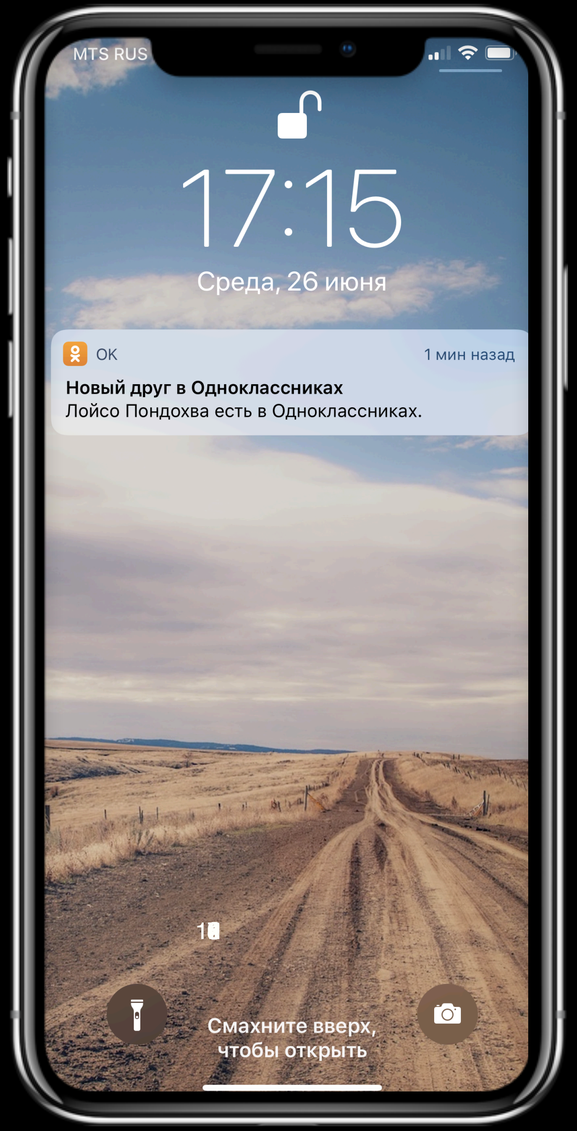
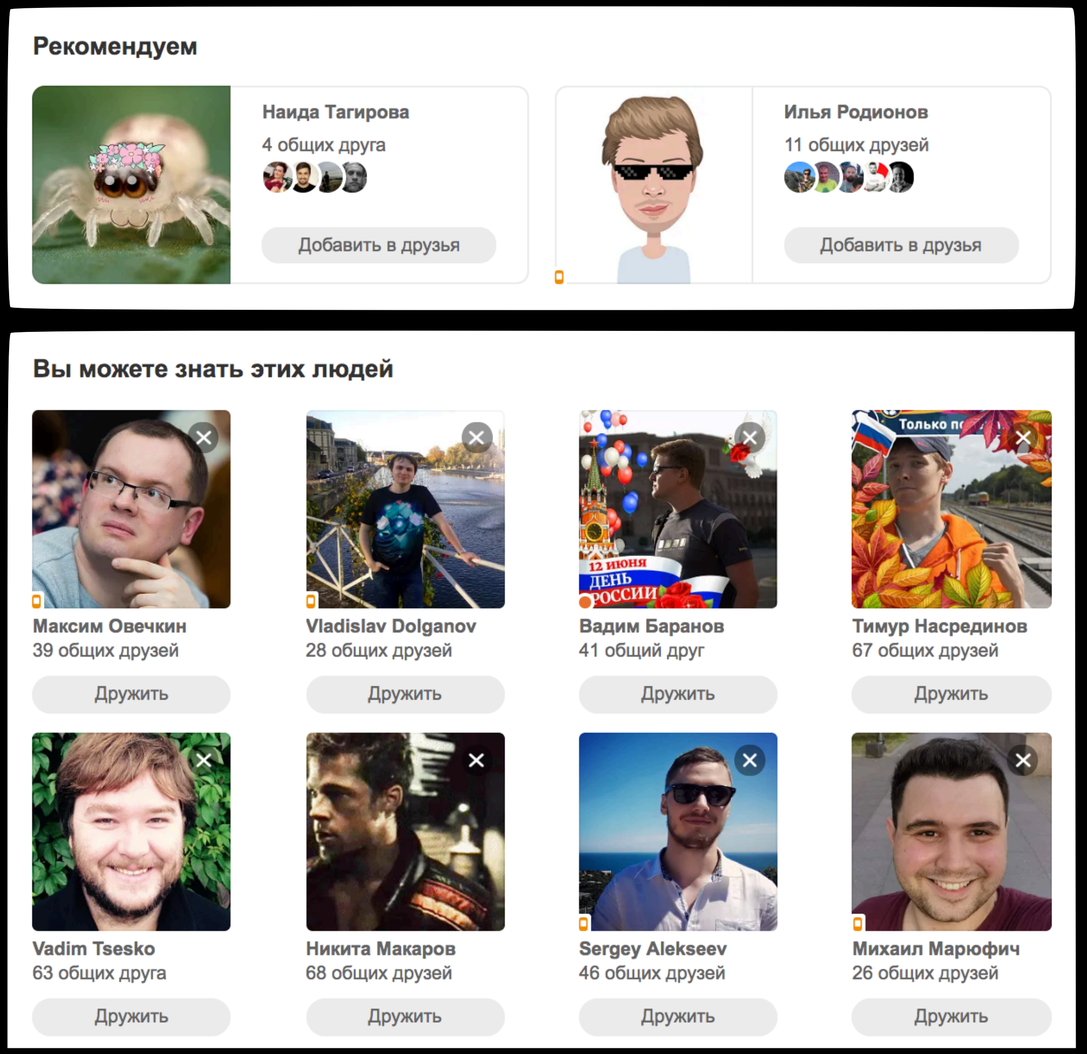
Recommendation Systems

Unclassical embeddings + Learning to **rank**

Eugeny Malyutin / Sergey Dudorov

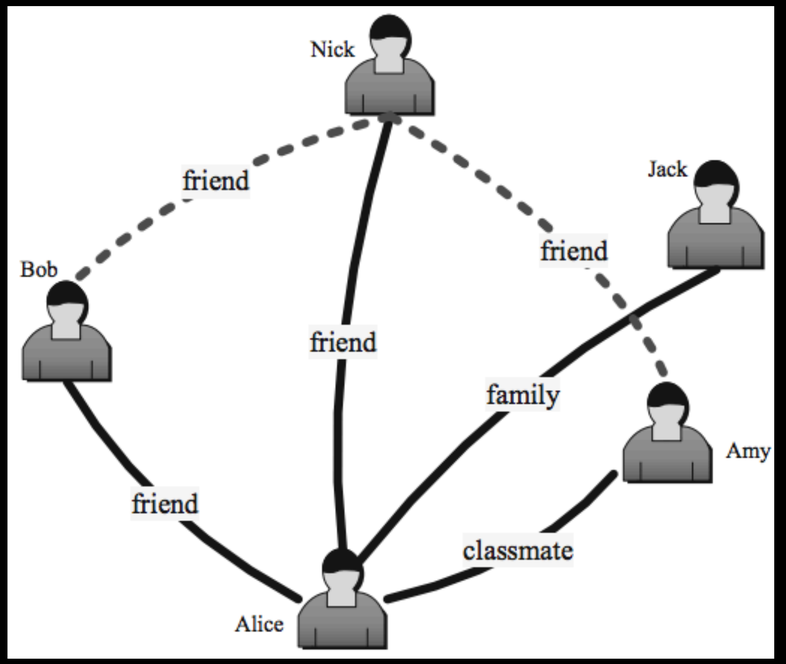
PYMK:


MTS RUS
17:15
Среда, 26 июня
OK
1 мин назад
Новый друг в Одноклассниках
Лойсо Пондохва есть в Одноклассниках.
Смахните вверх,
чтобы открыть


Рекомендуем
Наида Тагирова
4 общих друга
Добавить в друзья
Илья Родионов
11 общих друзей
Добавить в друзья
Вы можете знать этих людей
Максим Овечкин
39 общих друзей
Дружить
Vladislav Dolganov
28 общих друзей
Дружить
Вадим Баранов
41 общий друг
Дружить
Тимур Насреднов
67 общих друзей
Дружить
Vadim Tsesko
63 общих друга
Дружить
Никита Макаров
68 общих друзей
Дружить
Sergey Alekseev
46 общих друзей
Дружить
Михаил Марюфич
26 общих друзей
Дружить

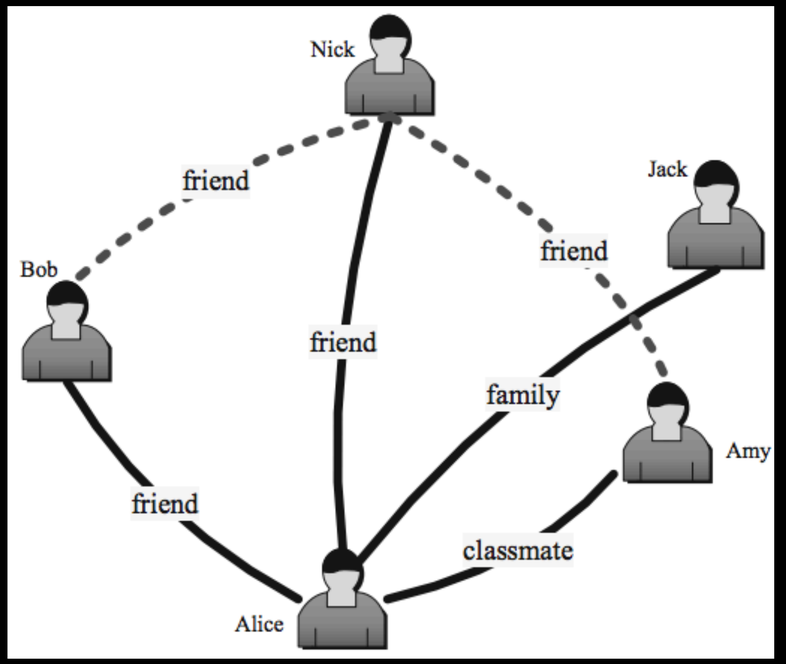
Link prediction(recovery):

- Treat link prediction as binary classification.
- Each unconnected pair — corresponds to an instance with features and label.
- If there is a potential link — instance labeled as «positive», otherwise — negative.



Link prediction(recovery):

- Treat link prediction as binary classification.
- Each unconnected pair — corresponds to an instance with features and label.
- If there is a potential link — instance labeled as «positive», otherwise — negative.



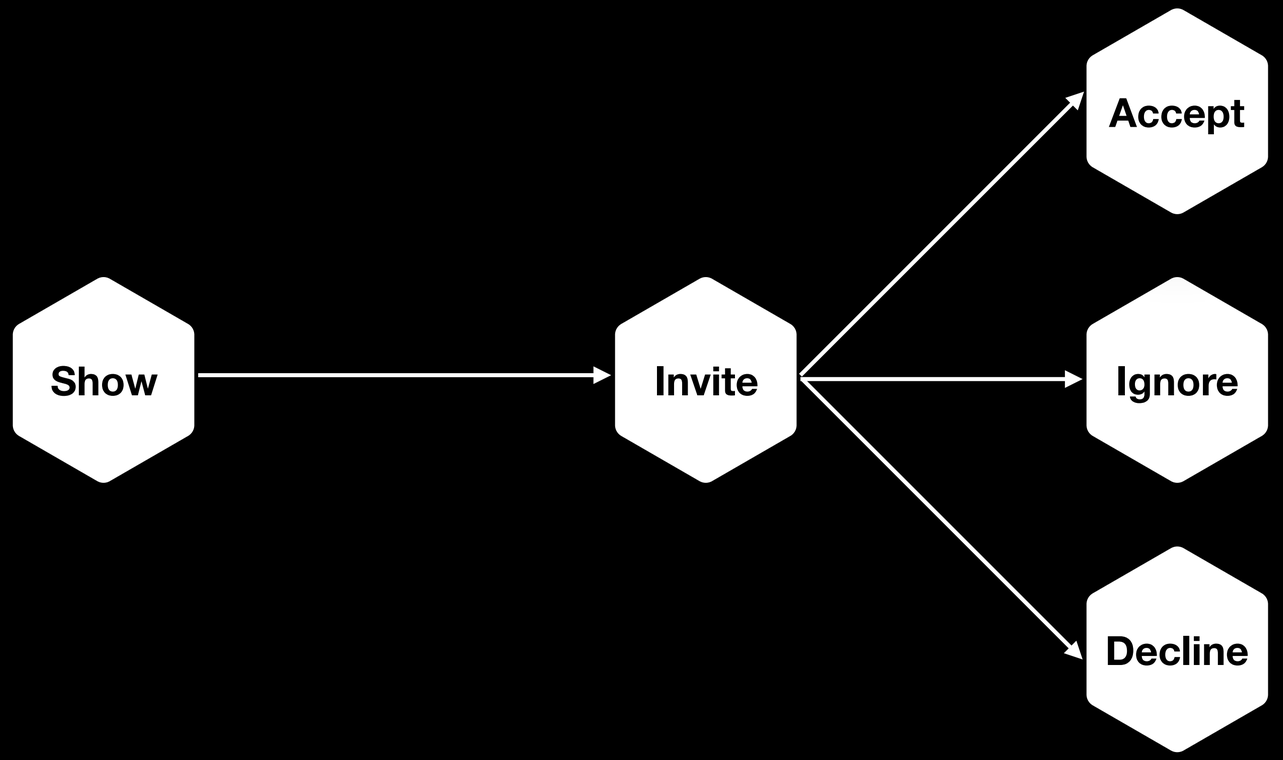
PROBLEMS?

Link prediction(recovery):

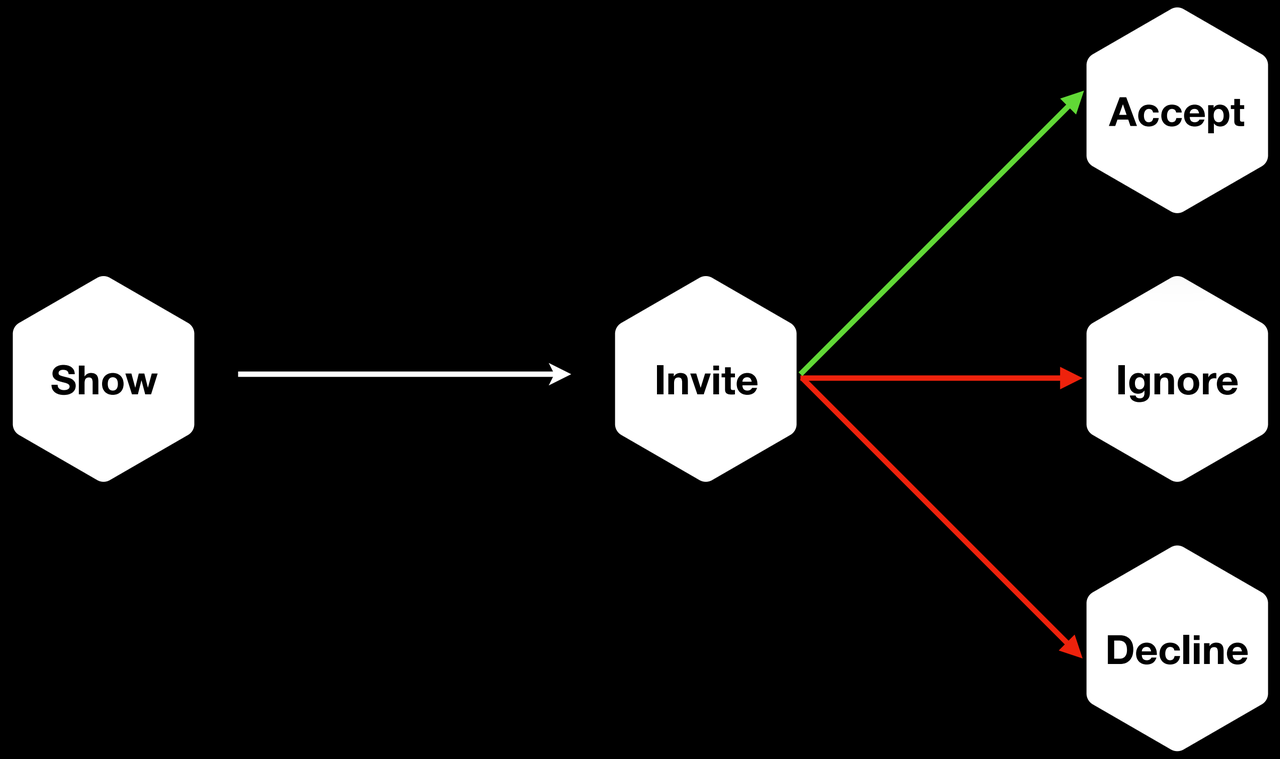
- Treat link prediction as binary classification.
- Each unconnected pair — corresponds to an instance with features and label.
- If there is a potential link — instance labeled as «positive», otherwise — negative.

- What should we use as a **target**?
- Should we score **each unconnected pair** of nodes in 200M+ nodes graph?
- So what should **we exactly do** when «there is a potential link»?

Target



Target



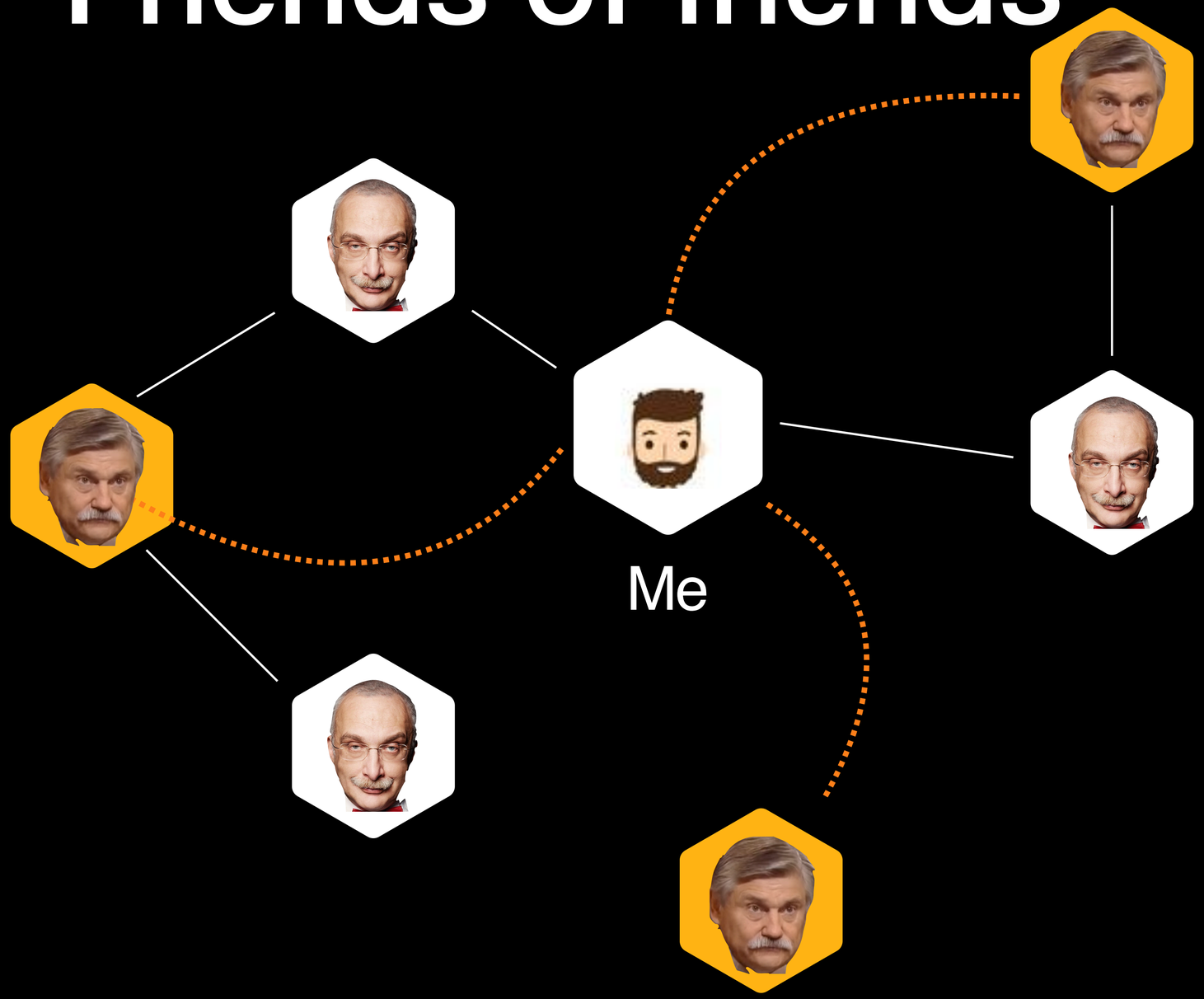
Recommendation System schema:

- Matrix factorisation + K-NN
- Recently acted
- Friends-of-friends
- Reverse index

  ...

- Embeddings dot
- Neural Networks
- ⭑ Machine learning model (xgboost ^^)

Friends of friends



- friends

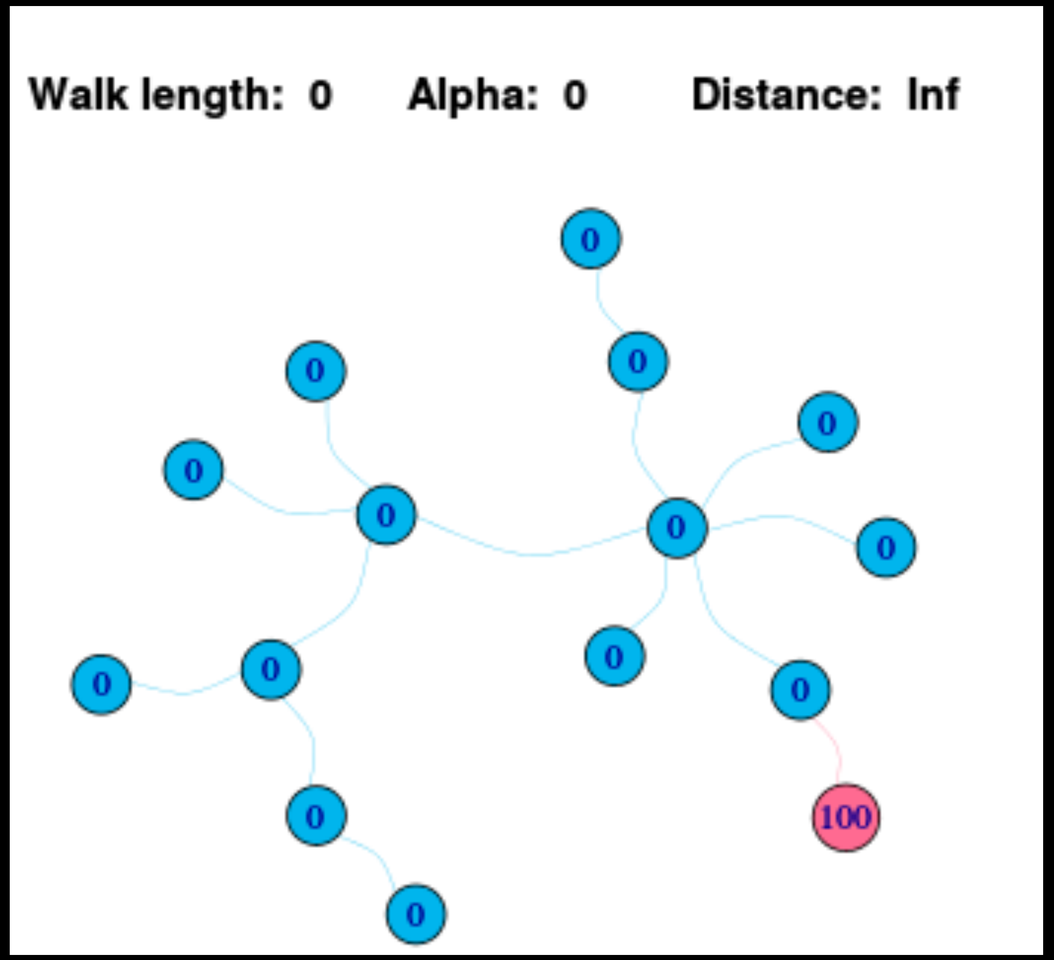
- candidates
(friends of friends)

What’s more?

- Last active interaction
- Phonebook
- Reverse phonebook
- Together on photo
- Graph embeddings… ?

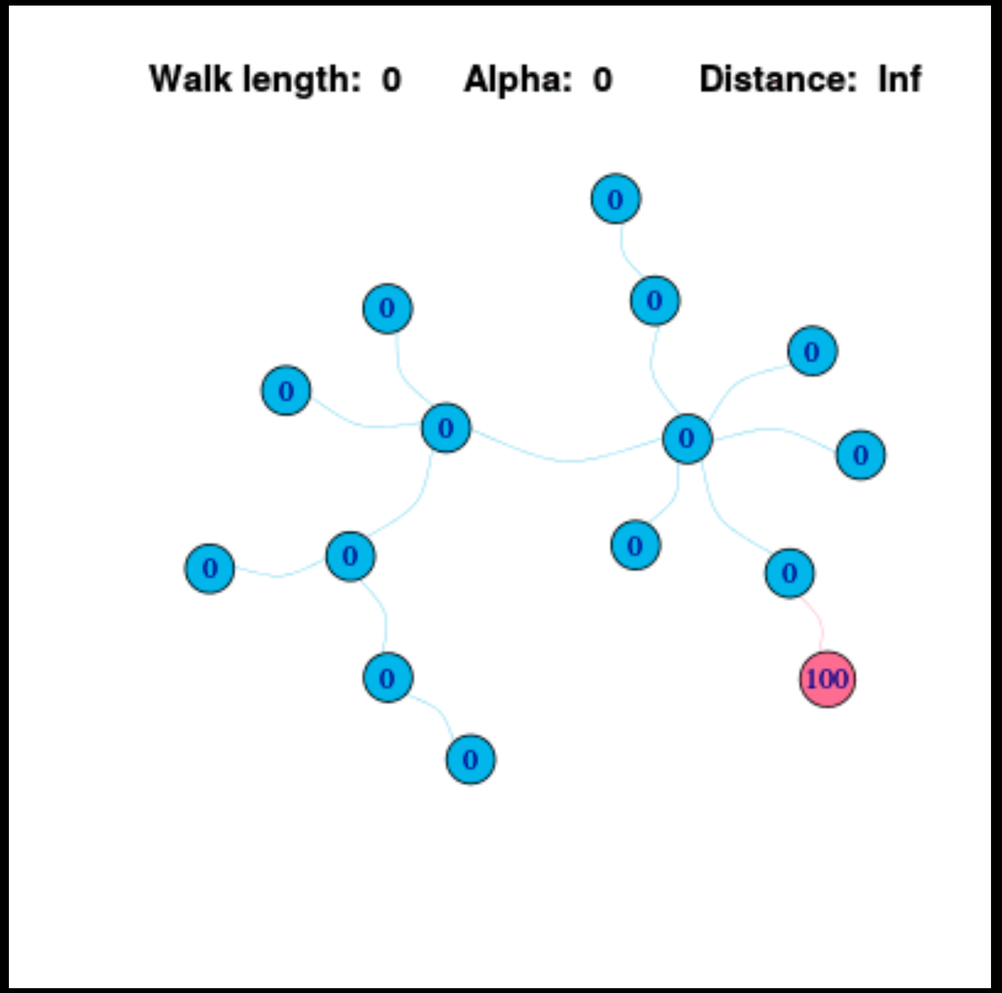
PageRank: random walks

- Where does it spread first?
- How far does it spread?
- Will the vertices, close to the red vertex obtain more information then those far away?
- Will the information continue to go back and forth forever or will we reach a stable distribution eventually?



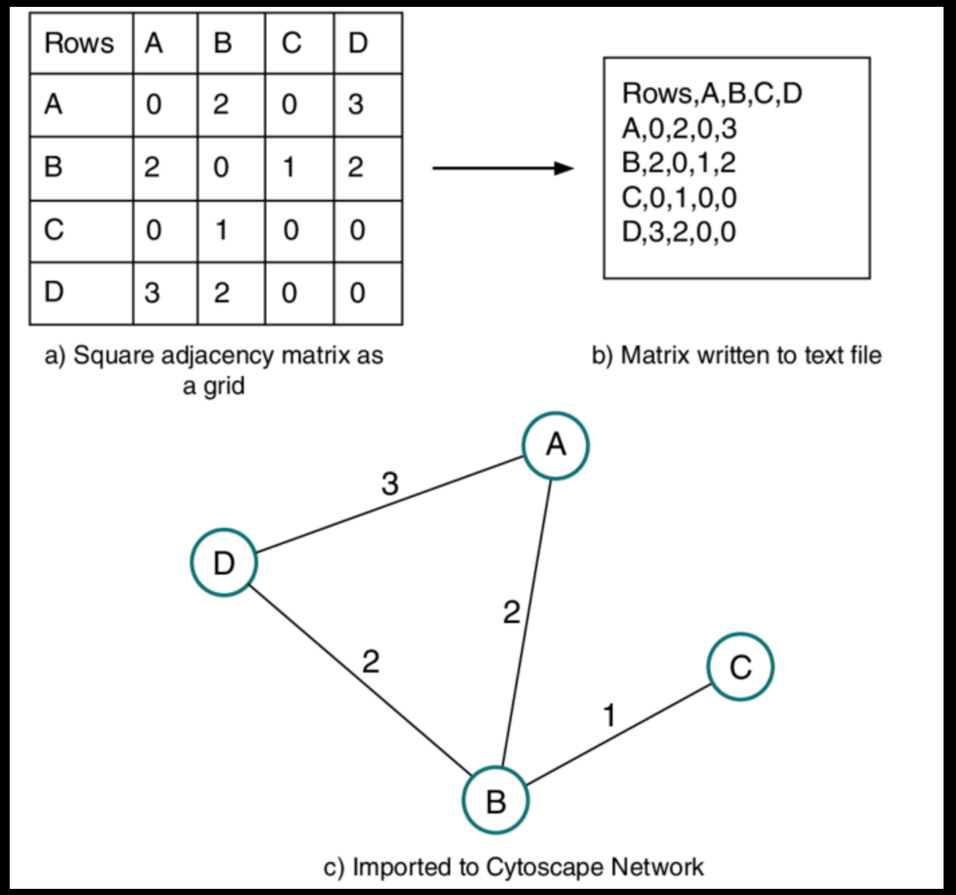
PageRank: random walks

- Goes from red through random edge to next adjacent vertex
- If we repeat thus experiment infinite number of times - will it converge?
- Let assume A — matrix of transition probabilities, x — initial distribution.
- 1st step x’ = Ax
- 2nd step x” = Ax’
- n’th step: x^n = A x^(n-1)



PageRank: random walks

- 1st step x’ = Ax
  2nd step x” = Ax’
  n’th step: x^n = A x^(n-1)
- Convergence: x^n = A * x^(n-1)…
- $Ax = \lambda x$ doesn’t look similar?
- Final distribution of weights - is Eigenvector of matrix A with eigenvalue lambda = 1.
- Do you feel any problems?
- But wait ... how do we know that A has an eigenvalue of 1, so that such a vector p exists? And even if it does exist, will it be unique (well-defined)?



Rows	A	B	C	D
A	0	2	0	3
B	2	0	1	2
C	0	1	0	0
D	3	2	0	0

a) Square adjacency matrix as a grid

```
Rows,A,B,C,D
A,0,2,0,3
B,2,0,1,2
C,0,1,0,0
D,3,2,0,0
```

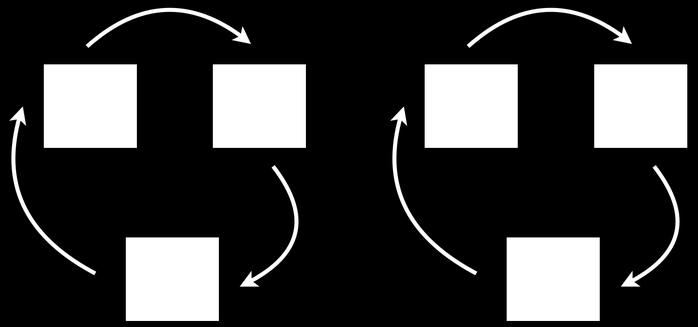
b) Matrix written to text file

c) Imported to Cytoscape Network

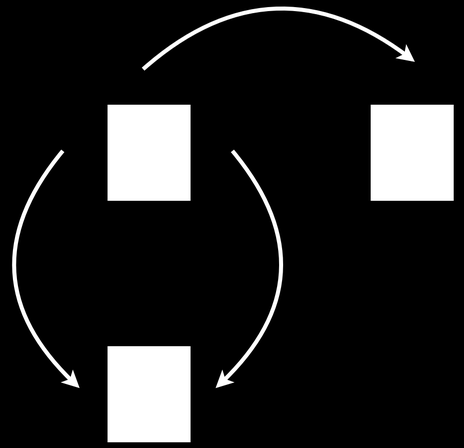
PageRank: as Markov chain

- Markov chain - random process over states $s_1, s_2 \ldots s_n$ defined by transition matrix **P** n x n.
- Let denote $p^{(0)}$ as initial distribution vector. $p^{(n)} = P * p^{(n-1)}$
- A stationarity distribution is probability vector $p = P * p$ (oops)
- If the Markov chain is **strongly connected**, meaning that any state can be reached from any other state, then the stationary distribution **p exists and is unique.** Furthermore, we can think of the stationary distribution as the of proportions of visits the chain pays to each state after a very long time (the ergodic theorem):
- P - problems?

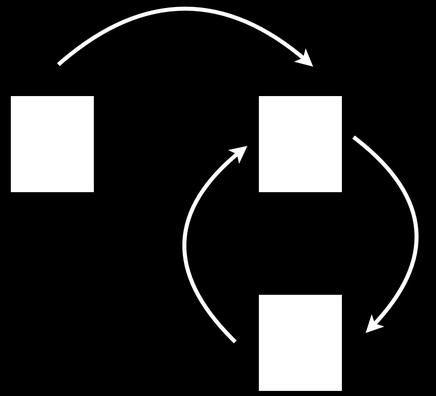
PageRank as Markov chain:



Disconnected components



Dangling links

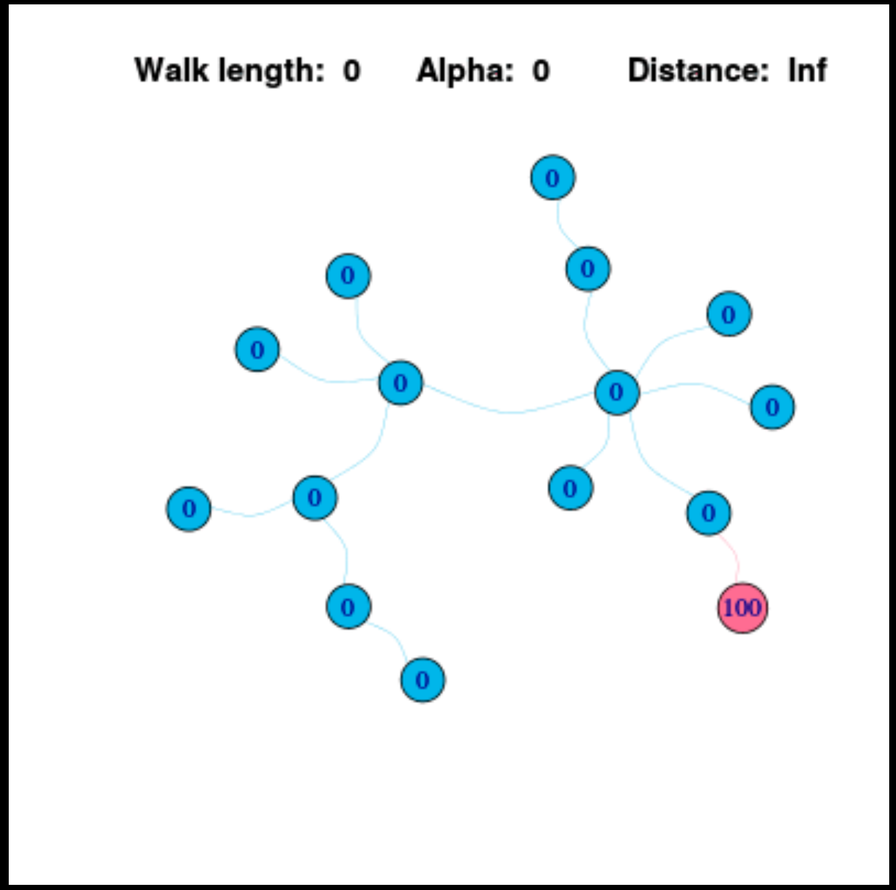


Loops

- Actually, even for Markov chains that are not strongly connected, a stationary distribution always exists, but may **nonunique**.

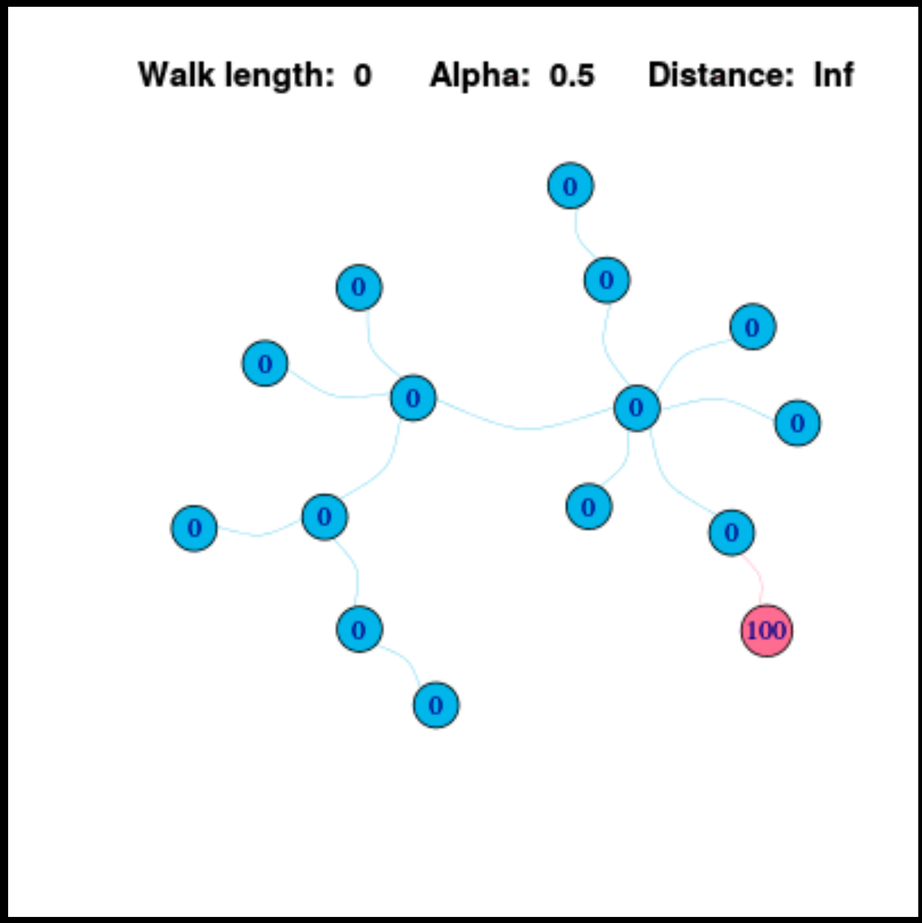
Lazy random walks

- The idea of **lazy** random walks: we allow the random walkers to remain on a vertex with probability 1/2.
- Leads to $x' = 1/2(A + I) * x$. There are some more complicated formulas for eigenvectors.
- Edges with an adjacent vertex that has a high degree tend to be colored red. These are the vertices which contain a significantly larger part of information than the rest of the network.
- No difference where information **originates from**. This means that high-degree vertices will contain proportionally more information than low-degree vertices.



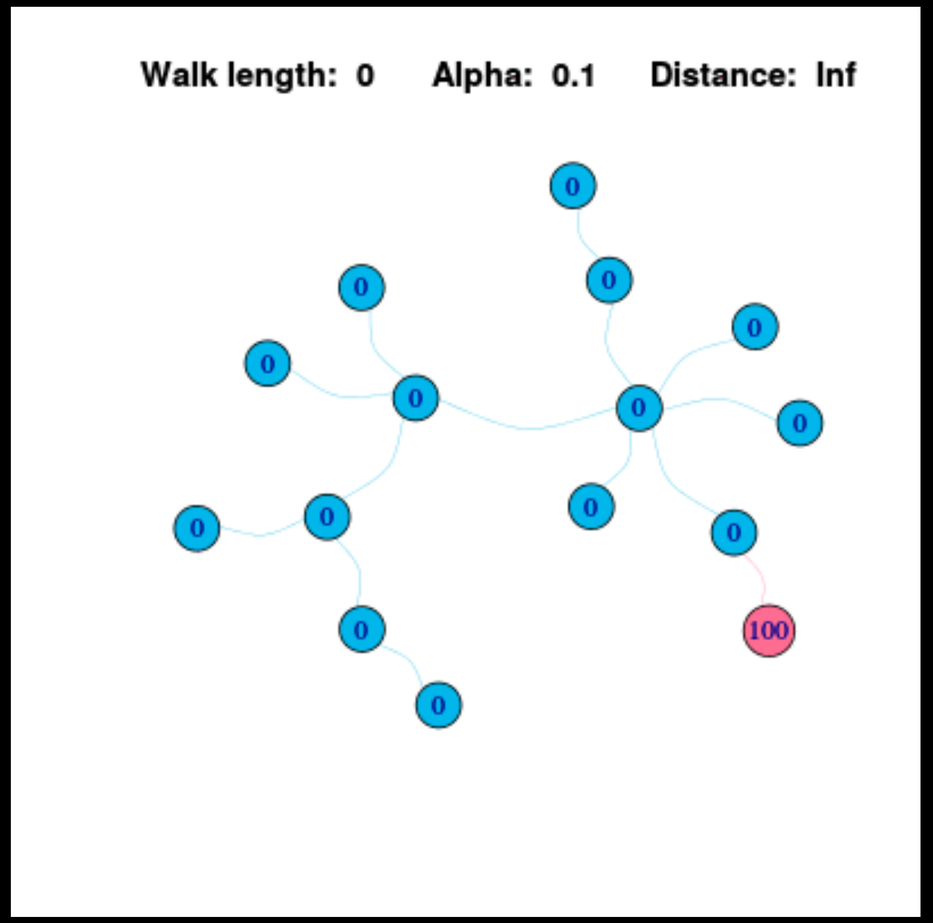
Personalized PageRank:

- Let introduce α as probability to teleport back to source vertex. It tends to make our distribution more «**centered**» on target vertex
- $x' = (1 - \alpha) * Ax + \alpha * E$, E - with «target» distribution, «one-hot» with 1 at target vertex, for example.
- This schema also known as «personalized page rank». We may interpret it as «walks» from home with **avg length: 1/alpha**

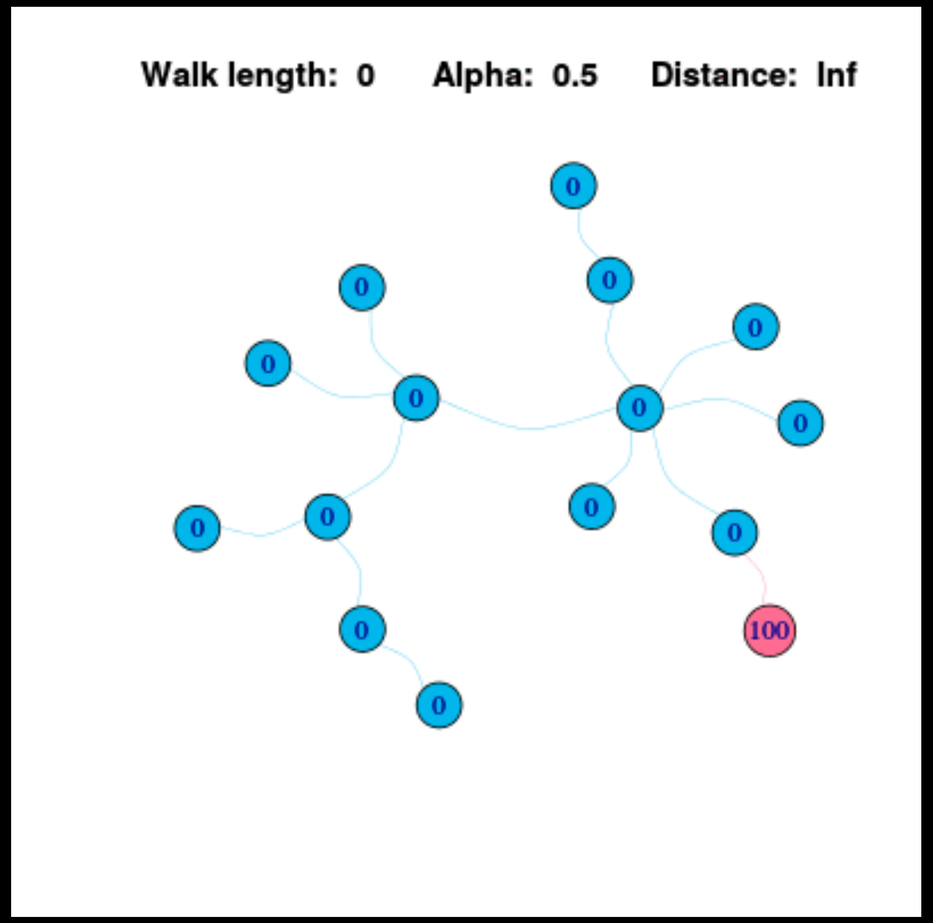


Alpha = 0.5

Personalized PageRank:



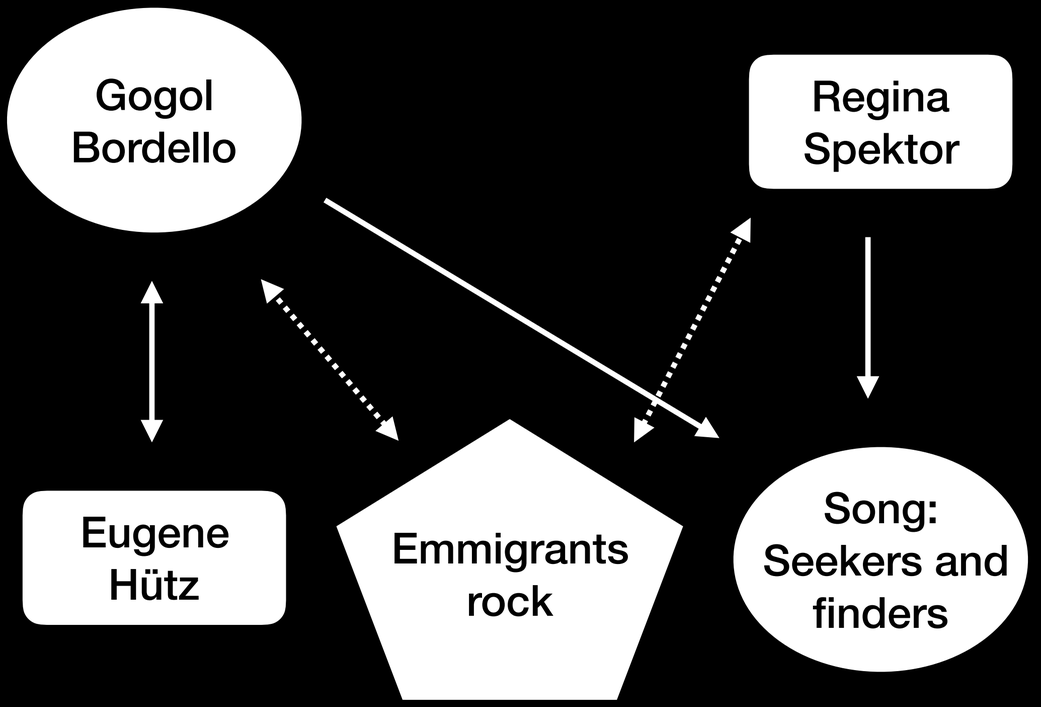
Alpha = 0.1



Alpha = 0.5

PageRank applications

- Google’s first score algorithm
- PYMK as candidates generator
- Not only PYMK, but any other graph-representable product. Music recommender, for example: https://arxiv.org/pdf/1310.7428.pdf



PageRank tips and tricks

- Do you really wanna search for eigenvalues thus matrixes for every user?
  $x' = (1 - \alpha) * Ax + \alpha * E$, in prod we just launches few raw random walks and collect results.

- There are a lot of hyperparameters. For example, different weights for different type of edges. I know no method to tune it but **grid search and a/b tests.**

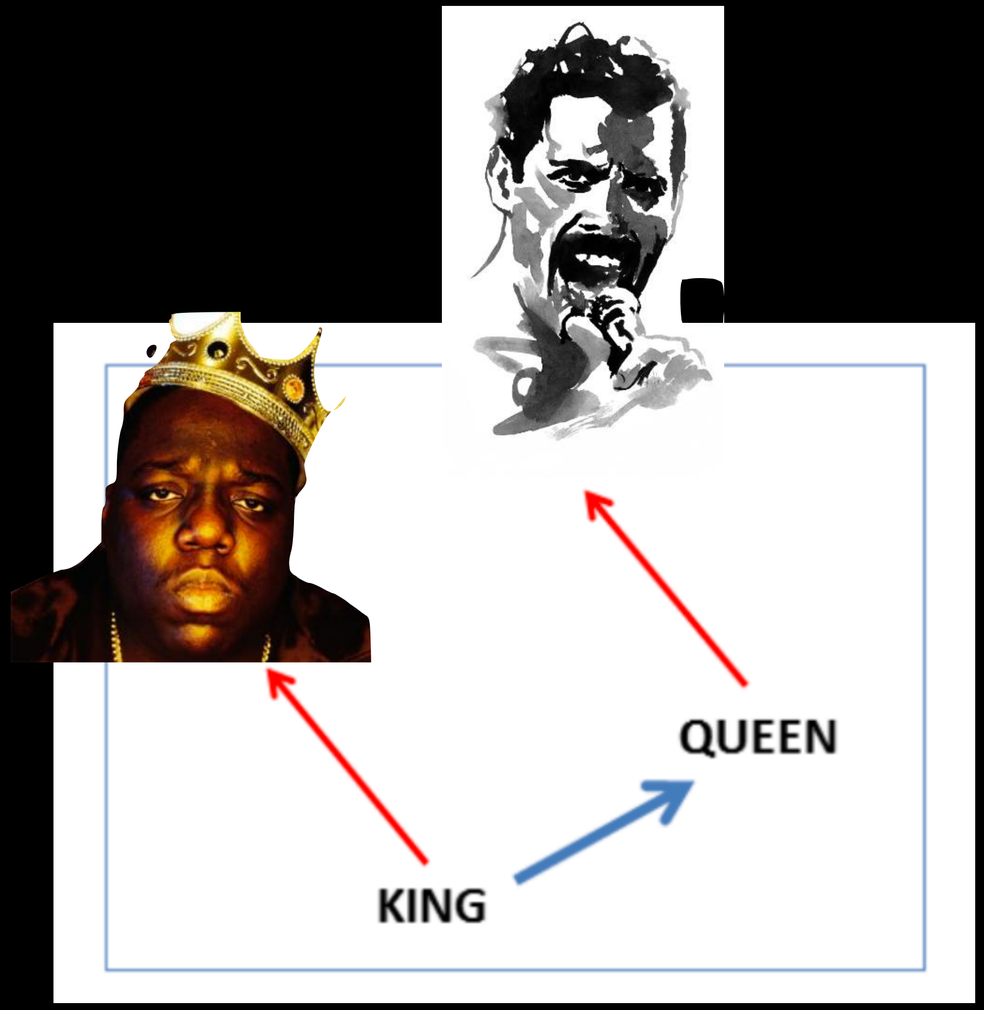
- Good starting point to bring outer worlds knowledge to your model:
  Look at SPARQL
  https://www.w3.org/TR/sparql11-query/

  Query language for semantic web DB based on wikipedia.

PageRank discussion:

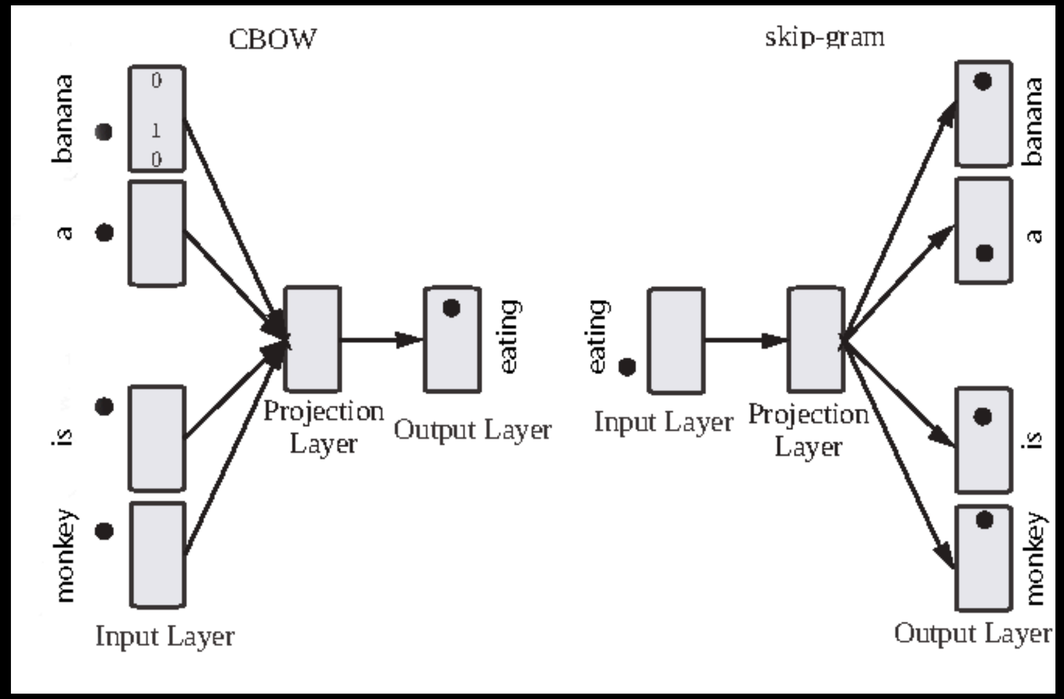
- Random walks instead of eigenvectors in runtime
- A lot of hyperparameters (different types of edge/vertex) and etc.
- **No embeddings** space, only scores and candidates
- …?

Word2vec



Word2vec

- It has an input layer that receives **D** one-hot encoded words which are of dimension **V** (the size of the vocabulary).
- It «averages» them, creating a single input vector.
- That input vector is multiplied by a weights matrix ***W*** (that has size VxD, being D nothing less than the dimension of the vectors that you want to create). That gives you as a result a D-dimensional vector.
- The vector is then multiplied by another matrix (**R** - reverse W), this one of size DxV. The result will be a new V-dimensional vector.
- That V-dimensional vector is normalized to make all the entries a number between 0 and 1, and that all of them sum 1, using the softmax function, and that's the output. It has in the i-th position the predicted probability of the i-th word in the vocabulary of being the one in the middle for the given context.



Word2Vec in RecSys

- Negative sampling
- Next item in sequence as a task
- Good initialisation for «Embedding layer»

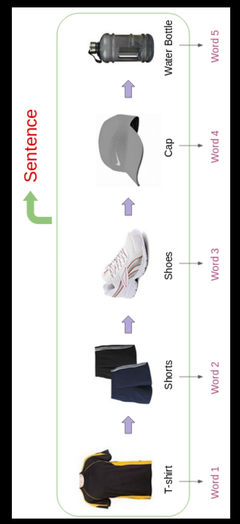
- Filter your datasets, please

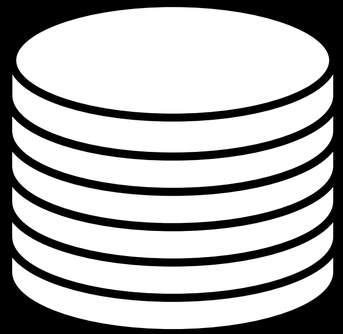
Word2Vec in RecSys

- Negative sampling
- Next item in sequence as a task
- Good initialisation for «Embedding layer»

- Filter your datasets, please

Problem?





KNN index

User vector

Candidates

ML for ranking

Word2vec offline model with embeddings

W2V:

- **No user** vector:
  - Sum all items user interacted with
  - Sum with weights ~ activity
  - Sum with TF-IDF
  - Cluster and select medoid…
  - Feed to an attent…

- **Pros:**
  - Easy to realise
  - No need to limit users interaction
  - Good at similars
  - Session-based
  - Consume CPU , not GPU
- **Cons:**
  - Nothing except interactions
  - Cold start with items
  - No user vector

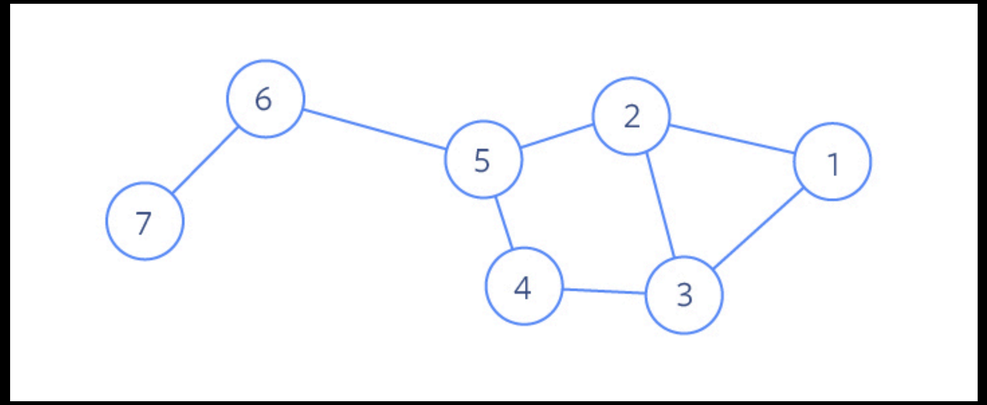
BigGraph by Facebook:

- Facebook graph-scale system to train graph embeddings
- G = (V, R, E) - nodes, set of relations, edges
- $f(\theta_s, \theta_r, \theta_d) = sim(g_s(\theta_s, \theta_r), g_d(\theta_d, \theta_r))$ - score function.

  $\theta_s, \theta_r, \theta_d$ — embeddings of source, relation and destination

  g_s, g_d — mapping operators to introduce different types of mapping for relations
- Negative sampling + distributed engine

- Imagine:
  $g_s(\theta, \theta_r) = g_d(\theta, \theta_r) = \theta$
  $sim(\theta_s, \theta_d) = cos(\theta_s, \theta_d)$
  ^^
  formally equals word2vec

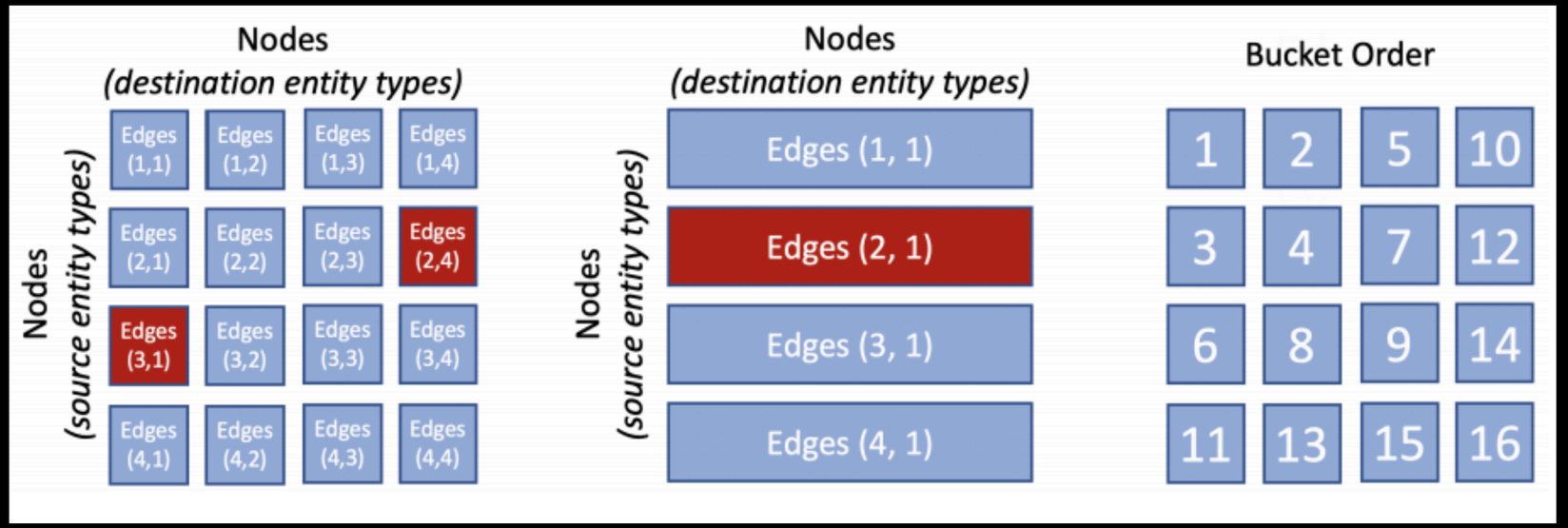


BigGraph:

- **Left:** nodes are divided into P partitions that are sized to fit in memory. Edges are divided into buckets based on the partition of their source and destination nodes. In distributed mode, multiple buckets with non-overlapping partitions can be executed in parallel (red squares).

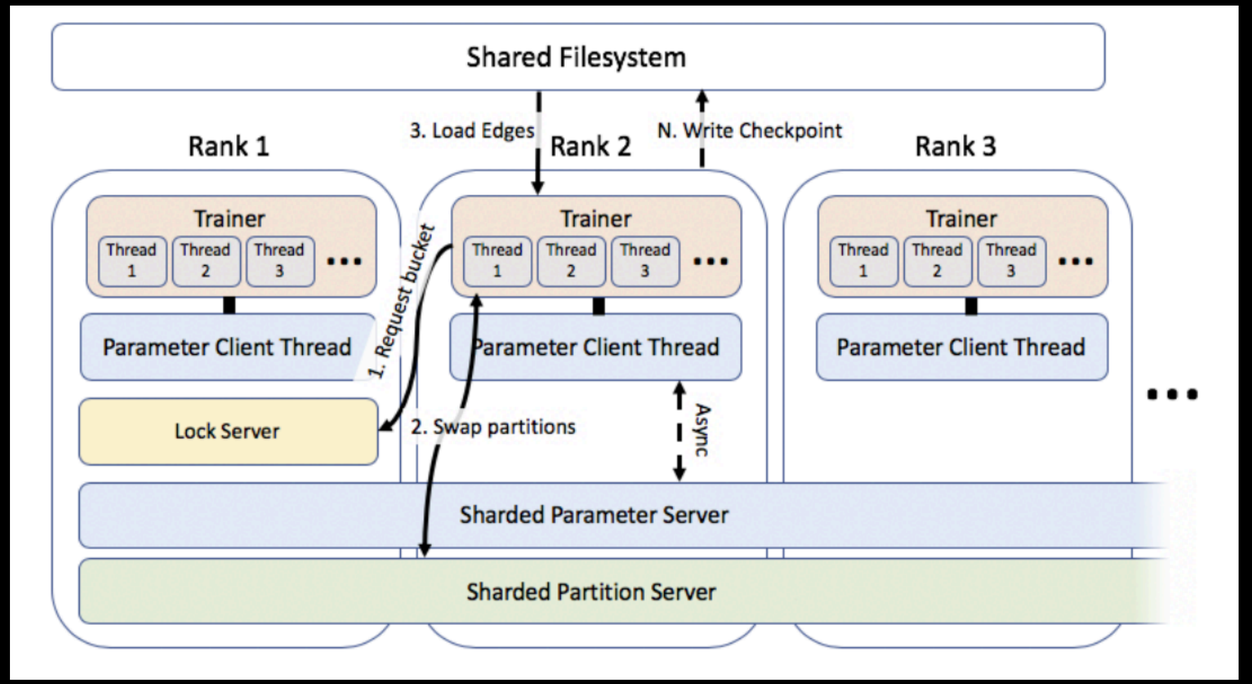
- **Center:** Entity types with small cardinality do not have to be partitioned; if all entity types used for tail nodes are unpartitioned, then edges can be divided into P buckets based only on source node partitions.

- **Right:** the ‘inside-out’ bucket order guarantees that buckets have at least one previously-trained embedding partition. Empirically, this ordering produces better embeddings than other alternatives (or random



BigGraph:

- Rank 2 Trainer performs for the training of one bucket.
- Trainer requests a bucket from **the lock server** on Rank 1, which locks that bucket's partitions.
- The trainer then saves any partitions that it is no longer using and loads new partitions that it needs to and from the sharded partition servers, at which point it can release its old partitions on the lock server.
- Edges are then loaded from a shared filesystem, and training occurs on multiple threads without inter-thread synchronization(Recht et al., 2011).
- In a separate thread, a small number of shared parameters are continuously synchronized with a sharded parameter server. Model checkpoints are occasionally written to the shared filesystem from the trainers.



BigGraph:

- **Pros:**
  - Good powerful engine with high CPU-utilization
  - Easy to use
  - Implicit weights
- **Cons:**
  - Still long for big graphs
  - Graph changes -> we need to re-train it each time
  - Still somy hyper-parameters

- **Personal:**
  - tried but got nothing =(
  - required ~12h on 63 cores for OK graph
  - Restored friends connections really good, not as good in PYMK

Ranking:

- X — set of objects. Exists: $x_i \prec x_j$ — **order** relation.

  Out task to implement ranking function: $x_i \prec x_j \rightarrow a(x_i) \prec a(x_j)$

- Most of times order exists alongside with groupId (query relevant documents search engine)
  if $y(q,d) = \{0,1\}$ — binary relevance, $d_q^{(i)}$ - i-th doc by a(x_i) desc

  Precision: $P_n(q) = 1/n \sum_{i=1}^{n} y(q, d_q^{(i)})$ (percentage of correct in first n)

  Average Precision: $AP(q) = \sum y(d, d_q^{(n)}) P_n(q)) / \sum y(q, d_n^q)$ (average P_n by all relevant docs position)

  Mean average precision: $MAP = 1/|Q| \sum_q AP(x)$

Ranking:

- **DCG** (discounted cumulative gain): $DCG_n(q) = \sum_i^n G_q(d_q^i) D(i)$

  $G_q(d_q^i) = (y(d,q))$ — gain, 1/0 for relevant doc's

  $D(i) = 1/log2(i+1)$ —discounts, higher relevant docs are — better

- **NDCG:** $NDCG(q) = DCG_n(q)/maxDCG_n(q)$ — normalized version

- Also use NDCG averaged over groups.

- Strongly correlates with AUC-per-groupId, but works with ranking objectives (unbounded)

- Also popular:

  $MRR = 1/|Q| \sum_{i=1}^{|Q|} 1/rank_i$

  Where rank_i — refers to the rank position of the *first* relevant document for the *i*-th query.

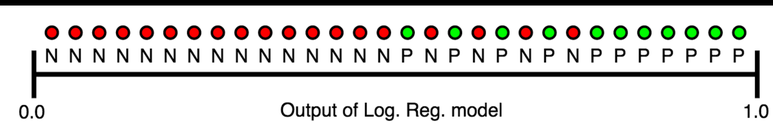


Figure 6. Predictions ranked in ascending order of logistic regression score.

AUC represents the probability that a random positive (green) example is positioned to the right of a random negative (red) example.

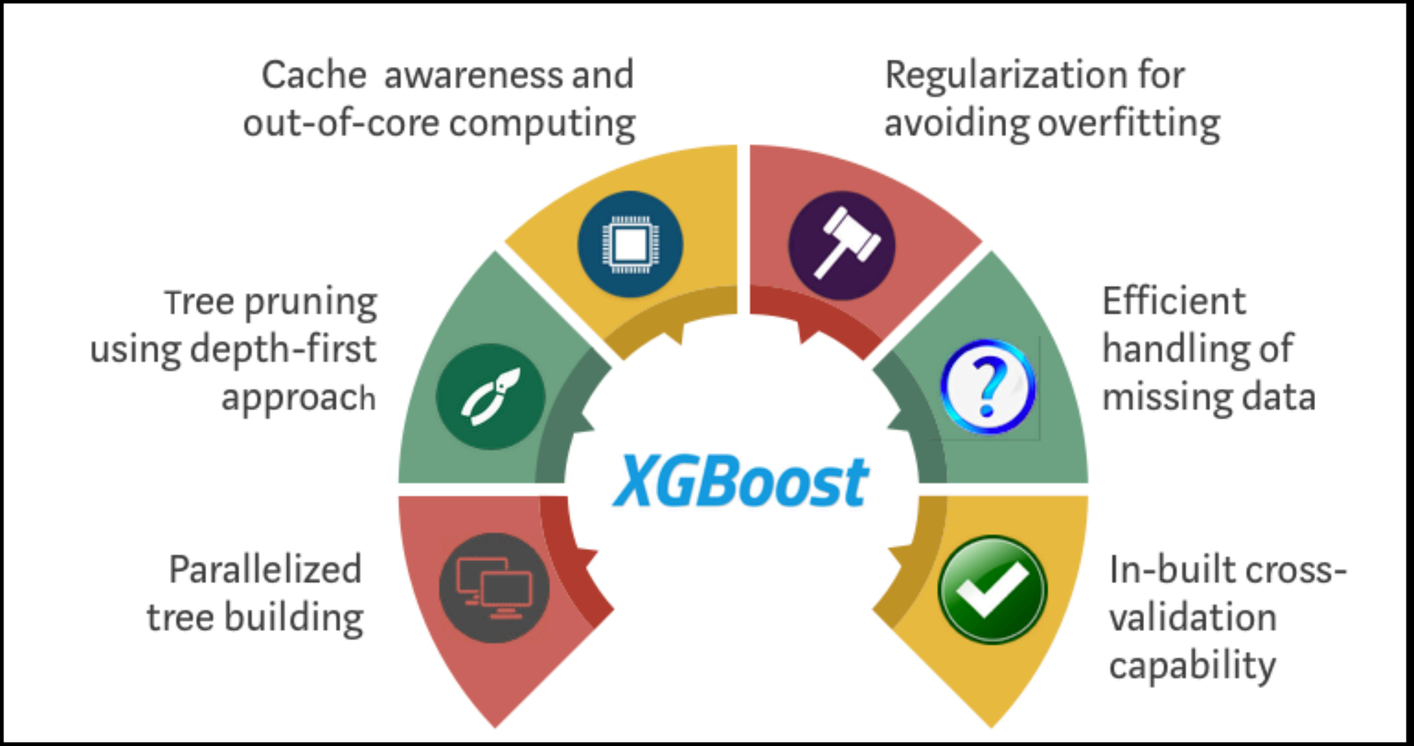
Ranking:

Ontology:

- Point-wise — works
- Pair-wise — works, but not always. Remember uninterpretable scores.
- List-wise — works but only on papers

  LambdaMART — exists!

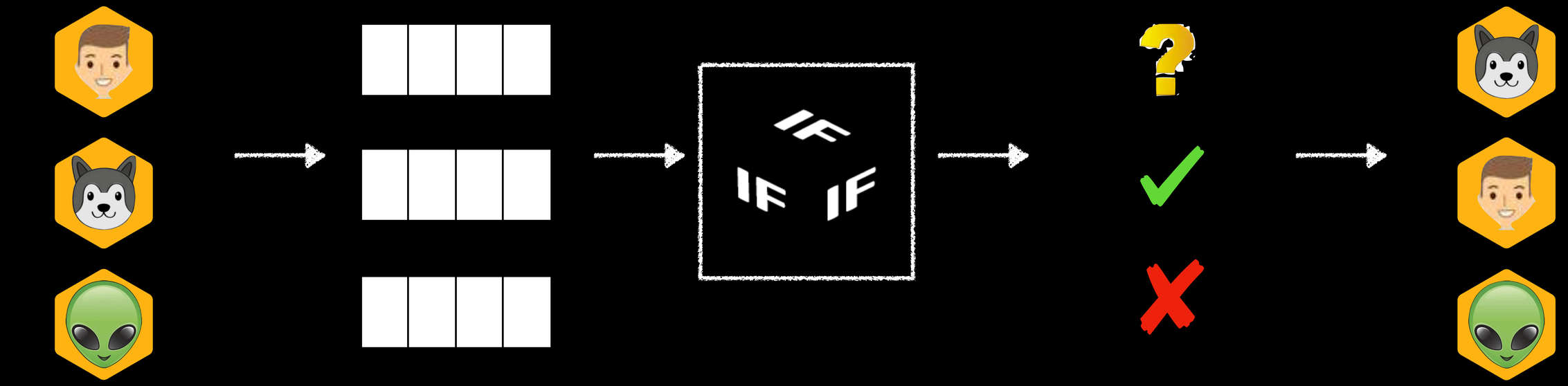
XGBoost:


Cache awareness and out-of-core computing
Regularization for avoiding overfitting
Tree pruning using depth-first approach
Efficient handling of missing data
XGBoost
Parallelized tree building
In-built cross-validation capability

XGBoost notes:

- XGBoost internal feature significance sucks, **use SHAP**.
- XGBoost objectives:
  - **`binary:logistic`**: logistic regression for binary classification, output probability
  - **reg:squarederror**: regression with squared loss. Practically better on classification oO
  - **rank:{pairwise/map/ndcg}:** lambda mart ranking. Sometimes works significantly better then regr losses.
- Hyperparameters matters, tune them wisely (look refs).
- High gamma -> feature selection -> low gamma and features removed

All together:



candidates features ML scores sorted candidates

Features

- User profile features: age/gender/zodiac sign
- User/Candidate matching features: same gender/geo, zodiac matching score.
- Graph features: all about local graph density estimation.

Refs:

- PageRank

https://www.r-bloggers.com/2014/04/from-random-walks-to-personalized-pagerank/

http://statweb.stanford.edu/~tibs/sta306bfiles/pagerank/ryan/01-24-pr.pdf

https://www.dhruvonmath.com/2019/03/20/pagerank/

https://arxiv.org/pdf/1310.7428.pdf

- Facebook:

https://arxiv.org/pdf/1903.12287.pdf

Refs(2):

- XGBooost

https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

https://medium.com/data-design/xgboost-hi-im-gamma-what-can-i-do-for-you-and-the-tuning-of-regularization-a42ea17e6ab6

https://github.com/slundberg/shap

https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

- Link Prediction in Social Networks: the State-of-the-Art
https://arxiv.org/pdf/1411.5118.pdf

- My data science major talk about PYMK
https://www.youtube.com/watch?v=C9_eQKzku1k