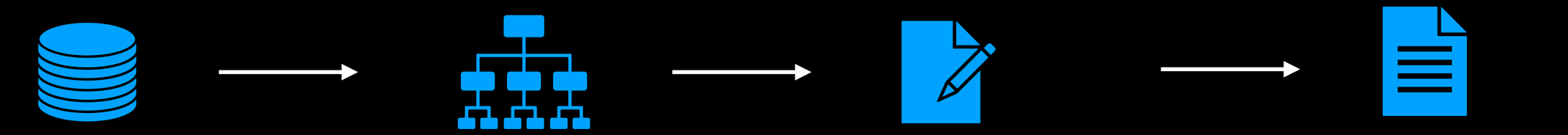
Recommendation Systems

Unclassical embeddings + Learning to **rank**

Eugeny Malyutin / Sergey Dudorov

RecSystem structure



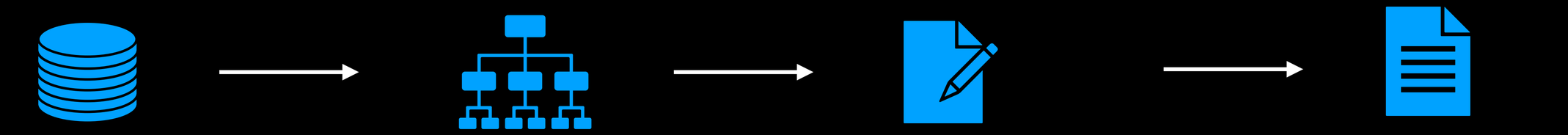
Candidates

Ranking engine

Business processing

Ranked candidates

RecSystem structure



Candidates

Ranking engine

Business processing

Ranked candidates

Ok, ML

How to define it?

- …
- $x \in X = \{x_1, \ldots x_n\}$ — items, $y \in Y = \{y_1, \ldots, y_n\}$ — «labels»
- There are X_n, Y_n — given dataset with «answers», we believe that there is implicit dependency $y^* : X \to Y$
- Need to define algorithm $\alpha : X \to Y$.
- If $Y \in \{0,1\}$ — this task is called binary classification
  If $Y \in \{y_0, y_1 \ldots y_n\}$ — this task is called multi class classification
  If $Y = R$ — this task is called regression

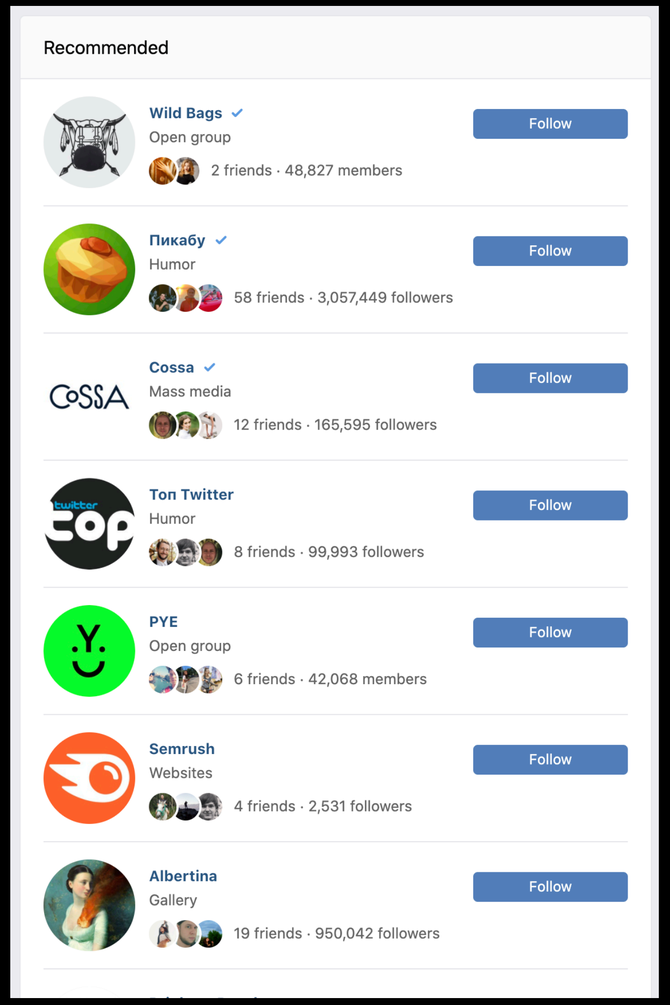
Ok, ML

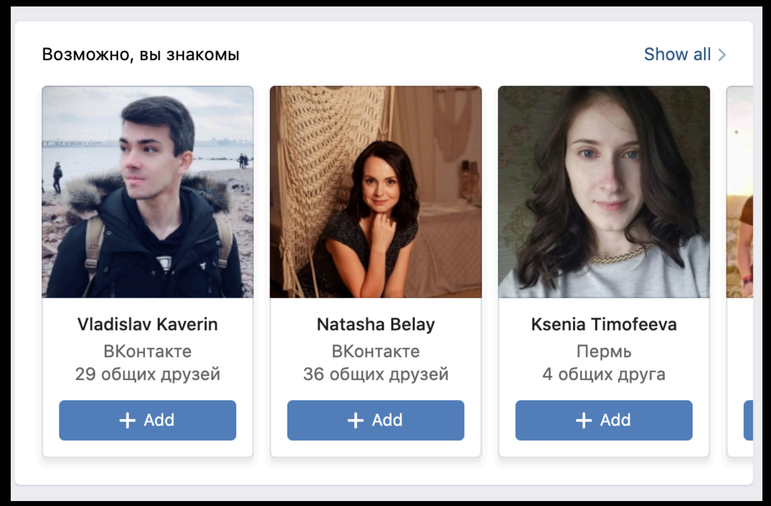
How to define it?

- …
- $x \in X = \{x_1, \ldots x_n\}$ — items, $y \in Y = \{y_1, \ldots, y_n\}$ — «labels»
- There are X_n, Y_n — given dataset with «answers», we believe that there is implicit dependency $y^* : X \to Y$
- Need to define algorithm $\alpha : X \to Y$.
- If $Y \in \{0,1\}$ — this task is called binary classification
  If $Y \in \{y_0, y_1 \ldots y_n\}$ — this task is called multi class classification
  If $Y = R$ — this task is called regression

Ok, binary classification.

Does our cases fits binary classification?





Learning to rank

Definition

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

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

- 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

- Ok, what's do you need more?

- Metrics

- Algorithms

- Tricks

Learning to rank

Metrics

- **Precision:** $P_n(q) = 1/n \sum_{i=1}^{n} y(q, d_q^{(i)})$ (percentage of correct in first n).
- Problems?
- **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)
- Problems?
- **Mean average precision:** $MAP = 1/|Q| \sum_q AP(x)$

Learning to rank

Metrics

- **MRR:** $MRR = 1/|Q| \sum_{i=1}^{|Q|} 1/rank_i$:
  - Sum over relevant!
  - Easy to interpret
- **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/unrelevant 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

Learning to rank

Metrics

- **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/unrelevant 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 NDCG exists w.r.t. groupId, possible to use averaged over groups.

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

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.

Learning to rank

NDCG by example

NDCG - Example

4 documents: d_1, d_2, d_3, d_4

i	Ground Truth		Ranking Function$_1$		Ranking Function$_2$	
	Document Order	r_i	Document Order	r_i	Document Order	r_i
1	d4	2	d3	2	d3	2
2	d3	2	d4	2	d2	1
3	d2	1	d2	1	d4	2
4	d1	0	d1	0	d1	0
	$NDCG_{GT}$=1.00		$NDCG_{RF1}$=1.00		$NDCG_{RF2}$=0.9203	

$$DCG_{GT} = 2 + \left(\frac{2}{\log_2 2} + \frac{1}{\log_2 3} + \frac{0}{\log_2 4} \right) = 4.6309$$

$$DCG_{RF1} = 2 + \left(\frac{2}{\log_2 2} + \frac{1}{\log_2 3} + \frac{0}{\log_2 4} \right) = 4.6309$$

$$DCG_{RF2} = 2 + \left(\frac{1}{\log_2 2} + \frac{2}{\log_2 3} + \frac{0}{\log_2 4} \right) = 4.2619$$

$$MaxDCG = DCG_{GT} = 4.6309$$

Learning to rank

Metrics

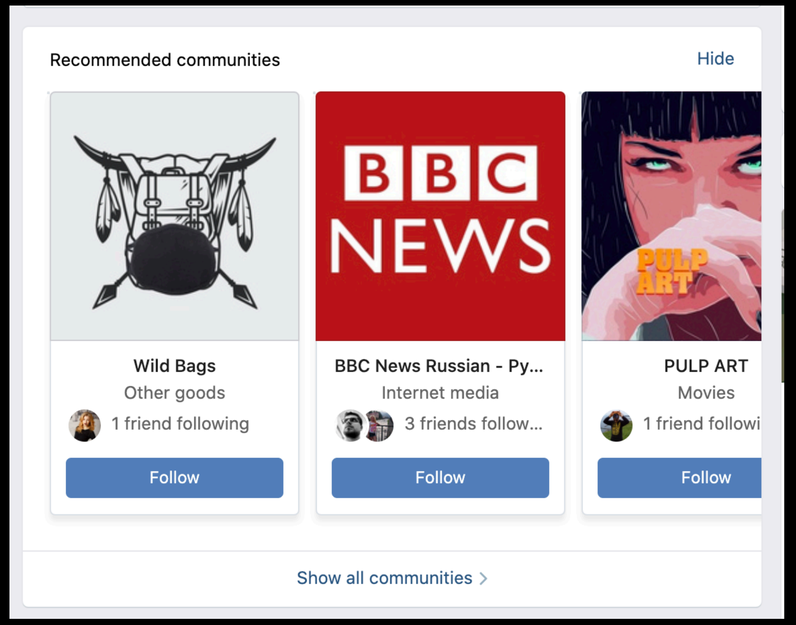
- **DCG** (discounted cumulative gain): $DCG_n(q) = \sum_i^n G_q(d_q^i)D(i)$

  $G_q(d_q^i) = (y(d,q))$ — gain, 1/0 for relevant/unrelevant 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 popular **NDCG-at-k** (k=3, for example)



Learning to rank

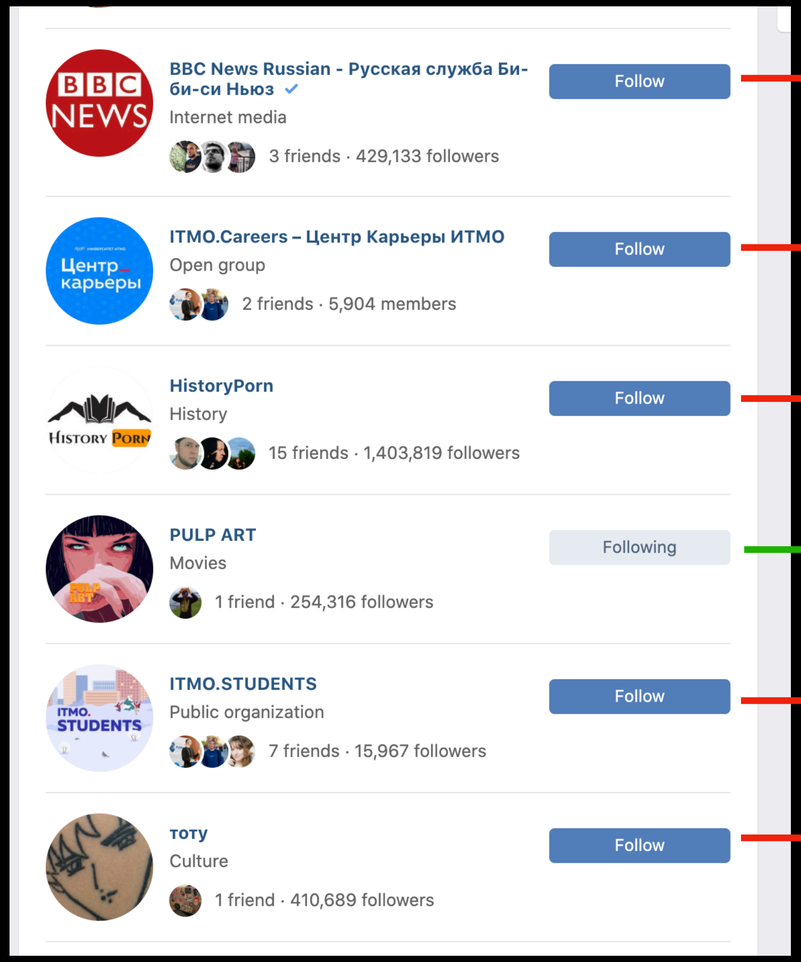
Classification ontology

- **Pointwise:** sorting by score $\alpha(x_i)$ estimating «relevance»
- **Pairwise:** compares objects $a(x_i, x_j) :> 0 \ if \ x_i \prec x_j -$ decide which one is more relevant
- **Listwise:** Takes **the entire list** of candidates and optimise it’s order

Learning to rank

Classification ontology

- **Pointwise:** sorting by score $\alpha(x_i)$ estimating «relevance»

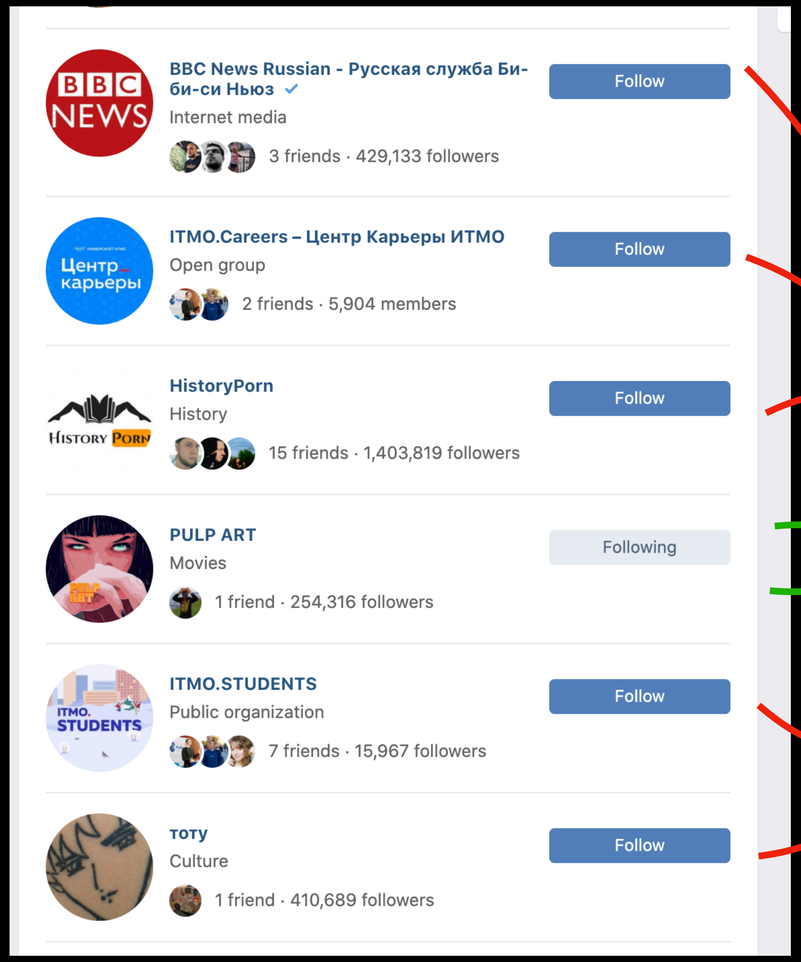


- **Positives** — $y_3 = 1$
- **Negatives** — $y_{0,1,2,4,5} = 0$
- $\alpha(x)$ gives relevance score $\in [0,1]$

Learning to rank

Classification ontology

- **Pairwise:** decide which one is more relevant



Negative

Negative

Negative

Positive

Negative

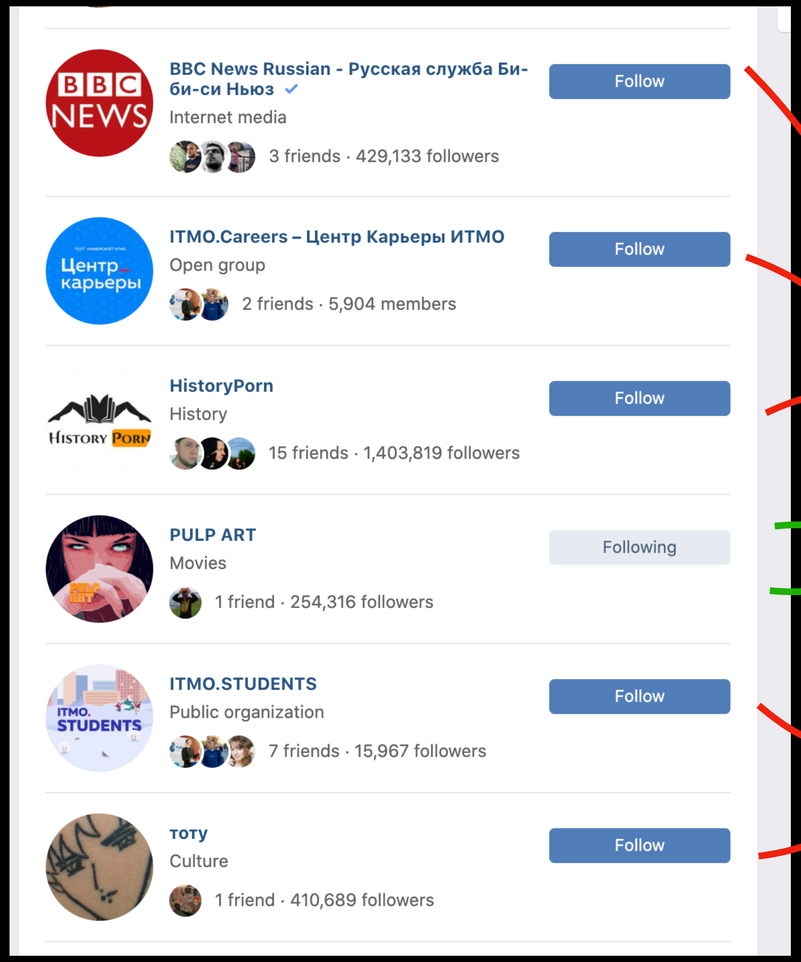
Negative

- **Train set:** $\{x_3 \prec x_0, x_3 \prec x_1, \ldots, x_3 \prec x_5\}$
- $\alpha(x_i, x_j)$ compares pair of objects
- OK, what's a problem?

Learning to rank

Classification ontology

- **Pairwise:** decide which one is more relevant



Negative

Negative

Negative

Positive

Negative

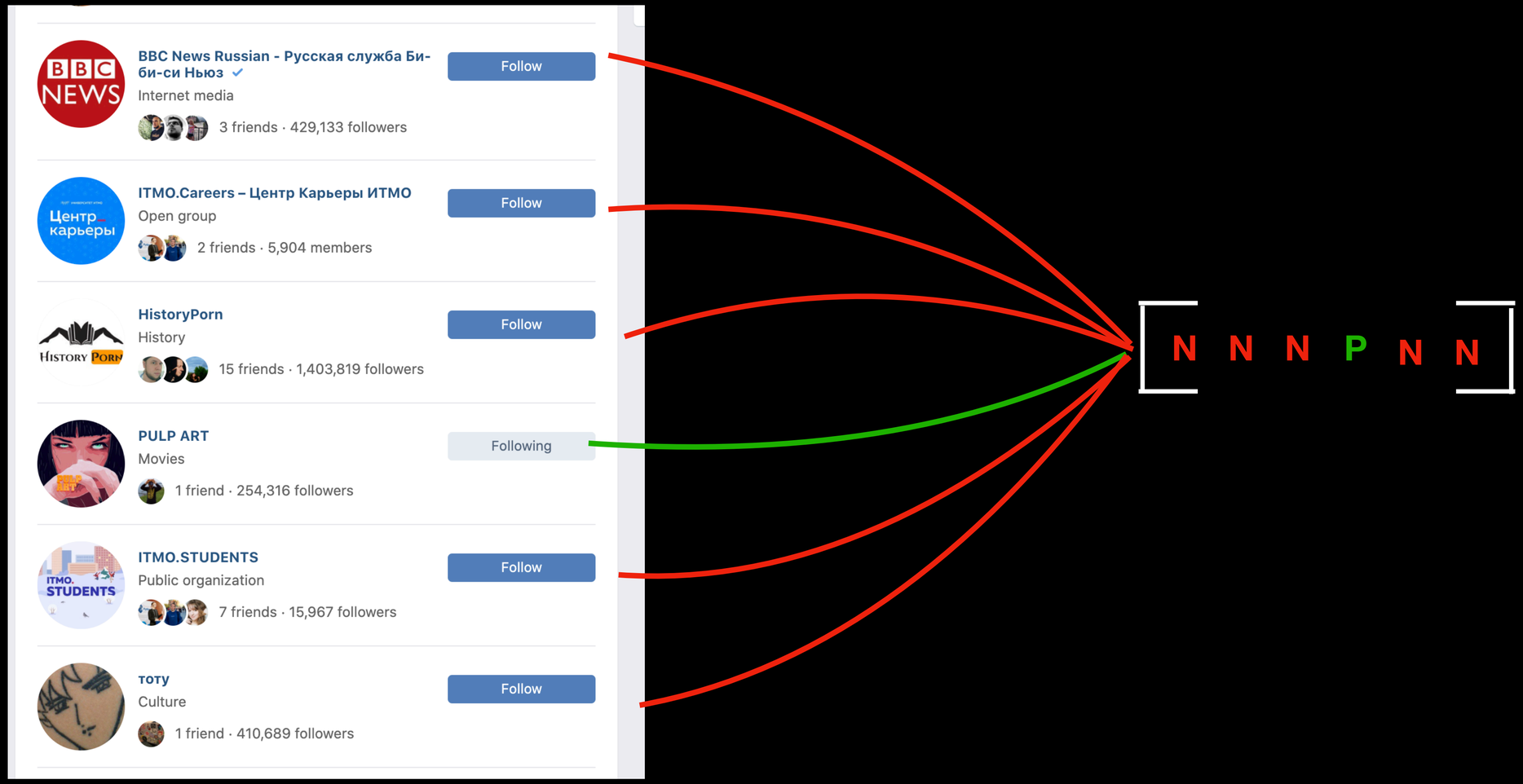
Negative

- **Train set:** $\{x_3 \prec x_0, x_3 \prec x_1, \ldots, x_3 \prec x_5\}$
- $\alpha(x_i, x_j)$ compares pair of objects
- OK, what's a problem?
- How much α applications do you need to sort 10000 - elemnts list?

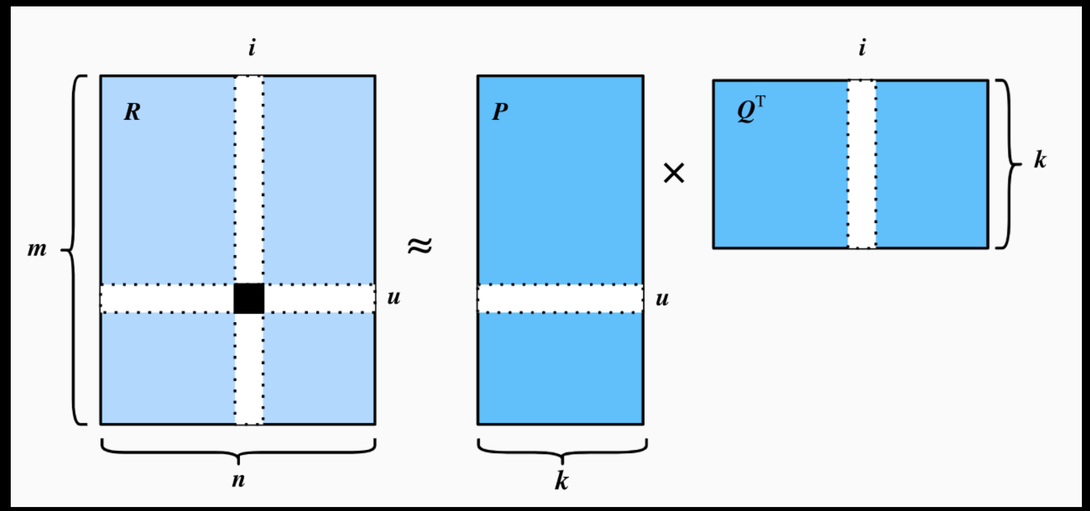
Learning to rank

Classification ontology

- **Listwise:** optimize the whole list



Learning to rank
SVD-factorisation



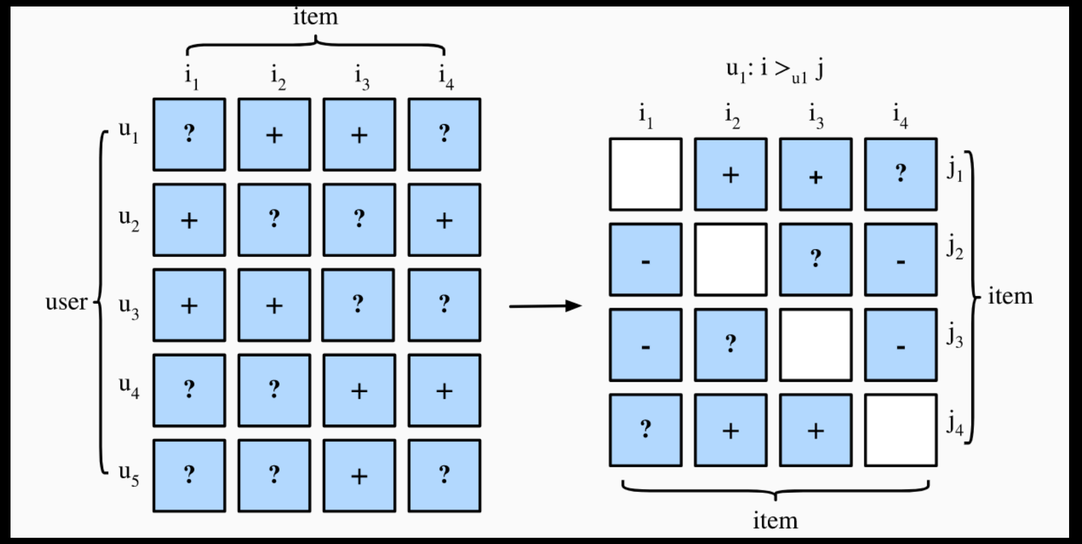
$$argmin_{P,Q,b} \sum_{u,i \in K} ||R_{ui} - \hat{R}_{ui}||^2 + \lambda(||P||^2 + ||Q||^2 + b_u^2 + b_i^2)$$

$$argmin_{p,q,b} \sum_{u,i \in K} (< p_u, q_j > + b_u + b_i + \mu - x_{ij})^2 + \alpha \sum_i ||u_i||^2 + \beta \sum_j ||p_j||^2 + \gamma ||b_u|| + \theta ||b_i||$$

$$\alpha(u, i) = < p_u, q_i > + b_u + b_i + \mu \longrightarrow$$ **Pointwise** algorithm(!)

https://d2l.ai/chapter_recommender-systems/mf.html

Learning to rank
BPR-factorisation



$$p(\Theta \mid >_u) \propto p(>_u \mid \Theta) p(\Theta)$$

BPR-OPT :

$$\begin{aligned} &= \ln p(\Theta \mid >_u) \\ &\propto \ln p(>_u \mid \Theta) p(\Theta) \\ &= \ln \prod_{(u,i,j \in D)} \sigma(\hat{y}_{ui} - \hat{y}_{uj}) p(\Theta) \\ &= \sum_{(u,i,j \in D)} \ln \sigma(\hat{y}_{ui} - \hat{y}_{uj}) + \ln p(\Theta) \\ &= \sum_{(u,i,j \in D)} \ln \sigma(\hat{y}_{ui} - \hat{y}_{uj}) - \lambda_\Theta \|\Theta\|^2 \end{aligned}$$

$$r_{ui} = < e_u, e_i > + b_u + b_i + \mu$$

$$\alpha(x_i, x_j, u) = \ln(\sigma(< e_u, e_i > + b_u + b_i - < e_u, e_j > - b_u - b_j))$$

$$\alpha(u, i) = < e_u, e_i >$$

Sort with α

PAIRwise algorithm while learn
POINTwise algorithm on runtime!

https://d2l.ai/chapter_recommender-systems/mf.html

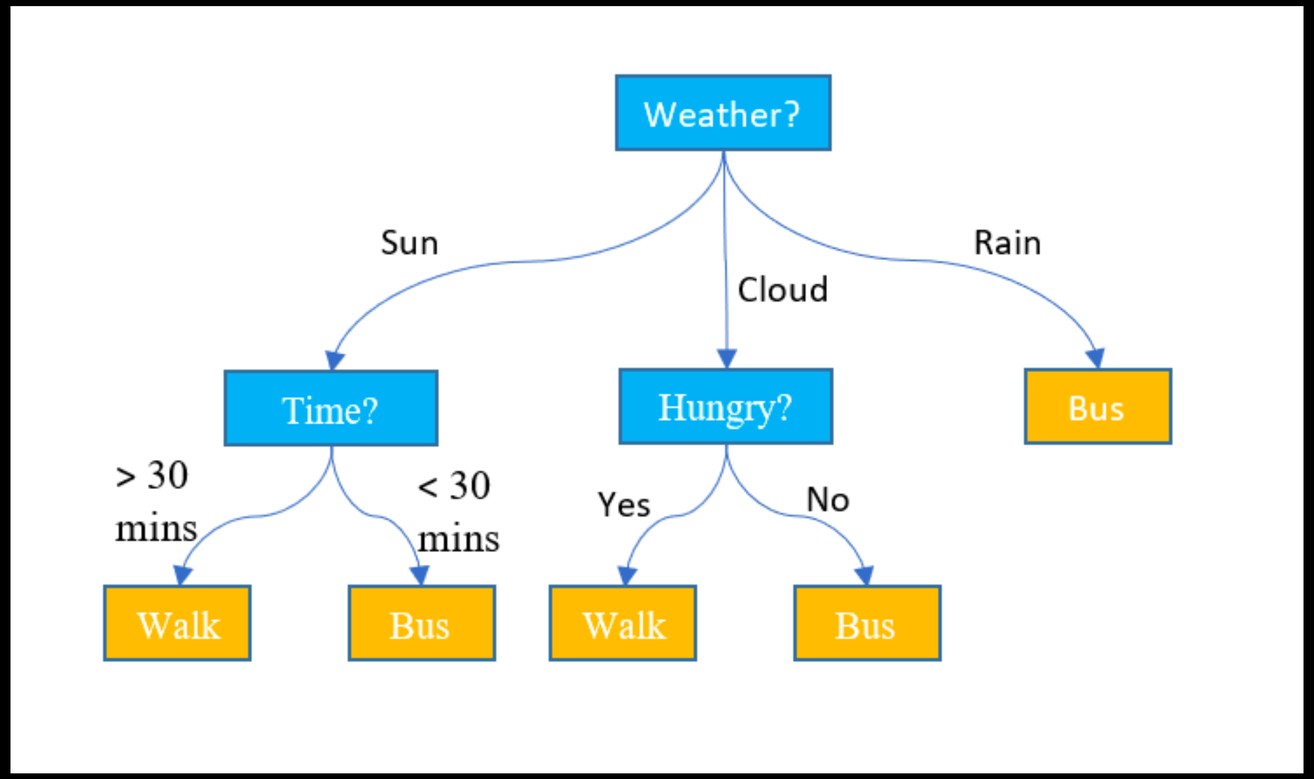
Learning to rank

Summary

- Ranking **differs** either classification or regression due to strict **order properties** on **relevance**
- **It** leads us to new metrics: MAP, MRR, AUC(~), NDCG
- Algorithms ontology: **point**-, **pair**-, **list**-wise
- There are **no** working «fair» **listwise** algorithms **IRL** (at least author saw no one)
- **Best pairwise** algorithms — camouflages themselves as **pointwise**.

Ranking algorithms

Tree reminder



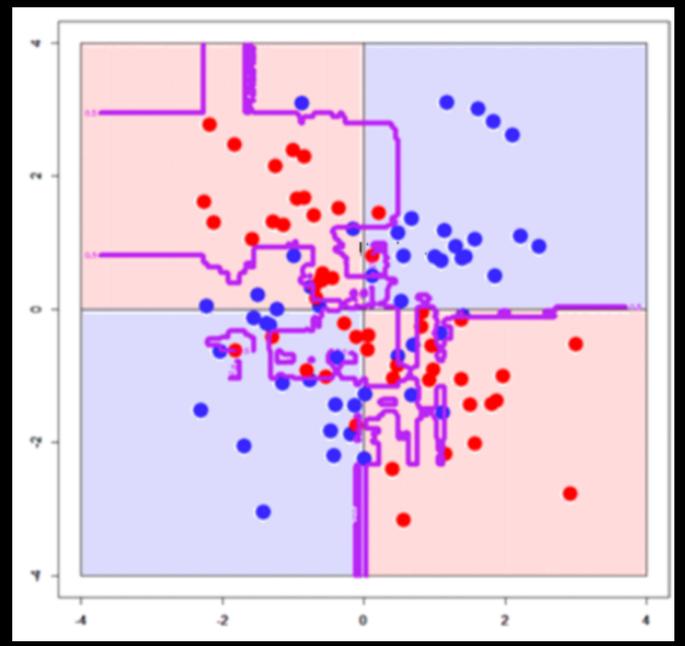
Ranking algorithms

Splitting criteria:

- **Split:** $[x_j < t]$
- **Loss criteria:** $Q(X_m, j, t) = \frac{|X_l|}{X_M} H(X_l) + \frac{|X_r|}{|X_m|} H(X_m)$
- **Information criteria:**
  - **Regression:** H(x) = $1/|X_m| \sum (y_i - \bar{y}(X))^2$
  - **Gini**: $p_k = \frac{1}{|X|} \sum_{i \in X} [y_i = k]$ and $H(x) = \sum_{k=1} p_k(1 - p_k)$
  - **Entropy:** $H(X) = \sum_{k=1}^{K} p_k ln(p_k)$

Ranking algorithms

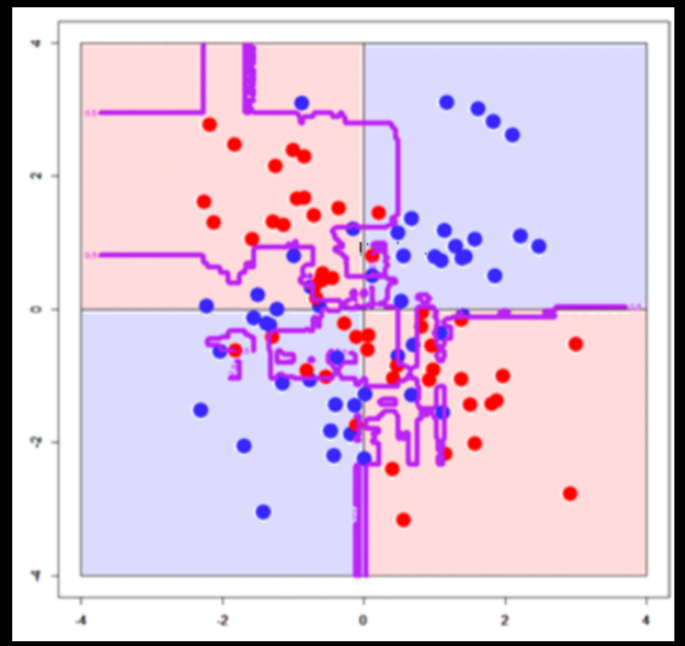
What's went wrong?



Ranking algorithms

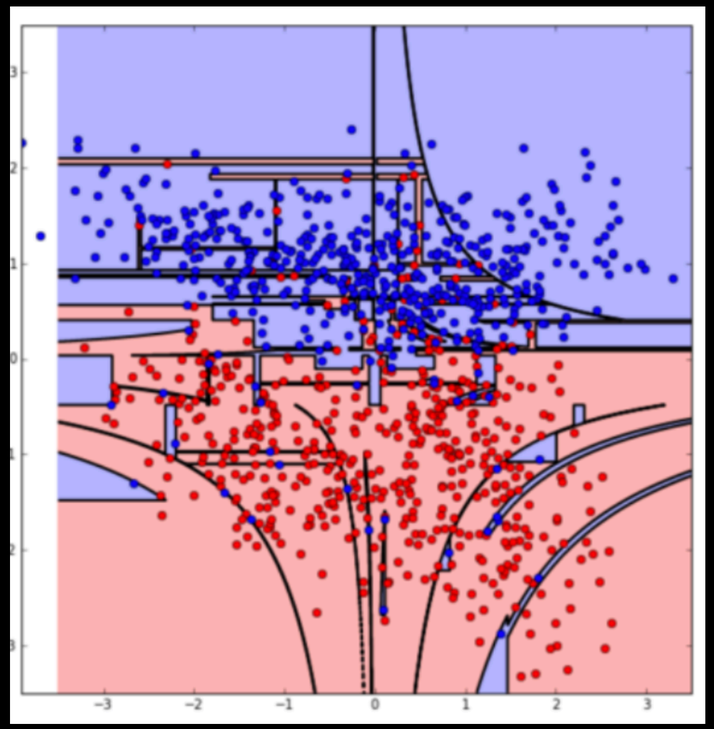
What's went wrong?

- Stop when:
  - Max length
  - Min elements
  - Pruning

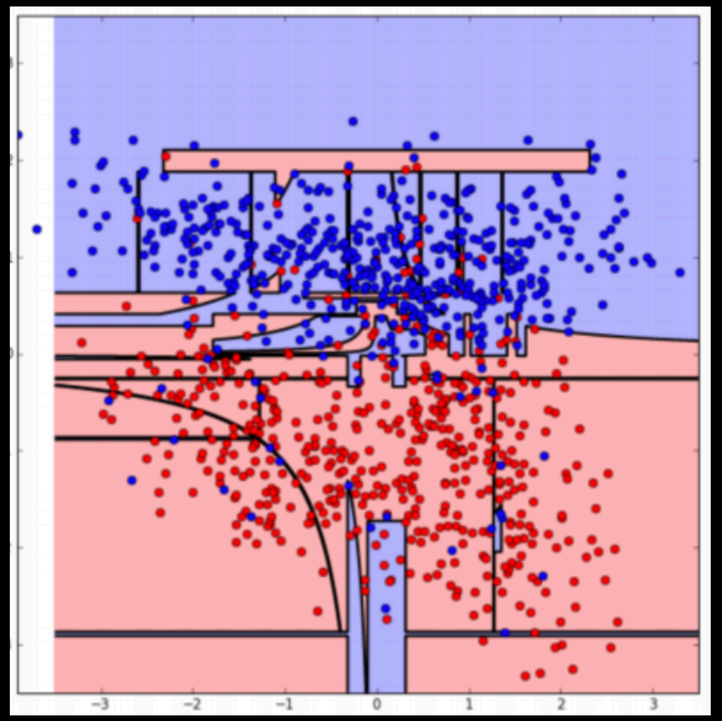


Ranking algorithms

What's went wrong?



Overfitted



Overfitted and unstable

Ranking algorithms

Error decomposition

- What is?

Ranking algorithms

Error decomposition

- **Error**:
  - **Noise:** world's imperfectness measure, exists even for ideal model on ideal data
  - **Bias:** deviation of concrete model from ideal one (averaged over all data sets)
  - **Variance:** dispersion of models answers caused by different datasets

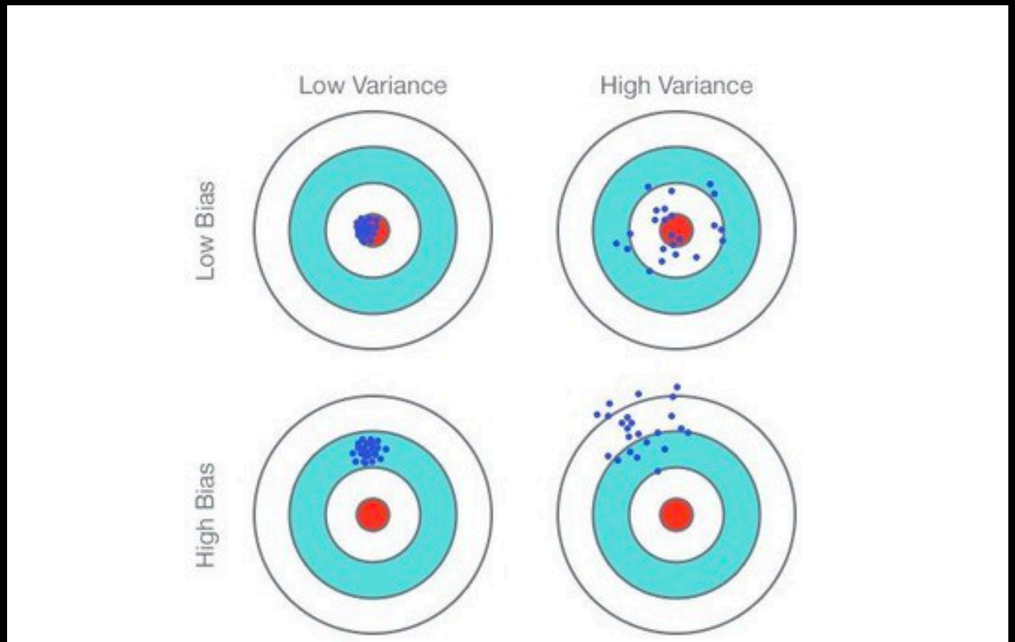


Fig. 1: Graphical Illustration of bias-variance trade-off , Source: Scott Fortmann-Roe., Understanding Bias-Variance Trade-off

Ranking algorithms

Composition idea

- Takes N different trees
- **Averages** answers:
  - **Regression:** $a(x) = \sum_{1}^{N} b_n(x)$
  - **Classification:** most popular answer
- **PRBLMS?**

Ranking algorithms

Error decomposition

- **Trees:**
  - High variance
  - Low bias
- **Linear algorithms:**
  - Low variance
  - High bias

- **Composition:**
  - Same bias
  - Variance = 1/N (algorithm_variance) + **correlation**
  - Decrease variance **N times** in case of **independent algorithms** — that's why **we need randomisation**

Ranking algorithms

Randomisation

- **Bagging (bootstrap):** learn algorithms on random subsamples from train set. Less subsamples — more randomised trees.
- **Random sub-spaces**: random sub-set of features for each tree
- **Extreme randomised trees:** choose random subset of features on each split

Ranking algorithms

Summary

- **Cons:**
  - Powerful
  - Easy to parallel
  - With bootstrap 1/3 of data samples (for concrete tree) wasn’t in train set — we can estimate test metrics on them (Out-of-bag score)
- **Cons:**
  - More trees -> more computational resources
  - Undirected search

Ranking algorithms

Boosting idea

- $b_0(x)$ – initial algorithm (zero, mean class, mean value)
- $a_m(x) = \sum_{i=1}^{m} b_i(x)$ – step of composition
- $F = \sum_{i=1}^{N} L(y_i, a_{m-1}(x_i) + b_i(x_i)) \to min_b$ – optimisation
- $s = (s_1, s_2 \dots s_N)$ – displacements vector then
  $F = \sum_{i=1}^{N} L(y_i, a_{m-1}(x_i) + s_i) \to min_s$
- Optimal shift $-\nabla F = [\frac{dF}{da_{m-1}}(x_i)]_{i=1}^{N} = [\frac{\sum_{i=1}^{N} L(y_i, a_{m-1}(x_i))}{da_{m-1}}]_{i=1}^{N} = [\frac{dL(y_i, a_{m-1})}{da_{m-1}} x_i]_{i=1}^{N}$

Ranking algorithms

Boosting idea

- $a_m(x) = \sum_{i=1}^{m} b_i(x)$ – step of composition, $F = \sum_{i=1}^{N} L(y_i, a_{m-1}(x_i) + b_i(x_i)) \to min_b$ – optimisation
- Optimal shift

$$s_i = -\nabla F = [\frac{dF}{da_{m-1}}(x_i)]_{i=1}^{N} = [\frac{\sum_{i=1}^{N} L(y_i, a_{m-1}(x_i))}{da_{m-1}}(x_i)]_{i=1}^{N} = [\frac{dL(y_i, a_{m-1})}{da_{m-1}}(x_i)]_{i=1}^{N}$$

- But wait, s_i is not an algorithm, it's a vector of numbers!
- Yep, but we can learn out algorithm $b_m(x_i) \to s_i$

http://www.machinelearning.ru/wiki/images/9/9a/fonarev.overview_of_boosting_methods.pdf

Ranking algorithms

Summary

- Powerful (extremely powerful)
- Allows all this tricks of gradient algorithms: learning rate, decay, …
- Hard to interpret (~)
- Works mostly with weak algorithms
- Allows you to set various range of functions(?) as loss

http://www.machinelearning.ru/wiki/images/9/9a/fonarev.overview_of_boosting_methods.pdf

XGboost

XGBoost

$$\hat{y}_i^{(0)} = 0$$
$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$
$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$
$$\cdots$$
$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

Composition structure

$$\text{obj}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$
$$= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \text{constant}$$

Objective function

$\Omega(f_i)$ **— regularisation**

https://xgboost.readthedocs.io/en/latest/tutorials/model.html

NB! — xgboost about regression trees only,
but possible to solve classification

XGboost

XGBoost

$$\text{obj}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$
$$= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \text{constant}$$

Objective function

$$\text{obj}^{(t)} = \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \sum_{i=1}^{t} \Omega(f_i)$$
$$= \sum_{i=1}^{n} [2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + \text{constant}$$

MSE example

$$\text{obj}^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + \text{constant}$$

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$
$$h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

2nd order Taylor's approximation

https://xgboost.readthedocs.io/en/latest/tutorials/model.html

XGboost

Regularisation

$$f_t(x) = w_{q(x)}, w \in R^T, q : R^d \to \{1,2,\cdots,T\} .$$

Formal tree

- w — vector of scores on j-th index
- q — function to assign object to leaf
- T — number of leaves

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Complexity

https://xgboost.readthedocs.io/en/latest/tutorials/model.html

https://stats.stackexchange.com/questions/202858/xgboost-loss-function-approximation-with-taylor-expansion

XGboost

Regularisation in learning

$$\text{obj}^{(t)} \approx \sum_{i=1}^{n} [g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

$$= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T$$

Obj function with Taylor and formal tree

$$\text{obj}^{(t)} = \sum_{j=1}^{T} [G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2] + \gamma T$$

$$G_j = \sum_{i \in I_j} g_i \qquad H_j = \sum_{i \in I_j} h_i$$

«Simplification»

https://xgboost.readthedocs.io/en/latest/tutorials/model.html

https://stats.stackexchange.com/questions/202858/xgboost-loss-function-approximation-with-taylor-expansion

XGboost

...

$$\text{obj}^{(t)} \approx \sum_{i=1}^{n} [g_i w_{q(x_i)} + \frac{1}{2} h_i w^2_{q(x_i)}] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

$$= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T$$

Obj function with Taylor and formal tree

$$\text{obj}^{(t)} = \sum_{j=1}^{T} [G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2] + \gamma T$$

$$G_j = \sum_{i \in I_j} g_i \qquad H_j = \sum_{i \in I_j} h_i$$

«Simplification»

https://xgboost.readthedocs.io/en/latest/tutorials/model.html

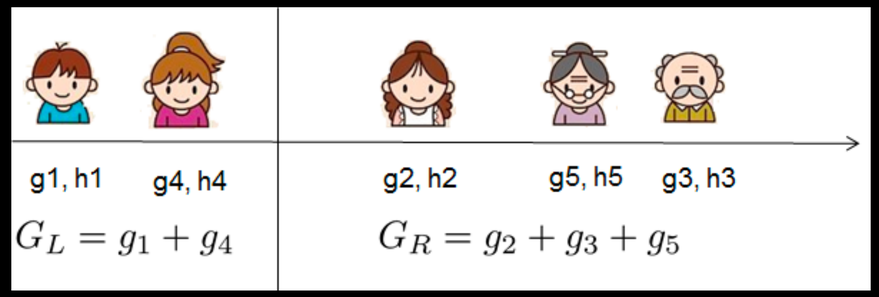
https://stats.stackexchange.com/questions/202858/xgboost-loss-function-approximation-with-taylor-expansion

XGboost:

Sorting schema:

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

«Pruning»



https://xgboost.readthedocs.io/en/latest/tutorials/model.html

https://stats.stackexchange.com/questions/202858/xgboost-loss-function-approximation-with-taylor-expansion

XGboost

Summing up:

- Decomposes loss-function in Taylor row
- Allows arbitrary set of loss functions
- Zip regularisation inside learning process
- Scan all dataset ones to find best split

https://xgboost.readthedocs.io/en/latest/tutorials/model.html

https://stats.stackexchange.com/questions/202858/xgboost-loss-function-approximation-with-taylor-expansion

XGboost

And his friends:

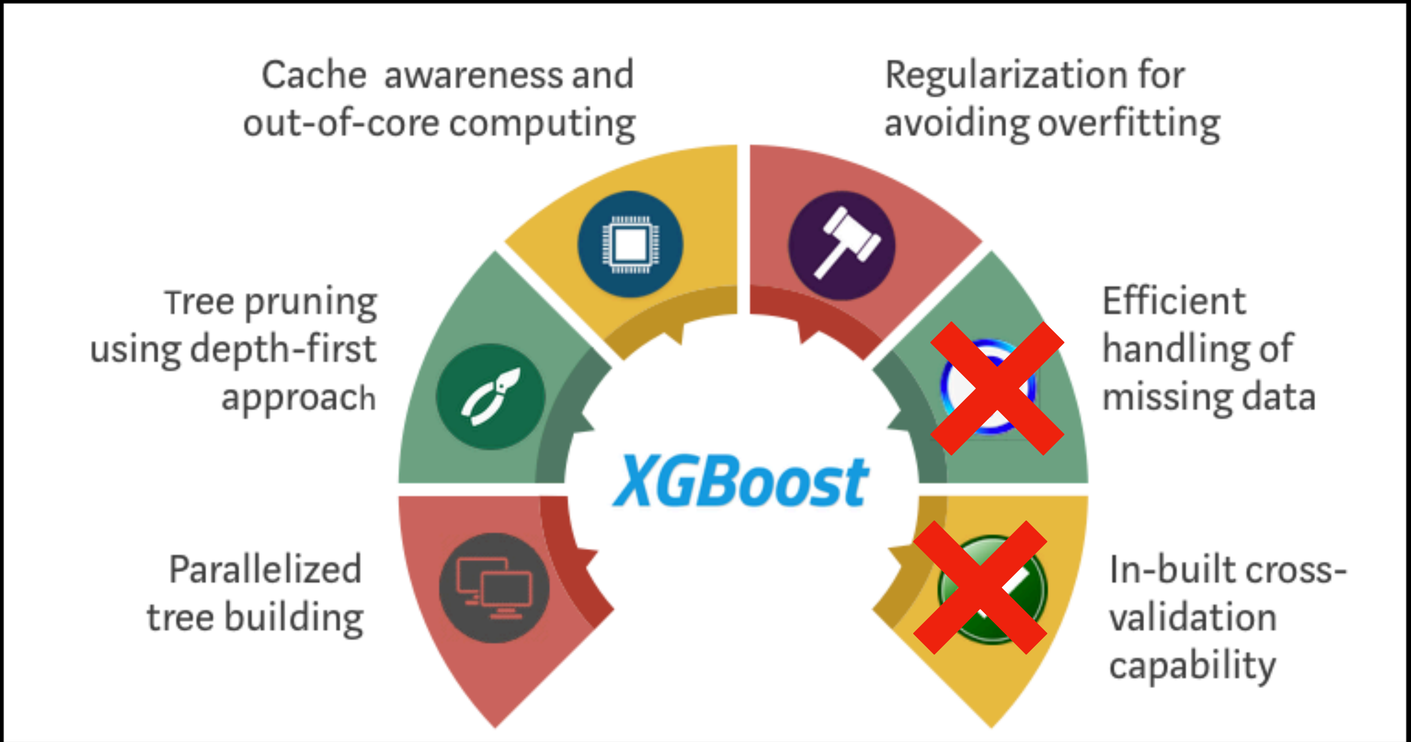
Pros:

- Well-known (est. 2016)
- Available on ton different platforms (as cpp binding, mostly)
- Allows distribute training (spark, for example)

Cons:

- Not so fancy comparing to IGBM, catBoost
- Not so powerful (~)
- Very strange sparsity

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 values**.
- 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

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

Learning to rank

«Lambda»-smth

$$w = w + \eta \frac{\sigma}{1 + exp(\sigma \langle x_i - x_j, w \rangle)} (x_i - x_j)$$

Gradient step optimising logit function on linear algorithm

$$w = w + \eta \frac{\sigma}{1 + exp(\sigma \langle x_i - x_j, w \rangle)} |\Delta NDCG_{ij}| (x_i - x_j)$$

NDCG delta
by changing x_i and x_j

Gradient step optimising NDCG function on linear algorithm

https://www.microsoft.com/en-us/research/uploads/prod/2016/02/MSR-TR-2010-82.pdf

Learning to rank

Summary

- Learning to rank - specific task with specific metrics and tricks.
- There are: point- pair- and list-wise algorithms.
- Best pair-wise algorithms looks as pointwise at runtime.
- There are tons ranking algorithms. Most powerful and/or used — GBDT
- XGBoost — allows arbitrary set of loss functions, encorporates regularisation into training, use Taylor 2nd order to estimate loss function.
- It’s possible to optimise ranking metrics directly using lambda- techniques

Vector embeddings

Eugeny Malyutin

Previously:

motel [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0] AND
hotel [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0] = 0

Поправь doc frequency Малютин

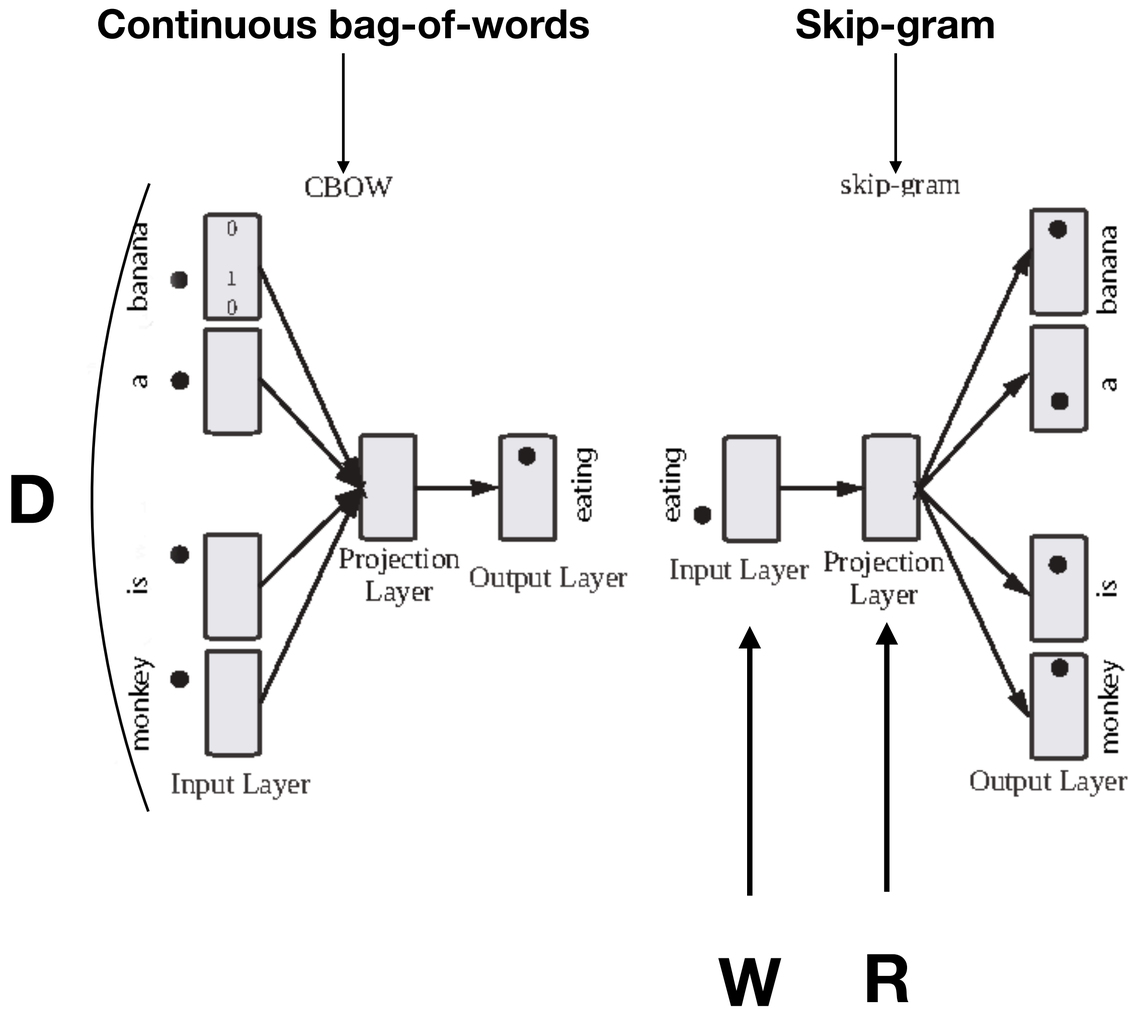
$$TF - IDF(w, d, C) = \frac{count(w, d)}{count(d)} * log(\frac{\sum_{d' \in C} 1(w, d')}{|C|})$$

One-hot encoding drawback:

- “monkeys eat bananas” and “apes consume fruits” - similarity equals to 0
- «Pouteria is widespread throughout the tropical regions of the world and monkeys eat their fruits»(c). What is Pouteria? Is it a tree?
- «a word is characterized by the company it keeps» – John Rupert Firth
- Ideally, we want vector representations where ***similar words*** end up with ***similar vectors***. **Dense** vectors. And when I say similar a mention some **similarity measure** (cosine).
- Even better, we’d want more similar representations when the words share some properties such as if they’re both plural or singular, verbs or adjectives or if they both reference to a male.

Word2vec scheme:

- 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.



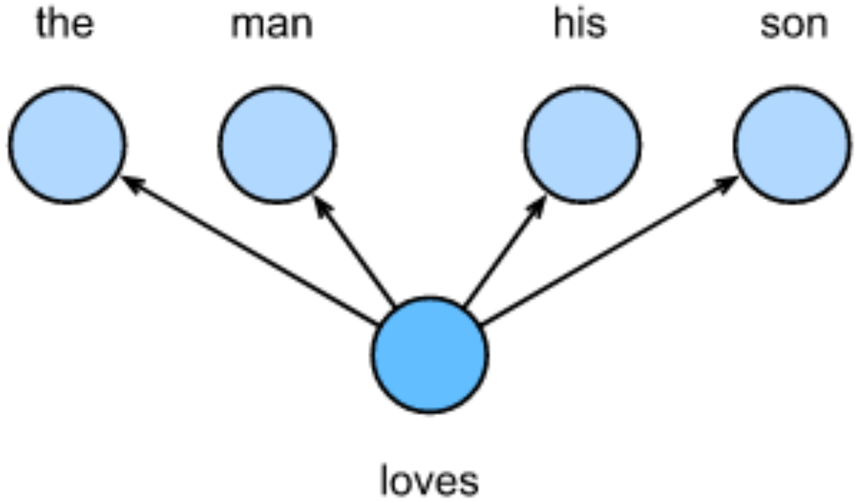
The skipgram model

- We assume that, given the central target word, the context words are generated independently of each other.
  P(the, man, his, son | loves) = P(the | loves) * P(man | loves) * P(his | loves) * P(son | loves)
- And $p(w_o \mid w_c) = \frac{exp(u_o^T v_c)}{\sum_{i \in V} exp(u_i^T v_c)}$ **cond. probability**, u and v — vector representations.

  u_0 - context,
  v_c — central target.

- The **likelihood function** of the skip-gram model:

$$\prod_{i=1}^{T} \prod_{-m \le j \le m} P(w^{(t+j)} \mid w^t)$$



Skipgram model training

- Loss function $-\sum_{t=1}^{T} \sum_{-m \le j \le m,\ j \neq 0} \log \mathbb{P}(w^{(t+j)} \mid w^{(t)})$

- If we want to SGD it - we need to compute gradient of conditional probability:

- Through differentiation, we can get the gradient from the formula above.

- Any problems?

$$\log \mathbb{P}(w_o \mid w_c) = \mathbf{u}_o^\top \mathbf{v}_c - \log \left(\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c) \right)$$

$$\begin{aligned} \frac{\partial \log \mathbb{P}(w_o \mid w_c)}{\partial \mathbf{v}_c} &= \mathbf{u}_o - \frac{\sum_{j \in \mathcal{V}} \exp(\mathbf{u}_j^\top \mathbf{v}_c) \mathbf{u}_j}{\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c)} \\ &= \mathbf{u}_o - \sum_{j \in \mathcal{V}} \left(\frac{\exp(\mathbf{u}_j^\top \mathbf{v}_c)}{\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c)} \right) \mathbf{u}_j \\ &= \mathbf{u}_o - \sum_{j \in \mathcal{V}} \mathbb{P}(w_j \mid w_c) \mathbf{u}_j \,. \end{aligned}$$

Skipgram model training

- Loss function $-\sum_{t=1}^{T} \sum_{-m \le j \le m,\ j \neq 0} \log \mathbb{P}(w^{(t+j)} \mid w^{(t)})$

- If we want to SGD it - we need to compute gradient of conditional probability:

$$\log \mathbb{P}(w_o \mid w_c) = \mathbf{u}_o^\top \mathbf{v}_c - \log \left(\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c) \right)$$

- Through differentiation, we can get the gradient from the formula above:

$$\begin{aligned} \frac{\partial \log \mathbb{P}(w_o \mid w_c)}{\partial \mathbf{v}_c} &= \mathbf{u}_o - \frac{\sum_{j \in \mathcal{V}} \exp(\mathbf{u}_j^\top \mathbf{v}_c) \mathbf{u}_j}{\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c)} \\ &= \mathbf{u}_o - \sum_{j \in \mathcal{V}} \left(\frac{\exp(\mathbf{u}_j^\top \mathbf{v}_c)}{\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c)} \right) \mathbf{u}_j \\ &= \mathbf{u}_o - \sum_{j \in \mathcal{V}} \mathbb{P}(w_j \mid w_c) \mathbf{u}_j \,. \end{aligned}$$

- Its computation obtains the conditional probability for **all the words in the dictionary** given the central target word w_c
  We then use **the same method** to obtain the gradients **for other word vectors**.

Negative sampling:

- Given a context window for the central target word w_c, we will treat it as an event for context word w_o to appear in the context window and compute the probability of this event from

$$\mathbb{P}(D = 1 \mid w_c, w_o) = \sigma(\mathbf{u}_o^\top \mathbf{v}_c),$$

- Now we consider maximizing the joint probability $\prod_{t=1}^{T} \prod_{-m \le j \le m,\ j \neq 0} \mathbb{P}(D = 1 \mid w^{(t)}, w^{(t+j)})$.
- However, the events included in the model only consider positive examples. We need to sample additional negative events (never occurred in the same context) and then:

$$\mathbb{P}(w^{(t+j)} \mid w^{(t)}) = \mathbb{P}(D = 1 \mid w^{(t)}, w^{(t+j)}) \prod_{k=1,\ w_k \sim \mathbb{P}(w)}^{K} \mathbb{P}(D = 0 \mid w^{(t)}, w_k).$$

Negative sampling:

- Now we consider maximizing the joint probability $\prod_{t=1}^{T} \prod_{-m \le j \le m,\ j \neq 0} \mathbb{P}(D = 1 \mid w^{(t)}, w^{(t+j)})$.

- However, the events included in the model only consider positive examples. We need to sample additional negative K events (never occurred in the same context) and then:

$$\mathbb{P}(w^{(t+j)} \mid w^{(t)}) = \mathbb{P}(D = 1 \mid w^{(t)}, w^{(t+j)}) \prod_{k=1,\ w_k \sim \mathbb{P}(w)}^{K} \mathbb{P}(D = 0 \mid w^{(t)}, w_k).$$

- The logarithmic loss for the conditional probability above is

$$-\log \mathbb{P}(w^{(t+j)} \mid w^{(t)}) = -\log \mathbb{P}(D = 1 \mid w^{(t)}, w^{(t+j)}) - \sum_{k=1,\ w_k \sim \mathbb{P}(w)}^{K} \log \mathbb{P}(D = 0 \mid w^{(t)}, w_k)$$

- Here, the gradient computation in each step of the training is no longer related to the dictionary size, but linearly related to K

$$= -\log \sigma\left(\mathbf{u}_{i_{t+j}}^\top \mathbf{v}_{i_t}\right) - \sum_{k=1,\ w_k \sim \mathbb{P}(w)}^{K} \log\left(1 - \sigma\left(\mathbf{u}_{h_k}^\top \mathbf{v}_{i_t}\right)\right)$$

$$= -\log \sigma\left(\mathbf{u}_{i_{t+j}}^\top \mathbf{v}_{i_t}\right) - \sum_{k=1,\ w_k \sim \mathbb{P}(w)}^{K} \log \sigma\left(-\mathbf{u}_{h_k}^\top \mathbf{v}_{i_t}\right).$$

Negative sampling:

- The logarithmic loss for the conditional probability above is

$$-\log \mathbb{P}(w^{(t+j)} \mid w^{(t)}) = -\log \mathbb{P}(D = 1 \mid w^{(t)}, w^{(t+j)}) - \sum_{k=1,\ w_k \sim \mathbb{P}(w)}^{K} \log \mathbb{P}(D = 0 \mid w^{(t)}, w_k)$$

$$= -\log\ \sigma\left(\mathbf{u}_{i_{t+j}}^\top \mathbf{v}_{i_t}\right) - \sum_{k=1,\ w_k \sim \mathbb{P}(w)}^{K} \log\left(1 - \sigma\left(\mathbf{u}_{h_k}^\top \mathbf{v}_{i_t}\right)\right)$$

$$= -\log\ \sigma\left(\mathbf{u}_{i_{t+j}}^\top \mathbf{v}_{i_t}\right) - \sum_{k=1,\ w_k \sim \mathbb{P}(w)}^{K} \log \sigma\left(-\mathbf{u}_{h_k}^\top \mathbf{v}_{i_t}\right).$$

- Here, the gradient computation in each step of the training is no longer related to the dictionary size, but linearly related to K

- **Key idea:** sample additional negatives and learn your positive probabilities «opposite to» them.

So what? (Synonyms)

```
get_similar_tokens('chip', 3, glove_6b50d)
```

```
cosine sim=0.856: chips
cosine sim=0.749: intel
cosine sim=0.749: electronics
```

```
get_similar_tokens('baby', 3, glove_6b50d)
```

```
cosine sim=0.839: babies
cosine sim=0.800: boy
cosine sim=0.792: girl
```

```
get_similar_tokens('beautiful', 3, glove_6b50d)
```

```
cosine sim=0.921: lovely
cosine sim=0.893: gorgeous
cosine sim=0.830: wonderful
```

- *get_similar_tokens* — top-K words by cosine measure to the target word;
- glove_6b50d — glove model on some common corpora (Wikipedia?) with 6B of words and vector dimension equals to 50;

So what? (2) (Finding Analogies)

```
get_analogy('man', 'woman', 'son', glove_6b50d)
```

```
'daughter'
```

“Capital-country” analogy

```
get_analogy('bad', 'worst', 'big', glove_6b50d)
```

```
'biggest'
```

“Adjective-superlative adjective” analogy

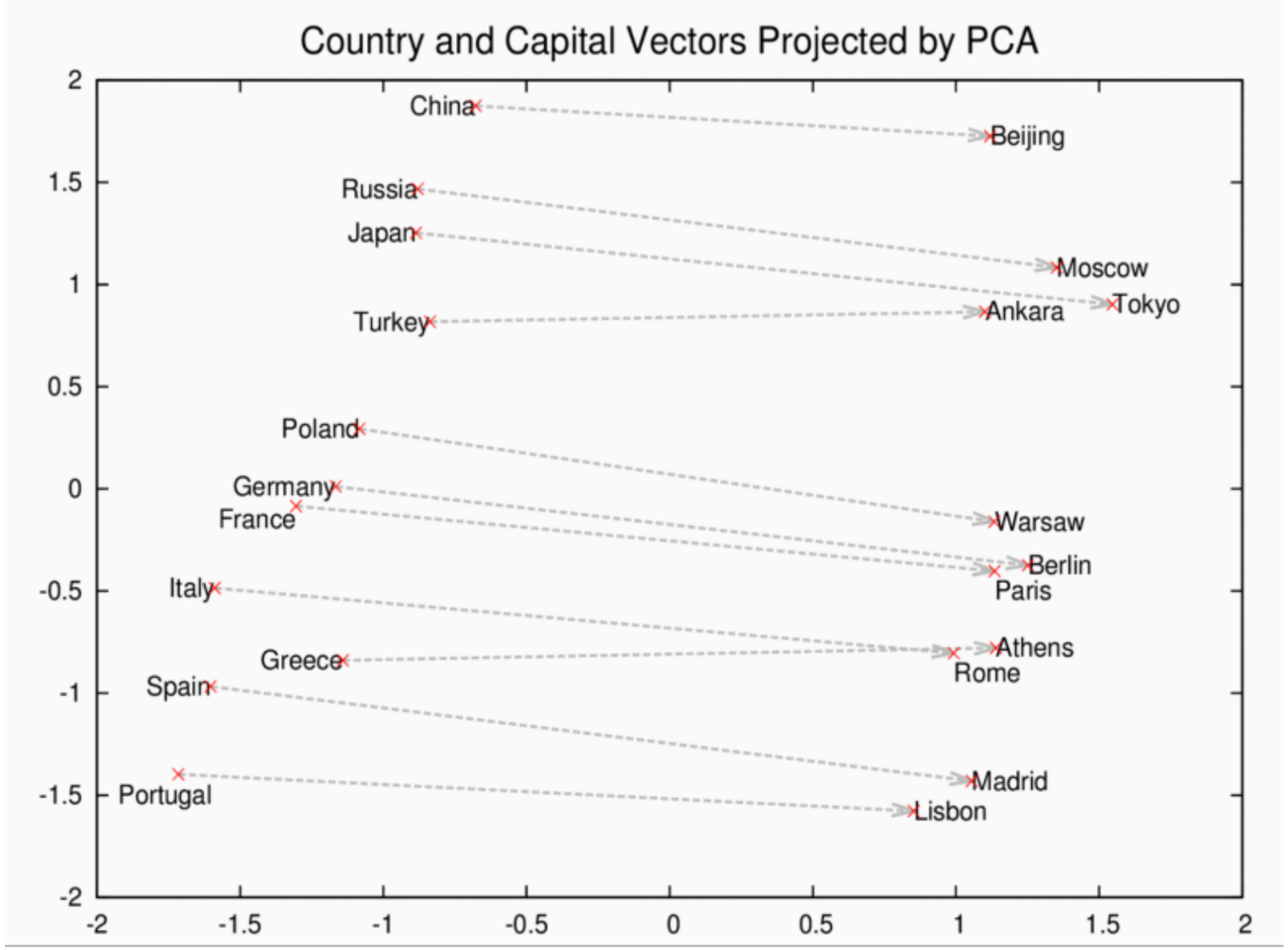
```
get_analogy('do', 'did', 'go', glove_6b50d)
```

```
'went'
```

“Present tense verb-past tense verb” analogy

- And it’s only x = $\mathbf{vec}(c) + \mathbf{vec}(b) - \mathbf{vec}(a)$
- And then top word for x

So what? (country-capital)



Based on Wikipedia training corpora

Any problems?

- Out-of-vocabulary

- How we can train it? How big our doc’s collection should be?

- Stop, firstly we talk about **text** and word2vec is about **words**

Any problems?

- Out-of-vocabulary

  Yeah, it’s true. But there are few extensions; (fastText)

- Stop, firstly we talk about **text** and word2vec is about **words**

  Ok, average it; Or average with weights; Or do not average and learn some averaging embedding; (look to BERT model)

- How we can train it? How big our doc’s collection should be?

  Really big; Starting from 10+M of symbols; Use pertained vectors;

Word2Vec in RecSys

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

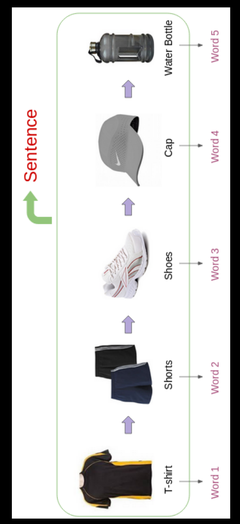
- Filter your datasets, please

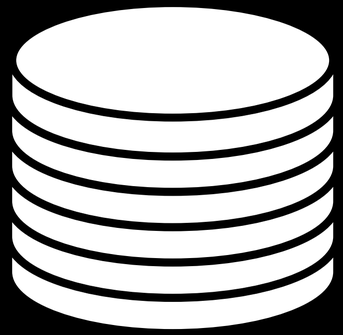
Word2Vec in RecSys

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

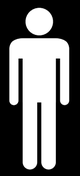
- Filter your datasets, please

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

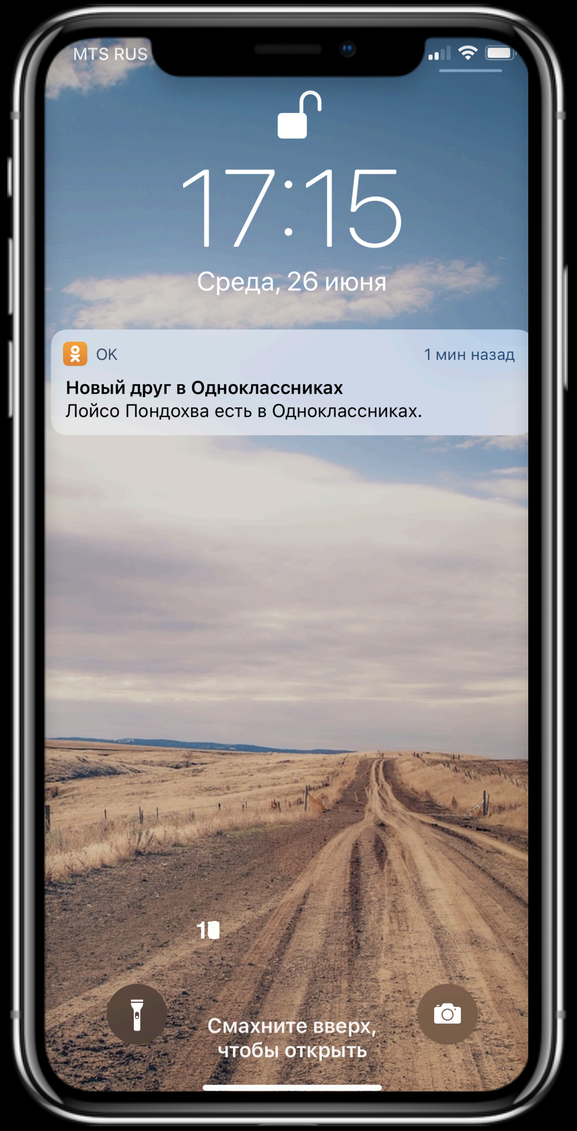
W2V

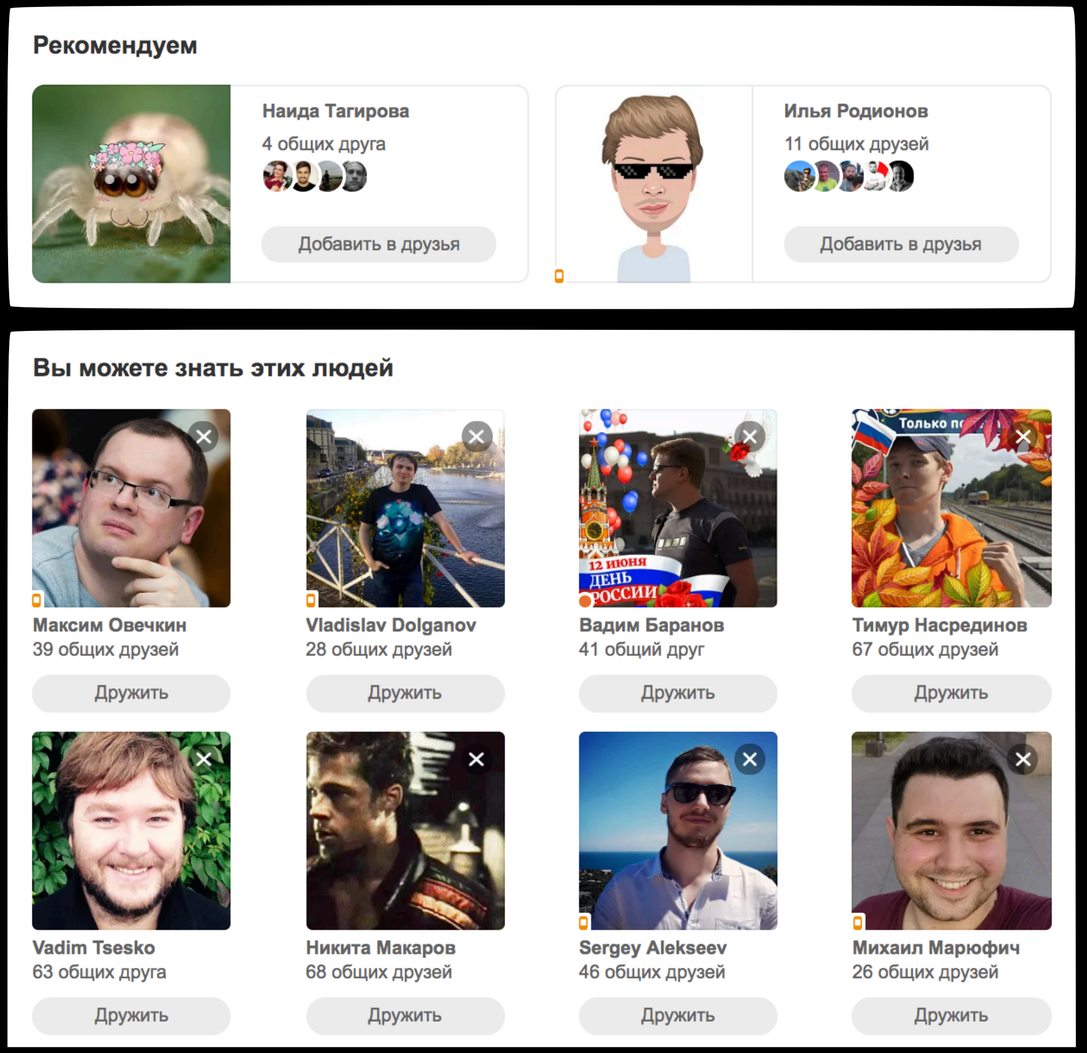
Discussion notes

- Items are limited by number of interactions — how to overcome it?
- Intrinsic evaluation. What to do?

- Rare/popular items lost their similars quality, why?

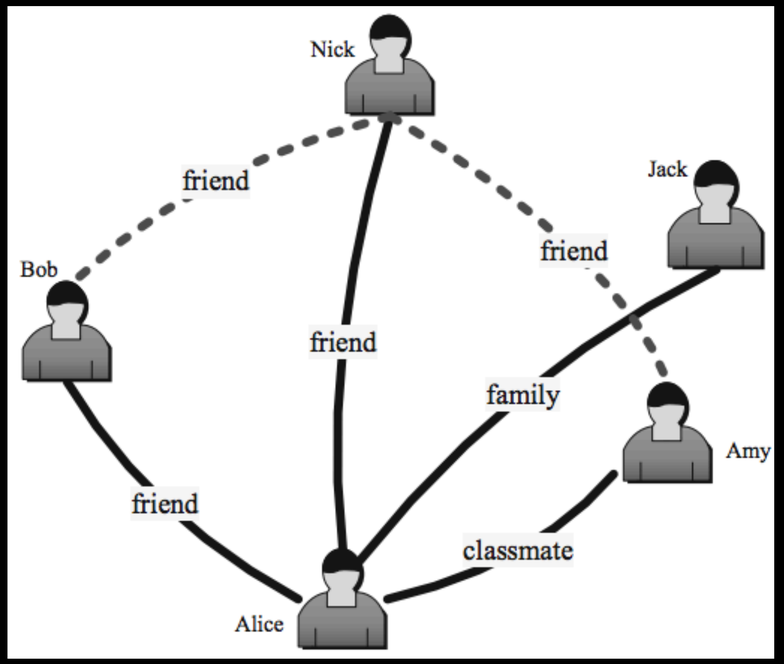
PYMK:


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

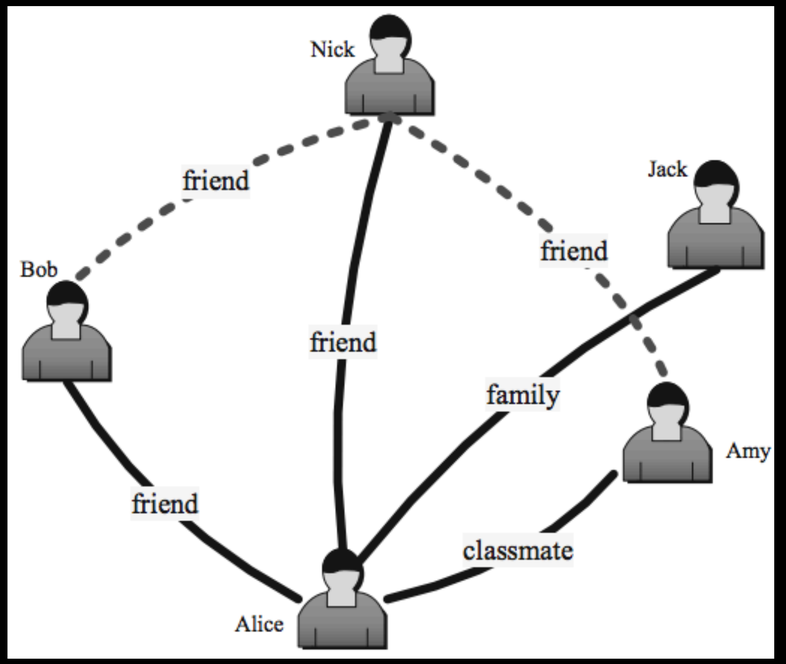
Link prediction(recovery):

- 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.



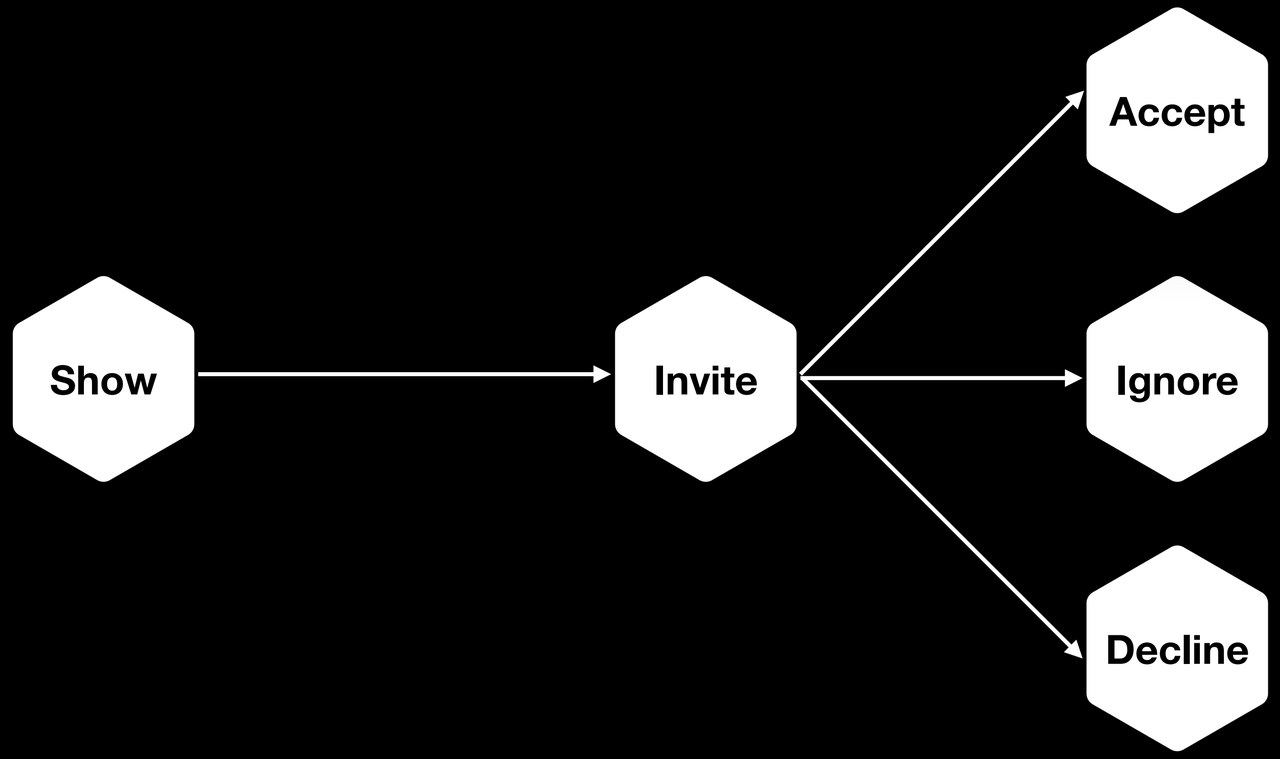
PROBLEMS?

Link prediction(recovery):

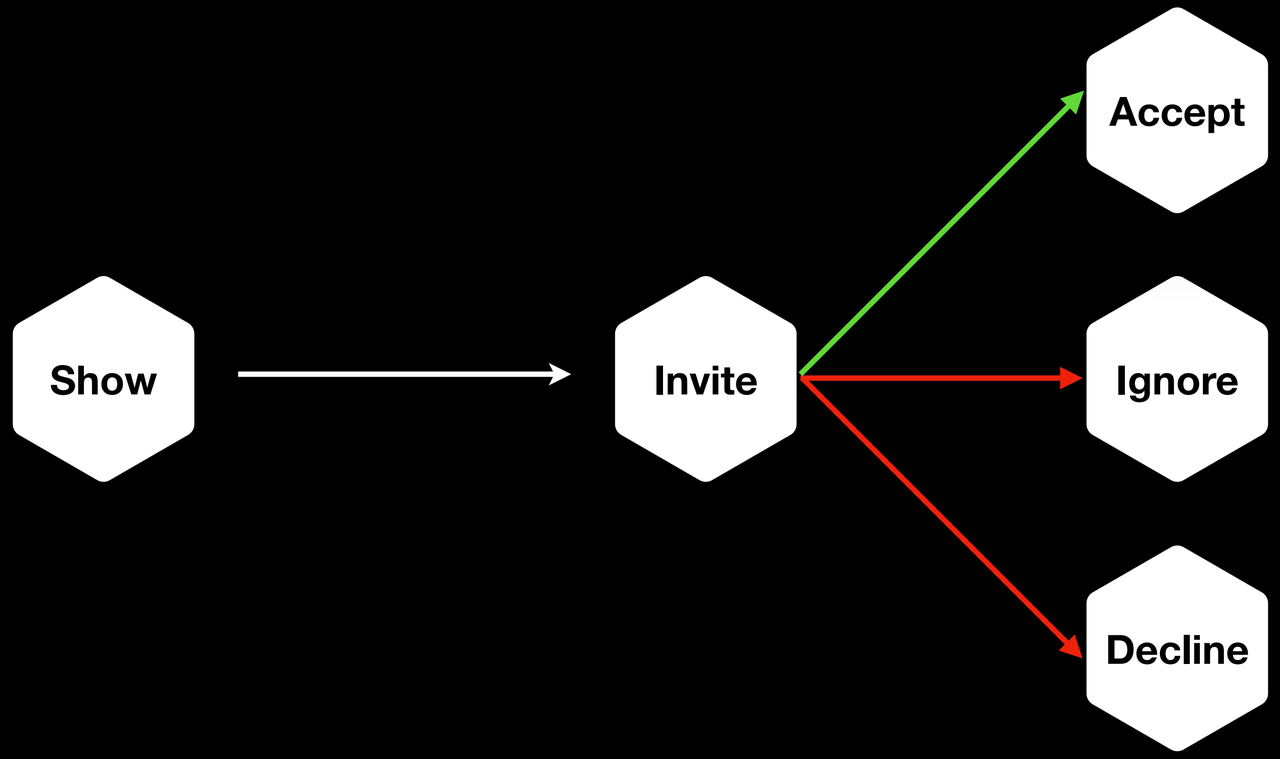
- Treat link prediction as binary classification.
- Each unconnected pair — corresponds to an instance with features and label.
- If there is a potential link — instance labeled as «positive», otherwise — negative.

- 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



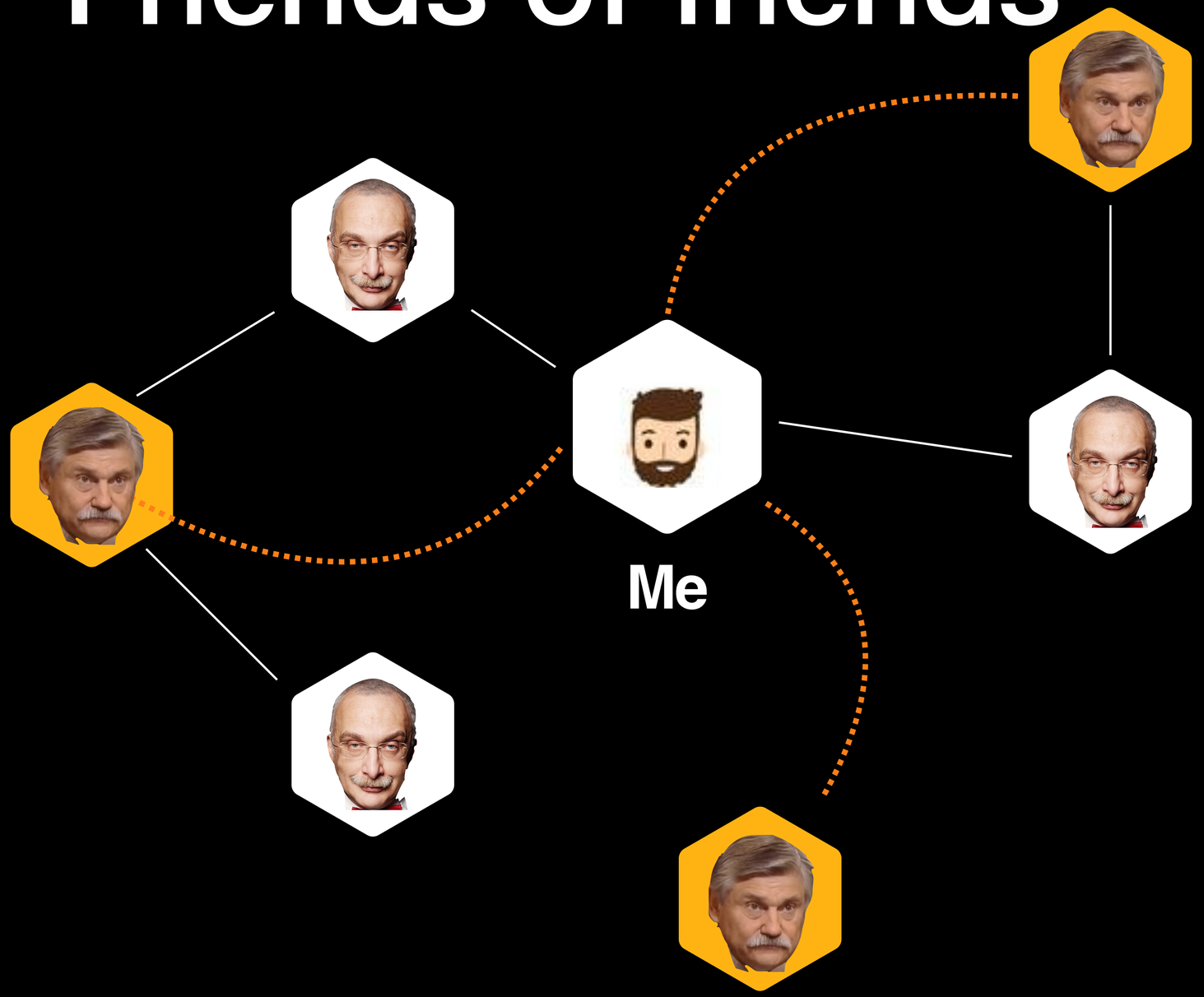
Recommendation System schema:

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

  ...

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

Friends of friends



- friends

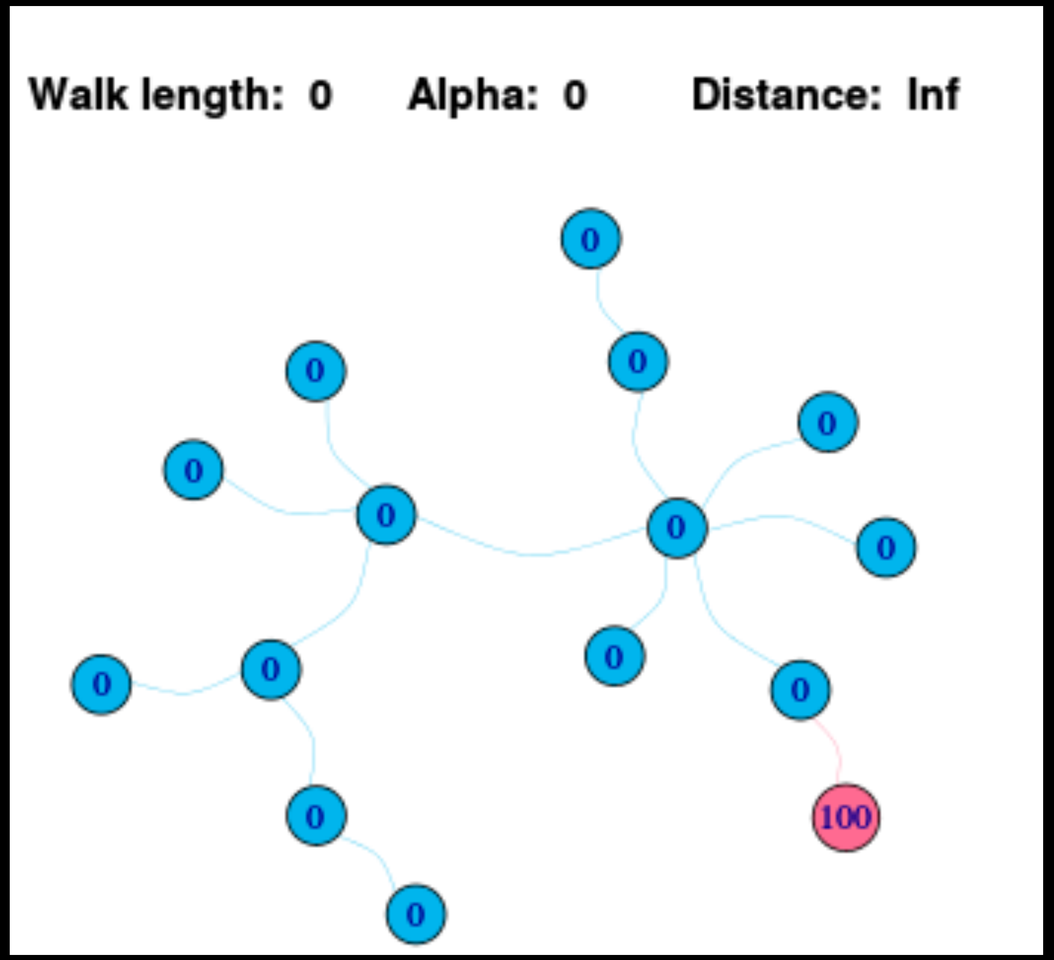
- candidates (FOF)

What’s more?

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

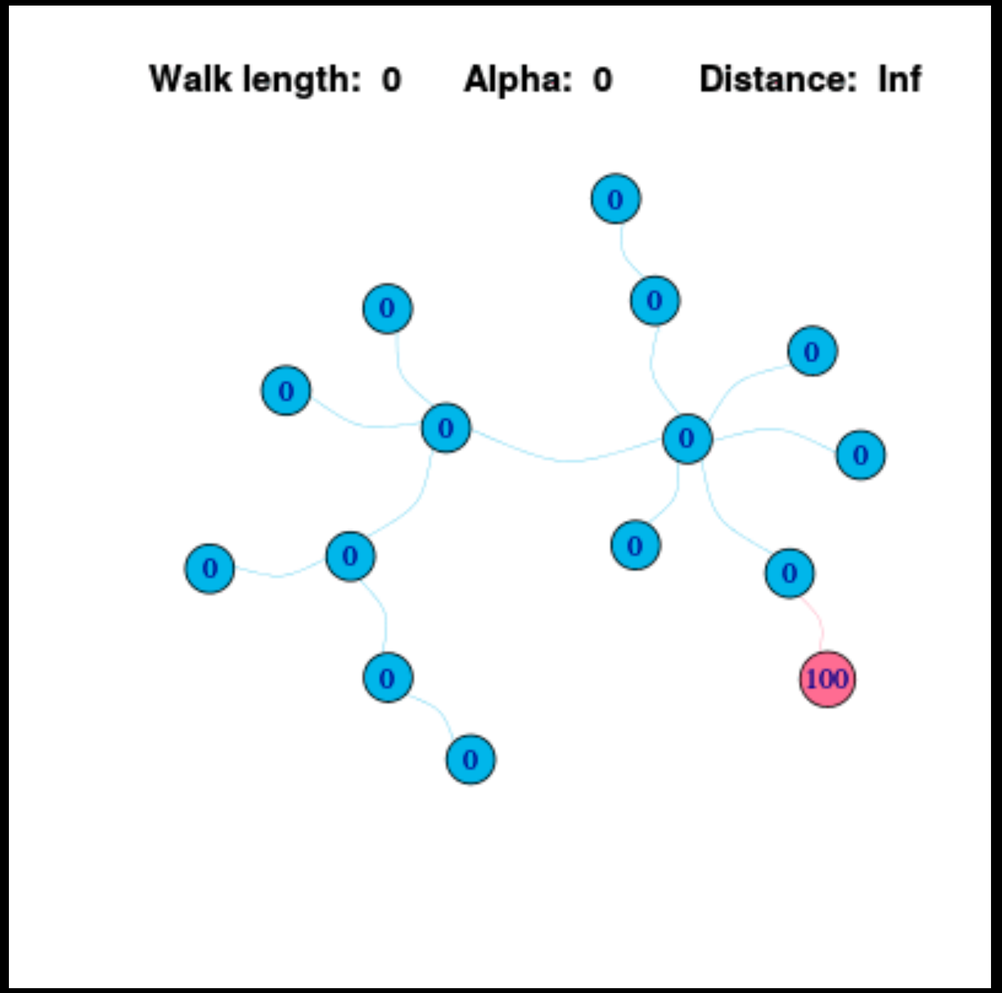
PageRank: random walks

- 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?



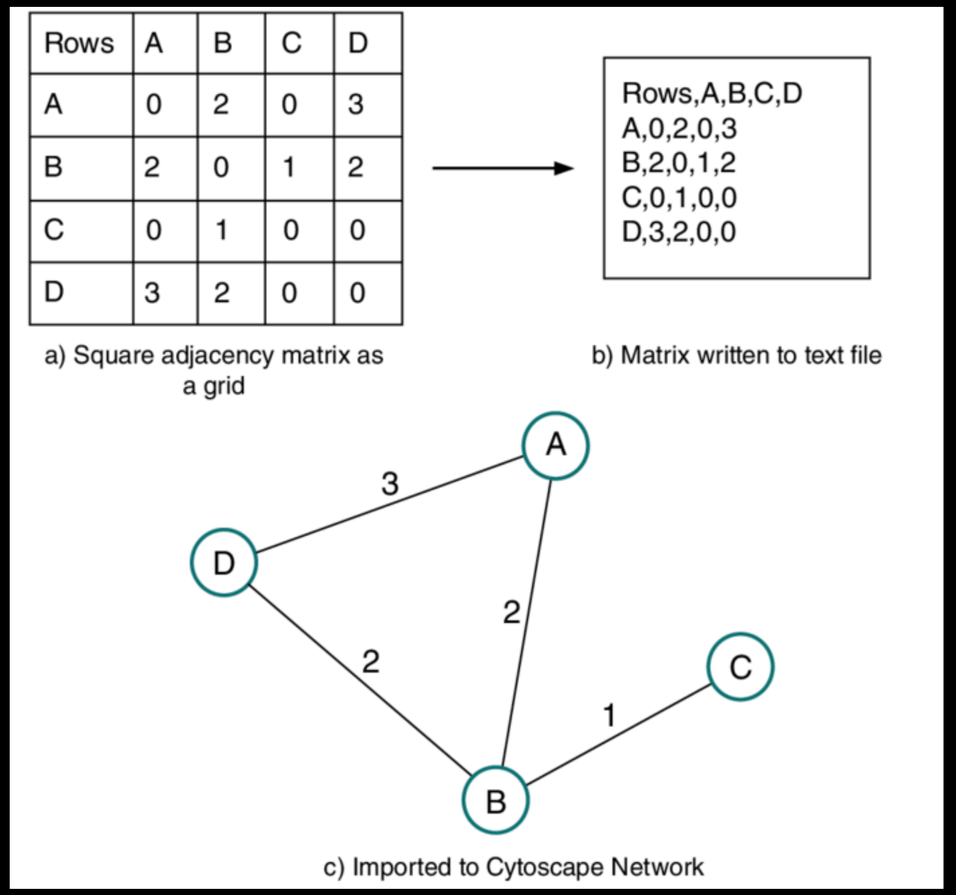
PageRank: random walks

- 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
```

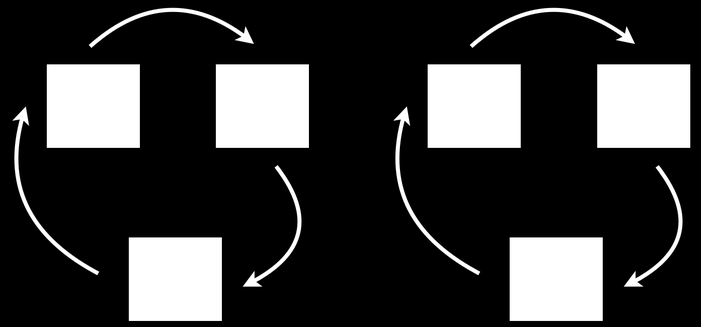
b) Matrix written to text file

c) Imported to Cytoscape Network

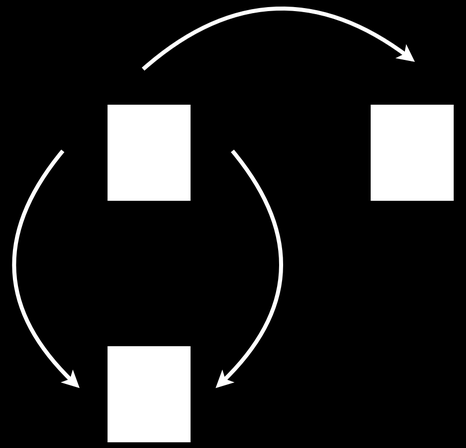
PageRank: as Markov chain

- 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?

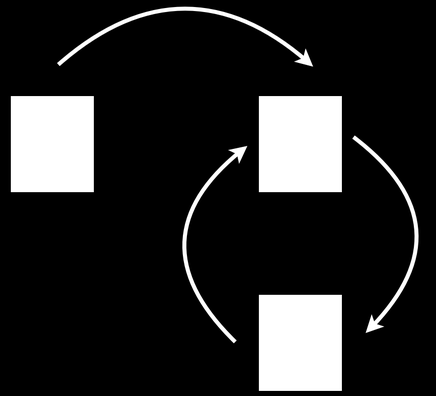
PageRank as Markov chain:



Disconnected components



Dangling links

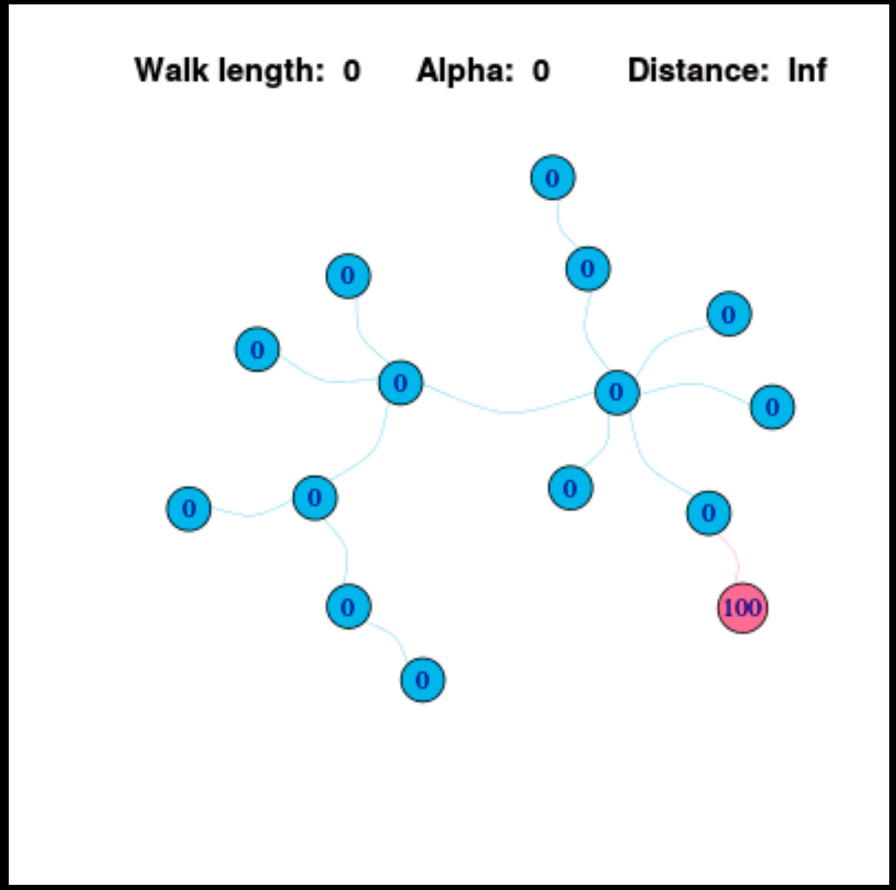


Loops

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

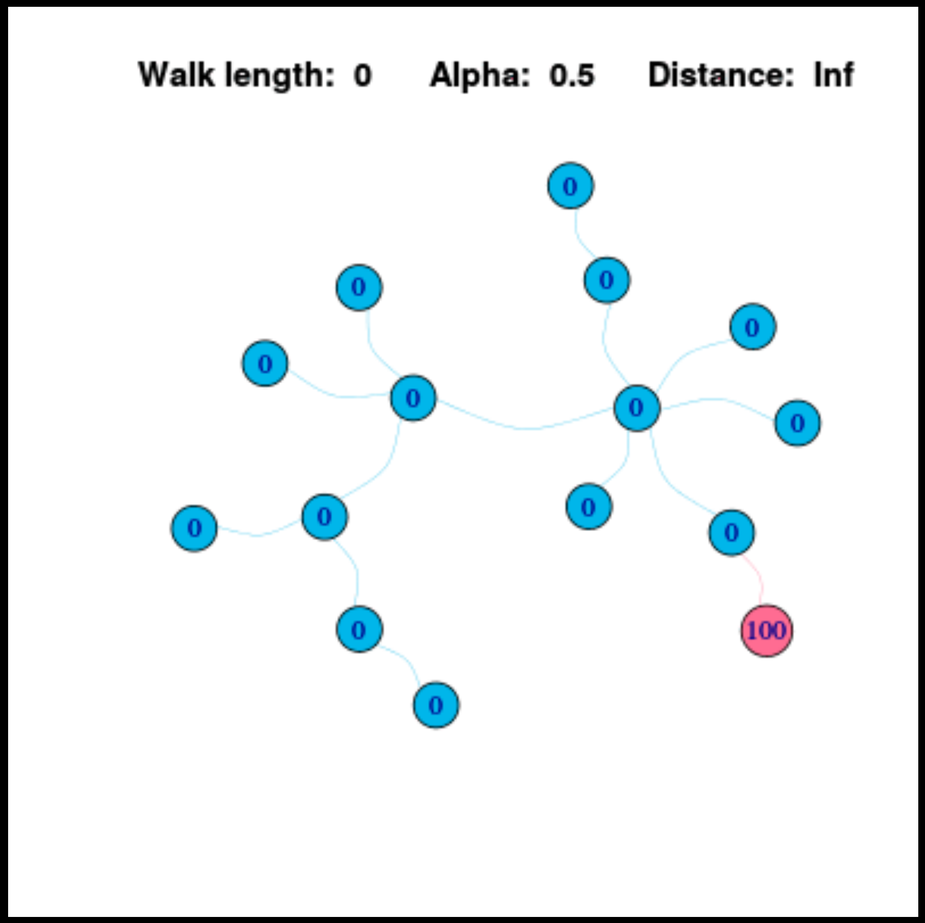
Lazy random walks

- 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.



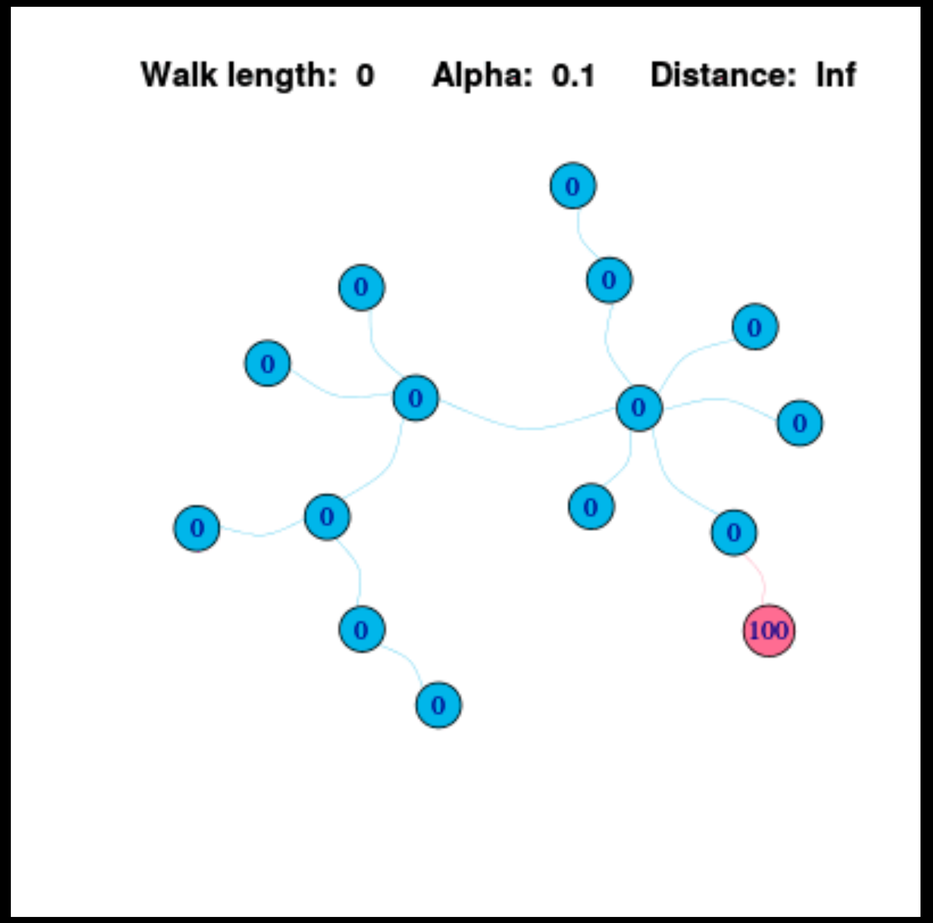
Personalized PageRank:

- 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**

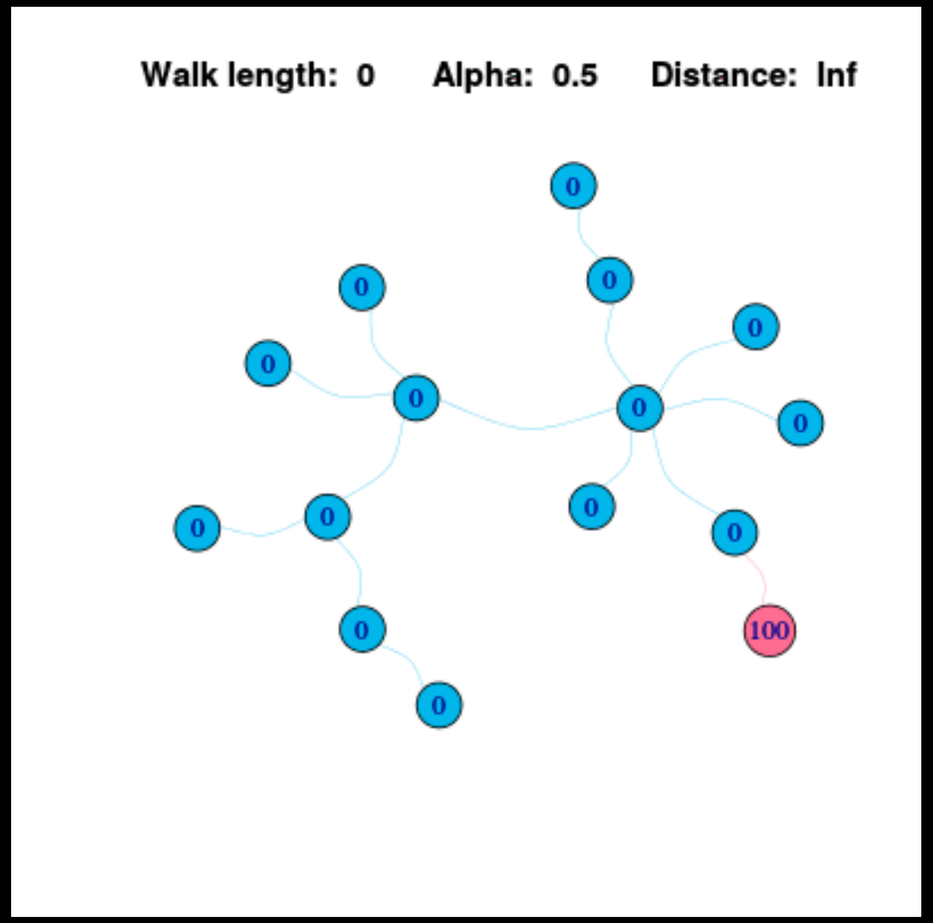


Alpha = 0.5

Personalized PageRank:



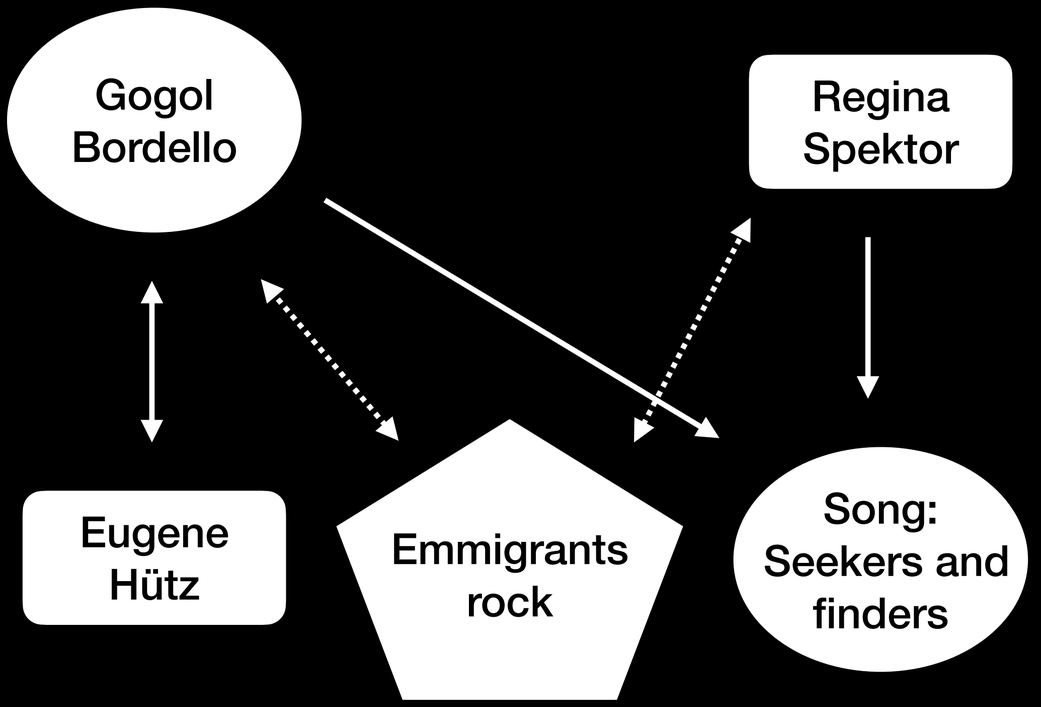
Alpha = 0.1



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
- …?

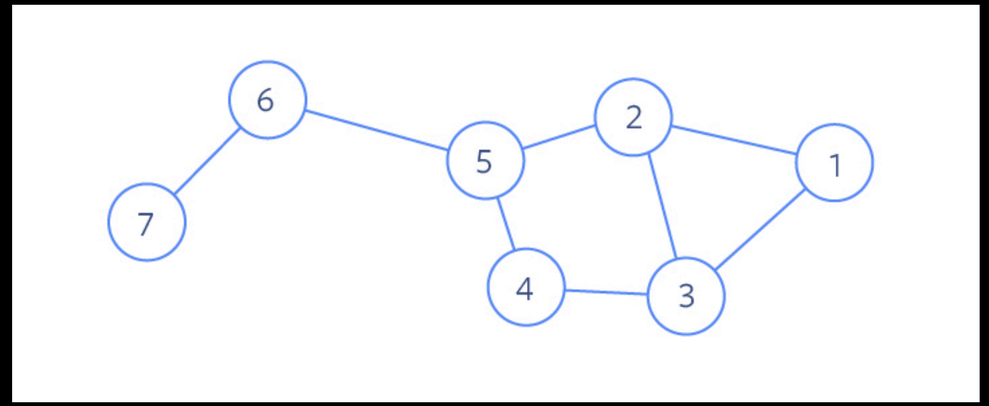
BigGraph by Facebook:

- 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

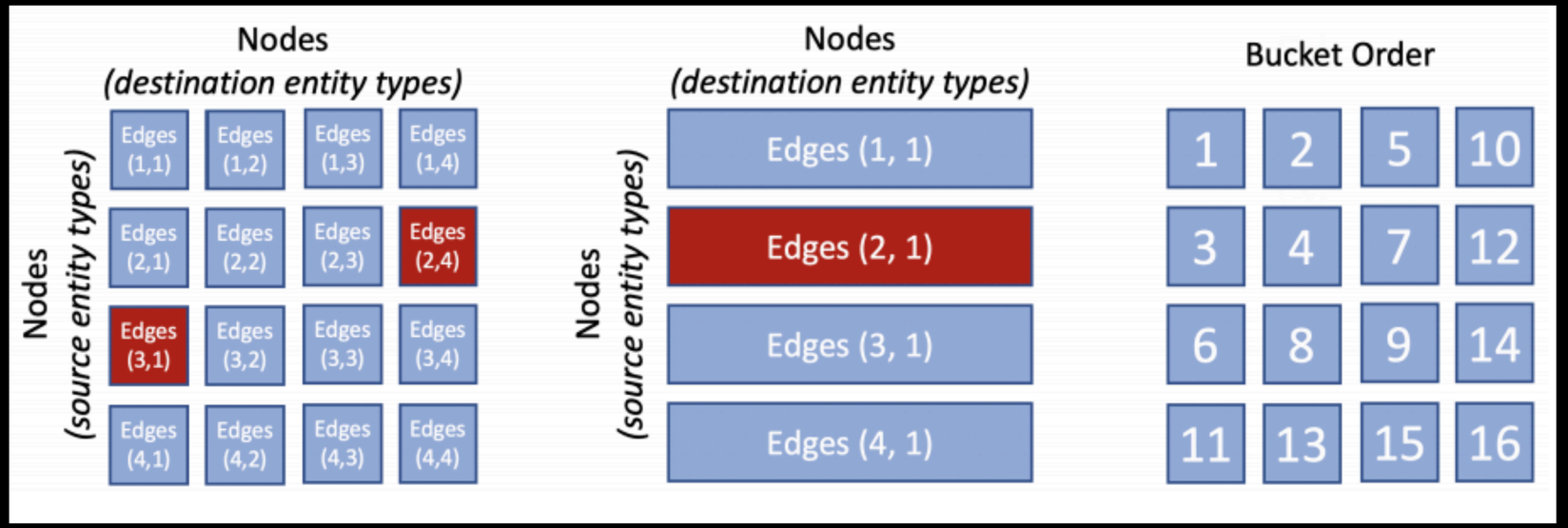


BigGraph:

- **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).

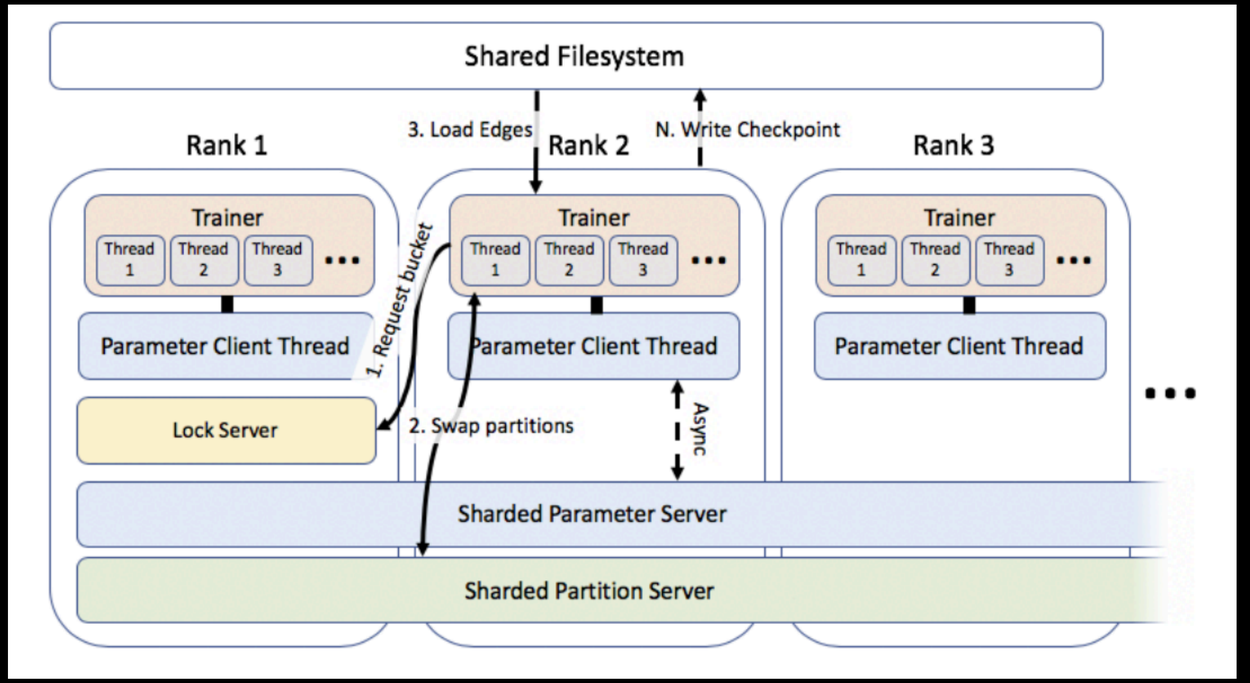
- **Center:** Entity types with small cardinality do not have to be partitioned; if all entity types used for tail nodes are unpartitioned, then edges can be divided into P buckets based only on source node partitions.

- **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 some 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

Unclassical embeddings

Summary

- Not only MF
- Different embeddings fits different purposes and restrictions
- Based on nature of your product (graph-, session- based)
- It doesn’t mean that different embeddings replaces one each other — combine it!
- There are still a lot of tricks to combine one with each other:
  - Similar to last positives
  - Score, is_from_source, num in similar top for each embedding in features