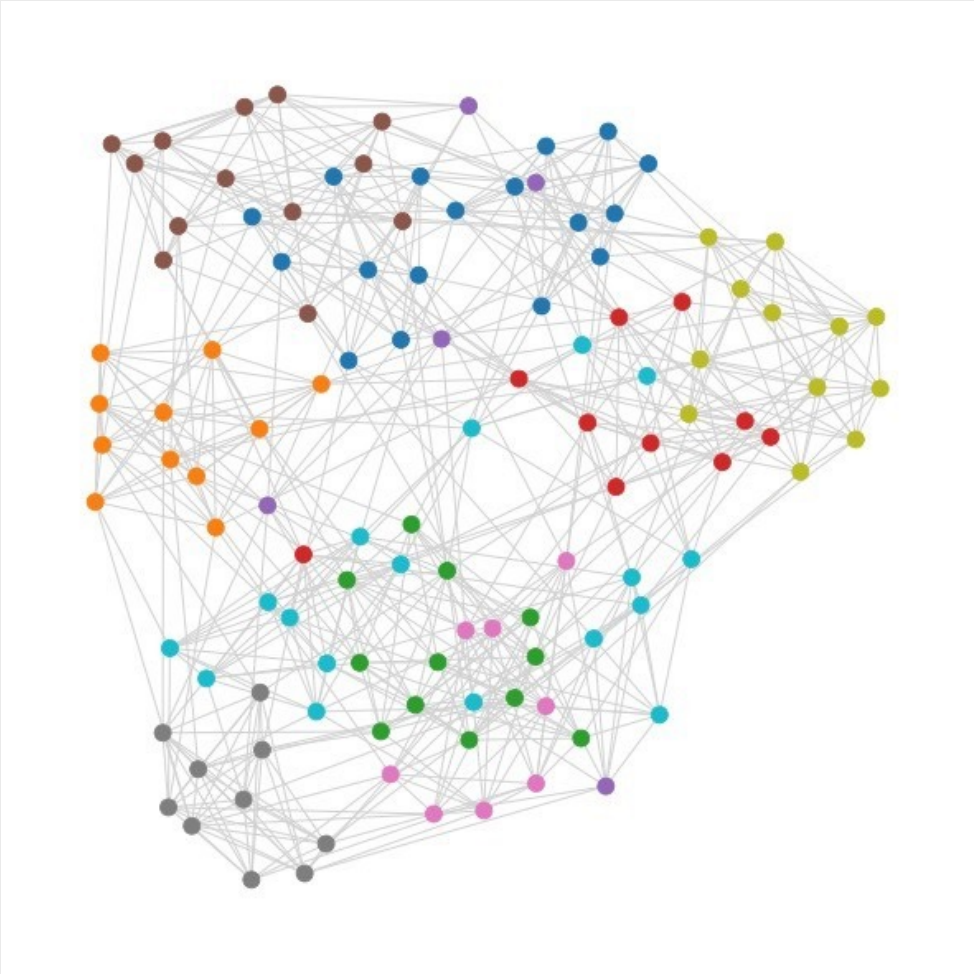
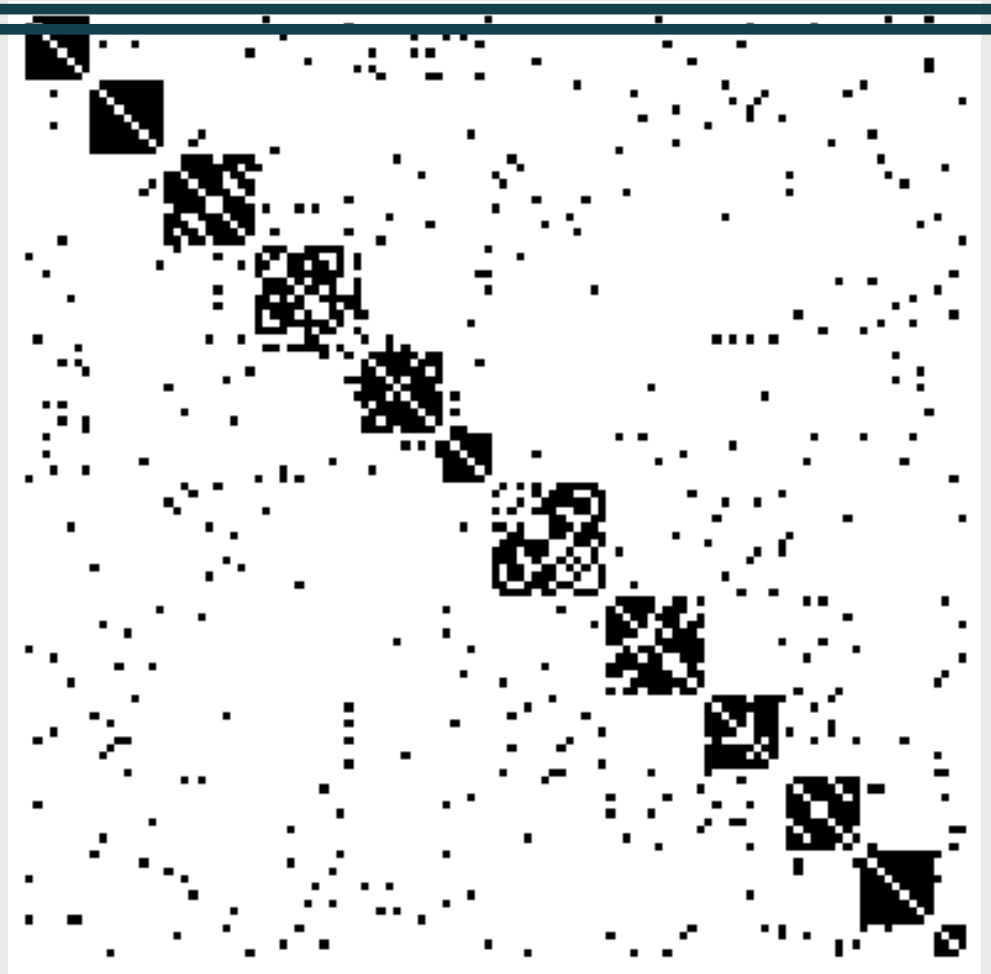
Node embeddings

We can define each node by its connections



Each node is represented by a vector



You can use it to predict something about the node:

- but that would mean thousands or millions of parameters!
- and it only provides information at the local level

Node embeddings

Idea: Create a low(er)-dimension representation of the node

Use those embeddings for:

1) Node classification

X = Embedding

Y = whatever we want to predict

2) Link prediction

X = combination of the two embeddings X_1 and X_2 (e.g. $X_1 * X_2$, or np.abs(X_1 - X_2))

Y = link/no link

Node embeddings

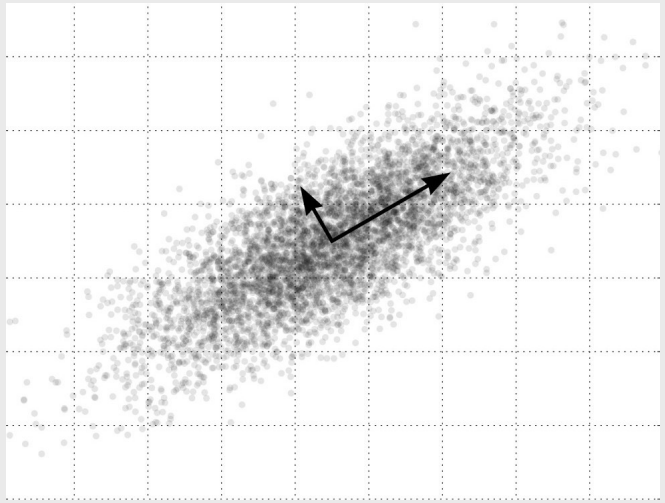
Similar nodes should have a similar representation

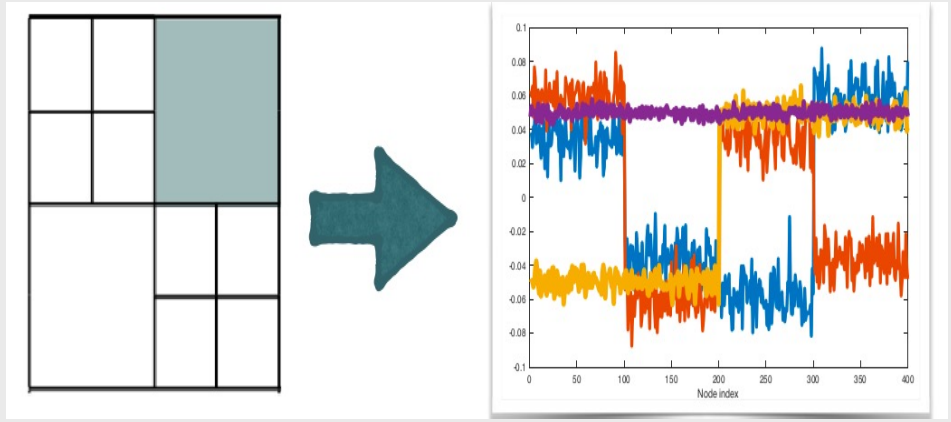
What do we mean with similar nodes?

- Nodes with the same neighborhood (e.g. being part of the same community, or being connected)
- Nodes with the same role (e.g. serving as bridges)
- Nodes with the same metadata (e.g. criminal nodes in a financial network)

Option 1: Spectral methods

Related to characteristic eigenvectors of matrices associated with the network





Principal Component Analysis (PCA)

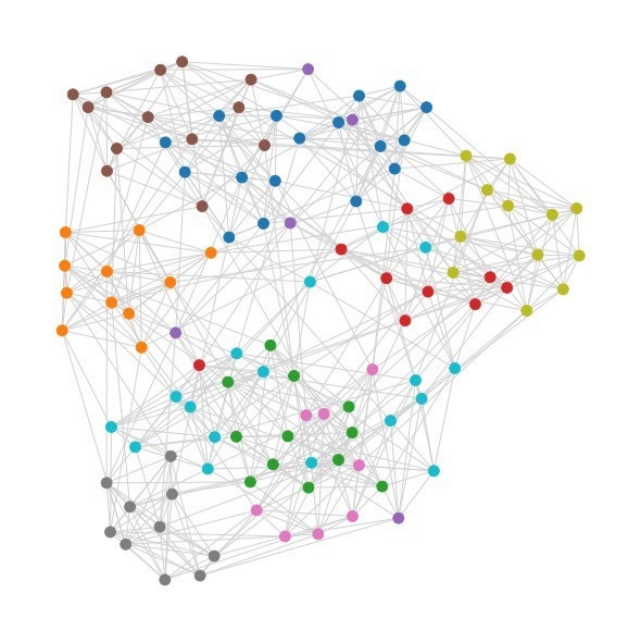
- Correlated variables → Linear combination of orthogonal variables
- Eigenvectors corresponding to the largest k eigenvalues of A^TA (centered)

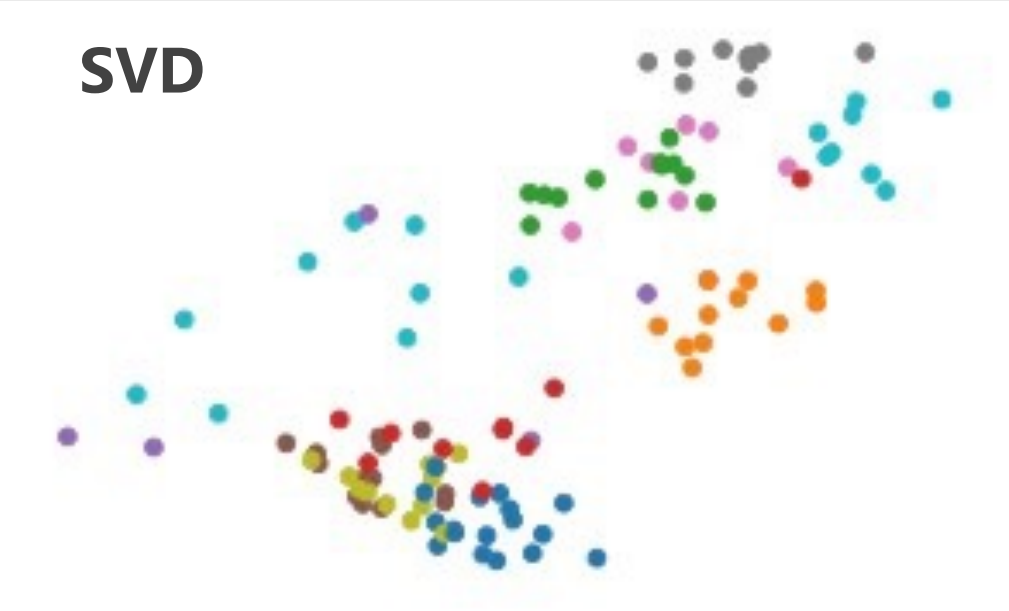
Singular Value Decomposition (SVD)

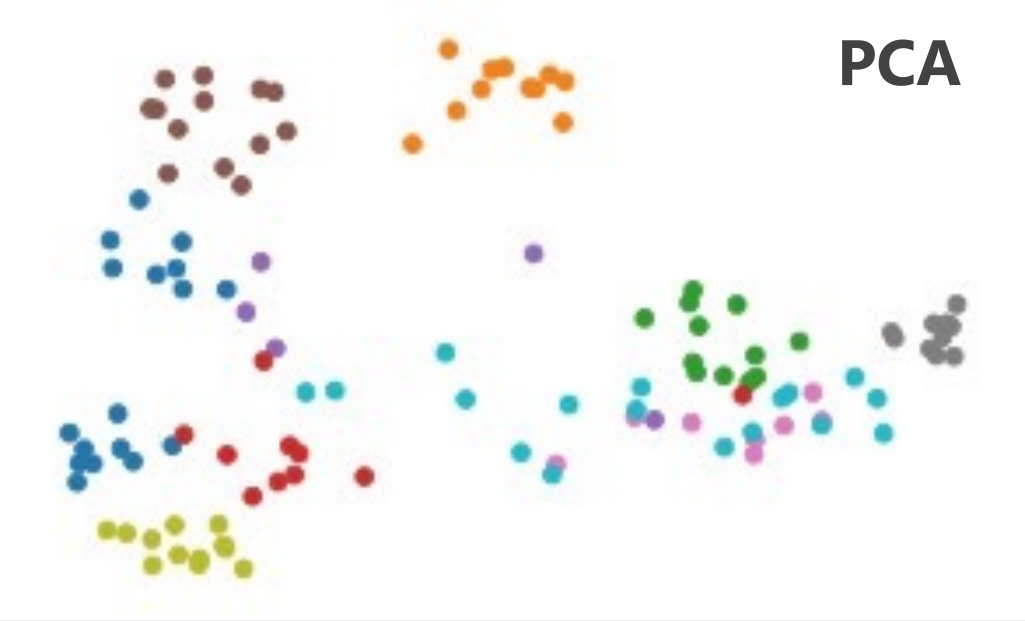
- Eigenvectors corresponding to the largest k eigenvalues of A^TA (uncentered)

Laplacian Eigenmaps (LE)

- Assumes that the nodes lie on a low-dimensional (with some constrains)
- Tries to find an embedding that minimizes the distances between connected nodes
- That mapping is created by the eigenvectors corresponding to the smallest k eigenvectors of the normalized Laplacian matrix $D^{-1}(D-A)$



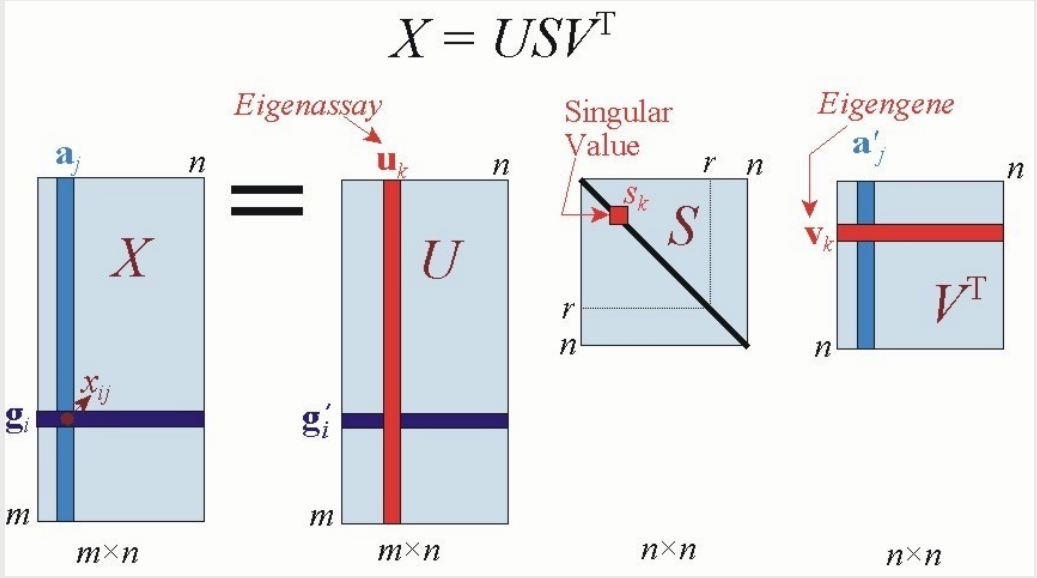

SVD


PCA

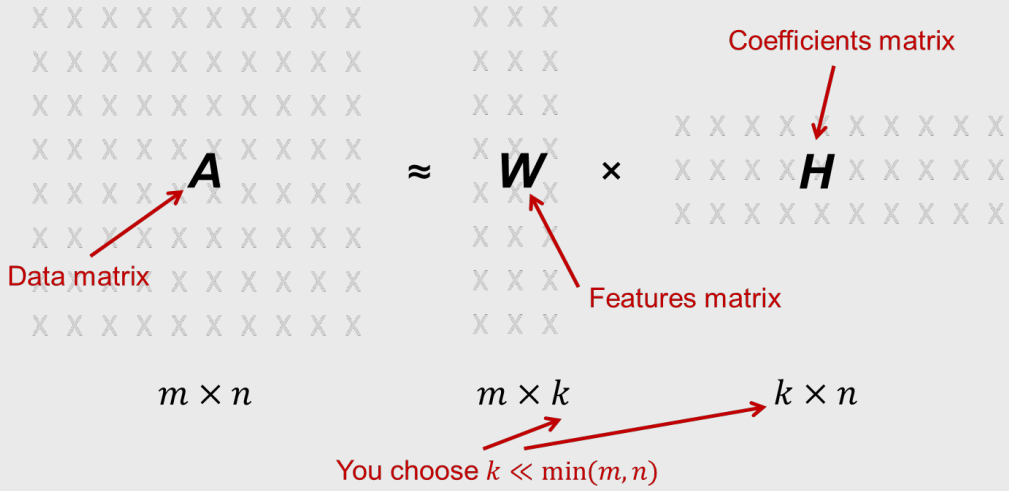

LE

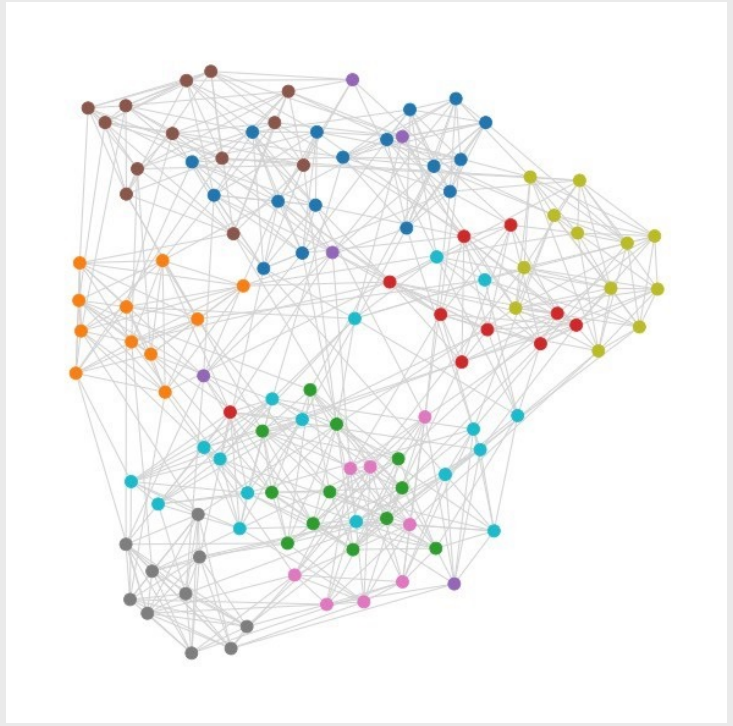
Option 2: Non-negative matrix factorization

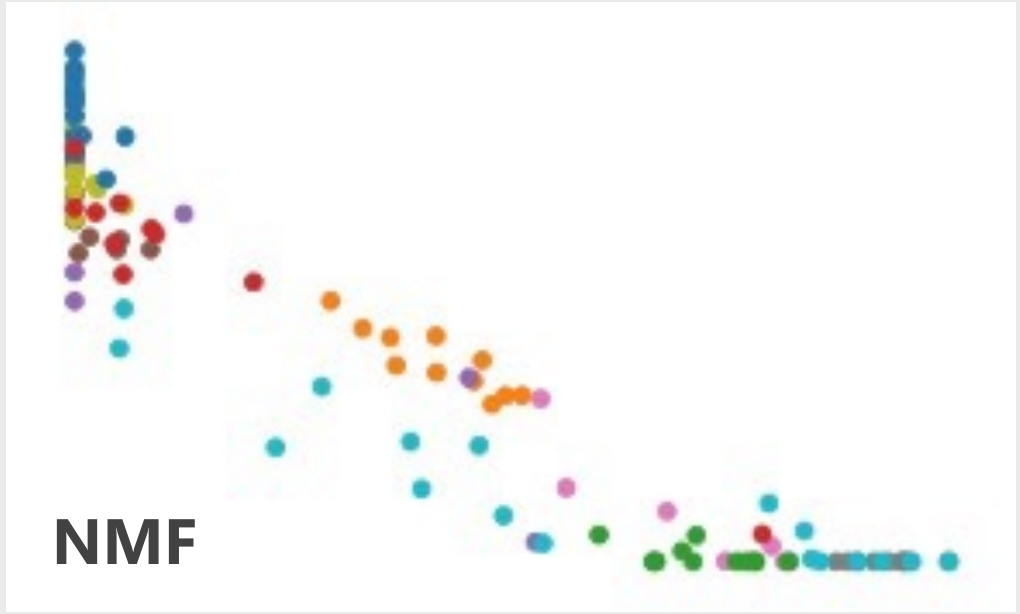
Singular Value Decomposition (SVD)



Non-negative Matrix Factorization (NMF)



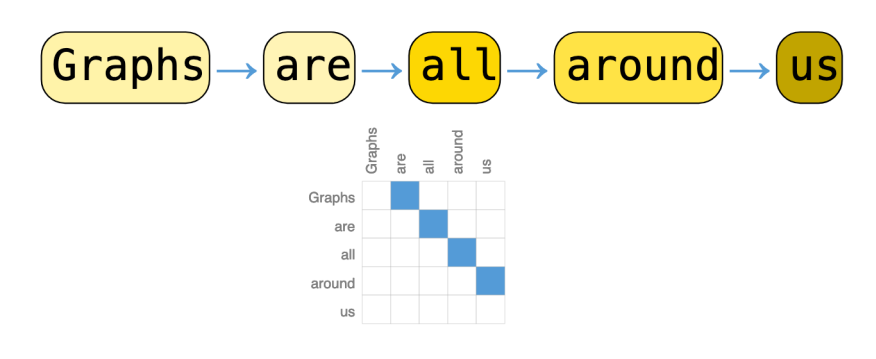


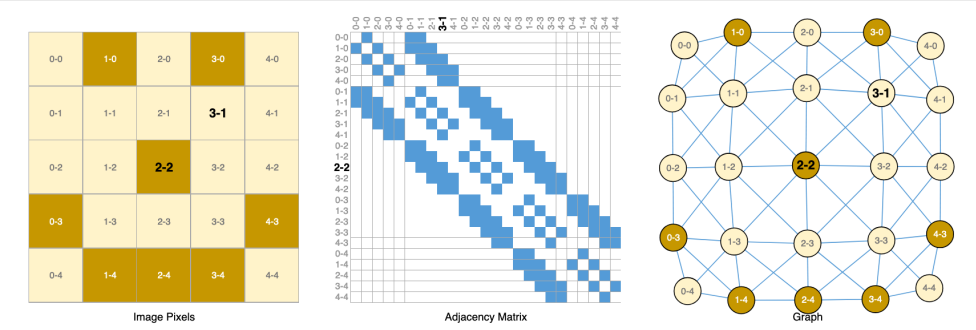

NMF

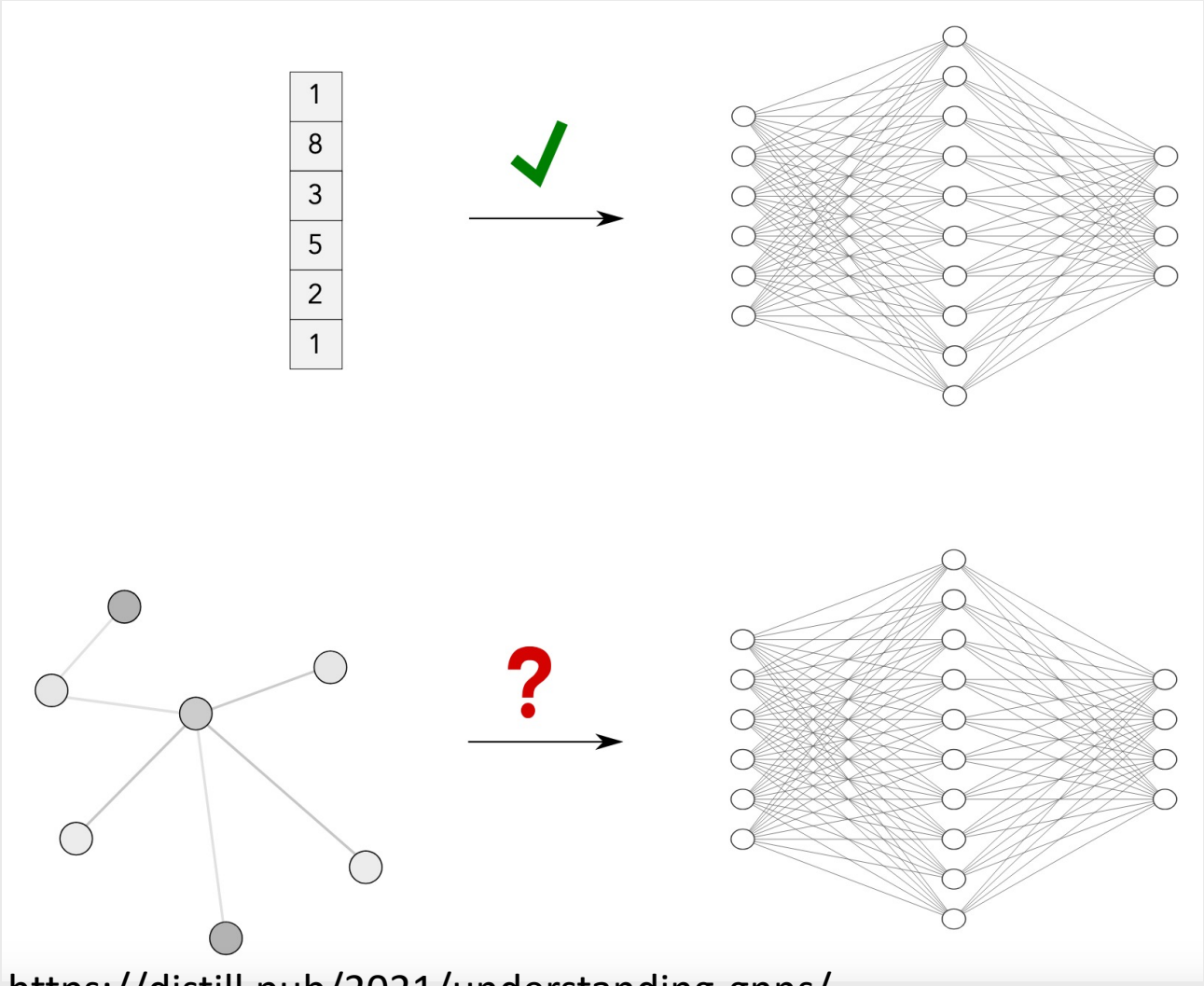
Option 3: Shallow neural Networks

Regular networks: Text, images

- Text analysis: Chain (nodes = words)
- Images: Lattices (nodes = pixels)





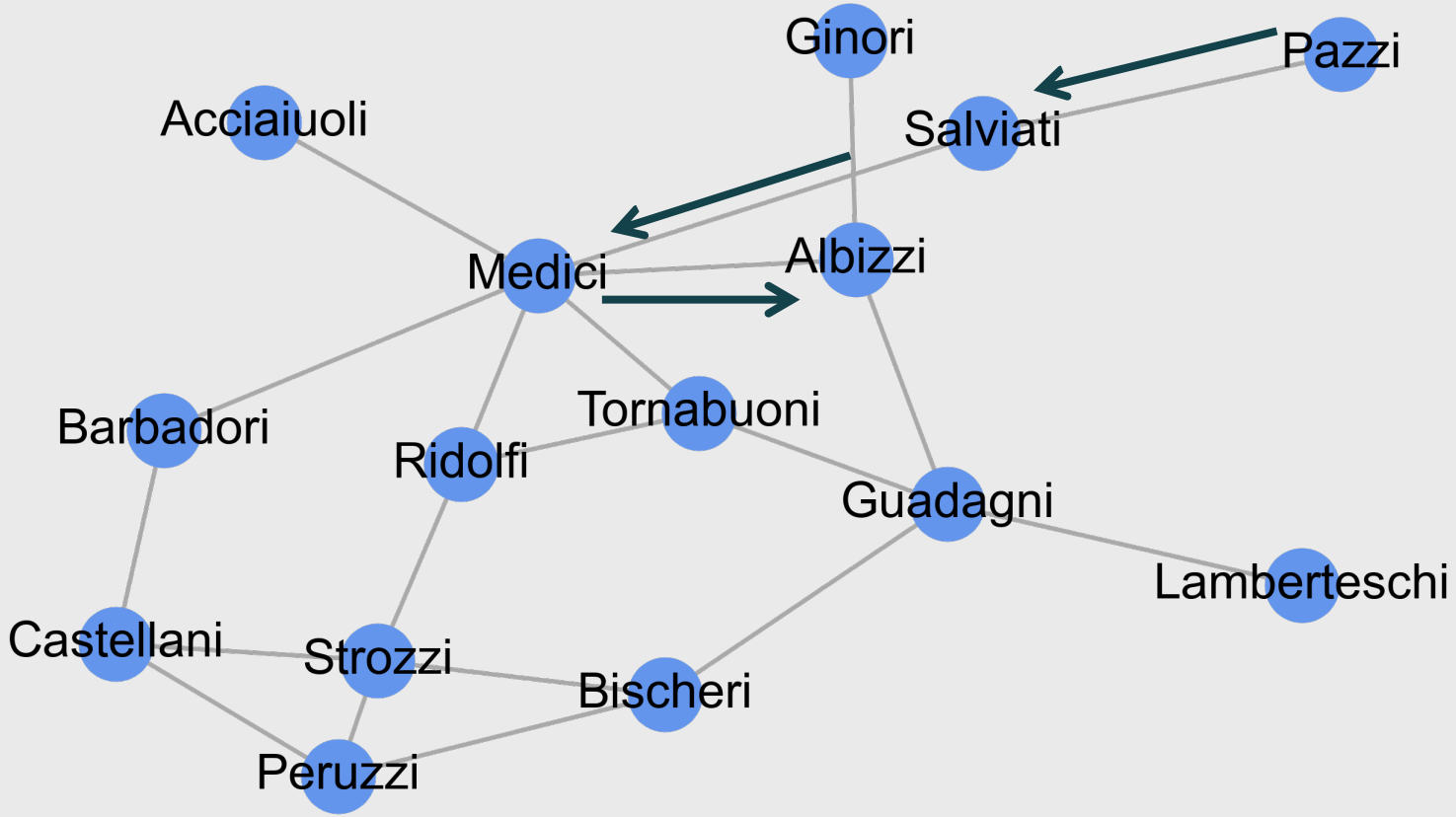


https://distill.pub/2021/understanding-gnns/

Node2vec / deepwalk

Idea:

1. Generate “sentences” using random walks.
2. Use methods from text analysis



Pazzi -> Salviati -> Medici -> Albizzi

…

Text analysis: Word2vec

Word2vec (SkipGram/CBOW)

Distributional hypothesis: similar words will be surrounded by similar words (you will know a word by the company it keeps)

What words appear around “Network” in text?

- Network Science
- Network Analysis

→ Words *science* and *analysis* are similar

Text analysis: Word2vec

Step 1: Create co-occurrance matrix

- I like deep learning
- I like NLP
- I enjoy flying

Context word

Target word

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Word Sense Disambiguation Using Vector Space Models

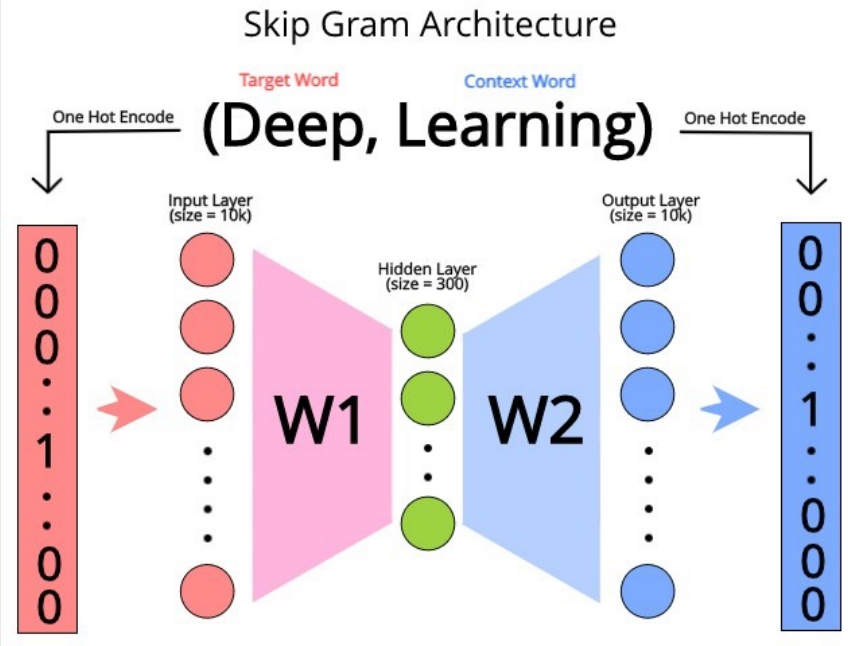
Text analysis: Word2vec

Step 2: Train a classification model

- *Positive examples*: target word should predict context words
- *Negative examples*: target word should not predict random words (i.e., not context words)

Two vectors are similar if they have a high dot product (~cosine similarity)

- Vector associated to "deep": w1["deep",:]
- Vector associated to "learning": w2[:,"learning"]



Intuition:

- Modify W1 and W2 so target embeddings are close (have a high dot product) to context embeddings for nearby words and further from context embeddings for noise words that don't occur nearby

Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition (Jurafsky and Dan, 2009)

In networks: Node2Vec (deepwalk)

Distributional hypothesis: The more often two nodes appear near the same nodes in the same random walk, the more similar their embeddings will be.

Step 1: Create co-occurrance matrix

Pazzi -> Salviati -> Medici -> Albizzi

Medici -> Albizzi -> Guadagni -> Medici

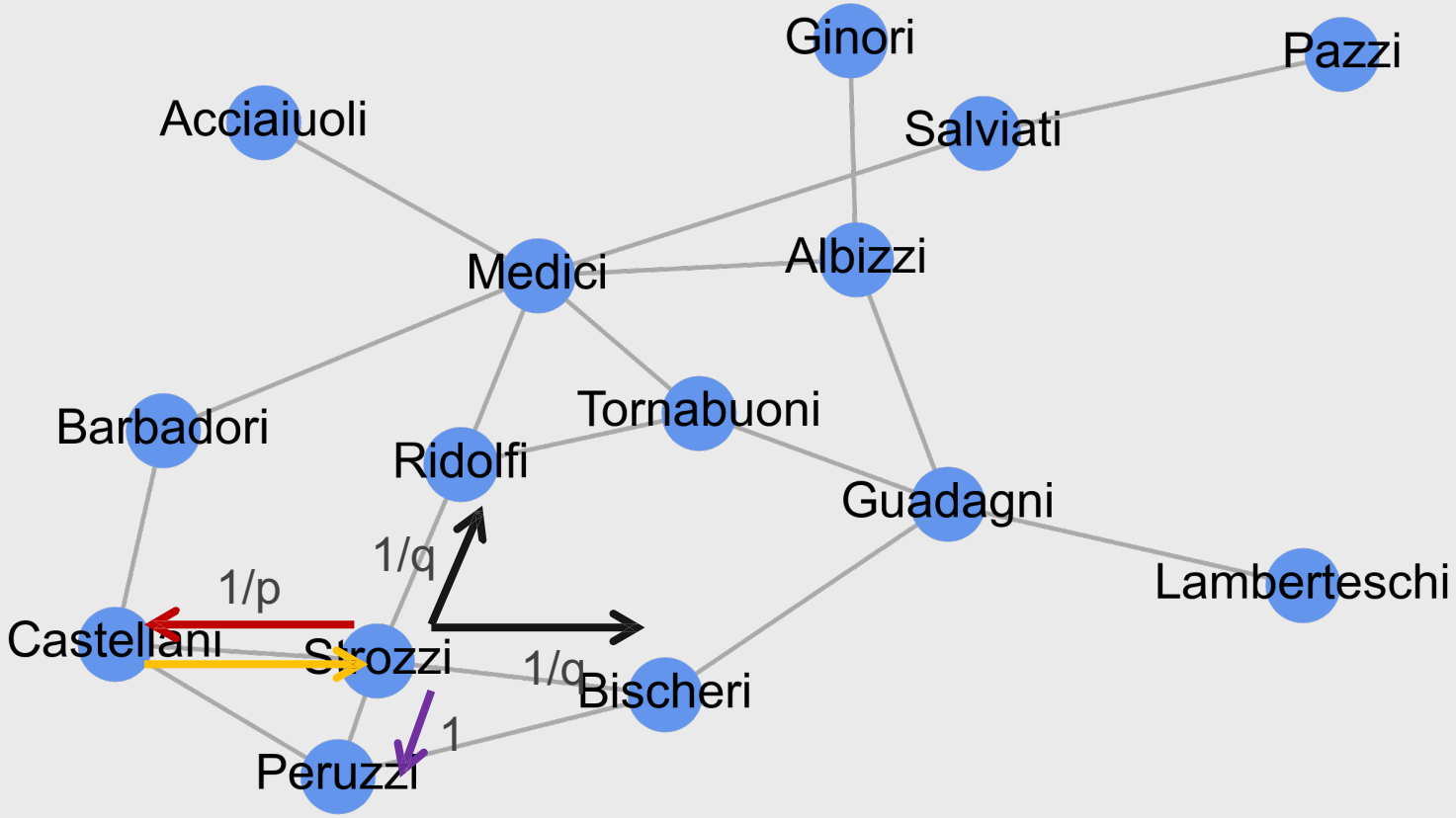
Context node

Target node	…	Pazzi	Salviati	Guadagni	Medici	Albizzi
	Pazzi		1		1	1
	Salviati	1				
	Guadagni					
	Medici	1		1	1	1
	Albizzi	1			1	

Step 2: Create embeddings representing how similar the neighbors of each node are

node2vec

Difference with deepwalk: generate “sentences” using biased random walks.
q = controls probability of going to new nodes
p = controls probability of going back to previous node



Using node2vec (deepwalk)

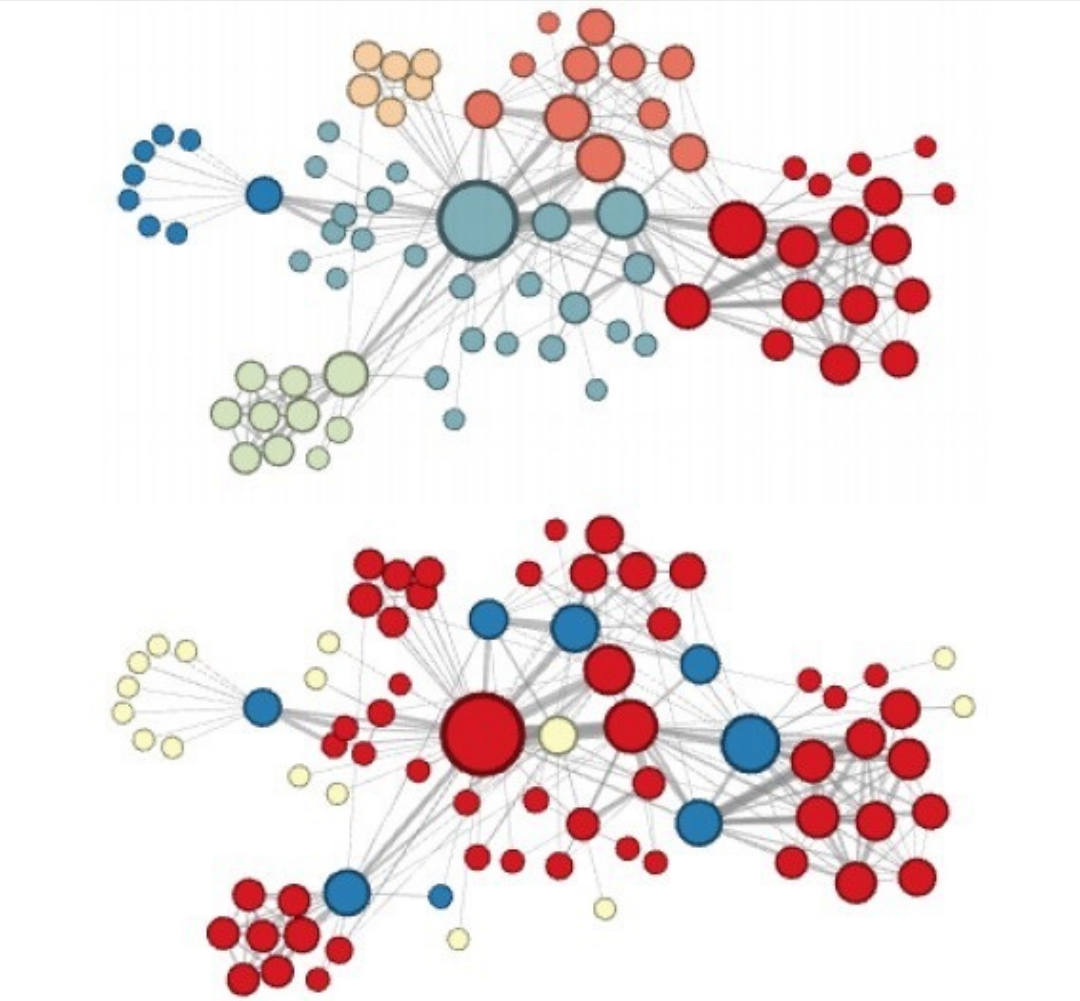


Figure 3: Complementary visualizations of Les Misérables coappearance network generated by *node2vec* with label colors reflecting homophily (top) and structural equivalence (bottom).

Depending on q

~ similiarity reflecting clusters

~ similiarity reflecting "structural roles"

Big problem: how to set up q and p?

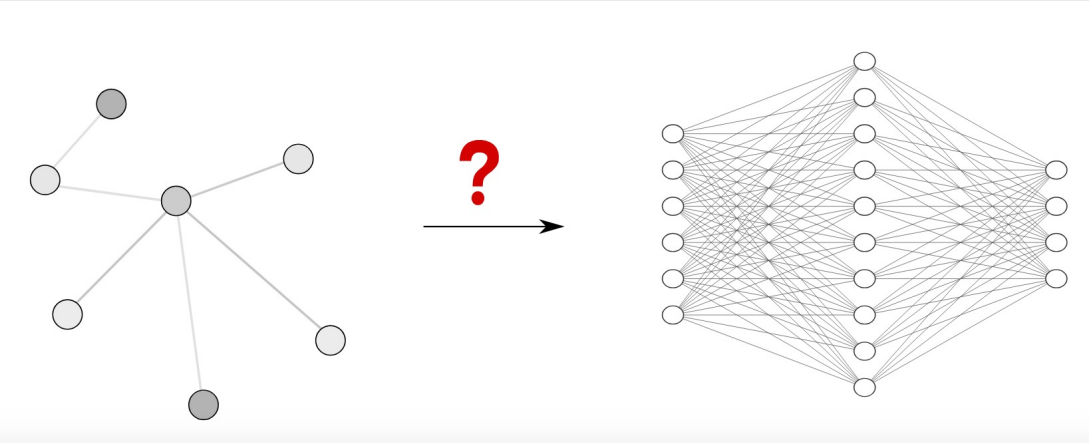
Option 4: Graph Neural Networks (GNNs)

Node2vec created the embeddings in an unsupervised (or self-supervised) way. But we can do it in a *supervised* way, so **node embeddings are similar if nodes have the same outcome**

In the context of link prediction → The same outcome = being connected

Many GNNs (GCN, GAT, graphSAGE…)

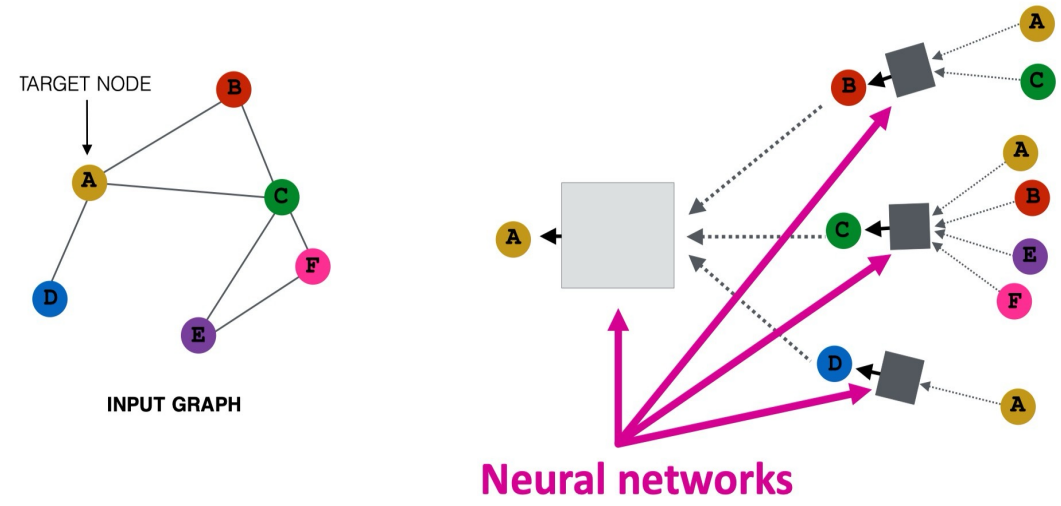
Problem:



https://distill.pub/2021/understanding-gnns/

Solution:

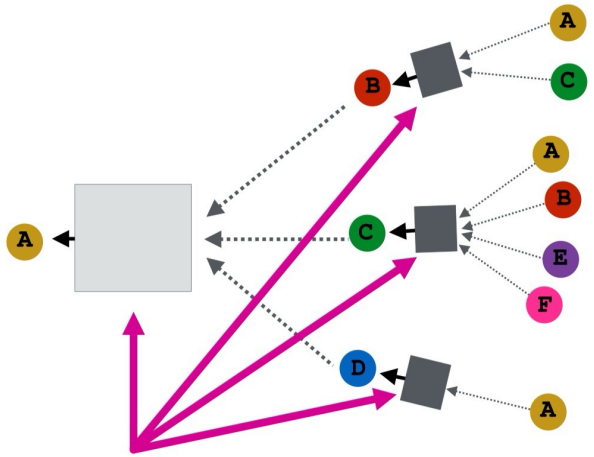
- **Intuition:** Nodes aggregate information from their neighbors using neural networks



https://web.stanfod.edu/class/cs224w/

Option 4: Graph Neural Networks

Example (Graph Convolutional Network)



https://web.stanfod.edu/class/cs224w/

$$h^{(k)} = f(D^{-1}A \cdot h^{(k-1)}W^{(k)^T} + h^{(k-1)}B^{(k)^T})$$

Node embedding (at layer k)

Normalized adjacency matrix

Trainable weights

https://distill.pub/2021/gnn-intro/
https://distill.pub/2021/understanding-gnns/

Recap

We want to create low-dimensional embeddings:

- Spectral methods
- Non-negative matrix factorization
- Shallow neural networks
- Graph Neural Networks

Primer on machine learning

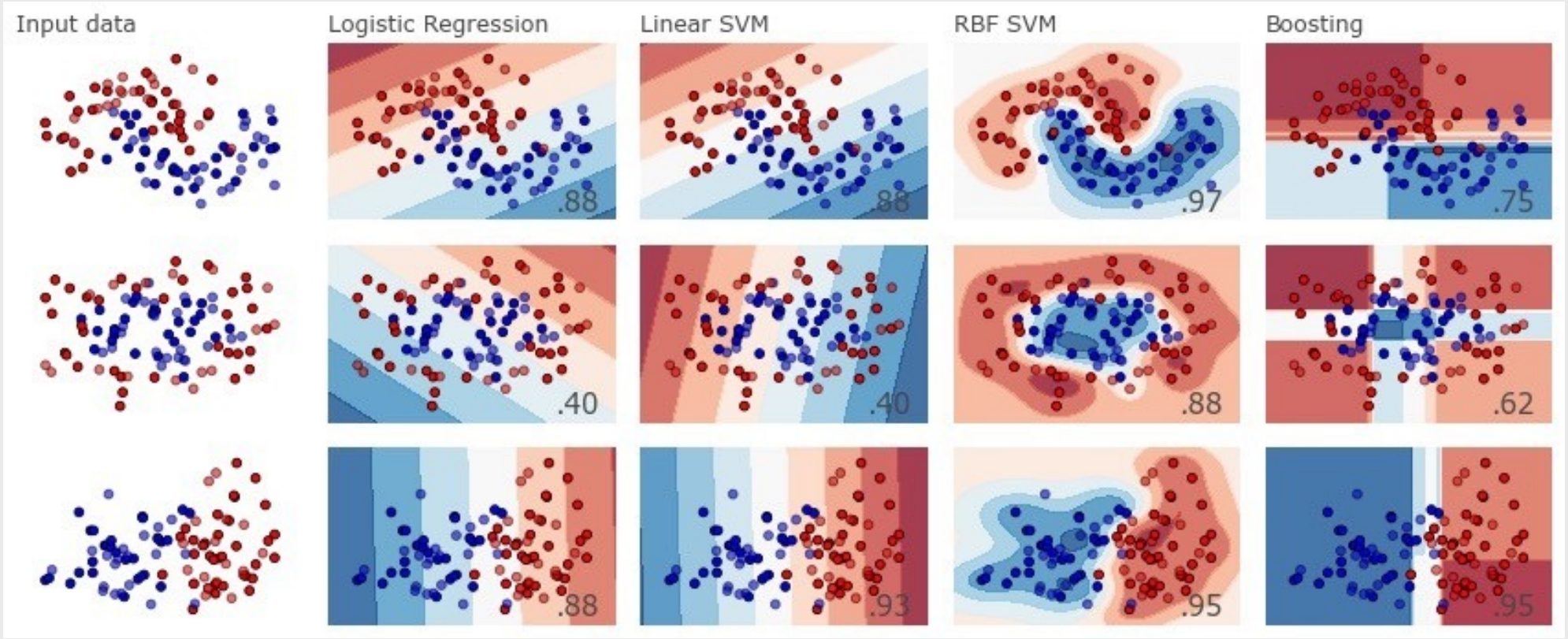
Machine learning

"A computer program is said **to learn from experience *E*** with respect to some class of **tasks *T*** and **performance measure *P*** if its performance at task *T*, as measured by *P*, improves with experience *E*." (Samuel/Mitchell, 1959)

- Experience: Data (networks)
- Task: Goal (link prediction)
- Performance measure: Accuracy, R^2, etc

Prediction link/no-link = classification

- Use link features (number of common neighbors, number of paths, or similarity of node embeddings)
- Predict the links: (Penalized) logistic regression; Support Vector Machines; Boosting…
- No best algorithm: *Stacking models for nearly optimal link prediction in complex networks; Ghasemian, Galstyan, Airoldi, Clauset (2020)*
- Each algorithm gives you a score: we can combine them in a model



Main issues in Machine Learning

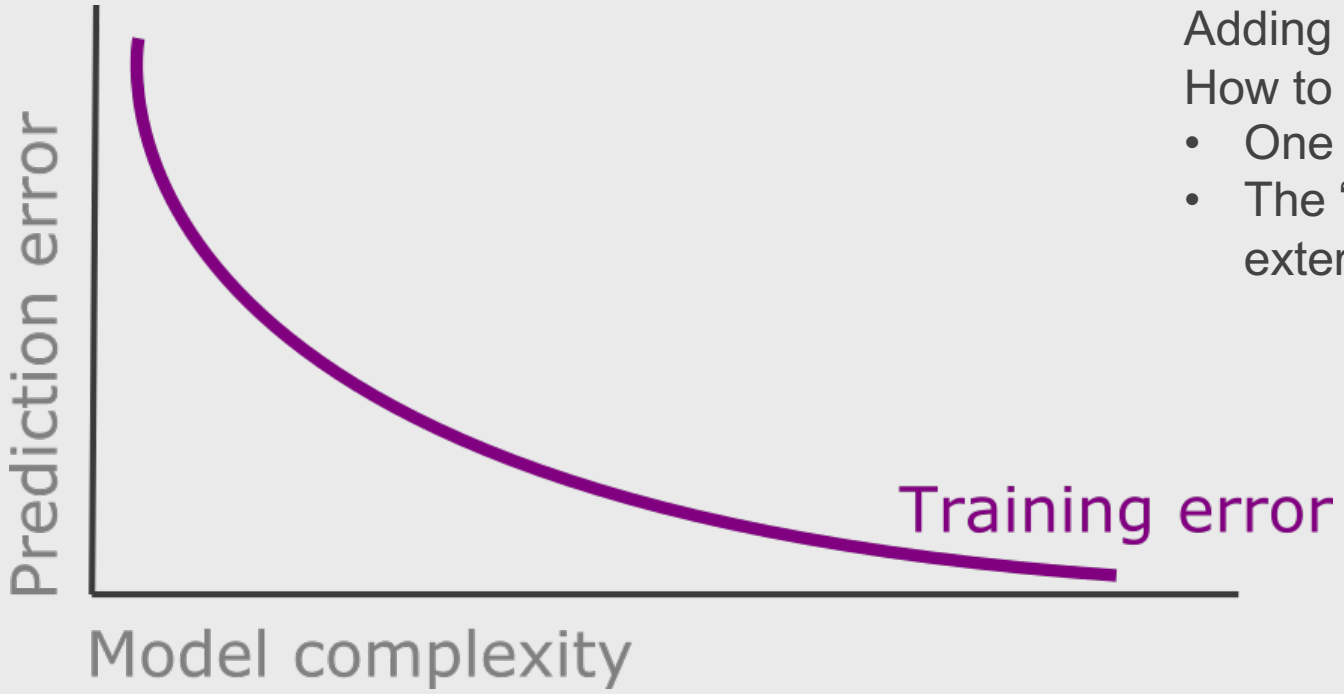
Issue 1: Overfitting

- Lots of data and features → can be a recipe for disaster

Issue 2: Interpretability of the model

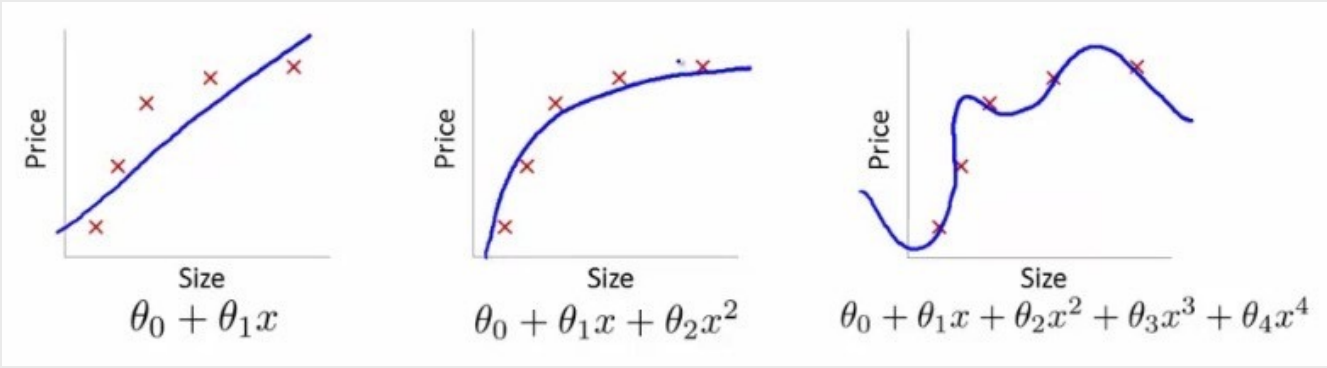
- Complex models are more difficult to interpret

Issue 1: Overfitting



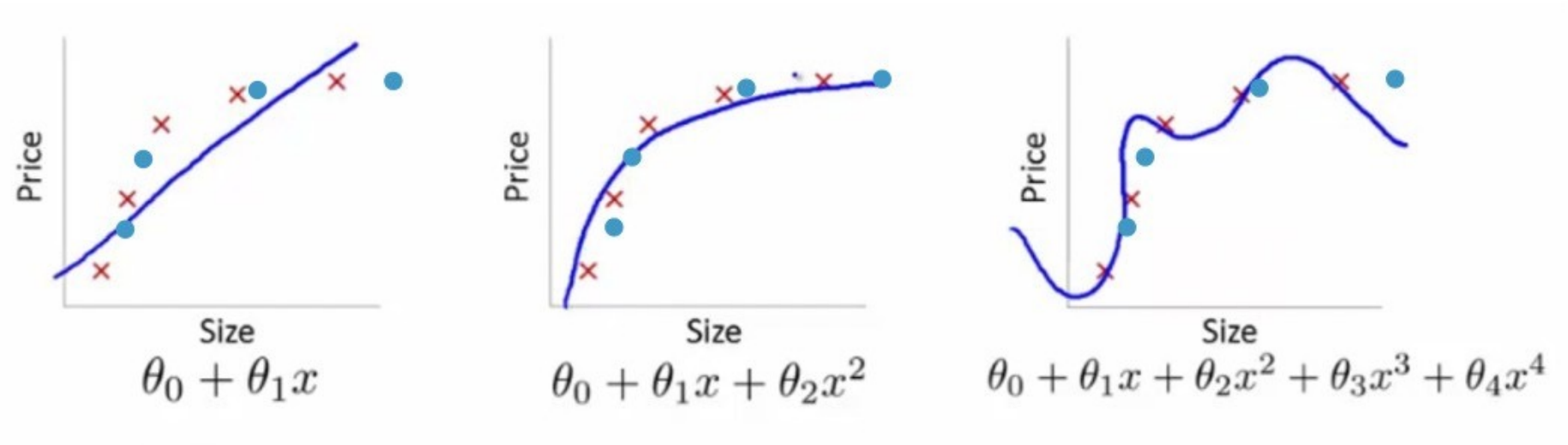
Adding more variables always decreases the error.
How to estimate real prediction error?

- One option: adjust for the complexity of model
- The “ML” option: evaluate prediction accuracy in an external dataset

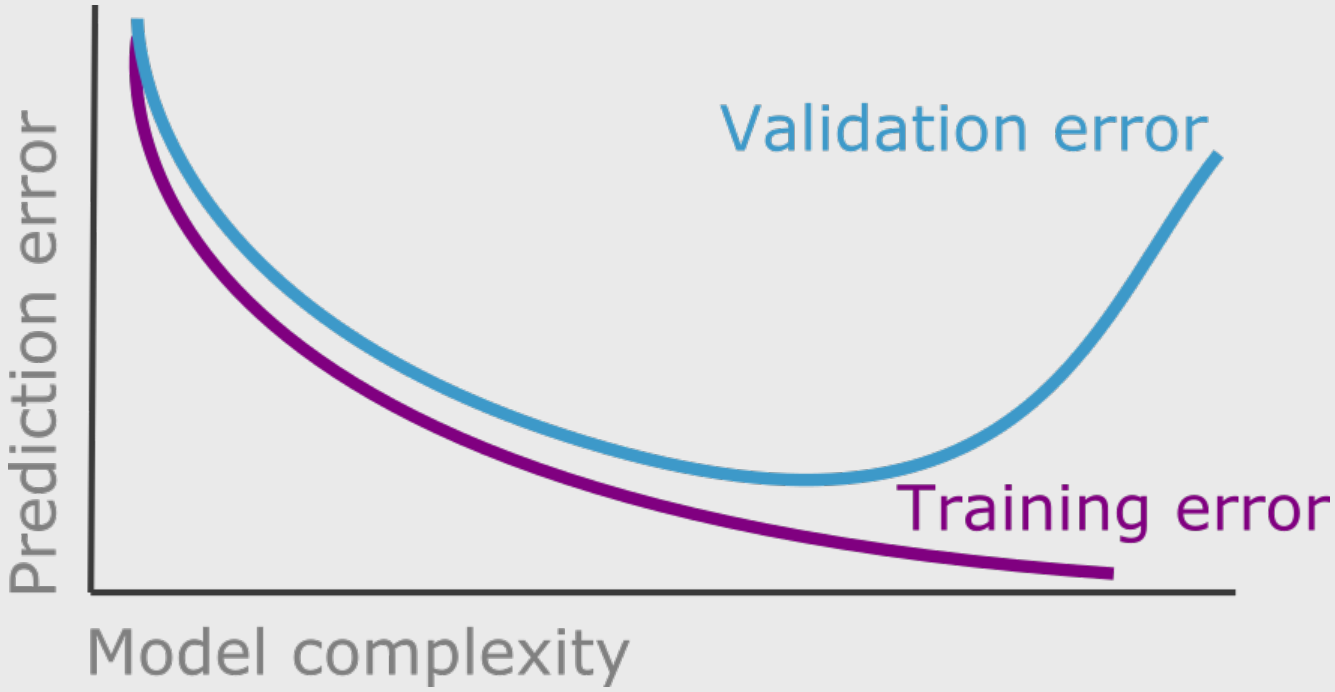


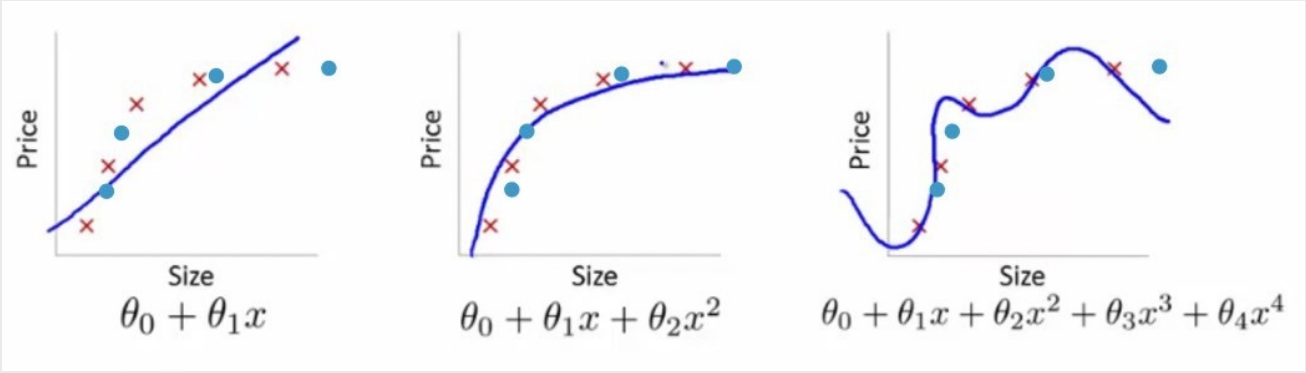
1: Evaluate overfitting using a validation dataset

Training dataset → Use to train different models
Validation dataset → Evaluate out-of-sample prediction error



1: Evaluate overfitting using a validation dataset



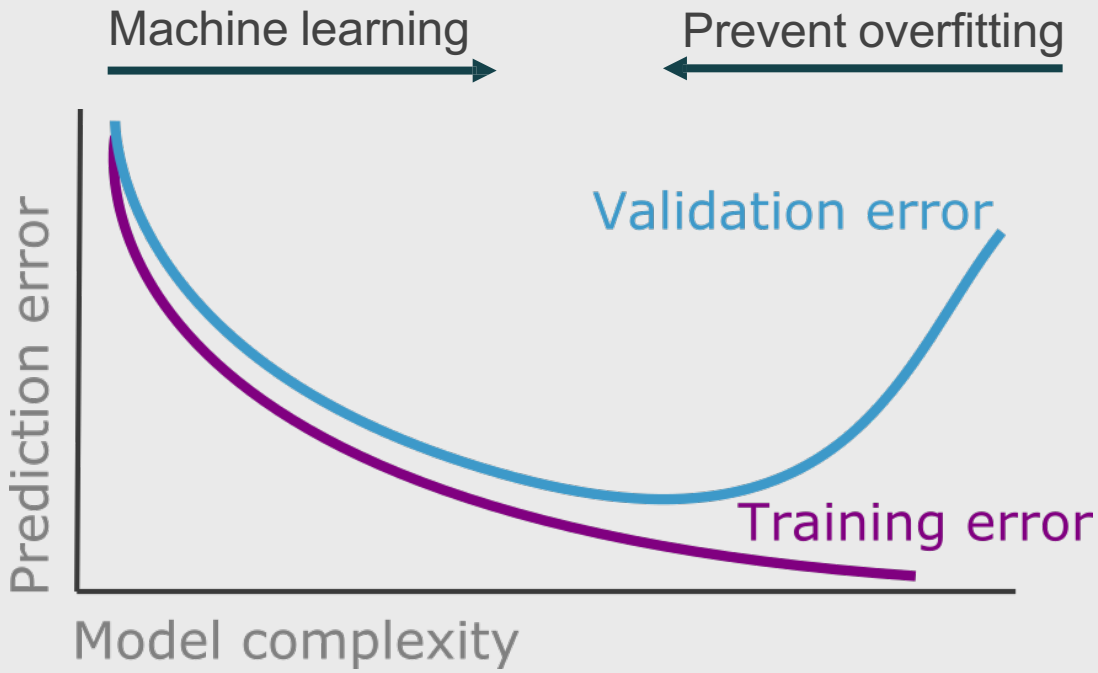


But with this:

- We reduce the training dataset (number of observations)
- We validate on a small dataset (maybe not representative)

Solution → Cross-validation

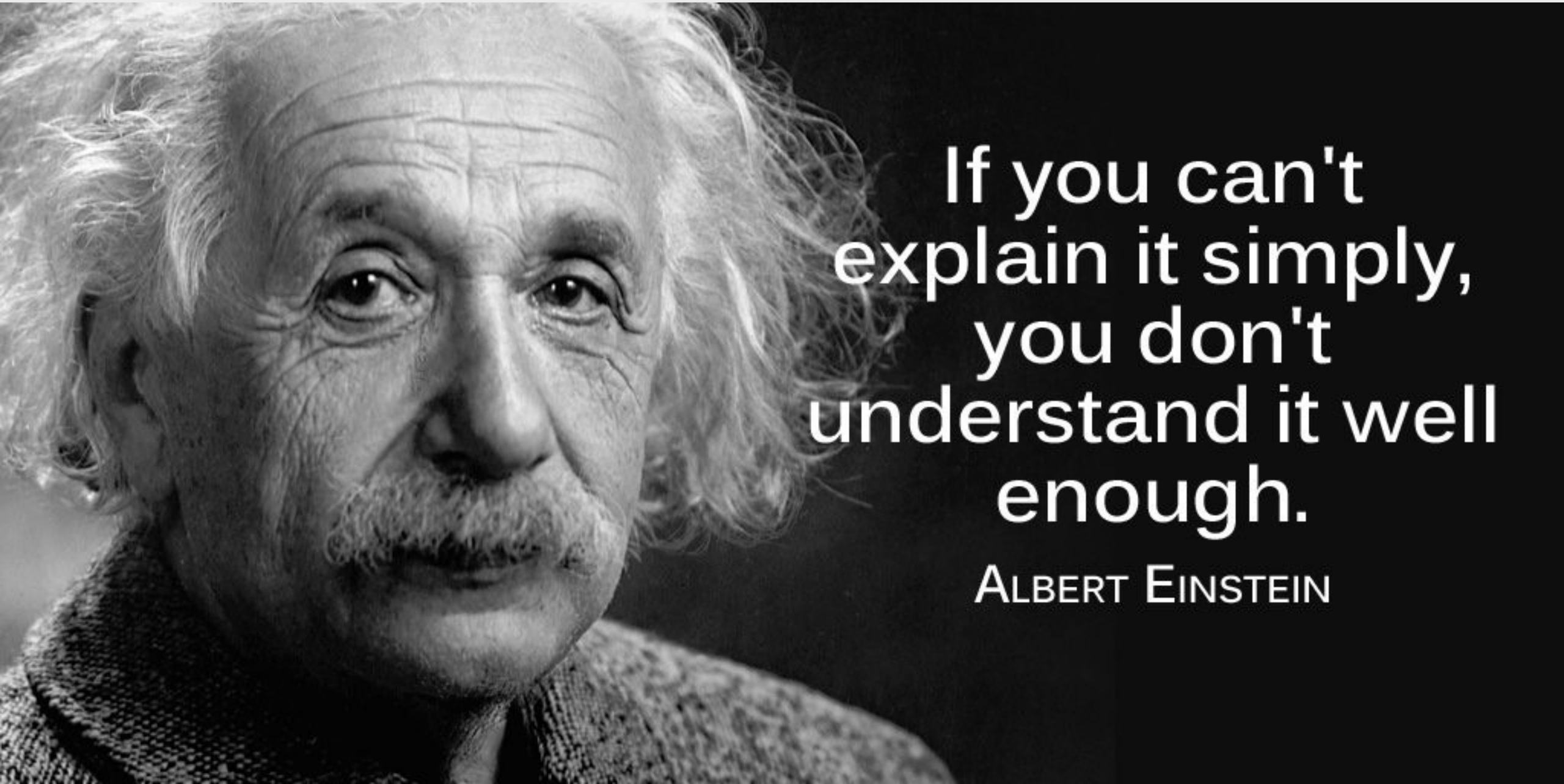
1: Hyperparameter tuning



Hyperparameter tuning using cross-validation →
Balance between flexibility and overfitting:

- Regularization (e.g. sum of |coefs| < x)
- Ensembles:
  - Train trees with different data
    - Bootstrap sample
    - Subset of predictors
  - Use shallow trees
- Neural networks:
  - Train disabling neurons (dropout)
  - Early stopping

Issue 2: Interpretability



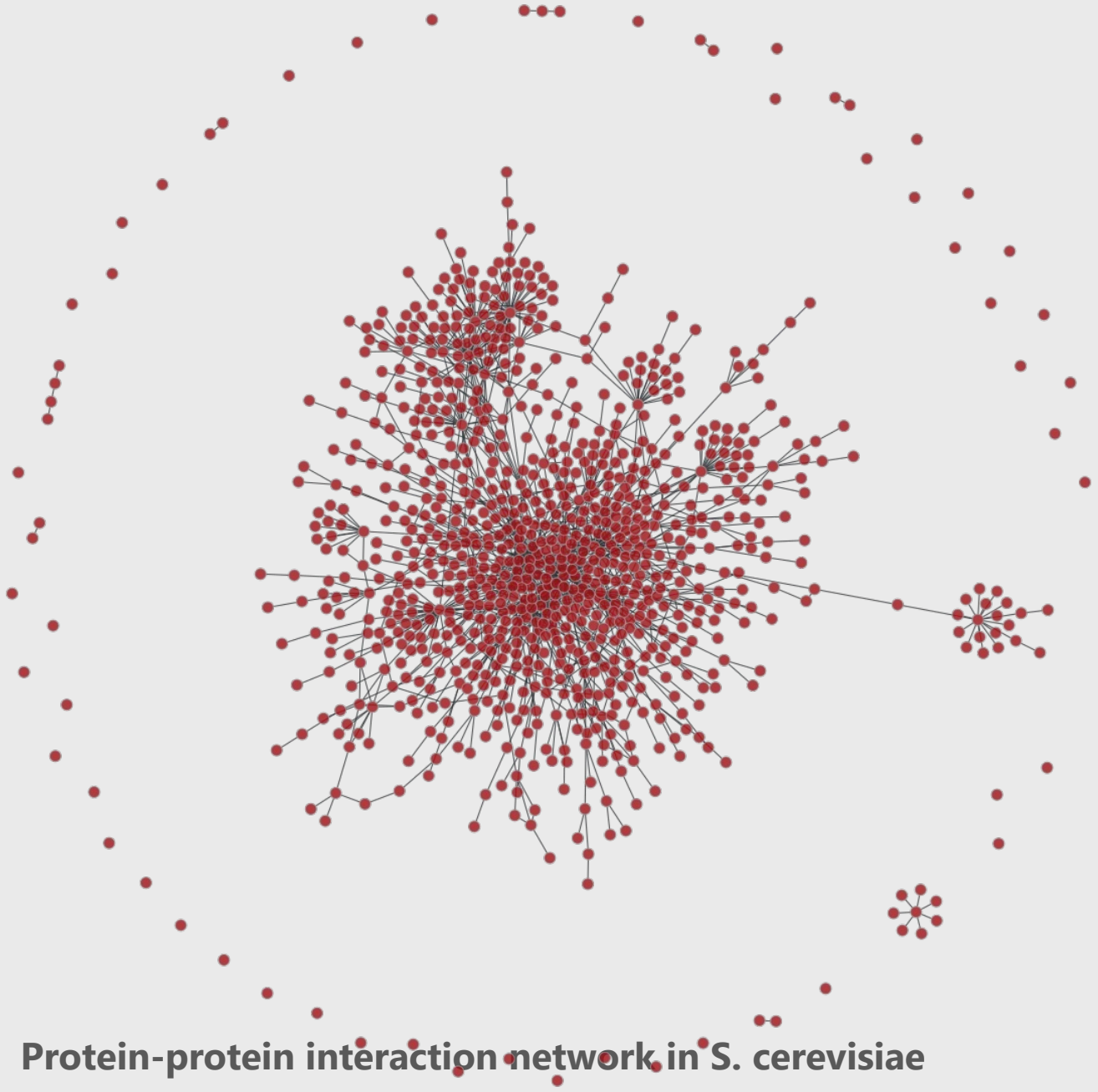
Typical workflow

```
# Split data into training and testing to evaluate generalization error
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)


# Hyperparameter tuning using your model: e.g., linear_model.LogisticRegression(), svm.SVC(),
ensemble. HistGradientBoostingClassifier()
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]} # (e.g. for a SVC)
clf = GridSearchCV(model() , parameters)
clf.fit(X_train, y_train)
print(clf.best_params_)


# Use the best model to predict the labels (link/no link) in the testing data
y_pred = clf.best_estimator_.transform(X_test)
print(F1_score(y_test, y_pred))
```

Structure of the challenge



Protein-protein interaction network in S. cerevisiae

Clustering ~ 0; Assortativity ~ -0.2

We have removed some edges, your objective is to predict those accurately.

We give you:

- Graph: Used for training
- Test dataset (a series of node pairs, some with a link associated)

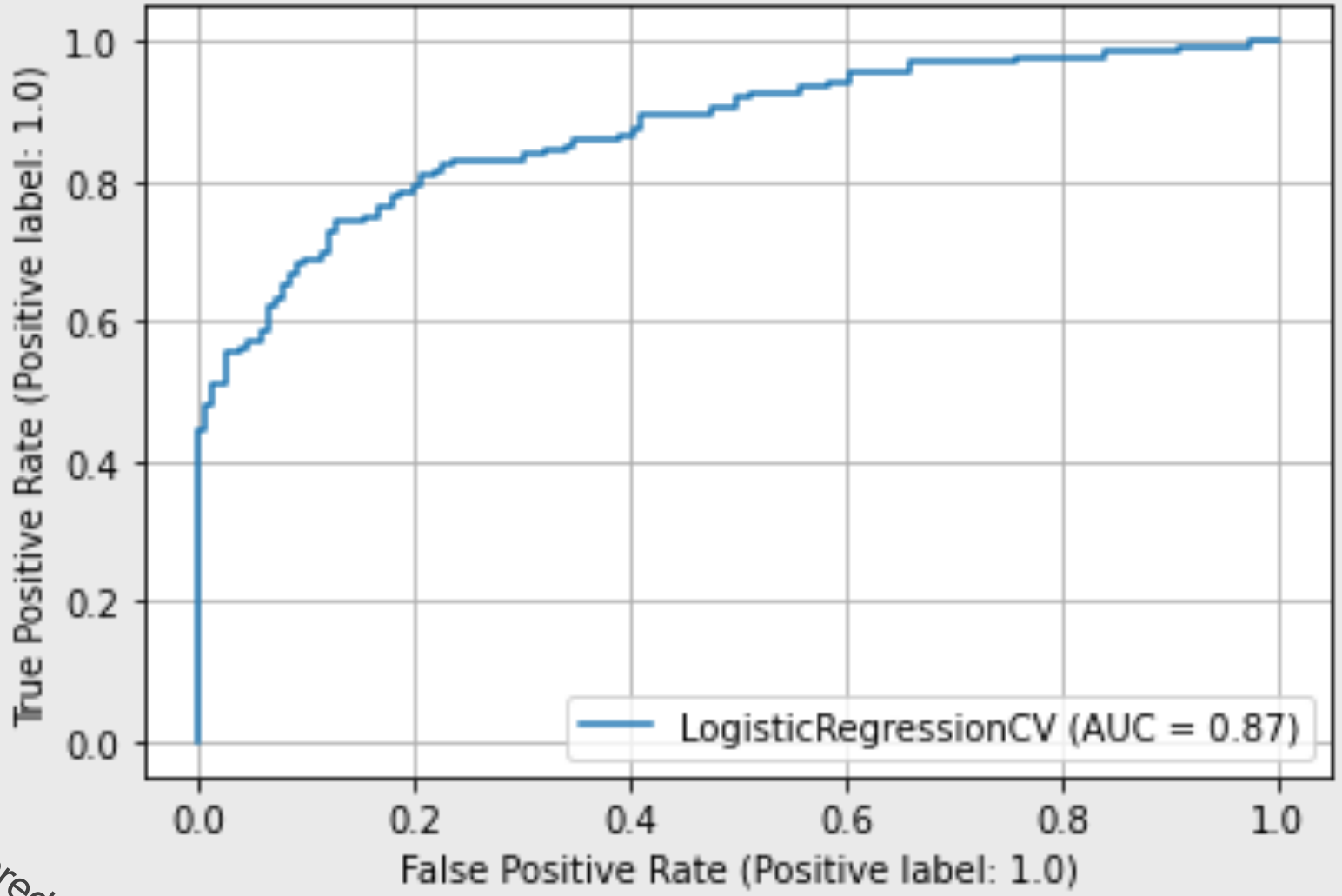
How:

- Methods based on common neighbors
- Methods based on paths
- Methods based on embeddings
  - Spectral methods
  - Matrix factorization
  - Node2vec
  - GraphSAGE

Evaluation

(all links predicted correctly)

$$TPR\ (sensitivity) = = \frac{Links\ predicted\ correctly}{\#\ \text{edges}}$$



(no links predicted)

(all no links predicted as links)

$$FPR = \frac{No\text{-}links\ predicted\ as\ links}{\#\ node\ pairs\ with\ no\ link}$$