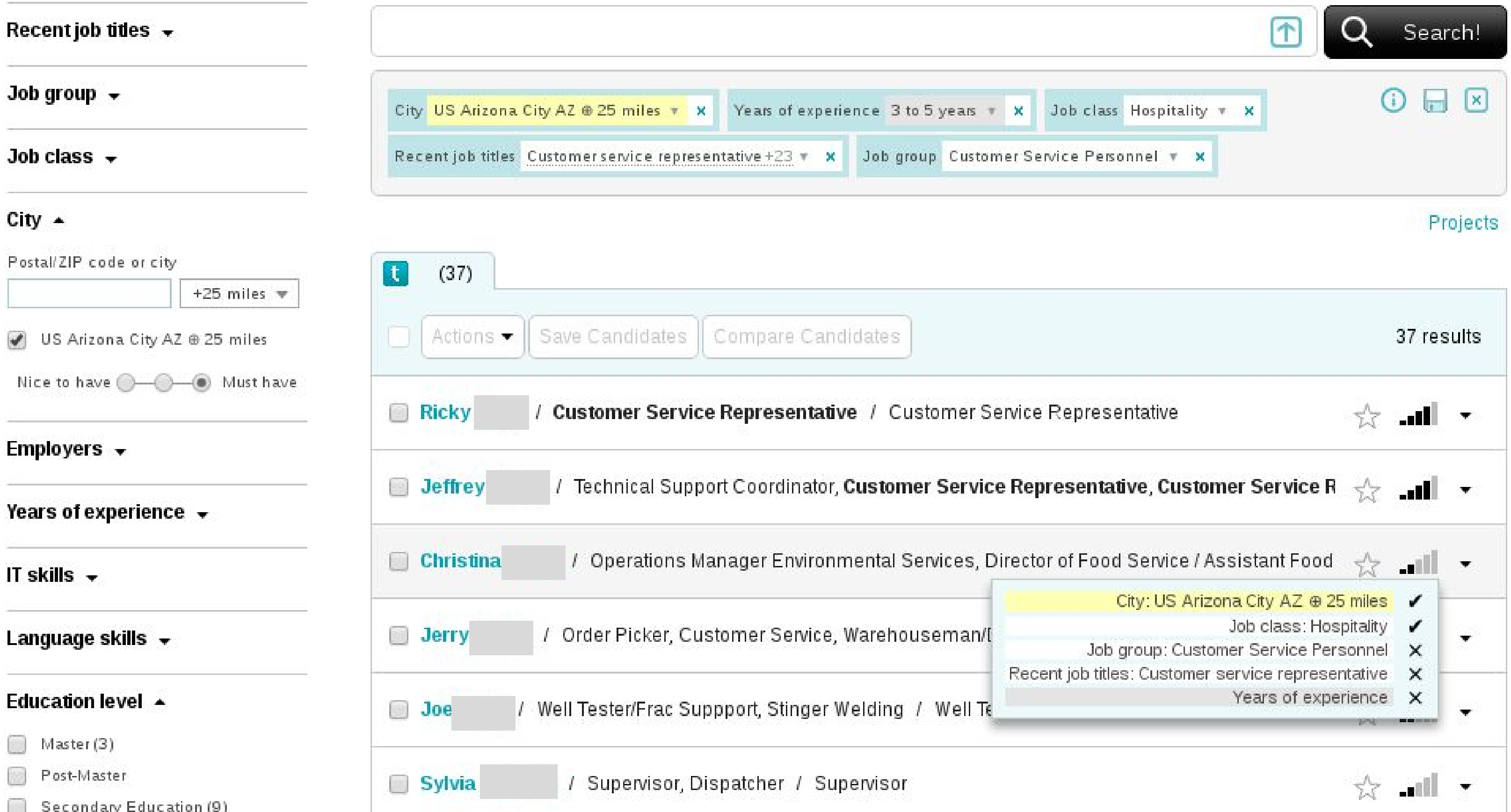
Learning to Rank for Faceted Search

Bridging the gap between theory and practice

Agnes van Belle @ Berlin Buzzwords 2017

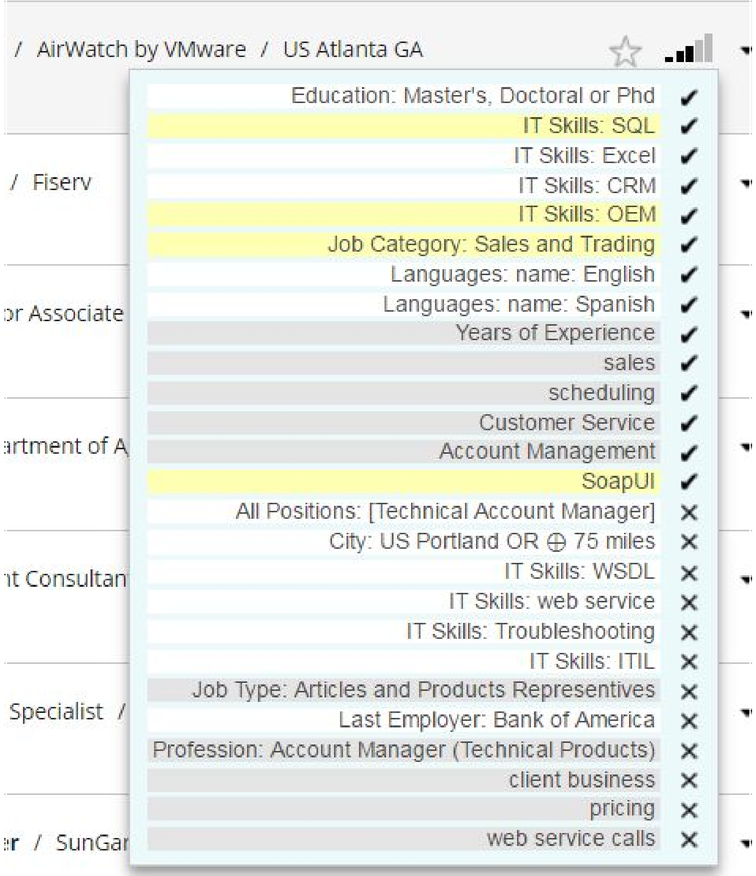
powered by textkernel

Job-to-person search system



Generated query

Match indicator



Faceted search

- Multiple explicit and independent dimensions, called facets
- Lets users refine search by choosing values
- No candidate is ideal: many should-have clauses

Scoring of search results

- Term-frequency based metric
  e.g. **BM25, TF-IDF**

- Facet weights

TF-IDF(jobtitle**)** ·	1.5	+
TF-IDF(skill**)** ·	2.0	+
TF-IDF(location**)** ·	0.7	+
TF-IDF(languages**)** ·	0.25	= score

Tuning the system: objectives

- “If I search for a skill ‘Java’ I want the candidates that also have ‘Java’ in their Jobtitle field to be weighted higher”
- “Education will be a less important match, the more years of experience a candidate has”
- “We should weight location matches less when finding candidates in IT”

Learning to rank

- Learn a parameterized ranking model
- That optimizes ranking order
- Re-learn for personalization or preference change

Learning to rank by tuning facet weights

- Do exhaustive search for optimal weights to set
- Improved our retrieval by **6%** (NDCG metric)

$$\left[\ \textbf{TF-IDF(}\text{jobtitle}\textbf{)},\ \textbf{TF-IDF(}\text{skill}\textbf{)}\ \ldots\ \textbf{TF-IDF(}\text{language}\textbf{)}\ \right] \cdot \begin{bmatrix} \mathbf{1.5} \\ \mathbf{2.0} \\ \vdots \\ \mathbf{0.25} \end{bmatrix} = \text{score}$$

Tuning facet weights: limitations

- Cannot consider interdependency of facet field dimensions
- Cannot take into account the actual *content* of fields
  - only match indicators

Learning objectives

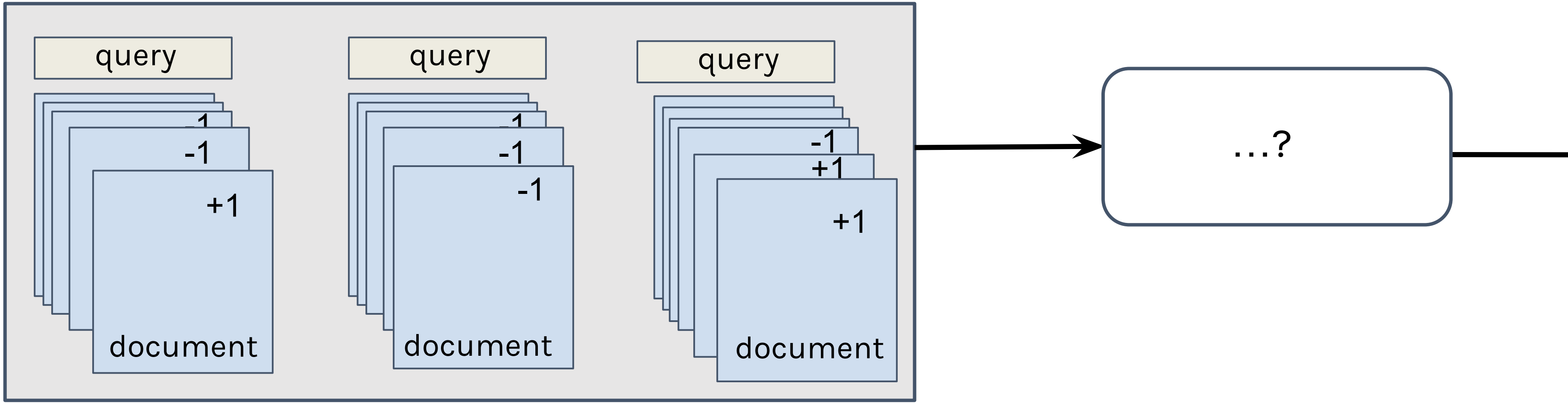
- Take into account facet field content
- Model facet field interdependencies

Learning to rank

- Machine Learning from user feedback
- Input: set of {query, lists of assessed documents}
    - Each document has a relevance indication from feedback

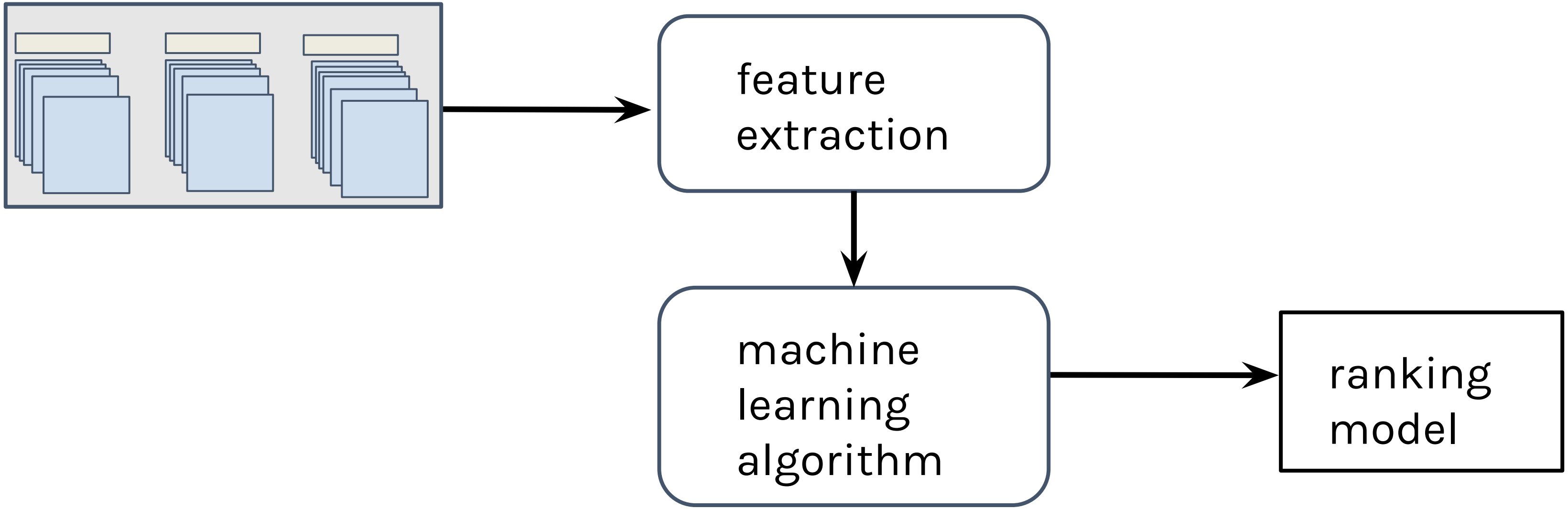
Learning to rank

- Machine Learning from user feedback
- Input: set of {query, list of assessed documents}
    - Each document has a relevance label from feedback



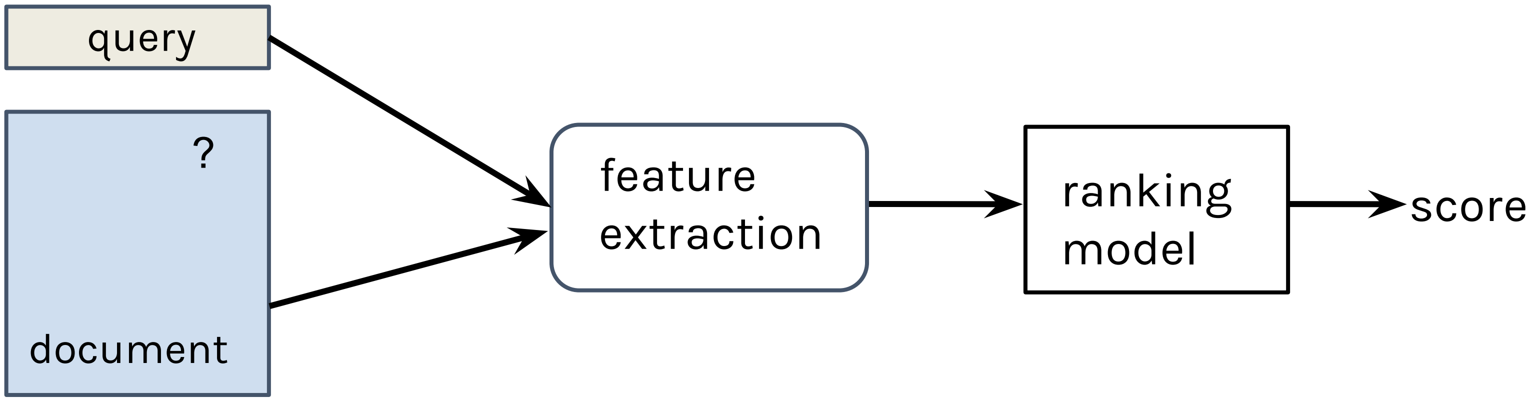
Learning to rank

- Algorithm learns how to combine query & document content to optimize ordering considering relevance labels

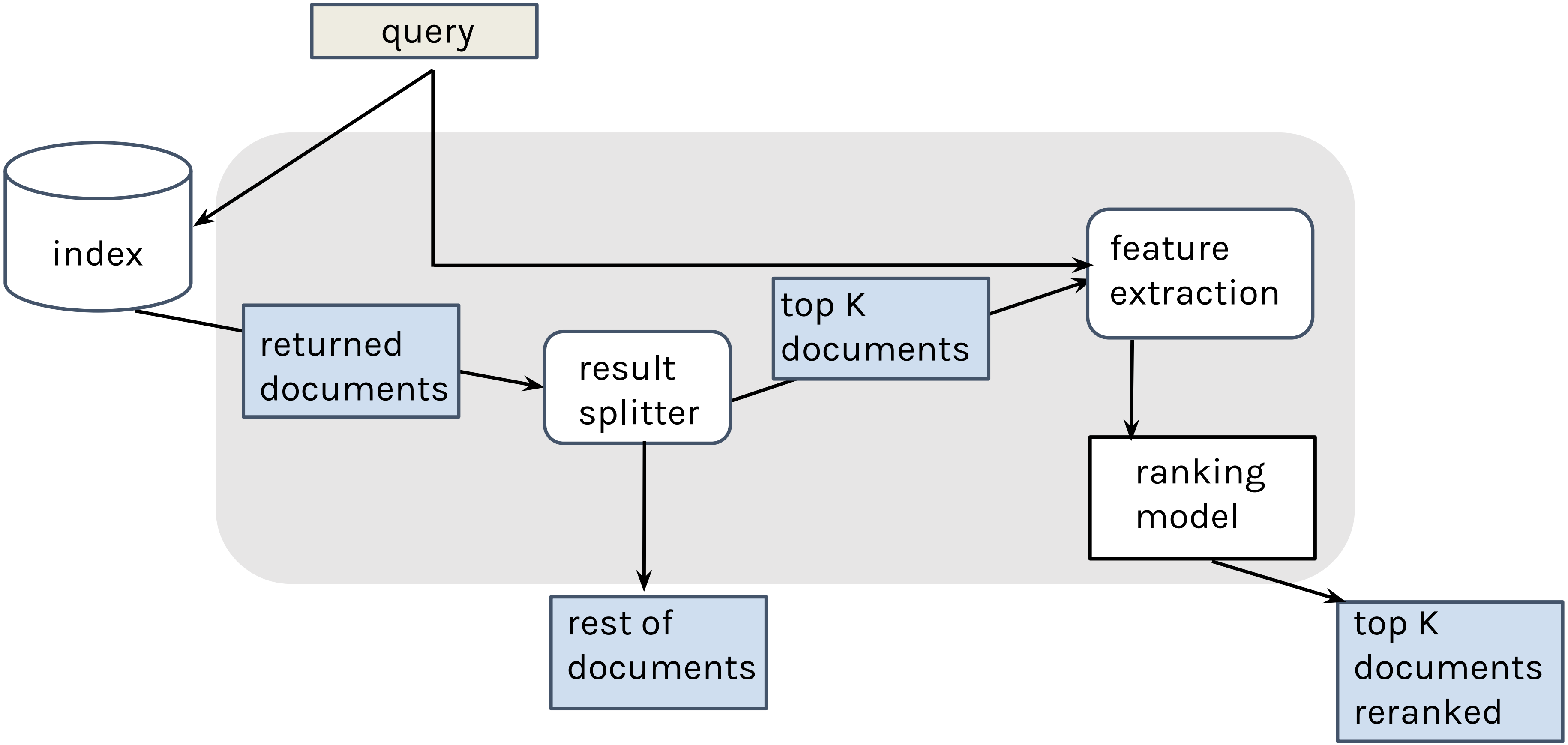


Learning to rank

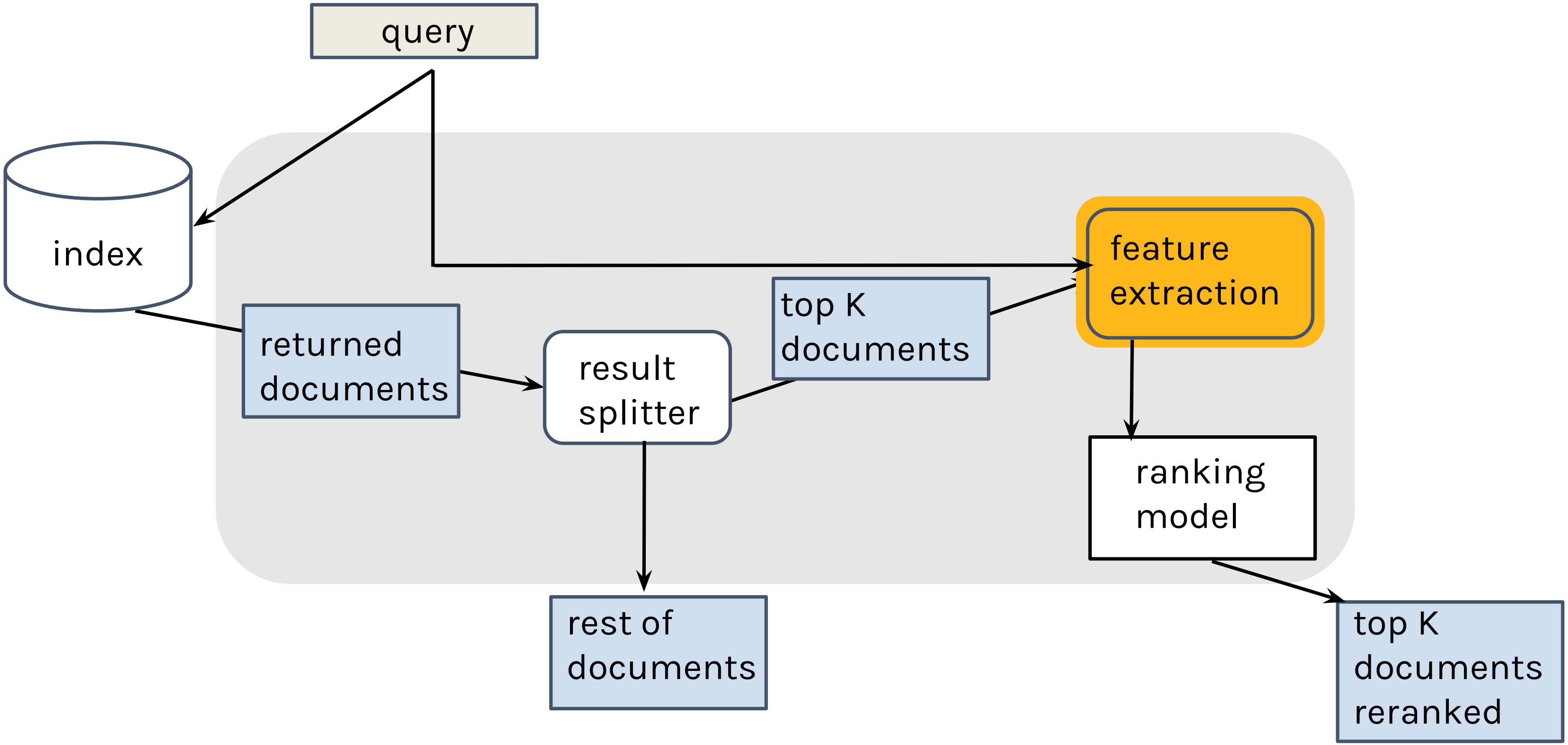
- Output: model that gives a relevance score given a query and document



Dynamic top K reranking



Dynamic top K reranking



Typical features

"*In learning to rank, each query-document pair is represented by a multi-dimensional feature vector, and* ***each dimension of the vector is a feature indicating how relevant or important the document is with respect to the query.***" [*]

Used in LTR papers: [1, 2, 3]

- TF-IDF, BM25, DFR, Language Model, cosine similarity, rank in other engines, etc.
- Match-indicator between *whole* query & *whole* document

* "LETOR: A Benchmark Collection for Research on Learning to Rank for Information Retrieval", T. Qin, T. Liu, J. Xu, Jun and H. Li, 2010
1 "Optimizing Search Engines using Clickthrough Data", T. Joachims, 2003
2 "AdaRank: A Boosting Algorithm for Information Retrieval", J. Xu and H. Li, 2007
3 "Multileave Gradient Descent for Fast Online Learning to Rank", A. Schuth, H. Oosterhuis, S. Whiteson and M. de Rijke, 2016

Bag of words

software engineer data mining java amsterdam english

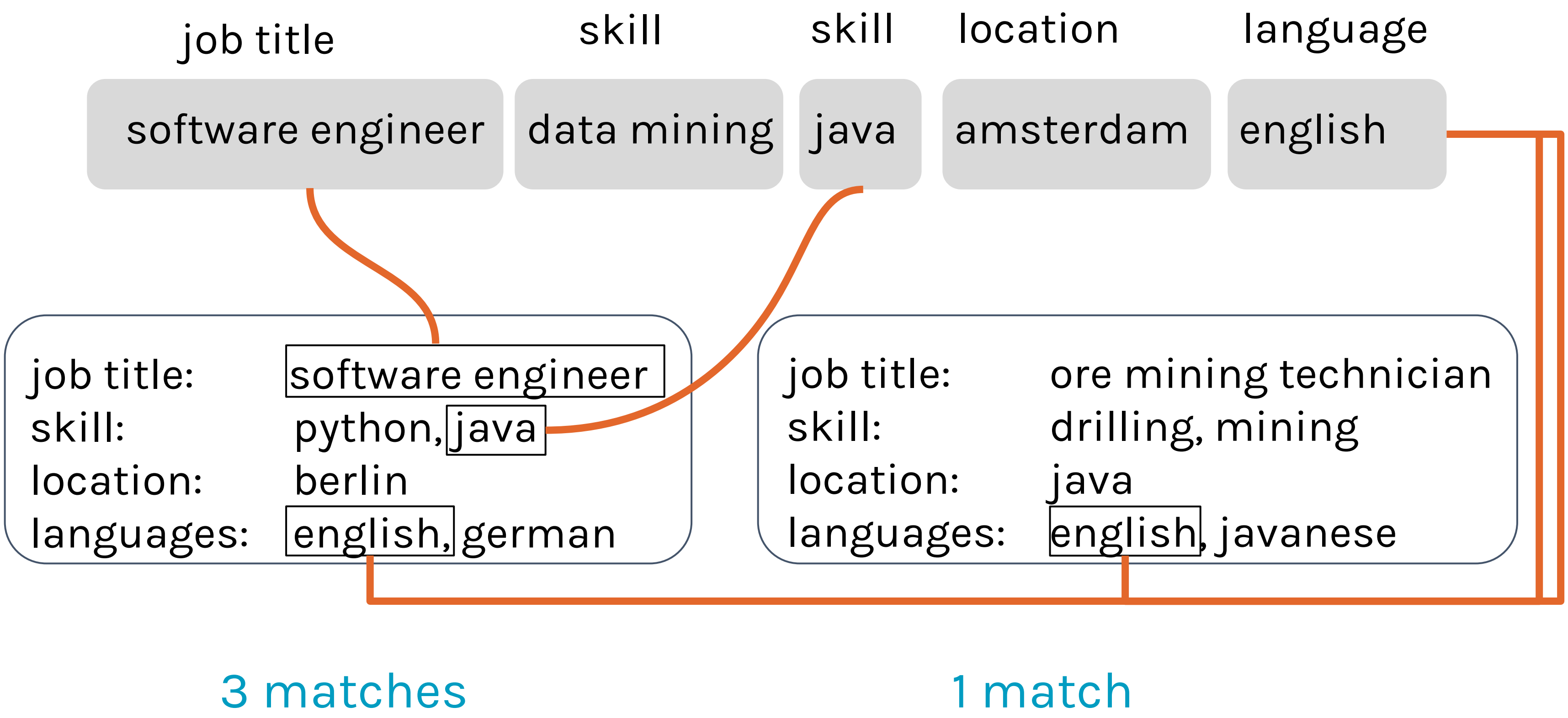
job title:	$\boxed{\text{software}}\boxed{\text{engineer}}$
skill:	python, $\boxed{\text{java}}$
location:	berlin
languages:	$\boxed{\text{english,}}$ german

4 matches

job title:	ore $\boxed{\text{mining}}$ technician
skill:	drilling, $\boxed{\text{mining}}$
location:	$\boxed{\text{java}}$
languages:	$\boxed{\text{english}}$, javanese

4 matches

Split up in facet fields



One feature per field

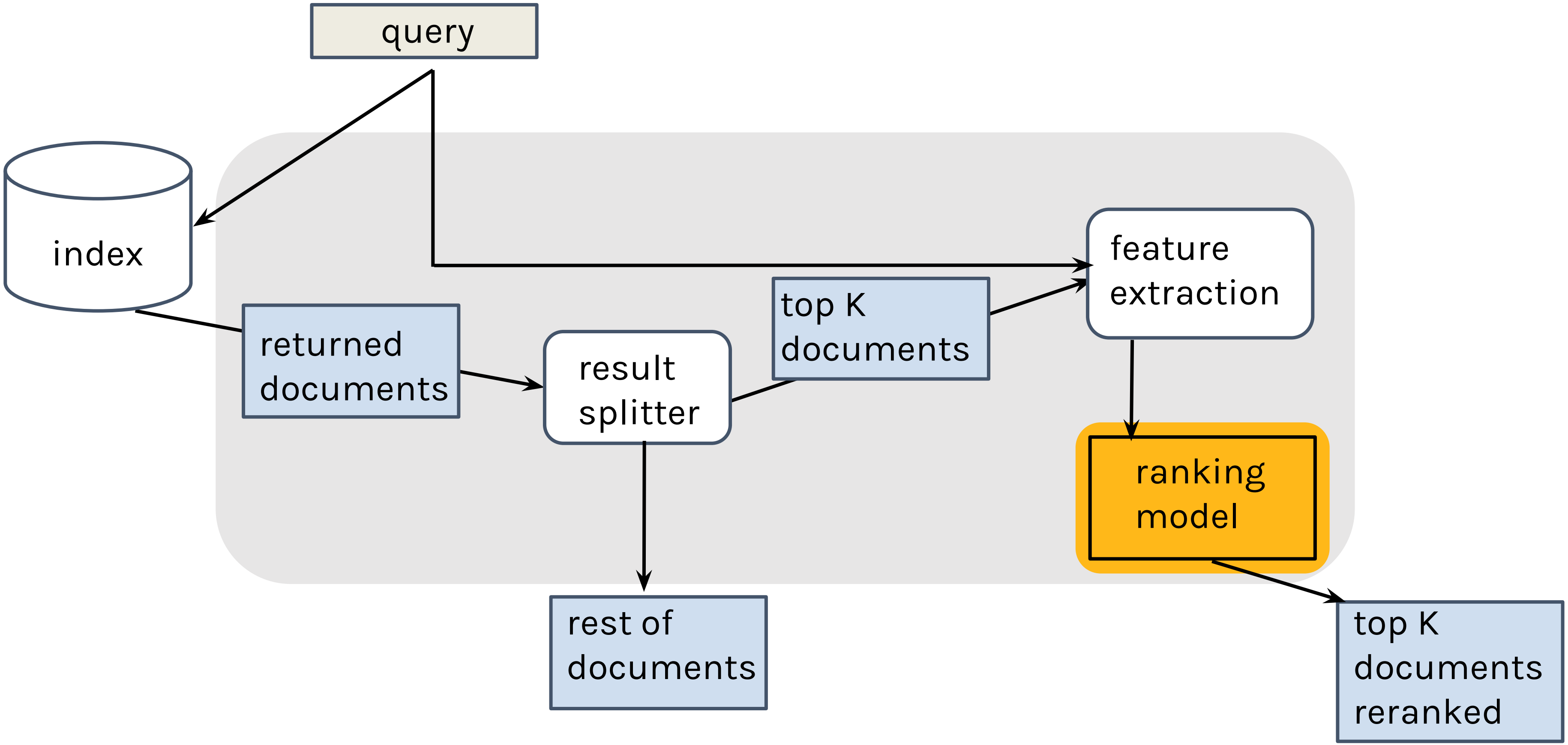
job title	skill	skill	location	language
software engineer	data mining	java	amsterdam	english

job title: software engineer
skill: python, java
location: berlin
languages: english, german

feature vector

jobtitle	1/1
skill	1/2
location	0
language	1/1

Dynamic top K reranking



Linear models

- Used in many papers:
  - seminal papers[1],
  - papers about leveraging user preferences[2]
  - papers about online learning / interleaving[3]
- Also in e.g. documentation about Solr's LTR contrib module

1 "Optimizing Search Engines using Clickthrough Data", T. Joachims, 2003
2 "A contextual-bandit approach to personalized news article recommendation", L. Li, W. Chu, J. Langford, and R. E. Schapire, 2010.
3 "Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval", K. Hofmann, S.Whiteson, M. de Rijke, 2013

Linear models

End up with weight vector you can multiply with feature vectors.

$$\begin{bmatrix} f_1 & f_2 & f_3 & \dots & f_n \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix} = \text{score}$$

Linear models

End up with weight vector you can multiply with feature vectors.

$$\begin{bmatrix} \overset{\text{jobtitle match}}{1.0} & \overset{\text{skill match}}{0.5} & \underset{\text{location match}}{0.0} & \ldots & \underset{\text{language match}}{1.0} \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix} = \text{score}$$

Tuning facet weights: limitations😿

- Cannot consider interdependency of facet field dimensions
- Cannot take into account the actual *content* of fields
  - only match indicators

jobtitle match   skill match

$$[1.0 \;\; 0.5 \;\; 0.0 \;\; \ldots \;\; 1.0] \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix} = \text{score}$$

location match   language match

Objectives

- “If I search for a skill ‘Java’ I want the candidates that also have ‘Java’ in their Jobtitle field to be weighted higher”
- “Education will be a less important match, the more years of experience a candidate has”
- “We should weight location matches less when finding candidates in IT”

Learning objectives

- Take into account facet field content
- Model facet field interdependencies

Take into account facet field content

jobclass:IT was in document

jobclass:Retail was not in document

the query asked for 5+ years of experience

$$[0.0 \quad 1.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.5 \quad 0.0 \quad \ldots]$$

5 possible “jobclass” categories in document

how many minimum years of experience in the query, normalized between 0 and 1

Categorical feature

Interval feature

Take into account facet field content

- Query-document match features
- Document features
- Query features

Categorical: e.g. denoting job-class, skill etc.

Interval: e.g. years of experience

Model facet field interdependencies

jobclass:IT was in document

jobclass:Retail was not in document

$$\begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.5 & 0.0 & \ldots \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix} = \quad \ldots$$

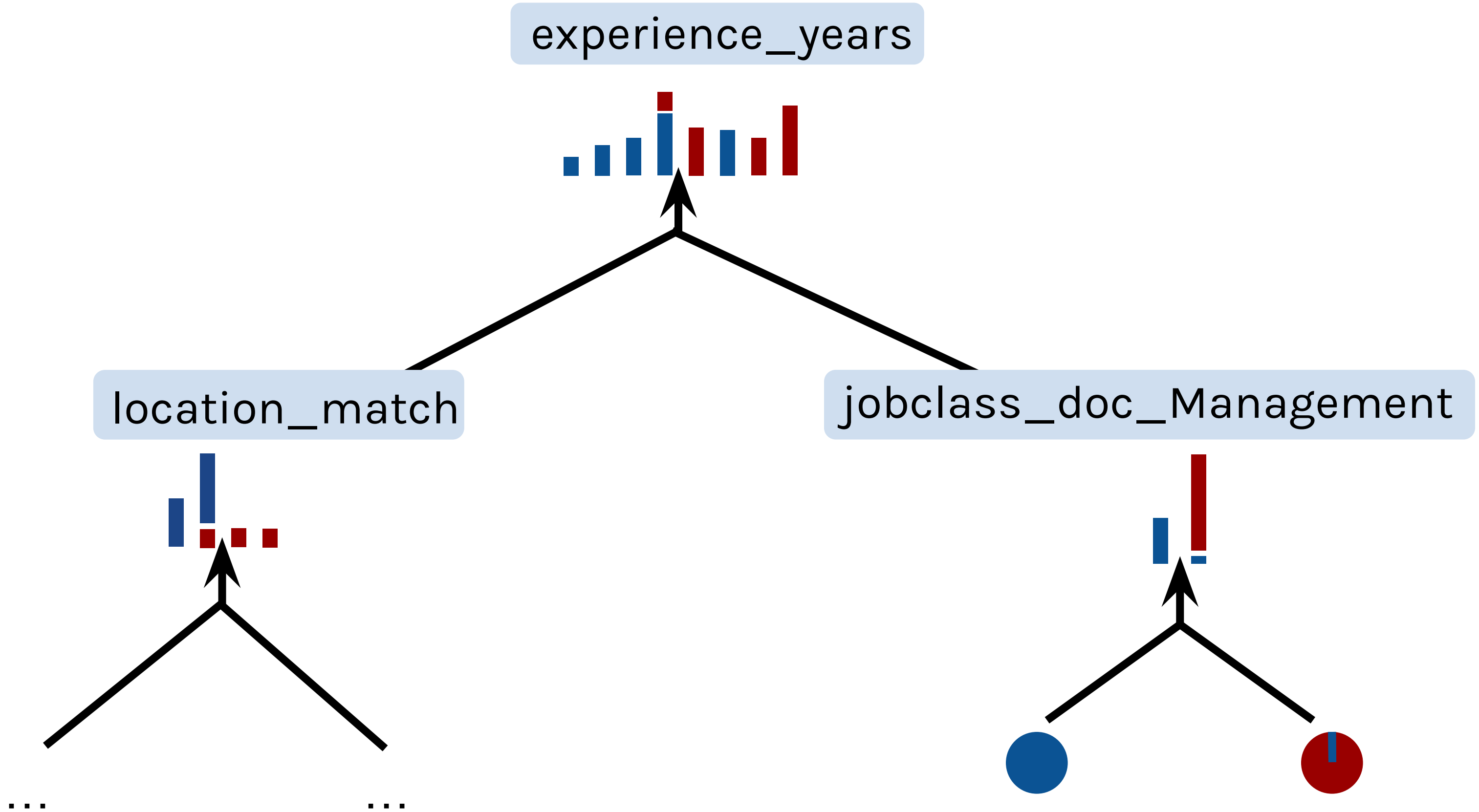
Model facet field interdependencies

Use nonlinear ranking model based on e.g.

- Nonlinear neural networks
- Nonlinear SVM
- Decision trees

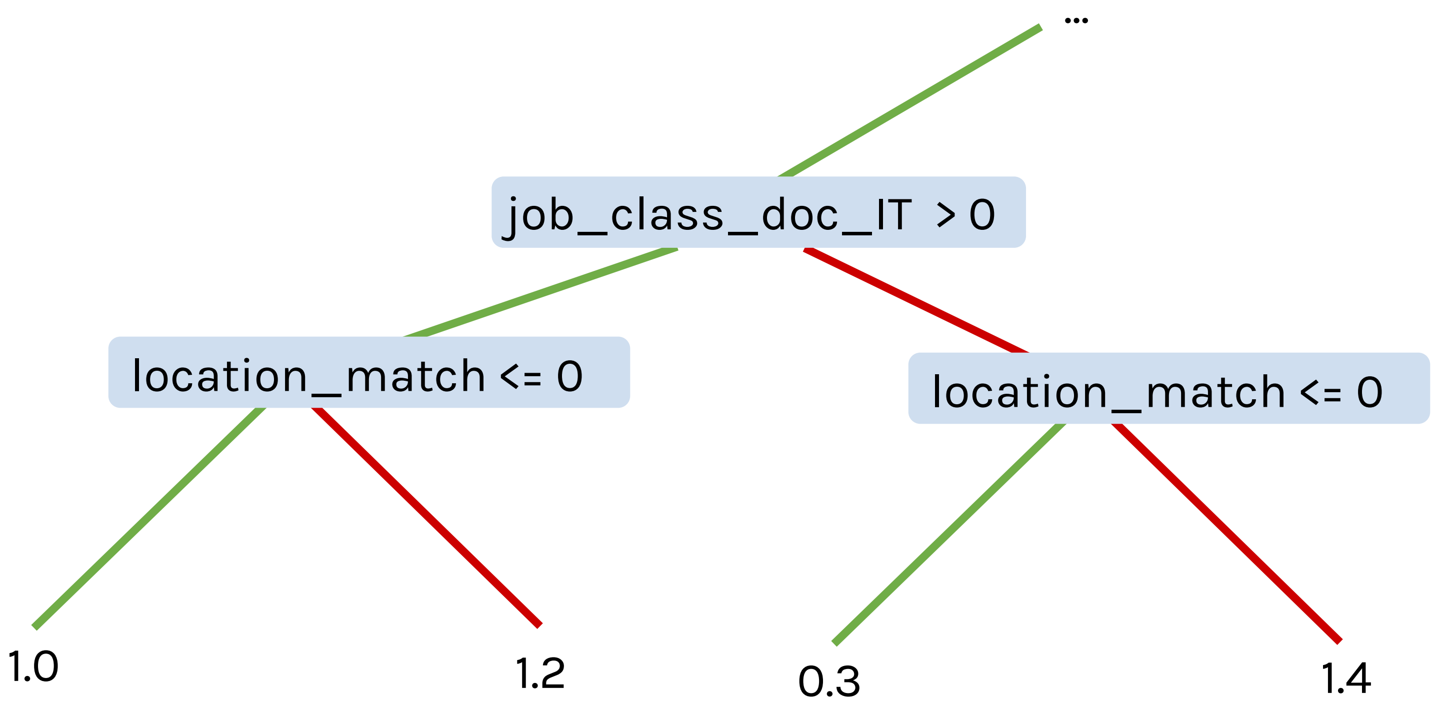
Model facet field interdependencies

Decision tree



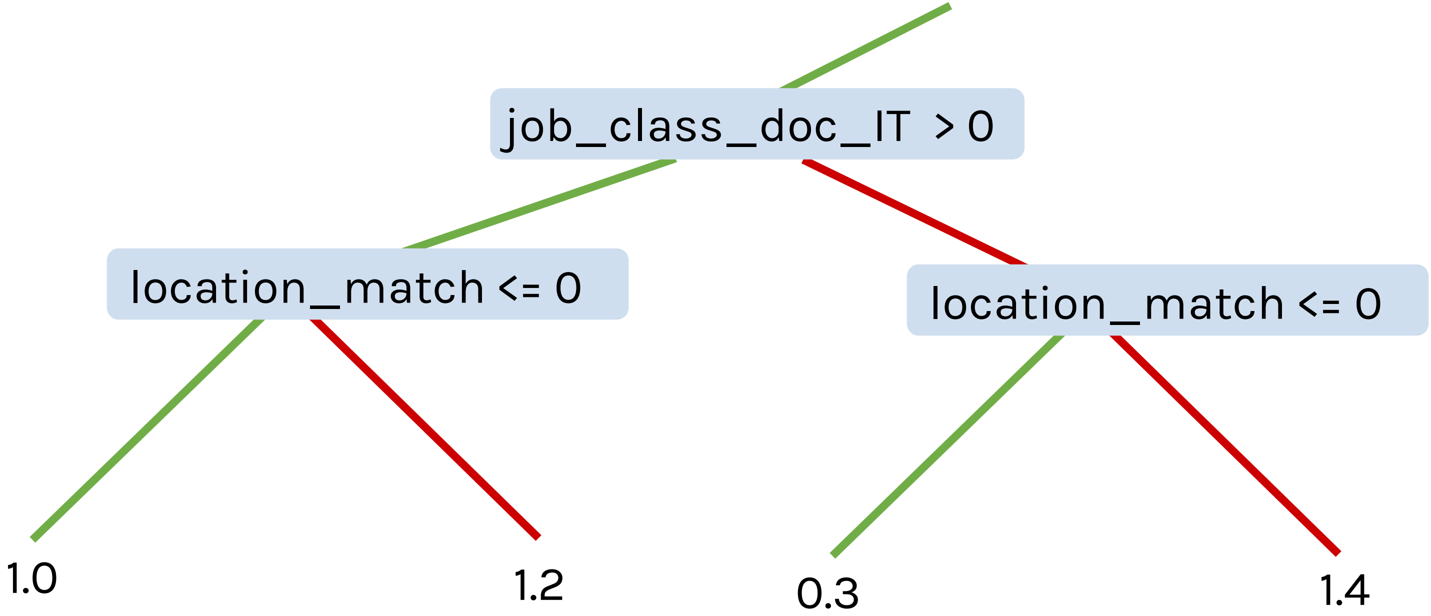
Model facet field interdependencies

Decision tree



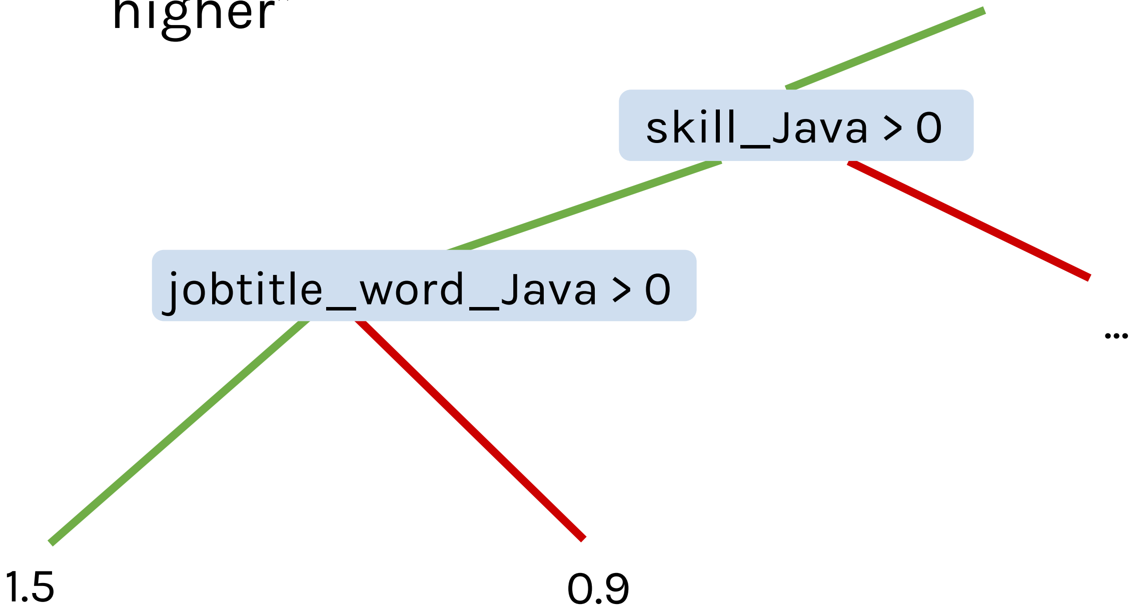
Model facet field interdependencies

- “We should weight location matches less when finding candidates in IT”



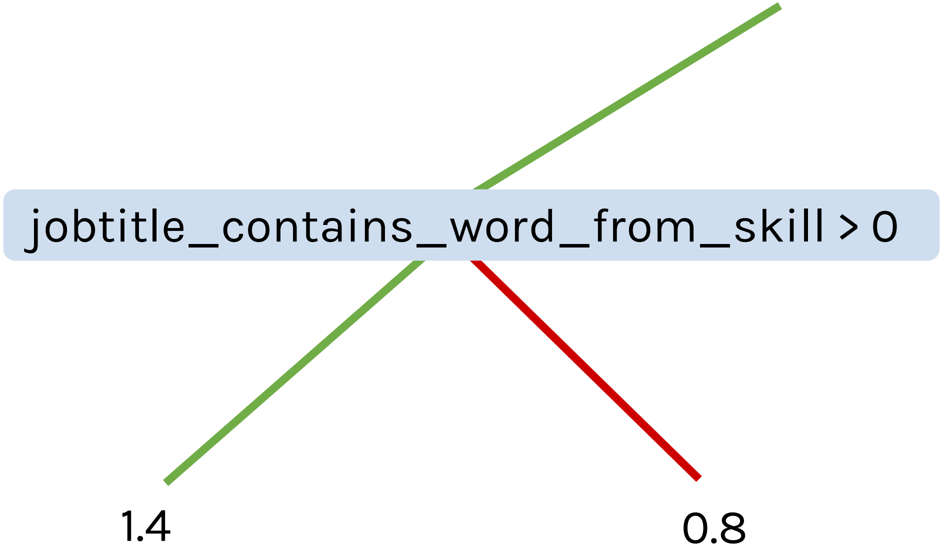
Model facet field interdependencies

- “If I search for a skill ‘Java’ I want the candidates that also have ‘Java’ in their Jobtitle field to be weighted higher”



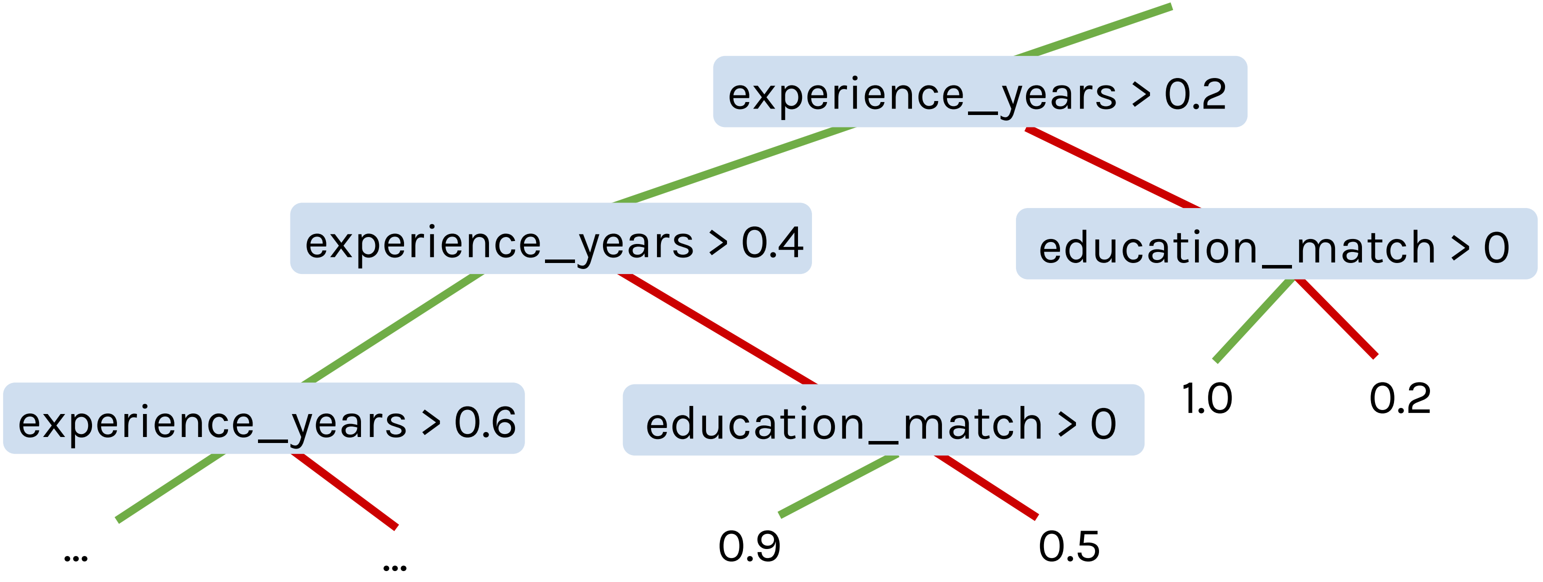
Model facet field interdependencies

- "If I search for a skill 'Java' I want the candidates that also have 'Java' in their Jobtitle field to be weighted higher"



Model facet field interdependencies

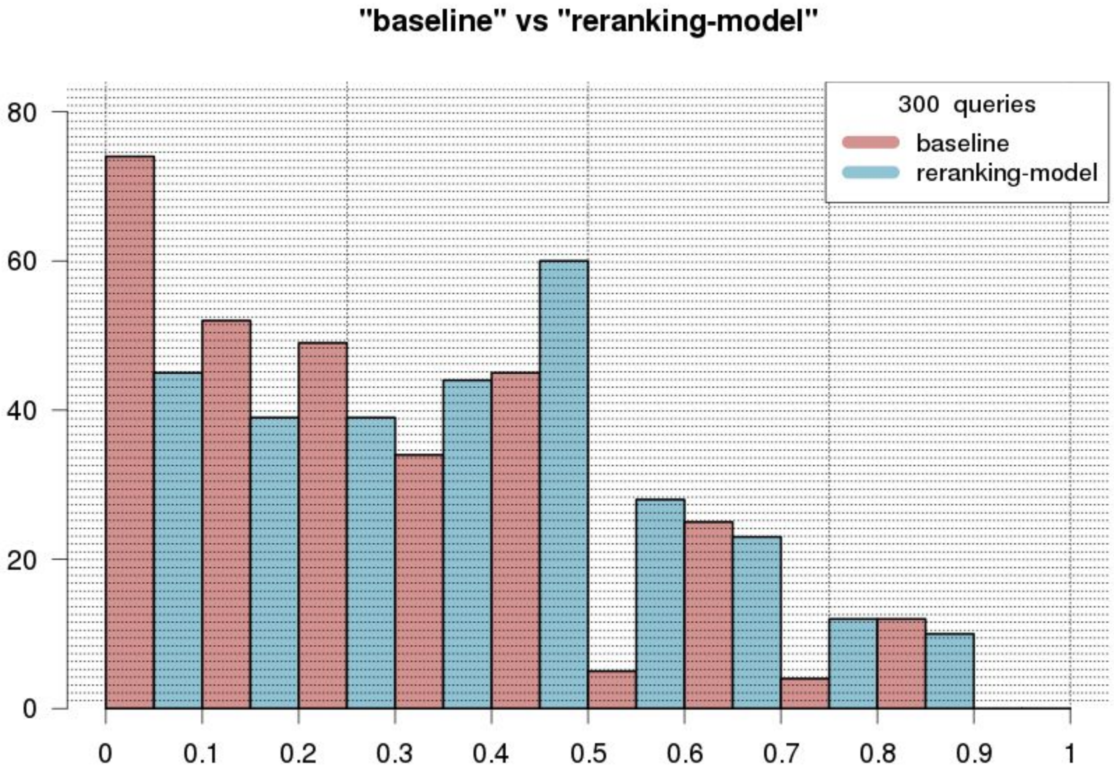
- “Education will be a less important match, the more years of experience a candidate has”



Scores

model type	algorithm	performance
Linear	Ridge regression	NDCG +6%
Decision tree	LambdaMART	NDCG +16%
Decision tree	Random Forests	NDCG +22%

Scores: risk vs. reward



Number queries

NDCG score

Execution time

- Applying reranking on top 100
  - index: 1,000,000 documents
  - model: 1000 trees, each max. 7 leaves
- Original library: **+22%**

Execution time

Culprit: transformation from internal API object to ranking-library object

(done for each query-document pair)

feature extraction

double[] features ->

String features ->

DataPoint { String relevance_label;

String query_id;

String description;

float[] features}

ranking model

Execution time

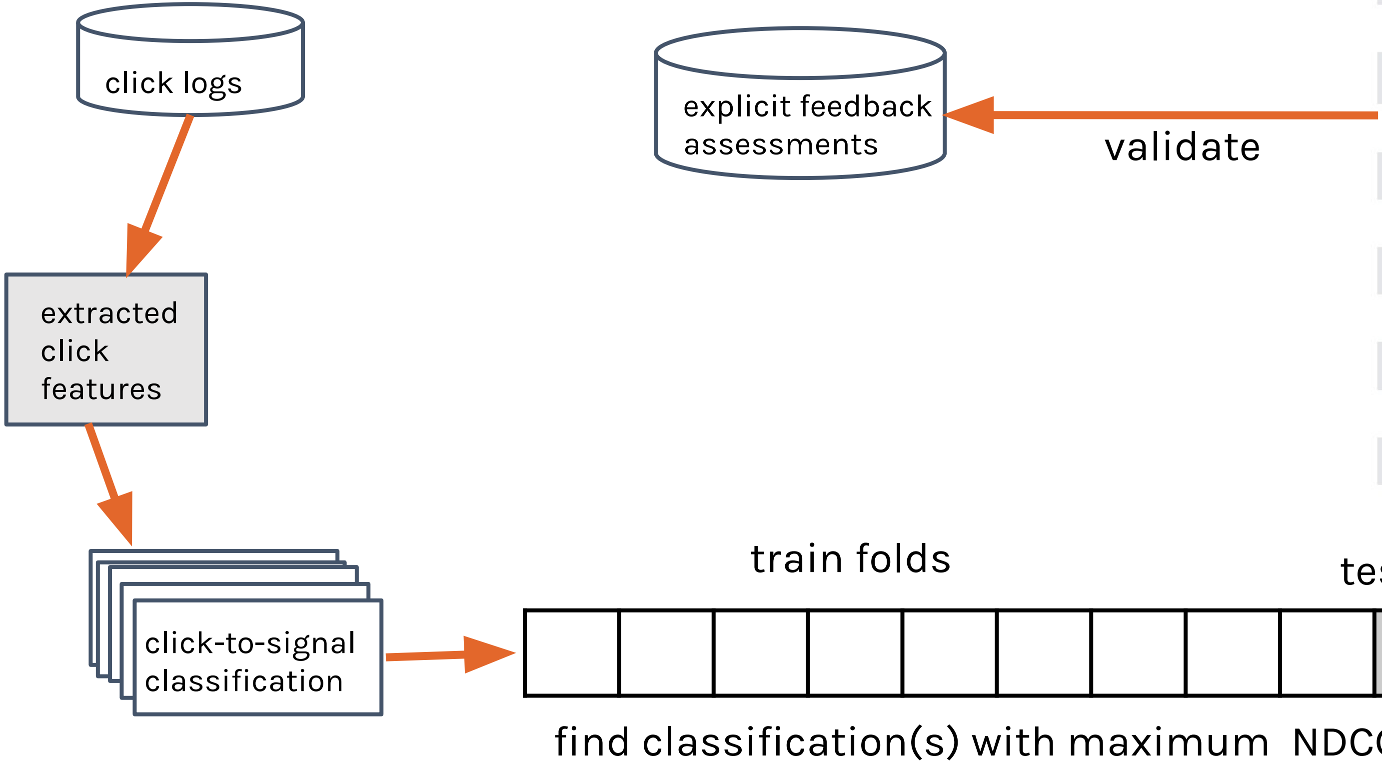
After refactoring model application

Avg. query execution time increase: **+4%**

Next steps: Implicit user feedback gathering

- Transform user actions to feedback signals
  - transformation model may differ per customer
- Avoid modeling an action loop
  - …unless you want to optimize an action
  - validate with human-made assessments
- Avoid modeling a reinforcing feedback loop
  - deal with position / selection bias

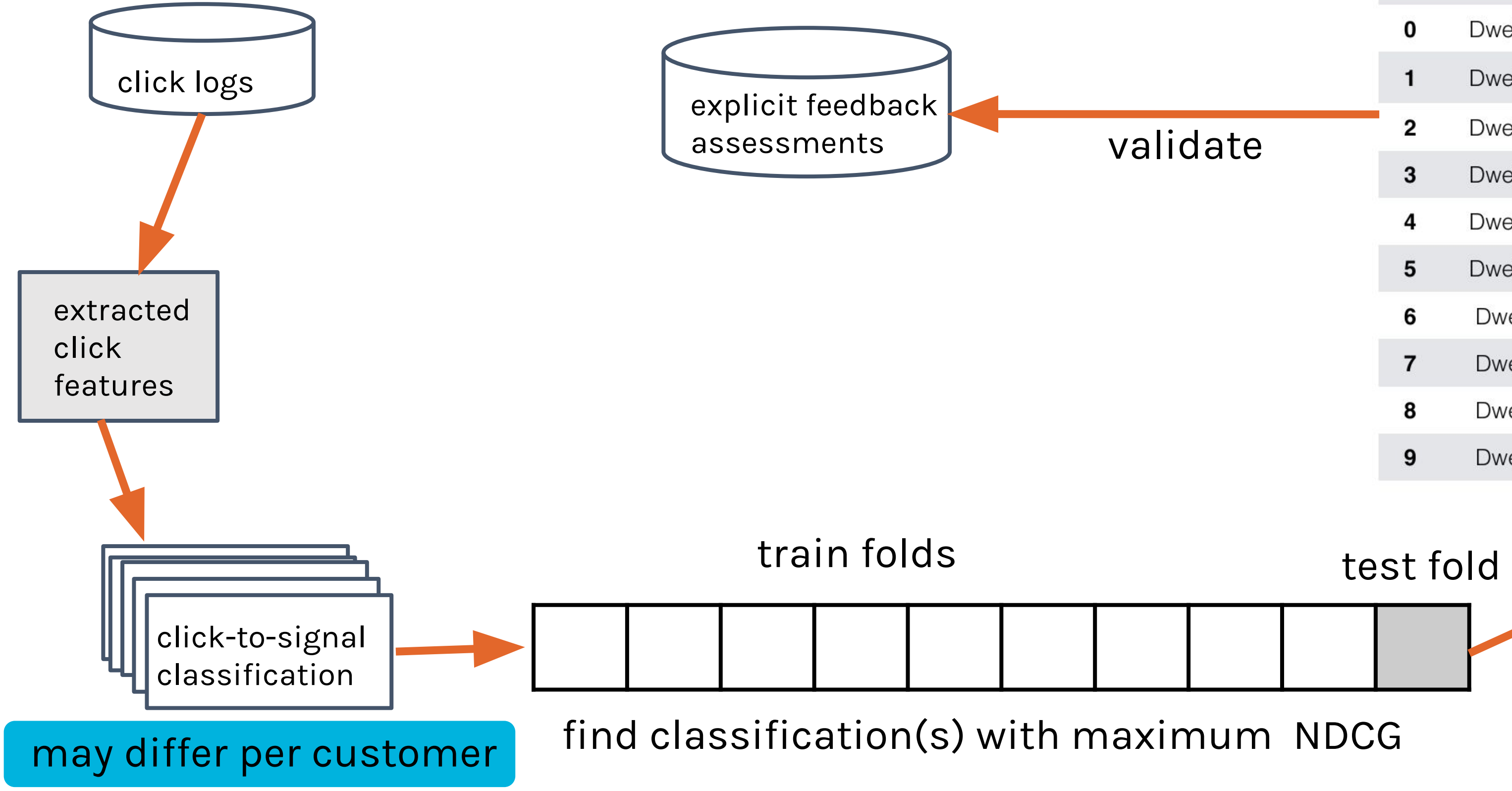
Implicit feedback gathering



NDCG

Fold	Max Classifier	Train	Test
0	Dwell=100s	0.41	0.37
1	Dwell=100s	0.42	0.28
2	Dwell=100s	0.41	0.37
3	Dwell=100s	0.41	0.42
4	Dwell=100s	0.40	0.51
5	Dwell=100s	0.41	0.37
6	Dwell=50s	0.40	0.42
7	Dwell=50s	0.41	0.38
8	Dwell=50s	0.42	0.29
9	Dwell=50s	0.40	0.45
		0.41	**0.39**

Implicit feedback gathering



NDCG

Fold	Max Classifier	Train	Test
0	Dwell=100s	0.41	0.37
1	Dwell=100s	0.42	0.28
2	Dwell=100s	0.41	0.37
3	Dwell=100s	0.41	0.42
4	Dwell=100s	0.40	0.51
5	Dwell=100s	0.41	0.37
6	Dwell=50s	0.40	0.42
7	Dwell=50s	0.41	0.38
8	Dwell=50s	0.42	0.29
9	Dwell=50s	0.40	0.45
		0.41	**0.39**

Conclusions

- Faceted search can be really improved by LTR
  - With minimal impact on execution times
- By determining your general learning objectives
  - Selecting features and algorithm accordingly and in harmony
- Ranking models aren’t static
  - Differ in performance per query type / user

Thanks!

Any questions?

contact: vanbelle@textkernel.nl

join us: textkernel.careers