Topic 8: Fusion

COMP3009J: Information Retrieval

Dr. David Lillis (david.lillis@ucd.ie)

UCD School of Computer Science
Beijing Dublin International College

Introduction

- You have already seen that there are many algorithms that can be used for Information Retrieval (IR):
  - Boolean Model
  - Vector Space Model
  - Probabilistic Model
  - BM25
  - … many more
- If there was one algorithm that worked better than all the others all the time, we could just use that.
- There isn't!

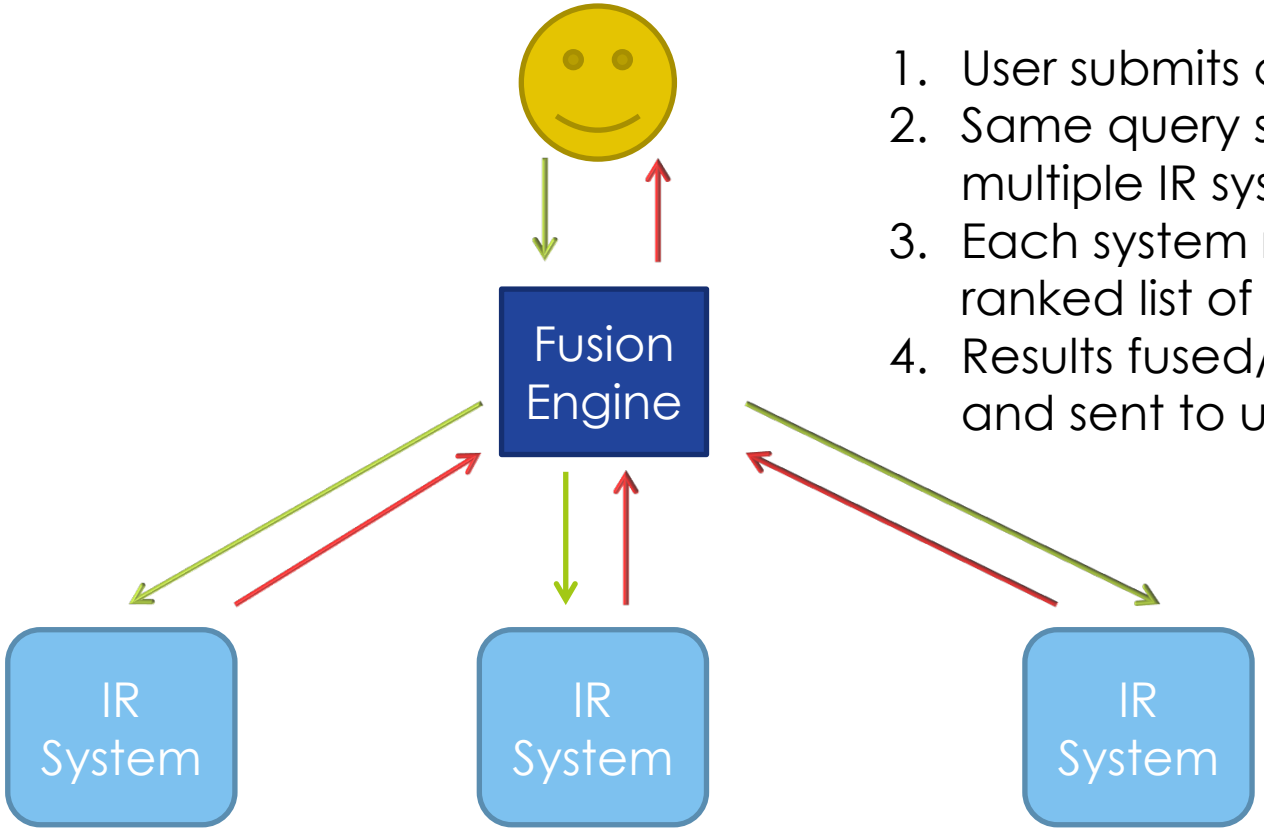
Introduction

- From the 1990s, an area of research called ***data fusion***, ***collection fusion***, or ***results aggregation*** became increasingly popular.
- **Basic definition:**
  - **Combining** the outputs of **multiple** different Information Retrieval systems/algorithms into a single ranked result set that can be shown to a user in response to a **query**.
- It is hoped that this combined result set will be of **better quality** than the invidual input result sets.

Better Quality?

- But what do we mean by *better quality*?
  - Higher **Recall** – almost inevitable if we bring in an additional set of results (more of the available relevant documents are retrieved)
  - Higher **Precision** – more difficult. Need to make sure we introduce relevant documents in place of non-relevant documents.
    - For metrics (such as MAP) that reward high positions for relevant documents, we should ensure that our ranking puts more relevant documents first.

Anatomy of a Fusion System



1. User submits query
2. Same query sent to multiple IR systems
3. Each system returns a ranked list of results
4. Results fused/merged and sent to user.

Anatomy of a Fusion System

- The final list returned to the user is ranked by calculating a **score** for **each document**: the document with the highest score goes at the beginning.
  - We don't look at the document contents.
- Generally, each underlying IR system will contribute something towards the score of each document it returns.
  - Might depend on the **position/rank** of the document in the result, the **score** explicitly given by the search engine, **quality** of the search engine itself.
- Main difference between algorithms: what information they take into account when allocating scores.

Applications of Fusion

- **Metasearch**: This involves fusion of result sets returned by **autonomous, complete search engines** (e.g. Google, Bing).
- **Distributed Information Retrieval:** Numerous IR systems are **designed to co-operate** with one another, each working on a subset of the document collection.
- **Internal Metasearch:** Numerous algorithms perform searches on the **same document collection** (data fusion).

Corpus Overlap

- Before developing a fusion algorithm, it is important to consider how much the document collections (corpora) used by the systems we wish to use **overlap**.
- There are three different **levels of overlap** that can occur between document corpora.
- The level of overlap will have a significant effect on how we treat the result sets when fusing.

Corpus Overlap: Disjoint Databases

- Here, the input systems search separate, disjoint document collections that have **no documents in common**.
- A document **cannot be returned by more than one input system**, since it does not appear in more than one index.
- Distributed IR is typically implemented using disjoint databases.
- Fusion of disjoint corpora is frequently known as *Collection Fusion*.

Corpus Overlap: Identical Databases

- The input systems each apply their own IR algorithm to the **same set of documents**.
- Documents will frequently appear in multiple result sets.
- Appearing in multiple result sets is frequently interpreted as further **evidence of relevance** (as multiple systems agree that it is relevant).
- This has become known as ***Data Fusion*** and is our **main focus**.
- Internal Metasearch is generally a Data Fusion task.

Corpus Overlap: Overlapping Databases

- The document collections being used by the various input systems have some level of overlap, but are **not identical**.
- Documents may appear in multiple result sets.
- However, it is **difficult to draw reliable conclusions** about these documents that appear in multiple result sets.
- External Metasearch generally involves overlapping databases.

Corpus Overlap: Overlapping Databases

- A common feature of fusion algorithms is to give a **higher score** to documents appearing in **multiple result sets**:
  - Appears in every result set => every system considers it relevant.
  - Appears in no result sets => no system considers it relevant.
  - Appears in one result set but not another =>
    - One system does not consider it to be relevant **OR**
    - One system does is not aware of the document.
- Difficult to decide how to treat the last situation, as it's often impossible to tell which possibility has occurred.

Three Fusion “Effects”

The Skimming Effect

- There are three "**effects**" that a fusion algorithm may try to leverage*
- In general, the most relevant documents in a result set appear at or near the top, when an effective IR algorithm is being used.
- The Skimming Effect argues that **"skimming" the top documents** from each result set and using these for fusion should give better performance.
- This principle is used for **all popular fusion algorithms**.

* Vogt, C. C., & Cottrell, G. W. (1999). Fusion Via a Linear Combination of Scores. *Information Retrieval*, *1*(3), 151–173

The Chorus Effect

- If **multiple input systems agree** that a document is relevant, the Chorus Effect argues that this evidence of relevance should be taken into account and that document should be **highly ranked** in the fused result set.
- Whether this effect is applicable depends on the level of overlap between the corpora used by the input systems.
- For Data Fusion, the Chorus Effect tends to be an important consideration.
- For Collection Fusion, it is not a factor at all (documents cannot appear in multiple result sets).
- Difficult to gauge for partially overlapping corpora.

The Chorus Effect

System A	
Rank	**Document**
1	d_{19}
2	d_5
3	d_{12}
4	d_4
5	d_{14}
6	d_{15}
7	d_1
8	d_9
9	d_{10}
10	d_{11}

System B	
Rank	**Document**
1	d_5
2	d_{14}
3	d_{20}
4	d_7
5	d_1
6	d_{11}
7	d_{18}
8	d_3
9	d_{10}
10	d_{12}

Document d_5 is given a high rank by both systems: should probably be ranked highly in fused result set.

Document d_{19} is given a high rank by one system but is not returned at all by System B: should be ranked below d_5 according to the Chorus Effect.

The Dark Horse Effect

- The Dark Horse Effect is where one input system returns results of a much **different quality** than the others.
- This may be as a result of returning either **unusually accurate** or **unusually inaccurate** results.
- This seems to **contradict the Chorus Effect**, as it would favour identifying a "dark horse" and just using its results, rather than fusing it with others.
- The Dark Horse Effect is very difficult to identify, and so is generally not used in fusion algorithms.

Categories of Fusion

Categories of Data Fusion Techniques

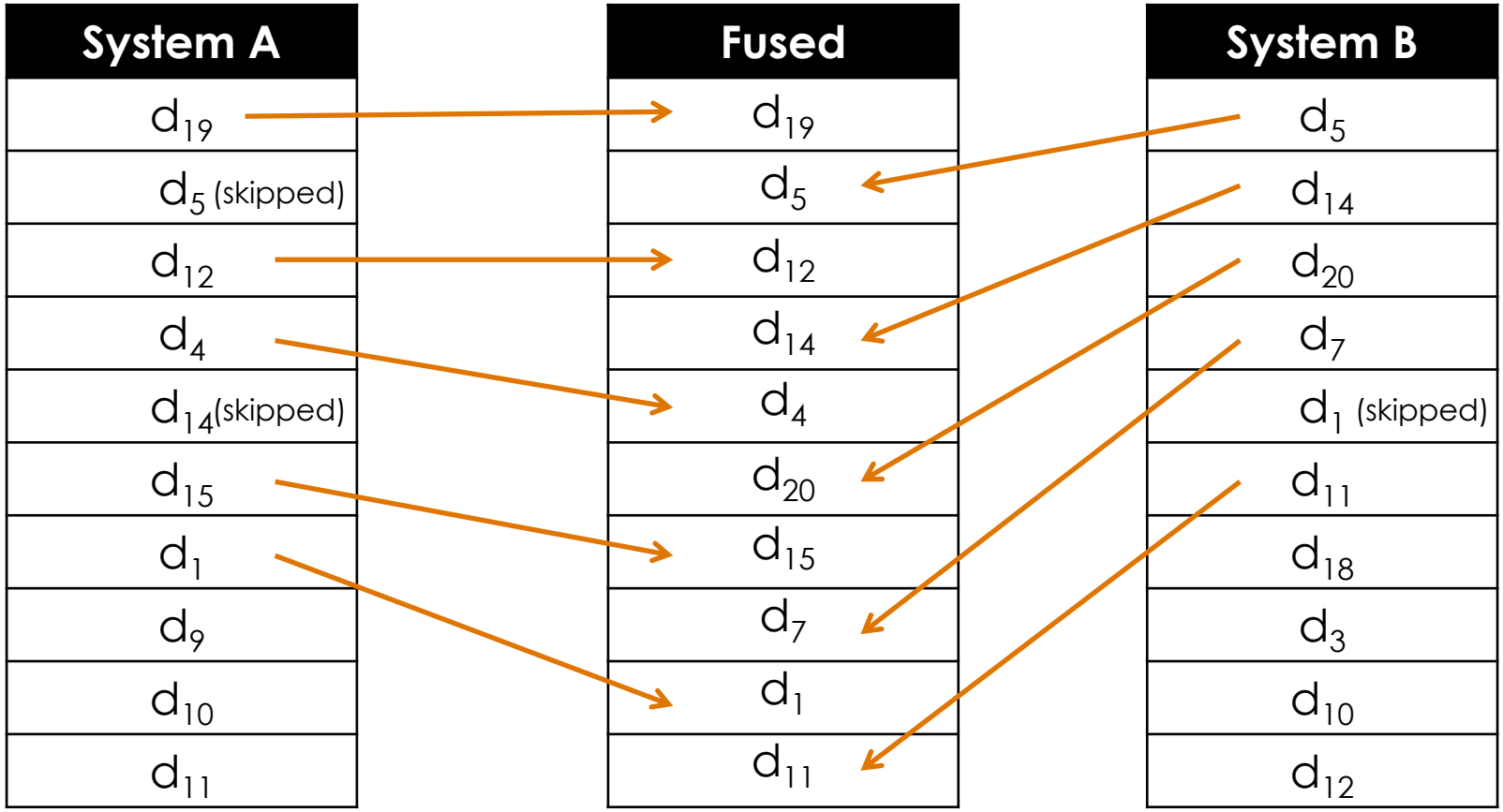
- There are three principal categories of Data Fusion techniques that we will look at:
  - **Rank-Based Fusion**: These examine only the **rank/position** that each document occupies in the input result sets. Sometimes necessary if relevance scores are unavailable.
    - **Voting models** generally operate on the rank level also.
  - **Score-Based Fusion**: The **relevance scores** given to a document by an input system can be used as a measurement of how confident it is as to the document's relevance.
  - **Segment-Based Fusion**: Result sets are divided into groups of documents, rather than using individual ranks or scores.

Rank-Based: Interleaving

- **Interleaving** is perhaps the simplest Fusion algorithm of all*
- Here, we take one document from the top of each input set in a "**round robin**" fashion and add it to the fused result set.
  - The document chosen is the highest-ranked document that is not yet included in the fused result set.
- The effectiveness of this technique is, however, poor.
- There is an assumption that every result set is of equal quality, which can have the result that the better result sets are diluted by being merged with non-relevant documents from poorer systems.

* Voorhees, E. M., Gupta, N. K., & Johnson-Laird, B. (1994). The Collection Fusion Problem. In *Proceedings of the Third Text REtrieval Conference (TREC-3)* (pp. 95–104)

Rank-Based: Interleaving (Example)



Other Rank-Based Techniques

- A variation on interleaving is to use **historical data** to estimate which input system(s) tends to perform better.
- A weighted version of interleaving is then used so that **more documents are taken from the better systems**. *
- Another approach is to treat the process as an election where a **few electors** (the input systems) must choose between **many candidates** (the documents). This is the approach taken in the *Borda-Fuse*** and *Condorcet-Fuse**** algorithms, proposed by Aslam and Monague.

* Voorhees et al. (1994)

** Aslam, J. A., & Montague, M. (2001). Models for metasearch. In *SIGIR {'01: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 276–284). New York, NY, USA.

*** Montague, M., & Aslam, J. A. (2002). Condorcet fusion for improved retrieval. In *CIKM ' 02: Proceedings of the eleventh international conference on Information and knowledge management* (pp. 538–548). New York, NY, USA.

Score-Based: CombSUM

- CombSUM is a popular algorithm for performing fusion using the **relevance scores** that each document is assigned by the input IR system. It was proposed by Fox & Shaw in 1994*.
- The final score on which a document is ranked in the fused result set is calculated by **adding the individual scores** it was given in each of the input result sets.
- High scores in the input result set are carried to the fused result set, so the **Skimming Effect** is in use.
- Also, because the scores are added, documents appearning in multiple result sets will also receive a boost, exploiting the **Chorus Effect**.

* Fox, E. A., & Shaw, J. A. (1994). Combination of Multiple Searches. In *Proceedings of the 2nd Text REtrieval Conference (TREC-2), National Institute of Standards and Technology Special Publication 500-215* (pp. 243–252)

Score-Based: CombSUM Example

- Example 1:
  - Document #1 is given a score of 0.45 by System A
  - Document #1 is given a score of 0.3 by System B
  - Document #1 is given a score of 0.35 by System C
  - Document #1's CombSUM score is 0.45 + 0.3 + 0.35 = 1.1
- Example 2:
  - Document #2 is given a score of 0.55 by System A
  - Document #2 is not returned by System B
  - Document #2 is given a score of 0.65 by System C
  - Document #2's CombSUM score is 0.55 + 0 + 0.65 = 1.2

Score Normalisation

- There is a difficulty with simply adding raw relevance scores.
- Consider the situation where different IR systems may calculate scores in **different ranges**:
  - System A: Assigns scores between 0 and 1000
  - System B: Assigns scores between 0 and 1
- We often don't know the theoretical minimum and maximum scores a system could produce.
- To get around this, we need to perform **score normalisation** so that each document's score is in a **comparable range**.

Score Normalisation

- There are a number of approaches to normalising scores*.
- The most common is known as **standard normalisation** and is calculated by:

$$normalised\_score = \frac{unnormalised\_score - min\_score}{max\_score - min\_score}$$

- The first-ranked document in each result set will have a normalised score of 1, and the lowest ranked document will receive a normalised score of 0.
- Once the score of every document has been normalised, we can then apply the CombSUM algorithm.

* Montague, M., & Aslam, J. A. (2001). Relevance score normalization for metasearch. In *CIKM ' 01: Proceedings of the Tenth International Conference on Information and Knowledge Management* (pp. 427–433). New York, NY, USA.

Score Normalisation: CombSUM

System A	
Document	Score
d_{19}	0.90
d_5	0.85
d_{12}	0.82
d_4	0.79
d_{14}	0.77
d_{15}	0.64
d_1	0.44
d_9	0.43
d_{10}	0.41
d_{11}	0.38

System B	
Document	Score
d_5	943
d_{14}	920
d_{20}	901
d_7	875
d_1	862
d_{11}	811
d_{18}	795
d_3	770
d_{10}	732
d_{12}	712

Fused	
Document	Score
d_5	943.85
d_{14}	920.77
d_{20}	901.00
d_7	875.00
d_1	862.44
d_{11}	811.38
d_{18}	795.00
d_3	770.00
d_{10}	732.41
d_{12}	712.82

Score Normalisation

System A		
Document	Score	Normalised
d_{19}	0.9	1.00
d_5	0.85	0.90
d_{12}	0.82	0.85
d_4	0.79	0.79
d_{14}	0.77	0.75
d_{15}	0.64	0.50
d_1	0.44	0.12
d_9	0.43	0.10
d_{10}	0.41	0.06
d_{11}	0.38	0.00

System B		
Document	Score	Normalised
d_5	943	1.00
d_{14}	920	0.90
d_{20}	901	0.82
d_7	875	0.71
d_1	862	0.65
d_{11}	811	0.43
d_{18}	795	0.36
d_3	770	0.25
d_{10}	732	0.09
d_{12}	712	0.00

Score Normalisation: CombSUM

System A	
Document	Normalised
d_{19}	1.00
d_5	0.90
d_{12}	0.85
d_4	0.79
d_{14}	0.75
d_{15}	0.50
d_1	0.12
d_9	0.10
d_{10}	0.06
d_{11}	0.00

System B	
Document	Normalised
d_5	1.00
d_{14}	0.90
d_{20}	0.82
d_7	0.71
d_1	0.65
d_{11}	0.43
d_{18}	0.36
d_3	0.25
d_{10}	0.09
d_{12}	0.00

Fused	
Document	Score
d_5* (*=in both)	1.90
d_{14}*	1.65
d_{19}	1.00
d_{12}*	0.85
d_{20}	0.82
d_4	0.79
d_1*	0.77
d_7	0.71
d_{15}	0.50
d_{11}*	0.43

Score-Based: CombMNZ

- CombMNZ* is a variation of CombSUM that has been shown to give superior results in a variety of scientific studies.
- Although CombSUM makes use of the **Chorus Effect** by adding the normalised scores, CombMNZ **emphasises this even further**.
- It does this by **multiplying** each document's CombSUM score by the **number of result sets** it was contained in (i.e. the number of input systems that give it a non-zero score).
- The MNZ stands for "Multiply by Non Zero".

* Fox and Shaw (1994)

Score-Based: CombMNZ Example

- Example 1:
  - Document #1 is given a score of 0.45 by System A
  - Document #1 is given a score of 0.3 by System B
  - Document #1 is given a score of 0.35 by System C
  - Document #1's CombMNZ score is (0.45 + 0.3 + 0.35) x 3 = 3.3
- Example 2:
  - Document #2 is given a score of 0.55 by System A
  - Document #2 is not returned by System B
  - Document #2 is given a score of 0.65 by System C
  - Document #2's CombMNZ score is (0.55 + 0 + 0.65) x 2 = 2.4

Score Normalisation: CombMNZ

System A	
Document	Normalised
d_{19}	1.00
d_5	0.90
d_{12}	0.85
d_4	0.79
d_{14}	0.75
d_{15}	0.50
d_1	0.12
d_9	0.10
d_{10}	0.06
d_{11}	0.00

System B	
Document	Normalised
d_5	1.00
d_{14}	0.90
d_{20}	0.82
d_7	0.71
d_1	0.65
d_{11}	0.43
d_{18}	0.36
d_3	0.25
d_{10}	0.09
d_{12}	0.00

Fused	
Document	Score
d_5* (* = in both)	3.80
d_{14}*	3.30
d_{12}* ($\uparrow^1$)	1.70
d_1* ($\uparrow^3$)	1.53
d_{19}	1.00
d_{11}* ($\uparrow^4$)	0.86
d_{20}	0.82
d_4	0.79
d_7	0.71
d_{15}	0.5

Score-Based: CombMNZ

- As with CombSUM, CombMNZ should only be used for **data fusion tasks**, since it gives higher rankings to documents that appear in multiple result sets.
- CombMNZ is very simple to calculate and has been shown to be quite effective in practice.
- For this reason, it tends to be the most common baseline technique that researchers compare new algorithms against.

Score-Based: Linear Combination

- One difficulty with CombMNZ and CombSUM is that (like interleaving) every input result set is **assumed to be of equal quality**.
- The *Linear Combination Model* addresses this by apply a **weighting** to each input system*.
- The score calculation is similar to CombSUM, except that each document's normalised scores are multiplied by the weighting associated with the input system that returned it before they are added together.

* Vogt & Cottrell (1999)

Linear Combination Example

- Example 1 (System A, B and C have weights of 1, 2, 3 respectively):
  - Document #1 is given a score of 0.45 by System A
  - Document #1 is given a score of 0.3 by System B
  - Document #1 is given a score of 0.35 by System C
  - Document #1's overall score is:

    (0.45 x 1) + (0.3 x 2) +(0.35 x 3) = 2.1

- Example 2:
  - Document #2 is given a score of 0.55 by System A
  - Document #2 is not returned by System B
  - Document #2 is given a score of 0.65 by System C
  - Document #2's overall score is:

    (0.55 x 1) + (0 x 2) + (0.65 x 3) = 2.5

Linear Combination: Weights

- Many approaches have been used to calculate weights for a linear combination. Here are two notable ones.
  1. The *CORI* algorithm*, which relies on having access to **details about the document collection** each input system is using (generally used for Distributed IR).
     - **Document Frequency**: number of documents in the document collection that contains a term.
     - **Inverse Collection Freqency**: based on the number of collections that contain the term.
  2. *LMS*** (using result Length to calculate Merging Score): longer result set results in higher weight. Worked surprisingly well in practice.

* Callan, J. P., Lu, Z., & Croft, W. B. (1995). Searching distributed collections with inference networks. In *SIGIR '95: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 21–28). New York, NY, USA.

** Rasolofo, Y., Abbaci, F., & Savoy, J. (2001). Approaches to Collection Selection and Results Merging for Distributed Information Retrieval. In *CIKM '01: Proceedings of the Tenth International Conference on Information and Knowledge Management* (pp. 191–198). New York, NY, USA.

Case Study: Probabilistic Fusion

ProbFuse - Motivation

- Using a single weight for each system only reflects that the **overall average performance** of one system is better (or worse) than another.
- This can miss some characteristics that we may wish to make use of.
  - An input system may tend to return a few relevant documents at the start of its results, so precision and recall may both be poor.
  - Another system could have good recall, but be poor at ranking its results so as to place relevant documents first.

ProbFuse - Motivation

- A better solution than a single weighting was sought.
- **Probabilistic** data fusion algorithms try to reflect the **positions/ranks** in the result sets where input systems **tend to return relevant documents**.
- **Past results** are analysed during a **training phase** to build a **series of weights**, depending on the position in a result set that a document appears in.

ProbFuse - Overview

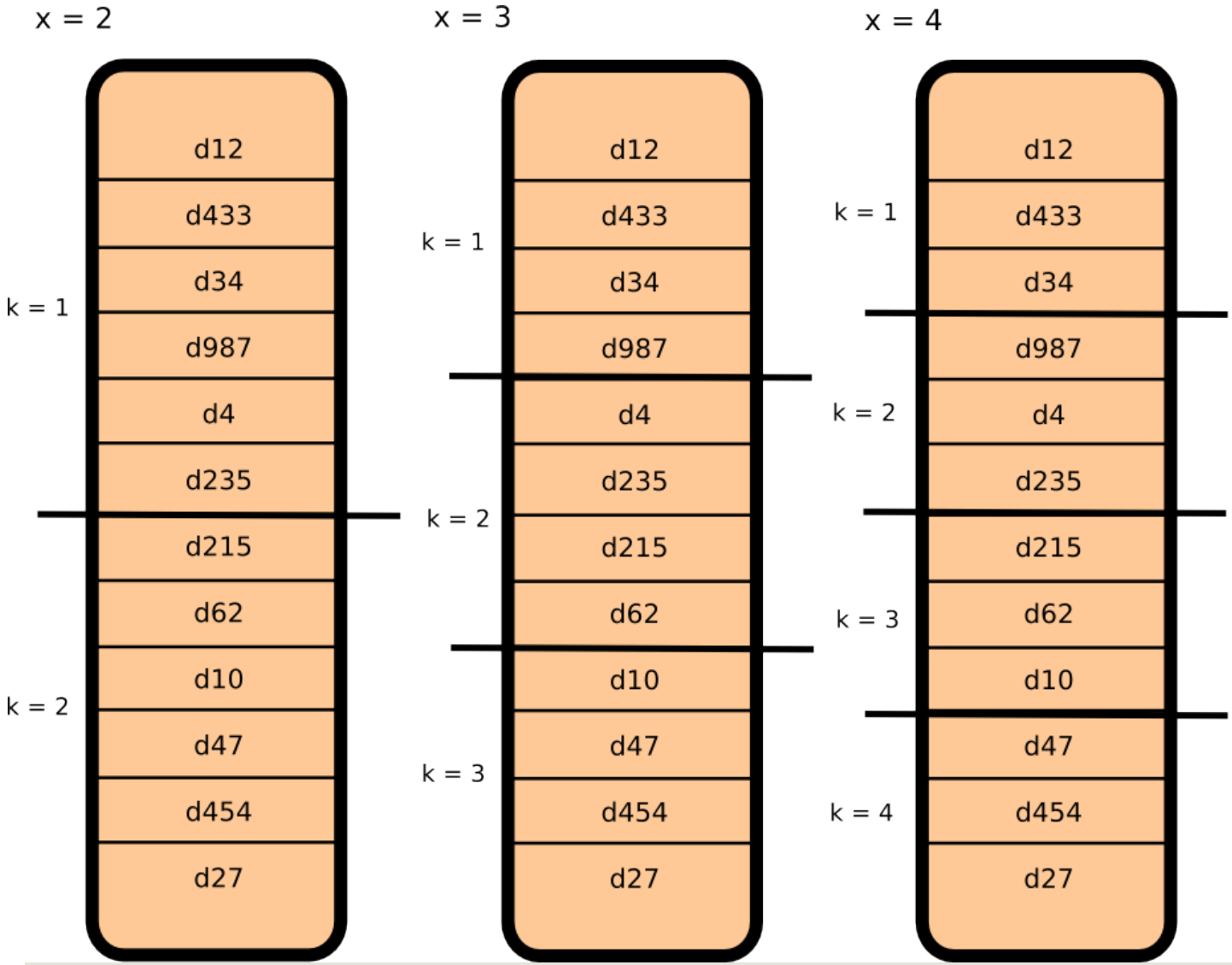
- *ProbFuse* was designed to take this information into account.*
- Each input result set is divided into x **equal-sized segments**.
- The score eventually used to rank a document is based on the **probability that the document is relevant**, given that it was returned by a particular input system in a particular segment.
- These probabilities are calculated by examining **historical data** from each input system.

* Lillis, D., Toolan, F., Collier, R., & Dunnion, J. (2006). ProbFuse: A Probabilistic Approach to Data Fusion. In *Proceedings of the 29th annual international ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '06)* (pp. 139–146). Seattle, WA, USA.

ProbFuse - Overview

- Using ProbFuse for fusion involves **two phases**:
  - **Training Phase:** Probabilities for each input system estimated, based on **historic queries** for which **relevance judgments are available**.
    - Variables:
      - *x*: how many segments to divide the result set into
      - *k*: segment number (k=1 is the first segment, k=2 the second, etc.)
  - **Fusion Phase**: Probabilities used to calculate ranking scores for each document. Used to rank final output result set.

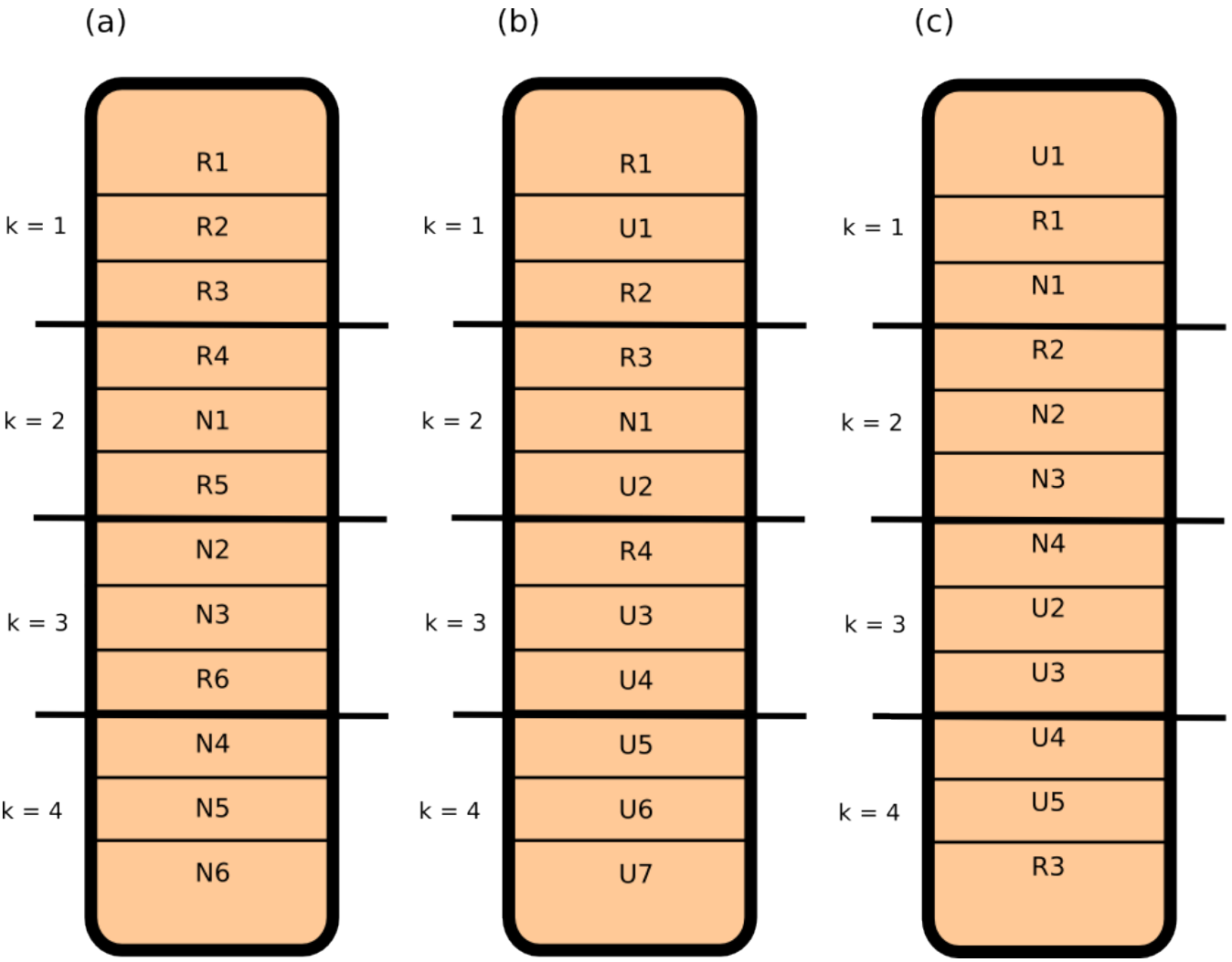
ProbFuse – What are "Segments"?



ProbFuse - Training

- A **set of probabilities** must be calculated for **every input system**.
- For this, a set of **training queries** is required, where relevance judgments are available.
- Calculate the probability that document *d* returned in segment *k* is relevant.
  1. Count all of the relevant documents in segment *k*
  2. Divide by total number of documents in segment *k*
  3. Average over all training queries

ProbFuse - Training



Key:

- R: Relevant
- N: Nonrelevant
- U: Unjudged

$$k=1: \frac{\frac{3}{3}+\frac{2}{3}+\frac{1}{3}}{3}=0.67$$

$$k=2: \frac{\frac{2}{3}+\frac{1}{3}+\frac{1}{3}}{3}=0.44$$

$$k=3: \frac{\frac{1}{3}+\frac{1}{3}+\frac{0}{3}}{3}=0.22$$

$$k=4: \frac{\frac{0}{3}+\frac{0}{3}+\frac{1}{3}}{3}=0.11$$

Result sets returned by the **same input system** in response to **different training queries**

ProbFuse - Fusion

- The score assigned to a document by each underlying input system is the **probability of relevance** associated with the **segment it is returned in**, divided by the **segment number**.
- Dividing by the segment number gives highly-ranked documents an additional boost, so as to **exploit the Skimming Effect**.
- The document's final ranking score is found by **adding each individual score**.
  - This exploits the **Chorus Effect** (higher score from being returned in multiple result sets)
- ProbFuse is therefore only suitable for **data fusion**.

SegFuse - Motivation

- Using ProbFuse, each segment is of **equal size**.
- Experiments on standard datasets showed that dividing result sets into **25 segments** resulted in **improvements over CombMNZ** in MAP and P@10 scores (bpref was inconclusive).
- The input result sets used were up to 1000 documents in length, meaining that **each segment contained 40 documents**.
- This means that document 40 would be treated the same as document 1.

SegFuse - Overview

- Shokouhi argues that this loses information, since relevant documents are more likely to be found in early positions.*
- Two major variations to ProbFuse proposed:
  - Instead of constant segment sizes, the size of the segments **increase exponentially** as we go down the result set.
  - **Normalised scores** used to boost the **Skimming Effect**, rather than dividing by the segment number.
    - For each document, the probability score of its segment was multiplied by its normalised score.

* Shokouhi, M. (2007). Segmentation of Search Engine Results for Effective Data-Fusion. *Advances in Information Retrieval, 4425*

SegFuse - Training

- When training SegFuse, the probability that a document in a given segment is relevant is calculated in **exactly the same way** as for ProbFuse.
- The only difference is that the **size of the segment** varies.
- It is given by: $Size_k = (10 \times 2^{k-1}) - 5$ where k is the segment number.

Segment Number	Size
1	5
2	15
3	35
4	75
5	155

SlideFuse - Motivation

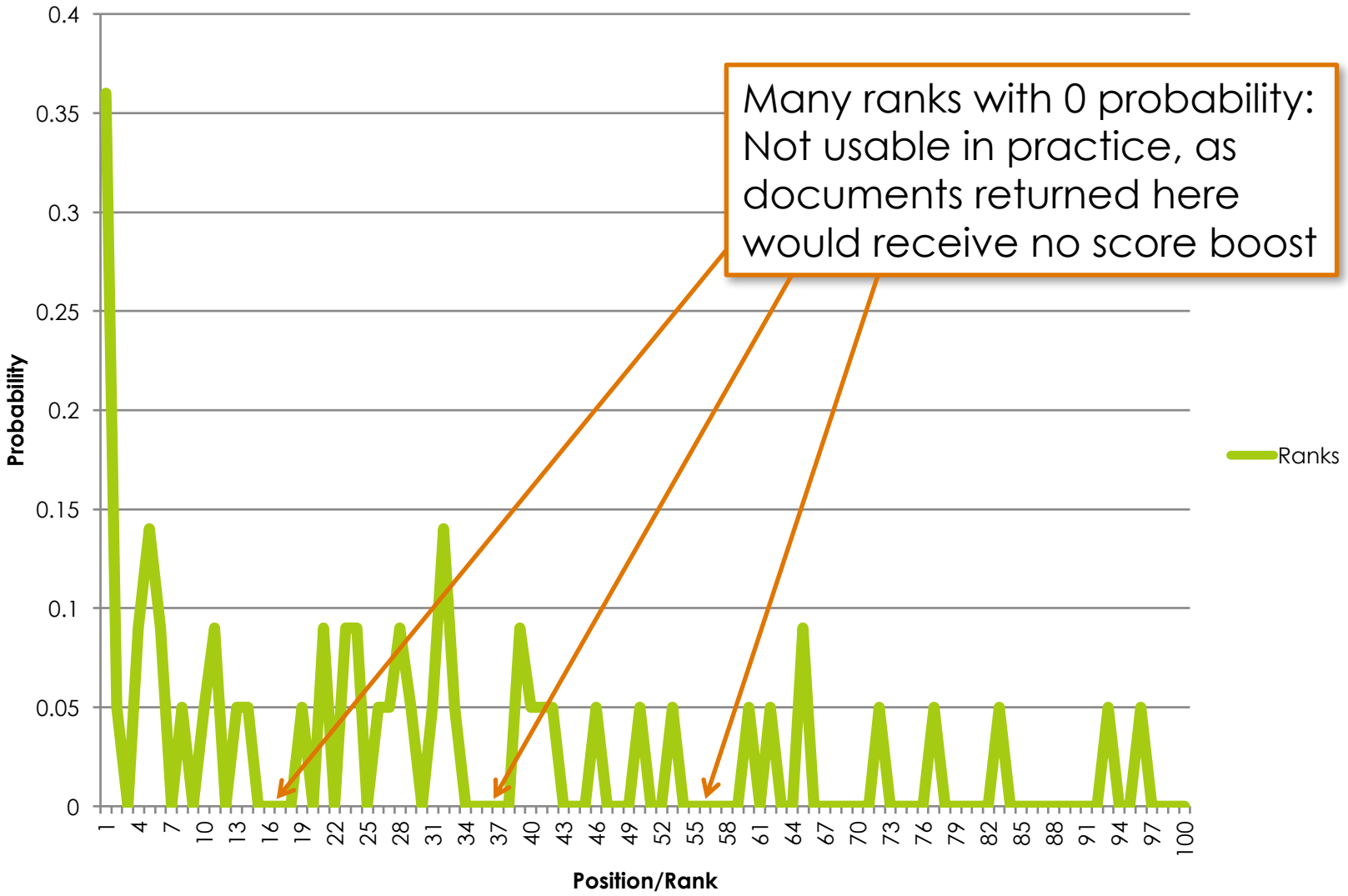
- SegFuse achieved **improvements over ProbFuse** in most of the experiments Shokouhi conducted, especially when measured using MAP.
- However, even with this improved method of dividing result sets into segments, there were still elements of concern.
- In particular, **adjacent documents** could be treated very differently, for instance under SegFuse:
  - Document 20 would be grouped with documents 6-20
  - Document 21 would be grouped with documents 21-55

SlideFuse - Overview

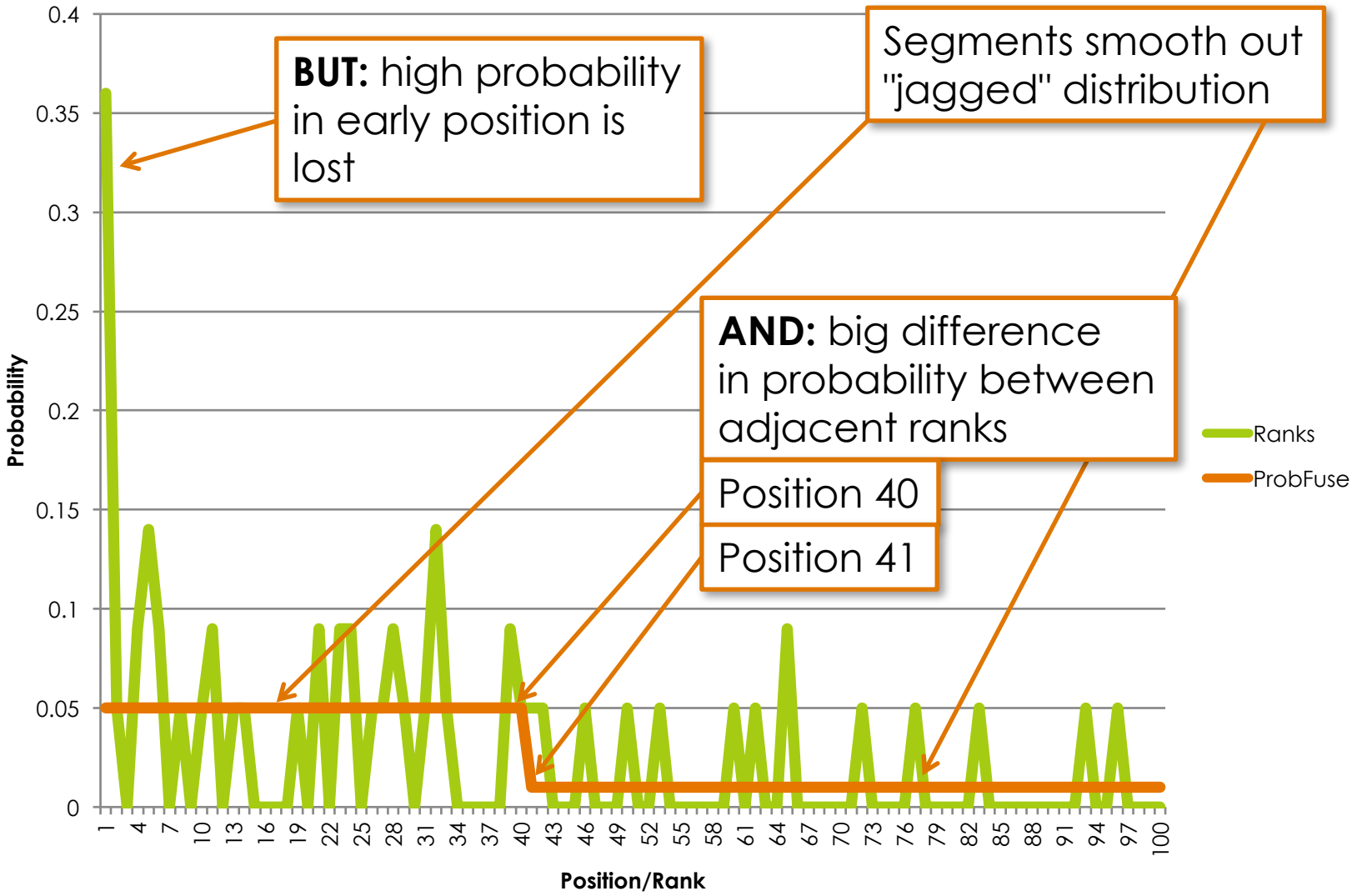
- This cutoff between segments means that documents that are side-by-side in a result set can be **treated very differently**.
- To deal with this, the ***SlideFuse*** algorithm was proposed.*
- **Aside:** Why not just use the probability at **each rank** in the result set?
- Expecially for large-scale tasks, there may be few judged relevant documents available, but we don't want to give a probability of zero to a rank just because no relevant document was returned in that exact rank during training.

* Lillis, D., Toolan, F., Collier, R., & Dunnion, J. (2008). Extending Probabilistic Data Fusion Using Sliding Windows. In C. Macdonald, I. Ounis, V. Plachouras, I. Ruthven, & R. W. White (Eds.), *Advances in Information Retrieval. Proceedings of the 30th European Conference on Information Retrieval Research (ECIR 2008)* (Vol. 4956, pp. 358–369)

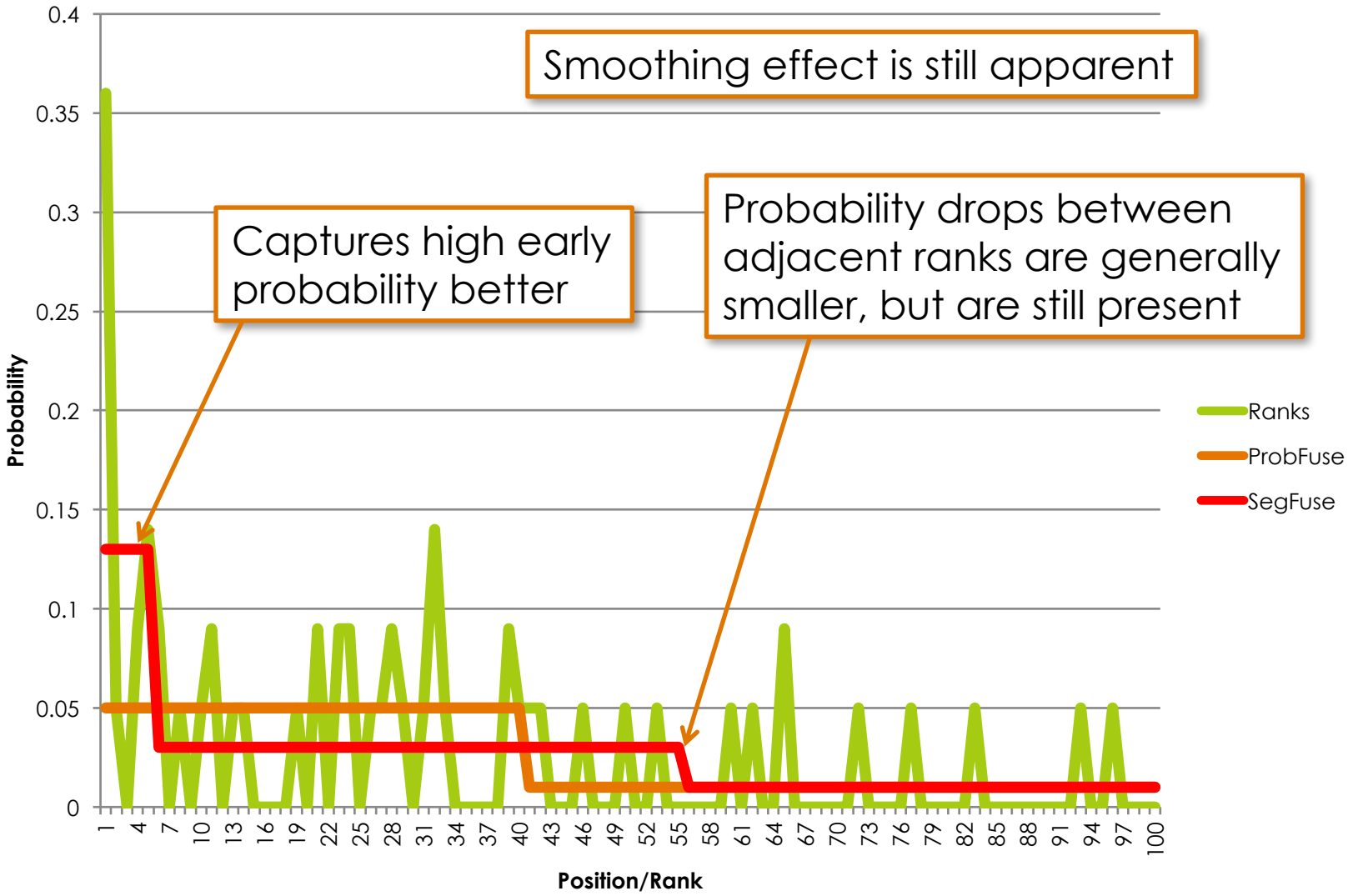
Distribution of Probabilities: Ranks



Distribution of Probabilities: ProbFuse



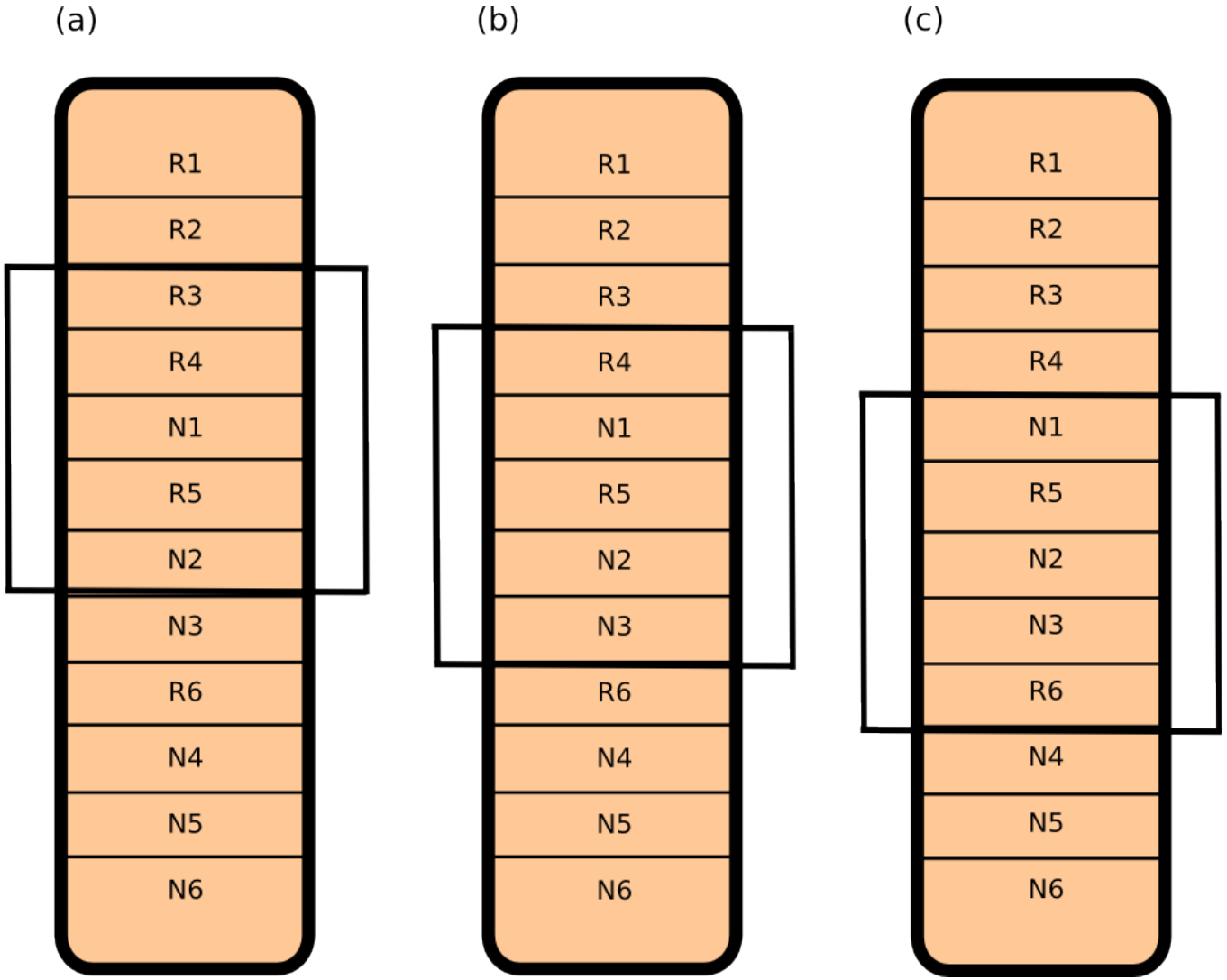
Distribution of Probabilities: SegFuse



SlideFuse - Overview

- Instead of dividing the result sets into segments, we instead use **each rank's neighbours** as evidence in estimating its probability.
- This will allow us to **smooth** the probability distribution graph to avoid the jagged peaks seen in the "ranks-only" distribution and also **avoid the sudden drops** in probability values seen with ProbFuse and SegFuse.
- We describe this as using "**sliding windows**", as the group of documents to be taken into account changes depending on which rank we are examining.

Example: Sliding Windows



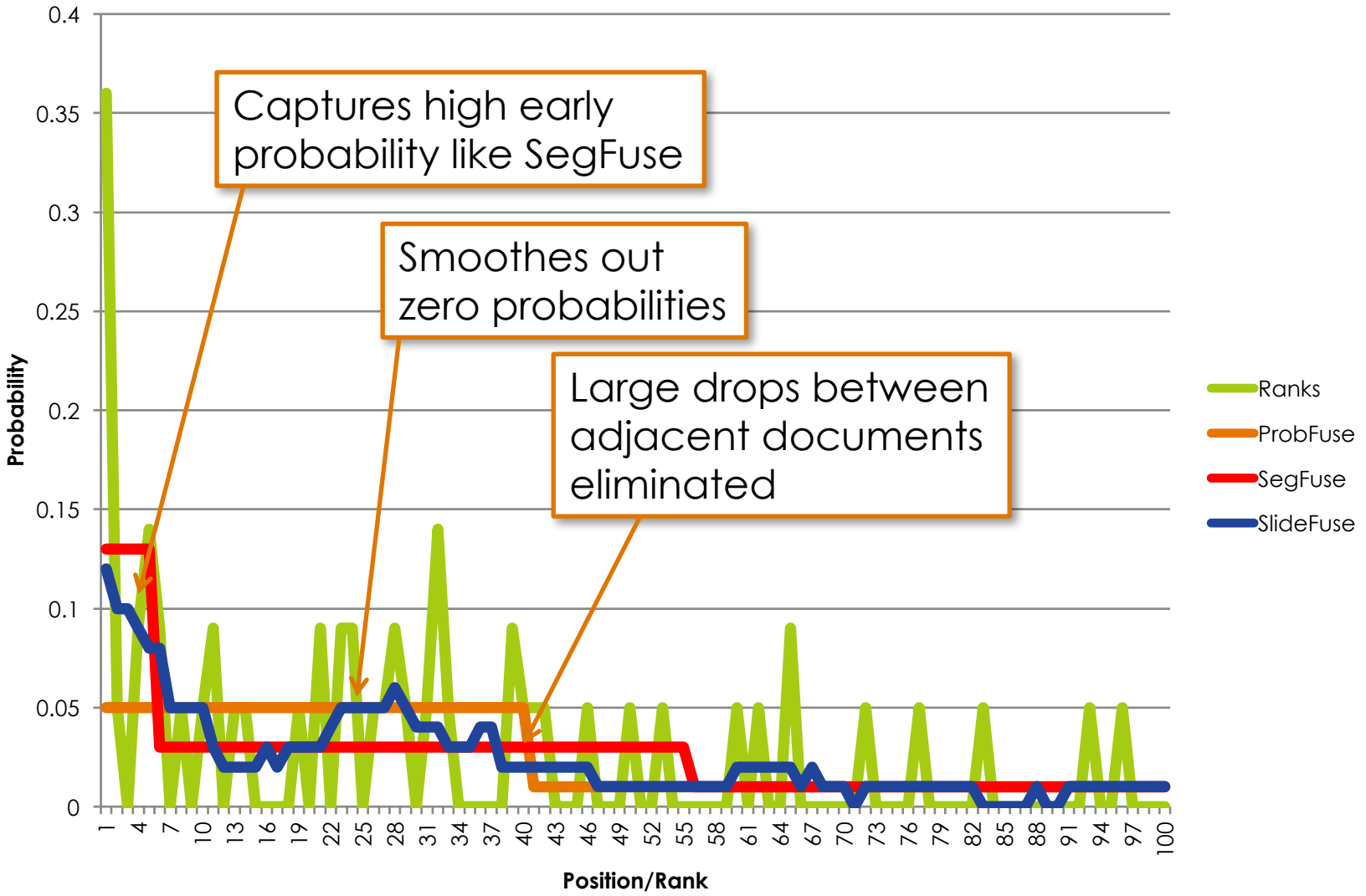
Result set for one training query returned by one system, showing sliding window for document:

a) N1
b) R5
c) N2

SlideFuse - Training

- **Training phase:** for each input system, first calculate the probability of relevance at each rank *r*.
  - Number of relevant documents returned at rank *r* divided by number of training queries.
- **Fusion phase:** final score is the sum of the scores a document receives from the input systems.
  - For each input system, get the average probability in the sliding window that surrounds the document,.

Distribution of Probabilities: SlideFuse



Conclusions

- SlideFuse achieves superior results to CombMNZ, ProbFuse and SegFuse on the TREC-2004 web track data used.
- Statistically significant performance increase for:
  - 4/5 runs measured with MAP
  - 3/5 runs measured with bpref
  - 5/5 runs measured with P@10
- Average difference is < 1% in all other cases