Encores? - Going beyond matching and ranking of search results

Berlin Buzzwords 2021

Eric Pugh, René Kriegler

Who we are

Combined 30 years of experience in search

Open Source enthusiasts

ASF member, Committers on: Solr, Querqy, SMUI, Quepid,

@renekrie @dep4b

https://mices.co

Search - beyond matching and ranking

We tend to focus on matching and ranking. Other search features are almost treated like an afterthought, like ‘encores’ that follow the main performance:

- Facets
- Query auto-completion
- Spelling correction
- Query relaxation

=> BUT: These are essential features that help the user formulate the query, understand and narrow down the results

Our main act today (not encores!)

- Facets
- Query auto-completion
- Spelling correction
- Query relaxation

Learn about solutions that come out of the box (in Solr)

Typical challenges and how to overcome them

Advanced solutions: understand the concepts, create your own

The Art of Facets

Facets help the user ...

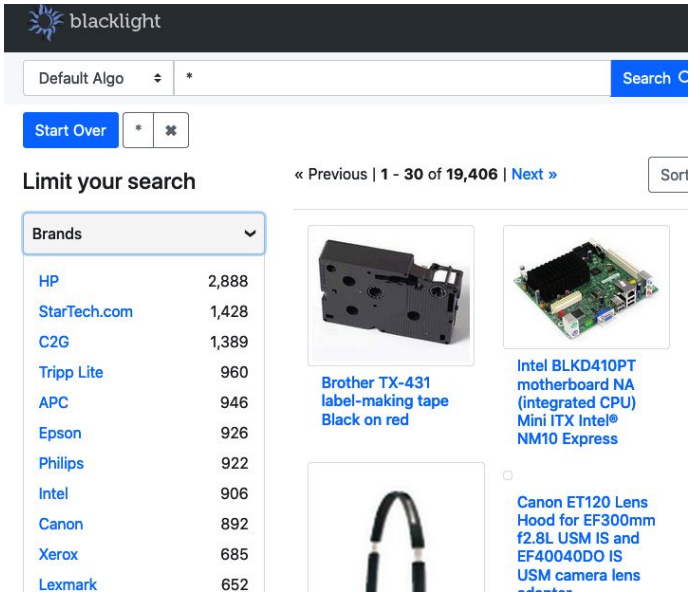
- understand the search results (see ‘what is there’, learn about the domain)
- narrow down search results

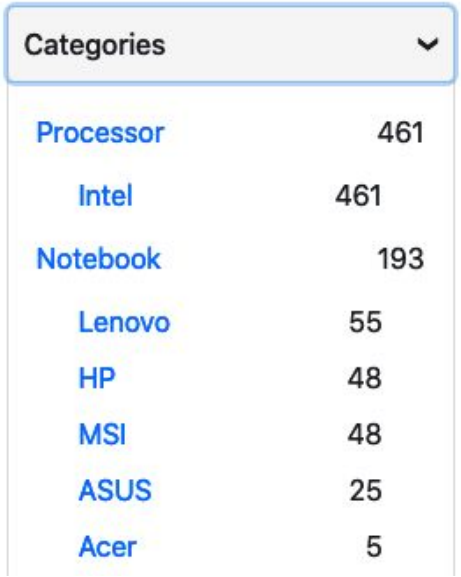
Chorus Electronics Project: https://github.com/querqy/chorus

Try the ***Demo*** Ecommerce Shop: http://chorus.dev.o19s.com:4000/

Facets help the user ...

- understand the search results (see ‘what is there’, learn about the domain)
- narrow down search results





Challenges

Getting the counts right in e-commerce search

Showing the best facets in the best order

Selecting the facet values to show

“Qui numerare incipit errare incipit”

Facet Counts

Facets and filters

A trivial example:

```
query=t-shirts

filter=color:black
```

Challenge: We still need to count all colours in the facets, even if the search result contains only black t-shirts

Solution: Tagging and exclusion of filters

Facets and filters: tagging and exclusion

Tagging:

```
fq={!tag=f_color}color:black
```

Exclusion:

Facet param

```
facet.field={!ex=f_color}color
```

JSON facets

```
"facet": {
  "color": {
    "type": "terms",
      "field": "color",
      "domain": {
        "excludeTags":"f_color"
      }
  }
}
```

Challenge: product variants

Product ID: 9739, brand: “inteemate”

Size: XS
Price: 11.99

Size: XL
Price: 11.99

Size: S
Price: 12.99

Size: S
Price: 12.99

Size: M
Price: 13.99

Size: L
Price: 13.99

Challenge: product variants

Product ID: 9739, brand: “inteemate”

Size: XS
Price: 11.99

Size: XL
Price: 11.99

Size: S
Price: 12.99

Size: S
Price: 12.99

Size: M
Price: 13.99

Size: L
Price: 13.99

color: [green, yellow, blue]
size: [XS, S, M, L, XL]
price: [11.99, 12.99, 13.99]

Merge into single document??
Facets would work great but
false matches for filter `color:green AND size:M`

Challenge: product variants

Best solution (in our opinion):

- Index one document for each variant
- Group variants at query time using the collapse query parser:

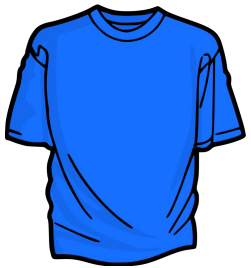
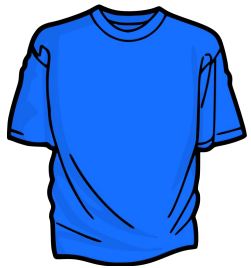
```
fq={!collapse field=productId}
```

=> Boolean filters work as expected

=> Great flexibility for counting facets

=> Fast enough

Challenge: product variants

					
Size: XS Price: 11.99 Product: 9739 Brand: inteemate	Size: XL Price: 11.99 Product: 9739 Brand: inteemate	Size: S Price: 12.99 Product: 9739 Brand: inteemate	Size: S Price: 12.99 Product: 9739 Brand: inteemate	Size: M Price: 13.99 Product: 9739 Brand: inteemate	Size: L Price: 13.99 Product: 9739 Brand: inteemate

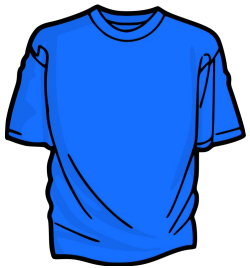
filter query
fq=brand:inteemate

Challenge: product variants

Size: XS
Price: 11.99
Product: 9739
Brand: inteemate

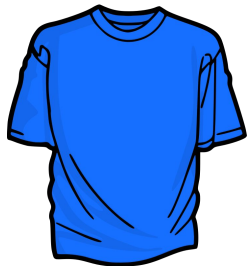
Size: XL
Price: 11.99
Product: 9739
Brand: inteemate

Size: S
Price: 12.99
Product: 9739
Brand: inteemate



Size: S
Price: 12.99
Product: 9739
Brand: inteemate

Size: M
Price: 13.99
Product: 9739
Brand: inteemate



Size: L
Price: 13.99
Product: 9739
Brand: inteemate

filter query
fq=brand:inteemate

→

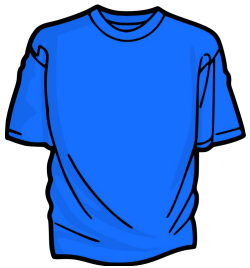
filter query
fq=color:blue

Challenge: product variants

Size: XS
Price: 11.99
Product: 9739
Brand: inteemate

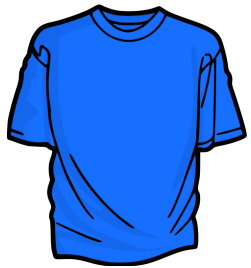
Size: XL
Price: 11.99
Product: 9739
Brand: inteemate

Size: S
Price: 12.99
Product: 9739
Brand: inteemate



Size: S
Price: 12.99
Product: 9739
Brand: inteemate

Size: M
Price: 13.99
Product: 9739
Brand: inteemate



Size: L
Price: 13.99
Product: 9739
Brand: inteemate

filter query
fq=brand:inteemate

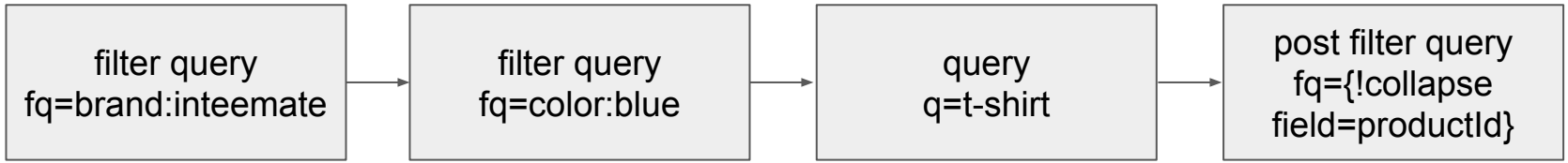
→

filter query
fq=color:blue

→

query
q=t-shirt

Challenge: product variants



Challenge: product variants in facets

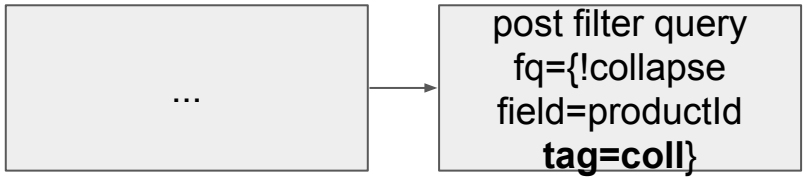
Facet counts will be correct for product attributes (“brand”)

Challenge: product variants in facets

For facet counts of variant attributes we'll have to tag and exclude collapse filter:

```
"facet": {
  "size": {
    "type": "terms",
      "field": "size",
      "domain": {
        "excludeTags":"coll"
      }
  }
}
```

Sizes S, M, L all shown as '1 result' in facets ✅

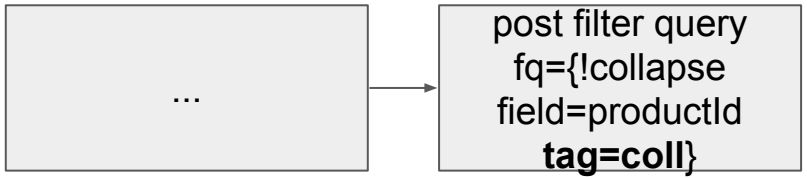


Challenge: product variants in facets

For facet counts of variant attributes we'll have to tag and exclude collapse filter:

```
"facet": {
  "color": {
    "type": "terms",
      "field": "color",
      "domain": {
        "excludeTags":"coll"
      }
  }
}
```

Sizes S, M, L all shown as '1 result' in facets ✅
Color 'Blue' shown as '3 results' in facets ❌



Challenge: product variants in facets

For facet counts of variant attributes we'll have to tag and exclude collapse filter:

```
"facet": {
  "color": {
    "type": "terms",
      "field": "color",
      "domain": {
        "excludeTags":"coll"
      },
      facet: {
        "numProducts":"unique(productId)"
      }
  }
}
```

...

post filter query
fq={!collapse
field=productId
tag=coll}

Sizes S, M, L all shown as '1 result' in facets ✅
Color 'Blue' shown as '1 result' in facets ✅

Collapse query parser - notes on implementation

- Beware of high cardinality of product IDs.
  - If you have 10M different product IDs in your index, the collapse query parser will allocate heap space for 2 arrays (float/int) x 10M elements (ca. 80 MB) per request!
  - Solution:
    - Many products have just 1 variant. It's better to leave the productId empty in this case.
    - Combine with nullPolicy=expand, which avoids reserving array space for products without a productId:

```
fq={!collapse field=productId nullPolicy=expand}
```

- All variants of a product must be indexed to the same shard

Facet Selection

Which facets should we show?

Some domains are rich in attributes. For example, electronics could use 10k different attributes.

Even if we reduced the number of attributes to be used in facets at index time, we could be left with several hundreds of candidates for facetting.

Building a request for hundreds of facets is not feasible. We'll show a simple solution, that will just use the search engine to select facets.

At the other end of the spectrum, you could train a model, that predicts which facets to show for a given query.

Which facets should we show? - Solution

Index a field multivalued field that holds the names of the facettable fields...

```
Doc1:
    screenSize: 17, ....
    facettableFields: [
        “screenSize”, “ramGB”, “height”, “width”, ...
    ]

Doc2:
    ...
    facettableFields: [
        “screenSize”, “numHDMIPorts”, “height”, “width”, ...
    ]
```

Which facets should we show? - Solution

... and execute an additional, prior facet request on this field. Add the facet values returned by this request as facet parameters to the main request:

```
"facet": {
  "facettable_fields": {
    "type": "terms",
    "field": "facettable_fields"
  }
}
```

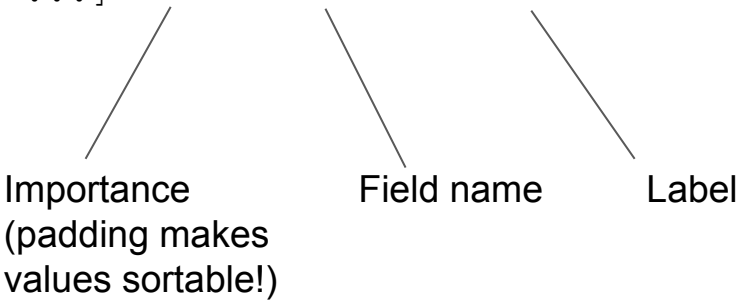
(query/filter queries are the same like in the ‘main request’)


```
"facets":{
    "facettable_fields":{
      "buckets":[{
          "val":"screenSize",
          "count":12},
        {
          "val":"ramGB",
          "count":4},
        {
          "val":"height",
          "count":3},
      ]
    }
    }
```

Which facets should we show? - Solution

Index additional information together with the names of facettable fields

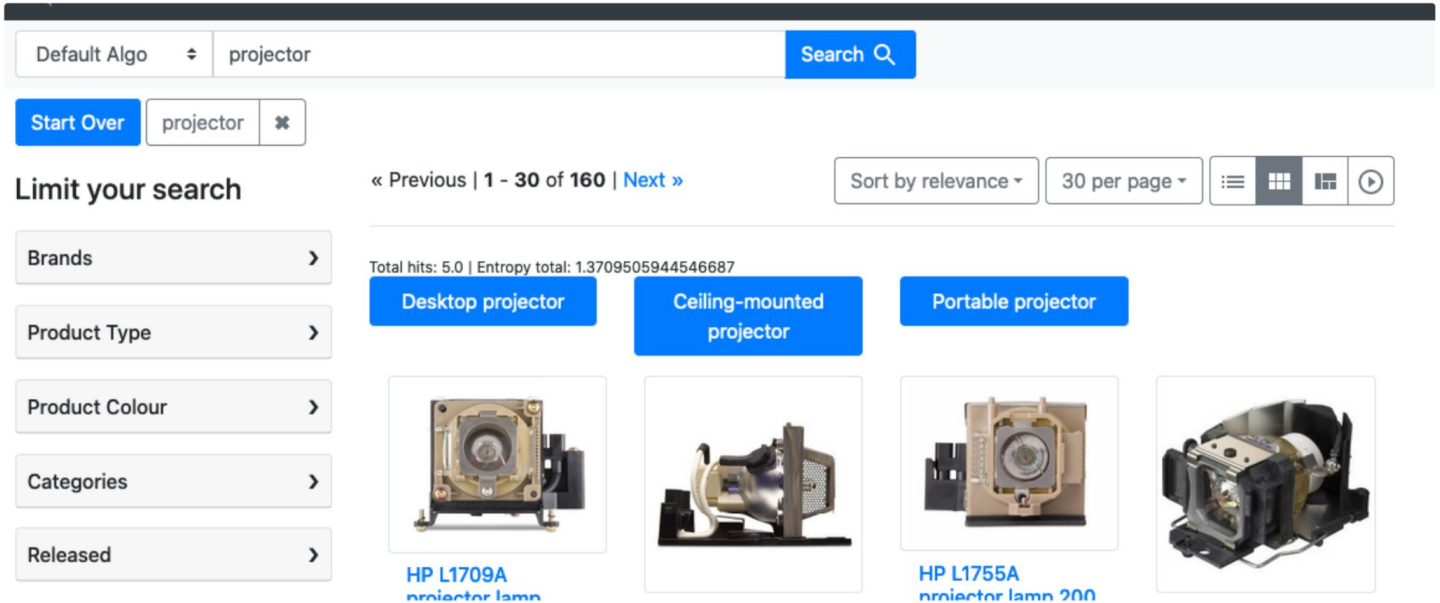
```
facettableFields: [
     "00010;screenSize;Screen size",
     "00100;ramGB;Memory (GB)",
     "00005;height;Height",
     "00005;width;Width",
...]
```



Facet Value Selection

Which facet values?

Category Pills being dynamically included IF the entropy model says they are meaningful for filtering the data



Shannon’s Entropy Worksheet

https://bit.ly/measure-diversity

$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log \mathrm{P}(x_i)$$

(https://en.wikipedia.org/wiki/Entropy_(information_theory))

Auto-completion & spelling correction

Autocompletion - Using a Suggester

```
<searchComponent name="suggest" class="solr.SuggestComponent">
  <lst name="suggester">
    <str name="name">mySuggester</str>
    <str name="lookupImpl">FuzzyLookupFactory</str>
    <str name="suggestAnalyzerFieldType">text_general</str>
    <str name="buildOnCommit">true</str>
    <str name="field">dictionary</str>
  </lst>
</searchComponent>
```

Experiment with combinations of Lookups & Dictionary implementations.


```
{
  "responseHeader":{
    "zkConnected":true,
    "status":0,
    "QTime":26},
  "suggest":{"mySuggester":{
      "coffee":{
        "numFound":5,
        "suggestions":[{
            "term":"coffee",
            "weight":80,
            "payload":""},
          {
            "term":"coffee",
            "weight":74,
            "payload":""},
          {
            "term":"coffeemaker",
            "weight":9,
            "payload":""},
          {
            "term":"coffeemaker",
            "weight":5,
            "payload":""},
          {
            "term":"coffeemachine",
            "weight":2,
            "payload":""}]}}}}
```

Spellchecking - Using Solr component

Two flavours: “cofffee --> coffee”, collations: “expresso machine” -->“espresso machine”

```
<str name="spellcheck">true</str>
<str name="spellcheck.dictionary">title</str>
<str name="spellcheck.onlyMorePopular">true</str>
<str name="spellcheck.extendedResults">true</str>
<str name="spellcheck.collate">true</str>
<str name="spellcheck.maxCollations">100</str>
<str name="spellcheck.maxCollationTries">5</str>
<str name="spellcheck.count">5</str>
<str name="spellcheck.collateParam.mm">100%</str>
```

```
<searchComponent name="suggest" class="solr.SuggestComponent">
  <lst name="suggester">
      <str name="name">mySuggester</str>
      <str name="lookupImpl">FuzzyLookupFactory</str>
      <str name="suggestAnalyzerFieldType">text_general</str>
      <str name="buildOnCommit">true</str>
      <str name="field">dictionary</str>
  </lst>
</searchComponent>
```

Good collations are what we want!


```
"spellcheck":{
  "suggestions":[
    "expresso",{
      "numFound":5,
      "startOffset":0,
      "endOffset":8,
      "origFreq":0,
      "suggestion":[{
          "word":"express",
          "freq":434},
        {
          "word":"espresso",
          "freq":44},
        {
          "word":"expression",
          "freq":17},
        {
          "word":"nespresso",
          "freq":13},
        {
          "word":"xpress",
          "freq":349}]}],
  "correctlySpelled":false,
  "collations":[
    "collation","espresso machine"]}}
```

Autocompletion - using a query index

User enters: `ja`

Show the best query completions in the best order!

Label	Match	Fingerprint	Weight	Category
jacket	jacket	jacket	1000	Fashion
	jakcet			
jack & jones	jack & jones	jackjones	800	Fashion
	jack jones			
jacket xs green	jacket xs green	greenjacketxs	300	Fashion
jacket green xs	jacket green xs	greenjacketxs	10	Fashion
jako	jako	jako	5	Fashion

Using ideas from Joshua Bacher/Christine Bellstedt, Search Suggestions - The Underestimated Killer Feature of your Online Shop. Berlin Buzzwords 2018

Autocompletion - using a query index

User enters: **`ja`**

Match prefix in field “match”

```
q=*:*&fq=match:ja
```

indexed with EdgeNGramFilter, lowercase, remove accents/ASCII folding, ...

Optionally index and match spelling variants (*jacket/jakcet*)

Label	Match	Fingerprint	Weight	Category
jacket	jacket	jacket	1000	Fashion
	jakcet			
jack & jones	jack & jones	jackjones	800	Fashion
	jack jones			
jacket xs green	jacket xs green	greenjacketxs	300	Fashion
jacket green xs	jacket green xs	greenjacketxs	10	Fashion
jako	jako	jako	5	Fashion

Autocompletion - using a query index

Sort by “weight desc”

```
q=*:*&fq=match:ja
  &sort=weight desc
```

Sorting might get slow for short prefixes if the query index is large - tag the top N queries for lengths 1 and 2 and add another filter (fewer matches to sort, nicely cacheable):

```
q=*:*&fq=match:ja
  &sort=weight desc
  &fq=top_len_2:true
```

Label	Match	Fingerprint	Weight	Category
jacket	jacket	jacket	1000	Fashion
	jakcet			
jack & jones	jack & jones	jackjones	800	Fashion
	jack jones			
jacket xs green	jacket xs green	greenjacketxs	300	Fashion
jacket green xs	jacket green xs	greenjacketxs	10	Fashion
jako	jako	jako	5	Fashion

Autocompletion - using a query index

If two queries have the same fingerprint, drop the one with the lower weight

Fingerprint: concatenated sorted, normalised query tokens

This increases the diversity of the suggestions.

Label	Match	Fingerprint	Weight	Category
jacket	jacket	jacket	1000	Fashion
	jakcet			
jack & jones	jack & jones	jackjones	800	Fashion
	jack jones			
jacket xs green	jacket xs green	greenjacketxs	300	Fashion
~~jacket green xs~~	~~jacket green xs~~	~~greenjacketxs~~	~~10~~	~~Fashion~~
jako	jako	jako	5	Fashion

Autocompletion - using a query index

Suggest the Labels as query completions!

Label	Match	Fingerprint	Weight	Category
jacket	jacket	jacket	1000	Fashion
	jakcet			
jack & jones	jack & jones	jackjones	800	Fashion
	jack jones			
jacket xs green	jacket xs green	greenjacketxs	300	Fashion
~~jacket green xs~~	~~jacket green xs~~	~~greenjacketxs~~	~~10~~	~~Fashion~~
jako	jako	jako	5	Fashion

Autocompletion - using a query index

You can show the best matching category for disambiguation and affirmation:

* ja**cket**
* jacket in Fashion

Label	Match	Fingerprint	Weight	Category
jacket	jacket	jacket	1000	Fashion
	jakcet			
jack & jones	jack & jones	jackjones	800	Fashion
	jack jones			
jacket xs green	jacket xs green	greenjacketxs	300	Fashion
~~jacket green xs~~	~~jacket green xs~~	~~greenjacketxs~~	~~10~~	~~Fashion~~
jako	jako	jako	5	Fashion

Spelling correction - using a query index

Structure similar to query index for autocompletion

Copy of the ‘match’ field indexed as n-grams

Label	Match	Fingerprint	Weight	Category
jacket	jacket	jacket	1000	Fashion
jacket	jakcet	jacket	1000	Fashion
jack & jones	jack & jones	jackjones	800	Fashion
jack & jones	jack jones	jackjones	800	Fashion
jacket xs green	jacket xs green	greenjacketxs	300	Fashion
jacket green xs	jacket green xs	greenjacketxs	10	Fashion
jako	jako	jako	5	Fashion

Spelling correction - using a query index

Filters on edit distance and rank based on n-grams (via TF*IDF)

```
q=jakc jones
&defType=edismax
&qf=match_ngram
&sow=false
&fq=match:jakc\ jones~2
```

Label	Match	Fingerprint	Weight	Category
jacket	jacket	jacket	1000	Fashion
jacket	jakcet	jacket	1000	Fashion
jack & jones	jack & jones	jackjones	800	Fashion
jack & jones	jack jones	jackjones	800	Fashion
jacket xs green	jacket xs green	greenjacketxs	300	Fashion
jacket green xs	jacket green xs	greenjacketxs	10	Fashion
jako	jako	jako	5	Fashion

Spelling correction - using a query index

Add boost by weight (or a function of it)

```
q=jakc jones
&defType=edismax
&qf=match_ngram
&sow=false
&fq=match:jakc\ jones~2
&boost=weight
```

Label	Match	Fingerprint	Weight	Category
jacket	jacket	jacket	1000	Fashion
jacket	jakcet	jacket	1000	Fashion
jack & jones	jack & jones	jackjones	800	Fashion
jack & jones	jack jones	jackjones	800	Fashion
jacket xs green	jacket xs green	greenjacketxs	300	Fashion
jacket green xs	jacket green xs	greenjacketxs	10	Fashion
jako	jako	jako	5	Fashion

General model for spelling correction & autocompletion

$$\hat{q} = \arg\max_{q \in Q} P(q|Input)$$

$$= \arg\max_{q \in Q} \frac{P(Input|q)P(q)}{P(Input)}$$

$$= \arg\max_{q \in Q} P(Input|q)P(q)$$

Noisy Channel Model / Bayesian Inference
(Kernighan et. al., 1990; Jurafsky & Martin, 2009)

Edit distance, n-gram model, keyboard layout (Symspell!), ... prefix match for autocompletion

Our ‘Weight’ field

Query relaxation

Query relaxation

Which query term should we drop if we can’t match all of them together?

jacket xs green

~~jacket~~ xs green

jacket ~~xs~~ green

jacket xs ~~green~~

iphone 12

~~iphone~~ 12

iphone ~~12~~

Query relaxation - ‘mm’ anti-pattern

Loosening ‘minimum should match’ (mm) constraint to < 100%

~~iphone~~ 12

iphone ~~12~~

You’ll get matches for “12”

She will just see probably imprecise results that don’t match her query exactly.

You cannot tell the user what happened and which term you dropped. She wouldn’t know what to do in order to improve the query.

Don’t do this!

At least not in e-commerce search

Query relaxation - Solutions

Judgment Type	Best previously seen relaxed query						Any previously seen relaxed query					
Data set	FREQ			COOC			FREQ			COOC		
Metric	P	R	F1	P	R	F1	P	R	F1	P	R	F1
0 - Drop random term	0.46	0.46	0.46	0.46	0.46	0.46	0.61	0.61	0.61	0.47	0.47	0.47
1 - Drop shortest term	0.38	0.38	0.38	0.48	0.48	0.48	0.54	0.54	0.54	0.49	0.49	0.49
2 - Drop shortest non-alphabetical term	0.52	0.05	0.09	0.45	0.04	0.08	0.55	0.05	0.09	0.46	0.04	0.08
3 - use 2, fallback to 1	0.40	0.40	0.40	0.49	0.49	0.49	0.56	0.56	0.56	0.50	0.50	0.50
4 - Drop most frequent term	0.25	0.17	0.20	0.44	0.35	0.39	0.56	0.38	0.45	0.45	0.36	0.40
5 - Drop least frequent term	0.79	0.79	0.79	0.60	0.60	0.60	0.90	0.90	0.90	0.61	0.61	0.61
6 - Drop term with highest entropy	0.29	0.27	0.28	0.43	0.41	0.42	0.45	0.43	0.44	0.44	0.42	0.43
7 - Drop term with lowest entropy	0.32	0.32	0.32	0.29	0.29	0.29	0.46	0.46	0.46	0.30	0.30	0.30
8 - keep most similar query (Word2vec)	0.82	0.81	0.82	0.61	0.61	0.61	0.91	0.90	0.90	0.63	0.62	0.62
9 - keep most similar query ('Query2vec')	0.66	0.07	0.13	0.64	0.11	0.18	0.87	0.10	0.18	0.65	0.11	0.19
10 - MNN, W2V embeddings as input	0.85	0.85	0.85	0.68	0.68	0.68	0.90	0.90	0.90	0.69	0.69	0.69
11 - like 10, plus wordshape features	**0.87**	**0.87**	**0.87**	**0.69**	**0.69**	**0.69**	**0.93**	**0.93**	**0.93**	**0.71**	**0.71**	**0.71**
12 - like 10, plus per-field DFs	0.85	0.85	0.85	0.69	0.69	0.69	0.92	0.92	0.92	0.70	0.70	0.70
13 - like 10, plus index frequency	0.77	0.77	0.77	0.62	0.62	0.62	0.86	0.86	0.86	0.63	0.63	0.63

René.Kriegler, Query Relaxation - a rewriting technique between search and recommendations. Haystack Conference 2019

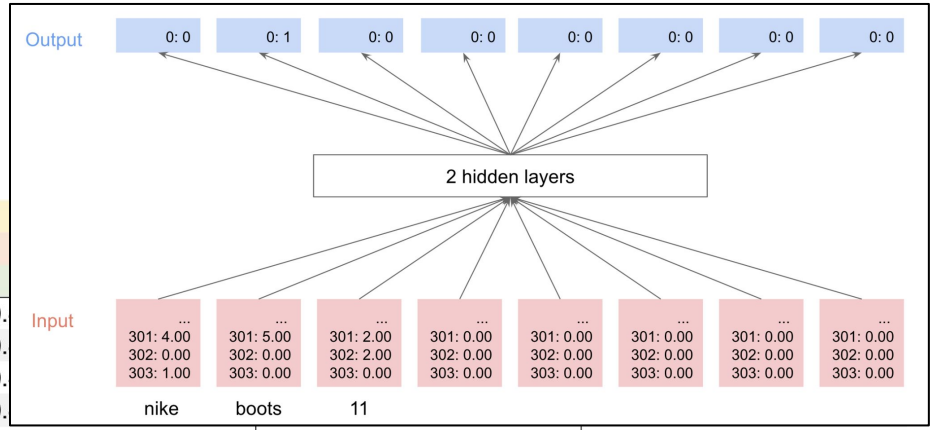
Query relaxation - Solutions

Judgment Type	Best previously seen relaxed query						Any previously seen relaxed query					
Data set	FREQ			COOC			FREQ			COOC		
Metric	P	R	F1	P	R	F1	P	R	F1	P	R	F1
0 - Drop random term	0.46	0.46	0.46	0.46	0.46	0.46	0.61	0.61	0.61	0.47	0.47	0.47
1 - Drop shortest term	0.38	0.38	0.38	0.48	0.48	0.48	0.54	0.54	0.54	0.49	0.49	0.49
2 - Drop shortest non-alphabetical term	0.52	0.05	0.09	0.45	0.04	0.08	0.55	0.05	0.09	0.46	0.04	0.08
3 - use 2, fallback to 1	0.40	0.40	0.40	0.49	0.49	0.49	0.56	0.56	0.56	0.50	0.50	0.50
4 - Drop most frequent term	0.25	0.17	0.20	0.44	0.35	0.39	0.56	0.38	0.45	0.45	0.36	0.40
5 - Drop least frequent term	0.79	0.79	0.79	0.60	0.60	0.60	0.90	0.90	0.90	0.61	0.61	0.61
6 - Drop term with highest entropy	0.29	0.27	0.28	0.43	0.41	0.42	0.45	0.43	0.44	0.44	0.42	0.43
7 - Drop term with lowest entropy	0.32	0.32	0.32	0.29	0.29	0.29	0.46	0.46	0.46	0.30	0.30	0.30
8 - keep most similar query (Word2vec)	0.82	0.81	0.82	0.61	0.61	0.61	0.91	0.90	0.90	0.63	0.62	0.62
9 - keep most similar query ('Query2vec')	0.66	0.07	0.13	0.64	0.11	0.18	0.87	0.10	0.18	0.65	0.11	0.19
10 - MNN, W2V embeddings as input	0.85	0.85	0.85	0.68	0.68	0.68	0.90	0.90	0.90	0.69	0.69	0.69
11 - like 10, plus wordshape features	**0.87**	**0.87**	**0.87**	**0.69**	**0.69**	**0.69**	**0.93**	**0.93**	**0.93**	**0.71**	**0.71**	**0.71**
12 - like 10, plus per-field DFs	0.85	0.85	0.85	0.69	0.69	0.69	0.92	0.92	0.92	0.70	0.70	0.70
13 - like 10, plus index frequency	0.77	0.77	0.77	0.62	0.62	0.62	0.86	0.86	0.86	0.63	0.63	0.63

Try searching with each term individually and drop the one from the query that yields the fewest results (might require additional rules to avoid just keeping number terms)

Query relaxation - Solutions

Multi-layer Neural Network,
Word embeddings as input to represent terms



Judgment Type	Best previously seen relaxed query											
Data set	FREQ											
Metric	P	R	F1									
0 - Drop random term	0.46	0.46	0.46	0.								
1 - Drop shortest term	0.38	0.38	0.38	0.								
2 - Drop shortest non-alphabetical term	0.52	0.05	0.09	0.								
3 - use 2, fallback to 1	0.40	0.40	0.40	0.								
4 - Drop most frequent term	0.25	0.17	0.20	0.44	0.35	0.39	0.56	0.38	0.45	0.45	0.36	0.40
5 - Drop least frequent term	0.79	0.79	0.79	0.60	0.60	0.60	0.90	0.90	0.90	0.61	0.61	0.61
6 - Drop term with highest entropy	0.29	0.27	0.28	0.43	0.41	0.42	0.45	0.43	0.44	0.44	0.42	0.43
7 - Drop term with lowest entropy	0.32	0.32	0.32	0.29	0.29	0.29	0.46	0.46	0.46	0.30	0.30	0.30
8 - keep most similar query (Word2vec)	0.82	0.81	0.82	0.61	0.61	0.61	0.91	0.90	0.90	0.63	0.62	0.62
9 - keep most similar query ('Query2vec')	0.66	0.07	0.13	0.64	0.11	0.18	0.87	0.10	0.18	0.65	0.11	0.19
10 - MNN, W2V embeddings as input	0.85	0.85	0.85	0.68	0.68	0.68	0.90	0.90	0.90	0.69	0.69	0.69
11 - like 10, plus wordshape features	**0.87**	**0.87**	**0.87**	**0.69**	**0.69**	**0.69**	**0.93**	**0.93**	**0.93**	**0.71**	**0.71**	**0.71**
12 - like 10, plus per-field DFs	0.85	0.85	0.85	0.69	0.69	0.69	0.92	0.92	0.92	0.70	0.70	0.70
13 - like 10, plus index frequency	0.77	0.77	0.77	0.62	0.62	0.62	0.86	0.86	0.86	0.63	0.63	0.63

Encores?

Facets, autocompletion, spelling correction, query relaxation are important features of a search application.

We’ve shown simple out-of-the-box solutions and a path to implement more advanced approaches.

Thank you!