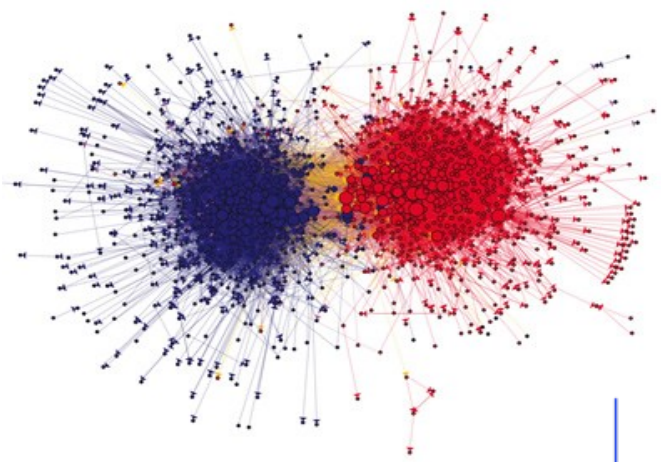
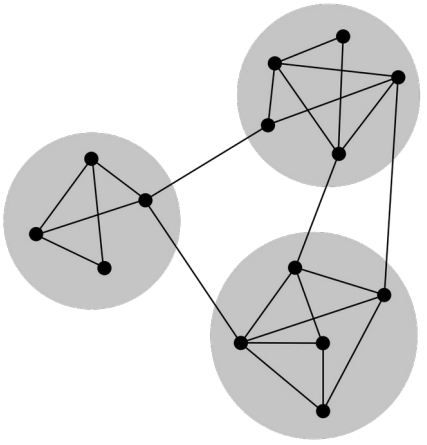
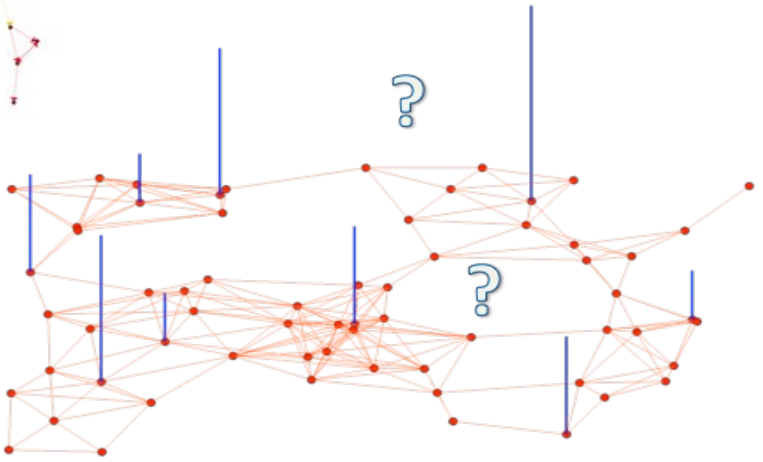
Machine learning & graphs

supervised, semi-supervised and unsupervised learning

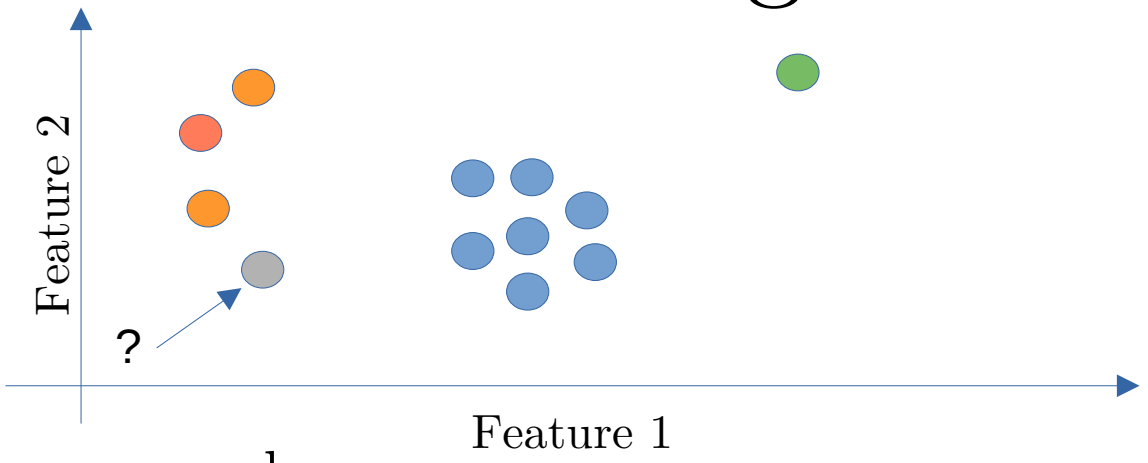






- Graph based methods: label propagation
  https://scikit-learn.org/stable/modules/semi_supervised.html#label-propagation
- Graphs and Spectral clustering:
  https://www.cs.cmu.edu/~aarti/Class/10701/readings/Luxburg06_TR.pdf

A first step towards graphs: K-nearest neighbors



To classify a new sample:

- Find the k closest samples in the feature space
- Assign the class by majority voting from the k-nn

Class of (first) nearest neighbor of sample x: $C_1(\mathbf{x})$

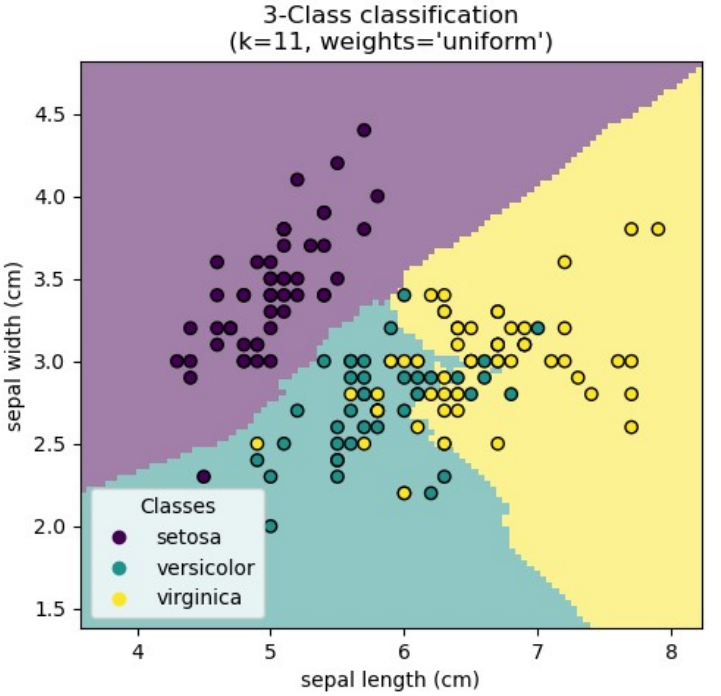
$$\hat{y} = mode\{C_1(\mathbf{x}), C_2(\mathbf{x}), ..., C_k(\mathbf{x})\}$$

Mode: the most common number that appears in your set of data.

K-nearest neighbors

K-nn is a nonlinear classifier: the class boundary is not linear.

It can be good and bad...



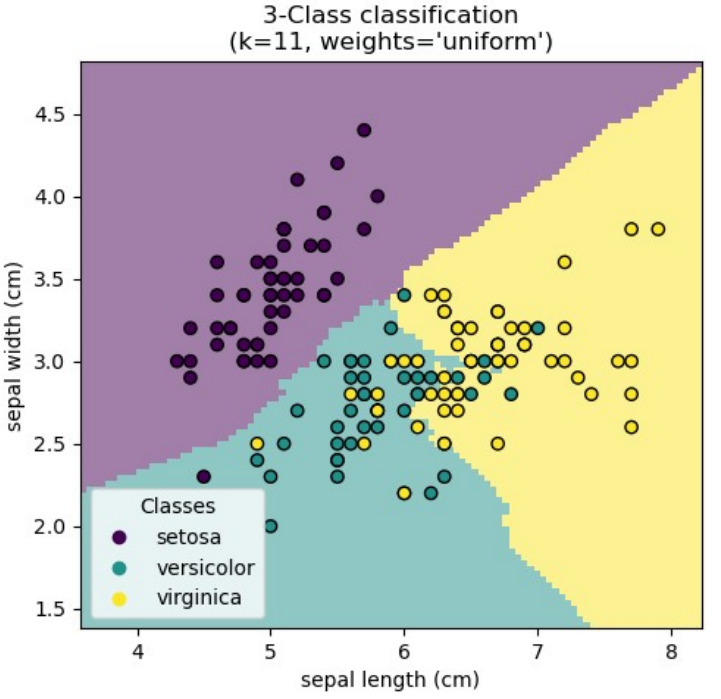
Example from scikit-learn

- Points are labelled data,
- Colored areas: class of a new point if in this area

Nearest Neighbors ?

Nearest neighbors: we need to define a distance between samples.
$\rightarrow$ See last lecture on clustering

For large datasets, computing all the distances can take time (compared to logistic regression).



Example from scikit-learn

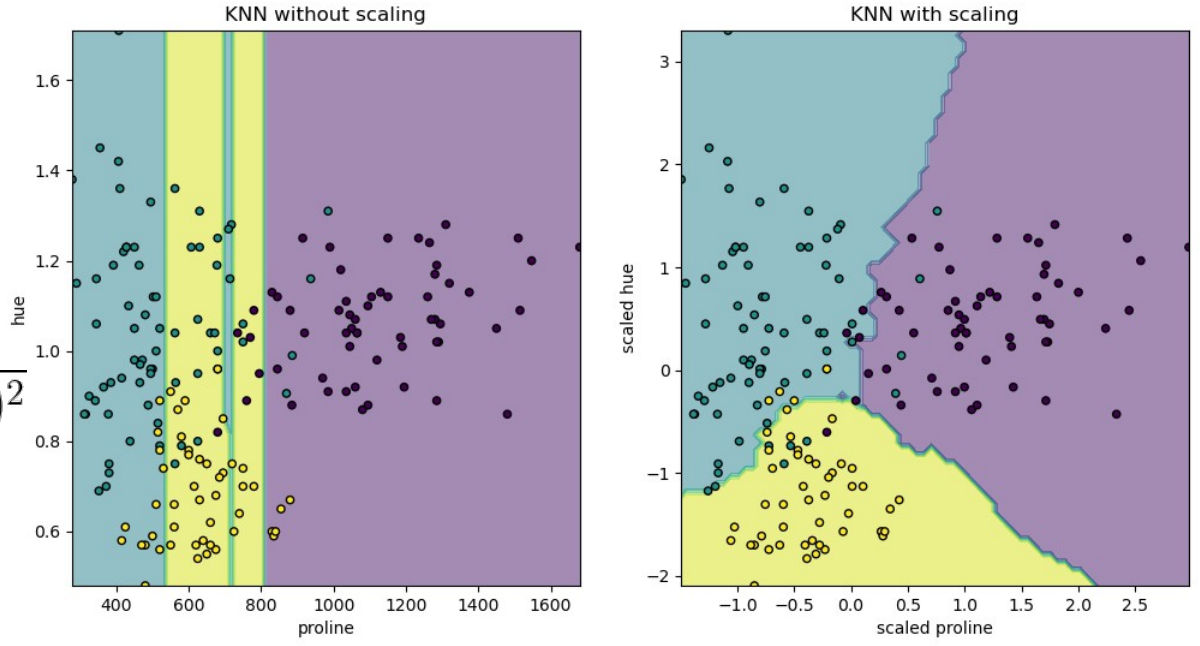
Importance of feature scaling

Scaling or standardization or z-score $z = \frac{x - \mu}{\sigma}$

μ mean, σ std deviation

Euclidean distance:

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$



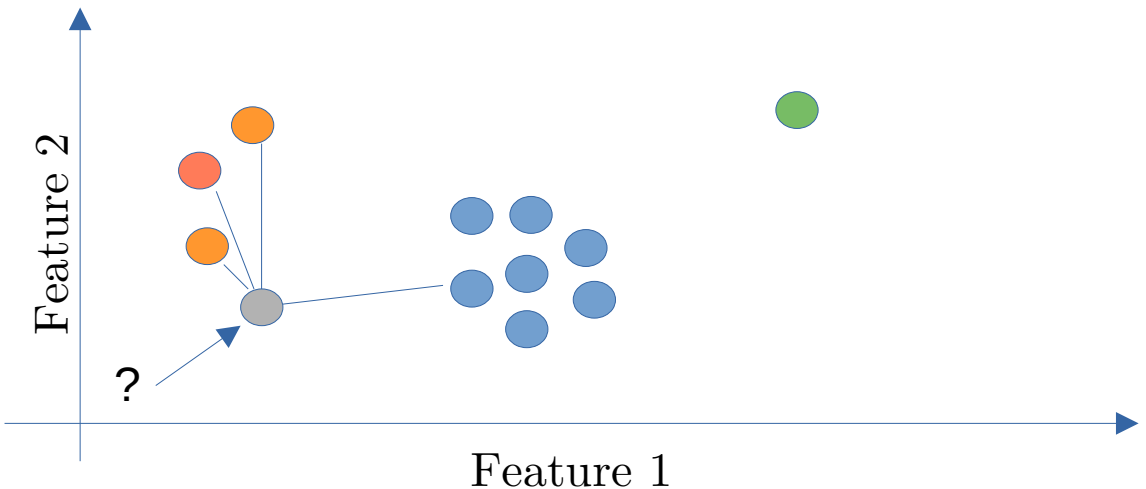
How the distance is measured is important!

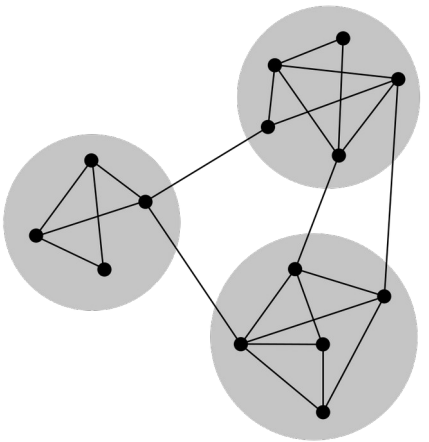
From neighbors to graphs

- Looking at the neighbors is useful. Why not looking at the neighbors of neighbors? → let us make a graph!

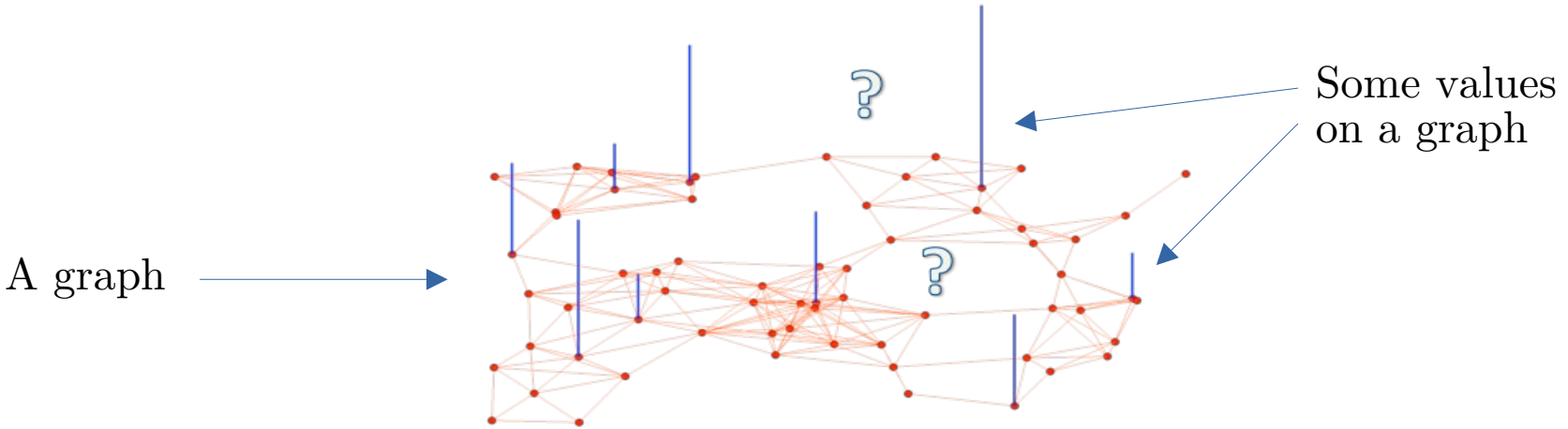
- A graph is a mathematical object made of vertices (nodes) and edges.

In our setting:
- a node is a sample
- An edge encode the similarity between samples, for examples if samples are close in the feature space.





Label propagation



- Problem: only a few samples are labeled (semi-supervised learning)
- Idea: neighbors are similar and should have the same label
- Solution: propagate the labels over the graph of nearest neighbors

How?

But first:
What is a graph?

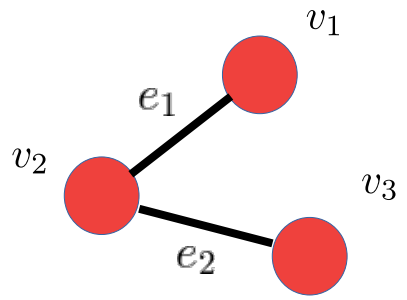
Graph definition

Vertices or nodes

$$V = \{v_1, \ldots, v_N\}$$

Edges or links

$$E = \{e_1, \ldots, e_M\}$$



Adjacency matrix

$$A = \begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} v_1 \\ v_2 \\ v_3 \end{matrix}$$

$$\mathbf{A}(i,j) = \begin{cases} +1 & \text{if there is an edge } (v_i, v_j) \text{ or } (v_j, v_i) \in E \\ 0 & \text{otherwise} \end{cases}$$

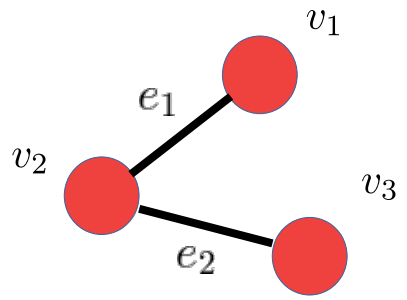
Graph definition

Vertices or nodes

$$V = \{v_1, \ldots, v_N\}$$

Edges or links

$$E = \{e_1, \ldots, e_M\}$$



Weight Matrix:

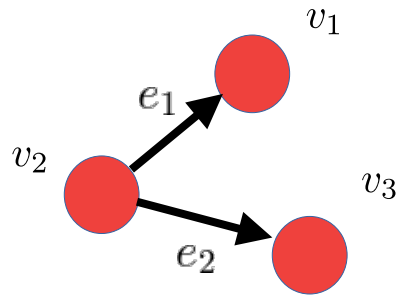
$$W = \begin{pmatrix} 0 & w_{12} & 0 \\ w_{21} & 0 & w_{23} \\ 0 & w_{32} & 0 \end{pmatrix}$$

W(i,j) is the weight ("strength") of the edge between *i,j* (if any).
W symmetric with positive entries

Directed Graph

Weight Matrix:

$$W = \begin{pmatrix} 0 & 0 & 0 \\ w_{21} & 0 & w_{23} \\ 0 & 0 & 0 \end{pmatrix}$$



There is a connection from v_2 to v_1 but not from v_1 to v_2. (example: hyperlinks in webpages)

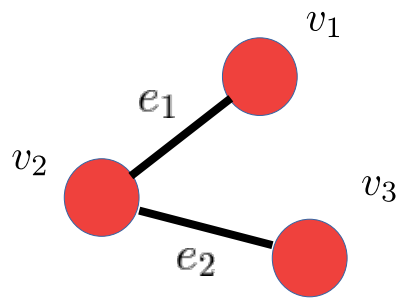
W with positive entries but not symmetric. $W \neq W^T$

Transition matrix

- Adjacency matrix with normalized columns (or rows)

$$T_{ij} = \frac{a_{ij}}{\sum_i a_{ij}} = \frac{a_{ij}}{d_j}$$

Degree of node j



Probability to jump from one node to another

Transition matrix

$$T = \begin{pmatrix} 0 & 1/2 & 0 \\ 1 & 0 & 1 \\ 0 & 1/2 & 0 \end{pmatrix} \begin{matrix} v_1 \\ v_2 \\ v_3 \end{matrix}$$

$v_1 \quad v_2 \quad v_3$

- Not symmetric

- Degree matrix D: diagonal matrix with node degree on the diagonal,

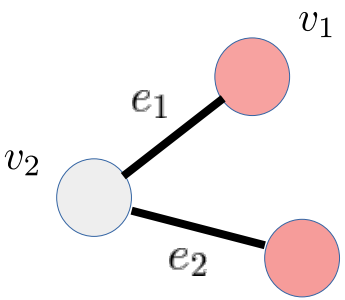
$$D_{ii} = \sum_j w_{ij} = d_i$$ Degree of node i

Transition matrix or **random walk matrix**

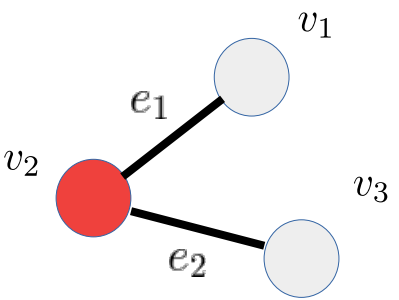
$$T = W D^{-1}$$

$$T_{ij} = [W D^{-1}]_{ij} = \frac{w_{ij}}{d_j}$$

Propagation on a graph



$$v_3 \begin{pmatrix} 1/2 \\ 0 \\ 1/2 \end{pmatrix} = \overset{\text{T}}{\begin{pmatrix} 0 & 1/2 & 0 \\ 1 & 0 & 1 \\ 0 & 1/2 & 0 \end{pmatrix}} \overset{\text{Y}}{\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}}$$

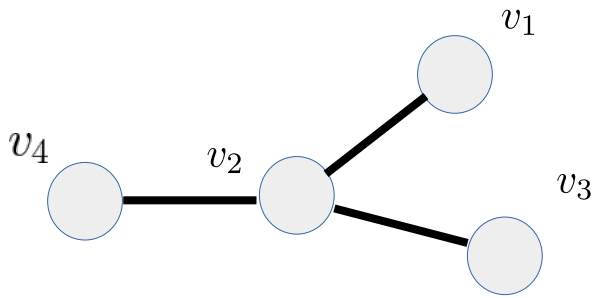


Split equally between node 1 and 3

Transition matrix

Value 1 on node 2

- Applying T to the vector of node values propagate the values over the graph
- Probabilistic interpretation of T^nY: probability to reach an node (starting from v_2) after exactly n steps

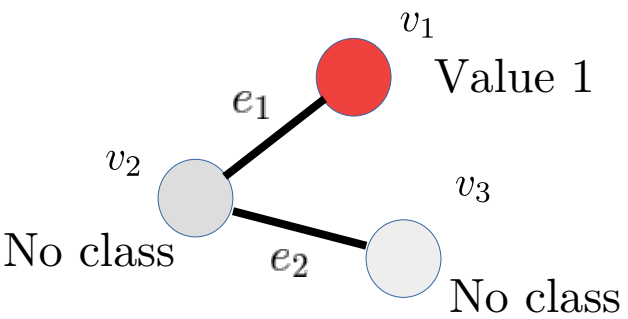


Adjacency matrix ?
Transition matrix ?

Propagation of labels

$$Y = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$ nodes

node 1 has value 1



Initial values Y_0
Propagate: $Y_{t+1} = TY_t$

Ok but we need to keep the labelled nodes at their initial value!

→ Propagate, normalize and clamp the labelled nodes:
$Y_{t+1} = n(TY_t) + Y_0$
Under some conditions on T it converges

From: Learning from Labeled and Unlabeled Data with Label Propagation, X. Zhu and Z. Ghahramani, 2002.

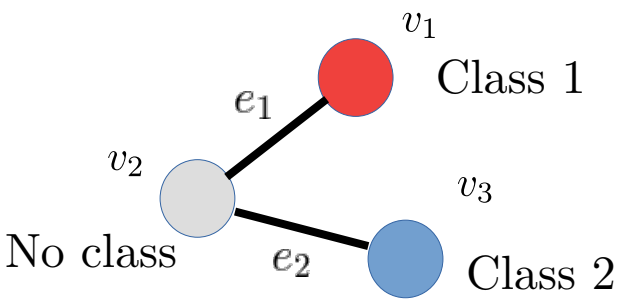
Propagation of labels

Several classes: Y is a matrix, each column is a label

$$Y = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$$

Classes

nodes



Probability of node 3 to be of class 2

All labels are diffused at the same time

Alternative propagation

Zhou, D., Bousquet, O., Lal, T., Weston, J., & Schölkopf, B. (2003). Learning with local and global consistency. Advances in neural information processing systems, 16.

1. Form the affinity matrix W defined by $W_{ij} = \exp(-\|x_i - x_j\|^2/2\sigma^2)$ if $i \neq j$ and $W_{ii} = 0$.
2. Construct the matrix $S = D^{-1/2}WD^{-1/2}$ in which D is a diagonal matrix with its (i,i)-element equal to the sum of the i-th row of W.
3. Iterate $F(t+1) = \alpha SF(t) + (1-\alpha)Y$ until convergence, where α is a parameter in $(0,1)$.
4. Let F^* denote the limit of the sequence $\{F(t)\}$. Label each point x_i as a label $y_i = \arg\max_{j \leq c} F^*_{ij}$.

$D^{-1/2}WD^{-1/2}$ Symmetrized transition matrix

PageRank

Famous Google PageRank, for Larry Page, Sergey Brin (1998)

Idea:

- Explore webpages by following the hyperlinks randomly
- Add +1 when a page is visited
- To avoid being stuck on pages without links, add a probability to randomly jump to any other page.

This is equivalent to a random walk on the graph with

Iterate $Y_{t+1}=MY_t$

$$M = (1-p)T + pB$$

Tuning parameter

Transition matrix

Random jump anywhere matrix

$$B = \frac{1}{n} \cdot \begin{bmatrix} 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

The maths behind

These methods consist in applying a matrix many times to an initial vector.
$Z = M^n Y$
This converge when $n \to \infty$ to a unique solution, the "steady state" S:
$S = M S$. So *S is an eigenvector of M with eigenvalue 1*

There must be some conditions on M such that this is possible!
→ Yes, see Perron-Frobenius theorem

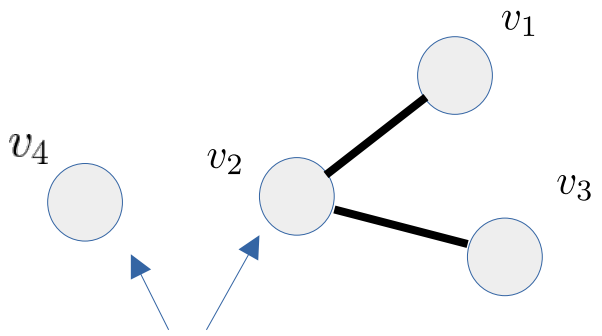
Idea:
P-F theorem tells us that the largest eigenvalue is 1, the others are smaller but positive.
→ we can use the power method to find S. Because for any eigenvector U:
$M^n U = \alpha^n U$ and $\alpha < 1$ except for S
So taking the power of M on a random vector will give a good approximation of S and will converge to S.

The maths behind

* Property on M for P-F Theorem: the matrix M is irreducible (M: weight matrix of a strongly connected graph, i.e. you can go from one node to any other node in the network by following connections)

+ in a transition matrix, rows (or columns) sum to 1 and

the Perron–Frobenius eigenvalue r satisfies the inequalities $\min_i \sum_j a_{ij} \leq r \leq \max_i \sum_j a_{ij}$.



Disconnected graph, made of 2 parts. Diffusion cannot spread to all the nodes!

The maths behind

Remark:
If not the transition matrix, and max eigenvalue r is above 1 we can still get the eigenvector associated to r :
normalize after each diffusion to avoid explosion: the power iteration method.
$\rightarrow$ at step n, divide M^nU by the norm $||M^nU||$

Power iteration method

$$b_{k+1} = \frac{Ab_k}{\|Ab_k\|}$$

```python
import numpy as np

def power_iteration(A, num_iterations: int):
    # Ideally choose a random vector
    # To decrease the chance that our vector
    # Is orthogonal to the eigenvector
    b_k = np.random.rand(A.shape[1])

    for _ in range(num_iterations):
        # calculate the matrix-by-vector product Ab
        b_k1 = np.dot(A, b_k)

        # calculate the norm
        b_k1_norm = np.linalg.norm(b_k1)

        # re normalize the vector
        b_k = b_k1 / b_k1_norm

    return b_k
```

Graph weights

2 cases:

- The data contains a graph already (social network, energy or transportation network ...)
  $\rightarrow$ the weight matrix is given or made from the list of connections

- A graph is computed from the data (samples x features matrix)
  $\rightarrow$ similarity measure to connect the nodes (samples)

Graph from features

Define:

1) Distance d(x,y) between 2 samples in the feature space (Euclidean or other)
2) Similarity s(x,y) from distance, ex. $s(x,y) = \exp(-d(x,y)/\sigma^2)$

Distance increase $\leftrightarrow$ similarity & edge weight decrease

Graph from features

Compute the distance matrix (pairwise distance between all points)
Ex: *sklearn.metrics.pairwise_distances*
Two main ways of creating connections:

- k-nearest neighbors (sort distances and take first k entries)

Or

- ε-radius neighbors (neighbors in a ball of radius ε, connect nodes with dist $<\varepsilon$)