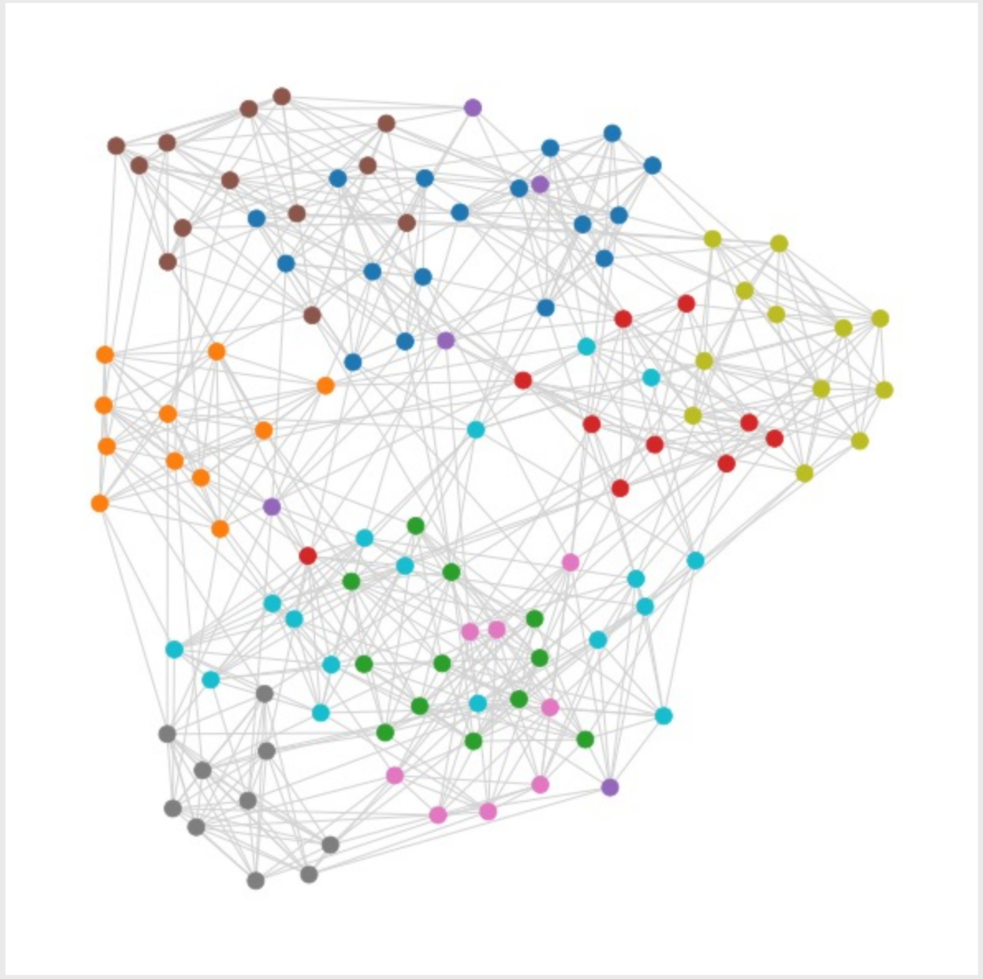
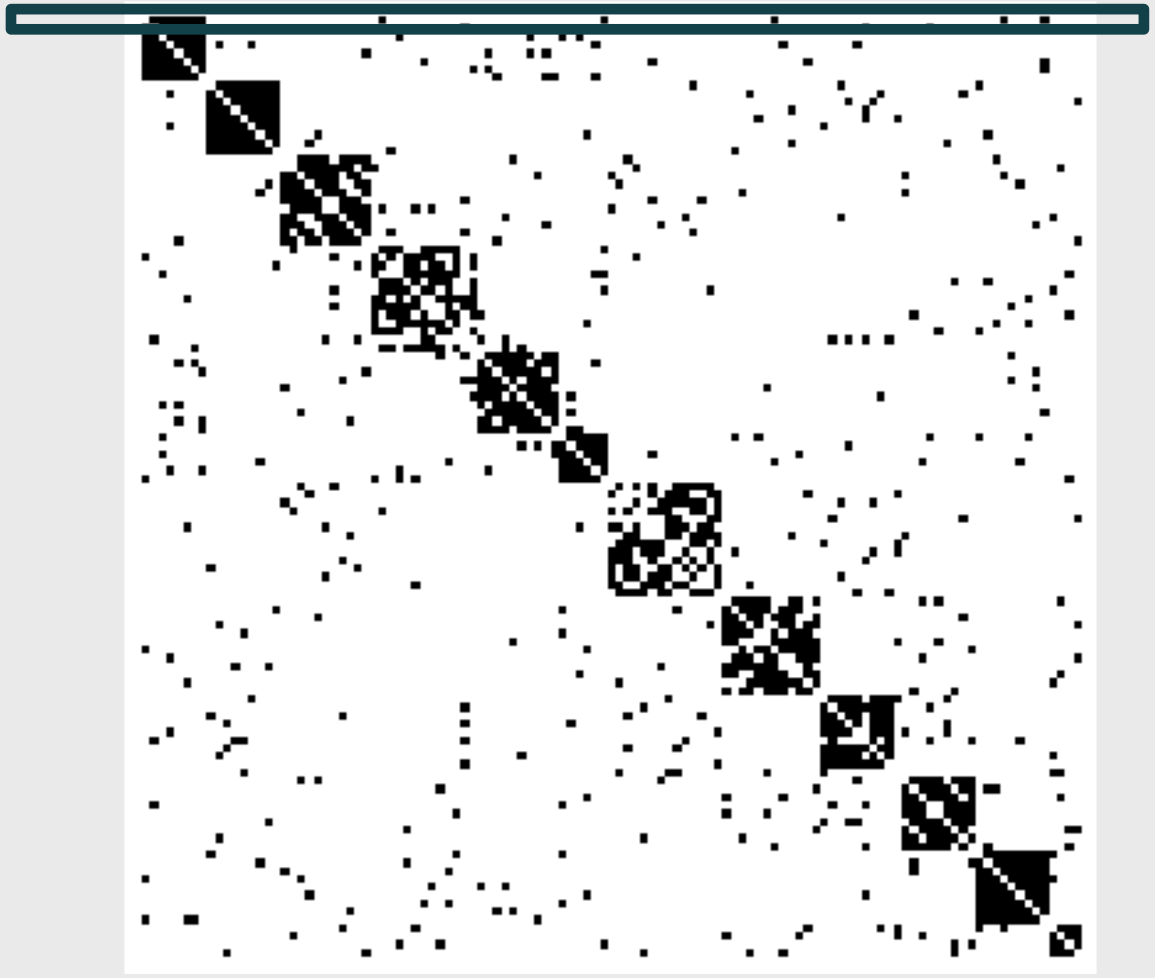
Node embeddings

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!
- andd only provides information at the local level

Node embeddings

Idea: Create a low(er)-dimension representation of the node
→**Similar nodes** should have a similar representation

What do we mean with similar nodes?

- Nodes with the same neighborhood (useful to predict e.g. clustering or assortative attributes)
- Nodes with the same role (useful to predict e.g. structural equivalence or disassortative attributes)
- Nodes with the same metadata

Unsupervised vs supervised learning

Supervised learning: Output is available. Performance = discrepancy between predicted output and real output

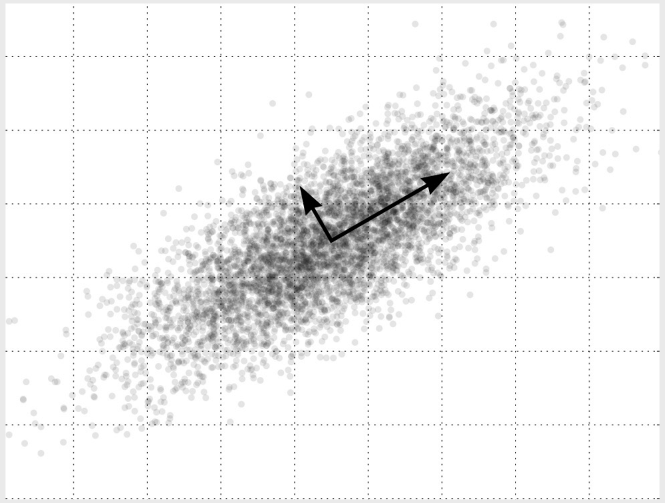
- Regression
- Classification (e.g. link prediction!)

Unsupervised learning: No labels/output. Performance = reduction of some error

- Clustering (e.g. cluster points so they are as close as possible within clusters, as far as possible between clusters)
- Dimensionality reduction (e.g. combine variables to maximize the amount of variability explained)

Option 1: Spectral methods

Related to characteristic eigenvectors of matrices associated with the network



Principal Component Analysis (PCA)

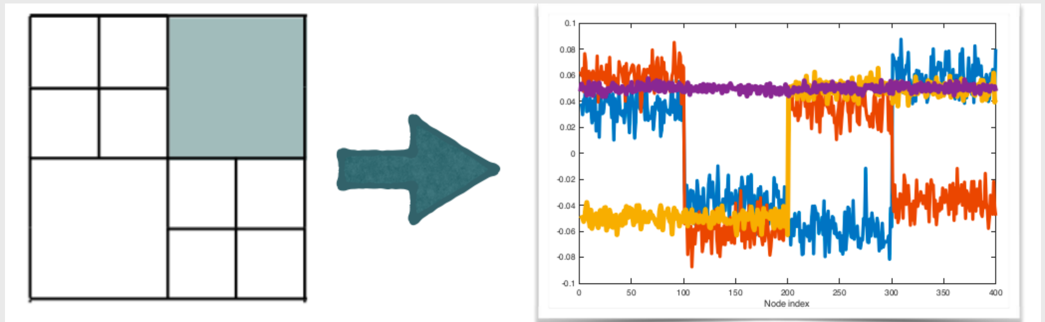
- Correlated variables → Linear combination of orthogonal variables
- Eigenvectors corresponding to the largest k eigenvectors of the covariance matrix

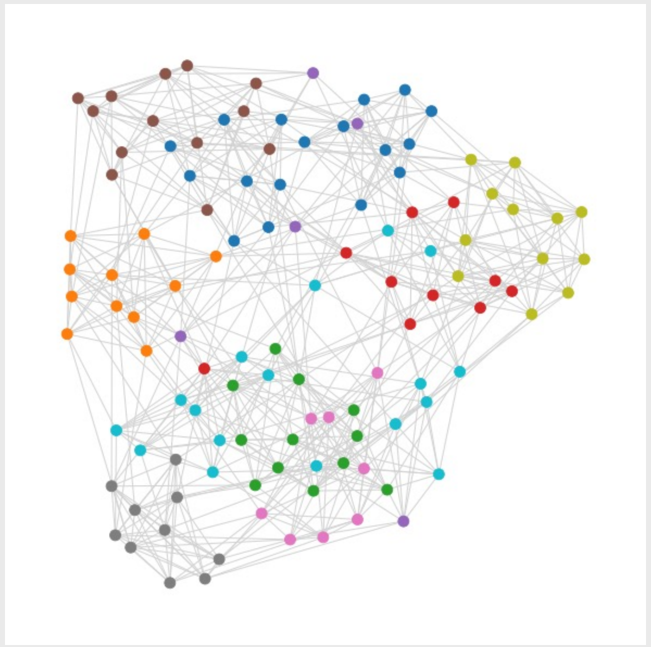
Singular Value Decomposition (SVD)

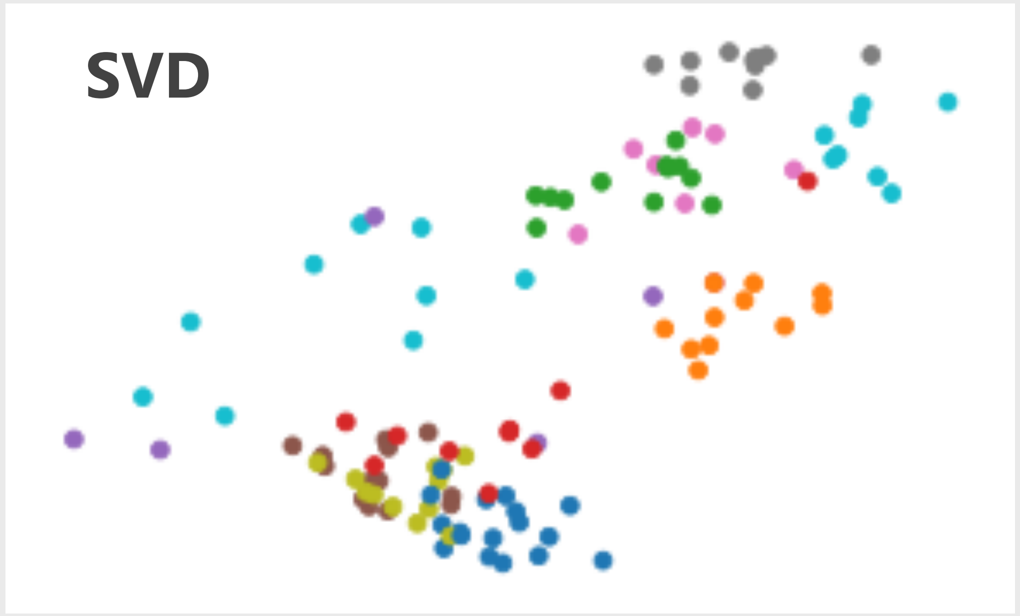
- Eigenvectors corresponding to the largest k eigenvectors of $A^T A$

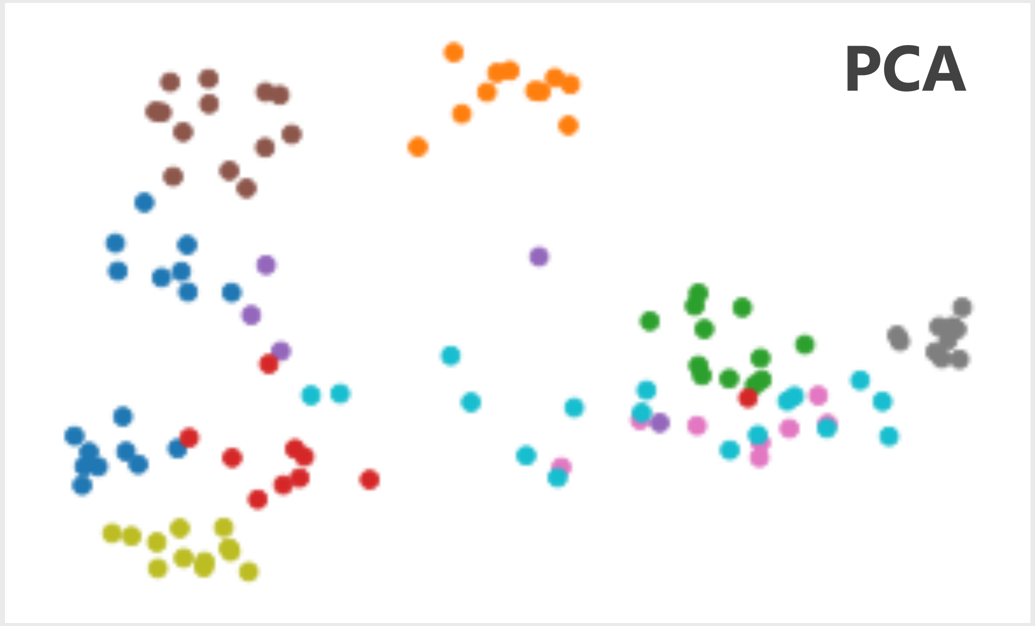
Laplacian Eigenmaps (LE)

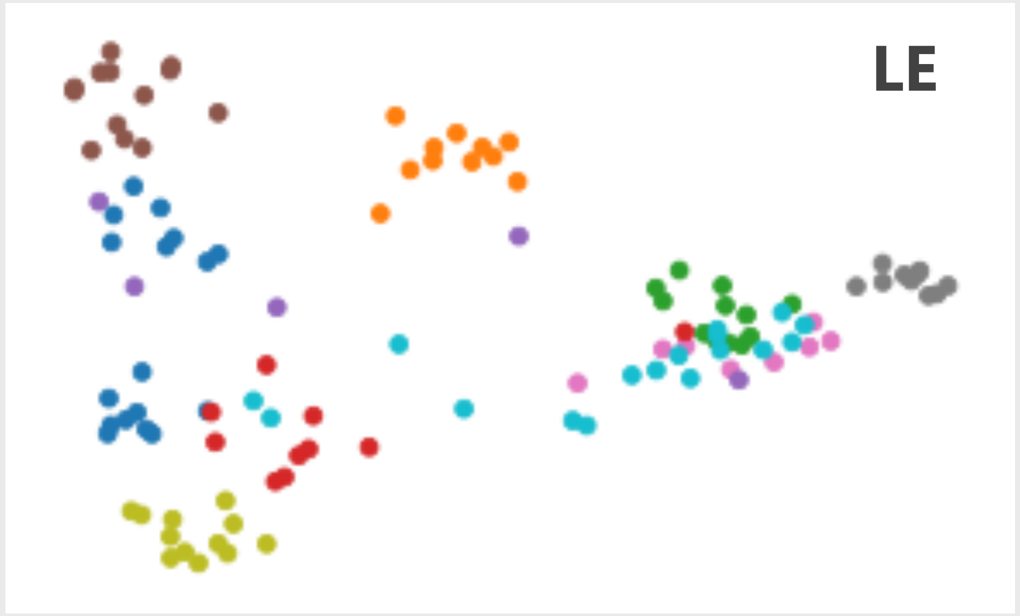
- Assumes that the nodes lie on a low-dimensional (with some constrains)
- Tries to find a mapping that minimizes the distances between all nodes
- That mapping is creates by the eigenvectors corresponding to the smallest k eigenvectors of the normalized Laplacian matrix D^-1(D-A)





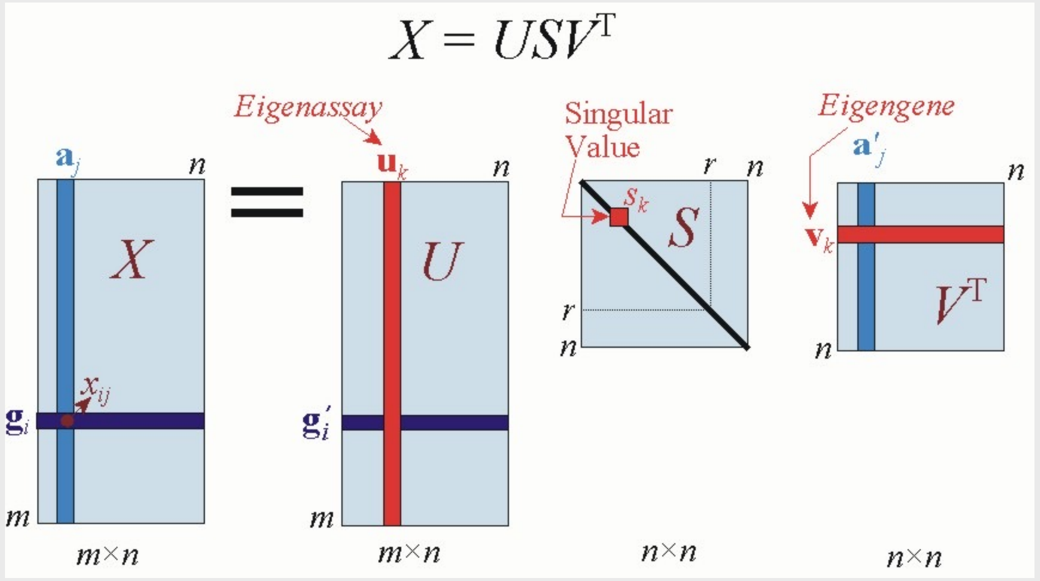

SVD


PCA

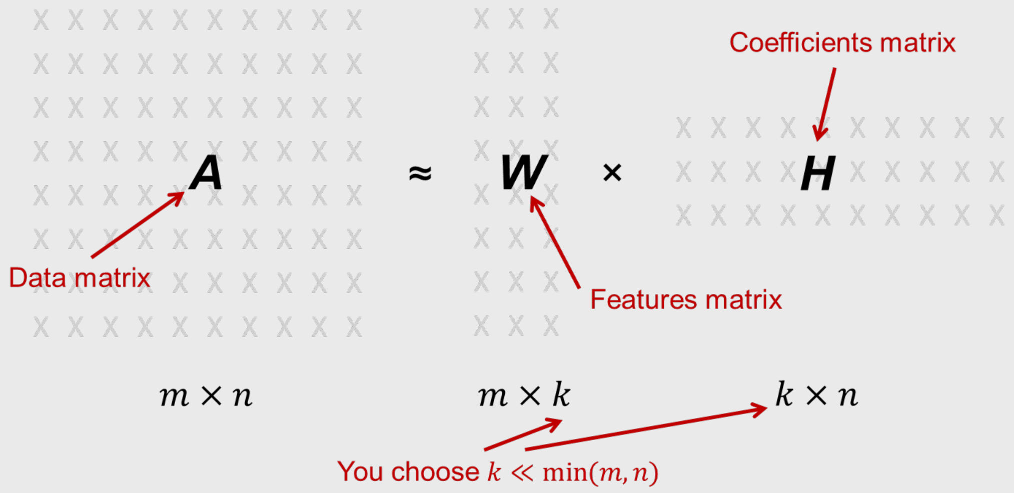

LE

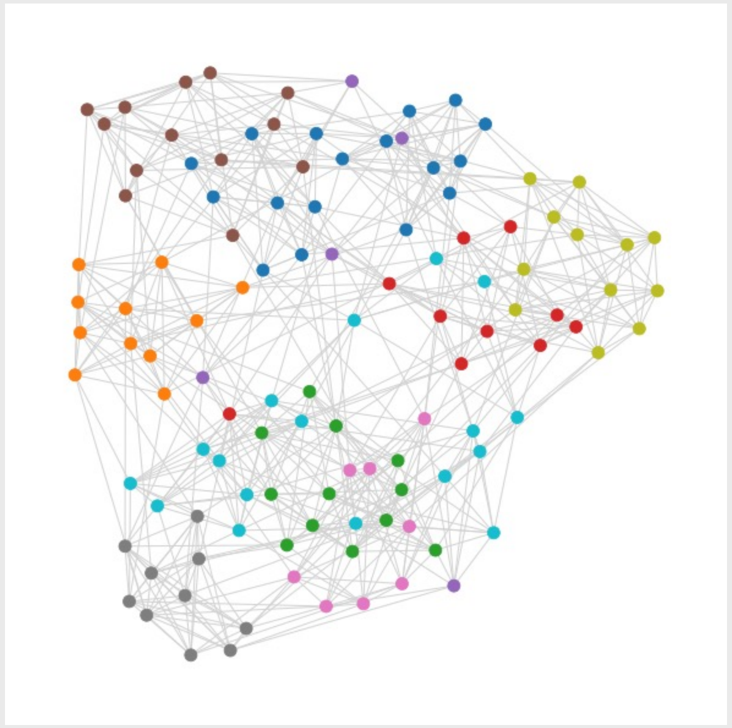
Option 2: Matrix factorization

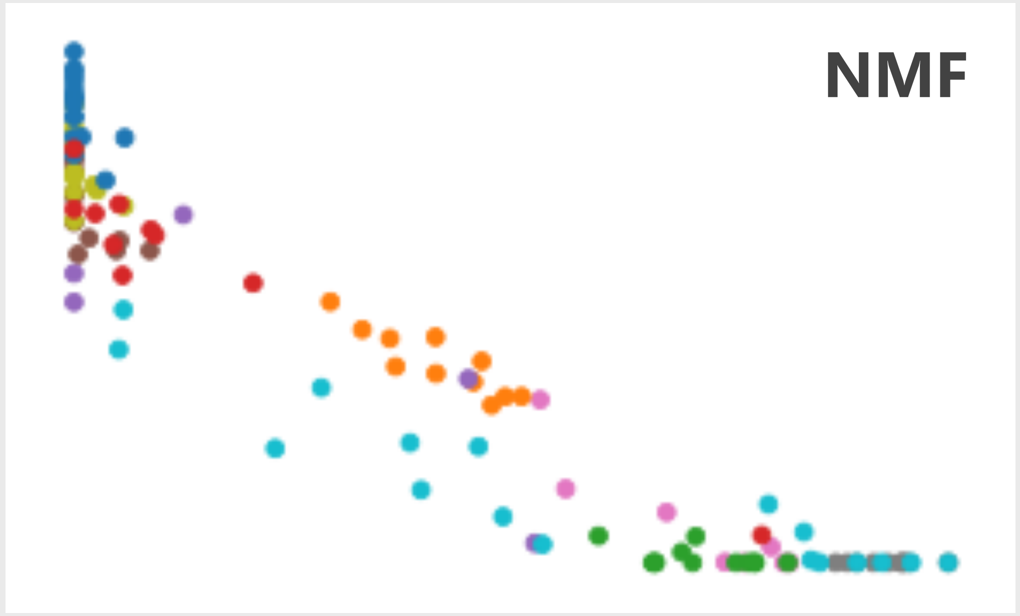
Singular Value Decomposition (SVD)



Non-negative Matrix Factorization (NMF)






NMF

Option 3: Shallow Neural Networks

In text analysis and image recognition we have a lot of neural network tools because the data is regular

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

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?

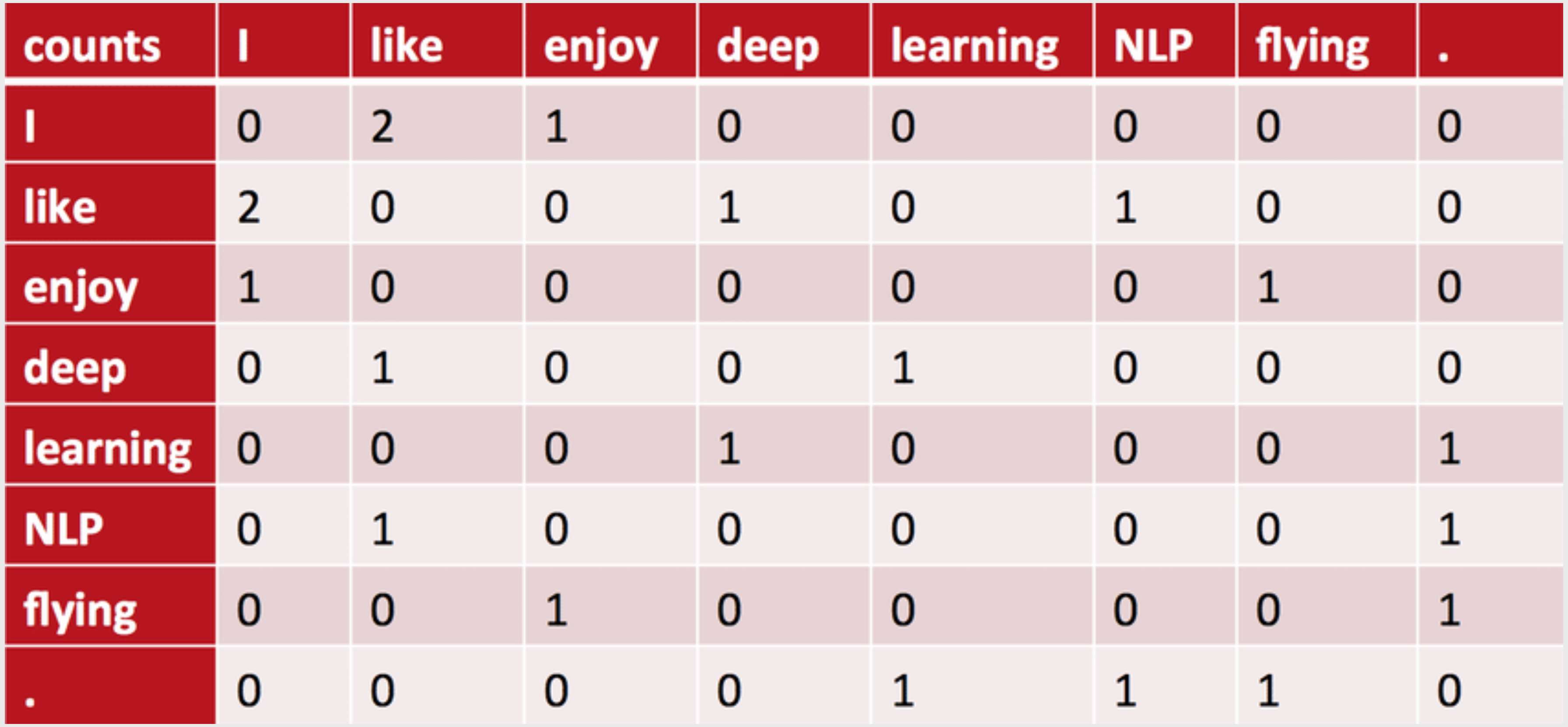
- Network Science
- Network Analysis

→ Words *science* and *analysis* are similar

Word2Vec

Step 1: Create co-occurrance matrix

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



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

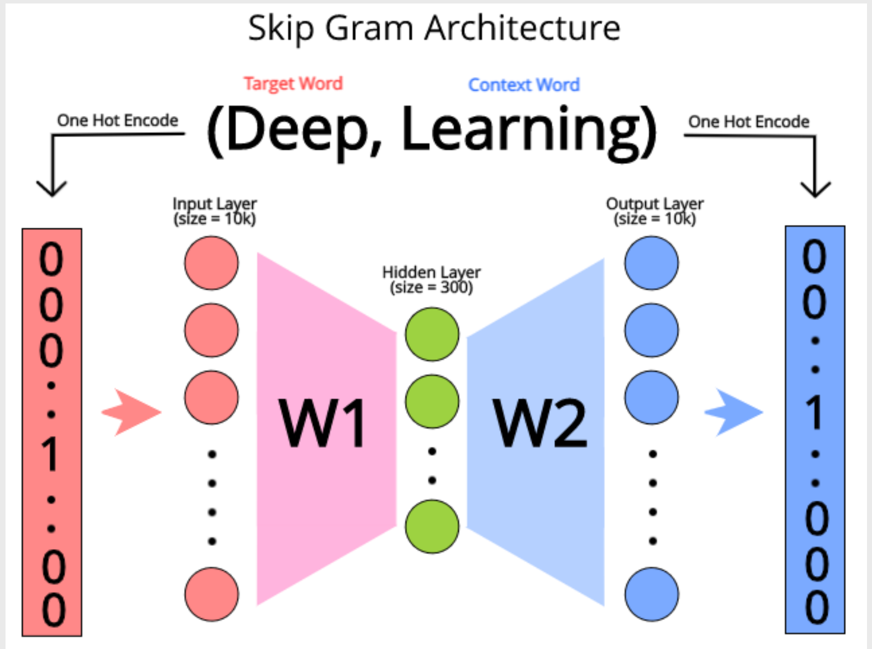
Word2Vec

Step 2: Train the model

- *Positive examples*: word i used to predict word j (context word in the co-ocurrance matrix)
- *Negative examples*: word i used as a negative example to predict word k (not a context word)

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)

Word2Vec

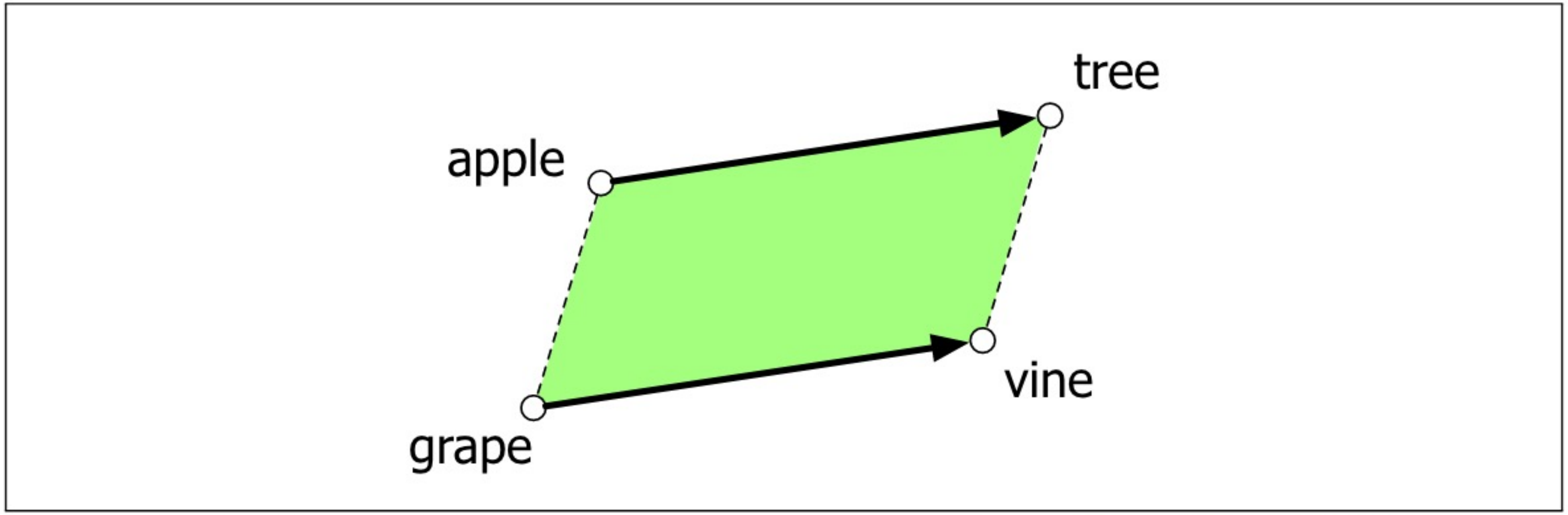


Figure 6.15 The parallelogram model for analogy problems (Rumelhart and Abrahamson, 1973): the location of $\overrightarrow{\text{vine}}$ can be found by subtracting $\overrightarrow{\text{apple}}$ from $\overrightarrow{\text{tree}}$ and adding $\overrightarrow{\text{grape}}$.

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

Option 3: Shallow Neural Networks

Word2vec

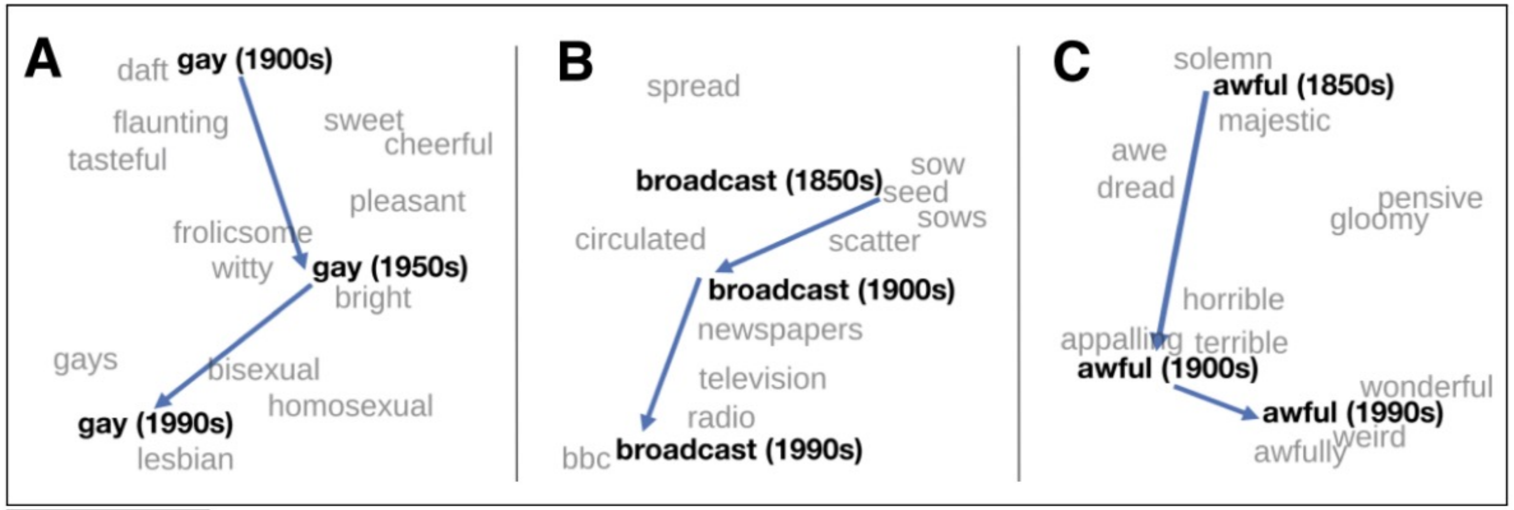
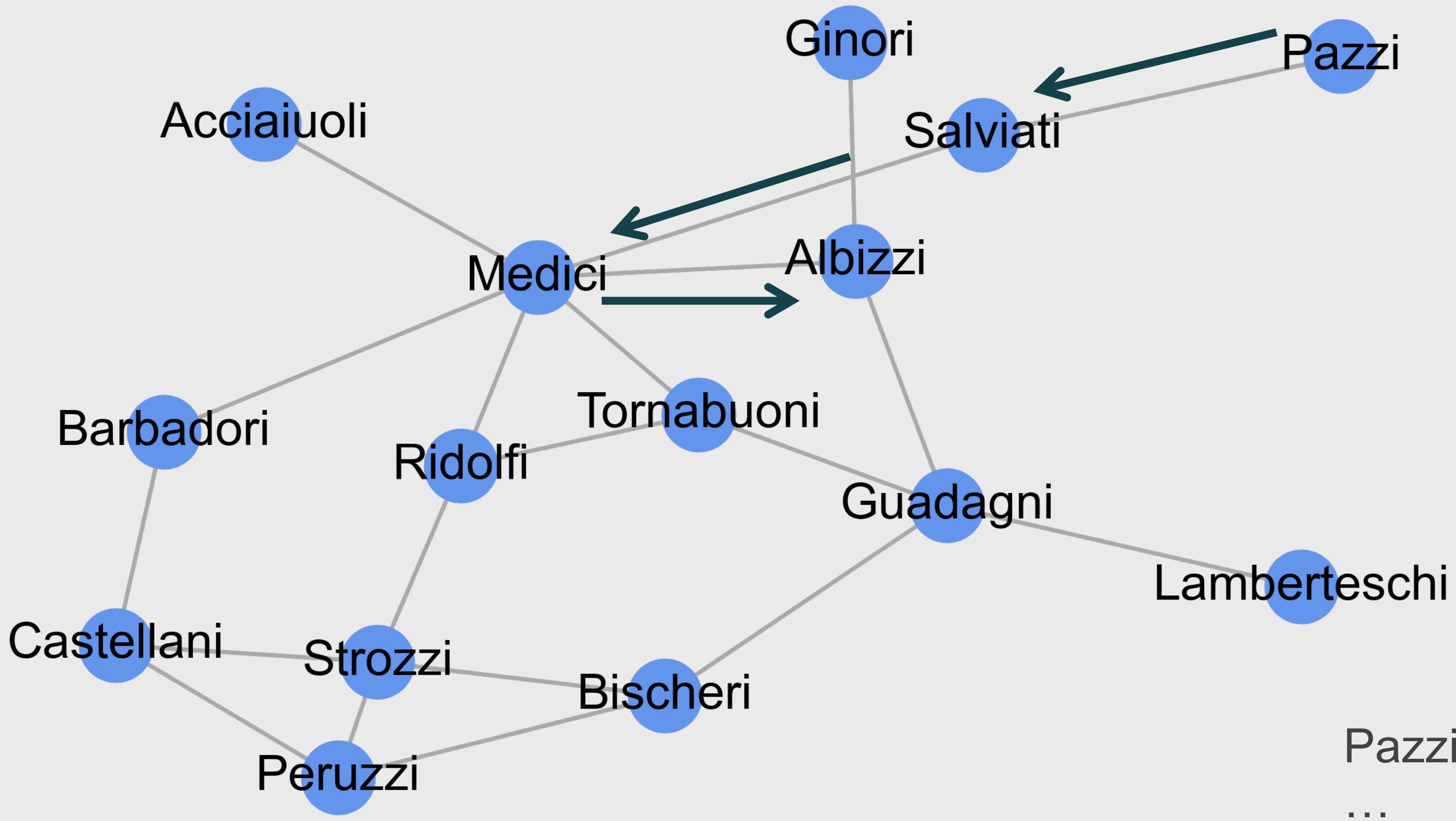


Figure 6.17 A t-SNE visualization of the semantic change of 3 words in English using word2vec vectors. The modern sense of each word, and the grey context words, are computed from the most recent (modern) time-point embedding space. Earlier points are computed from earlier historical embedding spaces. The visualizations show the changes in the word *gay* from meanings related to "cheerful" or "frolicsome" to referring to homosexuality, the development of the modern "transmission" sense of *broadcast* from its original sense of sowing seeds, and the pejoration of the word *awful* as it shifted from meaning "full of awe" to meaning "terrible or appalling" (Hamilton et al., 2016b).

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

In networks: node2vec (deepwalk)

Generate “sentences” using random walks.
The more times two nodes appear in the same random walk, the more similar their embeddings will be



Pazzi -> Salviati -> Medici -> Albizzi
…

Node2Vec (deepwalk)

Step 1: Create co-occurrance matrix

Pazzi -> Salviati -> Medici -> Albizzi

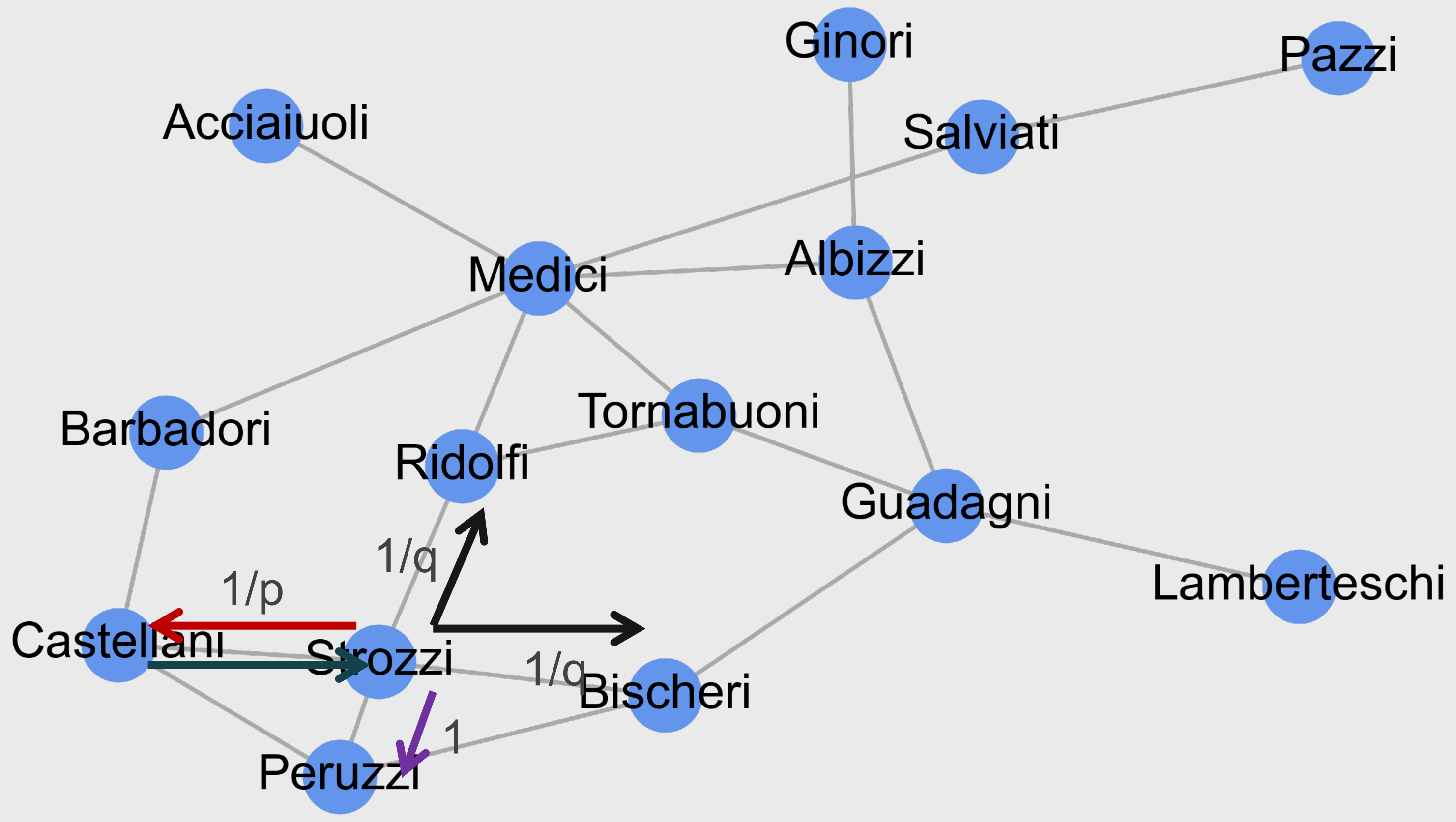
Medici -> Albizzi -> Guadagni -> Medici

…	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)

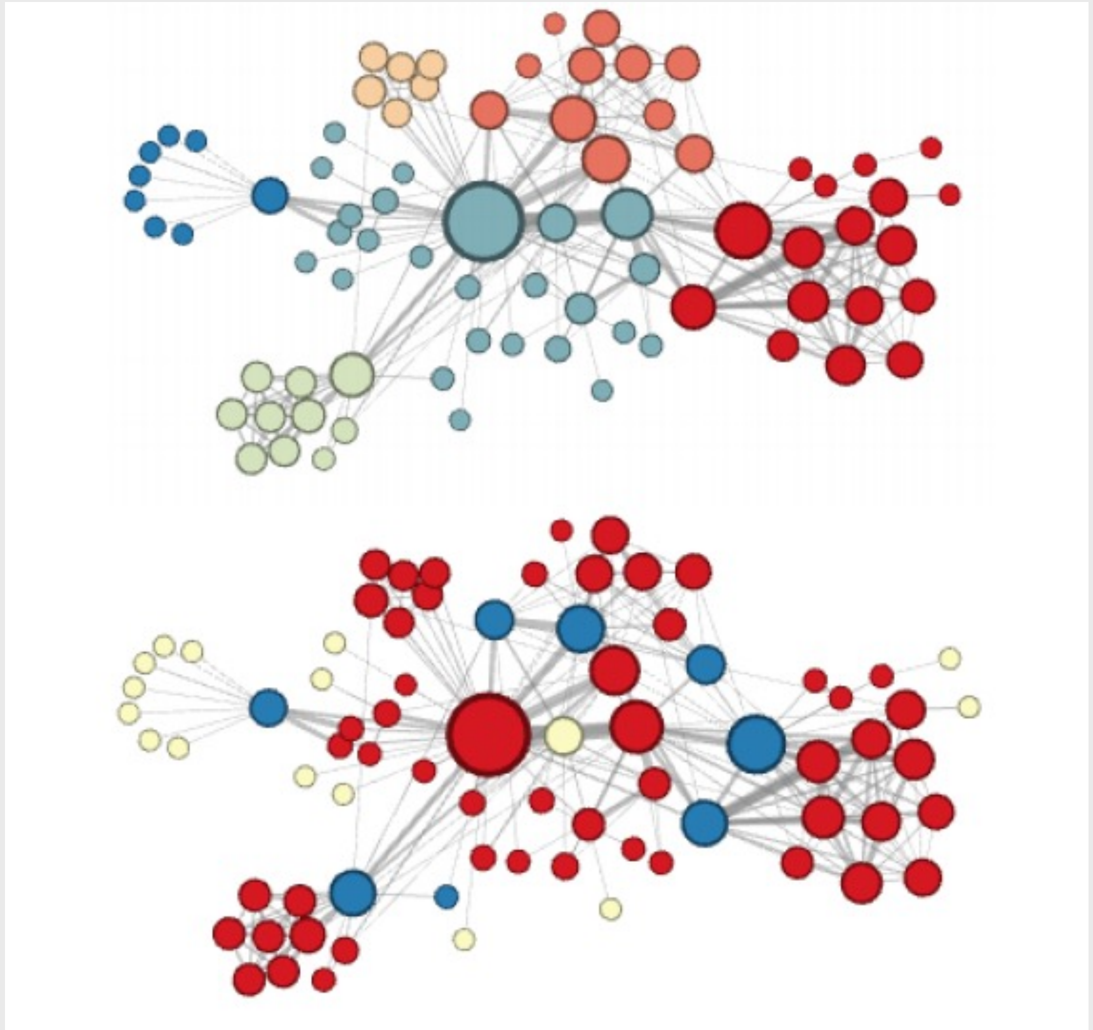


Figure 3: Complementary visualizations of Les Misérables coappearance network generated by *node2vec* with label colors reflecting homophily (top) and structural equivalence (bottom).

q = 0.5

~ similiarity reflecting clusters

q= 2

~ similiarity reflecting structural roles

What to do with the embeddings

1) **Node classification**

X = Embedding

Y = whatever we want to predict

2) Link prediction

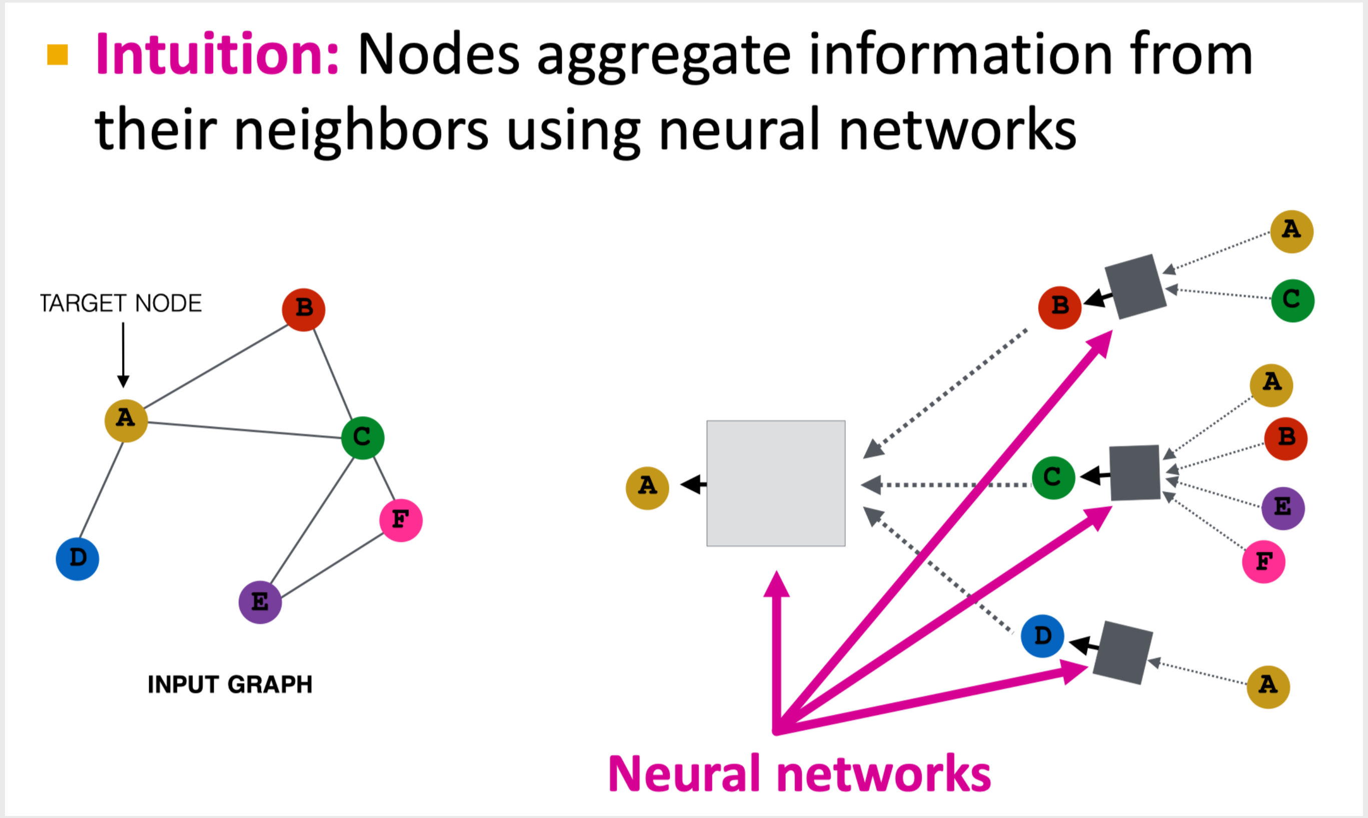
X = combination of the two embeddings (e.g. dot product, or np.abs(v1-v2))

Y = link/no link

Option 4: Deep neural networks

We 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 they are related to the same outcome.**

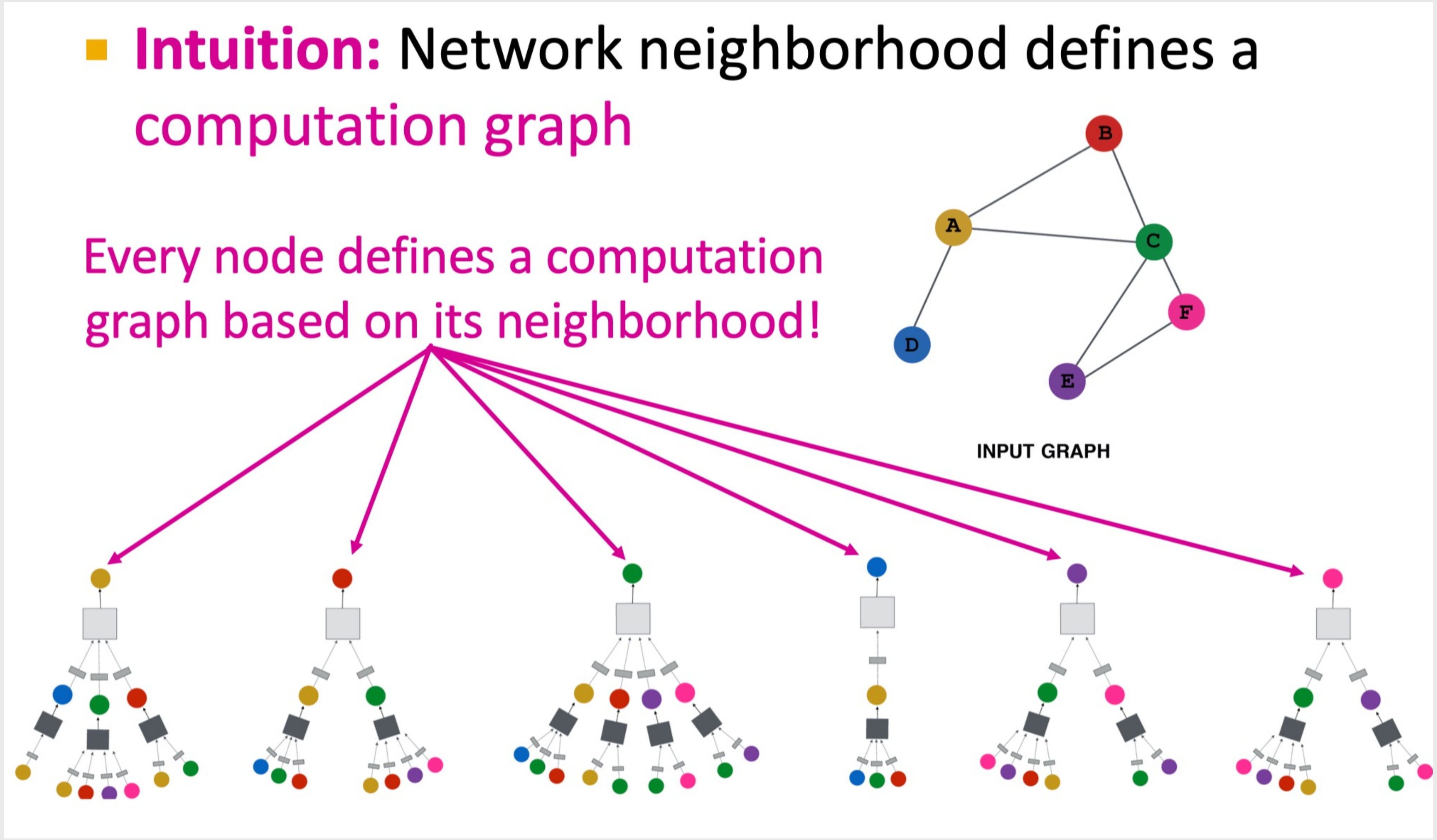
Many methods (GCN, GAT, graphSAGE...)



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

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

Option 4: Deep neural networks



- **Intuition:** Network neighborhood defines a computation graph

Every node defines a computation graph based on its neighborhood!

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

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
- Task: Goal
- Performance measure: Accuracy, R^2, etc

Machine learning

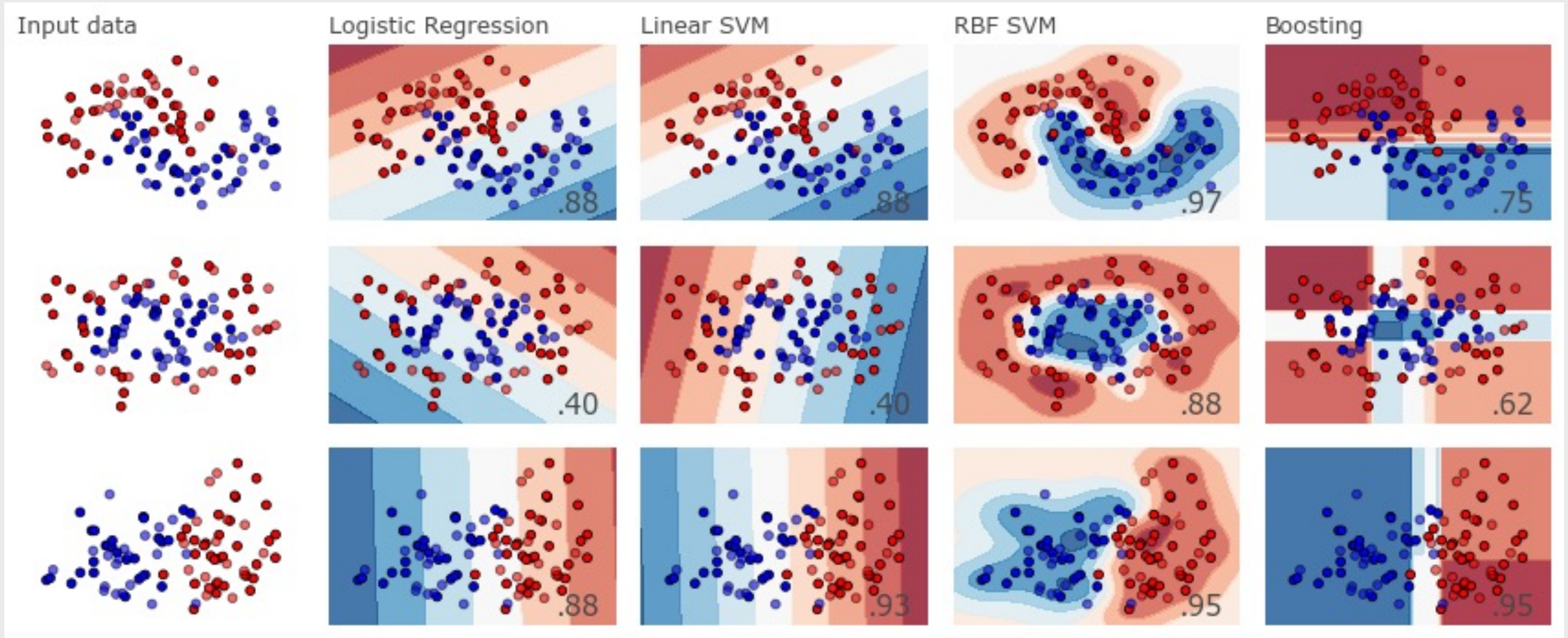
- Typically focuses on **large**, **high-dimensional** datasets with **complex interactions** between features

- **Output-driven**:
  - Typically aims to solve a problem (rather than to test a hypothesis): e.g. link prediction
  - Emphasizes predictive accuracy: Uses theory (to build new features) if it improves accuracy
  - The model works as intended, on new data

Cannot and should not replace thinking about causation

One use in link prediction: stacking classifiers

- 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
- Methods for this: (Penalized) logistic regression; Support Vector Machines; Boosting



Main issues in Machine Learning

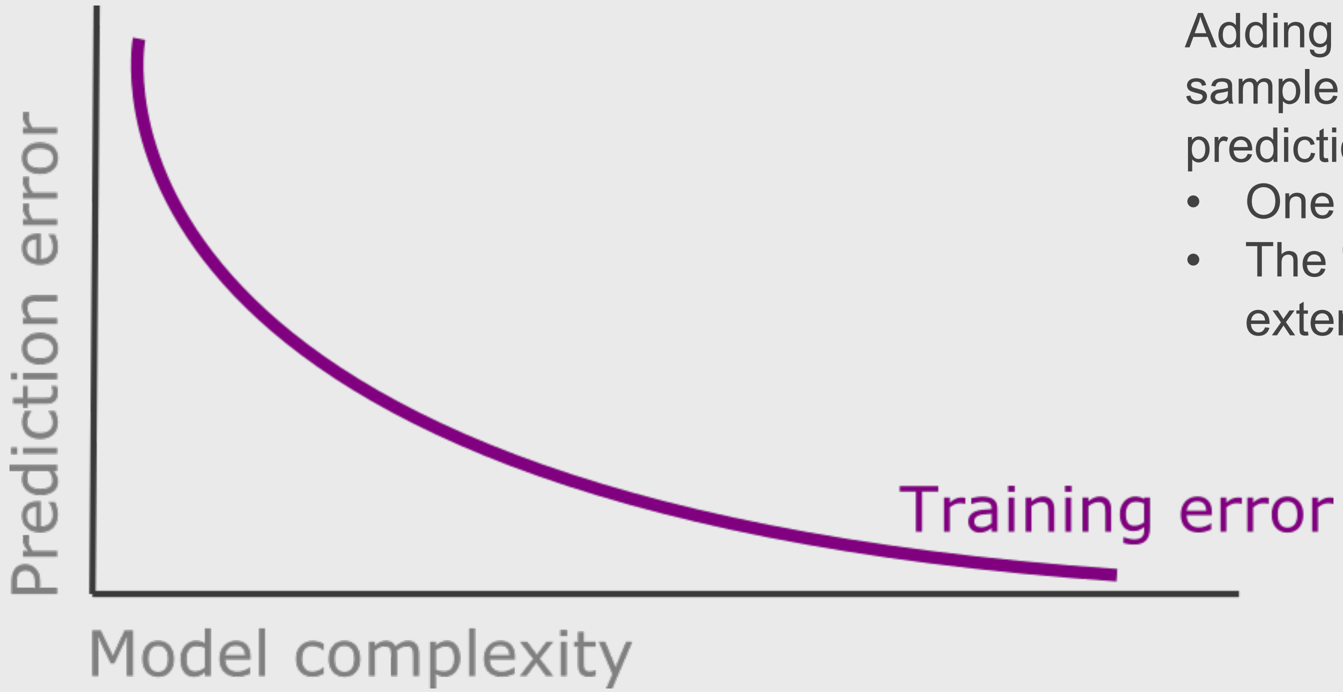
Issue 1: Overfitting

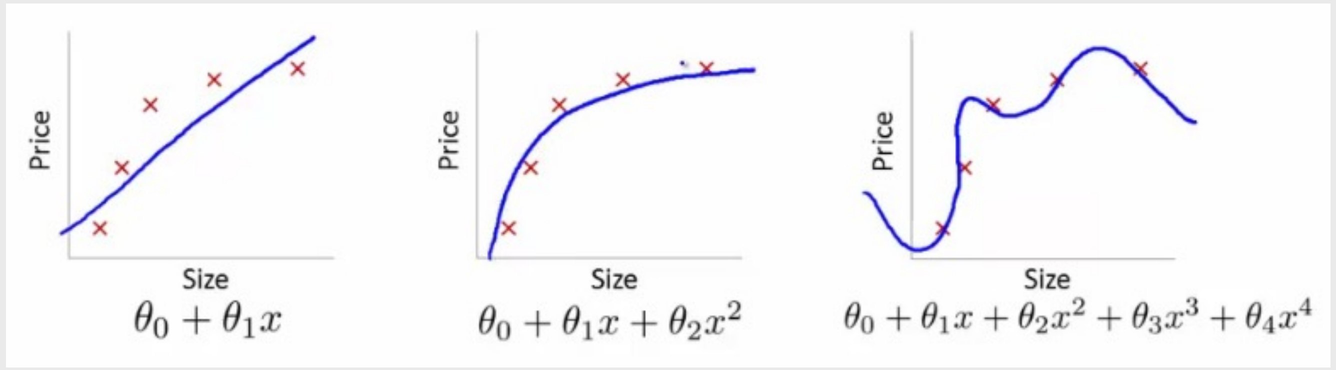
- Lots of data and features → can be a recipe for disaster
- Evaluation of overfitting: cross-validation
- Prevention of overfitting: Regularization, weak learners, dropout, etc

Issue 2: Interpretability of the model

- Complex models are more difficult to interpret
- New measures of interpretability

Issue 1: Overfitting

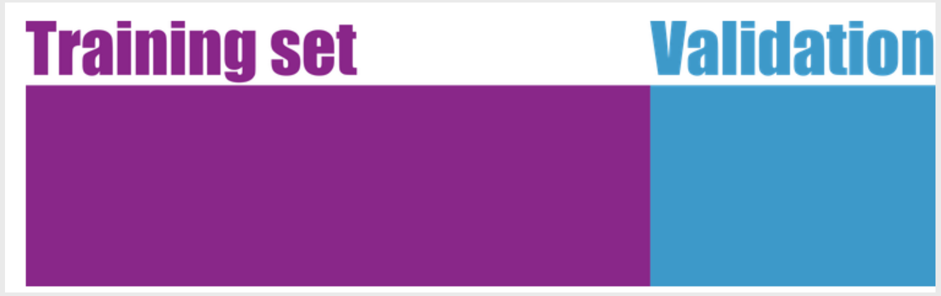




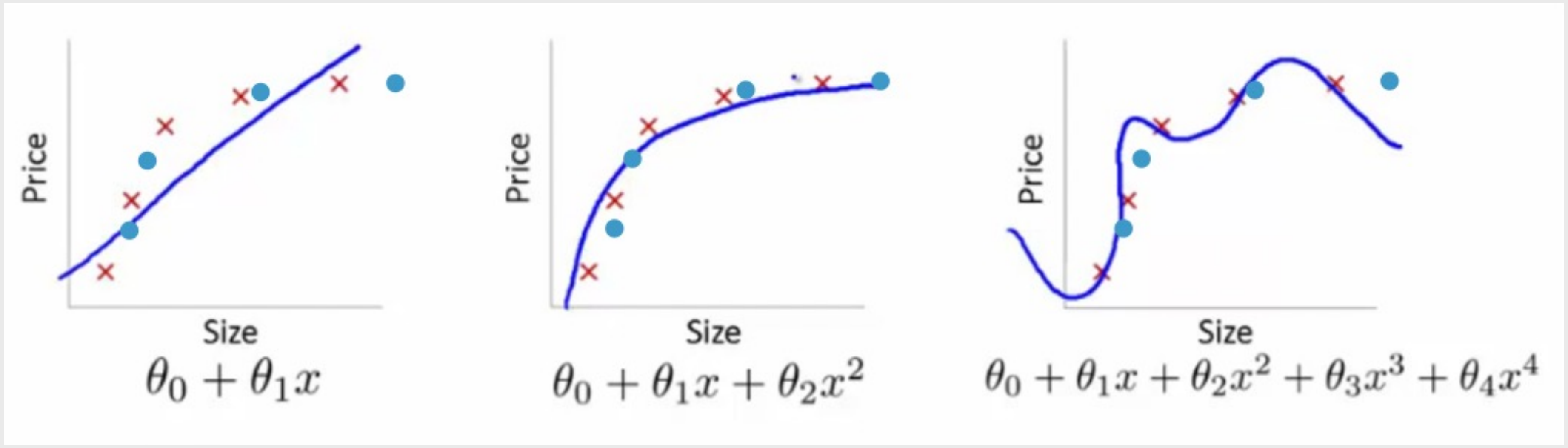
Adding more variables always decreases R^2 (within-sample prediction error). How to estimate real prediction error?

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

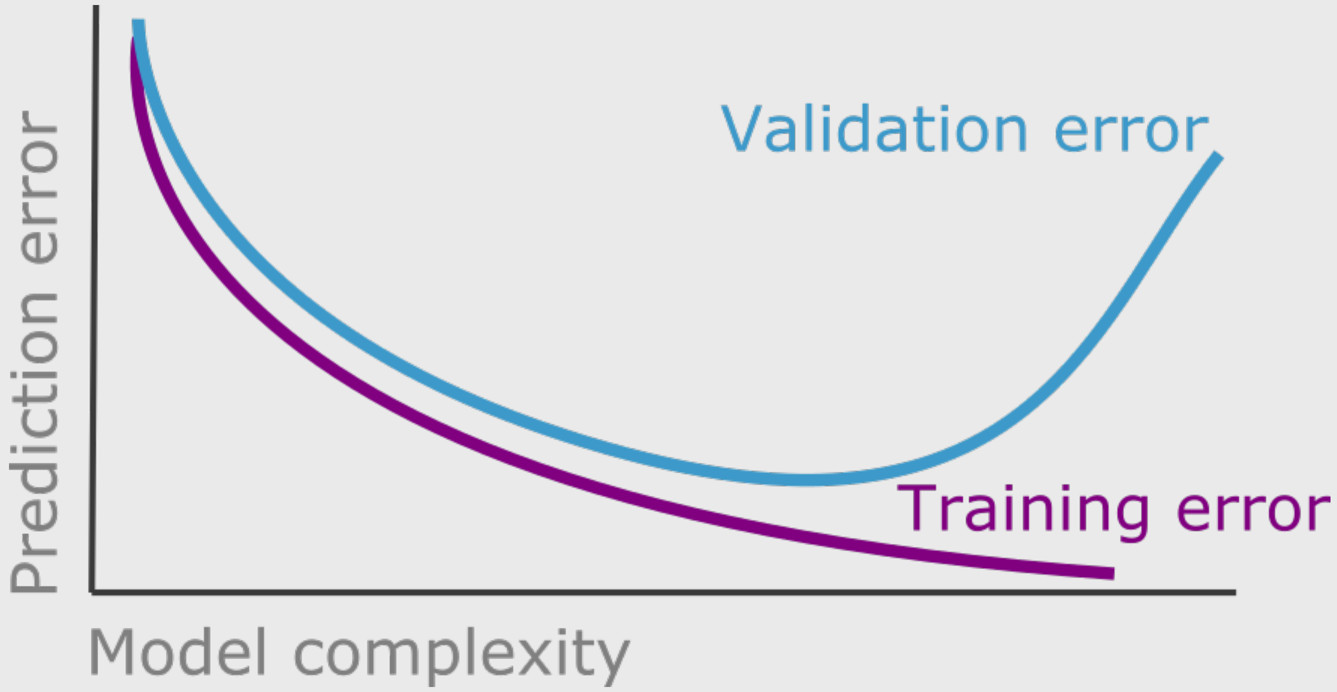
1: Evaluate overfitting using a validation dataset

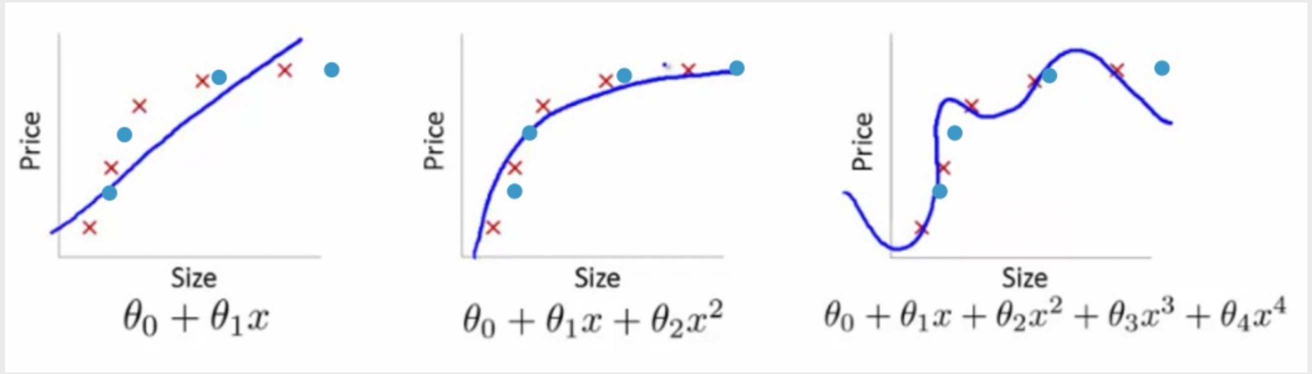


Training dataset → Use to train different models
Validation dataset → Evaluate out-of-sample prediction error
(Test dataset) → Evaluate out-of-sample prediction error of final model



1: Evaluate overfitting using a validation dataset



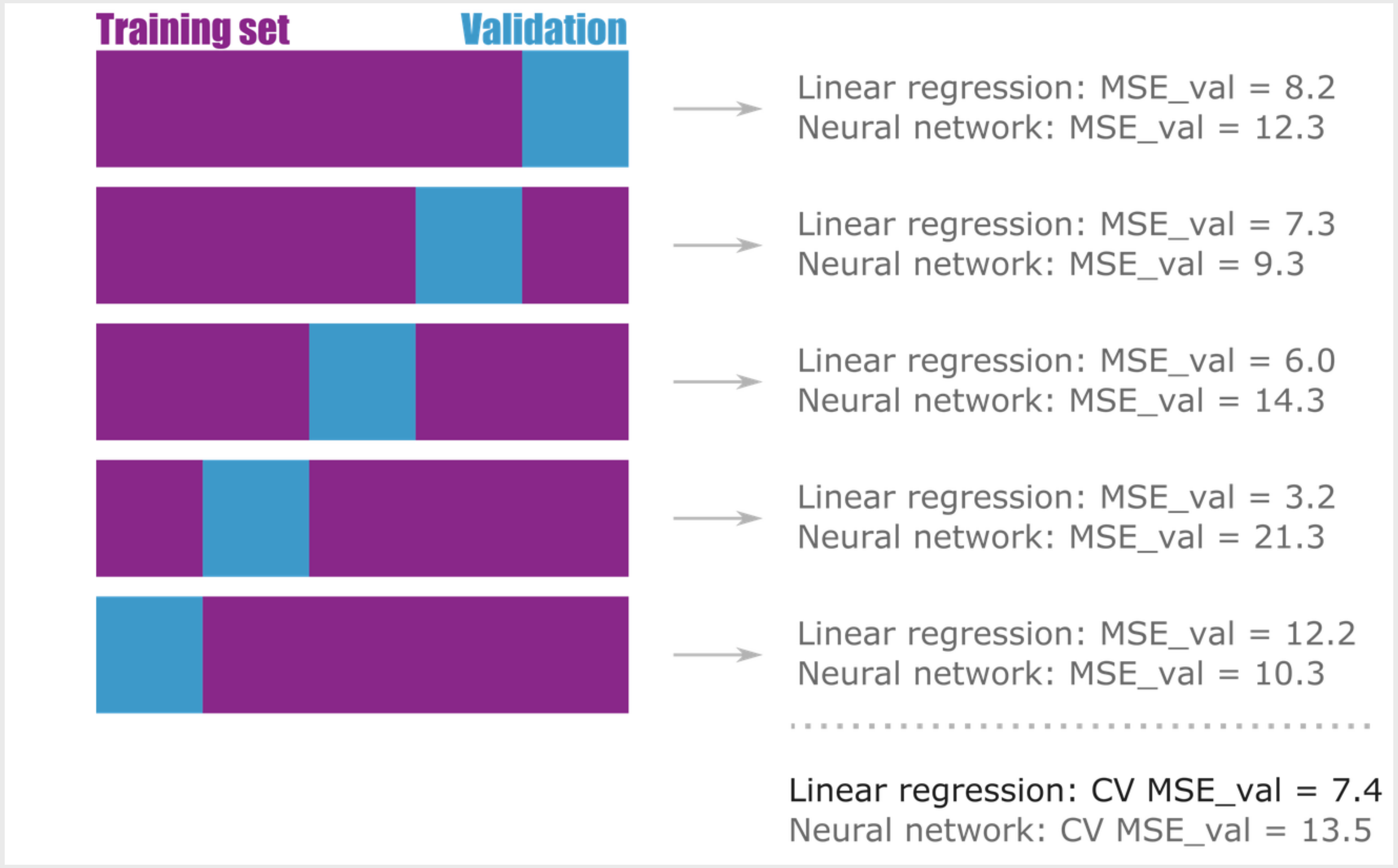


But with this:

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

Solution → Cross-validation

1: Evaluate overfitting using cross-validation



Linear regression: MSE_val = 8.2
Neural network: MSE_val = 12.3

Linear regression: MSE_val = 7.3
Neural network: MSE_val = 9.3

Linear regression: MSE_val = 6.0
Neural network: MSE_val = 14.3

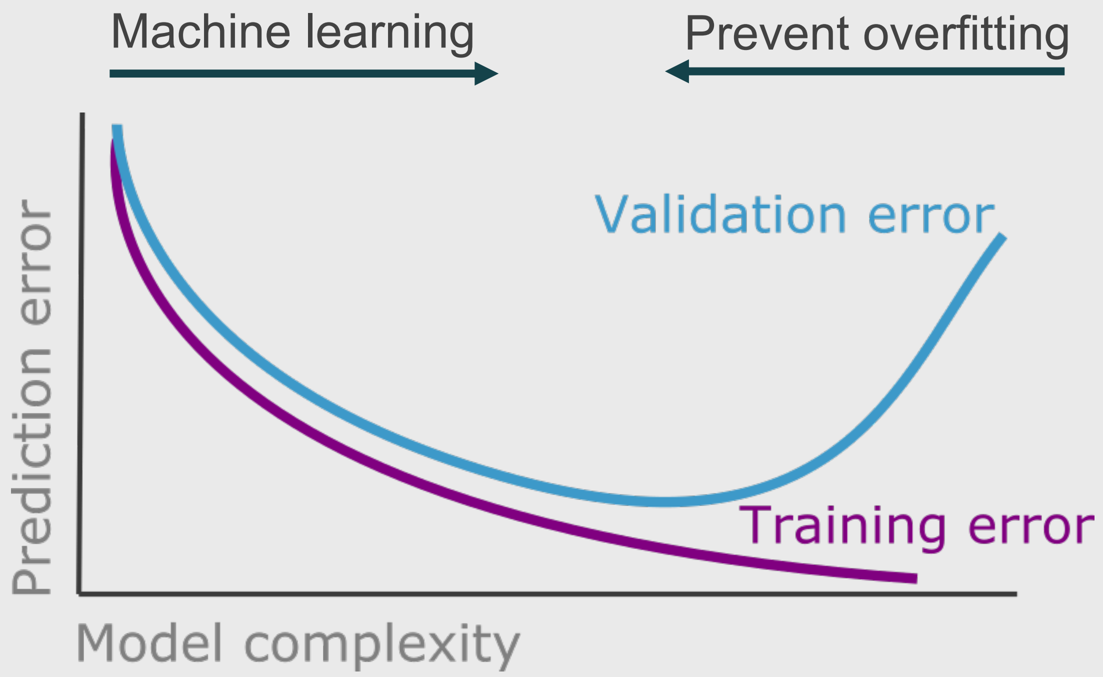
Linear regression: MSE_val = 3.2
Neural network: MSE_val = 21.3

Linear regression: MSE_val = 12.2
Neural network: MSE_val = 10.3

Linear regression: CV MSE_val = 7.4
Neural network: CV MSE_val = 13.5

Do you want to understand the error due to the splitting? → Run this procedure several times with random splittings

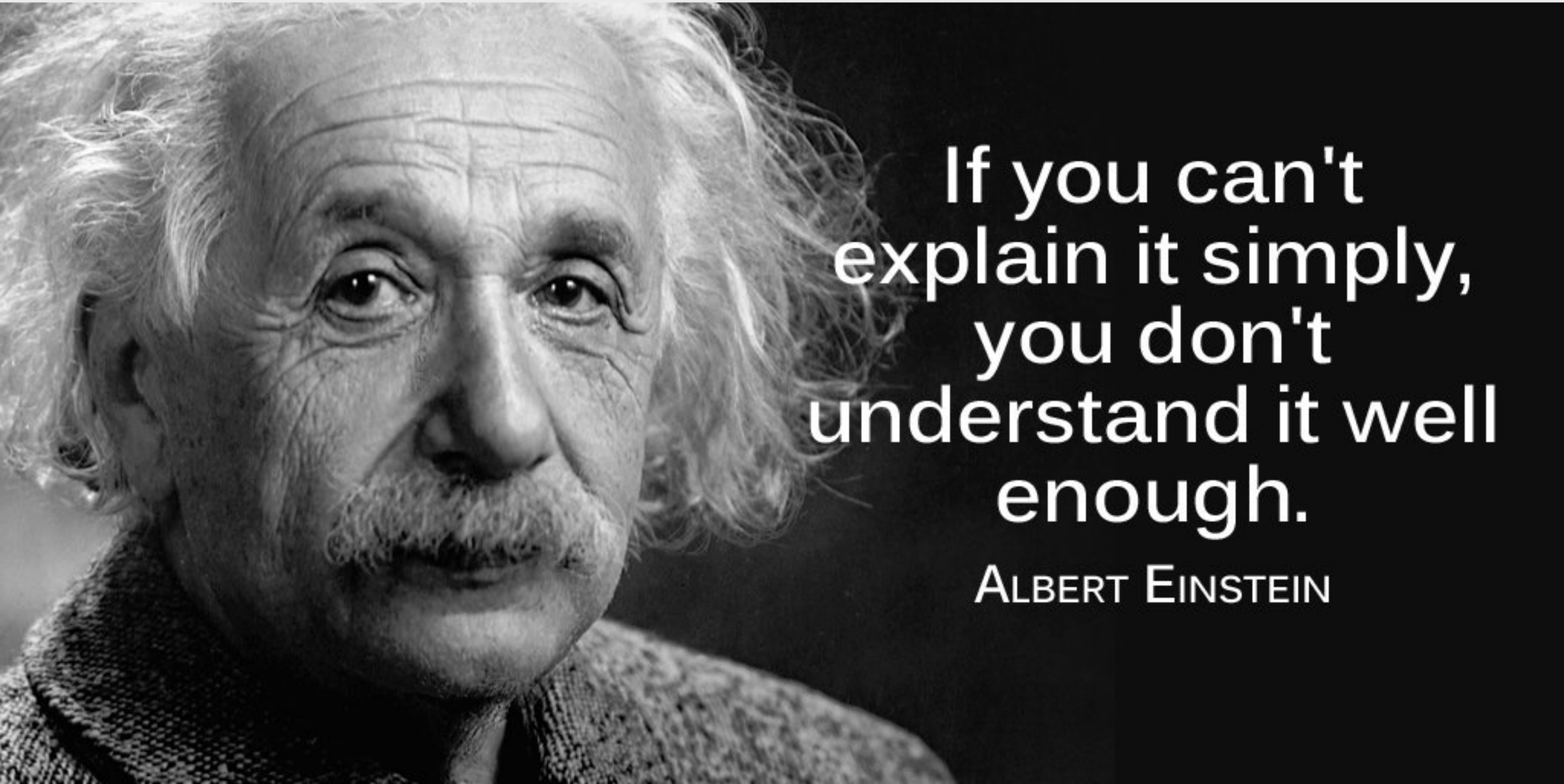
1: Hyperparameter tuning



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

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

Issue 2: Interpretability



If you can't explain it simply, you don't understand it well enough.
ALBERT EINSTEIN

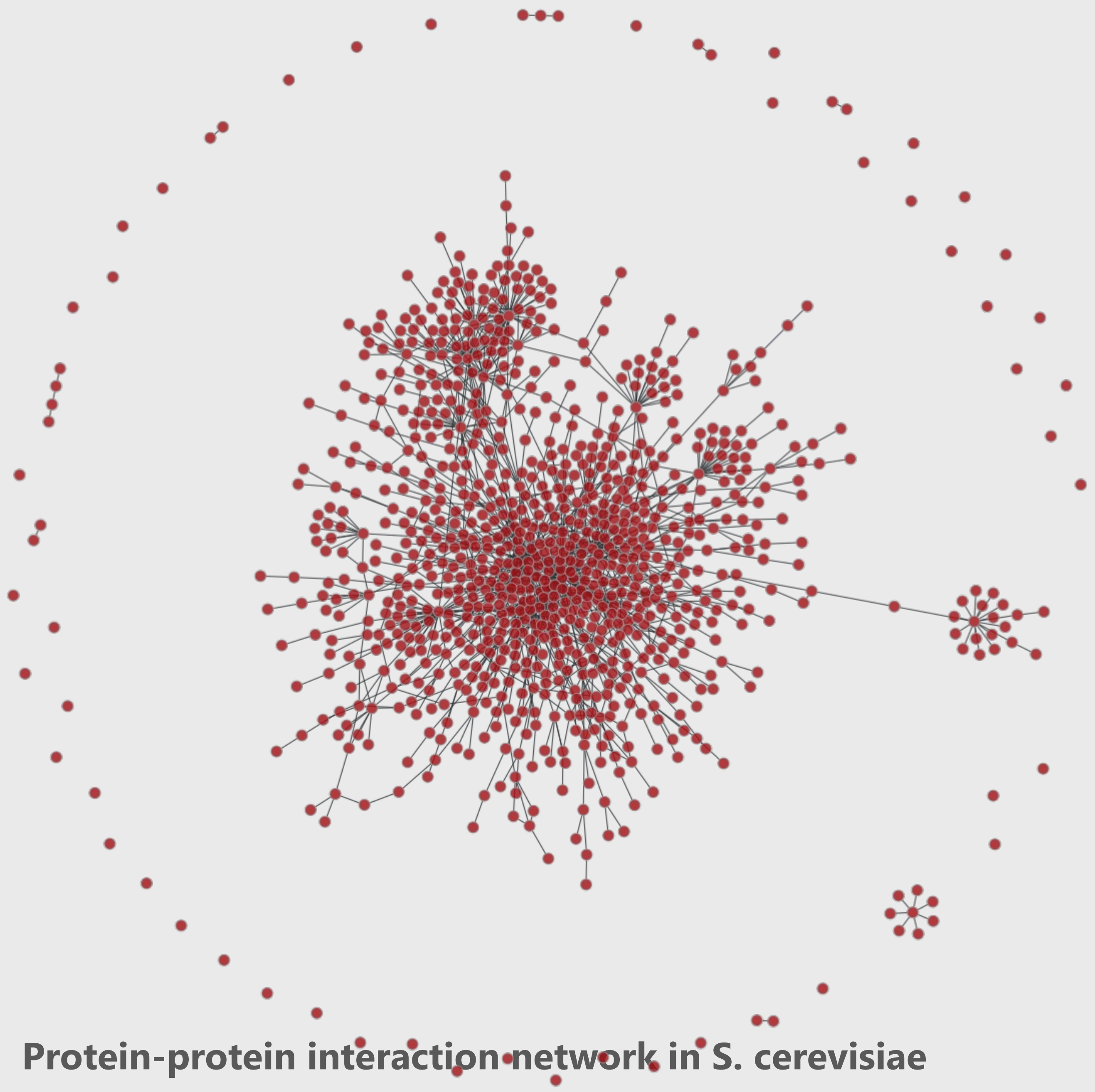
Typical workflow

```
# Split data into training and testing to evaluate generalization error (more on this later)
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:

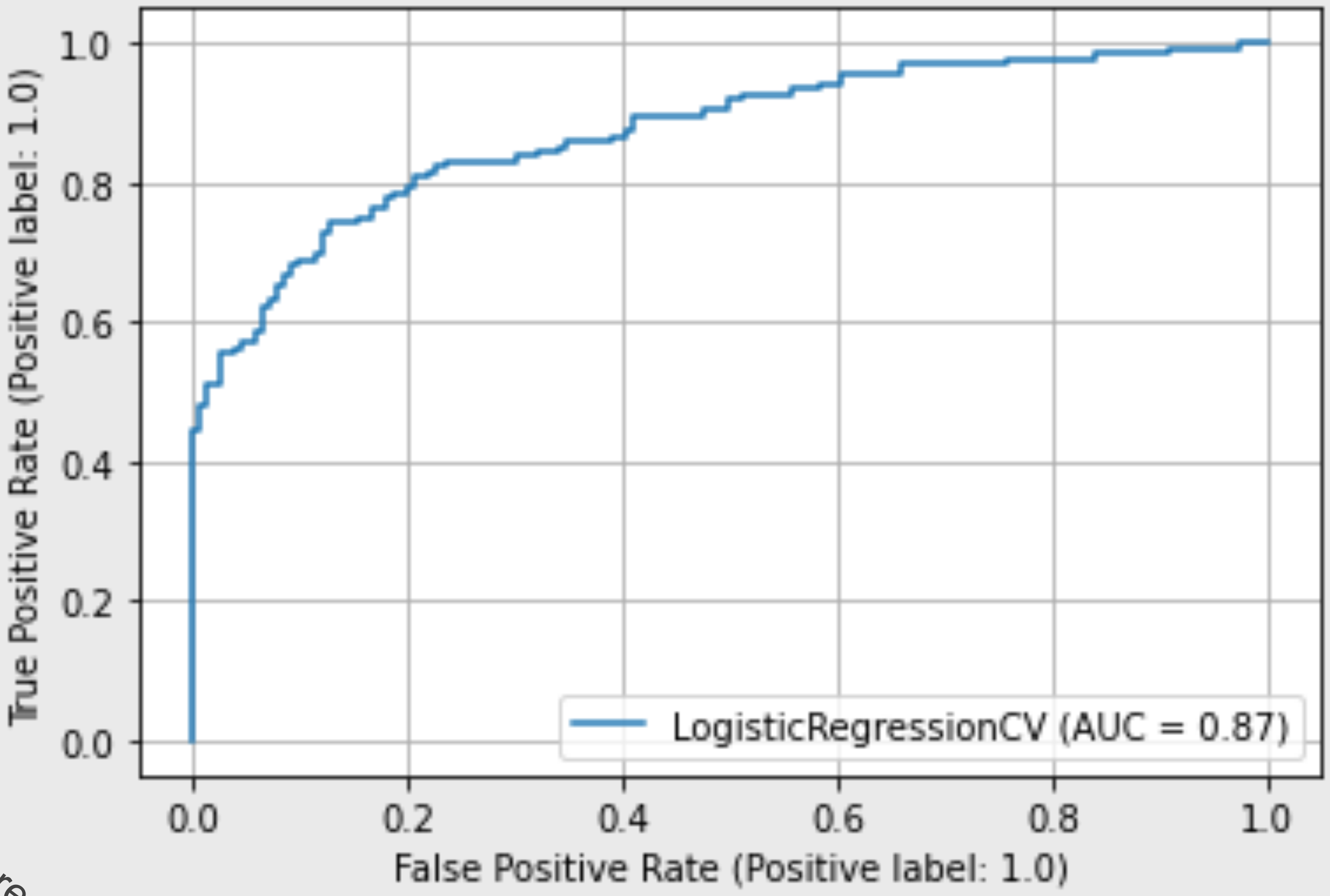
- Graph: Used for training
- Test dataset (a series of node pairs, some with a link associated)

How:

- Methods based on similarity
- Methods based on paths
- SBM
- Methods based on embeddings
  - Spectral methods
  - Matrix factorization
  - Node2vec

Evaluation

(all links predicted correctly)



$$TPR\ (sensitivity) = \\ = \frac{Links\ predicted\ correctly}{Node\ pairs\ with\ a\ link}$$

(no links predicted)

(all no links predicted as links)

$$FPR = \frac{No\ links\ predicted\ as\ links}{Node\ pairs\ with\ no\ link}$$