Politehnica University of Timisoara

Laboratory reports

Authors:
Marius Olariu
Tudor Stefan

Supervisor:
Dr. Cosmin Cernăzanu

Universitatea Politehnica Timișoara

Lab 2

0.1 Non-programming based

Ranking functions tunning

In order to solve this task we found out how the ranking functions work, the parameters they take and some information about their range. Afterwards we wrote python scripts that edited *config.toml* (wrote different values for the input parameters of a ranking function) and launched the *competition*. Based on the output, the MAP value, and input parameters we plotted a figure and observed for which input parameters values gave the biggest value of MAP. The figures can be found below. We tested the following functions: **BM25**, **Jelinek-Mercer**, **Dirichlet-Prior** and **Pivoted-Length**.

Ranking function	Input Param.	MAP
BM25	$k_1 = 1.7, b = 1, k_3 = insignificant$	0.508
Jelinek-Mercer	$\lambda = 0.73$	0.509
Pivoted-length	s = 0.13	0.509
Dirichlet-prior	$\mu = 0.0001$	0.240

Table 1: Ranking functions and their best score

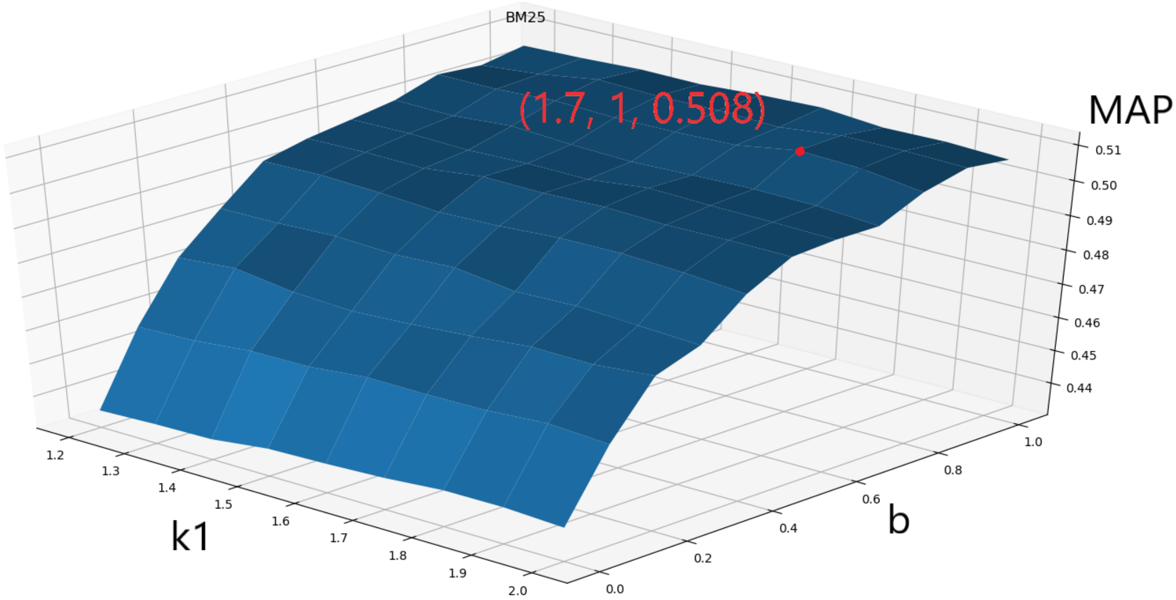

Figure 1: BM25

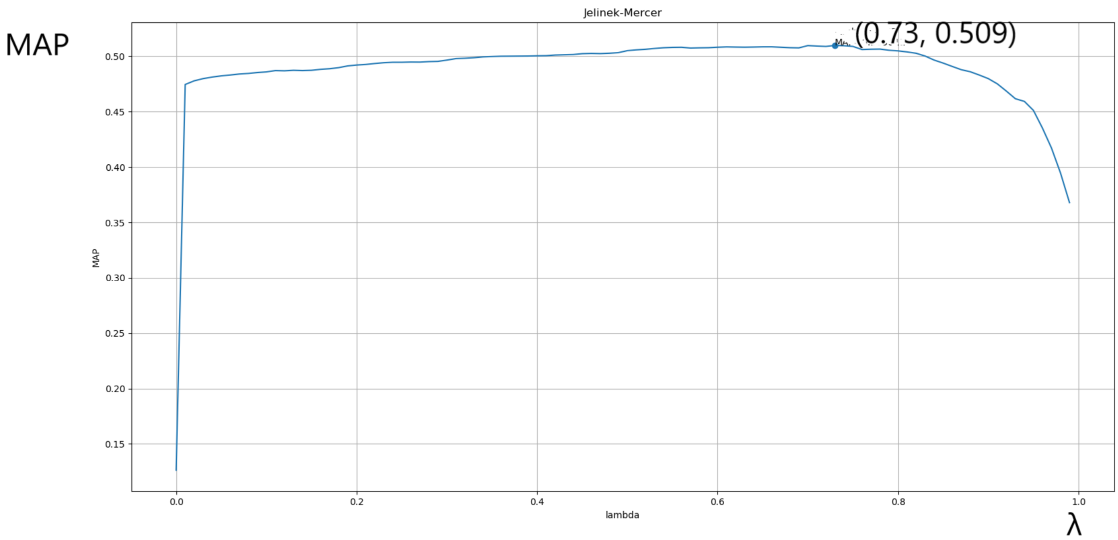

Figure 2: Jelinek-Mercer

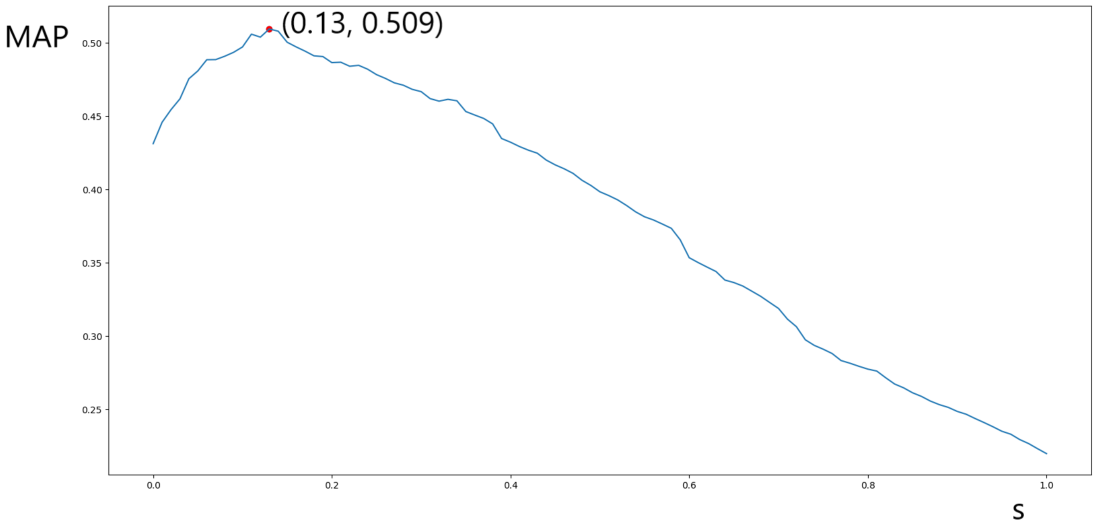

Figure 3: Pivoted-length

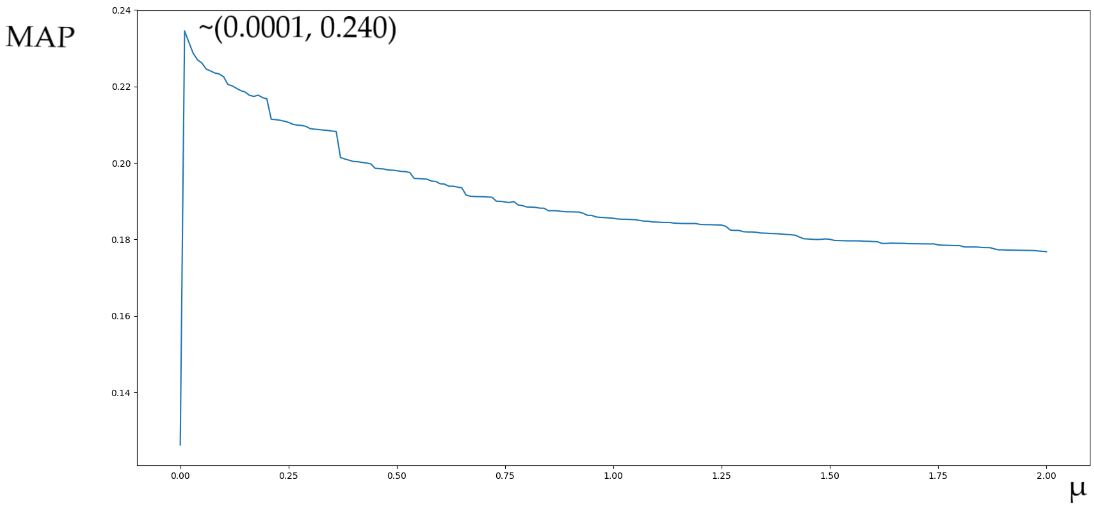

Figure 4: Dirichlet

Filtering

For this task once again we wrote some python scripts that edited the input parameters and run the *competition*. We have used combinations of *ngram-words* (monograms, bigrams) with the "default-unigram-chain" filter or tokenizers ("icu", "whitespace" and "character") and the "english-normalizer" as filter. However, no improvement was observed compared to the initial configuration of *config.toml* (monograms + "default-unigram-chain"), the MAP value was ≈ 0.504
.

Tune stopwords list

For this part we searched on internet a big list of stopwords and we found one on *https://www.ranks.nl/stopwords*, namely "Long Stopword List" with 491 words. Then we did a union and a difference between the two sets and a random selection of 300 words from the initial list, thus 3 scenarios. We have noticed an improvement when we used the union of the two stopwords list, i.e. 0.504568 -> 0.506875. The results are depicted in the following table.

Stopwords file	**MAP**	**Word count**
lemur-stopwords	0.504568	431
lemur-union	0.506875	793
lemur-difference	0.494568	260
lemur[0-9]	0.504568	300

Table 2: Different stopword lists and their results

Lab 3

Task 1: Part-of-Speech Tagging

For this task we run the built-in functionality of *MeTA* for pos-tagging on an short text provided. We observed how every word from the initial document was annotated with it's corresponding part-of-speech (e.g: cover_VB - verb base form, course_NN - noun).

Task 2: Word Association Mining

We were given a corpus of 10 000 restaurant reviews and asked to extract the words with the strongest syntagmatic relation. This technique can be useful when we want to find meaningful insight from a large text dataset (e.g. lots of reviews) without having to read it (i.e. save time). Another application of this would be to expand a query in order to get better retrieval results.

First, we were asked to run **association.cpp** with the option "–word 50" to get the top 50 word pairs with highest occurrence in the corpus. At first we were thinking that something is wrong since it took a long time to run, but after analyzing the code we noticed that is has a complexity of $O(n^2)$ where $n \approx 30000$. Out of the top 50 pairs, we selected the top 10 to put in this document.

1. 1811 good place
2. 1535 food place
3. 1511 food good
4. 1278 it place
5. 1248 great place
6. 1198 place time
7. 1104 good it
8. 1081 good time
9. 1032 good great
10. 987 friend place

As can be seen above a high-number of co-occurrences does not imply high correlation, thus in order to obtain better results we had to write few lines of code to complete the implementation of *mutual information* (a measure/metric for detecting interesting collocations).

Listing 1: added lines of code in MutualInformation() method

```
...
double PXab01 = PXb1 - PXab11;
double PXab10 = PXa1 - PXab11;
double PXab00 = PXa0 - PXab01;

return  PXab00 * log2(PXab00 / (PXa0 * PXb0)) +
        PXab10 * log2(PXab10 / (PXa1 * PXb0)) +
        PXab01 * log2(PXab01 / (PXa0 * PXb1)) +
        PXab11 * log2(PXab11 / (PXa1 * PXb1));
```

The top 10 results after running **association** with *mutual information* as metric for finding the words with strongest syntagmatic relation are:

1. 0.095037 cream ice
2. 0.0582135 ann arbor
3. 0.0434428 pad thai
4. 0.0430953 harvard squar

5. 0.0385587 burger fri
6. 0.0378376 crust pizza
7. 0.0362298 roll sushi
8. 0.0353419 alto palo
9. 0.0343221 price reason
10. 0.0337761 minut wait

Task 3: Topic Modelling

For this task we have worked with *Probabilistic Latent Semantic Analysis(PLSA)* technique to mine topics and themes in a corpus of reviews for four types of products: cars, boats laptops and wearables. The implementation of PLSA made use of the EM algorithm which was not completely implemented so we had to add some code for one of its equations.

Listing 2: needed lines of code to implement equation no. 11

```
...
double res = 0;
for (int j=0; j < num_topics; j++){
        res += pi[d][j]*Pw[j][w];
}

PzB[d][w] = (lambda_B * PB[w]) / ((lambda_B * PB[w]) + (1 -
    lambda_B) * res);
```

For two topics, i.e k = 2, the results were
:

Topic 1

1. s_> 0.022045
2. it 0.0119796
3. /_s 0.0117394
4. >_< 0.0112643
5. <_s 0.0110225
6. <_/ 0.0110225
7. ._< 0.0105307
8. the 0.0097954
9. to 0.00955309
10. you 0.00676573
11. of 0.00669329
12. that 0.00629335
13. on 0.00511801
14. watch 0.00412201
15. is 0.00393359
16. as 0.00365096
17. in 0.00341608
18. your 0.00336135
19. -_- 0.0031759
20. also 0.00311267

Topic 2

1. ?_? 0.00764276
2. ?_? 0.00764276
3. seat 0.00566883
4. boat 0.0048737
5. engin 0.00472251
6. wrangler 0.00467477
7. ?_? 0.00461225
8. wheel 0.00457531
9. drive 0.00452188
10. ?_? 0.00371765
11. liter 0.00368014
12. mpg 0.00358068
13. -_liter 0.00348121
14. lb 0.00338175
15. ?_? 0.00337379
16. car 0.00328229
17. control 0.00318055
18. ?_ 0.00318055
19. jeep 0.00308336
20. trim 0.00303334

The first topic summarizes "laptops" and "wearables" and the second "boats" and "cars", though we are not sure why we received some "words" that cannot be encoded as can be seen above.

The result for 4 topics were:

Topic 1

1. it 0.00893652
2. macbook 0.00831252
3. chromebook 0.00711023
4. that 0.00657309
5. inch 0.00581117
6. -_inch 0.00581061
7. pro 0.00505733
8. wrangler 0.00470677
9. new 0.00465991
10. machin 0.0046572
11. as 0.00441488
12. laptop 0.00435745
13. samsung 0.00405657
14. batteri 0.00395755
15. it_'s 0.00390866
16. intel 0.00370634
17. appl 0.00360721
18. ,_it 0.00345795
19. xps 0.00335499
20. life 0.00330622

Topic 2

1. the 0.0121042
2. watch 0.0090172
3. s_> 0.00719127
4. and 0.00665868
5. of 0.00651705
6. to 0.00645934
7. the_Watch 0.00620564
8. up 0.00504224
9. or 0.00417616
10. <_/ 0.00359531
11. <_s 0.00359498
12. boat 0.0033073
13. car 0.00326078
14. >_< 0.00310286
15. face 0.00300279
16. ?_? 0.00295555
17. ?_? 0.00295555
18. can 0.00295113
19. of_the 0.00282481
20. ._< 0.00264585

Topic 3

1. uxa 0.0100919
2. vivofit 0.00910247
3. ?_? 0.00731699
4. ?_? 0.00731699
5. asus 0.00677356
6. you 0.00652071
7. we 0.00596458
8. zenbook 0.00573851
9. the_Vivofit 0.00554063
10. this 0.00551861
11. garmin 0.00459907
12. the_UX31A 0.00455123
13. your 0.00449229
14. prime 0.00415547
15. Zenbook_Prime 0.00395759
16. tomtom 0.00385756
17. ?_ 0.0037488
18. strap 0.00355747
19. ASUS_Zenbook 0.00336395
20. display 0.00327464

Topic 4

1. nike 0.00724187
2. and 0.00719171
3. surg 0.00588154
4. s_> 0.00510551
5. explor 0.00494729
6. >_The 0.00450468
7. the_Surge 0.00392103
8. fitbit 0.00391301
9. soni 0.00387101
10. fuel 0.00375093
11. audi 0.00366694
12. trim 0.00356691
13. four 0.00346287
14. devic 0.00331282
15. gps 0.00327484
16. fuelband 0.00322883
17. seat 0.00315074
18. engin 0.0030047
19. you'r 0.00299668

Task 4: Text mining competition

In order to improve classification accuracy we tried:

1. **ngrams** of 2 => no improvement (same as for unigrams, 0.984)
2. combination of unigrams and bigrams => small decrease 0.984 -> 0.982
3. used the classifier "naive-bayes" => small decrease 0.984 -> 0.934

4. failed to use *k-nearest neighbour* classfier due to a "segmentation fault" although *config.toml* was configured as described in documentation

The python scripts and output files can be found at: https://github.com/mariusolariu/dm.git