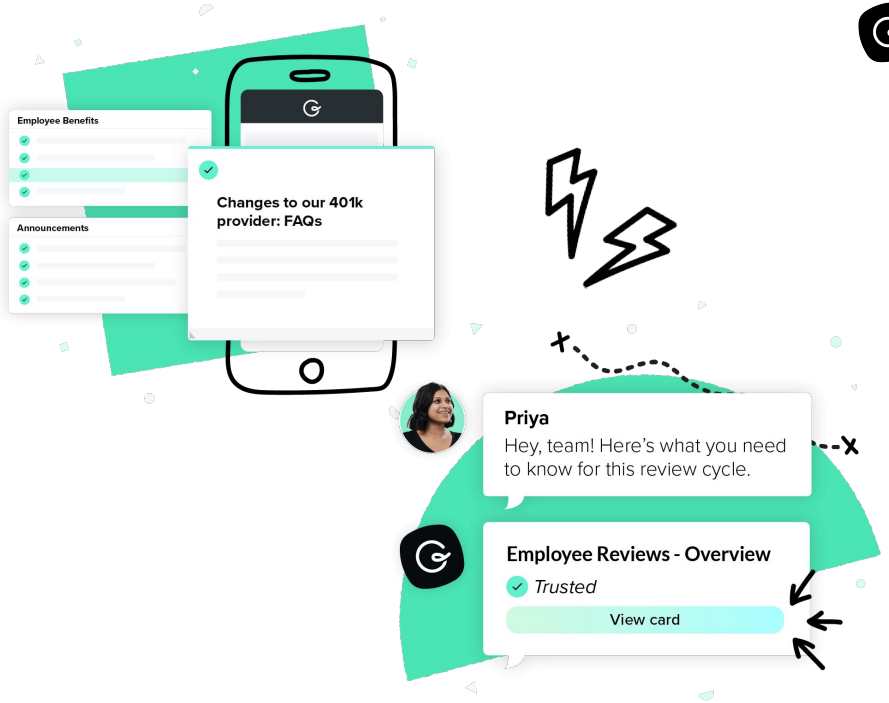
Learning to Boost

Logistic Regression to Optimize Elasticsearch Boosts

Nina Xu and Jenna Bellassai
Haystack Conference 2021

What is Guru?

Guru is a company wiki that works in your workflow.



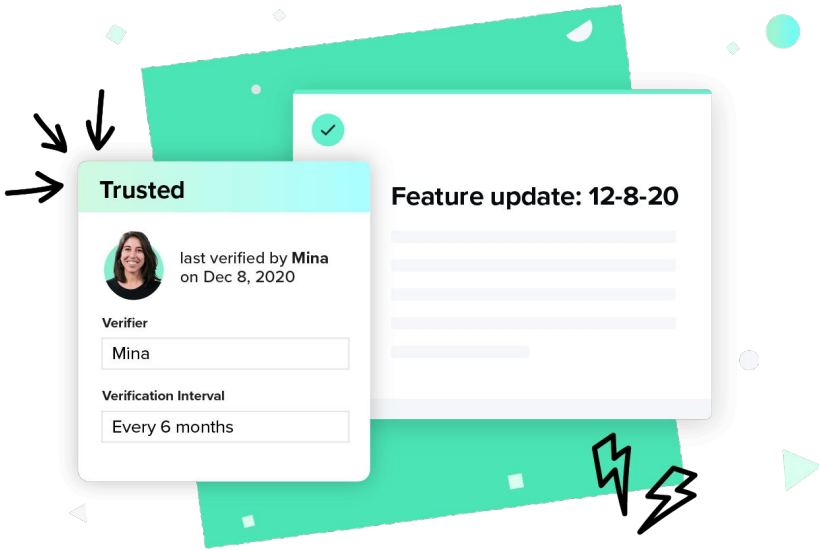
With Guru, you can asynchronously:

- Share critical product information
- Onboard employees autonomously
- Streamline internal communications

Helping the most innovative companies in the world work smarter—from anywhere.

zoom slack shopify noom GONG

Create and share trusted content



Who We Are

Search & Discovery Team

Jenna Bellassai
Data Science

Tracy Hall
FE Eng

Nabin Mulepati
ML Eng

Yev Meyer
Data Science

Adam Gruber
FE Eng

Bennett Ferris
Data Science

Jeff Plater
BE Eng (Lead)

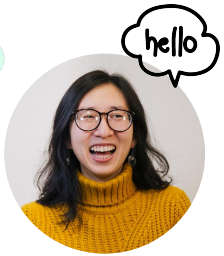
Bernie Gray
Data Science

Laura "LDB" Desmond-Black
PM

Sean Fleming
BE Eng

Michael Wangia
BE Eng

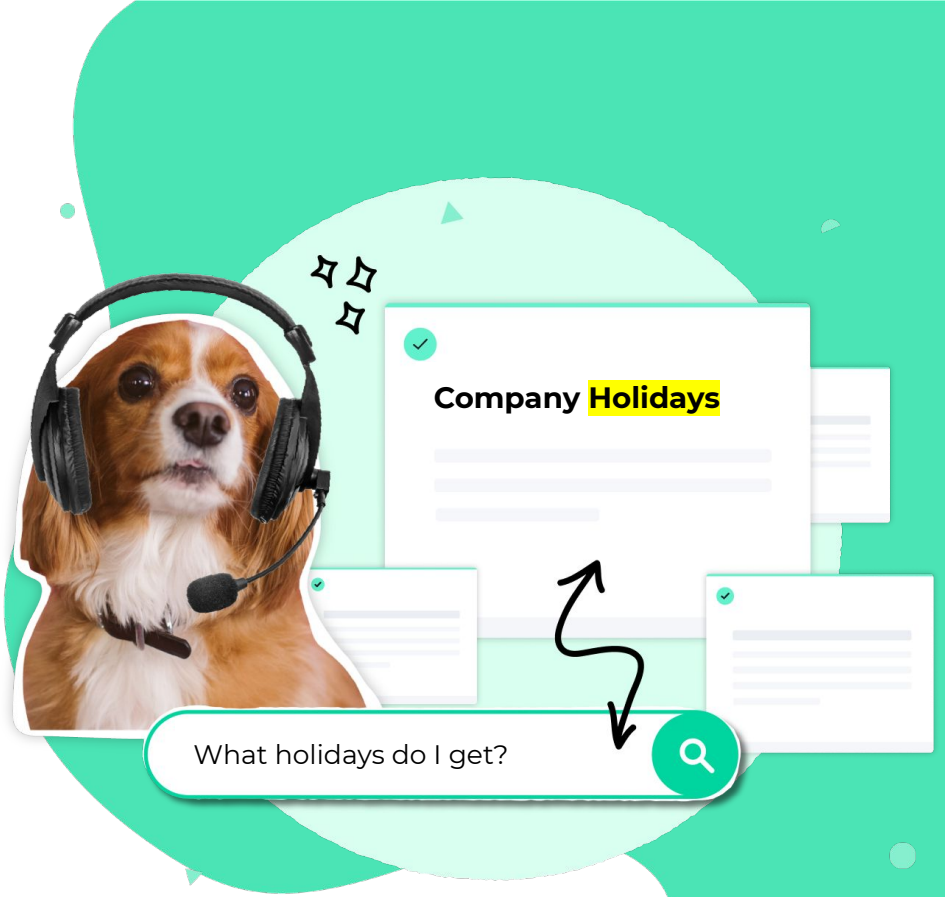
Jake Sauer
Design



Nina Xu
Data Science

Search at Guru

- Customers search for information contained in "Cards" in their own instance
- B2B use case: information is particular to each customer
- Elasticsearch



Information landscape

- Thousands of companies
  - From a variety of industries
  - Create and maintain their own documents
  - Company-specific jargons
  - Customers are the subject matter experts, not us
- Large companies
  - 10k+ documents
  - 5k+ of queries per day
- Mid-sized companies
  - ~100+ documents
  - A few queries per day

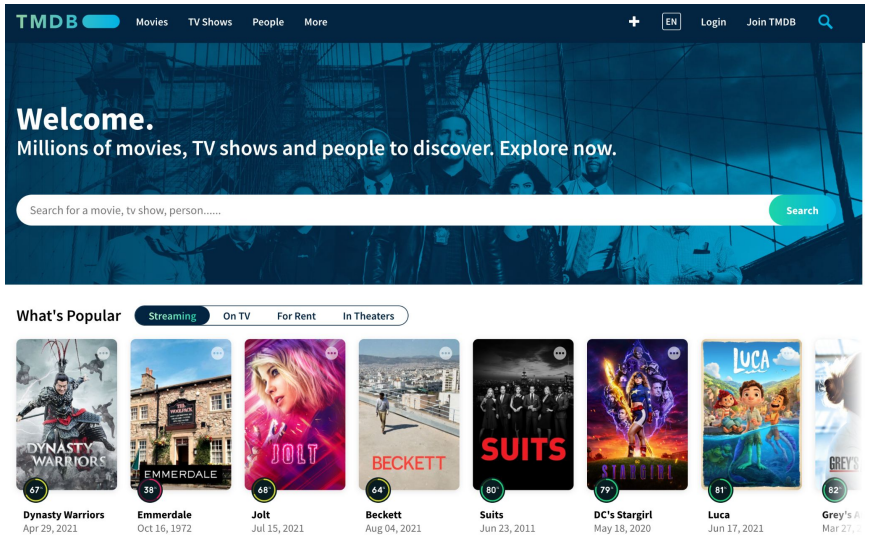
Outline

1. Problem Statement
2. Approach
3. Results
4. Special Considerations

Problem Statement

Elasticsearch query (simplified)

```
{
    "query": {
        "bool": {
            "should": [
                {
                    "match": {
                        "title": {
                            "boost": 1,
                            "query": "rocky"
                        }
                    }
                },
                {
                    "match": {
                        "overview": {
                            "boost": 0.5,
                            "query": "rocky"
                        }
                    }
                }
            ]
        }
    }
}
```



Elasticsearch query (realistic)

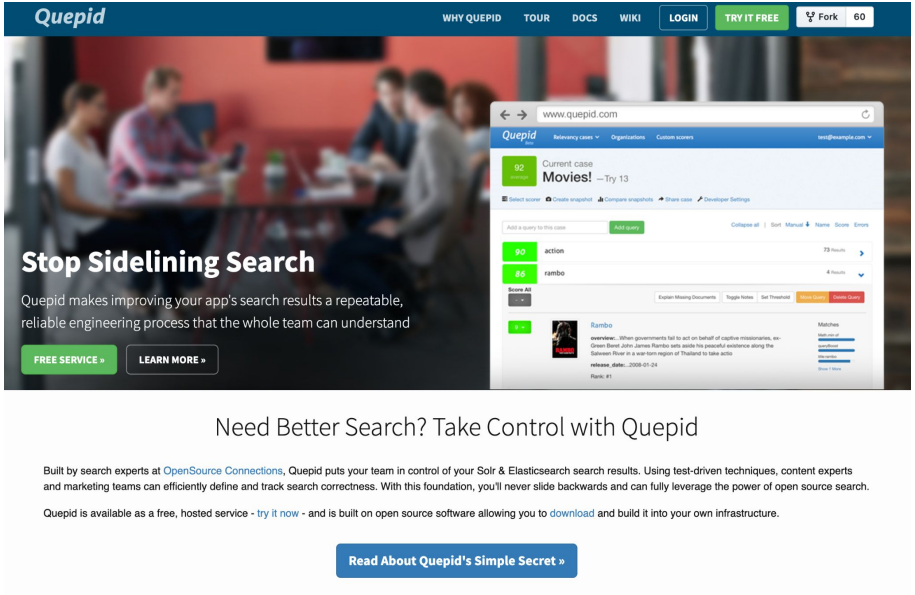
Many field boosts to tune

Query Sandbox:

```
    "bool": {
      "must": [
        {
          "bool": {
            "must": [
              {
                "bool": {
                  "should": [
                    {
                      "multi_match": {
                        "query": "rocky",
                        "fields": [
                          "title^10",
                          "title.text_std",
                          "title.text_std_gr_idx_syn",
                          "title.text_std_syn",
                          "title.text_std_idioms",
                          "cast^2",
                          "cast.text_std",
                          "cast.text_all",
                          "cast.text_people",
                          "cast.text_people_max",
                          "cast.text_people2",
                          "cast.text_people_notf",
                          "cast.text_std_gr_idx_syn",
```

Review: how to choose Elasticsearch boost values?

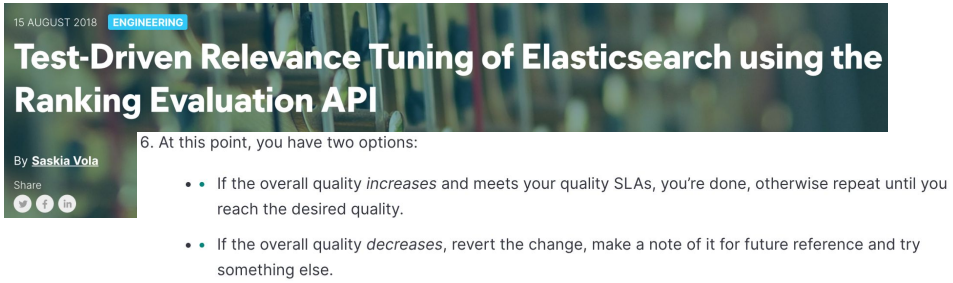
Manual tuning



- Advantages:
  - Easy to use
  - Guided by search metrics
- Limitations:
  - Requires explicit relevance judgements
  - Requires some domain expertise
  - Hard to extend to multi-tenant use case

quepid.com/

Grid search


15 AUGUST 2018 ENGINEERING

Test-Driven Relevance Tuning of Elasticsearch using the Ranking Evaluation API

By Saskia Vola

Share

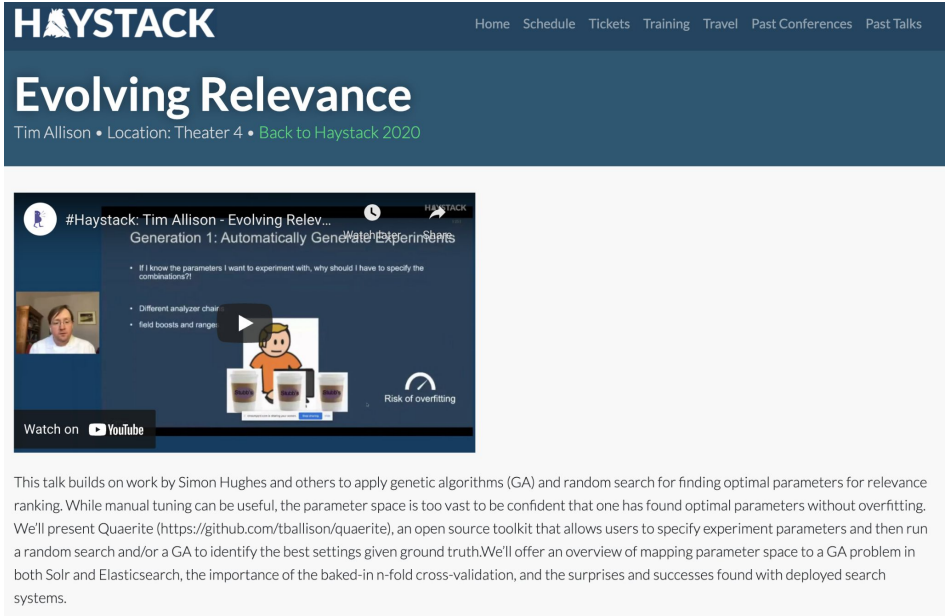
6. At this point, you have two options:

- If the overall quality *increases* and meets your quality SLAs, you're done, otherwise repeat until you reach the desired quality.
- If the overall quality *decreases*, revert the change, make a note of it for future reference and try something else.

elastic.co/blog/test-driven-relevance-tuning-of-elasticsearch-using-the-ranking-evaluation-api

- Advantages
  - More thorough than "try it and see"
- Limitations
  - Permutation explosion

Genetic algorithms

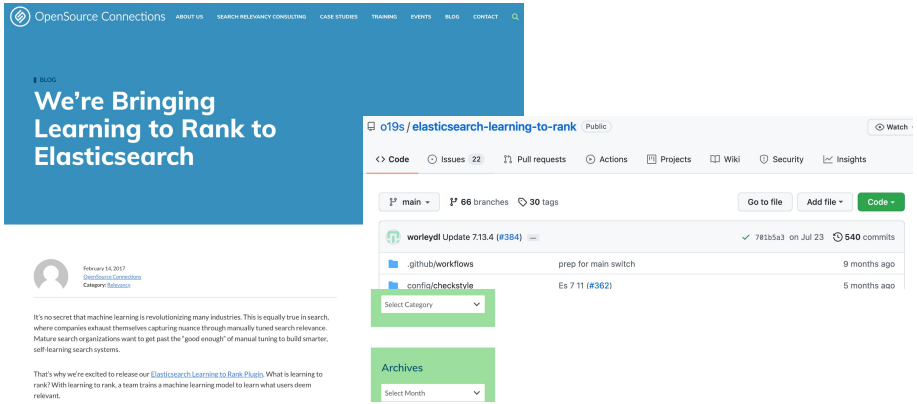


- Advantages
  - Data-driven
  - Can test many parameters besides just boosts
  - No linearity constraint (more on this later)
- Limitations
  - May not scale well with complexity

haystackconf.com/us2020/evolving-relevance/

github.com/tballison/quaerite

Learning to Rank



opensourceconnections.com/blog/2017/02/14/elasticsearch-learning-to-rank/

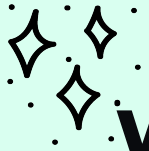
github.com/o19s/elasticsearch-learning-to-rank

- Advantages:
  - Data-driven
- Limitations:
  - High data need
  - High barrier to entry
  - Usually done at reranking step due to high computation demand

Approach

Learning to Boost - How It Works

Implementation

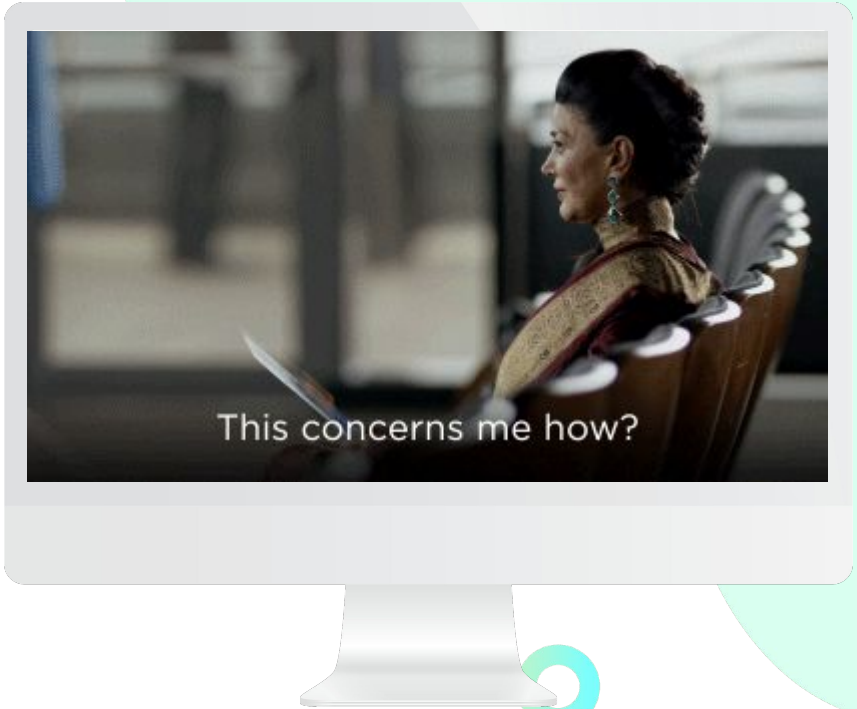


What It Is

Learning to Boost (LTB) is a logistic regression model that uses relevance judgements to determine the optimal Elasticsearch boost values for an Elasticsearch query.

Why should you care?

- Data-driven
- Easy to train
- Easy to productionize (you'll see)
- Automated for future iterations

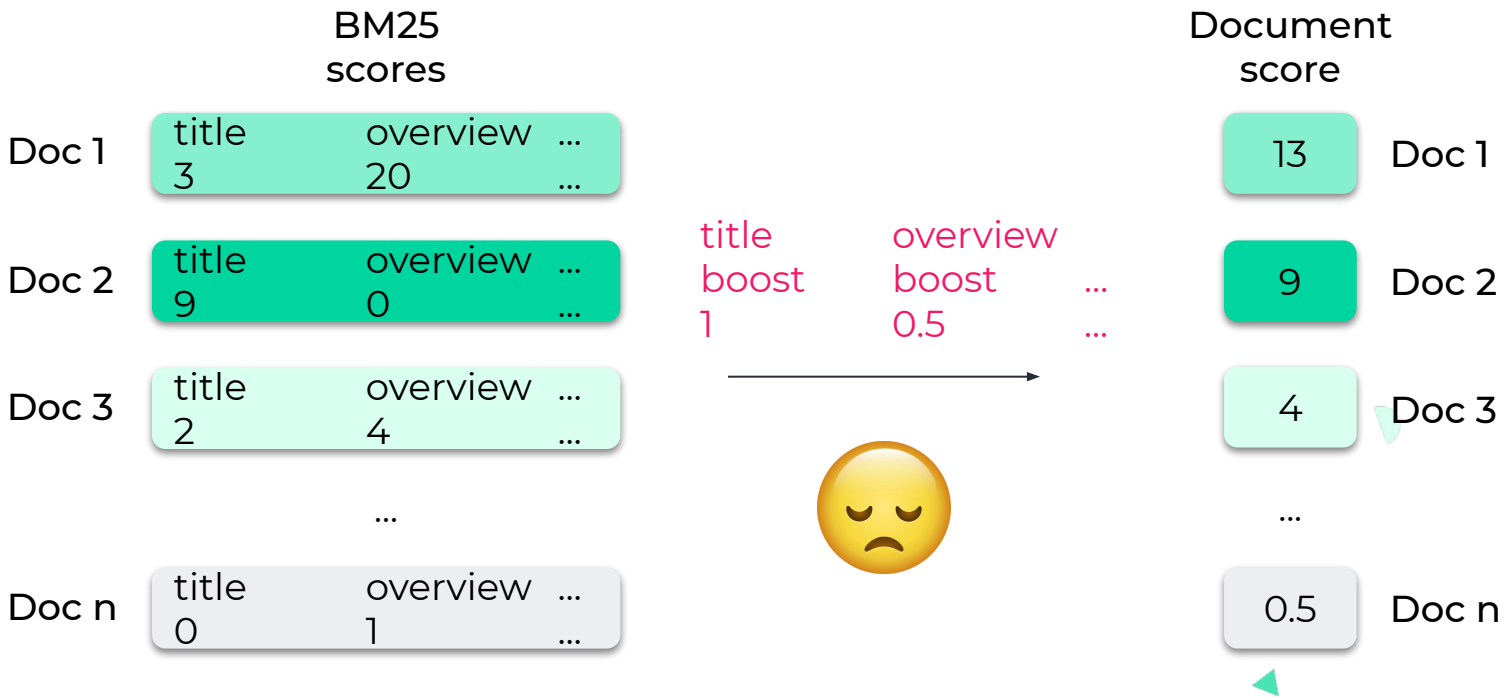


How it works - data requirement

Docs from a query

	BM25 scores			Relevance label
Doc 1	title 3	overview 20	... ...	0
Doc 2	title 9	overview 0	... ...	1
Doc 3	title 2	overview 4	... ...	0
	...			...
Doc n	title 0	overview 1	... ...	0

Elasticsearch scoring (commonly)



Optimal boost

How it works - regression coef. for boosts

Elasticsearch scoring (commonly):

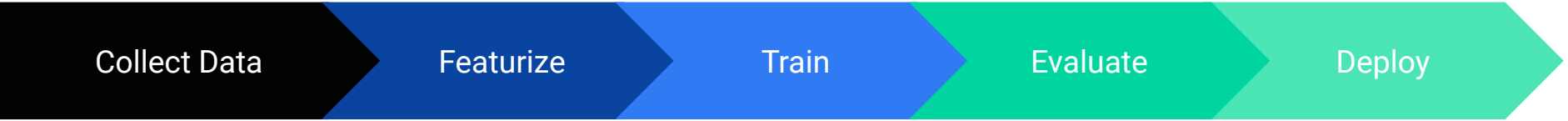
$$score = boost_1 * BM25_1 + boost_2 * BM25_2$$

Logistic regression:

$$\ell = \log_b \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

where p is the probability that a binary label is 1

Implementation



Replay past searches with all boosts set to 1. Record explanations for each result.

Transform search explanations from trial into features. Add labels according to user behavior.

Train logistic regression model.

Replay past searches using learned model coefficients as boost values. Evaluate rankings.

Deploy changes to Elasticsearch boosts in production. 😎

Implementation

Collect Data

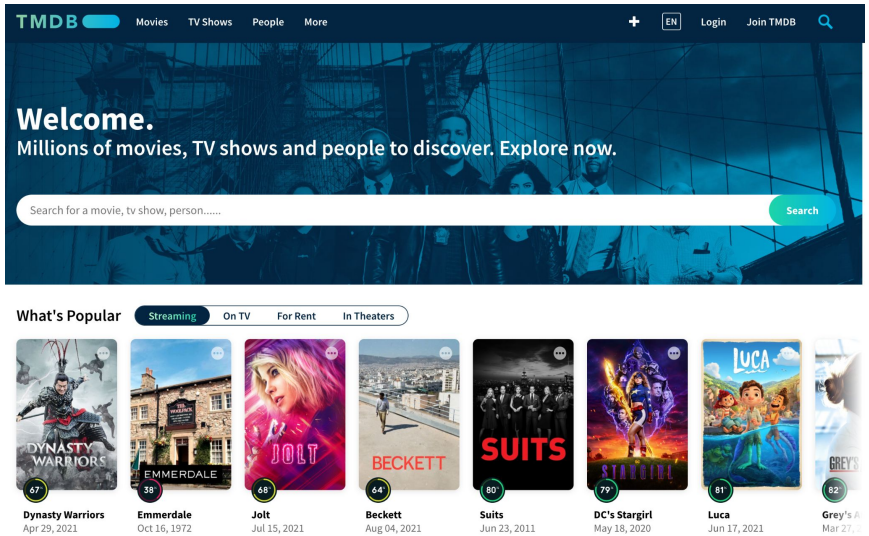
Replay past searches
with all boosts set to 1.
Record explanations
for each result.

How do we replay past searches?

- We use a homegrown framework ("offline search trials") to replay millions of past searches using a specified algorithm in an environment isolated from production.
- We compare the results of those replayed searches to the results we saw in prod.
- For Learning to Boost, an initial search trial using an Elasticsearch query with boosts set to 1 provides us with training data (search explanations and scores).

Collect Data

```
{
    "query": {
        "bool": {
            "should": [
                {
                    "match": {
                        "title": {
                            "boost": 1,
                            "query": "rocky"
                        }
                    }
                },
                {
                    "match": {
                        "overview": {
                            "boost": 0.5,
                            "query": "rocky"
                        }
                    }
                }
            ]
        }
    }
}
```



Collect Data

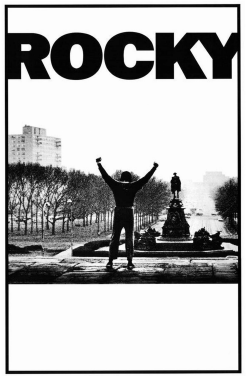
```
{
    "query": {
        "bool": {
            "should": [
                {
                    "match": {
                        "title": {
                            "boost": 1,
                            "query": "rocky"
                        }
                    }
                },
                {
                    "match": {
                        "overview": {
                            "boost": 0.5,
                            "query": "rocky"
                        }
                    }
                }
            ]
        }
    }
}
```

```
{
    "query": {
        "bool": {
            "should": [
                {
                    "match": {
                        "title": {
                            "boost": 1,
                            "query": "rocky"
                        }
                    }
                },
                {
                    "match": {
                        "overview": {
                            "boost": 1,
                            "query": "rocky"
                        }
                    }
                }
            ]
        }
    }
}
```

Collect Data

```
{
    "query": {
        "bool": {
            "should": [
                {
                    "match": {
                        "title": {
                            "boost": 1,
                            "query": "rocky"
                        }
                    }
                },
                {
                    "match": {
                        "overview": {
                            "boost": 1,
                            "query": "rocky"
                        }
                    }
                }
            ]
        }
    }
}
```

```
{
   value: 16.460304,
   description: "sum of:",
 - details: [
     - {
         value: 8.837944,
         description: "weight(title:rocki in 1335) [PerFieldSimilarity], result of:",
       - details: [
           - {
               value: 8.837944,
               description: "score(freq=1.0), computed as boost * idf * tf from:",
             - details: [
                 - {
                     value: 2.2,
                     description: "boost",
                   + details: [...]
                   },
                 - {
                     value: 6.909401,
                     description: "idf, computed as log(1 + (N - n + 0.5) / (n + 0.5))
                     from:",
                   + details: [...]
                   },
                 - {
                     value: 0.58141756,
                     description: "tf, computed as freq / (freq + k1 * (1 - b + b * dl /
                     avgdl)) from:",
                   + details: [...]
                   }
               ]
           }
       ]
     },
   - {
       value: 7.6223593,
       description: "weight(overview:rocki in 1335) [PerFieldSimilarity], result of:",
     - details: [
         - {
             value: 7.6223593,
             description: "score(freq=2.0), computed as boost * idf * tf from:",
           - details: [
               - {
                   value: 2.2,
                   description: "boost",
                 + details: [...]
                 },
               - {
                   value: 5.631858,
                   description: "idf, computed as log(1 + (N - n + 0.5) / (n + 0.5))
                   from:",
                 + details: [...]
                 },
               - {
                   value: 0.6151982,
                   description: "tf, computed as freq / (freq + k1 * (1 - b + b * dl /
                   avgdl)) from:",
                 + details: [...]
                 }
             ]
         }
     ]
 }
]
}
```



Implementation

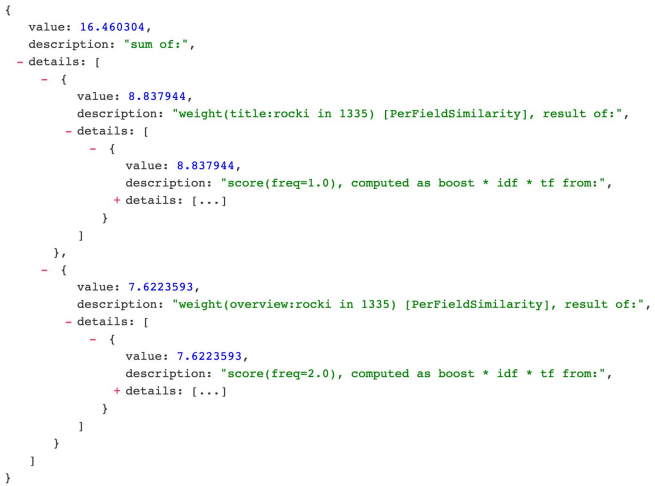
Collect Data

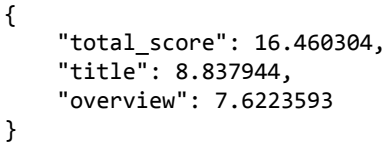
Replay past searches with all boosts set to 1. Record explanations for each result.

Featurize

Transform search scores and explanations from trial into features. Add labels according to user behavior.

Featurize





search_id	doc_id	title	overview	label
...	...	...	...	...
172	2	8.838	7.622	1
...	...	...	...	...

Implementation

Replay past searches with all boosts set to 1. Record explanations for each result.

Transform search scores and explanations from trial into features. Add labels according to user behavior.

Train logistic regression model.

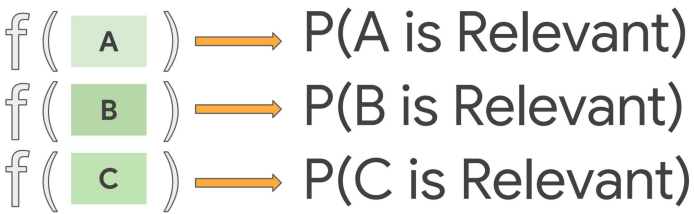
The pairwise approach

Why?

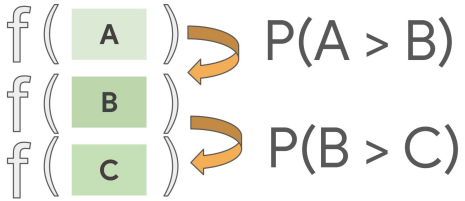
Elasticsearch scores are not comparable across queries

The pairwise approach

Pointwise LTR:



Pairwise LTR:



http://bendersky.github.io/res/TF-Ranking-ICTIR-2019.pdf

Pairwise data

How?

- Take difference of the feature values for each pair of docs from the **same query**
- Create new labels based on comparison of relevance

	BM25 scores			Relevance label		Score differences			Comparison label
Doc 1	title 3	overview 10	... ...	0	1 vs. 2	title -6	overview 10	... ...	0
Doc 2	title 9	overview 0	... ...	1	1 vs. 3				
Doc 3	title 2	overview 4	... ...	0	2 vs. 3	title 7	overview -4	... ...	1
	...			...		...			...

Logistic regression on pairwise data

Let ***i***, ***j*** denote two documents from the search result list.

From

$$l_i = \beta_1 x_{1i} + \beta_2 x_{2i}$$
$$l_j = \beta_1 x_{1j} + \beta_2 x_{2j}$$

Same boost values!

We have

$$l_i - l_j = \beta_1 (x_{1i} - x_{1j}) + \beta_2 (x_{2i} - x_{2j})$$

If $l_i > l_j$, doc ***i*** is more relevant than doc ***j***

Fit logistic regression model

Use your favorite package to make it happen!

- `pyspark.ml`
- `scikit-learn`

Make sure to restrict coefficients to non-negative values

Implementation

Replay past searches with all boosts set to 1. Record explanations for each result.

Transform search scores and explanations from trial into features. Add labels according to user behavior.

Train logistic regression model.

Run and evaluate a search trial that replays past searches using learned model coefficients as boost values.

Evaluate & Iterate

Train set
(700k+ searches)

Logistic regression

Test set
(1 mil+ searches)

Offline search trial

Ranking metrics, compared to baseline:

- MAP@k
- NDCG@k

Dev set
(80k+ searches)

Regression metrics, compared to baseline:

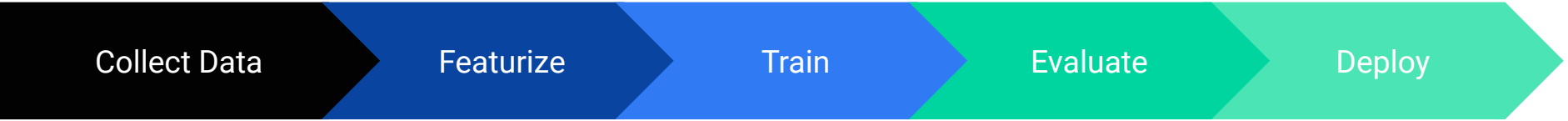
- False positive/negative rates
- AUC

Iterating ideas:

- Regularization parameters for multicollinearity & feature selection
- Normalize feature scores by query*

* eg. the normalization factor is the max feature score for each query

Implementation



Replay past searches with all boosts set to 1. Record explanations for each result.

Transform search explanations from trial into features. Add labels according to user behavior.

Train logistic regression model.

Replay past searches using learned model coefficients as boost values. Evaluate rankings.

Deploy changes to Elasticsearch boosts in production. 😎

Deploy

Just change the boost values.

What if we need to add a new field?

What if we need to add a new field?

Train set

Logistic regression

Dev set

Regression metrics, compared to baseline:

- False positive/negative rates
- AUC

Collect data and featurize with new field

Test set

Offline search trial

Ranking metrics, compared to baseline:

- MAP@k
- NDCG@k

Results

Results

Model	AUC	MAP@5
Baseline (hand-tuned)	0.897	0.332
LTB	0.911 (+1.6%)	0.336 (+1.2%)

Special Considerations

Some prerequisites (for us)

- Event tracking enabled us to obtain *implicit* judgements
- Our home-grown offline search trial framework allowed us to
  - Obtain training data
  - Replay searches with new boost values
  - Calculate search metrics

Linearity constraint

The regression approach is **useful for**

- Queries where field scores are summed together
- Identifying top-level boost values if granular-level field scores are not additive

The regression approach is **not useful for**

- the `script_score` portion of the query*
- `multi-match` queries with `best_fields` type
- `dis_max` queries with `tie_breaker`

Labeling considerations

Implicit vs. explicit judgment

- Both work
- Implicit judgment allows for bigger sample size and is suitable for the enterprise use case
- Explicit judgment is less prone to position bias

Binary vs. graded relevance

- Use logistic regression for binary labels
- Use ordinal regression for graded labels eg. 1, 2, 3, 4

Loss function vs. search metric

- The binary logistic loss (cross-entropy loss) does not take into account position of documents
- Ranking metrics in search usually consider position
- In practice, the logistic regression metrics & ranking metrics generally agree until AUC gets very close to 1

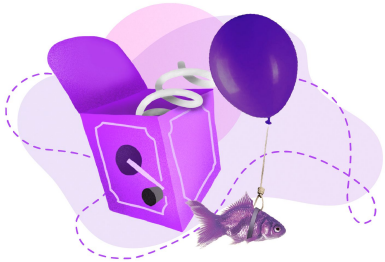
LTB is not intended to replace LTR 😊

Presentation resources

We are hiring!

getguru.com/careers

- Sr. Search Engineer
- Sr. Machine Learning Engineer
- Sr. Full Stack Engineer
- Sr. Backend Engineer
- Sr. Frontend Engineer
- ...



Don't take yourself too seriously

Seek balance. Assume good intent.

Seek and share knowledge

Be transparent. Seek diverse views for better outcomes.

Give first

Be generous to our colleagues. Give to our communities.

Based in Philadelphia, San Francisco, or remote!

Thank you!