Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies

Bc. Ľudovít Popelka

RECOMMENDER SYSTEM

Project 2

Field of study: Intelligent software systems
Year: 1st
Course: Information retrieval
Class: Thursday 11:00
Lab lecturer: Ing. Peter Gašpar
Acad. year: 2018/2019

ASSIGNMMENT BRIEFLY

Code recommender system for customers of unknown e-shop. Utilize provided dataset to recommend top K products and calculate standard evaluation metrics. Use of recommender frameworks is not allowed. Optionally participate in Kaggle challenge.

TOOLS

Used standard python data science toolkit (**pandas, matplotlib**, …). Here only conceptual decision process is discussed. For implementation details, see Appendix A with a single jupyter notebook. Contains also setup for running code at Google Colaboratory.

Additional tools (python libs):

- **dpath**: category tree construction
- **BeautifulSoap**: HTML parsing
- **elasticsearch** & **elasticsearch_**dsl: python ElasticSearch client & query language

DATA

2 dataset files from clothing vendor:

PRODUCT CATALOG

product ID, category ID and path, brand, gender, description (HTML), price

EVENTS LOG

customer ID, product ID, type (view, cart, purchase), timestamp

PRE-PROCESSING

Exploration is markdown commented within Appendix A.

Summary:

- Descriptive statistics

Both global for each dataset, and on per attribute basis.

- Visualization

bar charts – discrete attribute value counts, missing values

continuous attributes - histogram, scatter plot, QQplot, violin plot

- Category tree

Reverse construction of full eshop category tree. For this dpath library was used. Saved in *categories.npy* file.

- Missing attributes

Not identified to be problem considering key attributes (for example, no missing product and customer IDs).

- Attribute extraction

New attributes were discovered and extracted. Namely:

  - name: h2 tag or first strong tag
  - strong: strong tags – not used later due to rare occurrence
  - features: bullet points
  - desc: polished description
  - product code –used to detect duplicates

All identified within description. However, not contained within all products. For this BeautifulSoup was used.

- Duplicate products removal

Discovered by same product code value. Confirmed by all same: brand, gender, description, price. Thus, we deal with same products within more categories.

- Reflect duplicates to events

Events log product IDs were unified for each duplicate. In this way we increased customer+product unit size in events log (= information of previous activity per customer).

Key output: **duplicates removal reduced catalog** from 28 368 to 17 423 products **(by 38.58%)**.

IMPLEMENTATION IDEA

Dump ElasticSearch (later “elastic”) with reduced product catalog and try to recommend based on similarity score for different attribute combinations. Values passed to search for each attribute are suggested from customer’s previous activity, per test customer.

The point is to measure how much elastic can improve a base recommender. Base recommender here will be recommending products solely by previous activity of customer.

To get to objective implications, elastic will be compared to trivial approach. The most trivial would be random recommender. We use here not random but best sellers for this purpose (global statistics), since overcoming random does not say anything about recommender quality – random is not realistic to be deployed (fully random, not few random recommendations of all).

RECOMMENDER FUNCIONALITY

Selected recommender options:

- K-most-viewed

For each recommended K, we can choose how big max. portion (another K-most-viewed <= K) will be selected from customers' previous activity. Here **view means all 3 event types** contained within train set, not only view product event.

- best sellers

Global best sellers.

- best sellers by gender/price

Global best sellers from filtered catalog. Gender/brand value is determined as the most frequent value in the past customer activity.

- elastic

Query elastic for selected attributes. Attribute values are determined by

  - most viewed: a product that has been the most common in customer's previous activity.
  - ideal product: combination of previously seen products (max on keywords, join on lists etc., see Appendix A)

Sort by options:

  - _score
  - purchases: purchase count

Supported fields: **brand, gender, name, features, desc** (see elastic index for types, Appendix A for queries). The later three utilize more_like_this query. Work with elastic powered by curl and python elastic client/DSL.

TESTING

Events log dataset train-test split is different for each K, where K is the number of recommended products. For each K, test set is the last K purchases (by timestamp) per customer (for customers having at least K purchases). Train set is then all activity that occurred at least 12 hours before the first test set purchase, per customer.

Estimation-validation split was skipped because:

- Number of customers with at least 10 purchases is only 100+.
- Test set here is quasi the first validation only. The second is public part of Kaggle challenge. Real, final testing of solution will be private+public part of Kaggle challenge.

EXPERIMETS & THOUGHTS

All runs filled up to K (for example if strategy provides 8 of 10 only, other 2 are considered to be wrong). This was necessary to compare consistently among different approaches.

1st series of runs (exploration, Tab. 1):

- global best sellers
- global best sellers by customer’s most viewed gender
- global best sellers by customer’s most viewed brand
- most viewed by customer
- most viewed by customer + global best sellers
- most viewed by customer + global best sellers by customer’s most viewed gender
- most viewed by customer + global best sellers by customer’s most viewed brand
- elastic match attribute of most viewed, sort by _score
- elastic match attribute of most viewed, sort by purchases match
- elastic match 2 attributes of most viewed, sort by _score
- elastic match 2 attributes of most viewed, sort by purchases
- elastic match 3 attributes of most viewed, sort by _score
- elastic match 3 attributes of most viewed, sort by purchases
- …
- elastic match all attributes of most viewed, sort by _score
- elastic match all attributes of most viewed, sort by purchases
- elastic match attribute of ideal product, sort by _score
- elastic match attribute of ideal product, sort by purchases match
- elastic match 2 attributes of ideal product, sort by _score
- elastic match 2 attributes of ideal product, sort by purchases
- elastic match 3 attributes of ideal product, sort by _score
- elastic match 3 attributes of ideal product, sort by purchases

- …
- elastic match all attributes of ideal product, sort by _score
- elastic match all attributes of ideal product, sort by purchases

Implications from the 1st series of runs:

- We choose best sellers to be elastic's rival for further comparison.
- Best sellers by gender or brand are not effective.
- Brand and gender are poor indicators alone (logical).
- desc is the best indicator alone.
- Attribute combinations increase precision in general compared to attributes alone.
- Sort by purchases decreases precision for low K but increases for high K compared to sort by _score. An interpretation can be that a more specialized query (_score) hits, but customer, understandably, does not buy more of too similar products. Next, we focus on sort by _score.
- Not all attributes are necessary to be matched separately, a best-chosen combination should be enough.
- Search by attributes combined from more products (ideal product) fails compared to search by single most viewed product. With features (list-like attribute) we observe improvement which, however, vanish when we combine with another attribute.

2nd series of runs (optimization, Tab. 2):

- selected cross field runs

Based on results from the 1st run, we decided to search cross fields. Meaning not only search 1 attribute in 1 another, but also 1:N, N:1 and N:M. Recommender was designed to support such combined queries.

- selected cross field combined with selected match runs

The combination of both query approaches, even sometimes with the same attribute in both.

Implications from the 2nd series of runs:

Now we optimized elastic recommender to approximately highest % but we still need to evaluate how well it recommends not previously seen products. The reason is that the optimized algorithm can with increasing percentage just converge to recommending previously seen products.

3rd series of runs (comparison, Tab. 3):

For each k we recommended maximum 1-k most viewed products. First without any secondary strategy – we call this a base recommender. Then we tried to improve the base recommender by different secondary strategies.

Implications from the 3rd series of runs:

Check colours at Tab. 3 before reading further. We see that the very most from recommender success was created by previously seen products. The actual recommendation value of the proposed elastic recommender is ~0.2%, which not bad considering relations exclusively among products, no previous activity. Random product selection from the catalog is ~0,00588% (1:~17000). Our approach might be an option when we strictly want to recommend based to product similarity. On the other hand, if the vendor wants just to boost volume of products sold, no matter what, a trivial approach (such as the best sellers used here) overcomes elastic significantly.

RESULTS

Not all runs are documented here, only meaningful ones. For all full outputs including total number of hits see Appendix A.

Precision tables follow:
default sort by _score
default search by most viewed

Legend:

ES = elastic	
B = brand	
G = gender	
N = name	
F = features	
D = desc	
MV = most viewed	
BS = best sellers	

k	**1**	**3**	**5**	**10**
BS	2.705%	10.784%	14.092%	17.630%
BS by G	0.986%	2.609%	3.467%	5.111%
BS by B	2.319%	2.891%	3.245%	4.074%
ES match N	3.478%	3.755%	3.218%	3.852%
ES match F	3.440%	3.057%	2.275%	2.148%
ES match D	6.493%	5.999%	5.049%	5.407%
ES match B, G	0.271%	0.681%	0.472%	1.481%
ES match B, D	6.841%	6.314%	5.326%	5.630%
ES match G, D	7.014%	5.683%	4.743%	5.259%
ES match N, D	6.493%	5.999%	5.076%	4.963%
ES match F, D	6.512%	5.982%	5.076%	5.037%
ES match B, G, sort by purchases	4.116%	5.533%	5.548%	7.926%
ES match B, D, sort by purchases	4.213%	5.550%	5.770%	6.667%
ES match G, D, sort by purchases	2.261%	4.088%	4.688%	6.444%
ES match N, D, sort by purchases	1.913%	4.104%	5.243%	5.852%
ES match F, D, sort by purchases	1.778%	3.971%	5.437%	6.444%
ES match B, F, D	6.860%	6.314%	5.381%	5.185%
ES match G, F, D	7.092%	5.666%	4.660%	4.815%
ES match B, G, D	7.382%	6.015%	4.993%	5.407%
ES match B, G, D, sort by purch.	4.947%	5.733%	5.298%	7.259%
ES match all except G	6.841%	6.331%	5.381%	5.259%
ES match all except N	7.459%	6.015%	4.910%	4.963%
ES match all except F	7.324%	5.999%	4.910%	5.481%
ES match all except F, sort by purch.	5.063%	5.749%	5.270%	7.259%
ES match all	7.440%	6.015%	4.910%	5.037%
ES match all, sort by purch.	5.159%	5.749%	5.409%	6.889%
ES match F, search by ideal product	3.246%	4.271%	4.133%	3.037%
ES match F, D, search by ideal prod.	5.121%	5.351%	4.383%	3.778%

Tab. 1: 1st series of runs

k	**1**	**3**	**5**	**10**
ES F in F	3.440%	3.057%	2.275%	2.148%
ES F in N	0.329%	0.648%	0.527%	0.296%
ES F in D	3.150%	2.941%	2.441%	2.593%
ES N in D	2.783%	3.473%	3.107%	4.074%
ES G in G, D, match D	7.014%	5.683%	4.743%	5.259%
ES G, B in G, B, D, match D	7.227%	6.281%	5.354%	7.285%
ES G, B in G, B, D, match F, D	7.285%	6.298	5.381%	5.185%
ES G, B, F in G, B, D, match D	7.227%	6.281%	5.354%	5.630%

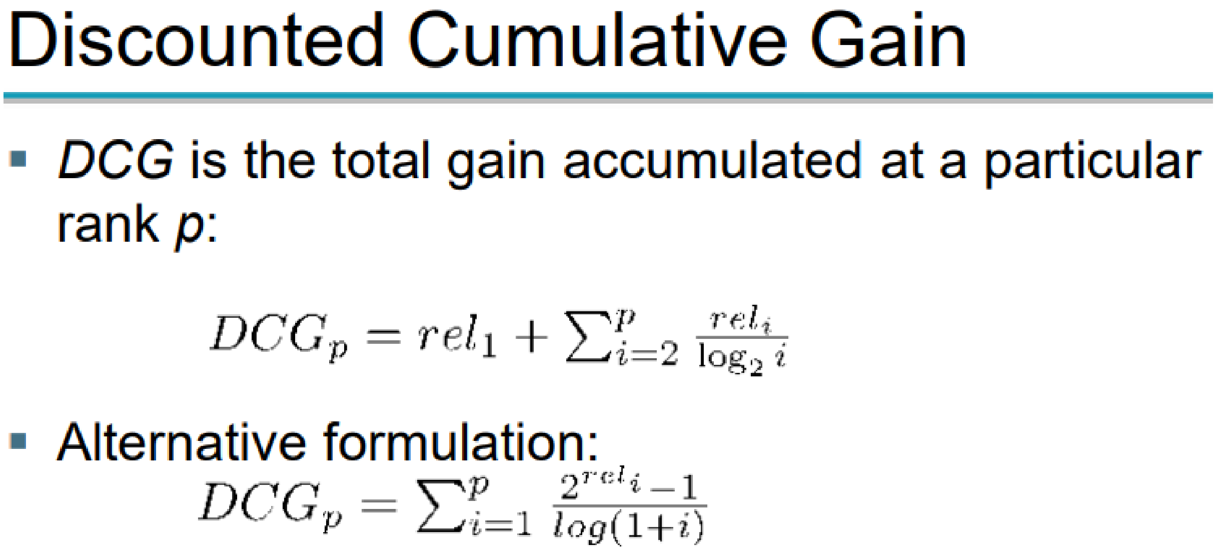
Tab. 2: 2nd series of runs

k	**1**	**3**	**5**	**10**
max. 1 MV, worst scenario	8.986%	5.849%	4.660%	4.815%
max. 3 MV, worst scenario	-	10.917%	9.570%	7.778%
max. 5 MV, worst scenario	-	-	11.789%	11.407%
max. 10 MV, worst scenario	-	-	-	14.074%
max. 1 MV, rest BS	10.647%	14.789%	16.865%	19.926%
max. 3 MV, rest BS	-	17.930%	19.723%	22.000%
max. 5 MV, rest BS	-	-	21.110%	24.519%
max. 10 MV, rest BS	-	-	-	25.852%
max. 1 MV, rest BS by G	8.986%	7.328%	7.074%	8.593%
max. 3 MV, rest BS by G	-	11.283%	10.846%	11.481%
max. 5 MV, rest BS by G	-	-	12.594%	14.741%
max. 10 MV, rest BS by G	-	-	-	14.741%
max. 1 MV, rest BS by B	8.986%	7.262%	6.574%	7.037%
max. 3 MV, rest BS by B	-	11.333%	10.541%	9.852%
max. 5 MV, rest BS by B	-	-	12.261%	13.037%
max. 10 MV, rest BS by B	-	-	-	14.963%
max. 1 MV, rest ES match D	8.986%	6.530%	5.326%	5.630%
max. 3 MV, rest ES match D	-	11.050%	9.931%	8.593%
max. 5 MV, rest ES match D	-	-	11.928%	11.778%
max. 10 MV, rest ES match D	-	-	-	14.074%
max. 1 MV, rest ES match B, G, D	8.986%	6.198%	4.993%	5.481%
max. 3 MV, rest ES match B, G, D	-	10.950%	9.736%	8.444%
max. 5 MV, rest ES match B, G, D	-	-	11.845%	11.630%
max. 10 MV, rest ES match B, G, D	-	-	-	14.074%
max. 1 MV, rest ES G, B in G, B, D, match D	8.986%	6.481%	5.354%	5.704%
max. 3 MV, rest ES G, B in G, B, D, match D	-	11.034%	9.931%	8.593%
max. 5 MV, rest ES G, B in G, B, D, match D	-	-	11.928%	11.778%
max. 10 MV, rest ES G, B in G, B, D, match D	-	-	-	14.074%

Tab. 3: 3rd series of runs (green is improvement from red)

nDCG

The recommender supports also normalized discounted cumulative gain calculation. Implemented based on Fig. 1.



Discounted Cumulative Gain

- *DCG* is the total gain accumulated at a particular rank *p*:

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i}$$

- Alternative formulation:

$$DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{log(1+i)}$$

Fig. 1: DCG formula (source: Stanford Intro to IR course)

We calculated nDCG just on some best elastic configurations (Tab. 4) and did not go to deeper exploration.

Customers do not rank products, so we come up with a following ranking function.

2	for the first product (the earliest timestamp) from the test set that customer really bought
1	for the last product (the latest timestamp) from the test set that customer really bought
1-2	for the rest of products from the test set that customer really bought (proportionally in time)
0	for recommended but not actually bought product

k=10	**precision**	**nDCG**
ES match D	5.407%	3.512%
max. 5 MV, rest ES match D	11.778%	7.094%
ES G, B in G, B, D, match D	5.630%	3.812%

Tab. 4: nDCG of selected runs

RECOMMENDER CATEGORIZATION

elastic similarity is purely content-based approach. So is analysing previous product activity per customer. However, we also rely on best sellers which is global statistics. That is why the proposed recommender is hybrid.

CONCLUSION

ElasticSearch is a great tool providing fast search but not universal similarity score for all kinds of our problems. The effective product-based recommender system should be based on in-depth, data driven, statistically supported decisions. In this manner, ElasticSearch can provide us with an additional help, however, cannot to serve as core of recommendations.

Surprisingly and ultimately, a basic more_like_this query on product description, cleaned from HTML elements, achieved the highest recommendation power, leaving super-complicated cross-field plus extracted attribute combinations behind.

KAGGLE CHALLENGE

We submitted several configurations from above. The generating scripts are archived in Appendix A.

Observed an inconsistent behaviour. Consider following example (2 recommended, ordered lists):

1. red socks, 2. blue shirt, 3. yellow T-shirt
2. red socks, 2. red socks, 3. blue shirt

The former should score higher even if red socks are preferred by customer. After submission, the latter (of course a bigger volume) scores more than double. Thus, we believe proposed Kaggle tester does not pay attention to duplicates.

Finally, we do not understand the necessity to achieve highest score in the challenge, since both collaborative and content-based recommenders have different, own qualities. Moreover, the challenge evaluation probably does not recognize duplicates.

BIBLIOGRAPHY

elastic.co. Elasticsearch Reference [online]. Available from: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html

Maher Malaeb. Recall and Precision at k for Recommender Systems. In Medium [online]. 2017. Available from: https://medium.com/@m_n_malaeb/recall-and-precision-at-k-for-recommender-systems-618483226c54

elasticsearch-py.readthedocs.io. Python Elasticsearch Client [online]. Available from: https://elasticsearch-py.readthedocs.io/en/master/

elasticsearch-dsl.readthedocs.io. Elasticsearch DSL [online]. Available from: https://elasticsearch-dsl.readthedocs.io/en/latest/

kaggle. VI Challenge [online]. Available from: https://www.kaggle.com/c/vi-challenge-2018

PyPI. dpath [online]. Available from: https://pypi.org/project/dpath/

Leonard Richardson. Beautiful Soup Documentation. In crummy.com [online]. Available from: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Google Research. Google Colaboratory [online]. Available from: https://colab.research.google.com

Stanford. Evaluation [online] In: Introduction to Information Retrieval. Available from: https://web.stanford.edu/class/cs276/handouts/EvaluationNew-handout-6-per.pdf

APPENDIX

A. jupyter notebook