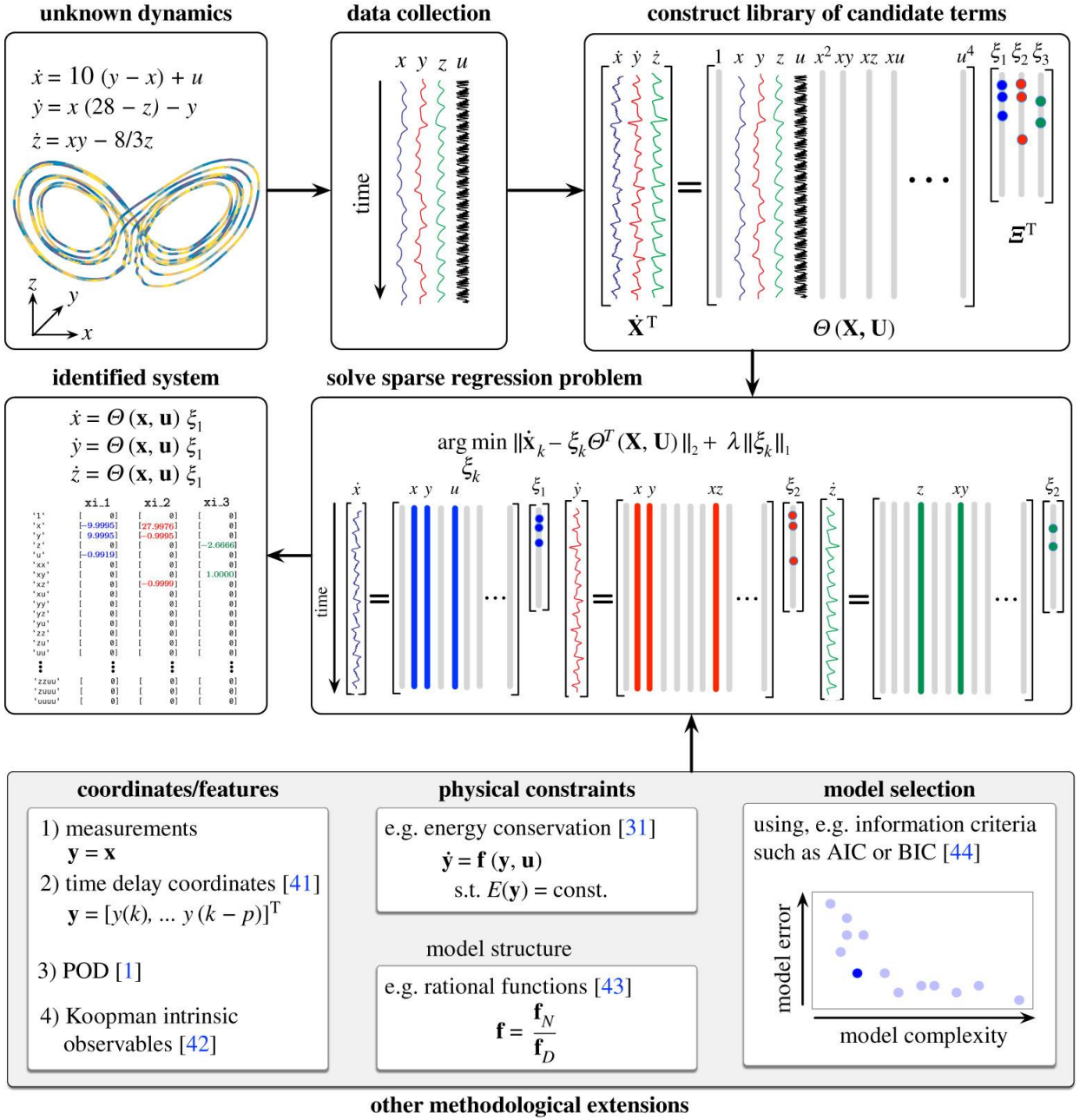
Schematic for how SINDy works



SINDy has been modified for use with rational functions…



…and functions with a control input (this has applications for problems such as aircraft control and optimal drug dosage)

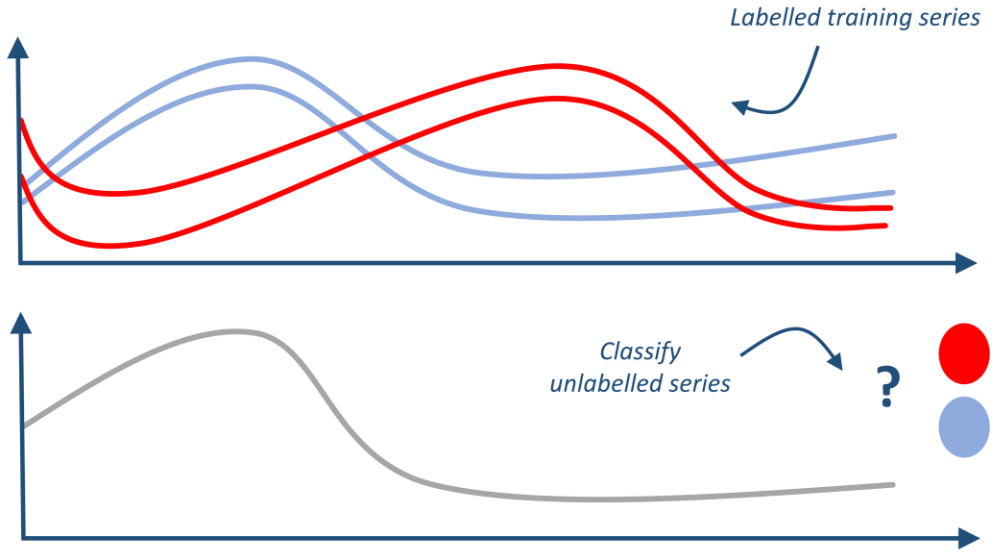


What if you have categories of time series, and want to know how you could push a given time series into a different category?

Time series counterfactuals

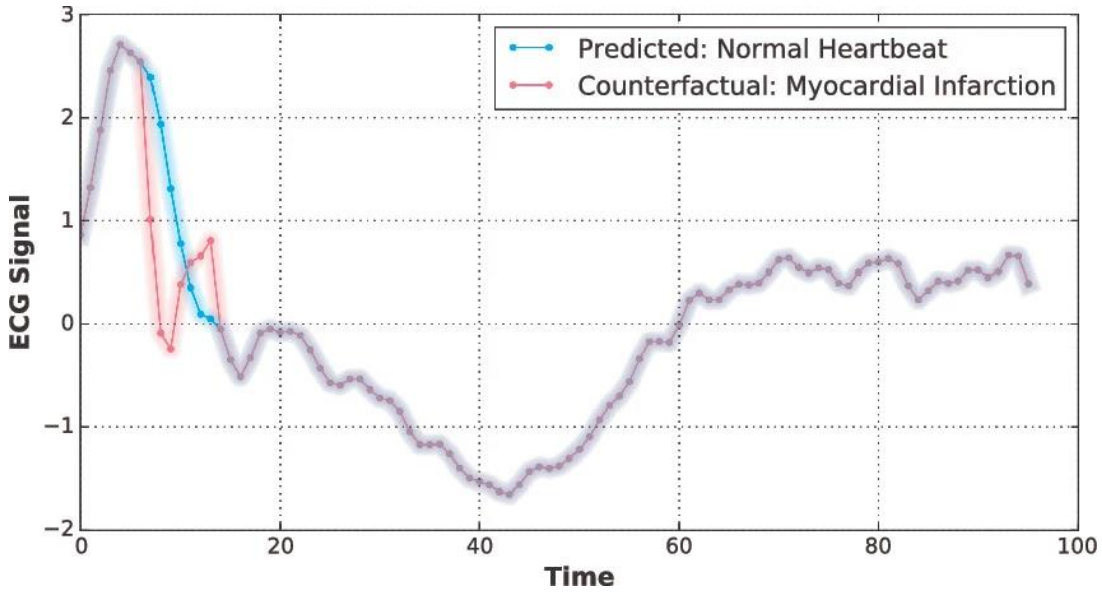
Categorizing time series

- This is mathematically similar to classification of individual points, but the objects being classified are (of course) full time series
- Time series classification can be a very high-dimensional problem, depending on length and temporal resolution
- Hence, it is typically done using machine learning methods



What if things were a little different?

- The principle behind time series counterfactuals is that if you change something about a given time series, it could be put into another category
- Often, methods for doing this ask what the smallest change that could recategorize a time series is
- This is typically done by replacing part of the time series in question with a different sequence of points



The UC Riverside time series archive

- This has been around since 2002 and contains many time series commonly used for testing; you can download them all
- There is also a multivariate time series archive (linked from the UCR archive)

ID	Type	Name	Train	Test	Class	Length	ED (w=0)	DTW (learned_w)	DTW (w=100)	Default rate	Data donor/editor
1	Image	Adiac	390	391	37	176	0.3887	0.3913 (3)	0.3964	0.9591	A. Jalba
2	Image	ArrowHead	36	175	3	251	0.2000	0.2000 (0)	0.2971	0.6057	L. Ye & E. Keogh
3	Spectro	Beef	30	30	5	470	0.3333	0.3333 (0)	0.3667	0.8000	K. Kemsley & A. Bagnall
4	Image	BeetleFly	20	20	2	512	0.2500	0.3000 (7)	0.3000	0.5000	J. Hills & A. Bagnall
5	Image	BirdChicken	20	20	2	512	0.4500	0.3000 (6)	0.2500	0.5000	J. Hills & A. Bagnall
6	Sensor	Car	60	60	4	577	0.2667	0.2333 (1)	0.2667	0.6833	J. Gao
7	Simulated	CBF	30	900	3	128	0.1478	0.0044 (11)	0.0033	0.6644	N. Saito
8	Sensor	ChlorineConcentration	467	3840	3	166	0.3500	0.3500 (0)	0.3516	0.4674	L. Li & C. Faloutsos
9	Sensor	CinCECGTorso	40	1380	4	1639	0.1029	0.0696 (1)	0.3493	0.7464	physionet.org
10	Spectro	Coffee	28	28	2	286	0.0000	0.0000 (0)	0.0000	0.4643	K, Kemsley & A. Bagnall
11	Device	Computers	250	250	2	720	0.4240	0.3800 (12)	0.3000	0.5000	J. Lines & A. Bagnall
12	Motion	CricketX	390	390	12	300	0.4231	0.2282 (10)	0.2462	0.8974	A. Mueen & E. Keogh

There are many time series counterfactual methods, each with their own advantages

- Ates et el. 2021, “Counterfactual Explanations for Multivariate Time Series”
- Delaney et al. 2021, “Instance-Based Counterfactual Explanations for Time Series Classification”
- ZD Wang et al. 2024, “Glacier: guided locally constrained counterfactual explanations for time series classification”
- Bahri et al. 2024, “Discord-based counterfactual explanations for time series classification”
- Maybe you can come up with something yourself