UNIVERSITÄT BASEL

Computer Science / 15731-01 / 2018

Multimedia Retrieval

Chapter 7: Relevance Feedback

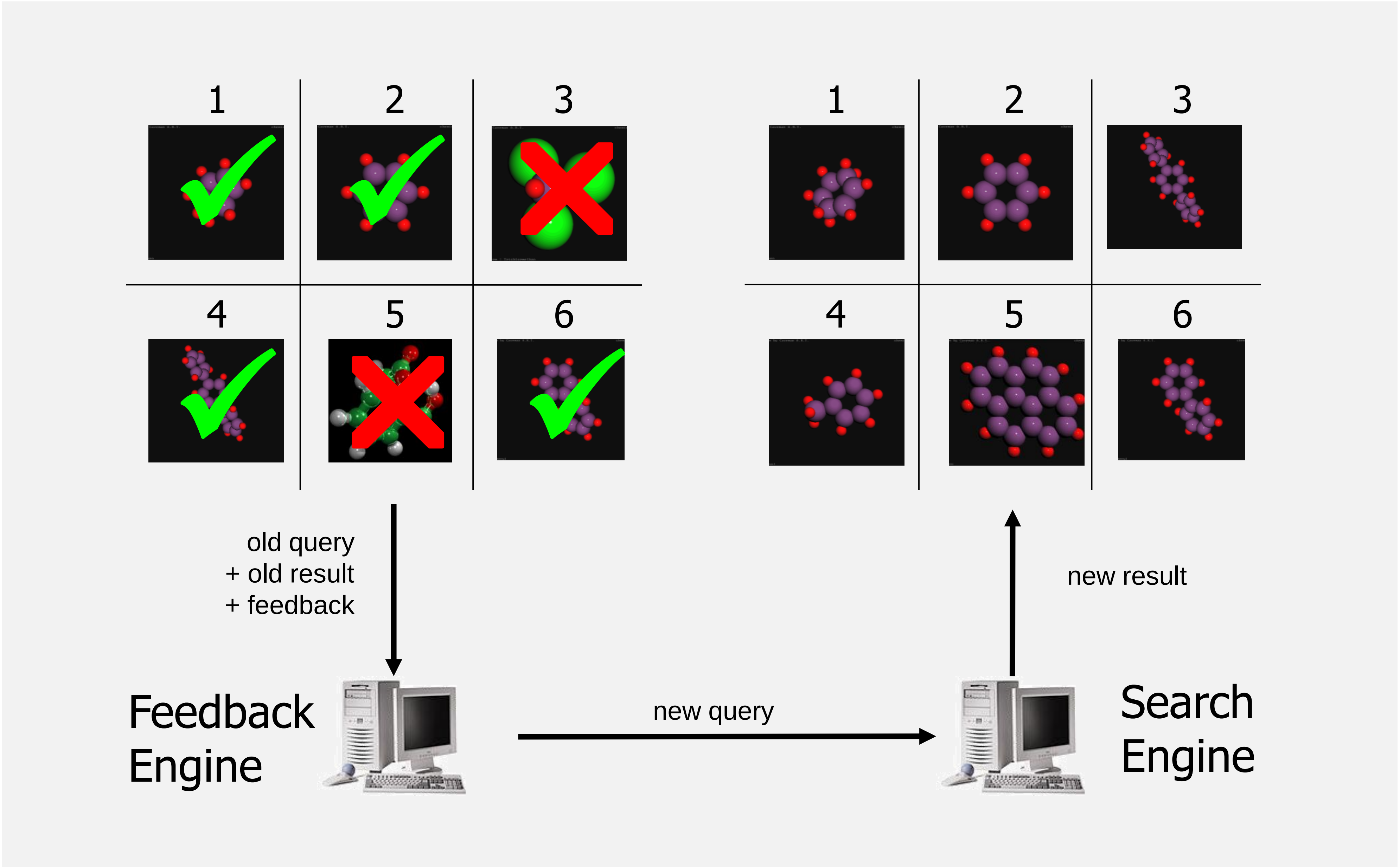
Dr. Roger Weber, roger.weber@ubs.com

7.1 Overview
7.2 Capturing Feedback
7.3 Feedback Methods for Text
7.4 Feedback Methods for Low-level Features
7.5 References

7.1 Overview

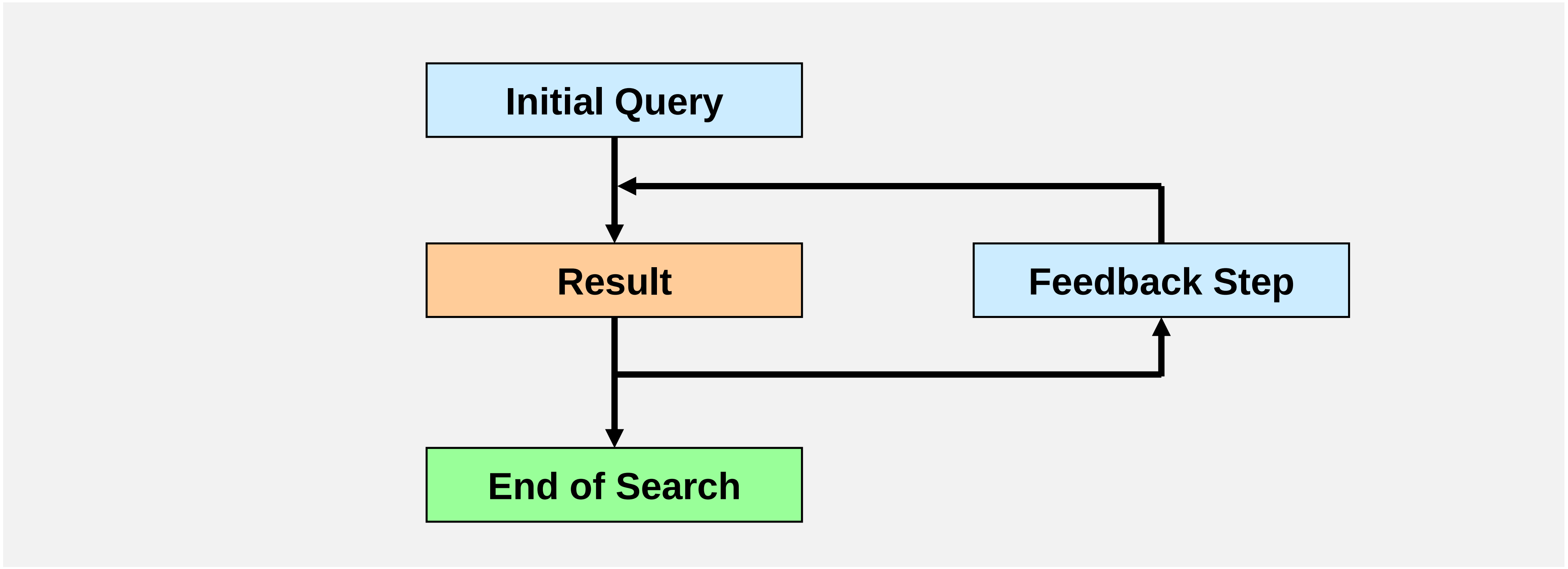
- One of the key challenges with similarity search is how to compose the query if one does not have the right samples at hand. This even can happen with keywords and terms if you are unaware of what else you could use to narrow or broaden the search.
- Relevance feedback is the art of adding, extending, or altering a query in such a way to better match the user's intentions based on feedback provided to previous results. We have seen this approach with probabilistic retrieval (BIR model) but now we will apply to other methods as well.
- Some search engines use an alternative, less intrusive method: they collect feedback by observing user behavior and use it in subsequent (similar) queries to adjust results. For instance, if user's never click on documents even though they appear in results, such documents will be penalized.
- In the following, we look at some of the basic methods to add feedback and adjust queries. We start first with capturing feedback and then how to adjust queries.

- Example: Image search with feedback loop

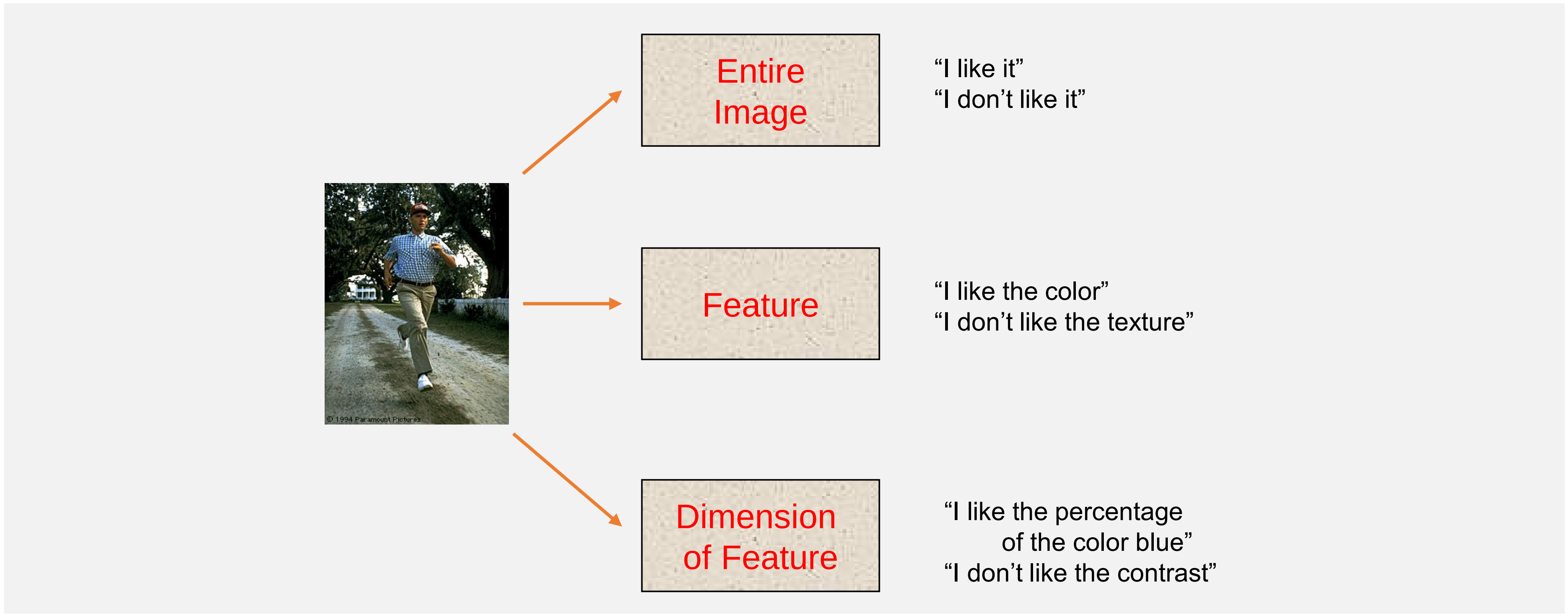


7.2 Capturing Feedback

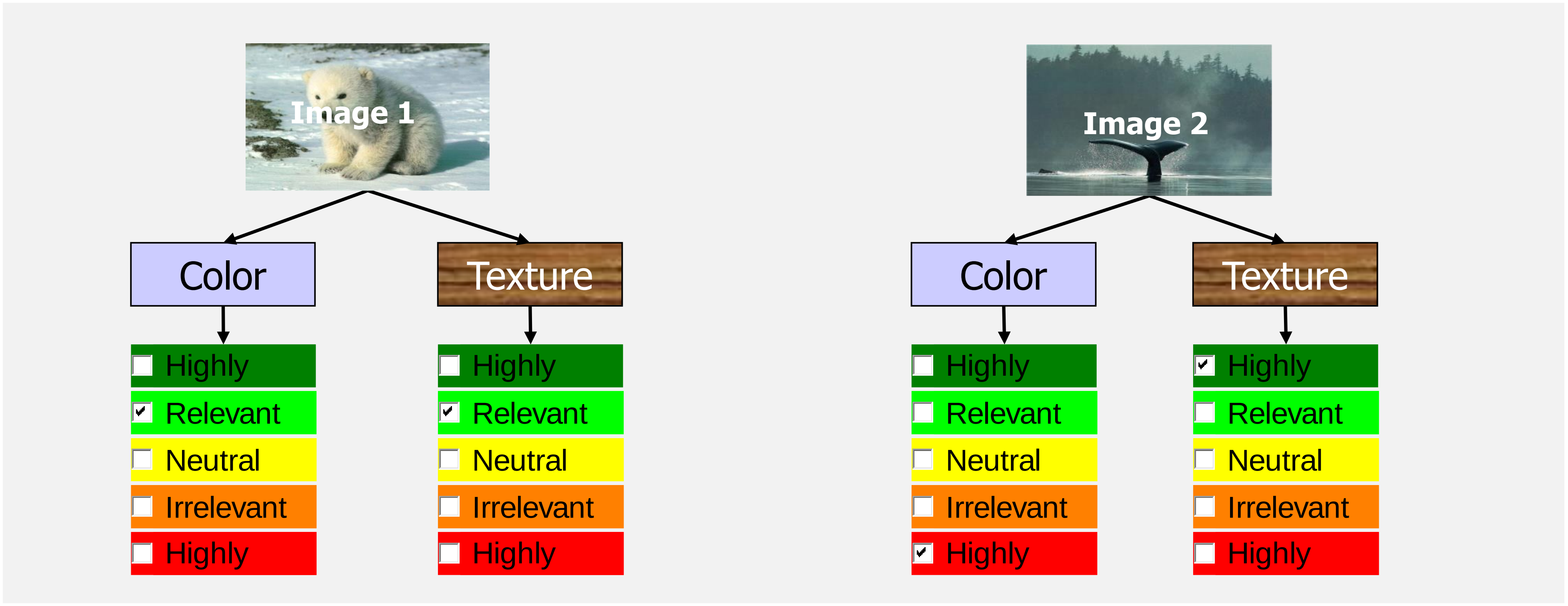
- The search process is sketched below
  - in a first step, the user provides an initial query that produces a first set of possible answers
  - if the answer is not sufficient, the user can provide positive and/or negative feedback which the engine uses to refine the query
  - the second step is repeated as long as the user is not satisfied
- The expectation is that, with enough feedback, the real intentions of the user are better captured. This seems especially useful in situation where the semantic gap prevents the user to really tell what he or she is looking for.



- We can capture feedback at different granularities
  - entire document: no particular information is given on why the document is good (or bad)
  - feature: the user provides specific feedback to a single aspect of a document (e.g., color)
  - dimension: the user assess a single dimension of a feature (e.g., too much blue)
- Obviously, we need to carefully assess how far we go with capturing feedback. Not all users are able to assess all aspects we can think of. Often, the most simple approach (yes/no) leads to best responses from the users; however, the more detailed feedback we can get, the better our chances are to adjust the query


© 1994 Paramount Pictures
Entire Image
"I like it"
"I don't like it"
Feature
"I like the color"
"I don't like the texture"
Dimension of Feature
"I like the percentage of the color blue"
"I don't like the contrast"

- Another dimension is the scale of relevance
  - we can ask users to rate beyond simple yes/no answer with a scale of values from highly dissimilar to highly similar
  - we can also simply rate between -5..5 or come up with any other sophisticated scale
- At first, it may feel like a good idea to provide more options of how much you like/dislike a document in the result list. On the other hand, the more values and options you provide, the more intimidated users become and may feel overwhelmed with the options at hand.



7.3 Feedback Methods for Text

- **Boolean Retrieval:** a simple method to incorporate feedback
  - Let $\boldsymbol{q}$ be the query, $\mathbb{R}$ be the set of relevant documents, and $\mathbb{N}$ be the set of non-relevant documents. The query is a binary, M-dimensional vector (M is the number of terms)
  - Based on the feedback, we add or remove terms and simply 'AND' them with the query. We can compute the set of terms to extend with two approaches
    - **Feedback with the relevant documents only:** to ensure that relevant document remains in the result set, we can only add terms that appear in all relevant documents. This leads to the following possible set of new query terms:

$$\forall j: 0 \le j < M, \hat{q}_j = \begin{cases} 1 & |\mathbb{R}| = \sum_{d \in \mathbb{R}} d_j \\ 0 & otherwise \end{cases}$$

      To reduce the number of terms (query costs), we can limit the search to at most k terms. To select the best terms, we select them based on $tf \cdot idf$ weights calculated over the set of relevant documents
    - **Feedback with the relevant and non-relevant documents:** we still need to ensure that all relevant documents contain the new query term, but if it appears in too many non-relevant documents, we may want to dismiss it

$$\forall j: 0 \le j < M, \hat{q}_j = \begin{cases} 1 & |\mathbb{R}| = \sum_{\boldsymbol{d} \in \mathbb{R}} d_j \;\wedge\; \alpha|\mathbb{N}| > \sum_{\boldsymbol{d} \in \mathbb{N}} d_j \\ 0 & otherwise \end{cases}$$

- **Vector Space Retrieval:** we can do more than just adding/removing terms
  - Let $\mathbb{R}$ be the set of relevant documents and $\mathbb{N}$ the set of non-relevant documents
  - Rocchio (1971) proposed a method to adjust a query based on document representations (vectors) rather than adding/removing single terms. He considered queries and documents as vectors; the j–th component represents term t_j with its $tf \cdot idf$ value
  - The basic idea is that relevant documents should attract the query while non-relevant should repel it. In vector notation, this means:

$$\hat{q} = \alpha \cdot q + \frac{\beta}{|\mathbb{R}|}\sum_{\boldsymbol{d} \in \mathbb{R}} \boldsymbol{d} - \frac{\gamma}{|\mathbb{N}|}\sum_{\boldsymbol{d} \in \mathbb{N}} \boldsymbol{d}$$

  - We obtain good values for α, β and γ through experiments. Schäuble (1997) found that $\alpha = 1.0$, $\beta = 0.75$ and $\gamma = 0.25$ worked best for his web search engine based on vector space retrieval.
  - As before with Boolean Retrieval, Rocchio's formula can result in many non-zero query components (terms used for the subsequent query). To speed up retrieval, we can
    - eliminate components that have very small absolute values (threshold based filtering)
    - keep the k components with the highest absolute value
  - We can repeat the process multiple times to obtain larger sets of relevant and non-relevant documents to obtain a stronger lead to the "right" location in the term space

- **Probabilistic Retrieval:** see in chapter 2

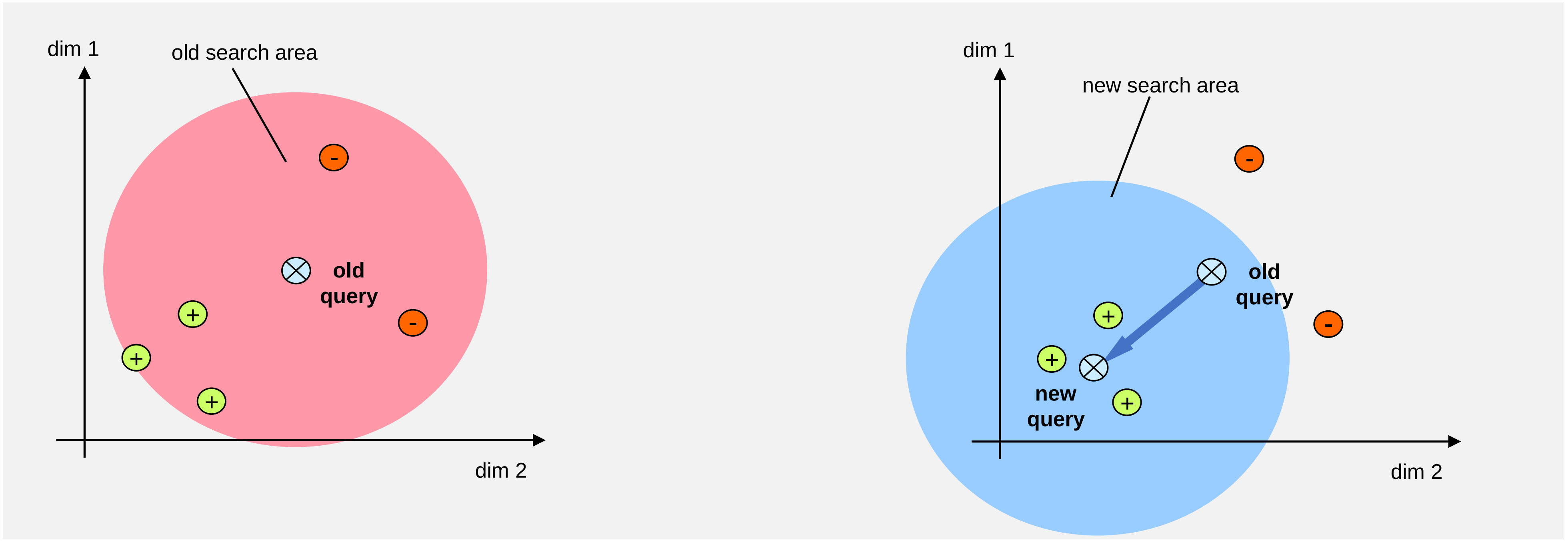
7.4 Feedback Methods for Low-level Features

- Feedback methods are especially useful to close the semantic gap for low-level features. An initial reference object may not be good enough to express the information need. But with sufficient feedback on intermediate results, a search engine can better capture the user’s real intent.
- With image retrieval, users can quickly identify relevant and non-relevant results; entering feedback is simpler and is thus more often used as with text retrieval. Similarly, for short video and audio sequences.
- In the following, we consider several methods to incorporate feedback into the query process. We assume that
  - low-level features for multimedia documents are some dimensional vectors
  - one or more features, and one or more reference objects exist
- In addition, it is possible to combine the relevance feedback method for text to keyword based features or to apply them to attributes and class memberships (e.g., high-level features).

- A first simple method is based on Rocchio: the query vector is combined with the vectore of relevant and non-relevant documents. The new vector should lie closer to the vectors of relevant documents, and farther away from the vectors of non-relevant document.
- Let $\boldsymbol{q}$ be the query vector, let $\mathbb{R}$ be the set of relevant documents and let $\mathbb{N}$ be the set of non-relevant documents. The new query vector is determined as follows

$$\hat{q} = \alpha \cdot q + \frac{\beta}{|\mathbb{R}|}\sum\nolimits_{\boldsymbol{d}\in\mathbb{R}} \boldsymbol{d} - \frac{\gamma}{|\mathbb{N}|}\sum\nolimits_{\boldsymbol{d}\in\mathbb{N}} \boldsymbol{d}$$

- We obtain good values for α, β and γ through experiments (we can not use the ones for text)
- Alternative (physical) models are possible as alternatives. For instance, some have suggested to use spring models or gravity models to adjust the position of the query vectors

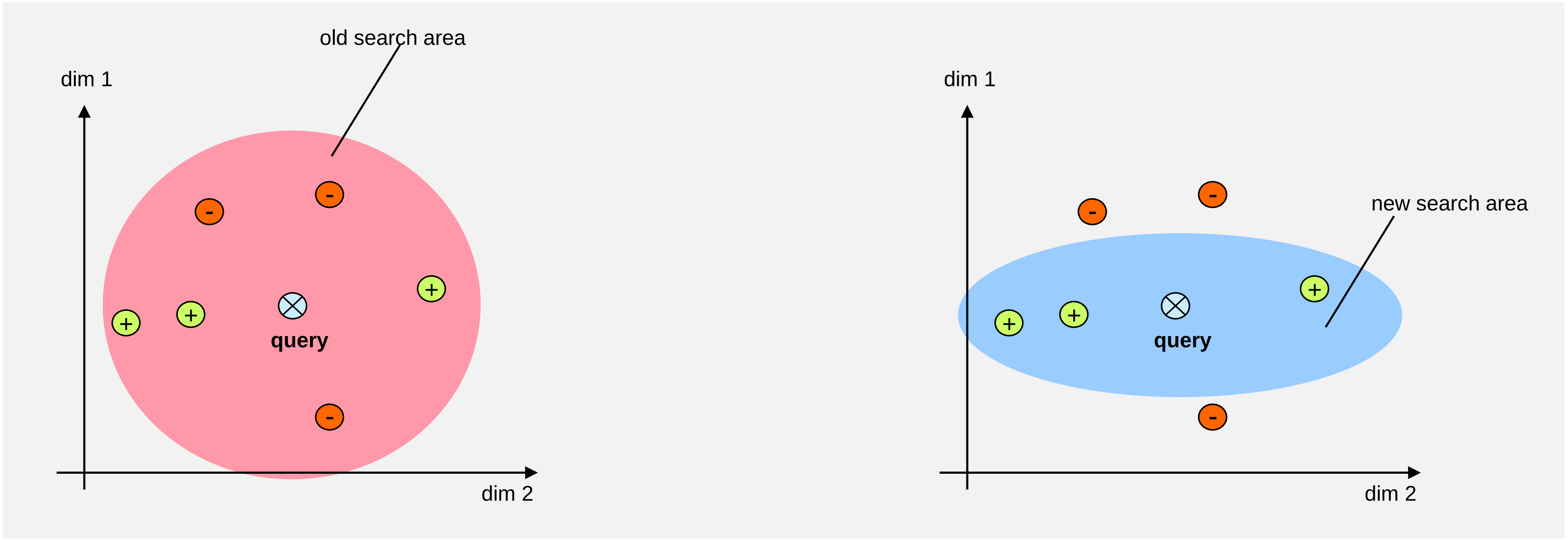


- In many cases, we may want to control how important a single dimension (or an aspect of the feature) is and adjust weights in the distance metrics. For instance, with Euclidean distances:

$$\delta(\boldsymbol{q}, \boldsymbol{p}_i) = \sqrt{\sum_j w_j \cdot \left(q_j - p_{i,j}\right)^2}$$

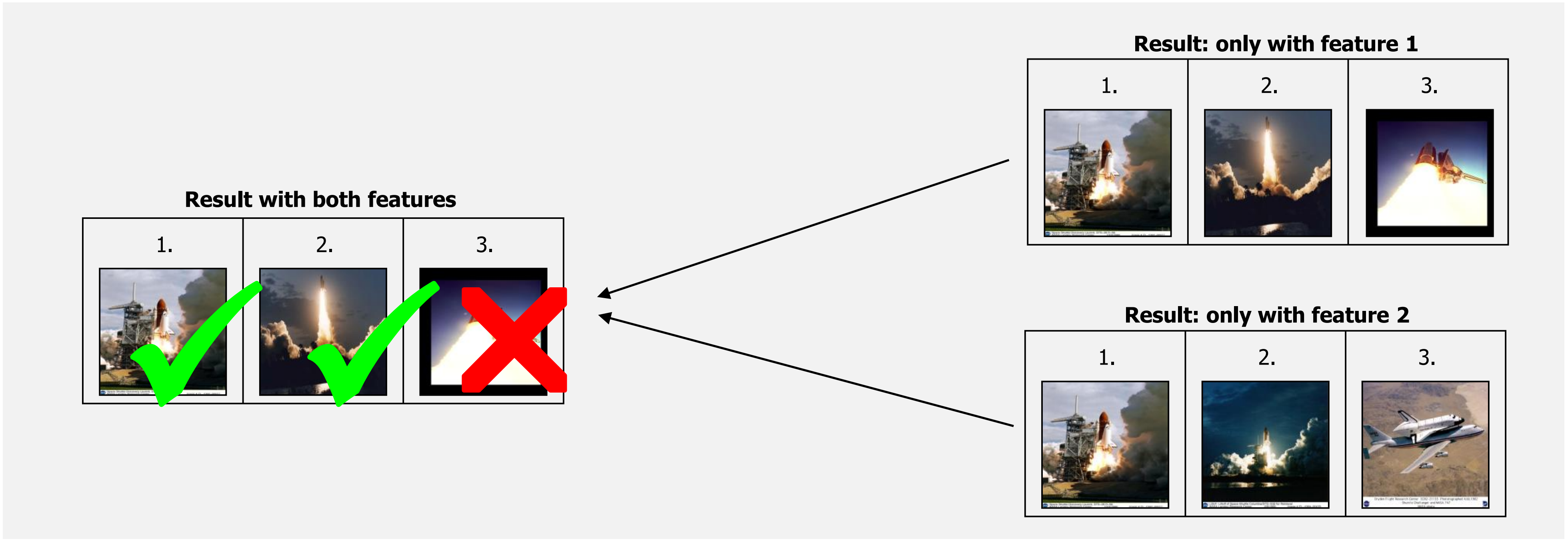
- In the example below, we observe that the relevant (+) objects all lie in a very narrow range in dimension 1 while the range of values in dimension 2 is large. A simple method to weight is to compute the standard deviation of values of relevant documents along each dimension:

$$w_j = \frac{1}{std\left\{d_j \mid \boldsymbol{d} \in \mathbb{R}\right\}}$$

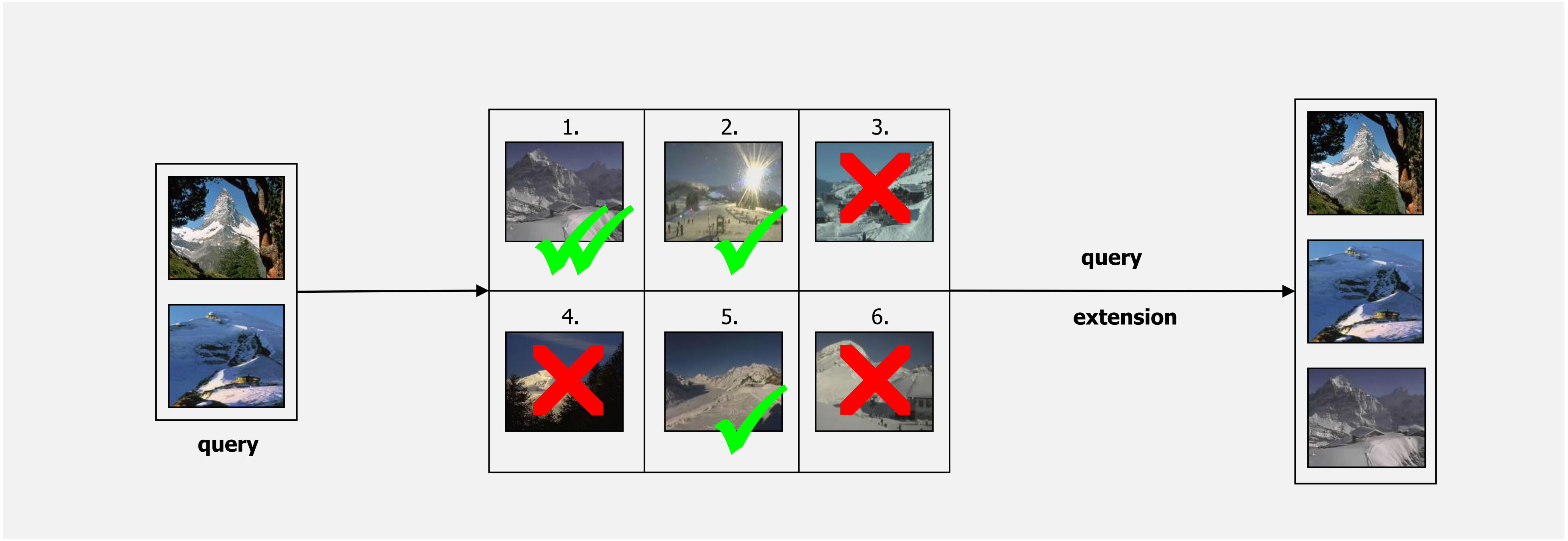


- With multi-feature and multi-object queries, we can weight the individual features and objects with the help of the feedback. The weights should express how well the feature or the object matches the user's intent. A simple method is as follows:
  - compute the results but use only a single feature (or object)
  - count how many relevant object from the overall result appear in these sub-queries
  - the higher the count, the higher the weights

- Example below: the sub-query with only feature 1 yields 2 relevant images while the sub-query with only feature 2 yields only one relevant query. We can use the counts to compute weights as follows: $w_1 = 2/3$ and $w_2 = 1/3$.


Result: only with feature 1
1.
2.
3.
Result with both features
1.
2.
3.
Result: only with feature 2
1.
2.
3.

- Finally, we can alter the entire structure of the query by:
  - adding a new reference object
  - removing a bad reference object
  - adding new, better suited features based on feedback
  - removing bad features
- A straightforward approach is to add all relevant objects to the query. More complex approaches compute results for several alternative queries by varying an aspect (much like genetic algorithms) and then select the structure that yields more relevant documents at the top. To be effective, this requires a sufficiently large number of relevant objects.
- Adding new objects does not significantly increase retrieval costs but the number of features can increase search times significantly



- Feedback information is not just useful to refine the current query
  - Feedback from previous loop should be kept to increase the known set of relevant and non-relevant documents for subsequent refinements
  - Once feedback was given, do not ask for it again
  - Non-relevant objects should be eliminated from subsequent refinements even if they still score high values. This is best done with a Black List that contains all objects that the user marked as non-relevant. The query is then extended with a predicate “not in Black List”.
  - We can use feedback from previous searches (also from other users) to prioritize objects that are more often considered relevant than non-relevant. For instance, users may generally dislike an image because of poor quality. This is very much like a spam filter in a web search engine

7.5 References

- [FB92] W.B. Frakes and R. Baeza-Yates: Information Retrieval, Data Structures and Algorithms, Prentice Hall, "Englewood Cliffs, New Jersey, USA, 1992
- [Schäuble97] Peter Schäuble, Multimedia Information Retrieval, Content-based Information Retrieval from Large Text and Audio Databases, Kluwer Academic Publishers, Zurich, Switzerland, 1997
- [BR99] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, Modern Information Retrieval, Addison-Wesley, 1999, ISBN 0-201-39829-X (weitere Infos zum Buch unter: **`http://www.dcc.ufmg.br/irbook/`**)