Strategies for using alternative queries to mitigate zero results and their application to online marketplaces

Jean Silva (Wallapop)
René Kriegler (OpenSource Connections)

Haystack EU 2023

About us

René Kriegler

Worked in search for 16 years

Focus on e-commerce search, worked with some of Germany's top 10 online retailers

Co-Founder/-Organiser of MICES - Mix-Camp Ecommerce Search (https://mices.co)

Maintainer of Querqy library, co-initiator of Chorus project

Director E-commerce at OpenSource Connections

About us

Jean Silva

Search Engineer @Wallapop.

Working with various programming languages since 2008 and fall in love with the search world in 2014.

Jean has worked in different industries such as travel, e-commerce, and classified listings.

About us

Wallapop is the leading platform in **conscious** and **human consumption**, that aspires to create a unique inventory ecosystem of reused products.

We are present in **Spain**, **Italy** and **Portugal**.

+17M users in the South of Europe

+640M articles have found a new home in the past 10 years.

About us

Wallapop is the leading platform in **conscious** and **human consumption**, that aspires to create a unique inventory ecosystem of reused products.
We are present in **Spain**, **Italy** and **Portugal**.
+17M users in the South of Europe
+640M articles have found a new home in the past 10 years.

The Team

Search Engineers
Principal Search Engineer
Data Scientists & Data Analytics

Product Manager
Frontend Engineers
Engineering Manager

The problem

- Wallapop business domain - classified ads.
- Large amount of user-generated content, covering goods from many domains.
- Very diverse queries, locality filter, item conditions, and more.
- Focus on precision.
- Wanted to make solving/mitigating zero results a priority.

How can we improve on zero results?

Better text analysis (e.g. better stemmer, tokenization)

Apply synonyms and hyponyms (*laptop* = *notebook*; *shoes* => *trainers*)

Spelling correction (Did you mean ...? / We've searched for ...)

Content (e.g., also search in low-quality data fields)

Loosen boolean constraints (AND => OR, mm<100%)

Apply hypernyms (*boots* => *shoes*)

Query relaxation (*iphone 13* => *iphone*)

Use more distant semantic relation (*beard balm* => *trimmer*)

Show more general recommendations (related to user's shopping history, popular items)

How can we improve on zero results?

Better text analysis (e.g. better stemmer, tokenization)

Apply synonyms and hyponyms (*laptop* = *notebook*; *shoes* => *trainers*)

Spelling correction (Did you mean ...? / We’ve searched for ...)

Content (e.g., also search in low-quality data fields)

Loosen boolean constraints (AND => OR, mm<100%)

Apply hypernyms (*boots* => *shoes*)

Query relaxation (*iphone 13* => *iphone*)

Use more distant semantic relation (*beard balm* => *trimmer*)

Show more general recommendations (related to user’s shopping history, popular items)

Solve by using vector search?

Vector search looks promising for Wallapop but…

- Needs at least medium-term development.
- Lots of documents, most of them not staying on the platform for long - adding embeddings can be challenging and costly.

=> Start by using simpler approaches to mitigate zero results & come back later!

If not vector search (yet) - which other strategy?

- ✓ Better text analysis - contributed new Spanish stemmer to Lucene
- ✓ Apply synonyms and hyponyms
- ✓ Spelling correction
- ✓ Content - increase geo distance
- ✓ Loosen boolean constraints
- Apply hypernyms
- Query relaxation
- Use more distant semantic relation
- Show more general recommendations

If not vector search (yet) - which other strategy?

- ✓ Better text analysis - contributed new Spanish stemmer to Lucene
- ✓ Apply synonyms and hyponyms
- ✓ Spelling correction
- ✓ Content - increase geo distance
- ✓ Loosen boolean constraints
- Apply hypernyms
- Query relaxation (*iphone 13* => *iphone*) & maybe token replacement (audi a1 => audi a2)
- Use more distant semantic relation
- Show more general recommendations

It's four years since the 2019 talk on query relaxation - can we find better solutions? What are the opportunities from LLMs?

Alternative queries - query relaxation: intuition

Intuition:

Dropping one query term makes the query less specific but it still relates to the original query.

Produces an alternative query - interesting to the user but not necessarily matching the original intent

Query relaxation: UX

iphone 14 mini

Showing result for: iphone ~~14~~ mini

Explainable!

iphone mini

iphone 13 mini

Interactive!

Query relaxation: key problem

Which query term shall we drop?

iphone 14	iphone ~~14~~	~~iphone~~ 14	
audi a4	audi ~~a4~~	~~audi~~ a4	
purple shoes	purple ~~shoes~~	~~purple~~ shoes	
black shoes	black ~~shoes~~	~~black~~ shoes	
usb charger 12v	~~usb~~ charger 12v	usb ~~charger~~ 12v	usb charger ~~12v~~

Query relaxation: data sets and offline evaluation

We need data for offline evaluation (and maybe for training)

Foundation: pairs of long and short queries, where the short query is a subquery of the long one

"long query"	"short query"
pisos alquiler	pisos
coches baratos	coches
audi a4 avant	audi a4
apple watch	watch

Query relaxation: data sets and offline evaluation

		drop at idx	
"long query"	**"short query"**	**label**	**predicted**
pisos alquiler	pisos	1	1
coches baratos	coches	1	1
audi a4 avant	audi a4	2	2
apple watch	watch	0	1

Evaluation metrics then test whether our algorithm dropped the term at the right position in the query (accuracy) or whether the algorithm could make a prediction for a given query at all and whether it's correct (recall, precision).

Query relaxation: strategies

Strategies from Haystack US 2019 revisited

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

Query relaxation: strategies

Strategies from 2019 revisited

Obvious intuition but low coverage

iphone 15 => iphone ~~15~~

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

Query relaxation: strategies

Strategies from 2019 revisited

Good quality, easy implementation
Preferred solution if team can't ramp up M/L easily

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

Query relaxation: strategies

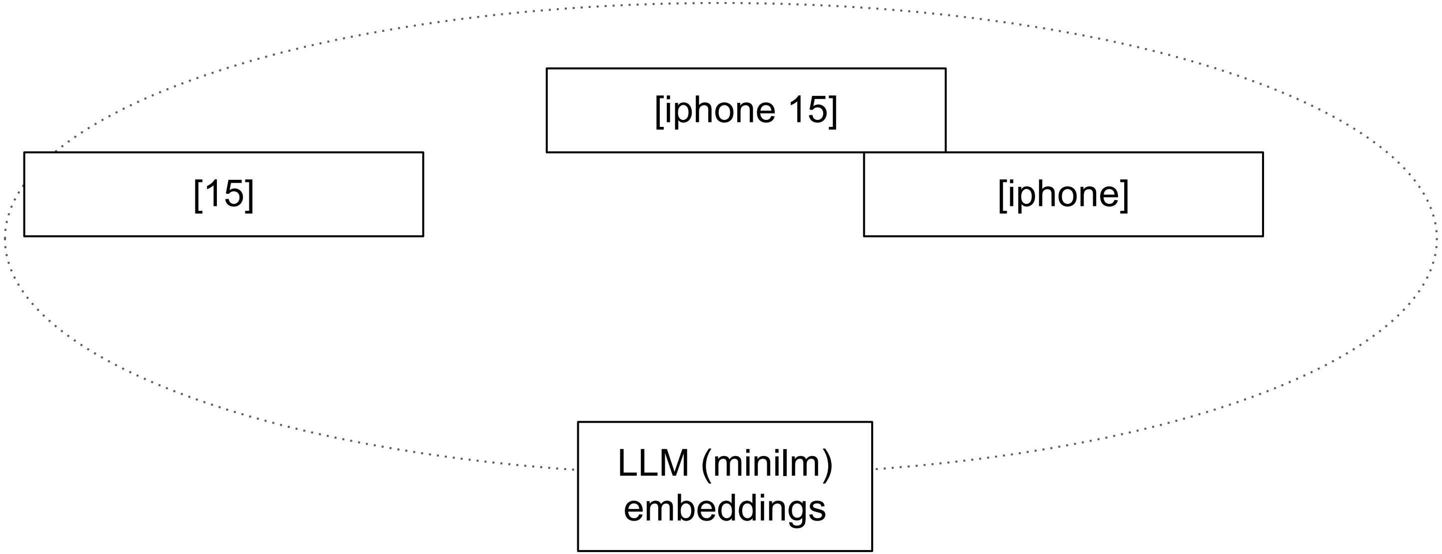
Strategies from 2019 revisited

Represent original query and relaxed query candidates by a vector embedding. Keep the candidate that is most similar to the original query (cosine).

Word embeddings: sum over word vectors
Query embeddings: train based on user sessions

Judgment Type	Best previously seen relaxed query											
Data set	FREQ											
Metric	P	R	F1	P								
0 - Drop random term	0.46	0.46	0.46	0.46								
1 - Drop shortest term	0.38	0.38	0.38	0.48								
2 - Drop shortest non-alphabetical term	0.52	0.05	0.09	0.45								
3 - use 2, fallback to 1	0.40	0.40	0.40	0.49								
4 - Drop most frequent term	0.25	0.17	0.20	0.44								
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

Strategies: Query similarity based on sequence embeddings



iphone 15
~~iphone~~ 15
iphone ~~15~~

Query relaxation: strategies

It's 2023 and LLMs have become available!

Represent original query and relaxed query candidates by a vector embedding. Keep the candidate that is most similar to the original query (cosine).

Query embeddings based on minilm

Advantage: we don't rely on embeddings for previously seen queries

Judgment Type	Best previously seen relaxed query												
Data set	FREQ												
Metric	P	R	F1	P									
0 - Drop random term	0.46	0.46	0.46	0.46									
1 - Drop shortest term	0.38	0.38	0.38	0.48									
2 - Drop shortest non-alphabetical term	0.52	0.05	0.09	0.45									
3 - use 2, fallback to 1	0.40	0.40	0.40	0.49									
4 - Drop most frequent term	0.25	0.17	0.20	0.44									
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
Keep most similar query (minilm query embedding)	0.60	0.60	0.60									

Heads-up: this is a different dataset!

OpenSource Connections wallapop HAYSTACK

Query relaxation: strategies

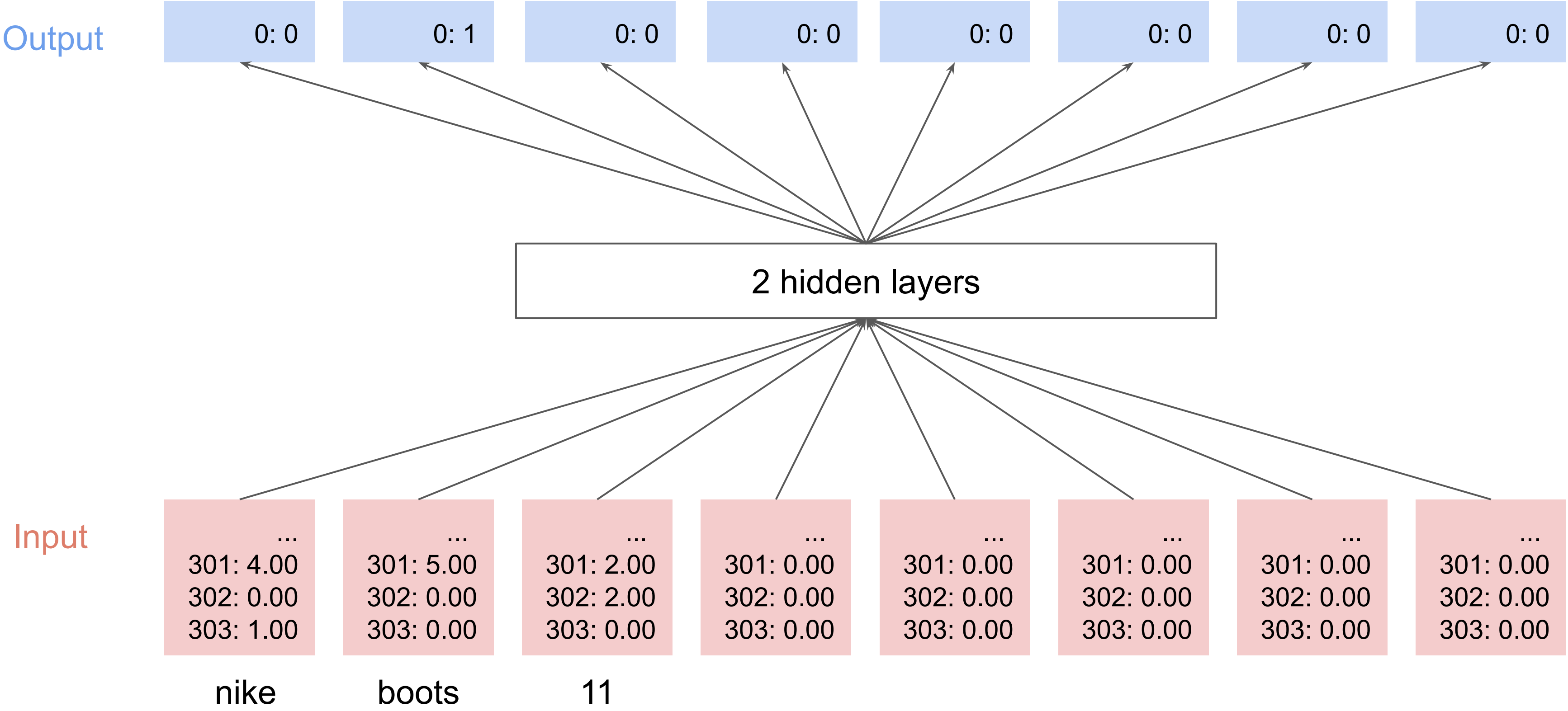
Can we beat the winner from 2019?

Judgment Type	Best previously seen relaxed query												
Data set	FREQ												
Metric	P	R	F1	P									
0 - Drop random term	0.46	0.46	0.46	0.46									
1 - Drop shortest term	0.38	0.38	0.38	0.48									
2 - Drop shortest non-alphabetical term	0.52	0.05	0.09	0.45									
3 - use 2, fallback to 1	0.40	0.40	0.40	0.49									
4 - Drop most frequent term	0.25	0.17	0.20	0.44									
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
Keep most similar query (minilm query embedding)	0.60	0.60	0.60									

Replicate the winner strategy from 2019 to the new dataset for comparison.

Multi-layer neural network with word embeddings and wordshape features as input and index of term to drop as output

MNN / Word2vec plus wordshape



(Dimensions 1-300 are word embeddings)

Query relaxation: strategies

Can we beat the winner from 2019?

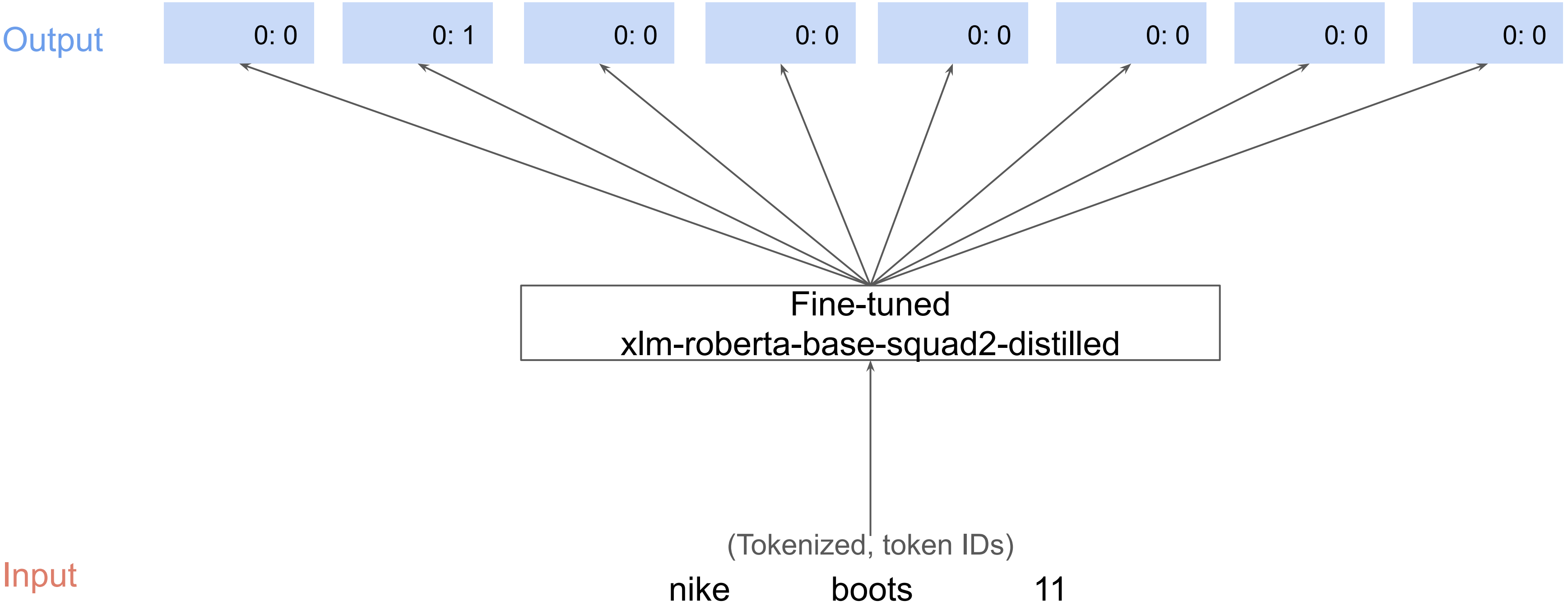
Judgment Type	Best previously seen relaxed query												
Data set	FREQ												
Metric	P	R	F1	P									
0 - Drop random term	0.46	0.46	0.46	0.46									
1 - Drop shortest term	0.38	0.38	0.38	0.48									
2 - Drop shortest non-alphabetical term	0.52	0.05	0.09	0.45									
3 - use 2, fallback to 1	0.40	0.40	0.40	0.49									
4 - Drop most frequent term	0.25	0.17	0.20	0.44									
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
Keep most similar query (minilm query embedding)	0.60	0.60	0.60									
MNN, W2V & wordshape as input	**0.98**	**0.98**	**0.98**									

Replicate the winner strategy from 2019 to the new dataset for comparison.

Multi-layer neural network with word embeddings and wordshape features as input and index of term to drop as output

- **Fixed: not ignoring query token order!**

Fine-tuned LLM to predict term to drop



OpenSource Connections wallapop HAYSTACK

Query relaxation: strategies

Can we beat the winner from 2019?

Judgment Type	Best previously seen relaxed query											
Data set	FREQ											
Metric	P	R	F1	P								
0 - Drop random term	0.46	0.46	0.46	0.46								
1 - Drop shortest term	0.38	0.38	0.38	0.48								
2 - Drop shortest non-alphabetical term	0.52	0.05	0.09	0.45								
3 - use 2, fallback to 1	0.40	0.40	0.40	0.49								
4 - Drop most frequent term	0.25	0.17	0.20	0.44								
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
Keep most similar query (minilm query embedding)	0.60	0.60	0.60									
MNN, W2V & wordshape as input	**0.98**	**0.98**	**0.98**									
Fine-tuned LLM	**0.98**	**0.98**	**0.98**									

Fine-tuned LLM to predict the term to drop

On par with winner from 2019!

- Only single model to fine-tune vs. training word embeddings and neural network
- We'll never have to deal with missing embeddings
- 0.98 means: we can only improve quality by changing the datasets that we train on

Alternative Queries: token replacement

UX: keep users engaged by providing them with alternative queries that relate to their original intent

Queries with narrow focus can be more interesting to interact with

audi a1 => audi a2 vs audi a1 => audi ~~a1~~

Token replacement: Known strategies

[audi a3]

[paper a4]

[audi a4]

Vector embeddings model

audi a3 (orig.)
audi a4
paper a4

Vector similarity between current query and known queries
String similarity (added or alone)

=> limited to known queries
=> try out generative model

Token replacement: Fine-tuned fill-mask LLM

Fill-mask model generates tokens to replace a masking token (`[MASK]`):

```
This is a new [MASK]
[MASK] is a great singer.
```

Experiment: fine-tune distilbert-base-uncased for mask-filling using ca. 400k queries that do have results

Token replacement: Fine-tuned fill-mask LLM

1) Find term to drop via query relaxation
2) Generate candidate terms to fill the gap
3) Apply some string similarity metrics between dropped and generated token, combine similarity with score from mask-filling

```
audi a1 => audi [MASK]
audi [MASK] =>
audi [MASK] => audi a2
```

```
mask_filler('audi [MASK]', top_k=10)
```

```
[{'score': 0.10849429666996002,
  'token': 23746,
  'token_str': 'tt',
  'sequence': 'audi tt'},
 {'score': 0.03976857289671898,
  'token': 12667,
  'token_str': 'rs',
  'sequence': 'audi rs'},
 {'score': 0.0217595137655735,
  'token': 1055,
  'token_str': 's',
  'sequence': 'audi s'},
 {'score': 0.0172894150018692,
  'token': 1037,
  'token_str': 'a',
  'sequence': 'audi a'},
 {'score': 0.011615365743637085,
  'token': 22441,
  'token_str': 'a2',
  'sequence': 'audi a2'},
```

Token replacement: Fine-tuned fill-mask LLM

Seems to work best for replacing model numbers

Many 0-result queries will be modified into the same target query

Maybe start with query relaxation and use alternative query for boosting?
(`audi a1 => audi OR (audi AND a2`)

Evaluation dataset could be based on per-session query modifications

Business success of queries could be modelled into the approach more easily than in query relaxation

query	relaxed_query	alt_query
pisos alquiler	pisos	pisos asturias
coches baratos	coches	coches camping
audi a3	audi	audi a2
patinete electrico	patinete	patinete antigua
coches de segunda mano	coches segunda mano	coches para segunda mano
sofas cheslong	sofas	sofas exterior
coches segunda mano particular	coches segunda mano	coches segunda mano vintage
audi a4	audi	audi a2
seat ibiza	ibiza	radio ibiza
honda civic	honda	honda cb
audi a5	audi	audi a2
armario ropero	armario	armario exterior
mercedes glc	mercedes	mercedes vito
pisos en venta	pisos venta	pisos de venta
audi q5	audi	audi tt
peugeot partner	peugeot	peugeot 206
mercedes vito	mercedes	mercedes sprinter
volkswagen polo	polo	motor polo
wolkswagen polo	polo	volkswagen polo

What we put into practice at

wallapop

Query Relaxation: Strategies

- Strategy:
  - Drop the **least frequent term** from the search query (step by step soon).
- Reason:
  - **Speed** is important. We need to operate at scale and complex machine learning models in production needs more time to implement and maintain properly.
  - **Iterative approach**. We will come back for the other **advanced techniques** later, but we need to start from somewhere.
- Limitations:
  - Document frequency is not an "always correct" approach to select tokens.
  - The least frequent terms in a query can sometimes carry important semantic meaning.

Query Relaxation: Dataset and offline evaluation

We need **data** for offline evaluation (and maybe for training)

Foundation: pairs of long and short queries, where the short query is a subquery of the long one

"long query"	"short query"
pisos alquiler	pisos
coches baratos	coches
audi a4 avant	audi a4
apple watch	watch

Found in query logs

Found in query logs

Should have results

Query Relaxation: Dataset generation

Approach to generate a dataset of "relaxed query" given a "long query" that returned no results:

- Collect you data:
  - Historical search queries that **had more than 1 token and returned results**.
  - Historical search queries that **returned no results**.

Query Relaxation: Dataset generation

- Generate your labeled dataset based on historical data.
  - Clean your input data: Lower case, remove/keep (un)wanted characters, etc…
  - Generate all combinations from the "long queries" with a missing token (These are your relaxed query candidates).
  - Find the relaxed query candidates in the historical searches that returned results (track the number of times they were searched as well).
- Eg.:
  - Long query: "iphone 14 plus".
  - Relaxed query candidates: [iphone 14, iphone plus, 14 plus]
    - Number of times the relaxed query was previously seen:
      - iphone 14: 150 times.
      - iphone plus: 1 time.
      - 14 plus: not found.

Query Relaxation: Dataset generation

- Save the dataset (as TSV):

(but you must have more, way more)

	search_term_zero_results	relaxed_query	relaxed_query_frequency	is_best	is_acceptable
0	iphone 14 plus	iphone 14	150	True	True
1	iphone 14 plus	iphone plust	1	False	False

Query Relaxation: Retrieve the Document Frequencies

- Get the document frequency of each individual token from your index.
- Save this dictionary as JSON (so we can use it as a cache later)

```
{
  "14": 210050,
  "iphone": 200000,
  "plus": 91000,
   ...
}
```

Query Relaxation: Dataset evaluation

- Generate relaxed query candidates based on the DF of the tokens.
  - Load the labeled dataset we generated earlier.
  - For each row, split the "long query" by space and for each token in the list, retrieves the document frequency.
  - Drop the term with the lowest DF (or a stop word, depending on your algorithm).
  - Add a column to the dataset called 'pred' with the 0-based position of the token to be dropped

Check the accuracy of your model: how many times does the label == pred?

Query Relaxation: Dataset generation

- Generate relaxed query candidates based on the DF of the tokens.
  - Load the labeled dataset we generated earlier.
  - For each row, split the "long query" by space and for each token in the list, retrieves the document frequency.
  - Drop the term with the lowest DF (or a stop word, depending on your algorithm).
  - Add a column to the dataset called 'pred' with the 0-based position of the token to be dropped

Check the accuracy of your model: how many times does the label == pred?

We've got ~80% accuracy.

Conclusion & Outlook

Alternative queries:

- Can be formed by query relaxation or token replacement
- Have a good explainability
- Query relaxation:
  - Quality mainly depends on what we tell the model in the training data
  - LLMs can be fine-tuned to predict the term to be dropped - overcoming limitation of unknown tokens in word embeddings
- Alternative queries:
  - Harder to test
  - We need to understand the potential better from the user perspective
  - AI opens the door for creativity in solving UX

Putting solutions into practice

- Ease of implementation and of putting solution into production beats accuracy for now - the simple solutions allows us to get user feedback quickly
- A/B test to be put into production - we'll iterate on the implementation