Advanced Analytics in Business [D0S07a]
Big Data Platforms & Technologies [D0S06a]

Social Network Mining
NoSQL
Graph Databases

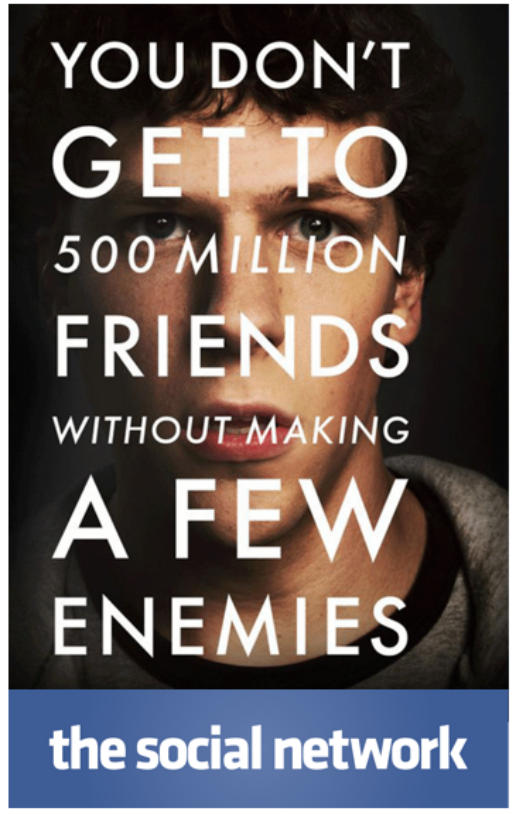
Overview

- Social network construction
- Social network metrics
- Community mining
- Relational learners
- Propagation techniques
- Featurization
- Example
- A word on validation
- Node2Vec and friends
- Tooling
- NoSQL
- Graph databases

Graphs are everywhere

Social networks

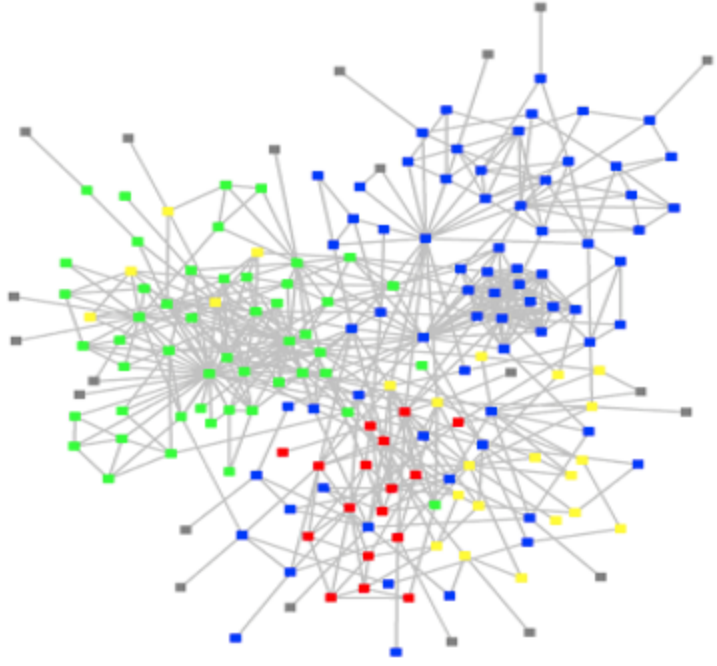
- Web pages connected by hyperlinks
- E-mail traffic
- Research papers connected by citations
- Financial transactions
- Telephone calls
- Social networks: LinkedIn, Facebook, Twitter, MySpace, Friendster, Xing, ...



Applications

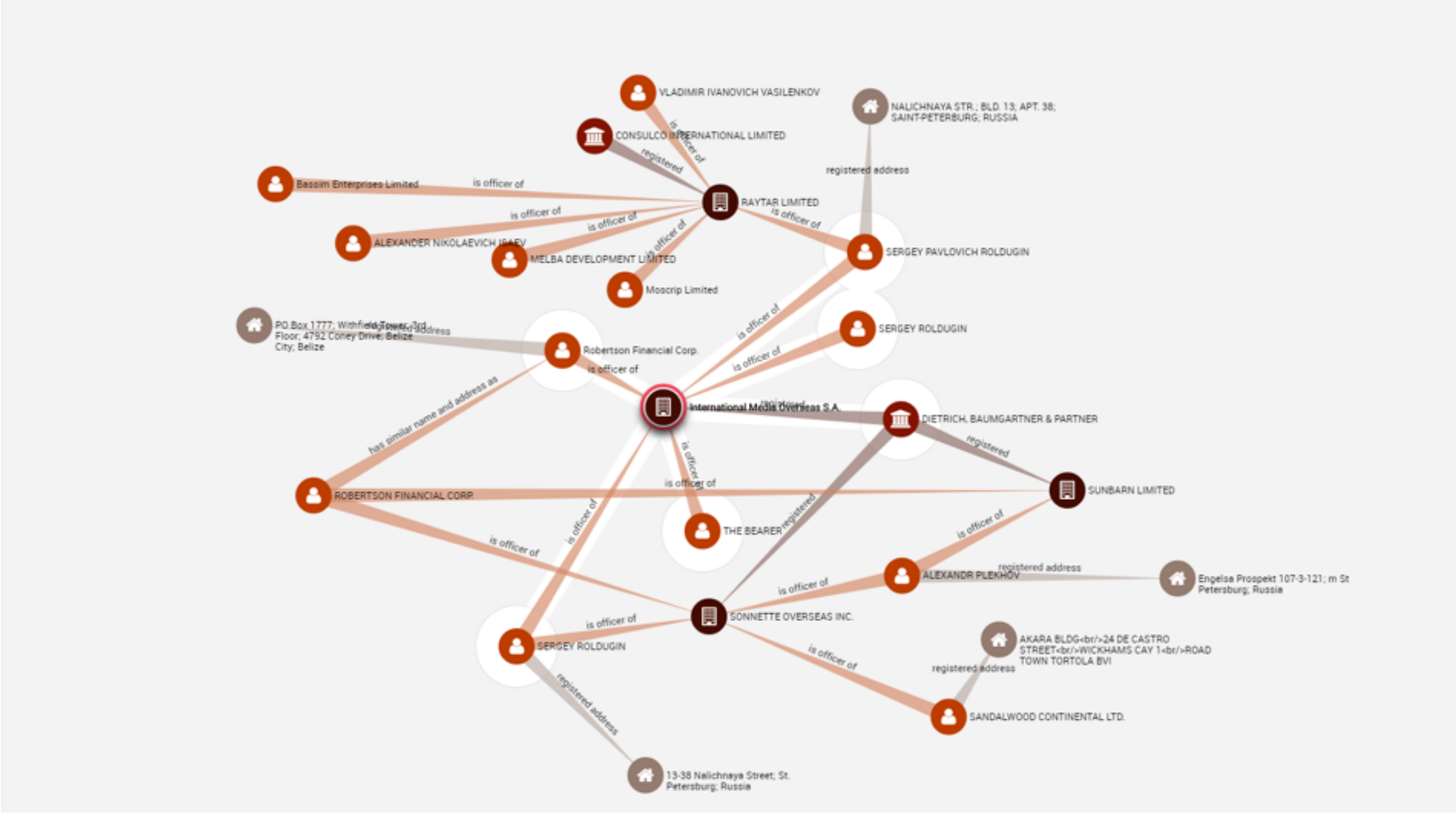
- Web community mining and web page classification
- Fraud detection
- Terrorism detection (suspicion scoring)
- Product recommendations
- Churn detection
- Epidemiology (spread of illness)

Graphs are everywhere

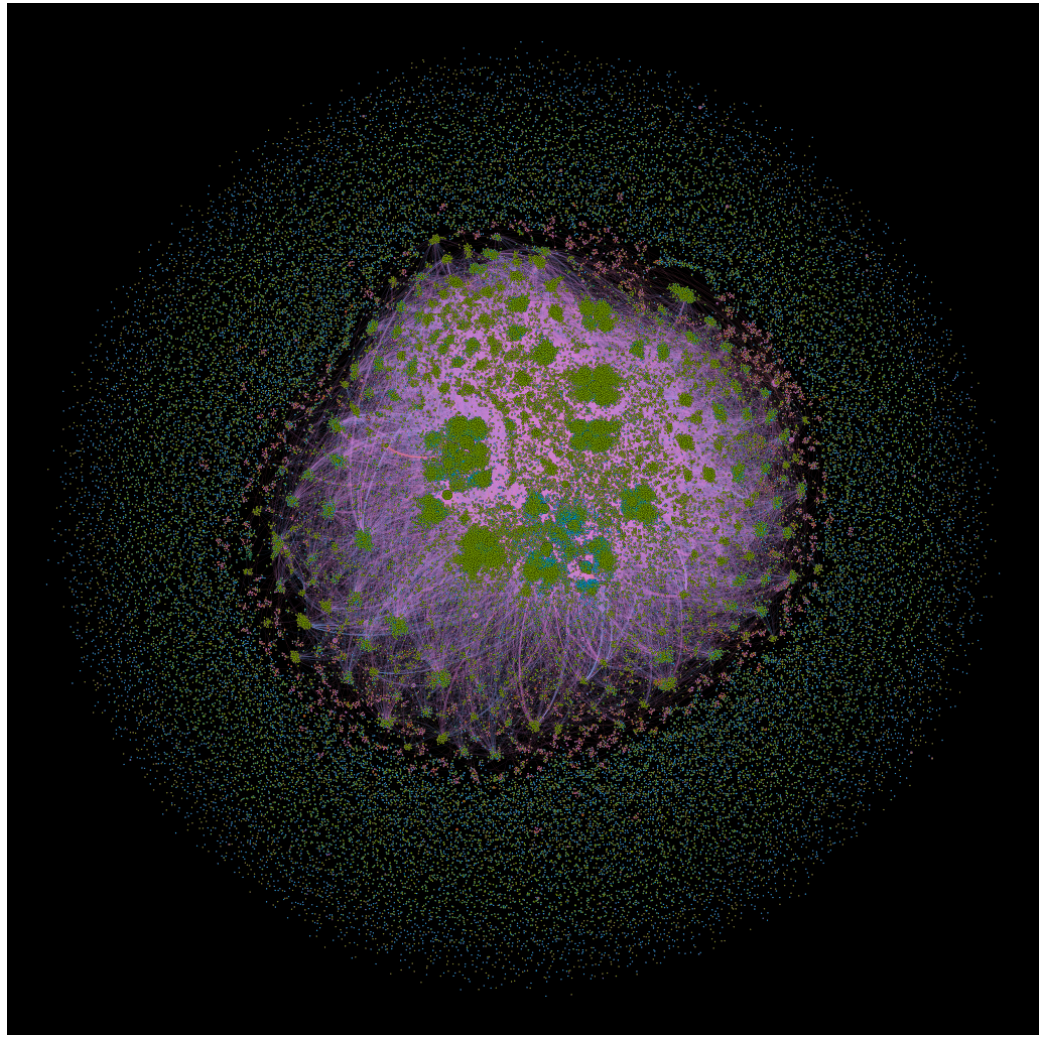


- E-mail flows amongst a project team
- Each person represented by a node; each node colored according to person's department
- Yellow nodes are consultants; grey nodes are external experts
- Built based on email's To: and From: fields

Graphs are everywhere



Graphs are everywhere



But is is unstructured

(Same as text mining...)

Or "semi-structured", rather, though still:

- No direct "feature vector" representation
- No linguistic issues, though featurization will still require a high degree of wrangling

Step one is to define/construct the network, which is already tricky in itself:

- Has an impact on the results, findings, outputs, predictions, ...
- Therefore, you need to take into account how (which techniques will you apply) and why (what is the objective, required output of the analysis) you will use it

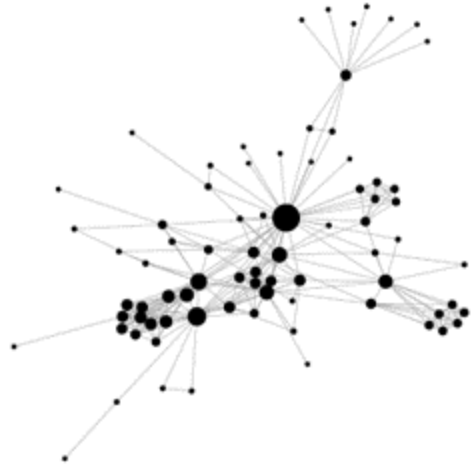
Network components

Nodes (vertices): the "actors" in the network

- People, companies, customers, authors, webpages, ...

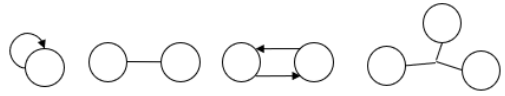
Edges (links): the "interactions" between the actors

- Friendship, co-authorship, transaction, a like



Network components

Things to consider:



- Note types and attributes
  - Do all nodes in your network represent the same entity type, or do you have multiple types of nodes (e.g. products and customers, "bipartite" or "unipartite"?)
  - Apart from a type, do nodes contain other attributes as well? (E.g. age, gender, address...)
  - Does it make sense to encode some of these attributes in their own node type (e.g. for address, this does in many use cases: think about possible interactions)
- Edge types, directionality, cardinality, and attributes
  - Do all edges represent the same interaction type, or do you have multiple edge types (e.g. `customer-[bought]->product` and `customer-[follows]->customer`)
  - Apart from type, do edges contain other attributes as well?
  - A weight on edges as a replacement for binary presence is very common, though might contain more than this alone
  - Are edges directed or not, or both?
  - How are self-edges supported? Reciprocal edges?
  - Edges between more than two nodes?
- What forms the centerpoint of analysis?
  - **Predict a label? In most cases, this will be based on the nodes (node carries features and labels)**
  - In cases of edges: might need to re-encode to a node

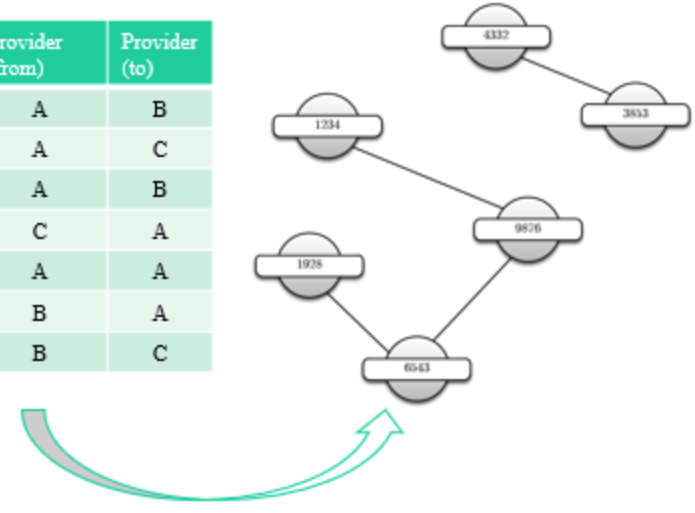
Network components

Basic support most tools: one node type, one edge type (potentially weighted), binary edges, directional

Construction:

- From a graph database (see later): though might still want to re-construct network
- From traditional flat and relational data sources, transactional data

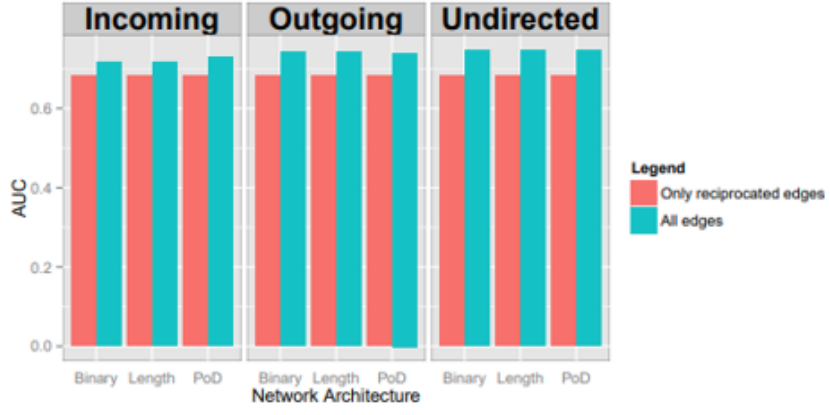
From	To	Duration	Location (from)	Location (to)	Provider (from)	Provider (to)
1234	9876	5:40	Brussels	Leuven	A	B
4567	6543	1:05	Leuven	Leuven	A	C
1234	9876	2:34	Brussels	Leuven	A	B
6543	1928	2:02	Leuven	Ghent	C	A
4332	3853	0:46	Bruges	Antwerp	A	A
9876	1234	3:51	Antwerp	Brussels	B	A
9876	6543	6:57	Leuven	Ghent	B	C

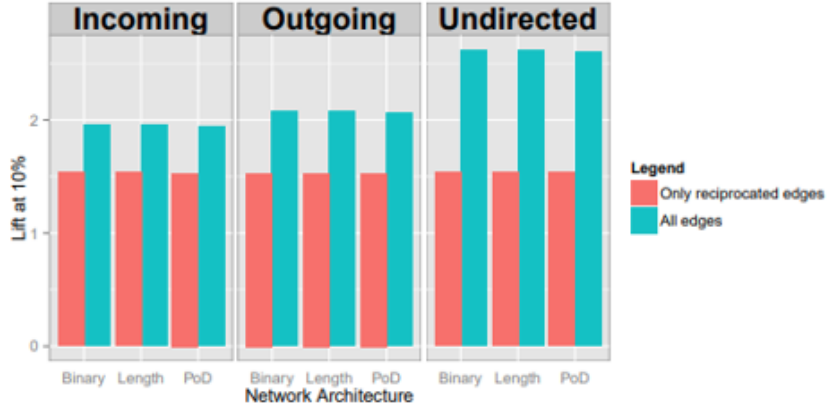


Network components

Start with a simple approach (Oskarsdottir, M., 2018)

- Non-directional rather than directional
- Unweighted
- Binary relationships
- Single node type
- Sufficient / additional relationship / edge information (e.g. **pseudo-edges**) to get a densely connected graph!



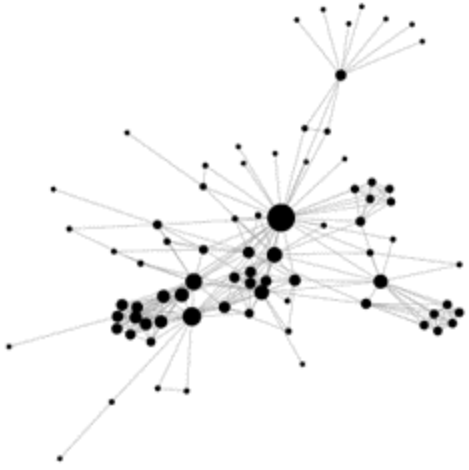


Network representation

Visually: "sociograms"

Note that figuring out an appealing layout requires techniques on its own

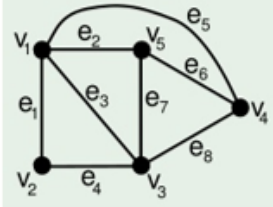
- Can, in many settings, already provide insights without predicting anything



Matrix based: adjacency matrix (node-node) and incidence matrix (node-edge) are common

adjacency

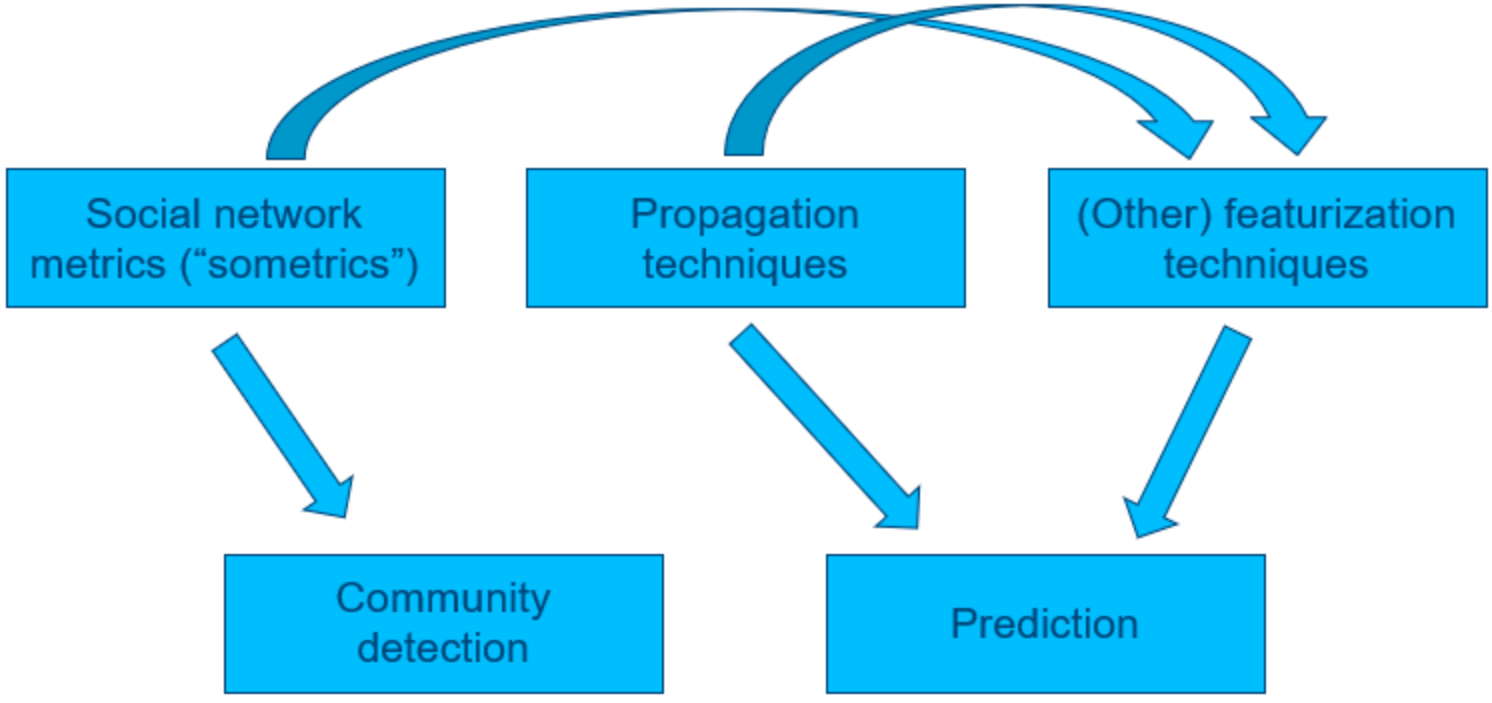
	C1	C2	C3	C4
C1	-	1	1	0
C2	1	-	0	1
C3	1	0	-	0
C4	0	1	0	-



incidence

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
v_1	1	1	1	0	1	0	0	0
v_2	1	0	0	1	0	0	0	0
v_3	0	0	1	1	0	0	1	1
v_4	0	0	0	0	1	1	0	1
v_5	0	1	0	0	0	1	1	0

Overview


Social network metrics (“sometrics”)
Propagation techniques
(Other) featurization techniques
Community detection
Prediction

Social network metrics

Measuring nodes

Nodes in a social network have different roles and structure within the network

Social metrics (sometrics), centrality measures are used to identify the most important nodes

- Most influential
- Key infrastructure
- Super spreaders

Common centrality measures

- Degree centrality
- Closeness centrality
- Betweenness centrality
- Eigenvector centrality - PageRank

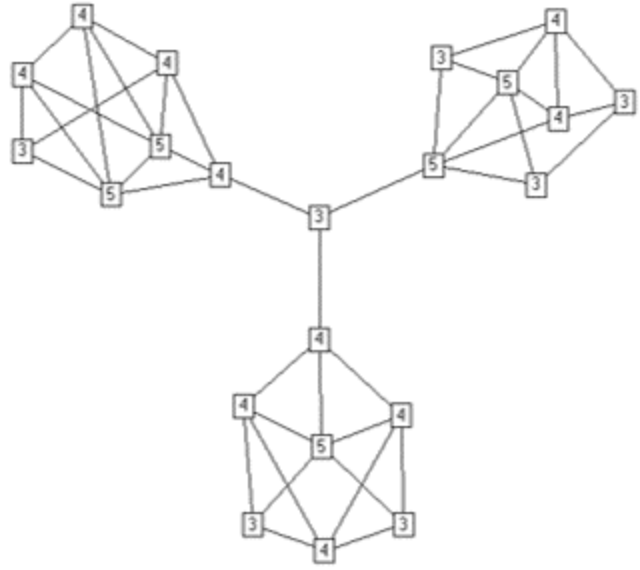
Degree centrality

The degree of a node simply equals the number of edges

- Difference can be made between in-degree, out-degree
- Normalized degree: dividing with the maximum degree

Example: number of friends, followers

- Very simple measure of importance

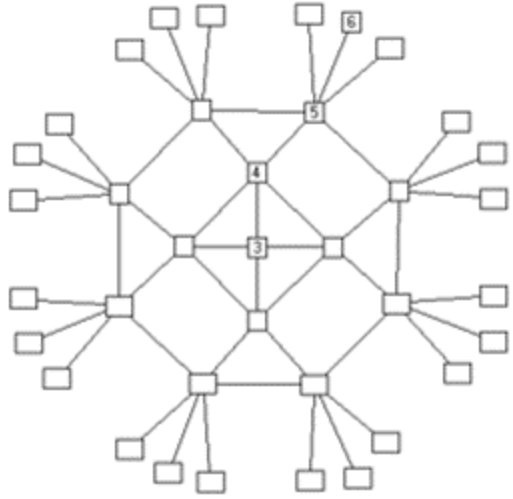


Geodesics

The geodesic path between two nodes is the shortest path between them

- Can include edge weights, or simply consider the same weight for every edge
- Computationally intensive to calculate for large graphs

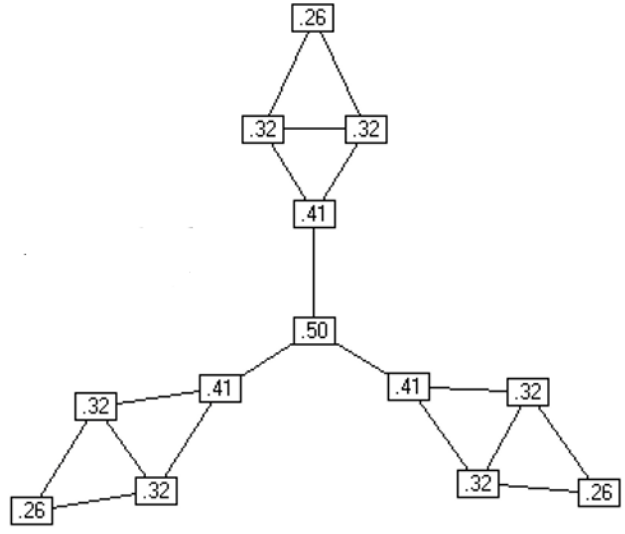
Graph theoretic center: the node(s) with the lowest, maximum distance to all other nodes $argmin_{n \in N}(max(\{|geod(n,m)| : m \in N\}))$



Closeness centrality

The extent to which a node is near all other nodes in the network

- Measures the capacity of a node to reach the rest of the nodes of the network (reciprocal of farness)
- The inverse distance of a node to all other nodes
- Calculated using the geodesic

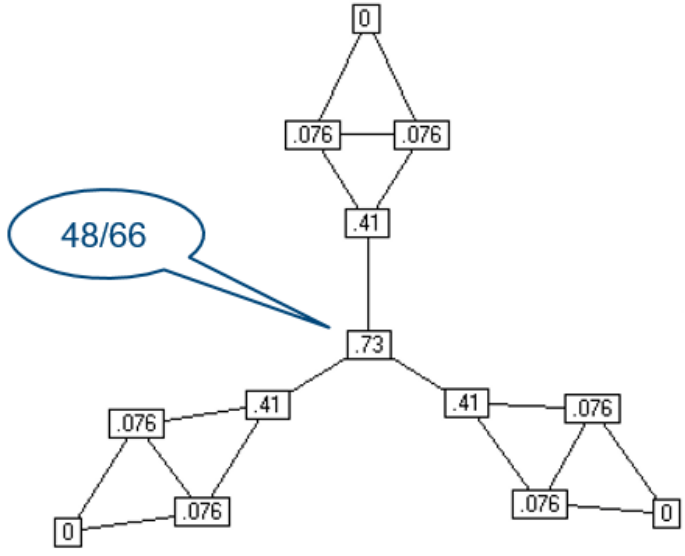


$$C(x) = \frac{N-1}{\sum_{y \neq x} |geod(x,y)|}$$

Betweenness centrality

The number of times a node appears in the geodesics of the network

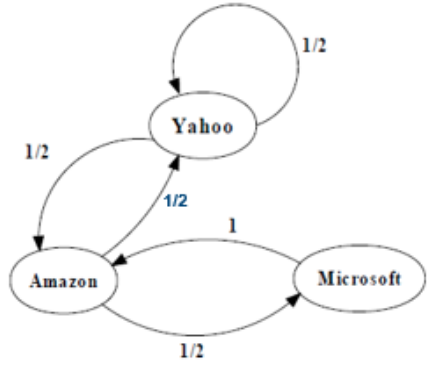
- More information passes through nodes with a high betweenness
- Can also be calculated for edges



Eigenvector centrality (PageRank)

A measure of influence of a node in a network

- Based on the algorithm Google used to rank webpages
- Connections to high scoring nodes contribute more to the score than connections to low scoring nodes



$$M = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix}.$$

$$\begin{bmatrix} \text{yahoo} \\ \text{Amazon} \\ \text{Microsoft} \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

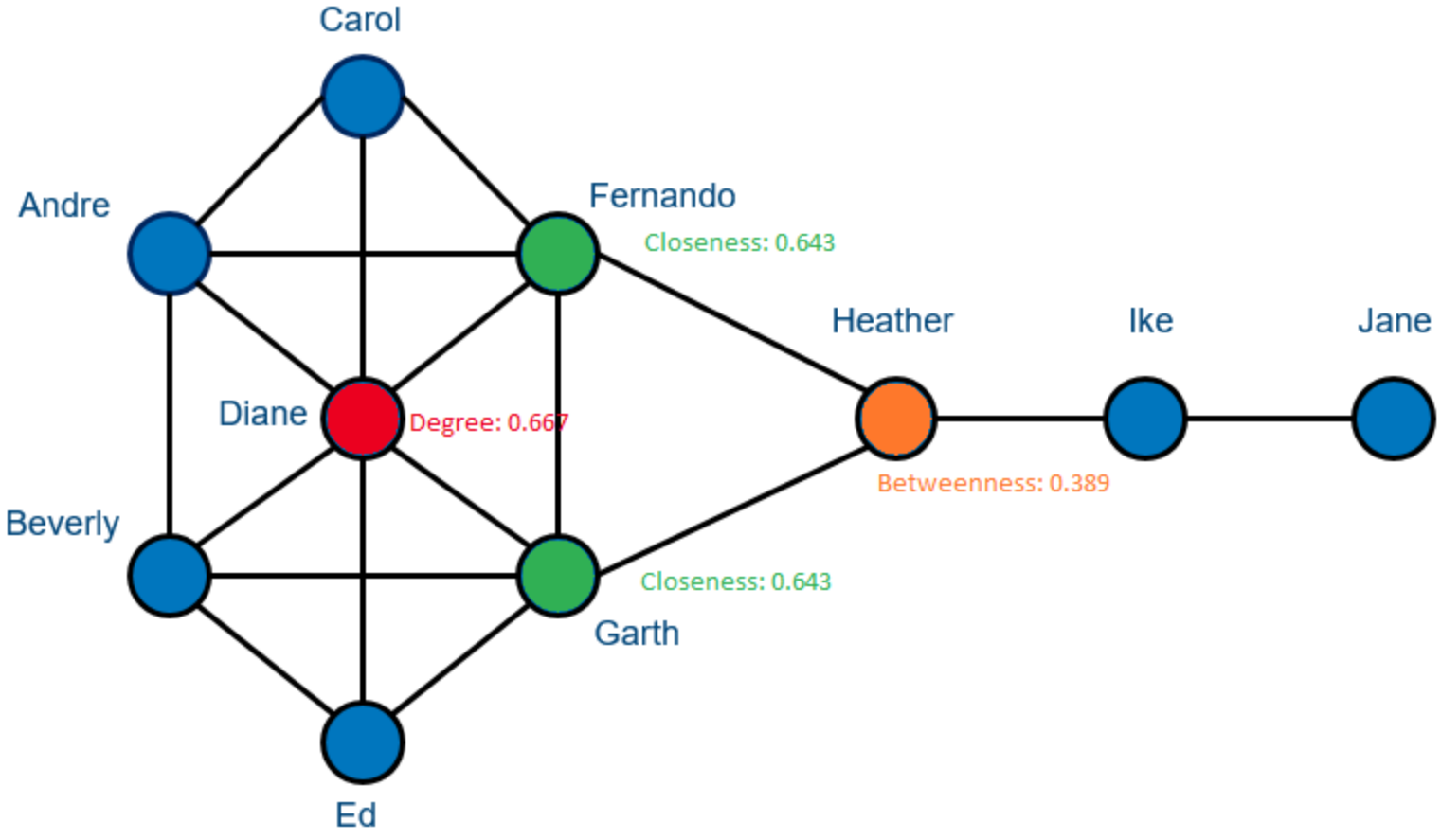
$$\begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1 \\ 0 & 1/2 & 0 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/2 \\ 1/6 \end{bmatrix} \rightarrow \begin{bmatrix} 5/12 \\ 1/3 \\ 1/4 \end{bmatrix} \rightarrow \ldots \rightarrow \begin{bmatrix} 2/5 \\ 2/5 \\ 1/5 \end{bmatrix}$$

Eigenvector centrality (PageRank)

Note: most implementations utilize a smarter approach then a simple matrix convergence to handle scale, loops, disconnected graphs, dangling links...

- Based on "random walkers"

Kite network (Krackhardt 1988)



Kite network (Krackhardt 1988)

Name	Degree	Centrality	Betweeness	PageRank
Andre	0.444	0.529	0.023	0.102
Beverly	0.444	0.529	0.023	0.102
Carol	0.333	0.5	**0**	0.079
Diane	**0.667**	0.6	0.102	**0.147**
Ed	0.333	0.5	**0**	0.079
Fernando	0.556	**0.643**	0.231	0.129
Garth	0.556	**0.643**	0.231	0.129
Heather	0.333	0.6	**0.389**	0.095
Ike	0.222	0.429	0.222	0.086
Jane	**0.111**	**0.310**	**0**	**0.051**

Why?

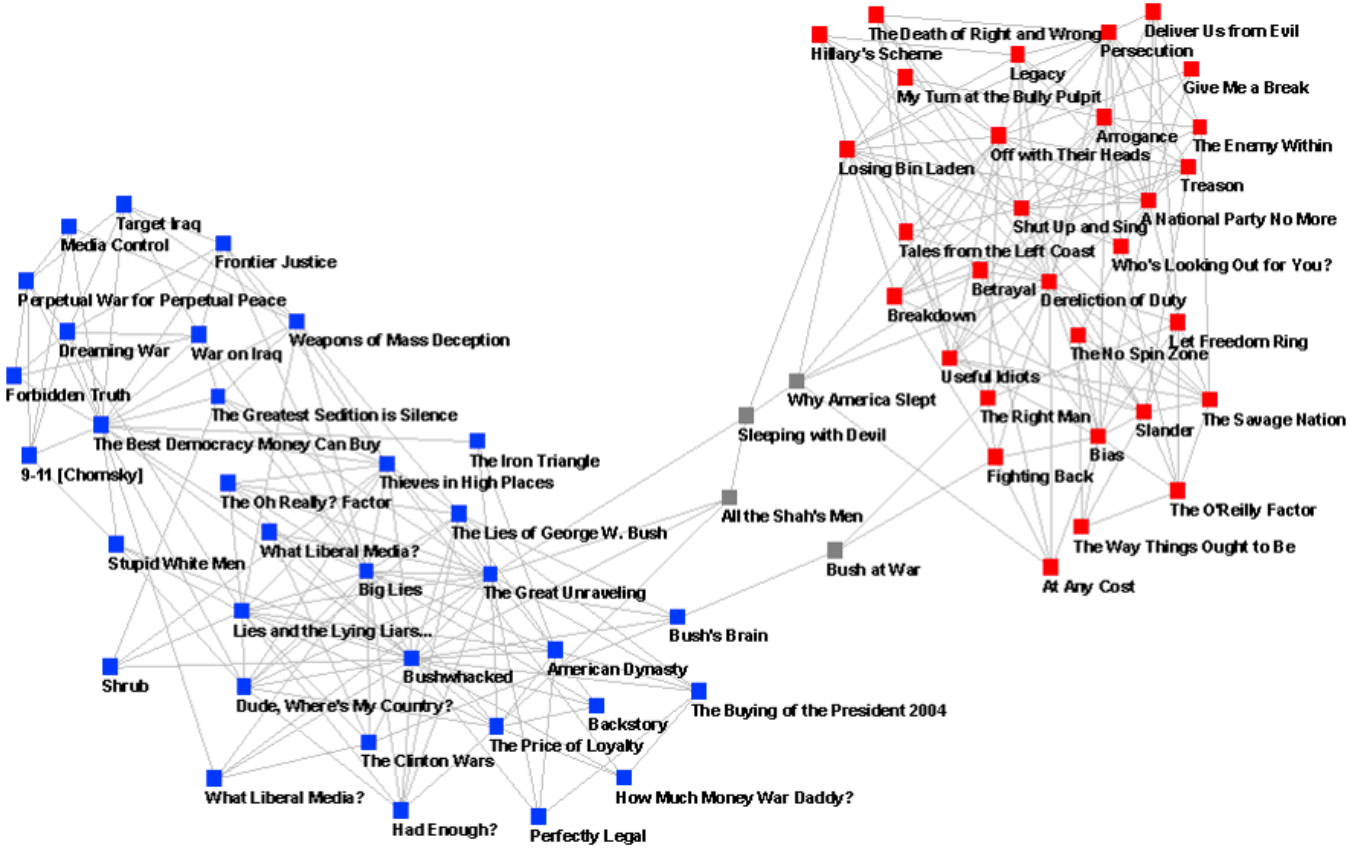
Practical applications?

- Simply for exploratory/visualization purposes
- Combine with filters and visualizations for the class of interest on nodes
- Blocking viral effects: betweenness
- Identifying viral marketing targets
- Who potentially spreads messages, could we reward from passing on to friends, has a large action radius?
- Physical places with high centrality?

Use the values as features to include in your data set

Community mining

Community mining



Community mining

Community mining: finding clusters in a network

- A community is generally described as a substructure (subset of vertices) of a graph with dense linkage between the members of the community and sparse density outside the community
- Communities often occur in the WWW, telecommunication networks, academic networks, friendship networks, ….

How to define a community? What makes a community to be a community?

- Depends on visualization?
- Depends on how we define links and link-weights?
- Links to outside world: inter-group heterogeneity
- Links within community: intra-group homogeneity
- Overlapping communities allowed or not?

Community mining

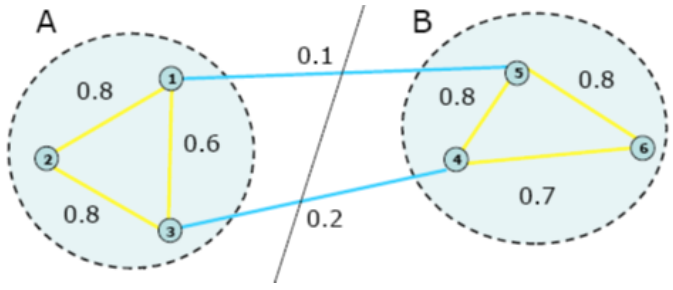
Two simple techniques (compare with hierarchical clustering)

1. Girvan-Newman algorithm

- The betweenness of all existing edges in the network is calculated first
- The edge with the highest betweenness is removed
- The betweenness of all edges affected by the removal is recalculated
- Steps 2 and 3 are repeated until no edges remain

2. Min-cut

- Find the minimal cut in the network and repeat



Note: both of these are computationally expensive!

- Most community mining techniques are
- Compare with divisive hierarchical clustering

Community mining

Many (other) methods available (e.g. spectral clustering based)

- Though most methods assume entire network structure is known!
- Detecting communities with overlap is hard
- When communities have been found, what do we learn from them? How do we gain insight in why these nodes form a community, according to the algorithm?

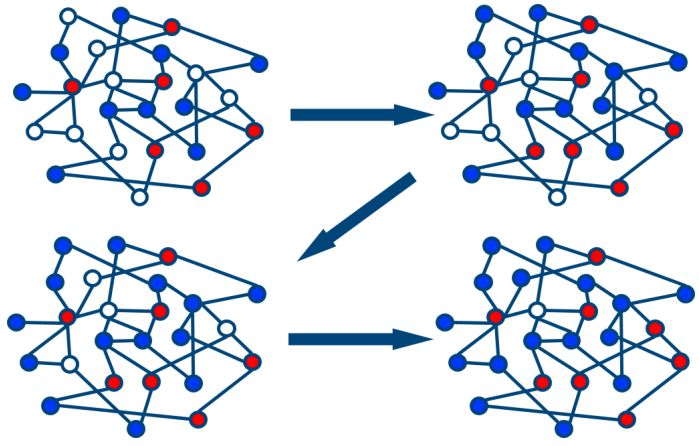
Practical applications of community mining?

- Targeted marketing
- Background information
- Viral marketing
- Or stopping viral effects: e.g. churn
- Segmentation analysis
- Use community labels as an additional feature

Note that a smart layout algorithm and visualization techniques can already help here (and be sufficient to spot patterns)

Making predictions

Making predictions



Common assumption: nodes will carry the class labels

Two types:

1. Network learning: use the network structure directly (e.g. community mining)
2. Featurization: extract features from the network, obtain a flat dataset, use normal analytics techniques (most common approach)

Network based inference

Goal: infer class membership/label of unknown nodes

- Fraud, churn, age, …

Not very easy: nodes can end up influencing each other

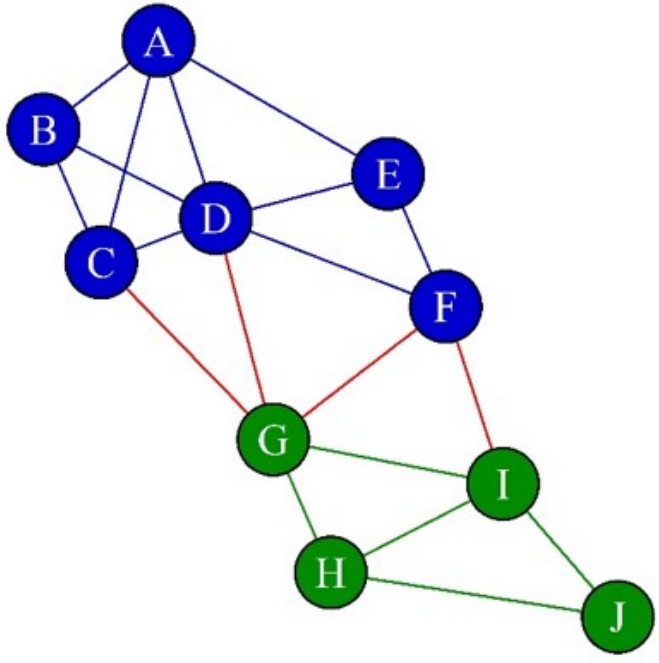
Most techniques assume that the Markov property holds: a node's outcome is only determined based on its first-order neighbors

- Makes construction of many techniques much easier

This is also commonly described based on the principle of "homophily" ("birds of a feather block together", or "guilt by association")

Is this a workable assumption?

Homophily



Assessed by looking at the distribution of edges in a social network relative to node properties

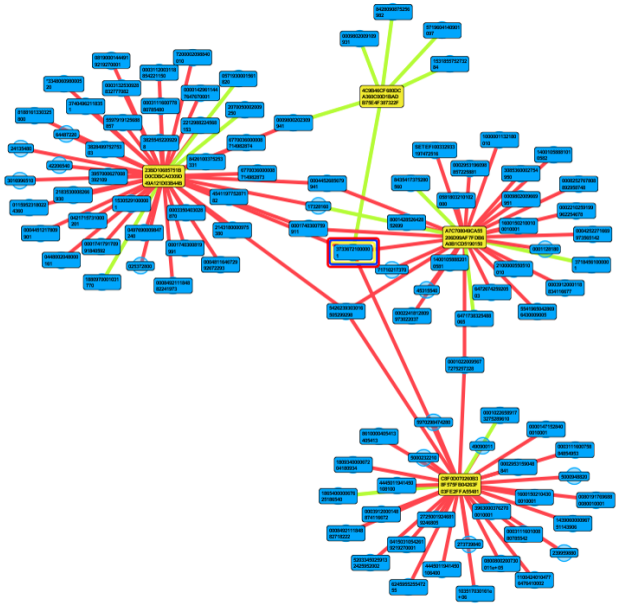
- In case of homophily edges among blue nodes and edges among green nodes are more common then edges between blue and green nodes
- In case of no homophily edges among blue nodes, among green nodes and between blue and green nodes are equally common – random configuration of edges

So what do we observe?

Homophily in fraud

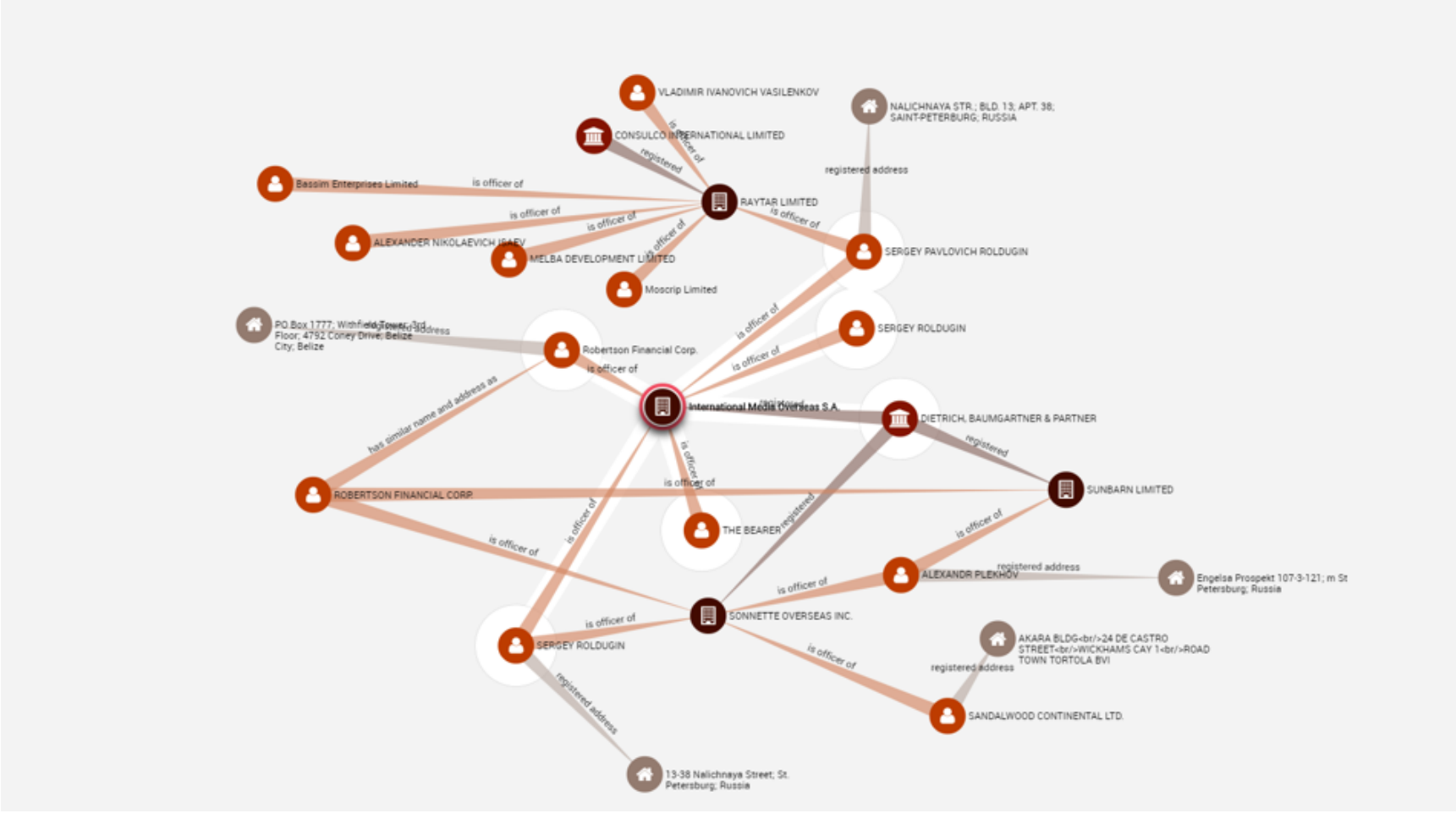
Fraudsters tend to cluster together

- Exchange knowledge how to commit fraud, use the same resources, are often related to the same event/activities, are sometimes one and the same person (identify theft)…
- Fraudsters are more likely to be connected to other fraudsters
- Fraudsters commit fraud in multiple instances (leading to more tight links)
- Legitimate people are more likely to be connected to other legitimate people



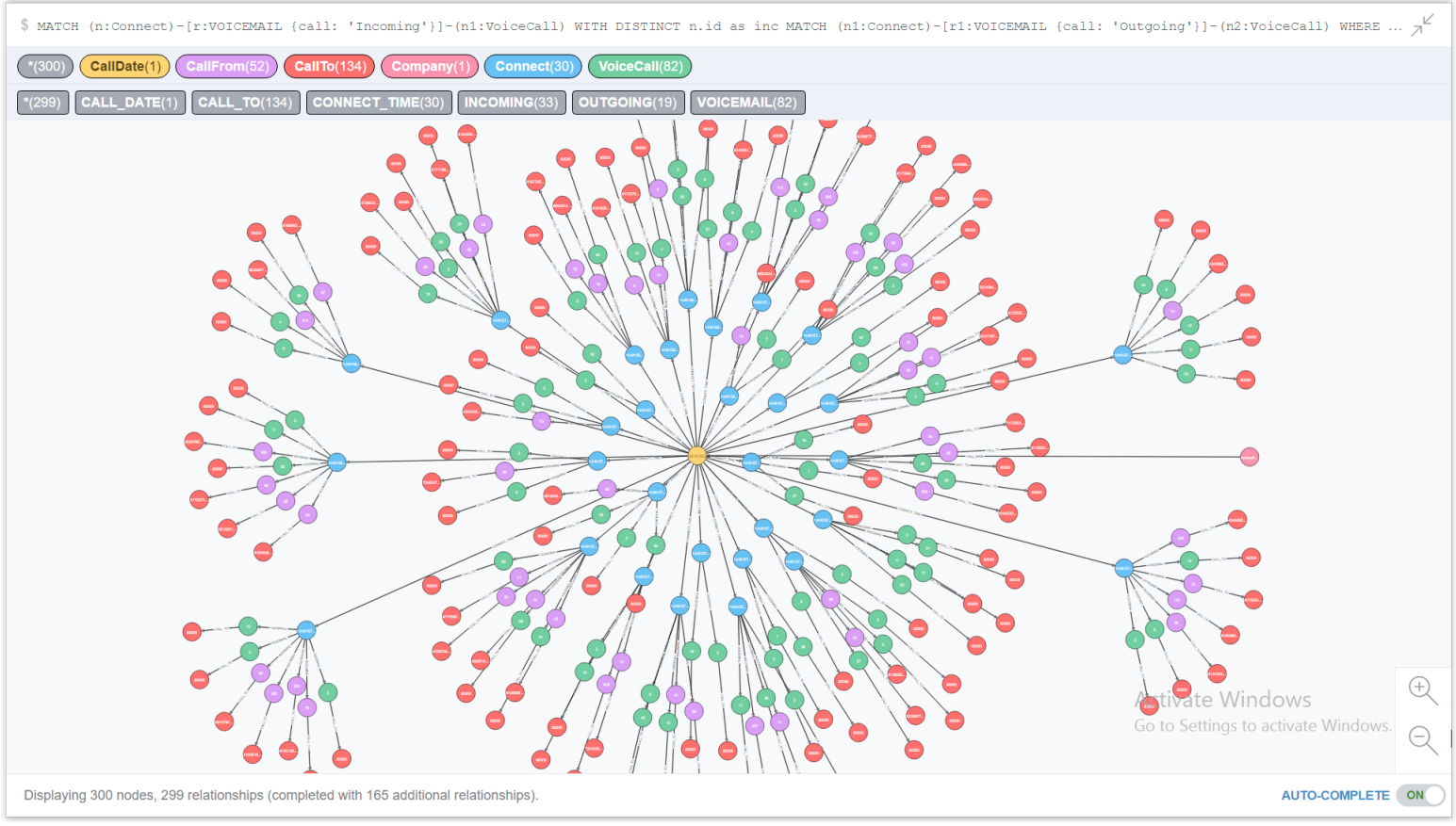
Stolen credit cards are used in the same store

Homophily in fraud



Homophily in churn

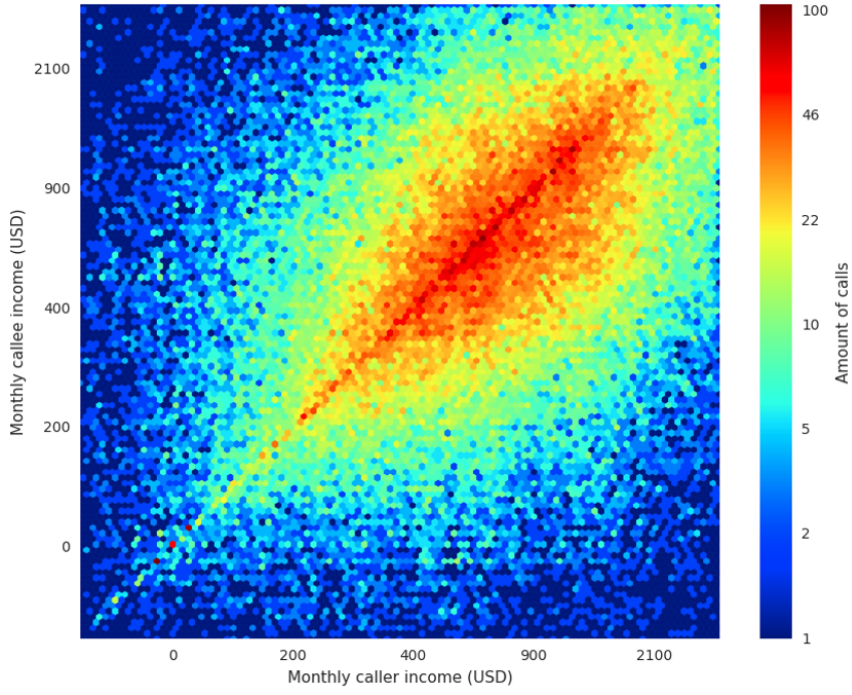
A customer who has a strong connection with a customer who recently churned is more likely to churn as well



Homophily in economy

People tend to call other people of the same economic status

Strong evidence of homophily between people with similar income levels



Fixman, Martin, et al. "A Bayesian approach to income inference in a communication network." Advances in Social Networks Analysis and Mining (ASONAM), 2016 IEEE/ACM International Conference on. IEEE, 2016.

Network based inference

Goal: infer class membership/label of unknown nodes

- Fraud, churn, age, …

Not very easy: nodes can end up influencing each other

Most techniques assume that the Markov property holds: a node's outcome is only determined based on its first-order neighbors

- Makes construction of many techniques much easier

This is also commonly described based on the principle of "homophily" ("birds of a feather block together", or "guilt by association")

Is this a workable assumption? Looks to be valid in many settings

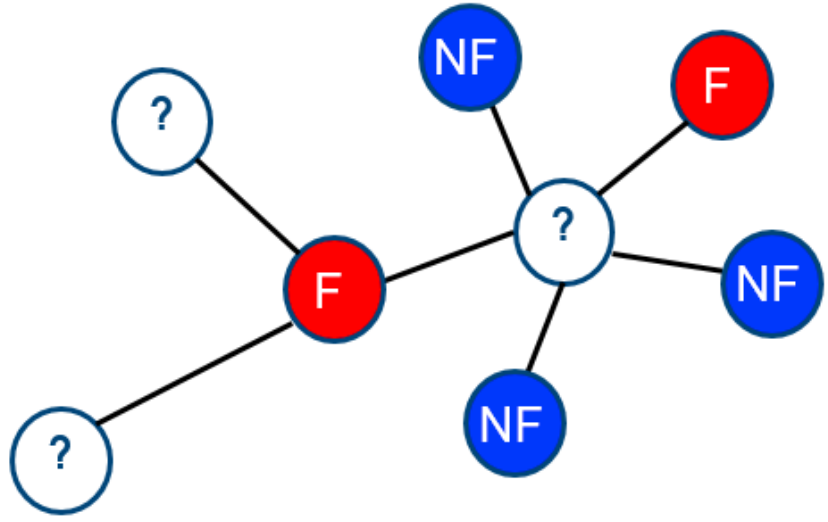
Network based inference

Goal: infer class membership/label of unknown nodes

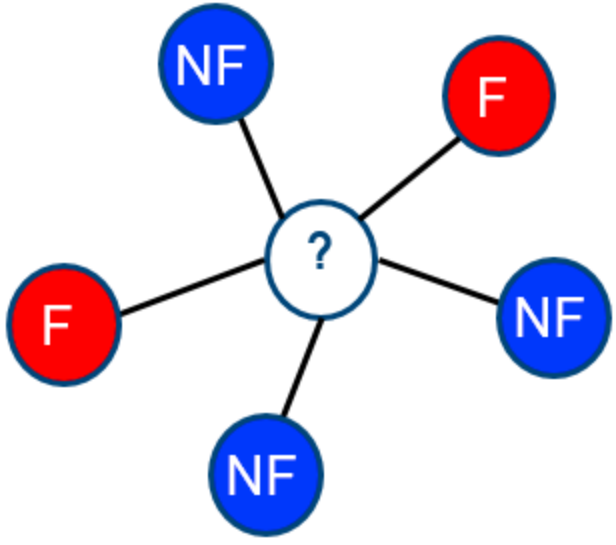
- Fraud, churn, age, …

1. Relational learners (Macskassy & Provost)
2. Diffusion/simulation/spreading/propagation activation approaches (Dasgupta et al., ...)

Relational learners



Relational learners: Relational Neighbor Classifier

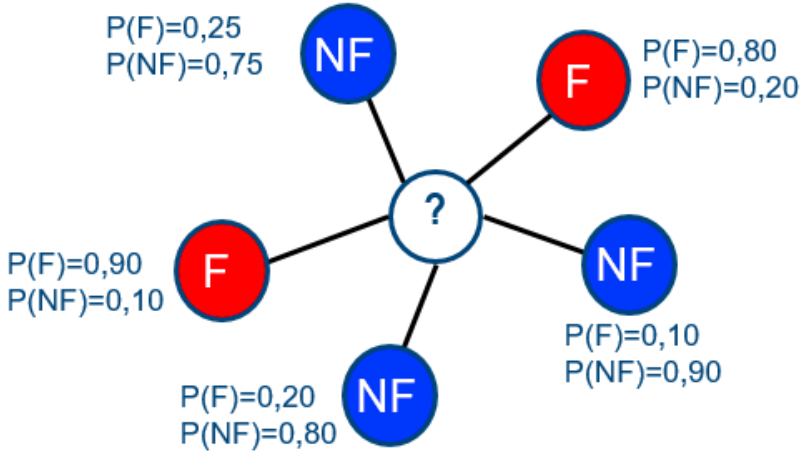


Z=5, so probabilities become:

- P(F|?) = 2/5
- P(NF|?) = 3/5

(Indeed, not that spectacular.)

Relational learners: Probabilistic Relational Neighbor Classifier



Z=5, so probabilities become:

- P(F|?) = 2,25/5 = 0,45
  (0,20 + 0,10 + 0,80 + 0,90 + 0,25)
- P(NF|?) = 2,75/5 = 0,55

(Indeed, not that spectacular.)

Propagation based techniques

Social network diffusion, behavior that cascade from node to node like an epidemic (Kleinberg 2007)

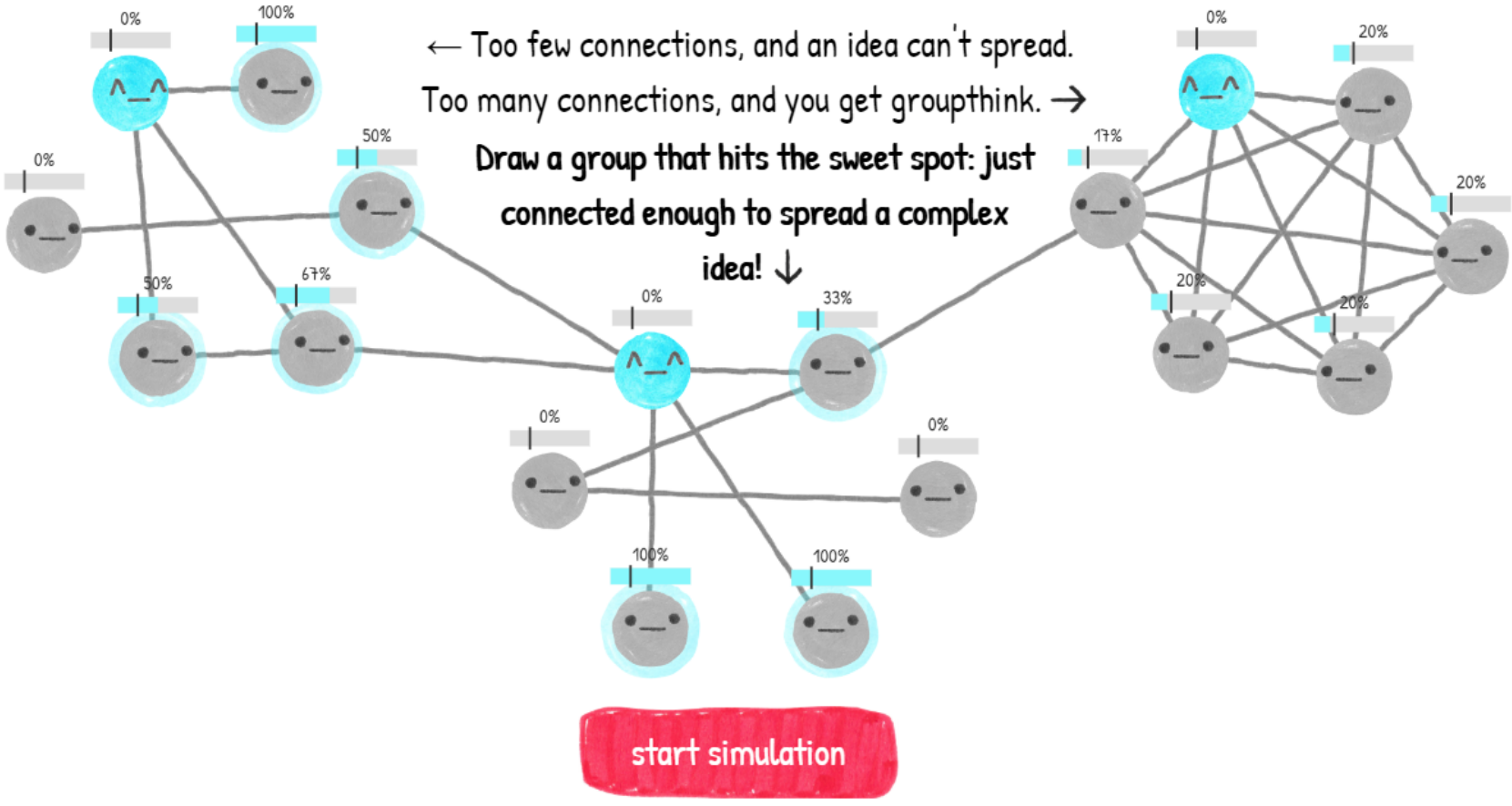
- News, opinions, rumors
- Public health
- Cascading failures in financial markets
- Viral marketing

A collective inference method infers a set of class labels/probabilities for the unknown nodes

- Gibbs sampling: Geman and Geman 1984
- Iterative classification: Lu and Getoor 2003
- Relaxation labeling: Chakrabarti et al. 1998
- Loopy belief propagation: Pearl 1988
- Personalized Pagerank

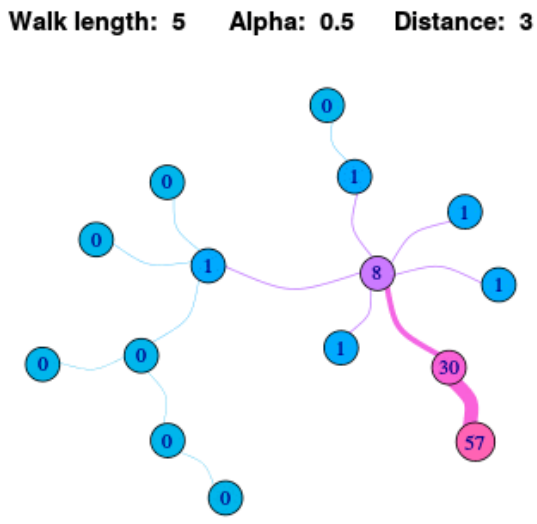
I.e. same goal as relational learners, but smarter approaches

"Madness of crowds"



https://ncase.me/crowds/

Personalized PageRank



Model how information spreads within a given graph

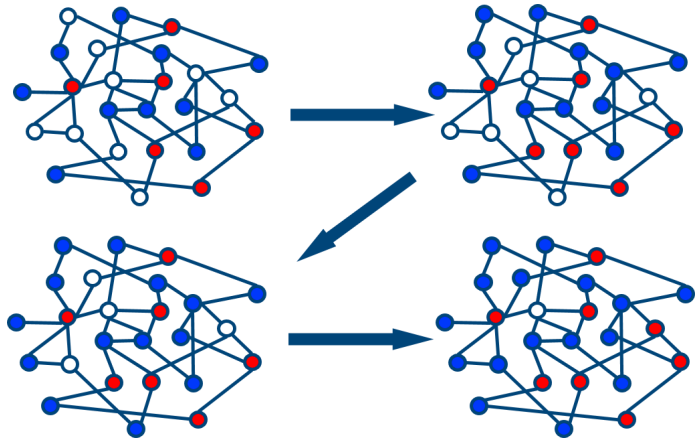
"Random walk" is one approach, but has the problem of back-and-forth effects

- "Lazy random walks" resolve this issue by allowing a chance for the walk to "rest/stay" at one of the vertices, in "normal" Page Rank, a random walk through the graph is performed, but can be interrupted with a small probability which sends the "walker" to a random node of the graph
- This random node that the walker jumps to is chosen with a uniform distribution

- But what if we would change this?
- In personalized Page Rank, the probability of the walker jumping to a node is not uniform, but determined by a certain distribution (i.e. the teleport probability, alpha), this is what we can use to influence the spread from a class of interest (Y=1)
- The resulting propagated scores can be used as predictions
- https://www.r-bloggers.com/from-random-walks-to-personalized-pagerank/

Featurization approaches

Making predictions



Common assumption: nodes will carry the class labels

Two types:

1. Network learning: use the network structure directly (e.g. community mining)
2. Featurization: extract features from the network, obtain a flat dataset, use normal analytics techniques (most common approach)

Wait a second...

Couldn't we also use the "predictions" of the relational learners as a feature to include?

Couldn't we also use propagated personalized PageRank scores as a feature?

Together with other centrality metrics, community labels?

Indeed...

Relational logistic regression (Lu and Getoor, 2003)

Combine local attributes

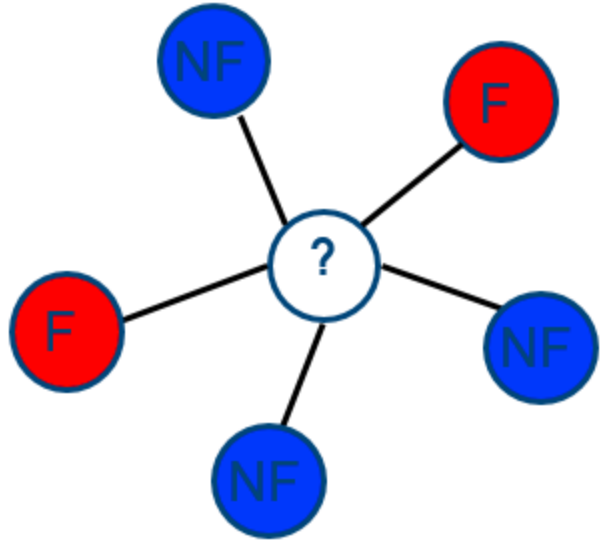
- For example, describing a customer's behavior (age, income, RFM, …)

With network attributes

- Most frequently occurring class of neighbor (mode-link)
- Frequency of the classes of the neighbors (count-link)
- Binary indicators indicating class presence (binary-link)

Combine local and network attributes in a single logistic regression model

Relational logistic regression (Lu and Getoor, 2003)



CID	Age	Income	…	Mode link	Frequency no fraud	Frequency fraud	Binary no fraud	Binary fraud
Seppe	33	1000		NF	3	2	1	1

Relational logistic regression (Lu and Getoor, 2003)

Though obviously, you could also include any other feature that might be helpful

- Social network metrics (see before)
- Probabilities resulting from the relational learners (see before)
- Other smart ideas

And obviously, you can use any classifier you want

Customer	Age	Avg duration	Avg revenue	Promo-tions	Avg age **friends**	Avg duration **friends**	Avg revenue **friends**	Promo-tions **friends**	Fraud
John	25	50	123	X	20	55	250	X	Yes
Sophie	35	65	55	Y	18	44	66	Y	No
Victor	50	12	85	None	50	33	50	X, Y	No
Laura	18	66	230	X	65	55	189	X	No

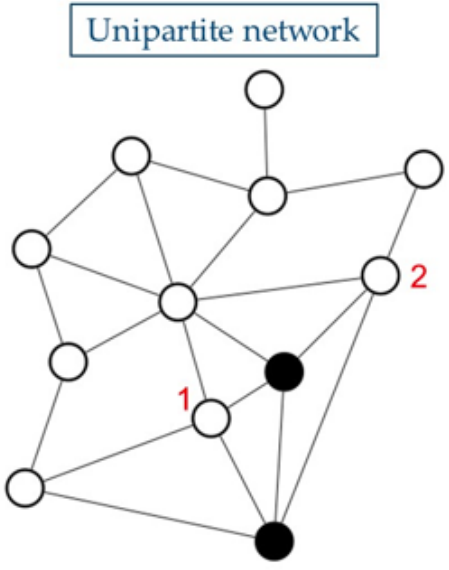
Featurization

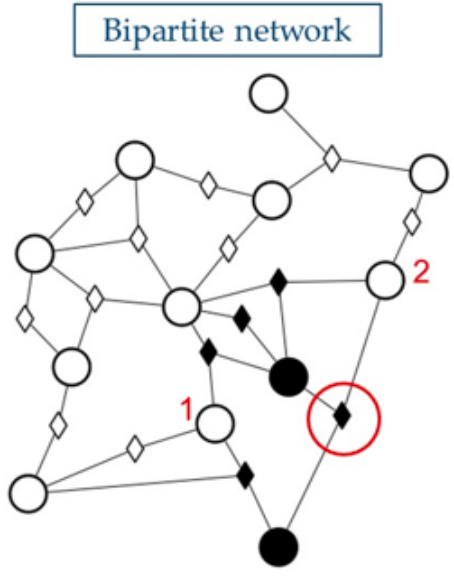
- Keep the network simple
- Nodes as label and attribute carriers
- Non-directional edges rather than directional, though additional relationships
- Domain-driven features on egonets
- Personalized PageRank as additional "global network" feature

Example

Context: fraud analytics in social security (fraudulent bankcruptcy) (Van Vlasselaer, Baesens et al., 2014)

Network construction: bipartite graph

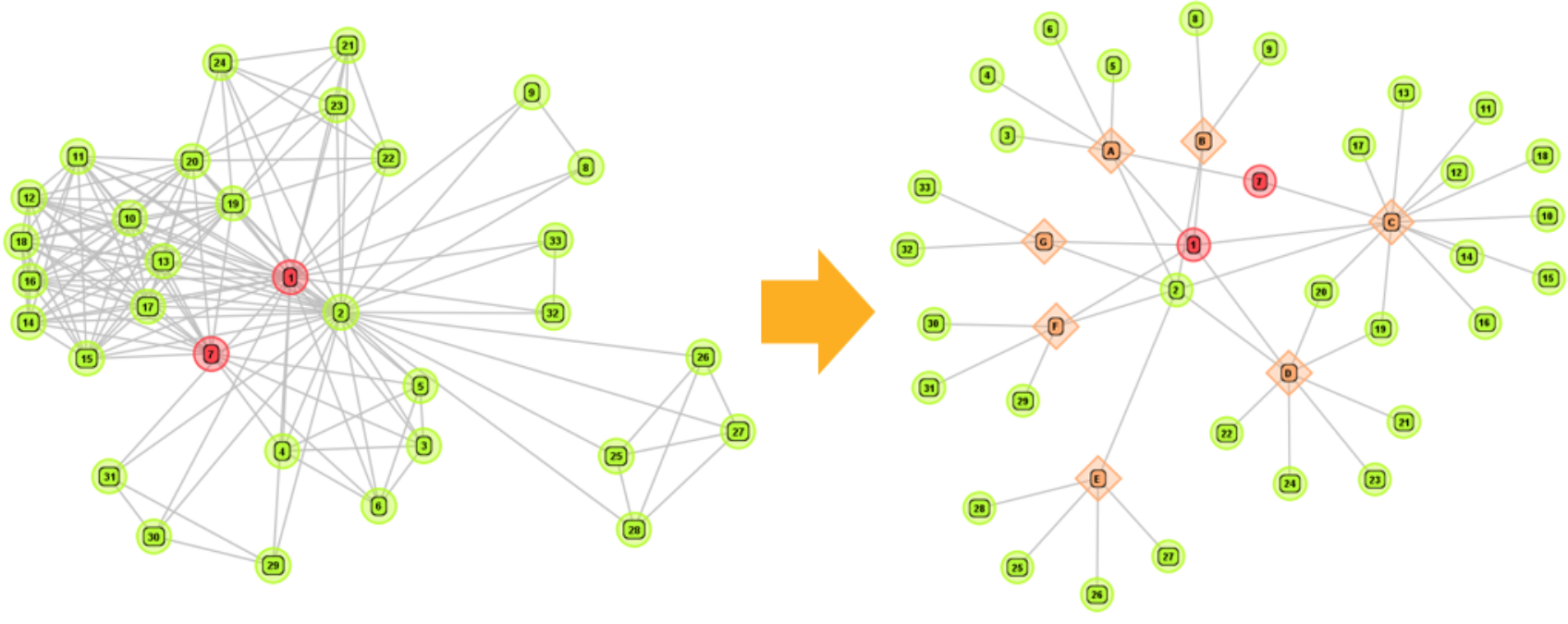




	Fraud Link	Non-Fraud Link
Employer 1	2	2
Employer 2	2	2

	0% Fraud	50% Fraud	100% Fraud
Employer 1	1	2	0
Employer 2	1	1	1

Example



Example

Nodes = {Companies, Resources}
Links = associated-to
Link Weight = recency of association

Local information and label for company-nodes

Featurization on company-egonets:

- Number of links to fraudulent resources
- Number of links to non-fraudulent resources
- Relative number of links to fraudulent resources
- …

Example

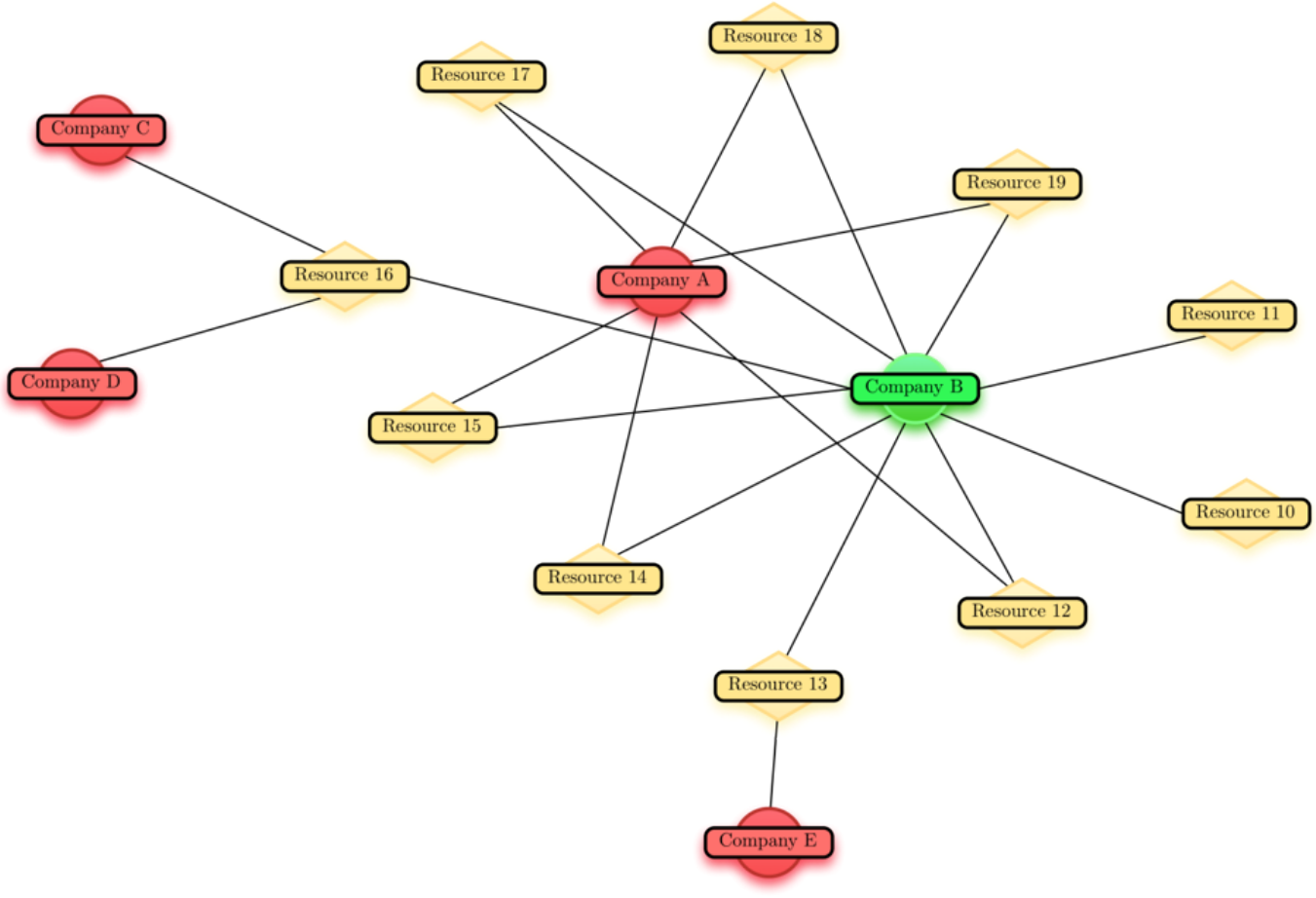
Additional featurization based on triangular associations

- Added in as pseudo edges

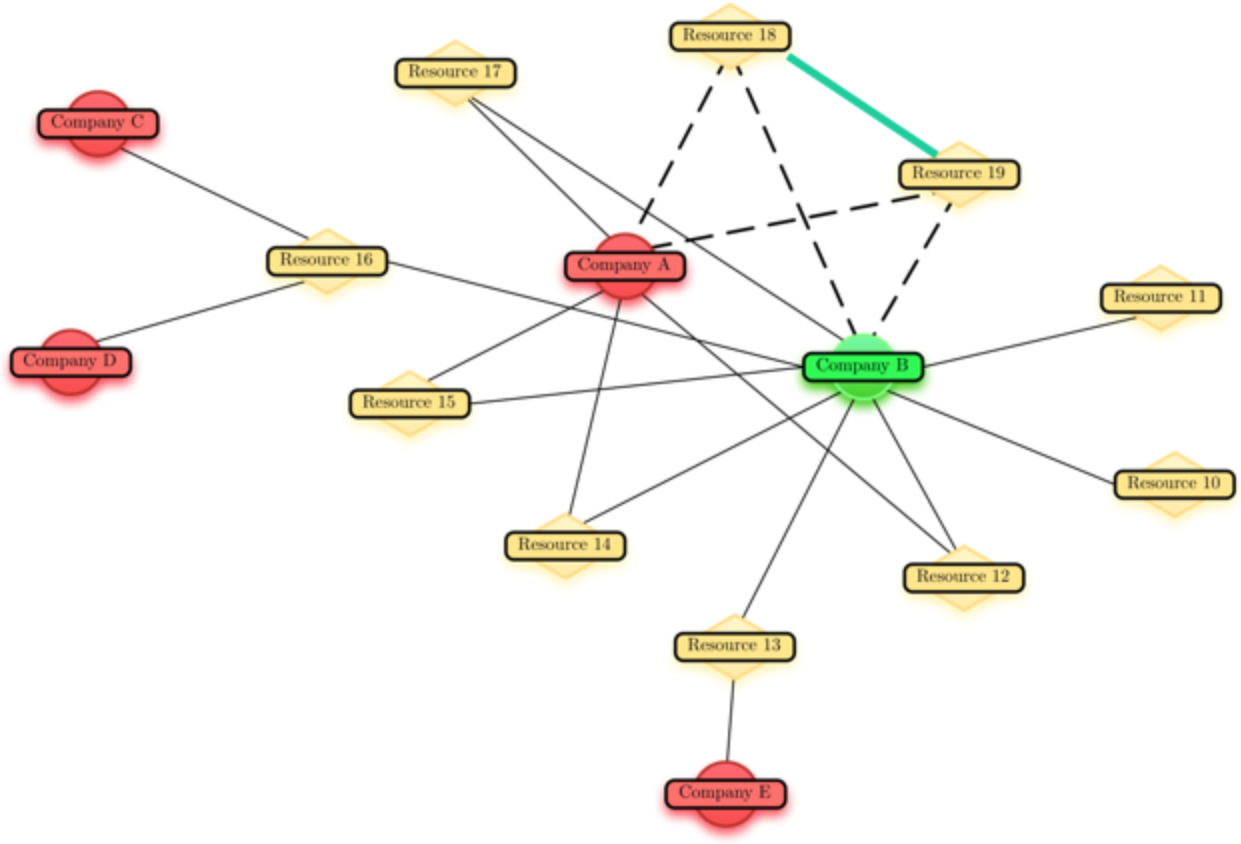
Another useful property of social-network graphs is the count of triangles (and other simple subgraphs)

- If a graph is a social network with n participants and m pairs of "friends," we would expect the number of triangles to be much greater than the value for a random graph. The reason is that if A and B are friends, and A is also a friend of C, there should be a much greater chance than average that B and C are also friends
- Thus, counting the number of triangles helps us to measure the extent to which a graph looks like a social network
- You can consider this as another social network metric type: count of the number of triangles involving a particular focus node

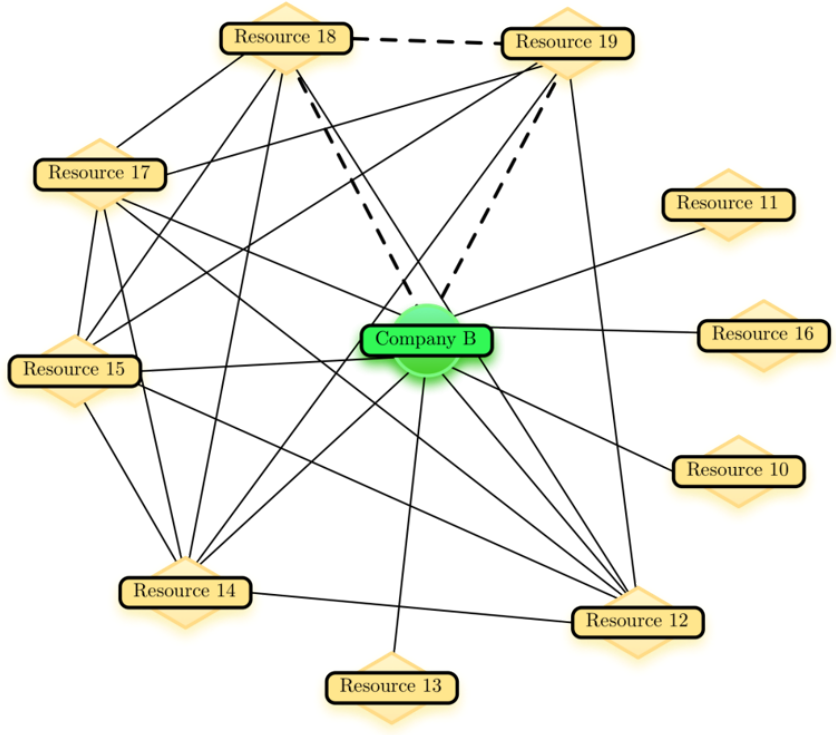
Example


Resource 18
Resource 17
Company C
Resource 19
Resource 16
Company A
Resource 11
Company D
Company B
Resource 15
Resource 10
Resource 14
Resource 12
Resource 13
Company E

Example


Resource 18
Resource 17
Company C
Resource 19
Resource 16
Company A
Resource 11
Company D
Company B
Resource 15
Resource 10
Resource 14
Resource 12
Resource 13
Company E

Example



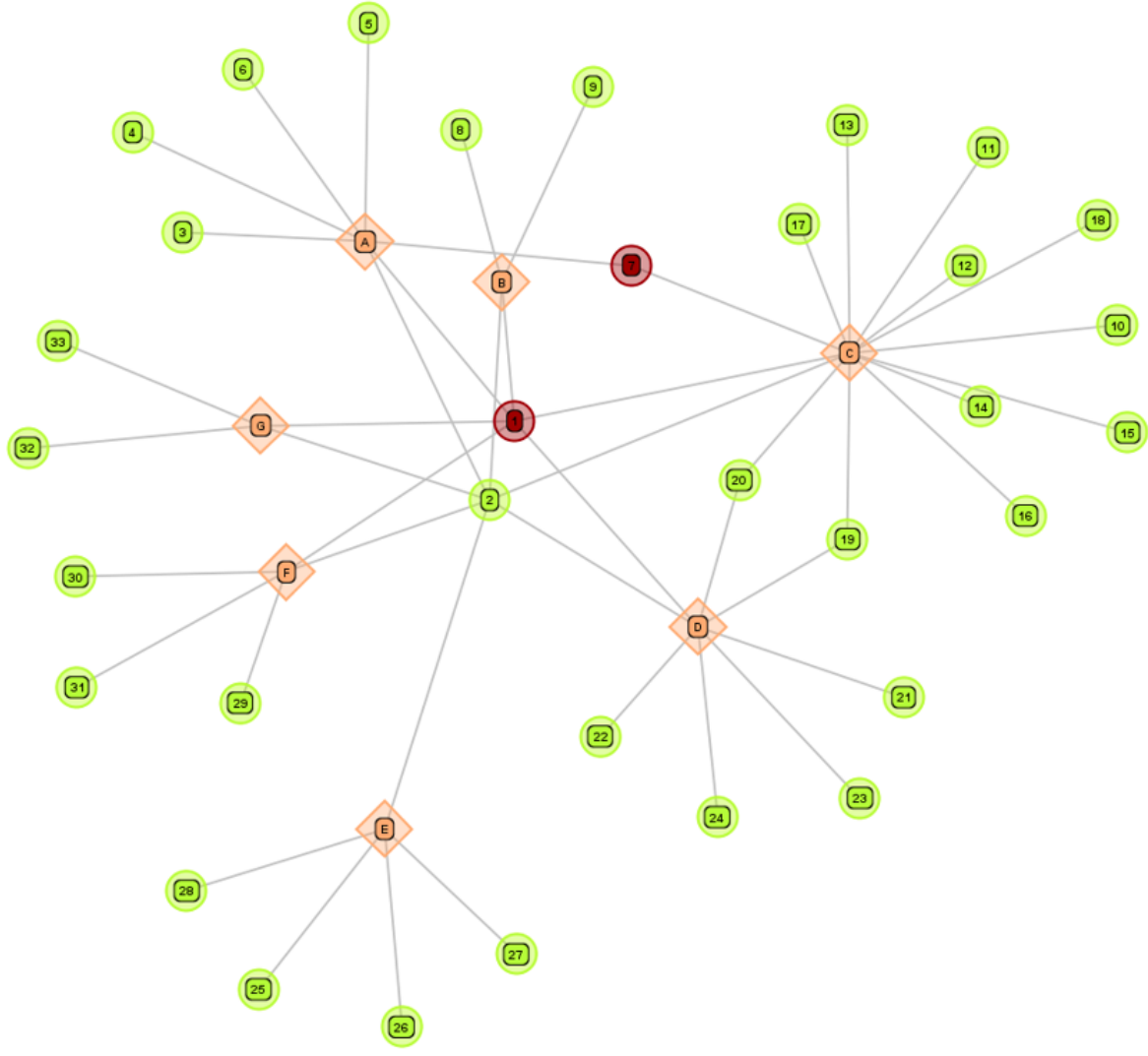
Count number of triangles involving focus node

Example

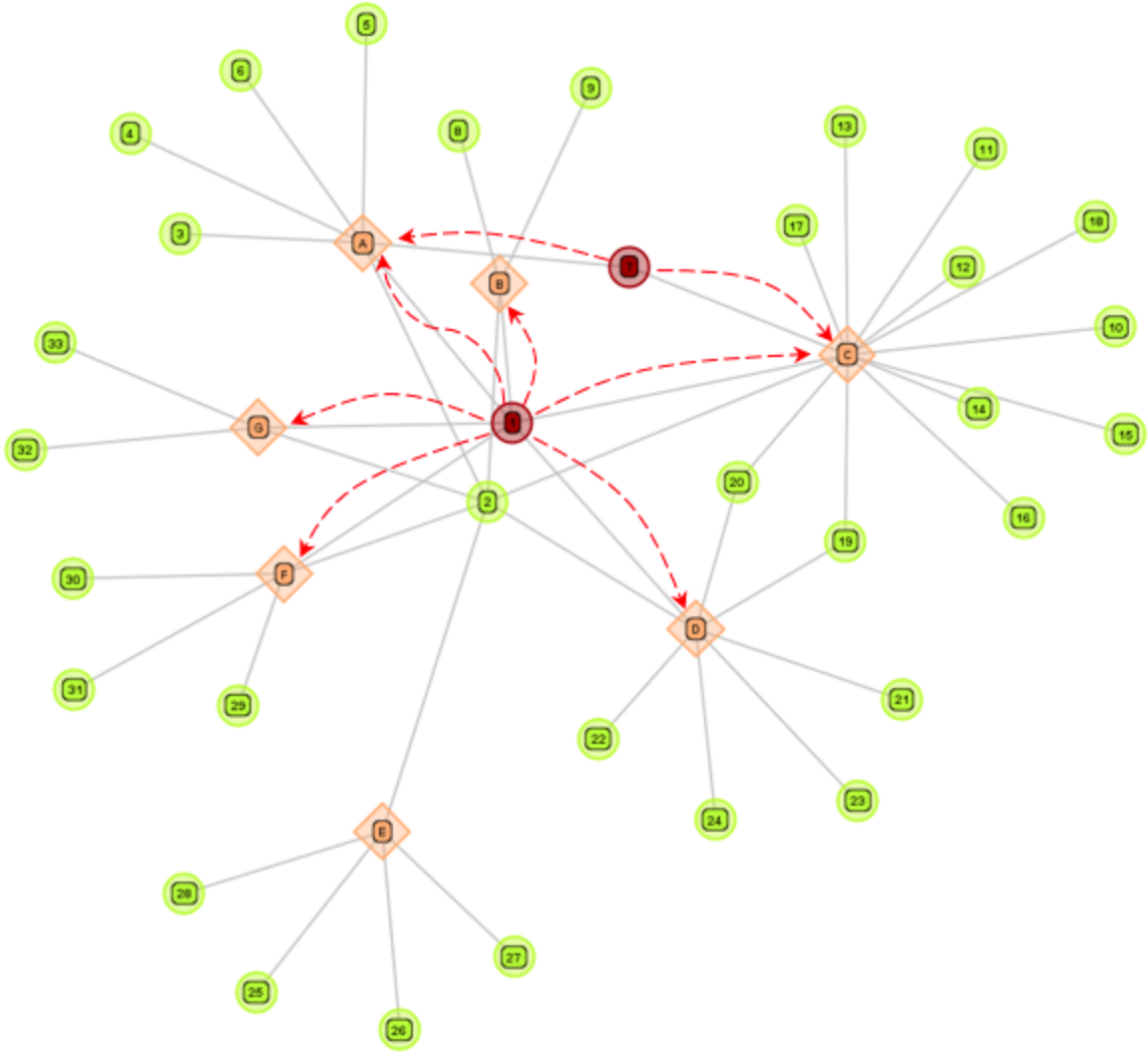
Featurization beyond the egonet

- Based on personalized PageRank
- Modified to work on bipartite graph and take edge recency into account

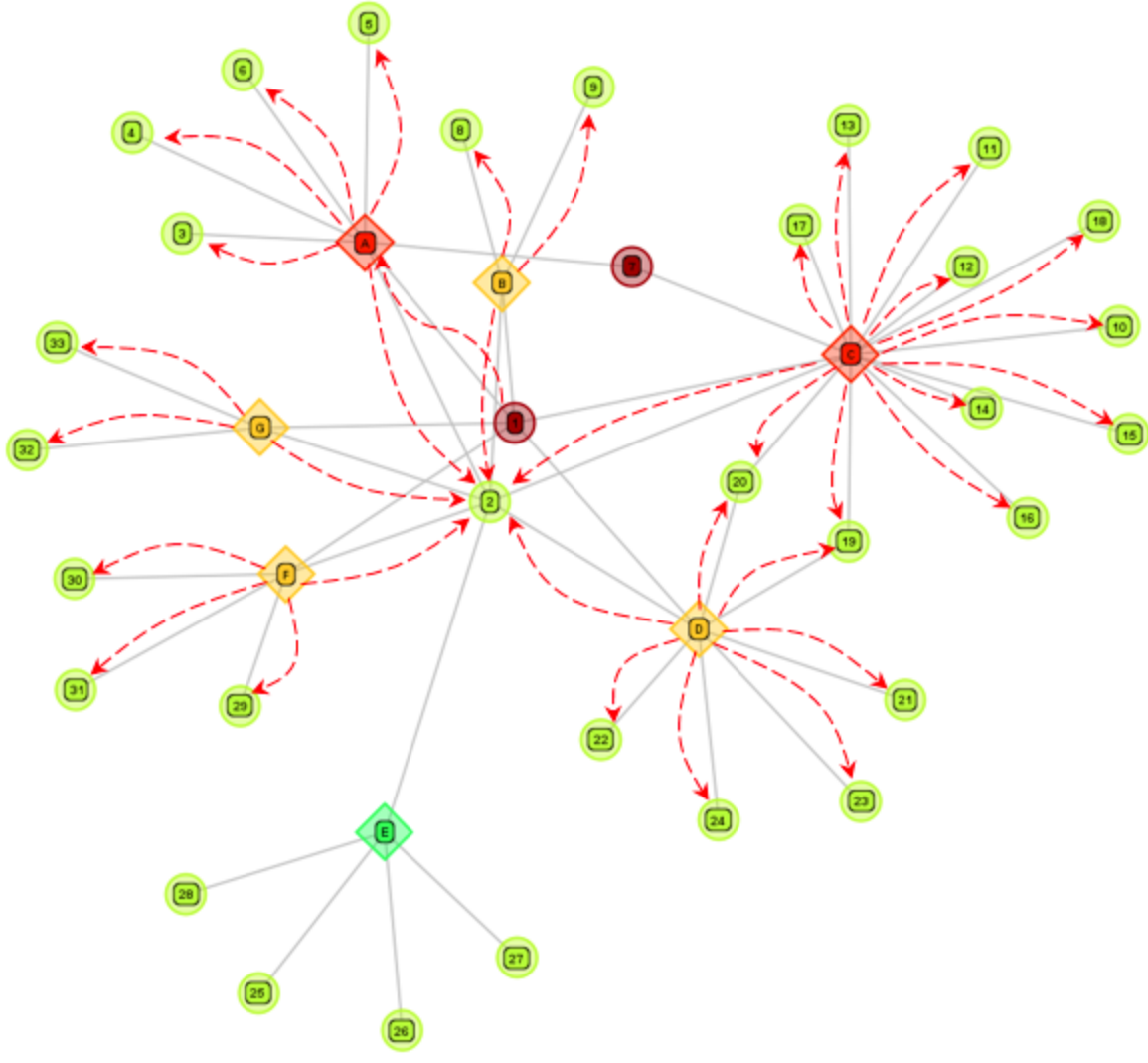
Example


5
6
9
4
8
13
11
3
A
17
18
B
12
10
33
C
14
15
G
32
20
2
16
19
30
F
D
31
29
21
22
23
24
E
28
27
25
26

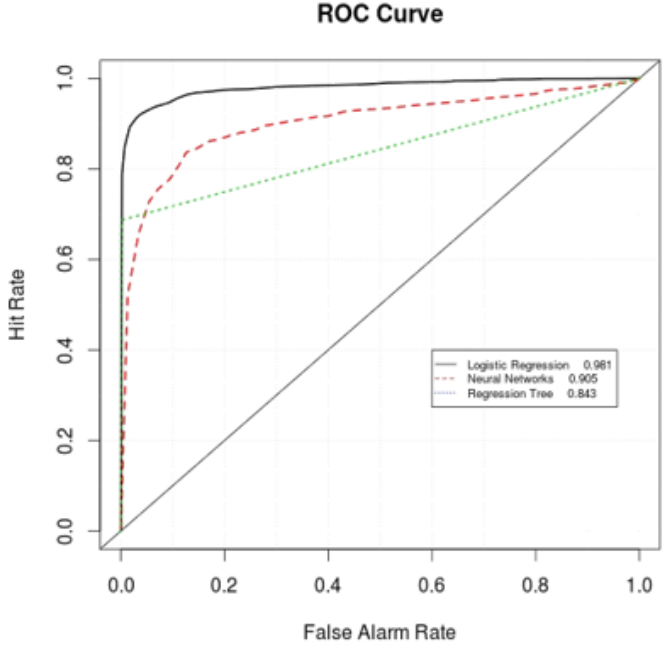
Example

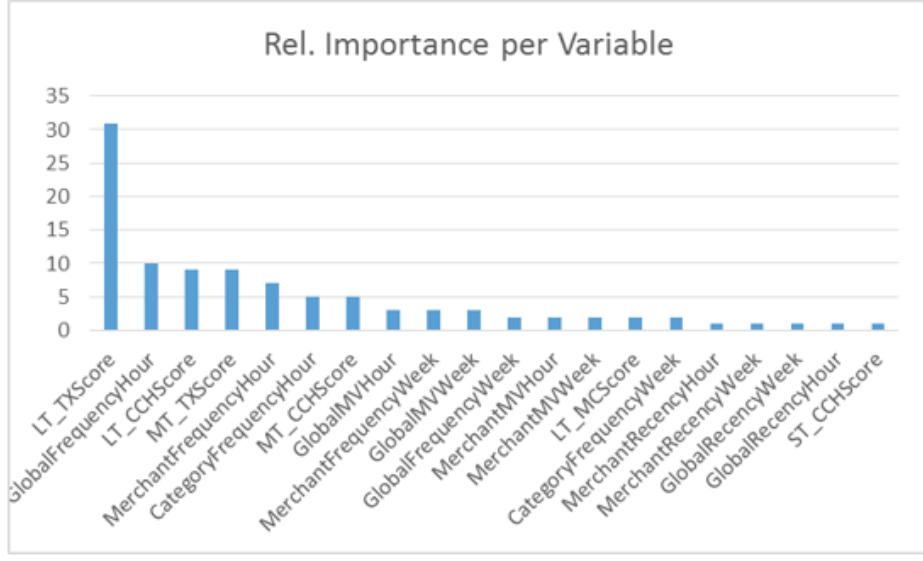

5
6
4
9
8
13
11
3
A
B
17
18
12
33
10
G
C
14
15
32
2
20
16
19
30
F
D
31
29
21
22
23
24
E
28
27
25
26

Example


5
6
4
3
A
8
9
B
13
11
17
18
12
10
C
33
G
32
1
14
15
20
2
16
19
30
F
D
31
29
21
22
23
24
E
28
27
25
26

Example

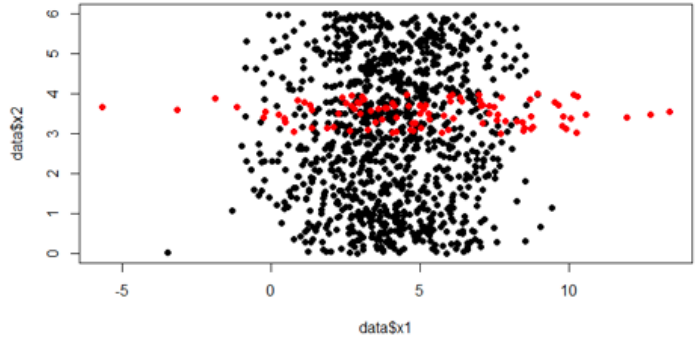

ROC Curve
Hit Rate
False Alarm Rate
0.0
0.2
0.4
0.6
0.8
1.0
Logistic Regression 0.981
Neural Networks 0.905
Regression Tree 0.843

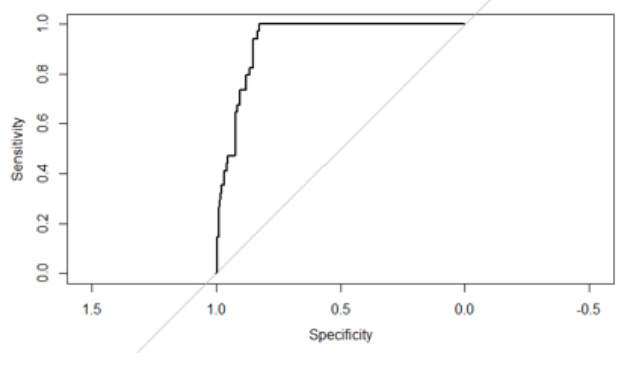

Rel. Importance per Variable
35
30
25
20
15
10
5
0
LT_TXScore
GlobalFrequencyHour
LT_CCHScore
MT_TXScore
MerchantFrequencyHour
CategoryFrequencyHour
MT_CCHScore
GlobalMVHour
MerchantFrequencyWeek
GlobalMVWeek
GlobalFrequencyWeek
MerchantMVHour
MerchantMVWeek
LT_MCScore
CategoryFrequencyWeek
MerchantRecencyHour
MerchantRecencyWeek
GlobalRecencyWeek
GlobalRecencyHour
ST_CCHScore

A word on validation

Let's take a look at another toy example

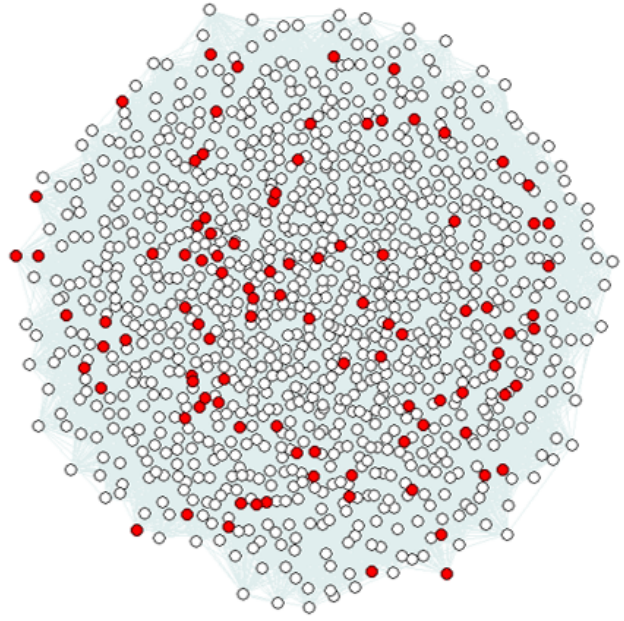
```
data <- data.frame(y=y, x1=x1, x2=x2)
tidx <- createDataPartition(data$y, p=0.33, list=F)
data.test <- data[tidx,]
data.train <- data[-tidx,]
data.train.bal <- ROSE(y ~ ., data=data.train)$data

plot(data$x1, data$x2, col=y+1, pch=16)

model.local <- randomForest(factor(y) ~ ., data=data.train.bal)

plot.roc(
  roc(data.test$y, predict(model.local, data.test, type='prob')[,'1']))
```





Let's take a look at another toy example

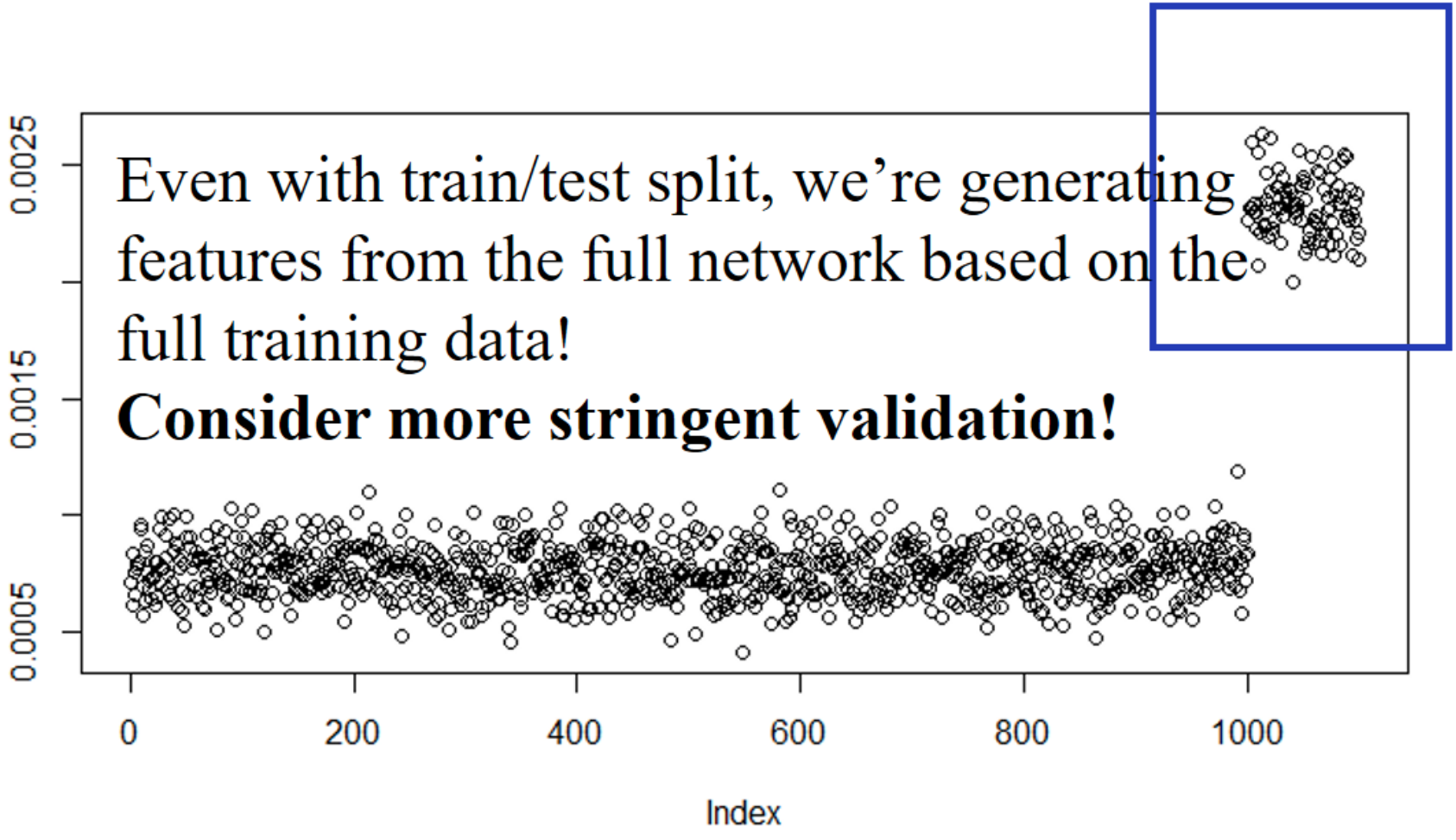
```
graph <- graph_from_data_frame(edges, directed=F)
V(graph)$color <- ifelse(data$y > 0, "red", "white")
E(graph)$color <- 'azure2'
plot(graph, layout=layout_with_lgl(graph), vertex.size=4, vertex.label='')
```



Let's take a look at another toy example

```
get_degree <- function(graph, id, positive_nodes) {
  av <- adjacent_vertices(graph, id, 'all')
  av <- av[[names(av)[1]]]
  ava <- length(av)
  avp <- sum(av %in% positive_nodes)
  data.frame(degree=ava, pos_degree=avp, neg_degree=ava - avp,
       pos_degree_frac=avp / ava,
       neg_degree_frac=1 - avp / ava)
}

network_vars <- as.data.frame(do.call(rbind,
  lapply(data$r, function(r) get_degree(graph, r, data[data$y == 1,'r']))
))

network_vars$page_rank <- page_rank(graph, personalized=data$y)$vector
network_vars$page_rank %>% plot
```

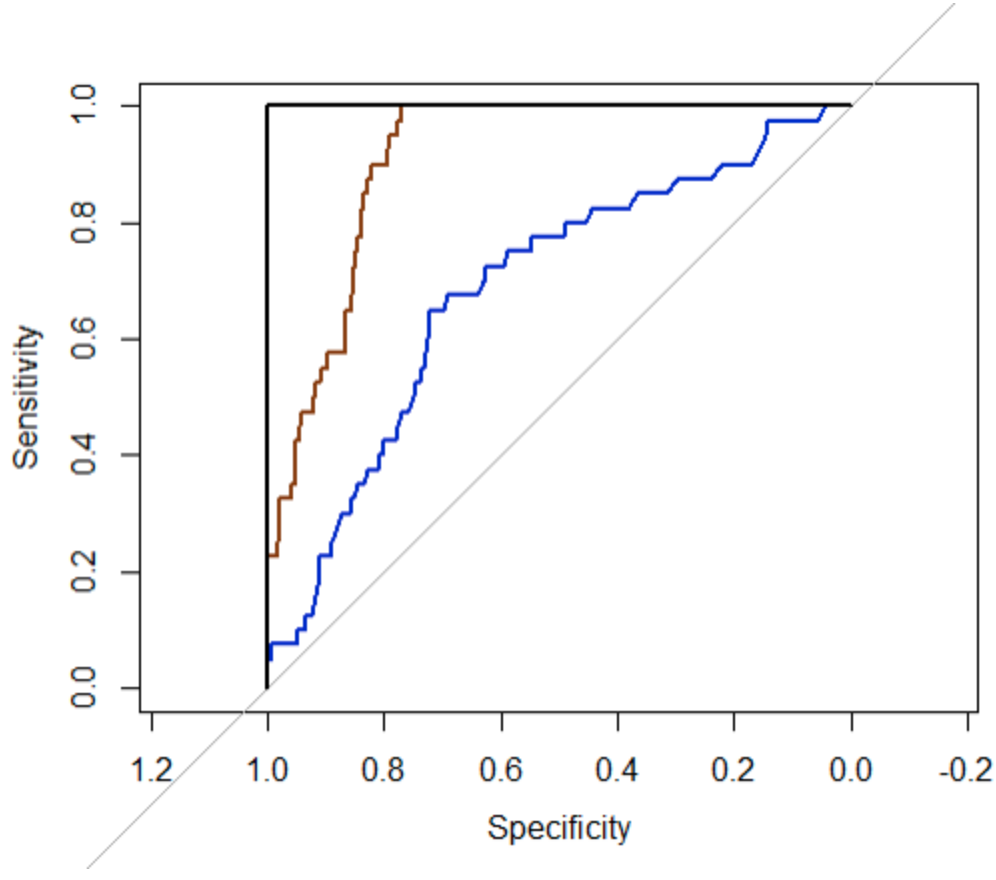
Let's take a look at another toy example



Let's take a look at another toy example

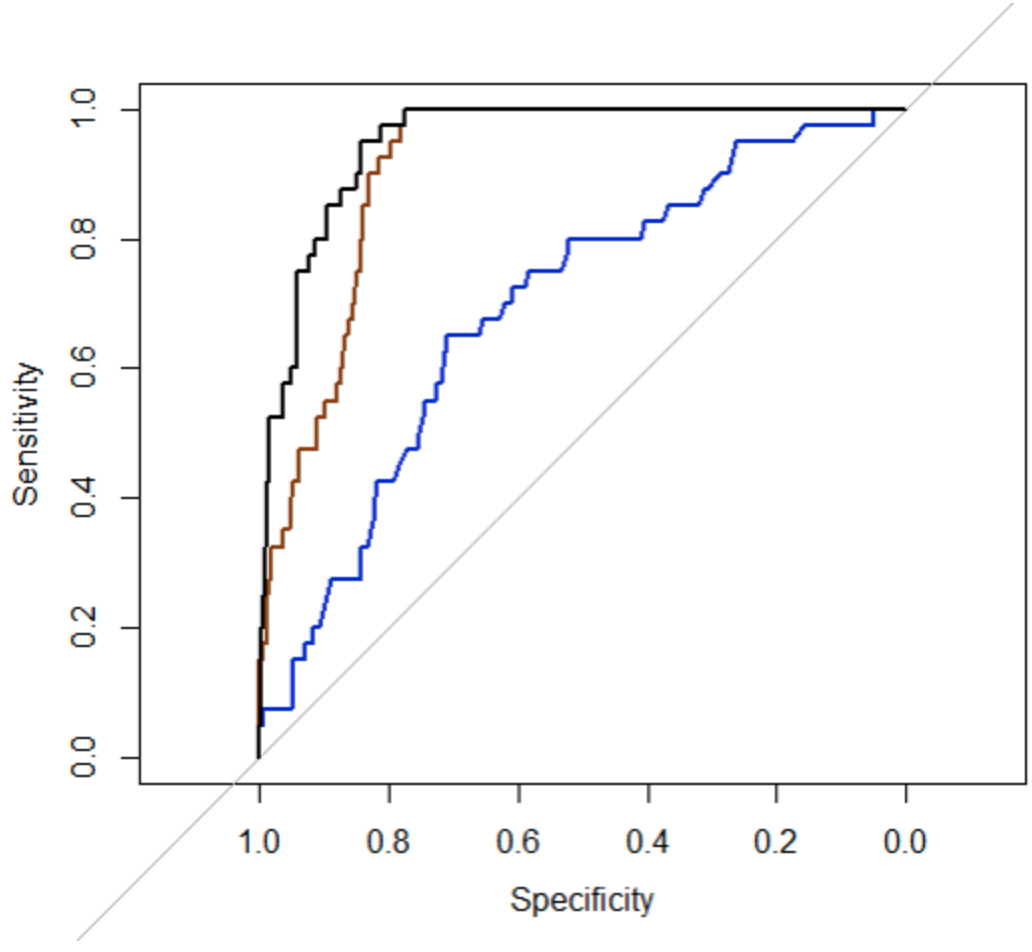
```
model.local <- randomForest(
  factor(y) ~ ., data=data.train.bal %>% select(y, x1, x2))
model.networked <- randomForest(
  factor(y) ~ ., data=data.train.bal %>% select(-x1, -x2, -page_rank))
model.networked_pr <- randomForest(
  factor(y) ~ ., data=data.train.bal %>% select(-x1, -x2))
model.all <- randomForest(
  factor(y) ~ ., data=data.train.bal)

plot.roc(roc(data.test$y,
  predict(model.local, data.test, type='prob')[, '1']),
  col='chocolate4')
plot.roc(roc(data.test$y,
  predict(model.networked, data.test, type='prob')[, '1']), add=T,
  col='blue3')
plot.roc(roc(data.test$y,
  predict(model.networked_pr, data.test, type='prob')[, '1']), add=T,
  col='blue4')
plot.roc(roc(data.test$y,
  predict(model.all, data.test, type='prob')[, '1']), add=T,
  col='black')
```

Let's take a look at another toy example



This is an issue...

Let's take a look at another toy example



(Without PageRank)

Validation is hard with networks

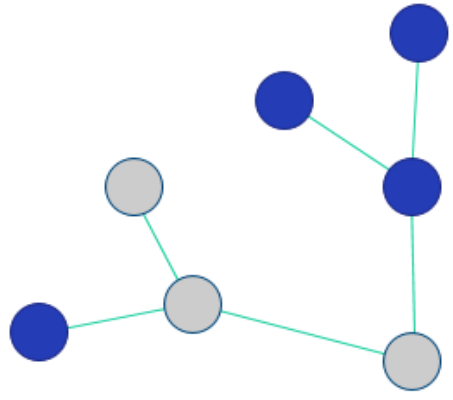
We've always stated earlier that all feature engineering should happen **after** the train/test split

Train on train, apply on test...

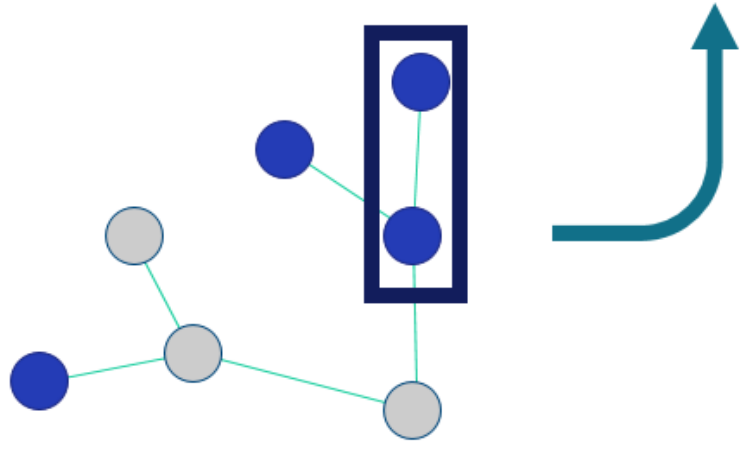
Even if we do this, we're still introducing data leakage as our network (and features we extract from it) are based on the whole data set!

Validation is hard with networks

Customer	Features	Churn
1	…	Yes
2	…	Yes
3	…	No
4	…	No



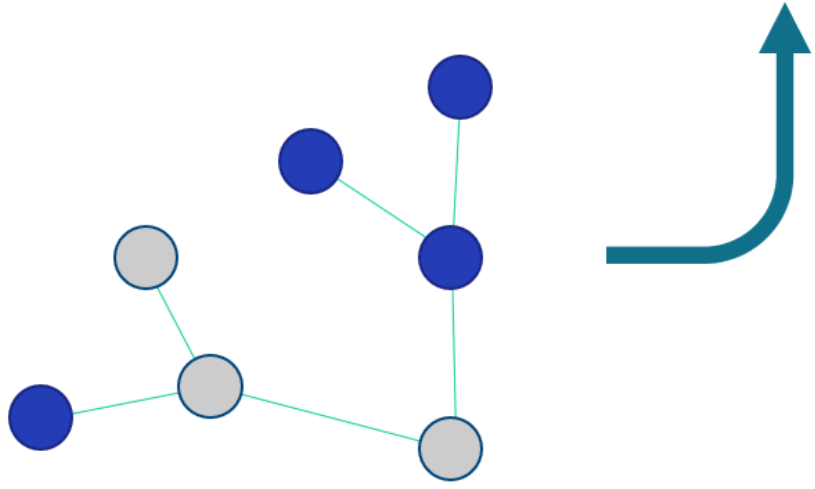
Customer	Features	Churn
1	…	Yes
2	…	Yes
3	…	No
4	…	No



Train / test split?
Feature generation will break

Validation is hard with networks

Customer	Features	Churn	Net. Feats,
1	…	Yes	…
2	…	Yes	…
3	…	No	…
4	…	No	…

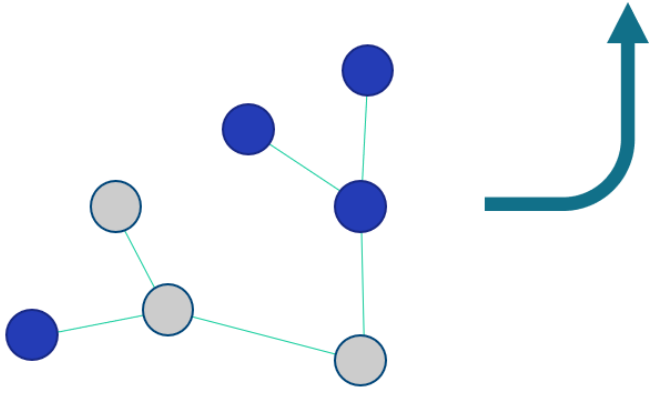


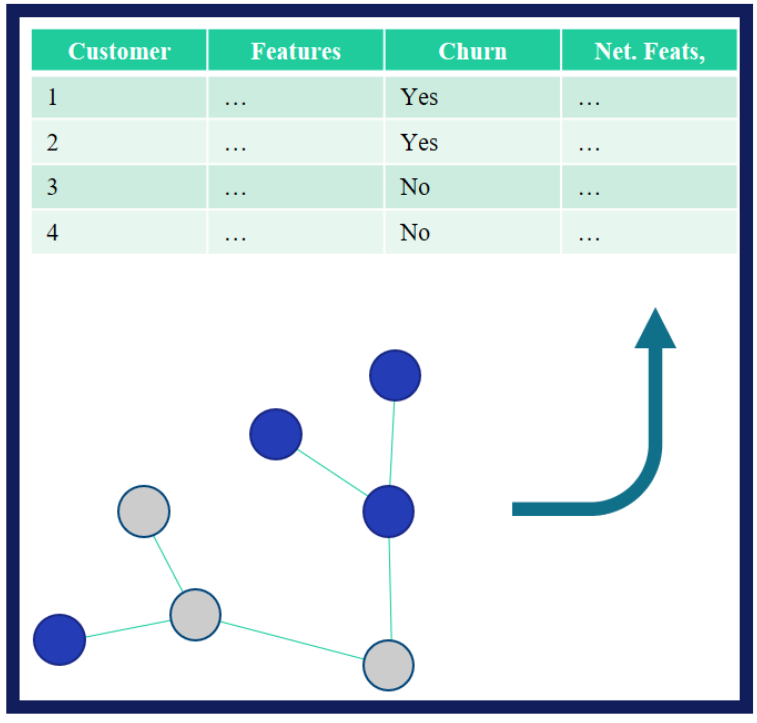
Option 1: retain the network for feature generation

Split on instance-level – **but can still lead to overfitting**

Validation is hard with networks

Customer	Features	Churn	Net. Feats,
1	…	Yes	…
2	…	Yes	…
3	…	No	…
4	…	No	…





Option 2: time-based split

Validation is hard with networks

Neither validation strategy is perfect: also with out-of-time testing, there is a large degree of overlap between network structure in train and test

- Make sure time difference is large enough, test at multiple points
- Even better: randomly censor positive labels in the network during feature generation
- For some features: less of an issue (e.g. Personalized PageRank uses the label information, other centrality metrics do not...)
- Hence also best to include domain features that do not assume knowledge of the label, but are based on features of neighbors only!

Same concerns in terms of applying the model

- At prediction-time: up-to-date state of the network needs to be known in order to featurize
- More stringent data-requirements!
- Historical state of the network

Privacy concerns: using your relationships to predict for you?

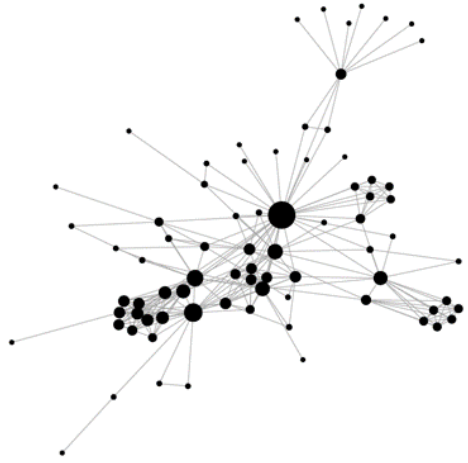
Node2Vec and friends

Node2Vec

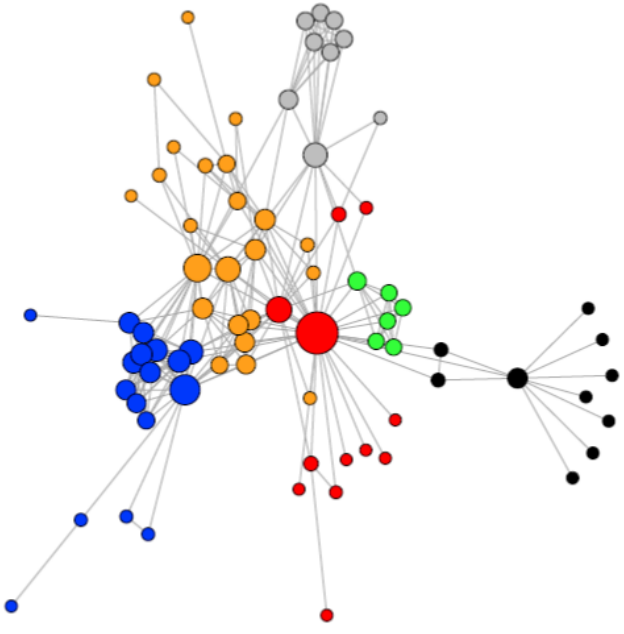
node2vec (Grover & Leskovec, 2016)

- Learn continuous features for the network using random walks and neural networks
- Basically: first perform a series of random walks to construct “sentences”
- Then apply normal word2vec

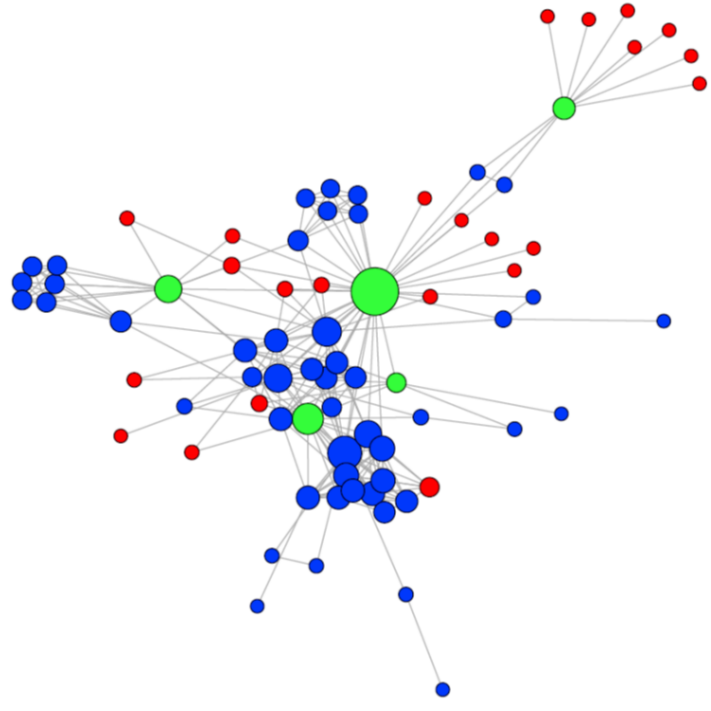
Les Misérables Network:



Node2Vec



Clustering the generated vectors for community detection



Clustering the generated vectors for structural detection

Node2Vec

Very versatile technique thanks to the ability to play with the random walks and how the "words" are generated.

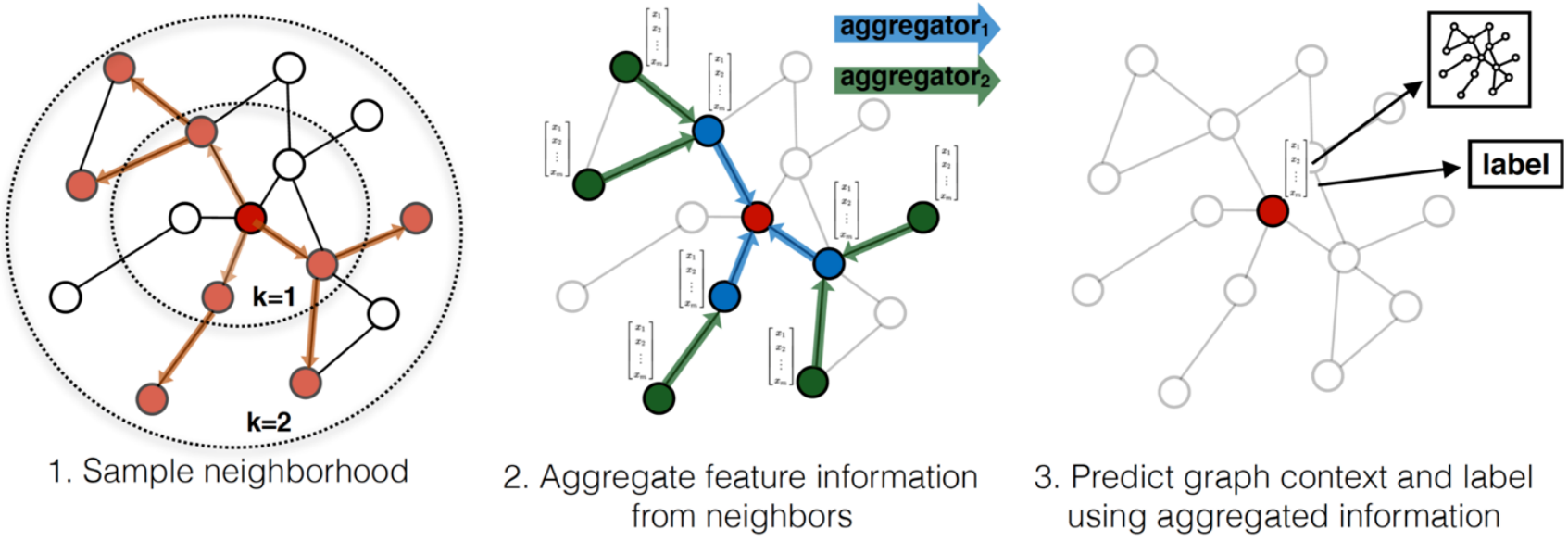
However: harder to utilize in a predictive setup, since network structure is assumed to be known

I.e. how to keep vector stability in case the network changes?

Friends

GraphSage: http://snap.stanford.edu/graphsage/

Deepwalk: https://arxiv.org/abs/1403.6652



1. Sample neighborhood

2. Aggregate feature information from neighbors

3. Predict graph context and label using aggregated information

Can be applied in an online training setting

Tooling

Tooling

Graph data management tools (graph databases): storage, querying

Graph wrangling and analytics tools: feature generation, social metrics, predictive modeling

Graph layout and visualization tools: Gephi and others

- This is what the majority of “network analytics” still refers to!

Tooling

- Python: NetworkX (https://networkx.github.io/)
- R: `igraph` (`ggraph`, `ggnet2`, `sna`, `network`, `tidygraph`) (https://igraph.org/r/)
  - https://www.jessesadler.com/post/network-analysis-with-r/
- Gephi (visualization and querying tool) or CytoScape, Graphviz, or JavaScript based tools
- Spark: GraphX
- Graph databases: Neo4j
- Data: http://snap.stanford.edu/index.html

Graph databases: Neo4j (includes support for algorithms in recent release: https://neo4j.com/blog/efficient-graph-algorithms-neo4j/)

GraphX

GraphX is a newer component in Spark for graphs and graph-parallel computation

- At a high level, GraphX extends the Spark RDD by introducing a new Graph abstraction: a directed multigraph with properties attached to each vertex and edge
- To support graph computation, GraphX exposes a set of fundamental operators (e.g., subgraph, joinVertices, and aggregateMessages)
- In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks

GraphX

GraphFrames is a package for Apache Spark which provides DataFrame-based Graphs

- https://github.com/graphframes/graphframes
- https://graphframes.github.io/graphframes/docs/_site/index.html
- It provides high-level APIs in Scala, Java, and Python. It aims to provide both the functionality of GraphX and extended functionality taking advantage of Spark DataFrames
- Still work-in-progress, however

> “ The GraphX component of Apache Spark has no DataFrames- or Dataset-based equivalent, so it is natural to ask this question. The current plan is to keep GraphFrames separate from core Apache Spark for the time being ”

GraphX

```
# Create a Vertex DataFrame with unique ID column "id"
v = sqlContext.createDataFrame([
  ("a", "Alice", 34), ("b", "Bob", 36), ("c", "Charlie", 30),
], ["id", "name", "age"])

# Create an Edge DataFrame with "src" and "dst" columns
e = sqlContext.createDataFrame([
  ("a", "b", "friend"), ("b", "c", "follow"), ("c", "b", "follow"),
], ["src", "dst", "relationship"])

# Create a GraphFrame
from graphframes import *
g = GraphFrame(v, e)

# Get in-degree of each vertex.
g.inDegrees.show()

# Count the number of "follow" connections in the graph.
g.edges.filter("relationship = 'follow'").count()

# Run PageRank algorithm, and show results.
results = g.pageRank(resetProbability=0.01, maxIter=20)
results.vertices.select("id", "pagerank").show()
```

NoSQL

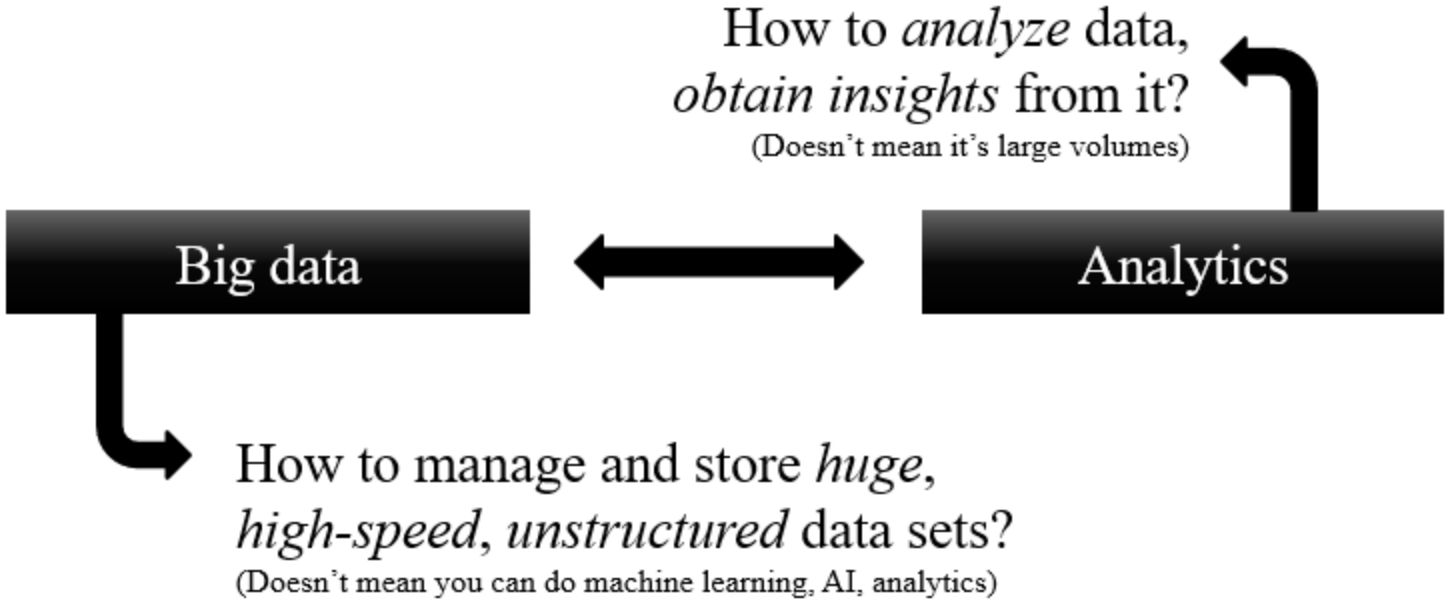
NoSQL

We'll also take a look at a Graph database in more depth: Neo4j

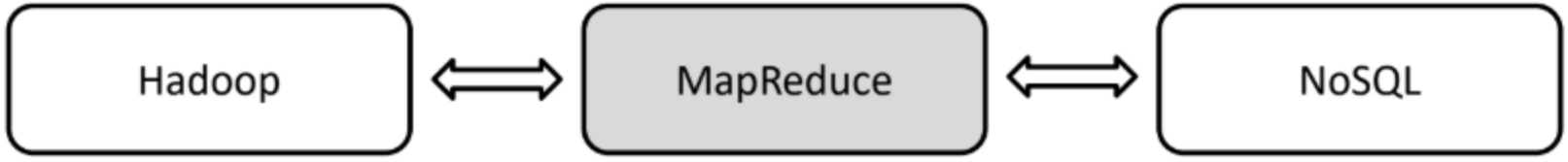
This is a NoSQL database, so we discuss what that means first...

(A discussion which brings us back to the big data landscape as well)

NoSQL


How to *analyze* data,
obtain insights from it?
(Doesn't mean it's large volumes)
Big data
Analytics
How to manage and store *huge*,
high-speed, *unstructured* data sets?
(Doesn't mean you can do machine learning, AI, analytics)

NoSQL



While the “Hadoop” world (good at large data volumes but not so much at querying) was busy trying to add in query possibilities on top of HDFS and MapReduce…

The database world (good at querying but not so much at scaling) was busy trying to make databases scalable…

Relational databases

A relational database management system (RDBMS) is a database management system based on the relational model

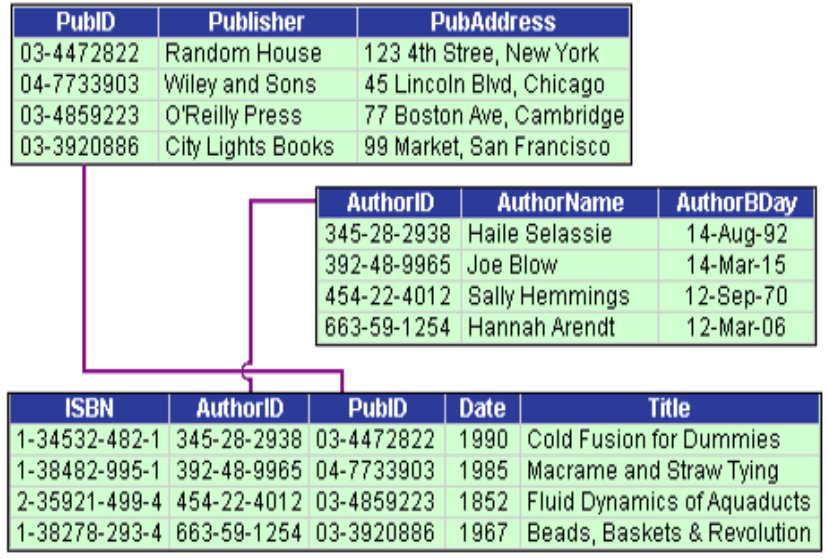
- Still today, many of the databases in widespread use are based on the relational database model
- RDBMSs have been a common choice for the storage of information in new databases used for financial records, manufacturing and logistical information, personnel data, and other applications
- Relational databases have received unsuccessful challenge attempts by object database management systems (in the 1980s and 1990s) and also by XML database management systems in the 1990s
- Despite such attempts, RDBMSs keep most of the market share

Examples: Oracle Database, Microsoft SQL Server, MySQL (Oracle Corporation), IBM DB2, IBM Informix, SAP Sybase Adaptive Server Enterprise, SAP Sybase IQ, Teradata, SQLite, MariaDB, PostgresQL

Relational databases

Structure data:

- Tables (storing records of information)
  - Identified by their primary key
- Linked together through relations (foreign keys)
  - One-to-many: every book record in the “Book” table refers to one record in the “Author” table (an author can hence be referred to by many books but each book would have at most one author)
  - Many-to-many: a book has multiple authors, and author has multiple books (needs an in-between cross table)



PubID	Publisher	PubAddress
03-4472822	Random House	123 4th Stree, New York
04-7733903	Wiley and Sons	45 Lincoln Blvd, Chicago
03-4859223	O'Reilly Press	77 Boston Ave, Cambridge
03-3920886	City Lights Books	99 Market, San Francisco

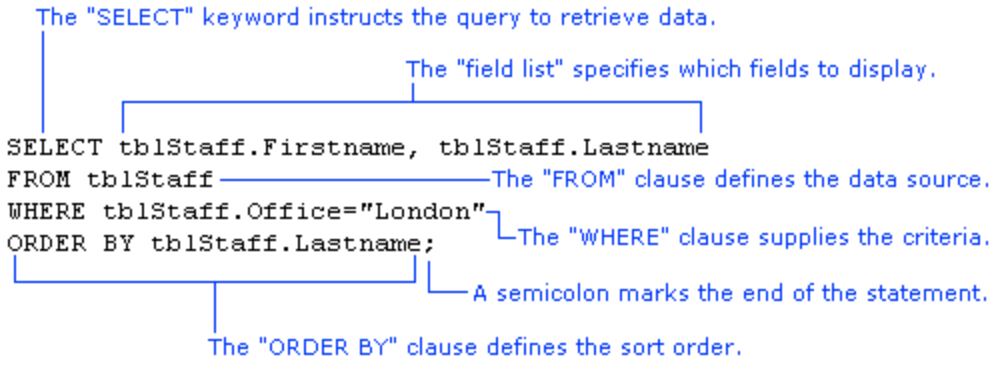
AuthorID	AuthorName	AuthorBDay
345-28-2938	Haile Selassie	14-Aug-92
392-48-9965	Joe Blow	14-Mar-15
454-22-4012	Sally Hemmings	12-Sep-70
663-59-1254	Hannah Arendt	12-Mar-06

ISBN	AuthorID	PubID	Date	Title
1-34532-482-1	345-28-2938	03-4472822	1990	Cold Fusion for Dummies
1-38482-995-1	392-48-9965	04-7733903	1985	Macrame and Straw Tying
2-35921-499-4	454-22-4012	03-4859223	1852	Fluid Dynamics of Aquaducts
1-38278-293-4	663-59-1254	03-3920886	1967	Beads, Baskets & Revolution

Relational databases

Data is queried using SQL

- Recall: the SQL in the whole big data story
- Hive, SparkSQL and so on…
- Powerful data wrangling language



Relational databases

RDBMSs are solid systems:

- Can handle large volumes of data
- Rich and fast query support
- And put a lot of emphasis on keeping data consistent

They require a formal database schema: a specification of all tables, relations, columns with their data type; quite a lot of modeling/design work

- New data or modifications to existing data are not accepted unless they comply with this schema in terms of data types, referential integrity etc.
- Moreover, the way in which they coordinate their transactions guarantees that the entire database is consistent at all times
- Of course, consistency is usually a desirable property; one normally wouldn't want for erroneous data to enter the system, nor for e.g. a money transfer to be aborted halfway, with only one of the two accounts updated

NoSQL enters the field

And then came Big Data

- Volume + Variety + Velocity
- Storage of massive amounts of (semi-)structured and unstructured, highly dynamic data
- Need for flexible storage structures (no fixed schema)
- Availability and performance often favored over consistency
- Complex query facilities not always needed: just put/get data
- Need for massive horizontal scalability (server clusters) with flexible reallocation of data to server nodes

Yahoo!... LiveJournal… MySpace… Google… Amazon… Facebook

- All this relational database overhead was slowing things down!
- Google and Yahoo! heavily invested in HDFS and Mapreduce (Hadoop) for large computational workloads
- Though very unstructured data model, extremely simple “query” facilities (e.g. see HBase)
- Some progress was necessary…

NoSQL enters the field

The term “NoSQL” has become incredibly overloaded throughout the past decade, so that the moniker now relates to a variety of meanings and systems

- The name “NoSQL” itself was first used in 1998 by the NoSQL Relational Database Management System, a DBMS built on top of input/output stream operations as provided by Unix systems. It actually implements a full relational database to all effects, but chooses to forego SQL as a query language
- But: this system has been around for a long time and has nothing to do with the more recent “NoSQL movement”. The home page of the NoSQL Relational Database Management System even explicitly mentions that it has nothing to do with the “NoSQL movement”
- The modern NoSQL movement describes databases that store and manipulate data in other formats than tabular relations, i.e. non-relational databases (movement should have more appropriately been called NoREL, especially since some of these non-relational databases actually do provide query language facilities which are close to SQL)
- Because of such reasons, people have started to change the original meaning of the NoSQL movement to stand for “not only SQL” or “not relational” instead of “not SQL”

NoSQL

What makes NoSQL databases different from other, legacy, non-relational systems which have existed since as early as the 1970s?

- The renewed interest in non-relational database systems stems from the advent of Web 2.0 companies in the early 2000s. Companies such as Facebook, Google, and Amazon were increasingly being confronted with huge amounts of data that needed to be processed, oftentimes under time-sensitive constraints
- Often rooted in the open source community, the characteristics of the systems that were developed to deal with these requirements are very diverse
- Many of them aim at **near linear horizontal scalability, which is achieved by distributing data over a cluster of database nodes for the sake of performance** (parallelism and load balancing) as well as availability (replication and failover management). A certain measure of data consistency is often sacrificed in return
- A term frequently used in this respect is **eventual consistency; the data, and respective replicas of the same data item, will become consistent at some point in time after each transaction, but continuous consistency is not guaranteed**
- The relational data model is cast aside in favor of other modelling paradigms, which are typically less rigid and better able to cope with quickly evolving data structures. Note that **different categories of NoSQL databases exist** and that even the members of a single category can be very diverse

NoSQL

	Relational Databases	NoSQL Databases
Data paradigm	Relational tables	Key-value (tuple) based Document based Column based Graph based XML, object based (see previous chapters) Others: time series, probabilistic, etc.
Distribution	Single-node and distributed	Mainly distributed
Scalability	Vertical scaling, harder to scale horizontally	Easy to scale horizontally, easy data replication
Openness	Closed and open source	Mainly open source
Schema role	Schema-driven	Mainly schema-free or flexible schema
Query language	SQL as query language	No or simple querying facilities, or special-purpose languages
Transaction mechanism	ACID: Atomicity, Consistency, Isolation, Durability	BASE: Basically available, Soft state, Eventual consistency
Feature set	Many features (triggers, views, stored procedures, etc.)	Simple API
Data volume	Capable of handling normal-sized data sets	Capable of handling huge amounts of data and/or very high frequencies of read/write requests

Most often used as the method to "classify" NoSQL databases

Key-value stores

Concept: storage of `(key, any data value)` couples

- The unique keys are the only criterion for data retrieval
- Store and retrieve across multiple nodes by hashing the key (‘consistent hashing’)
- Data values = BLOBs, no meaning, no search criteria
- No schema; data interrelations managed at application level
- Mainly useful for simple put/get functionality, based on (part of) key
- Scalability and performance
- Often foundation layer to more complex systems

Examples: Memcached, Redis, Membase, Dynamo (Amazon), Bigtable (Google), HBase

- Clearly a link with MapReduce...

Document stores

Concept: storage of `(key, document)` couples

- The DBMS is aware of the document type and interprets the document content
- Document formats: semi-structured data, a.o. XML, JSON (JavaScript Object Notation), YAML (YAML Ain't Markup Language), …
- Documents contain attributes
- Therefore, we also speak of tuple stores, i.e. the document is a vector of data
- Document processing (add/change attributes); attributes as search criteria
- Complex data structures and nested objects; no fixed schema

Examples: MongoDB, CouchDB, Cassandra

```
_id=345 -> {
   Title      = "Harry Potter"
   ISBN       = "111-1111111111"
   Authors    = [ "J.K.  Rowling" ]
   Price      = 32
   Dimensions = "8.5 x 11.0 x 0.5"
   PageCount  = 234
   Genre      = "Fantasy"
}
```

Querying

Document stores deal with semi-structured items, meaning that they do not impose a particular schema on the structure of items stored in a particular collection, but that items nevertheless exhibit an implicit structure following from their representational format, representing a collection of attributes, using e.g. JSON or XML

Just as with key-value stores, the primary key of each item can be used to rapidly retrieve a particular item from a collection

Since items are composed of multiple attributes, most document stores allow to specify more complicated queries as well… but again often using map-reduce (at least for a very long time)

Querying


So, how do I query the database?
It's not a database. It's a key-value store!
Ok, it's not a database. How do I query it?
You write a distributed map reduce function in Erlang!
Did you just tell me to go <beep> myself?
I believe I did, Bob.
Image credit: http://browsertoolkit.com/fault-tolerance.png

Querying

```
public static void reportAggregate(MongoDatabase db) {
  String map = "function() { " +
        "  var nrPages = this.nrPages; " +
        "  this.genres.forEach(function(genre) { " +
        "    emit(genre, {average: nrPages, count: 1}); " +
        "  }); " +
        "} ";
  String reduce = "function(genre, values) { " +
        "  var s = 0; var newc = 0; " +
        "  values.forEach(function(curAvg) { " +
        "    s += curAvg.average * curAvg.count; " +
        "    newc += curAvg.count; " +
        "  }); " +
        "  return {average: (s / newc), count: newc}; " +
        "} ";

  MapReduceIterable<Document> result = db.getCollection("books")
        .mapReduce(map, reduce);
  for (Document r : result)
        System.out.println(r);
}
```

Yes SQL?

Some early-adaptors of NoSQL were confronted with some sour lessons

- The FreeBSD maintainers speaking out against MongoDB's lack of on-disk consistency support
- Digg struggling with the NoSQL Cassandra database after switching from MySQL
- Twitter facing similar issues as well (which also ended up sticking with a MySQL cluster for a while longer)
- The fiasco of HealthCare.gov, where the IT team also went with a badly-suited NoSQL database

Some queries or aggregations particularly difficult, with MapReduce interfaces harder to learn and use

Large consistency problems in e.g. early MongoDB versions

NoSQL databases focusing on scalability often do so by using an "eventual consistency" mechanism

Yes SQL?

In recent years, the line between NoSQL and relational databases has become blurred throughout the past years, and why we saw vendors of relational databases catching up and implementing some of the interesting aspects which made NoSQL databases, and document stores especially, popular in the first place, such as:

- Focus on horizontal scalability and distributed querying
- Dropping strict schema requirements
- Support for nested data types or allowing to store JSON directly in tables
- Support for map-reduce operations

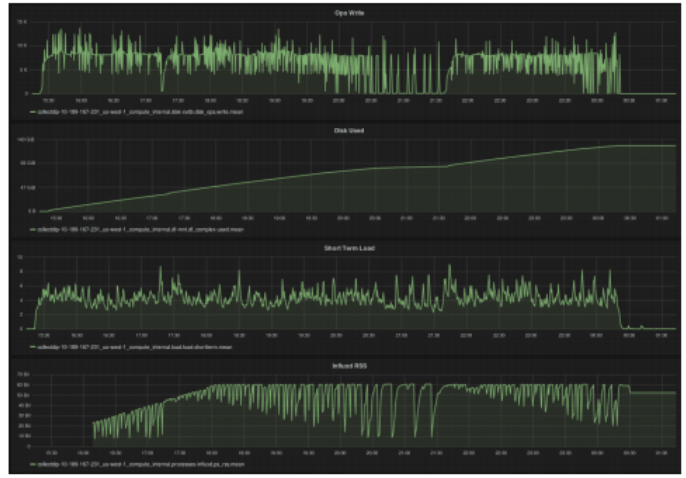
This comes backed by a strong querying backend and SQL querying capabilities!

We also see many NoSQL vendors focusing again on robustness and durability, and moving away from map-reduce based pipelines e.g. CockroachDB

NoSQL

As such, the most interesting databases that evolved from the field are not those with an extreme focus on distributed storage or scalability, but those that come with interesting new data paradigms...

E.g. time series databases



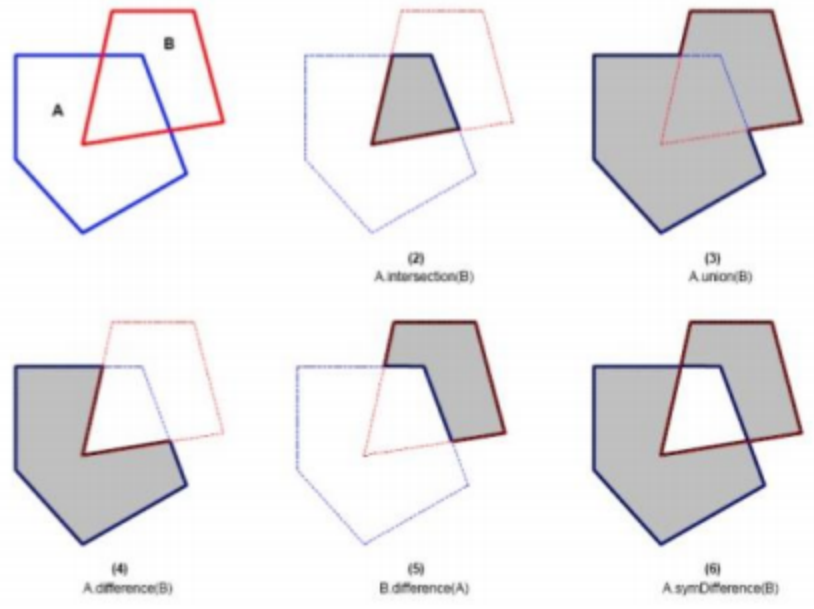
- Optimized for time-stamped or time series data
- Server metrics and performance monitoring
- Network data
- Sensor data
- Events, clicks, trades in a market…
- Queries based on analysis tasks over time: windowing, aggregating, joining on time series
- E.g. InfluxDB, Kdb+, TimescaleDB (some extending SQL, some with their own query languages)

NoSQL

As such, the most interesting databases that evolved from the field are not those with an extreme focus on distributed storage or scalability, but those that come with interesting new data paradigms...

E.g. geospatial databases

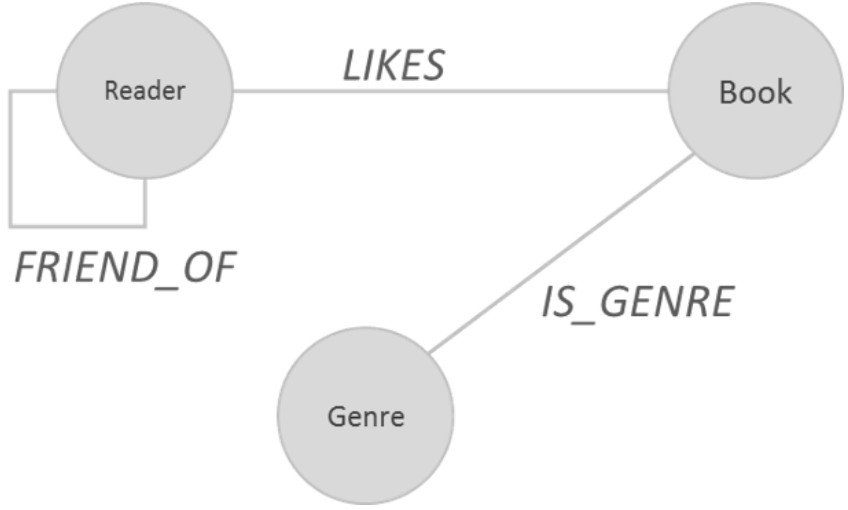
- Optimized for storing and querying data that represents objects defined in a geometric space
- Represented as points, lines, line segments, polygons, complex polygons with holes
- Query operations based on spatial operators: spatial indexes to improve query speed
- E.g. PostgreSQL with PostGIS, ESRI GIS Tools (Hadoop extension), Microsoft SQL Server (supports geospatial extensions), GIS tools such as ESRI



NoSQL

As such, the most interesting databases that evolved from the field are not those with an extreme focus on distributed storage or scalability, but those that come with interesting new data paradigms...

E.g.graph databases



- Graph databases apply graph theory to the storage of information of records
- The reason why graph databases are an interesting category of NoSQL is because, contrary to the other approaches, they actually go the way of **increased relational modeling, rather than doing away with relations**
- That is, one-to-one, one-to-many and many-to-many structures can easily be modeled in a graph based way as well

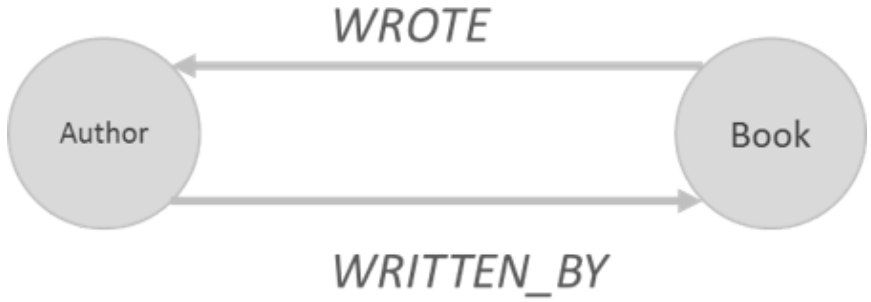
Graph databases

Consider for instance again books having many authors and vice versa

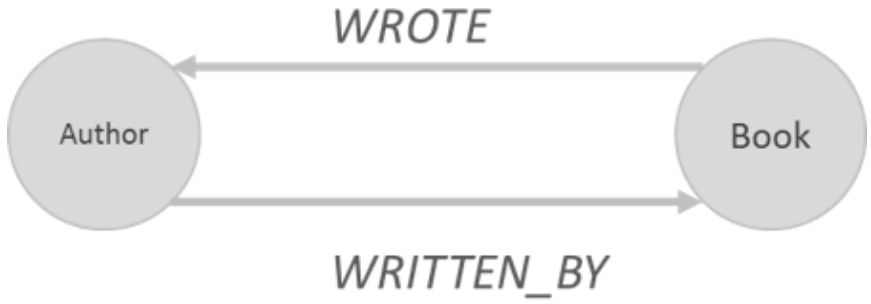
- In an RDBMS, this would be modeled by three tables: one for books, one for authors, and one modeling the many-to-many relation
- A query to return all book titles for books written by a particular author would look like:

```
SELECT title FROM books, authors, books_authors
WHERE author.id = books_authors.author_id
AND books.id = books_authors.book_id
AND author.name = "Seppe vanden Broucke"
```

In a graph database, this structure would be represented as follows:



Graph databases



What would a query look like now?

```
MATCH (b:Book)<-[:WRITTEN_BY]-(a:Author)
WHERE a.name = "Seppe vanden Broucke"
RETURN b.title
```

Here, we're using the Cypher query language, the graph based query language introduced by Neo4j, one of the most popular graph databases

- Other notable implementations of graph databases include AllegroGraph, GraphDB, InfiniteGraph and OrientDB

Graph databases

In a way, a graph database can be seen as a hyper-relational database, where JOIN tables are replaced by more interesting and semantically meaningful relationships that can be navigated (graph traversal) and/or queried, based on graph pattern matching

Note that graph databases differ in terms of representation of the underlying graph data model

- Neo4j, for instance, supports nodes and edges having a type (Book) and a number of attributes (title), next to a unique identifier
- Other systems are geared towards speed and scalability and only support a simple graph representation
- FlockDB, for instance, developed by Twitter, only supports storing a simplified directed graph as a list of edges having a source and destination identifier, a state (normal, removed, or archived), and an additional numeric "position" to help with sorting results
- Twitter uses FlockDB to store social graphs (who follows whom, who blocks whom) containing billions of edges and sustaining hundreds of thousands read queries per second
- Different implementations position themselves differ regarding the trade-off between speed and data expressiveness

Neo4j

Neo4j and Cypher

Like SQL, Cypher is a declarative, text-based query language, containing many similar operations as SQL

However, because it is geared towards expressing patterns found in graph structures, it contains a special MATCH clause to match those patterns

Nodes are represented by parenthesis, representing a circle:

```
()
```

Neo4j and Cypher

Nodes can be labeled in case they need to be referred to elsewhere, and be further filtered by their type, using a colon:

```
(b:Book)
```

Edges are drawn using either `--` or `-->`, representing a unidirectional line or an arrow representing a directional relationship respectively. Relationships can also be filtered by putting square brackets in the middle:

```
(b:Book)<-[:WRITTEN_BY]-(a:Author)
```

Neo4j and Cypher

This is a basic SQL SELECT query:

```
SELECT b.* FROM books AS b;
```

Which can be expressed in Cypher as follows:

```
MATCH (b:Book) RETURN b;
```

`ORDER BY` and `LIMIT` statements can be included as well:

```
MATCH (b:Book) RETURN b ORDER BY b.price DESC LIMIT 20;
```

Neo4j and Cypher

`WHERE` clauses can be included explicitly

```
MATCH (b:Book)
WHERE b.title = "Beginning Neo4j"
RETURN b;
```

… or as part of the `MATCH` clause:

```
MATCH (b:Book {title:"Beginning Neo4j"})
RETURN b;
```

Neo4j and Cypher

`JOIN` clauses are expressed using direct relational matching

The following query returns a list of distinct customer names who purchased a book written by Seppe, are older than 30, and paid in cash:

```
MATCH (c:Customer)-[p:PURCHASED]->(b:Book)<-[:WRITTEN_BY]-(a:Author)
WHERE a.name = "Wilfried Lemahieu"
AND c.age > 30
AND p.type = "cash"
RETURN DISTINCT c.name;
```

Neo4j and Cypher

Say we have a tree of book genres, and books can be placed under any category, with categories organized as a tree. Performing a query to fetch a list of all books in the category “Programming” and all its subcategories can become problematic in SQL, even with extensions that support recursive queries

Yet, Cypher can express queries over hierarchies and transitive relationships of any depth simply by appending a star `*` after the relationship type and providing optional `min..max` limits in the `MATCH` clause:

```
MATCH (b:Book)-[:IN_GENRE]->(:Genre)-[:PARENT*0..]-(tg:Genre)
WHERE tg.name = "Programming"
RETURN b.title;
```

As a result, all books in the category “Programming”, but also in any possible subcategory, sub-subcategory, and so on will be retrieved. (I.e. friend-of-a-friend)

Neo4j and analytics

> Neo4j is optimized for online transaction processing (OLTP) and is intended to be used as your primary database

It originally wasn't specifically with the intention of being used for graph compute or analytics

- But now: https://neo4j.com/blog/efficient-graph-algorithms-neo4j/
  - https://github.com/maxdemarzi/graph_processing: Page Rank, Label Propagation, Union Find, Betweenness Centrality, Closeness Centrality, Degree Centrality (now merged into Neo4j)
  - https://github.com/neo4j-contrib/neo4j-graph-algorithms/issues/271: not yet personalized PageRank :/
- https://github.com/graphaware/neo4j-framework
  - GraphAware Framework speeds up development with Neo4j by providing a platform for building useful generic as well as domain-specific functionality, analytical capabilities, (iterative) graph algorithms, etc.
  - https://github.com/graphaware/neo4j-noderank: GraphAware Timer-Driven Runtime Module that executes PageRank-like algorithm on the graph: now retired and merged into the graph algorithm module

Neo4j and Spark

https://github.com/neo4j-contrib/neo4j-spark-connector

- The Neo4j Spark Connector uses the binary Bolt protocol to transfer data from and to a Neo4j server
- Normally Neo4j is access through a JSON-driven REST api
- Bolt is a binary, speedier access method
- It offers Spark-2.0 APIs for RDD, DataFrame, GraphX and GraphFrames, so you're free to chose how you want to use and process your Neo4j graph data in Apache Spark
- Still maintained

Neo4j and Spark

```
import org.neo4j.spark._
val neo = Neo4j(sc)
import org.graphframes._

val graphFrame = neo.pattern(("Person","id"),("KNOWS",null),
        ("Person","id")).partitions(3).rows(1000).loadGraphFrame

graphFrame.vertices.count
//     => 100
graphFrame.edges.count
//     => 1000

val pageRankFrame = graphFrame.pageRank.maxIter(5).run()
val ranked = pageRankFrame.vertices
ranked.printSchema()

val top3 = ranked.orderBy(ranked.col("pagerank").desc).take(3)
// => top3: Array[org.apache.spark.sql.Row]
// => Array([236716,70,0.62285...], [236653,7,0.62285...],
//          [236658,12,0.62285])
```

Neo4j and Spark

https://github.com/neo4j-contrib/neo4j-mazerunner

- (Retired)

Neo4j and R/Python

`igraph` and `NetworkX`: https://github.com/versae/ipython-cypher: queries are send through Cypher and results can be can be stored in a variable and then converted to a Pandas DataFrame

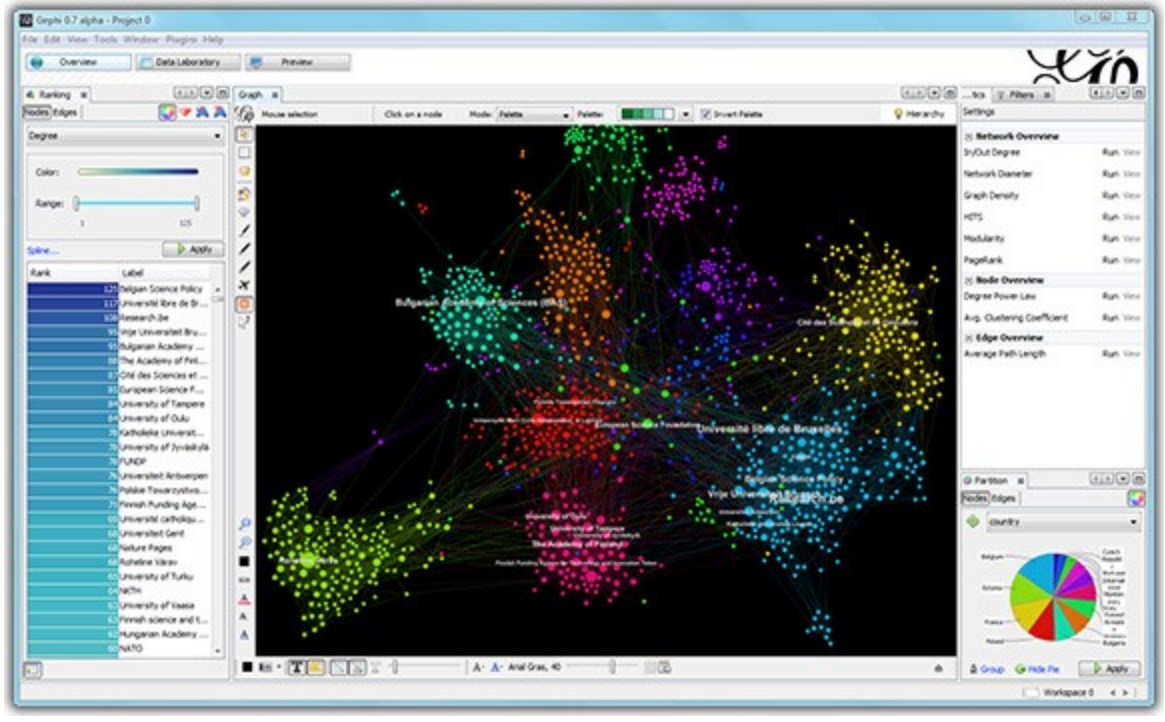
`RNeo4j` package: similar approach to R data frames

`Py2Neo`: https://py2neo.org/v4/: Python client package to connect to Neo4j database

Graph visualization and exploration

- Most graph layout techniques make use of a physics-inspired “force based” approach, where the edges between nodes are regarded as “springs” and the layout algorithm goes through a number of iterations to let the graph stabilize towards a comprehensible, attractive layout
  - ForceAtlas2
  - Examples to play with in the browser: https://bl.ocks.org/steveharoz/8c3e2524079a8c440df60c1ab72b5d03
  - Webcola is a JavaScript library to layout graphs: http://marvl.infotech.monash.edu/webcola/
- Many JavaScript-based tools are available to visualise and layout graphs in the browser:
  - http://js.cytoscape.org/
  - http://sigmajs.org/
  - http://visjs.org/
- GraphViz is a standalone tool for graph-based visualizations and layout, which are described by means of the DOT language. It has lots of bindings with programming languages available and is still widely used as a behind-the-scenes layout driver in many products: http://www.graphviz.org/
- Gephi is a tool for graph layout, analysis and visualisation: https://gephi.org/

Graph visualization and exploration: Gephi



Graph visualization and exploration: Gephi

https://gephi.org/:

- Exploratory analysis: by networks manipulations in real time
- Link analysis: revealing the underlying structures of associations between objects
- Easy creation of social data connectors to map community organizations and small-world networks
- Metrics: e.g. centrality, degree, betweenness, closeness
- And more: density, path length, diameter, HITS, modularity, clustering coefficient
- Can load in various Graph file formats: GDF (GUESS), GraphML (NodeXL), GML, NET (Pajek), GEXF
- Customizable by plugins: layouts, metrics, data sources, manipulation tools, rendering presets and more
- Not really spurious active development, but still one of the better and easier tools to get started with and capable to handle larger graphs
- Anything beyond in size will require lots of custom coding anyway

Neo4j and Gephi

Either export out the data you want to analyze to a file format Gephi can understand

- E.g. through the aforementioned R / Python packages
- Or use neo4j-shell-tools to do a GraphML export
- Or use APOC: https://tbgraph.wordpress.com/2017/04/01/neo4j-to-gephi/

https://gephi.org/plugins/#/plugin/neo4j-graph-database-support

- Plugin which loads in Neo4j's data files directly
- Doesn't work well with newer Neo4j versions

https://github.com/gephi/gephi-plugins/tree/neo4j-plugin

- Also old…

Use cases


GraphGist Challenge Entries


Sports and Recreation


Master Data Management


Real-Time Recommendations


Optimization


Fraud Detection


Pop Culture


Network and IT Operations


Holidays


Graph-Based Search


General Business


Graph Gist How-tos


Data Analysis


Public Web APIs


Internet of Things

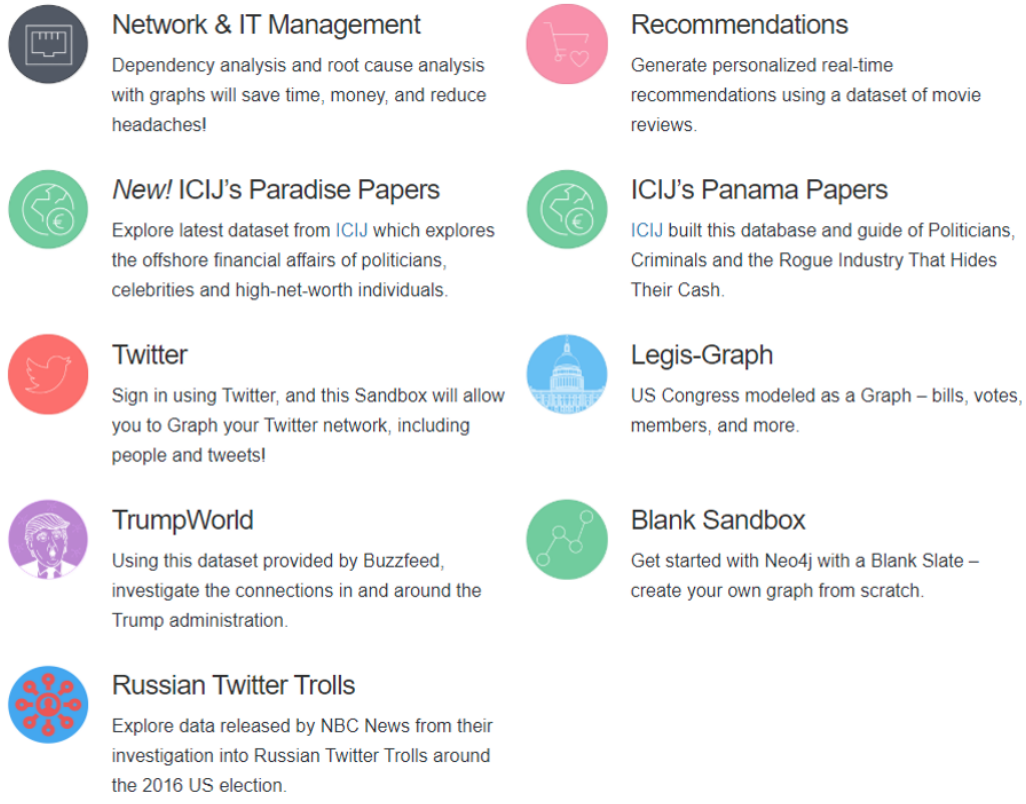

Investigative Journalism


Open Government Data and Politics


Identity and Access Management

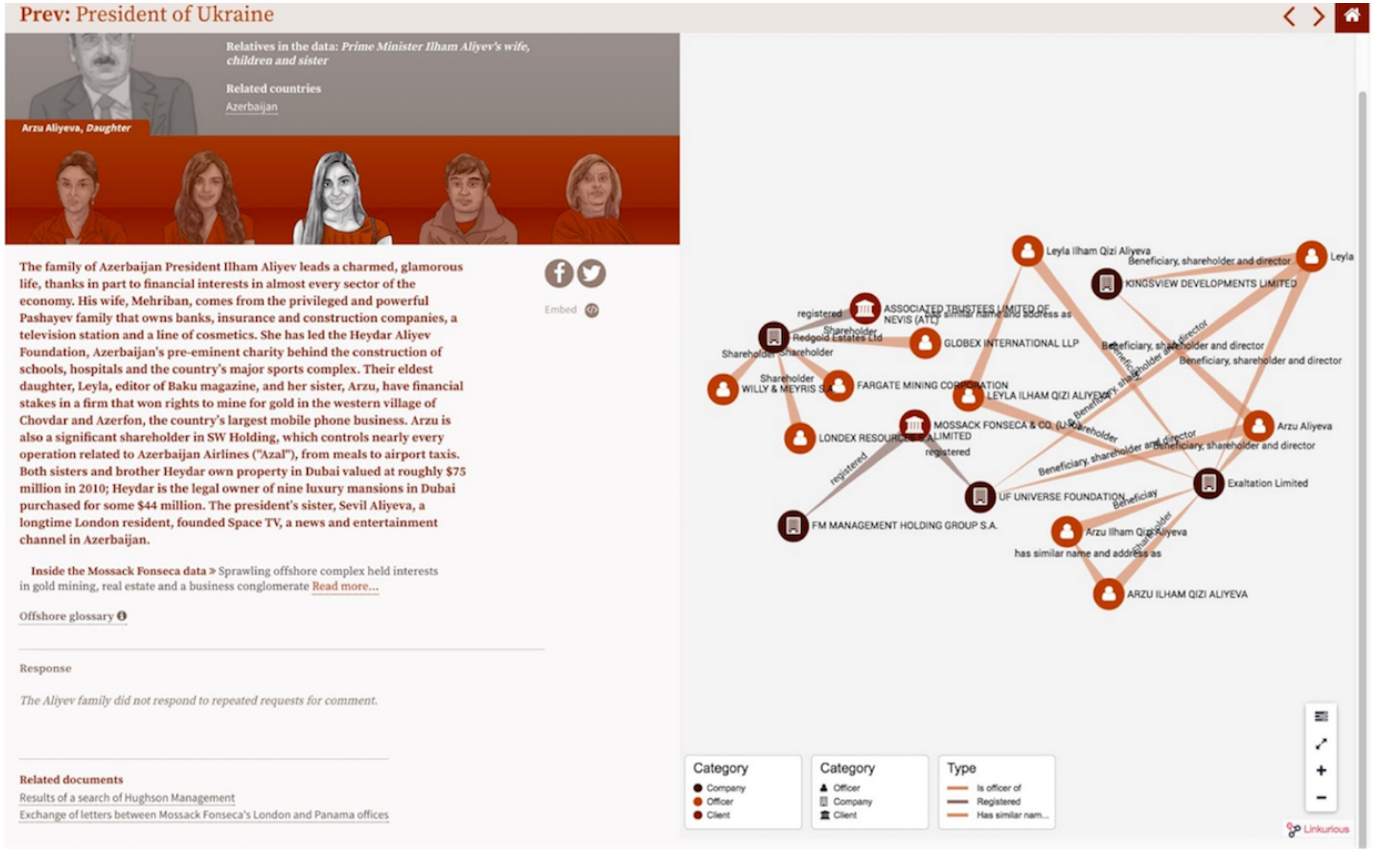
https://neo4j.com/graphgists/

Use cases



https://neo4j.com/sandbox-v2/

Use cases



https://neo4j.com/blog/analyzing-panama-papers-neo4j/

Use cases