ProbFuse2006: Information Retrieval Course Project to Implement and Evaluate an advanced Rank Fusion technique

Cazzaro Davide - 1138635
Venir Luca - 1139089
Group Name: ProbFuse2006

March 19, 2018

1 Introduction

Rank Fusion denotes a set of methods to combine documents retrieved by multiple IR models in order to obtain an higher overall efficacy. In this it resembles the machine learning technique of *ensemble* models or the *Wisdom of the Crowd* effect, in which the aggregation of different estimations smooths out the individual model flaws and leads to a better final prediction.

In this work we set out to implement and evaluate the *comb* fusion methods and an advanced method: we chose ProbFuse, a probabilistic algorithm to combine together the runs.

The code developed is publicly available on the *github* repository at the address https://github.com/davidecazzaro/ProbFuse2006 (minus the collection) and can be used to replicate our results.

This report is organized as follows. Section 2 describes the IR models used as a base to implement the rank fusion. We report the details of the comb methods implementation and the normalization technique used in section 3. The ProbFuse algorithm is reported in section 4. The results obtained are presented and analyzed in section 5. Finally in section 6 we discuss the problems encountered and the conclusions of the overall project.

2 Ten models

First of all we needed to generate ten runs, using different models, that would be used for the rank fusion techniques. To do so we used the disks 4 and 5 of the TIPSTER collection (minus the congressional record), that belongs to the TREC-7 tracks, for a total of about 528,000 documents. The runs were generated with Terrier using the 50 topic (from 351 to 400) of the TREC conference. We decided to use different types of models, with and without stoplists and varying the stemmers used. Table 1 reports the models used. Note that we also set the *ignore.low.idf.terms* to True for the Divergence From Randomness models since in the newer version of Terrier it is disabled by default. The runs thus obtained were used in the rank fusion techniques described in the following sections.

3 Comb methods and Normalization

The *Comb* fusion methods described in Fox and Shaw [1] use the score of each document in a run. So, in order to generate the new runs, there is the need to normalize the scores given by each model. In this way the scores are comparable and can be *fused* together.

To normalize the score Lee [2] proposed two similar ways. The first one calculates the new similarity score using the maximum value of similarity score obtained in the run and so is called Max_norm:

$$Max = \frac{old\_sim}{maximum\_sim}$$

The second one uses also the minimum score obtained and is called Min_Max_norm:

$$Min\_Max = \frac{old\_sim - min\_sim}{maximum\_sim - min\_sim}$$

We implemented both methods and compared their performances in Figure 1. Max_norm and Min_Max_norm have similar performaces, with a slight edge for the second technique. In fact

Run	Model	Stoplist	Stemmer
1	BM25	Stoplist	Porter
2	BM25	No stoplist	Porter
3	InL2	Stoplist	EnglishSnowball
4	InL2	No stoplist	WeakPorter
5	TF_IDF	Stoplist	NoOp
6	TF_IDF	No stoplist	Porter
7	DirichletLM	Stoplist	WeakPorter
8	DirichletLM	No stoplist	EnglishSnowball
9	LGD	Stoplist	NoOp
10	LGD	No stoplist	Porter

the first normalization method is the same as the second if the minimum similarity score is 0. For this reason, and since it was also used in the ProbFuse paper [3] to calculate the combMNZ score, we decided to use the Min_Max_norm.

At this point we implemented the six *comb* fusion techniques:

- CombMIN, that uses the lowest doc scores among the models
- CombMAX, that chooses the highest doc scores among the models
- CombMED, that takes the median of the doc scores
- CombSUM, that sums the doc similarity scores
- CombANZ, that divides CombSUM similarity score by the number of non-zero similarity values
- CombMNZ, that multiplies CombSUM by the number of non-zero similarity values

These *comb* fusion methods have been applied fusing together the ten runs of the ten IR models described above. The fused runs have been limited to the top 1000 documents with the highest similarity scores, the same number of entries as the original runs, to permit a meaningful comparison.

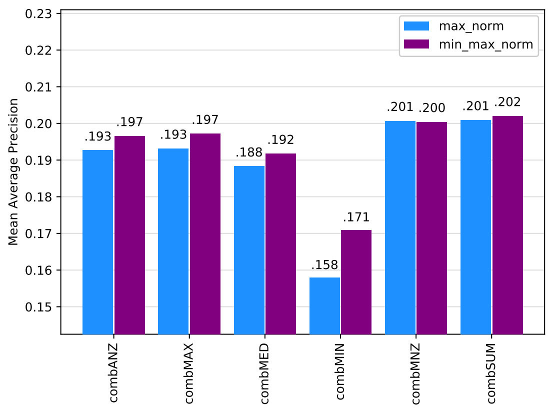

Figure 1: MAP scores comparison on TREC-7 topics for comb techniques with max_norm and min_max_norm.

4 ProbFuse

ProbFuse [3] is a probabilistic approach to data fusion described by Lillis et al. in its paper.
The technique is based on the assumption that the performance of the input models on a certain number or *training queries* (whose set is called Q) is **indicative of their future performance**; furthermore, for each topic, this approach divides the input sets into a *number of segments* (called k), locating the position of the documents in such segments rather than using their exact score. Under these two (strong) assumptions, ProbFuse fuses the input runs by performing a training process first, where, for each model (which set is called M), the probability that a certain document belongs in a certain segment is computed:

$$P(d \in k|m) = \frac{\sum_{q \in Q} \frac{|R_{k,q}|}{|k|}}{|Q|}; \qquad (1)$$

then, ProbFuse uses such probabilities to compute the score (called S_d) for each document, as it is shown in equation (2).

$$S_d = \sum_{m \in M} \frac{P(d \in k|m)}{k}. \qquad (2)$$

Note how in equation (1) $|k|$ is the cardinality of the segment k, whereas in (2) k is the number of the segment to which the document d belongs to (i.e. 1, 2, ...).

4.1 Variants

Lillis et al. showed two variants of the algorithm. The first one, *ProbFuseJudged*, takes into account every document that is judged to be **relevant** or **non-relevant** but leaving out all those documents that aren't judged; as a result the equation (1) will be replaced by the following:

$$P(d \in k|m) = \frac{\sum_{q \in Q} \frac{|R_{k,q}|}{|R_{k,q}|+|NR_{k,q}|}}{|Q|}. \quad (3)$$

Therefore, for every training query, such ratio tells us: "in this training query, how many documents happen to be relevant in such segment, **in relation to the sum of the relevant and the not relevant?**".
The second one, instead, is called *ProbFuseAll* and computes the probabilities using equation (1); such choice tells us: "in this training query, how many documents happen to be relevant in such segment, **in relation to every document in this segment?**".

4.2 Implementation choices

We chose **Python 3** as our programming environment: such choice allows us to exploit its easy-to-use data structures, such as the DICT, which simplifies the implementation.
The fusion technique we've shown depends on two tuning parameters:

1. X: segments in which we divide the input;
2. t: % of topics used to train the algorithm.

Following the ProbFuse paper [3], we chose to run our algorithm over the possible $X \in \{2, 4, 6, 8, 10, 15, 20, 25, 30, 40, 50, 100, 150, 200, 250, 300, 400, 500\}$, the possible $t \in \{.1, .2, .3, .4, .5\}$ and over the two variants described above (*ProbFuseJudged* and *ProbFuseAll*), obtaining 180 different combinations (and results): our goal is to find the best model, which is given by the best chosen $(X, t, variant)$ that gives the higher overall performance.
Please refer to our git repository for the implementation details about what we're about to describe. First, the runs given in output by *Terrier* are ".res" files in the standard TREC format: therefore, we chose to perform a once-and-for-all preprocessing phase, in which, for every line, we keep the topic number and the retrieved document name; using the ground truth file, we write the relevance information at the end of every line ("1" if it's relevant, "0" if it's not, "−1" if the document is left unjudged) into a new file. Such files are the actual input to our *ProbFuse* core function.
Second, after choosing Q by randomly extracting $50 \cdot t$ training topics, we train our algorithm by computing the probabilities described above; a function called COMPUTE_PROBABILITIES does it for each model and segment by returning a Python's DICT with *the probabilities that a document in that segment is relevant under such model.* Third, we call SCORE_EVALUATE, which is a function that returns a Python's DICT containing the score of a document within a topic. Internally, this function scans the lines of the preprocessed files and, for each document found (within a certain topic), adds up its probability within the current model (and divides by its segment, as it is shown in equation (2)).
Finally, we write down the scores in the TREC format, cutting the extracted documents to the top 1000.

4.3 Issues and Troubles

During this implementation, several problems and choices to be made came up. One case is worth analyzing, since it affects the performance of ProbFuse and, most importantly, such choice is not explicated in the paper of Lillis et al.
Consider the segmentation process and the following (simplified) example: 1000 documents must be divided into 150 segments. Clearly, the number of documents for each segment (which is $\frac{1000}{150} = 6.67$) is not an integer: we have to choose how to manage this exception. Choosing the floor function (i.e. rounding down to 6) is not a good option, because this would lead to unwanted extra segments; the ceiling function (i.e. rounding up to 7) is not an option either, because this would lead, in some cases, to completely empty segments. Although other solutions are possible, such as choosing the rounding down option, and increasing the last segment with the remaining documents, the option we came up with is the following: take the **reminder** of such division (i.e. $1000\%150 = 100$); that will be the **number of segments** which sizes are **rounded up** (i.e. 100 segments of size 7), while the remaining segments will have their size **rounded down** (i.e. 50 segments of size 6).

This option is efficient, because it requires a constant number of operations with respect to the size of the problem, and, most importantly, it is effective: we obtain balanced segments.

5 Results

Since it was used in the ProbFuse paper [3], we used *trec_ eval* to calculate the performances of the various models considered. We also use the TREC-7 topics and a subset of the TIPSTER collection, as described previously.

Probfuse is a non-deterministic technique since it uses a random sample of the topics in the training phase. For this reason all the scores reported for the ProbFuse methods are the average of 5 different runs (so the results are less biased for a particular training set) as done on the ProbFuse paper.

Figure 2 shows all the combinations of number of segments and of percentage of training topics for the two ProbFuse variants, ordered by number of segments. In this graph it is noticeable how a low number of segments, such as 2 or 4, causes the algorithm to perform badly, since it uses a too rough division of the run (documents) to extract a meaningful ordering. Starting from about 25/30 segments and for higher numbers we see a stabilization on the performance.

The best scores obtained are 0.2173 for ProbFuseAll with $X = 500$ segments and $t = 40\%$ of training and 0.2154 for ProbFuseJudged with $X = 400$ segments and $t = 50\%$ of training.

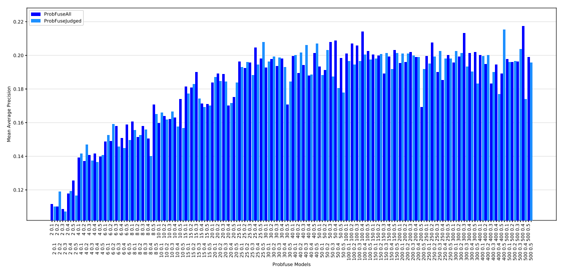

Figure 2: Probfuse MAP scores, averaged on 5 runs, on TREC-7 topics, ordered by number of segments. ProbFuseAll is in blue, ProbFuseJudged in light blue.

Another interesting point is that, even if a larger percentage of training topics leads generally to a (slightly) better score, the number of segments chosen have more influence on the final performance than the percentage of training used, probably because a low number of segments heavily penalizes the final score.

In Figure 3 we plotted the interpolated 11-point Recall Precision curve for the two variants of ProbFuse algorithm. The two have very close formulas and indeed show very similar performances, with ProbFuseJudged achieving a slightly better result.

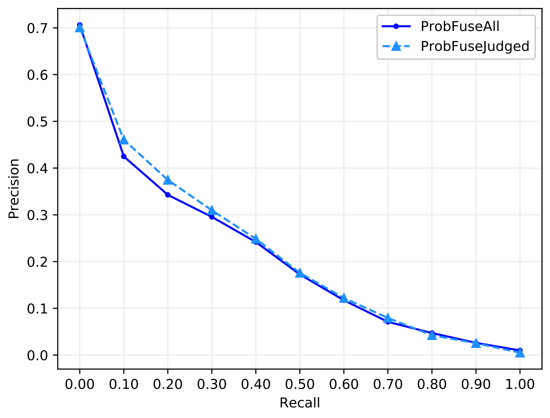

Figure 3: Comparing the Interpolated 11-point Recall Precision Curve across 50 topics for TREC-7 for ProbFuseAll and ProbFuseJudged.

In order to have an indication that our implementation of the algorithm matches the one on the ProbFuse paper, we decided to replicate the results for the TREC-5 topics. In particular we reproduced the first run of the Table 6 of their paper [3] that uses $X = 25$ segments and $t = 50\%$ for training. The runs to fuse are those indicated in their paper (table 1): *acqnt1*, *citri1*, *crnlea*, *padre2*, *xerox3* and *xerox4*.

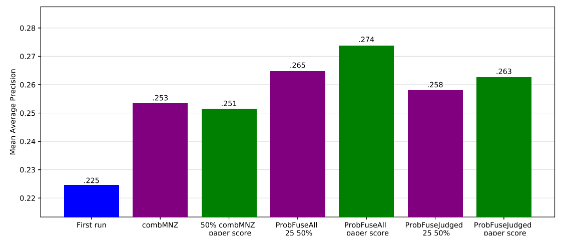

Figure 4: Comparing the MAP scores reported on the TREC-5 first run of ProbFuse paper with our results. In blue the MAP of the models not fused together is reported as a reference. In purple the scores obtained in our project and in green the scores on the ProbFuse paper.

Figure 4 shows the comparison. Our CombMNZ implementation achieves a better score that theirs: this is expected since we computed the combMNZ using all the topics while they removed a portion of the topics used for the training phase of the ProbFuse algorithm. Anyway the difference is less than the 0.02% so it is negligible for our purposes. The ProbFuseAll and ProbFuseJudged MAP scores are instead slightly worse than the results reported in their paper, but within the 1% from their scores. The difference is probably due to the

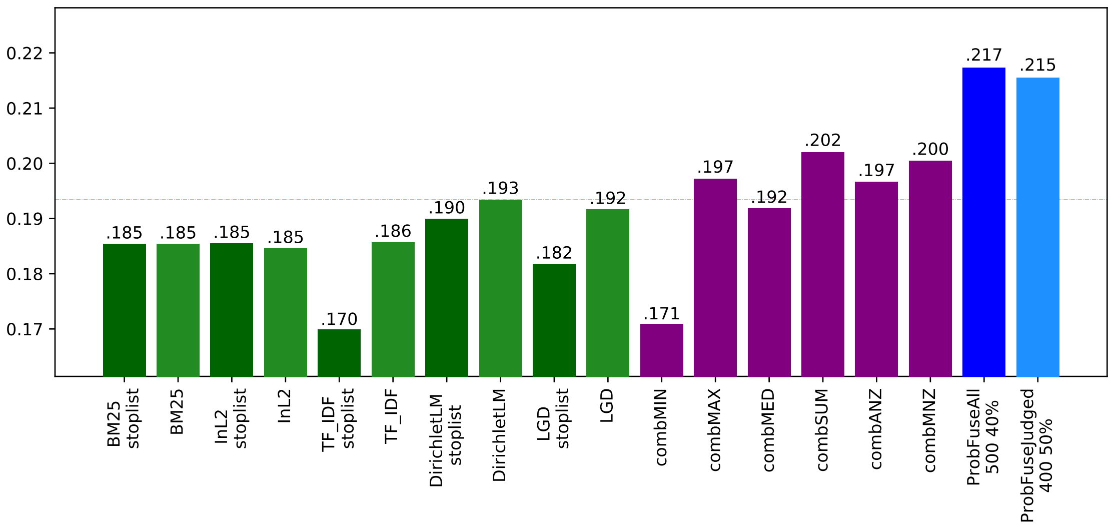

Figure 5: MAP scores on TREC-7 topics for ten different models generated with Terrier and for the six *Comb* methods implemented. Darker green is for models which use the stoplist, lighter green for the same models without the stoplist. In blue the best scores for ProbFuseAll (500 segments, 40% training) and for ProbFuseJudged (400 segments, 50% training)

non-deterministic nature of the training phase. In both our results and theirs the ProbFuse techniques outperforms CombMNZ. These observations give a first indication that our implementation is correct and that their results are replicable. Finally Figure 5 summarize the results obtained in the TREC-7 topics, comparing the ten models MAP scores with the comb techniques scores and with the best scores for ProbFuseAll and ProbFuseJudged. It can be observed that the comb methods achieve a better score than the base models, except for CombMIN and CombMED, and that the ProbFuse methods obtain a 1.5% gain over the best among the Comb methods. This confirms that the ProbFuse algorithms can be worth implementing and bring an increase in performance in a different collection than the ones in the original paper.

6 Conclusion

We implemented the Comb techniques and an advanced method for rank fusion. The results confirm that those methods improve the overall score of retrieval system. We also replicated a small part of the ProbFuse paper and obtained comparable results.

It is also worth mentioning that initially we had a bug in our implementation of ProbFuse. We noticed it because plotting the ProbFuse MAPs ordered by segments we observed an unexpected pattern in which higher training percentages worsened the performance. This was because of a subtle error when we selected the training portion of the topics and underlines how visualizing the data can be beneficial.

In conclusion we think that our goals for the project have been achieved and also we learned to use the tools that are employed in Information Retrieval.

References

[1] Edward A Fox and Joseph A Shaw. Combination of multiple searches. *NIST SPECIAL PUBLICATION SP*, 243, 1994.

[2] Joon Ho Lee. Combining multiple evidence from different properties of weighting schemes. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 180–188. ACM, 1995.

[3] David Lillis, Fergus Toolan, Rem Collier, and John Dunnion. Probfuse: a probabilistic approach to data fusion. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 139–146. ACM, 2006.