Evaluating Information Retrieval Algorithms with Significance Testing Based on Randomization and Student's Paired t-Test

Avinash Kak

Purdue University

June 14, 2023
2:43pm

An RVL Tutorial Presentation

Originally presented in Fall 2019

CONTENTS

Back to TOC

1: Precision and Recall for Characterizing a Retrieval Algorithm

- Let's say we are evaluating the performance of an information retrieval algorithm on a database with 100 documents in it. Let's see how we might characterize what the algorithm retrieves for a specific query q.

- A retrieval algorithm will, in general, return a ranked list of documents from the database. For the given query q, let's say that the returned list looks like what is shown on the next page in Table 1.

- In order to characterize the performance of the retrieval algorithm, you need to know the "ground truth", which, in this case, is the list of documents that are known to be relevant to the query q. Let's say that only the documents d_3, d_6, d_7, d_9, and d_{10} are relevant to query q. Assume all other documents are irrelevant.

- The performance of a retrieval algorithm is measured by two properties: Precision and Recall at a given rank r.

- We denote the Precision at rank r by $P@r$ and the Recall at rank

Ranked list of the docs retrieved for query q that has 5 relevant docs in the database	**Is the retrieved document relevant to query q ?**
d_1	
d_2	
d_3	yes
d_4	
d_5	
d_6	yes
d_7	yes
d_8	
d_9	yes
d_{10}	yes
d_{11}	
d_{12}	
d_{13}	
d_{14}	
d_{15}	
.	
.	
.	
d_{100}	

Table 1: An example of the 100 items returned in the form of a ranked list by an IR algorithm for a query q.

r by $R@r$.

- Given rank r, we define $P@r$ and $R@r$ as follows:

$$P@r = \frac{|retrieved@r \bigcap relevant|}{|retrieved@r|}$$

$$R@r = \frac{|retrieved@r \bigcap relevant|}{|relevant|}$$

where *retrieved@r* denotes the *set* of the r top-ranked documents in what's returned by the algorithm. On the other hand, *relevant* denotes the set of **all** the documents in the dataset that are **relevant** to the query. The symbol || denotes the cardinality of the sets involved — that is, the number of items in the sets. Since the cardinality of the set $retrieved@r$ is equal to r, we have $|retrieved@r| = r$. Therefore, the definitions shown above can also be expressed as

$$P@r = \frac{|retrieved@r \bigcap relevant|}{r}$$

$$R@r = \frac{|retrieved@r \bigcap relevant|}{|relevant|}$$

- The numerators for both $P@r$ and $R@r$ are the same and, for a given query q, are equal to the number of items that are actually relevant to q among the r topmost returned items.

- The denominators, however, are different. For $P@r$, the denominator is the *number* of documents returned up to rank r — that would obviously be just r itself. For $R@r$, the denominator is independent of rank and equal to the *total number* of the documents in the dataset that are relevant to q.

- Even more succinctly, $P@r$ measures the fraction of the top-ranked r documents that are actually relevant to the query. And $R@r$ measures the fraction of all the relevant documents that show up in the top-ranked r documents.

- Using these formulas, the $P@r$ and $R@r$ values for the retrieval in Table 1 are shown in Table 2. [In Table 2, for values of rank r from 15 through 100, $P@r$ decreases monotonically as $5/r$. On the other hand, for the same range of r values, $R@r$ remains constant at 1.]

- Figure 1 shows at the top a point-plot of the Precision vs. Recall values and, at the bottom, a Precision-vs.-Recall curve drawn through the points for the Precision at rank and Recall at rank values shown in Table 2.

- Regarding the vertical drops you see in the figure at the bottom of Figure 1: Starting at the origin, a walk along the red curve corresponds to you scanning the retrieved list from the highest ranked documents to the lowest ranked documents. The vertical drops in the curve are caused by those documents that are

irrelevant to the query. [When you run into an irrelevant document during the scan, the Precision decreases, while the Recall remains unchanged.]

- To show that the shape of the $P@r$ vs. $R@r$ curve can change dramatically depending on the manner in which the top-ranked retrieved documents are relevant to the query, consider the retrieval shown in Table 3.

- Table 4 shows the $P@r$ and $R@r$ values for the retrieval example in Table 3 where the top-ranked five documents are all relevant to the query; all others are irrelevant. [In Table 4, for values of rank r from 15 through 100, $P@r$ decreases monotonically as $5/r$. On the other hand, $R@r$ remains constant at 1.] Figure 2 shows the $P@r$ and $R@r$ values in the form of a point plot and as a Precision-vs.-Recall curve.

- In the plot at the bottom of Figure 2, note how the $P@r$ values remain constant through consecutive retrievals as we start at the top of the ranked list of retrievals and work our way down. That happens until we reach the $R@r$ value of 1, at which point all the subsequent $P@r$ values fall on a vertical line.

- For our third example of what the $P@r$ vs. $R@r$ curve may look like for a retrieval algorithm, consider the retrieval shown in Table 5.

rank	Precision at rank	Recall at rank
1	$P@1 = 0$	$R@1 = 0$
2	$P@2 = 0$	$R@2 = 0$
3	$P@3 = 1/3$	$R@3 = 1/5$
4	$P@4 = 1/4$	$R@4 = 1/5$
5	$P@5 = 1/5$	$R@5 = 1/5$
6	$P@6 = 2/6 = 1/3$	$R@6 = 2/5$
7	$P@7 = 3/7$	$R@7 = 3/5$
8	$P@8 = 3/8$	$R@8 = 3/5$
9	$P@9 = 4/9$	$R@9 = 4/5$
10	$P@10 = 5/10 = 1/2$	$R@10 = 5/5 = 1$
11	$P@11 = 5/11$	$R@11 = 5/5 = 1$
12	$P@12 = 5/12$	$R@12 = 5/5 = 1$
13	$P@13 = 5/13$	$R@13 = 5/5 = 1$
14	$P@14 = 5/14$	$R@14 = 5/5 = 1$
15	$P@15 = 5/15 = 1/3$	$R@15 = 5/5 = 1$
.	.	.
.	.	.
.	.	.
100	$P@100 = 5/100$	$R@100 = 5/5 = 1$

Table 2: $P@r$ and $R@r$ values at different ranks for the retrieval result shown in Table 1.

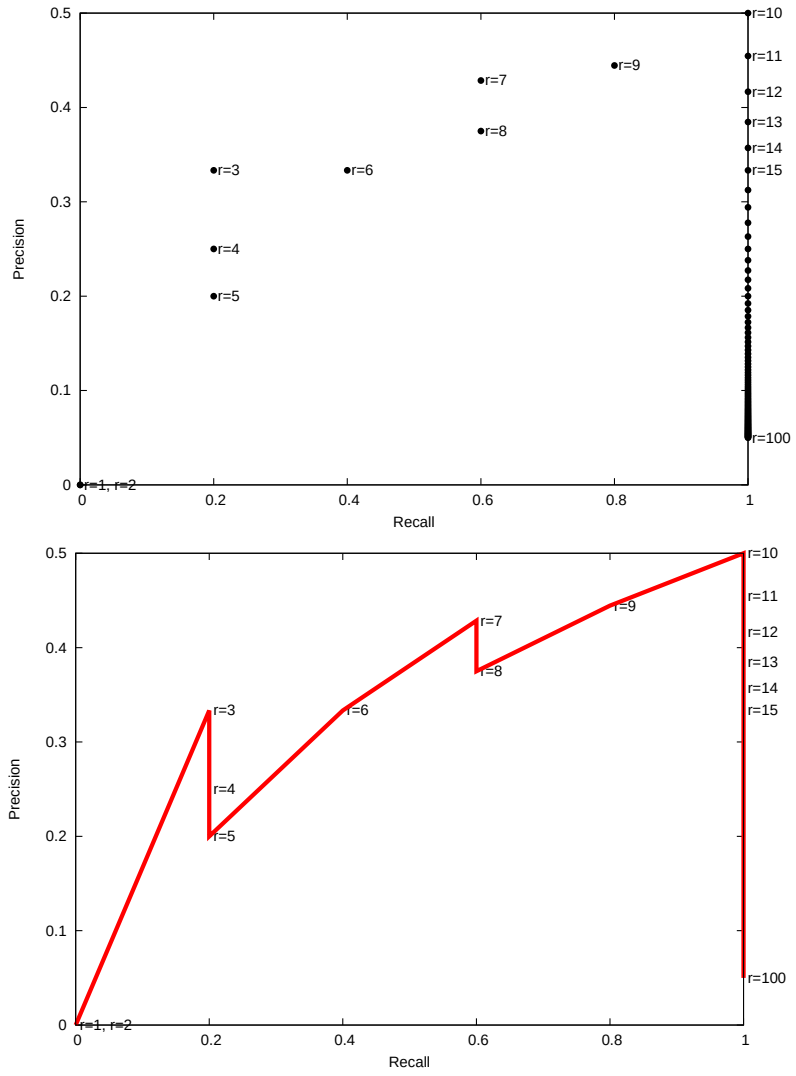


Figure 1: Shown above is a point-plot of $P@r$ vs. $R@r$ for the retrieval in Table 1. Shown in the lower figure is a curve drawn through the point plot.

Ranked list of the docs retrieved for query q that has 5 relevant docs in the database	**Is the retrieved document relevant to query q ?**
d_1	yes
d_2	yes
d_3	yes
d_4	yes
d_5	yes
d_6	
d_7	
d_8	
d_9	
d_{10}	
d_{11}	
d_{12}	
d_{13}	
d_{14}	
d_{15}	
.	
.	
.	
d_{100}	

Table 3: Another example of the 100 items returned in the form of a ranked list by an IR algorithm for a query q.

rank	Precision at rank	Recall at rank
1	$P@1 = 1$	$R@1 = 1/5$
2	$P@2 = 1$	$R@2 = 2/5$
3	$P@3 = 1$	$R@3 = 3/5$
4	$P@4 = 1$	$R@4 = 4/5$
5	$P@5 = 1$	$R@5 = 5/5 = 1$
6	$P@6 = 5/6$	$R@6 = 5/5 = 1$
7	$P@7 = 5/7$	$R@7 = 5/5 = 1$
8	$P@8 = 5/8$	$R@8 = 5/5 = 1$
9	$P@9 = 5/9$	$R@9 = 5/5 = 1$
10	$P@10 = 5/10 = 1/2$	$R@10 = 5/5 = 1$
11	$P@11 = 5/11$	$R@11 = 5/5 = 1$
12	$P@12 = 5/12$	$R@12 = 5/5 = 1$
13	$P@13 = 5/13$	$R@13 = 5/5 = 1$
14	$P@14 = 5/14$	$R@14 = 5/5 = 1$
15	$P@15 = 5/15 = 1/3$	$R@15 = 5/5 = 1$
.	.	.
.	.	.
.	.	.
100	$P@100 = 5/100$	$R@100 = 5/5 = 1$

Table 4: $P@r$ and $R@r$ values at different ranks for the retrieval result shown in Table 3.

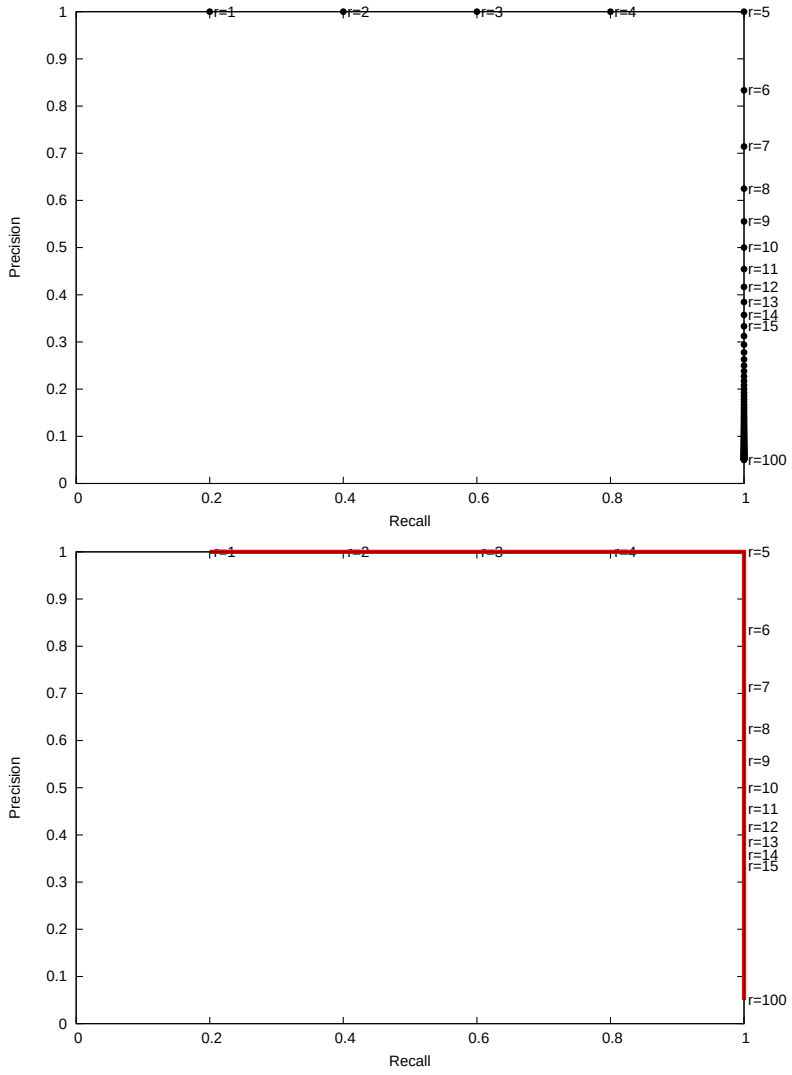


Figure 2: Shown above is a point-plot of $P@r$ vs. $R@r$ for the retrieval in Table 3. Shown below is a curve drawn through the point plot.

- Table 6 shows the $P@r$ and $R@r$ values for the retrieval depicted in Table 5. [In Table 5, for values of rank r from 15 through 100, $P@r$ decreases monotonically as $5/r$ and $R@r$ remains constant at 1.] Figure 3 shows the $P@r$ and $R@r$ values both in the form of a point plot and as a Precision-vs.-Recall curve.

- Overall, the relevancy pattern shown in Table 5 is not that radically different from what you see in Table 1, yet the Precision-vs.-Recall curves for the two cases look so dramatically different.

- In much academic literature, what you see for a Precision-vs.-Recall curve is a smoothed version of the curve shown in Figure 3.

- However, for real retrieval algorithms and real databases, the curve could take any form along the lines of what you see in Figures 1, 2, and 3.

- In all cases, though, the largest value of Precision at any rank is bounded by 1.0, which happens when **all** of the items retrieved up to that rank are relevant to the query. And the Relevance is also bounded by 1.0, which happens at a rank at which all the retrievals up to that rank include **all** documents relevant to the

Ranked list of the docs retrieved for query q that has 5 relevant docs in the database	**Is the retrieved document relevant to query q ?**
d_1	yes
d_2	
d_3	yes
d_4	
d_5	
d_6	yes
d_7	
d_8	
d_9	
d_{10}	yes
d_{11}	
d_{12}	
d_{13}	
d_{14}	
d_{15}	yes
.	
.	
.	
d_{100}	

Table 5: A third example of the 100 items returned in the form of a ranked list by an IR algorithm for a query q.

rank	Precision at rank	Recall at rank
1	$P@1 = 1$	$R@1 = 1/5$
2	$P@2 = 1/2$	$R@2 = 1/5$
3	$P@3 = 2/3$	$R@3 = 2/5$
4	$P@4 = 2/4 = 1/2$	$R@4 = 2/5$
5	$P@5 = 2/5$	$R@5 = 2/5$
6	$P@6 = 3/6 = 1/2$	$R@6 = 3/5$
7	$P@7 = 3/7$	$R@7 = 3/5$
8	$P@8 = 3/8$	$R@8 = 3/5$
9	$P@9 = 3/9$	$R@9 = 3/5$
10	$P@10 = 4/10 = 2/5$	$R@10 = 4/5$
11	$P@11 = 4/11$	$R@11 = 4/5$
12	$P@12 = 4/12 = 1/3$	$R@12 = 4/5$
13	$P@13 = 4/13$	$R@13 = 4/5$
14	$P@14 = 4/14$	$R@14 = 4/5$
15	$P@15 = 5/15 = 1/3$	$R@15 = 5/5 = 1$
16	$P@15 = 5/16$	$R@16 = 5/5 = 1$
.	.	.
.	.	.
.	.	.
100	$P@100 = 5/100$	$R@100 = 5/5 = 1$

Table 6: $P@r$ and $R@r$ values at different ranks for the retrieval result shown in Table 5.

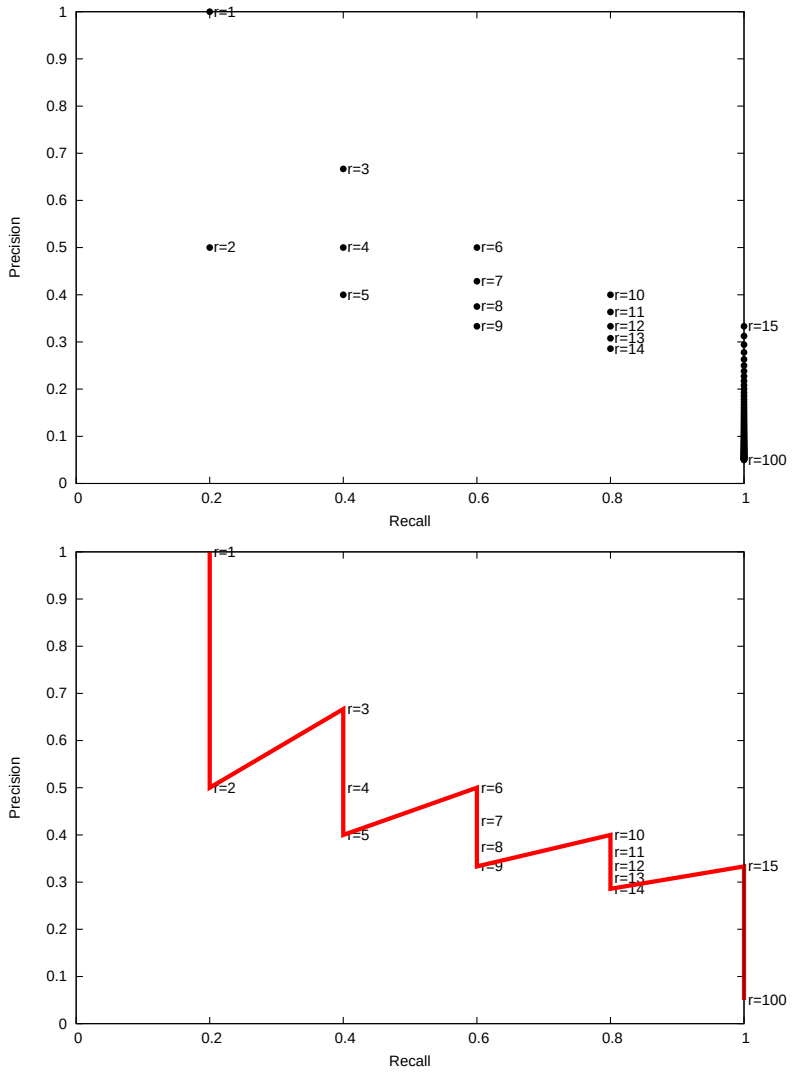


Figure 3: Shown above is a point-plot of $P@r$ vs. $R@r$ for the retrieval in Table 5. Shown below is a curve drawn through the point plot.

query.

- To further enhance our intuitions about the nature of Precision-vs.-Recall curves, in the next two sections I will consider the following two limiting cases of retrieval algorithms: (1) Retrieval by a monkey (meaning a completely random retrieval from a database); and (2) a perfect retrieval algorithm that always returns the relevant documents at the topmost ranks.

Back to TOC

2: Precision vs. Recall for Retrieval by a Monkey

- Monkeys are my favorite animals. You can see them frolicking in the trees in several parts of India. Baby monkeys particularly are a lot of fun to watch.

- Let's have a monkey retrieve documents for us from a database. Not to underestimate the intelligence of the monkeys, let's just say that the monkey is bored and it merely carries out a purely random retrieval.

- Let D represent a database of documents and let's assume that, on the average, c percent of the database is relevant to each query in the Q of queries.

- Let's further assume that the **the relevant documents are uniformly distributed** in the ranked list of documents retrieved by the monkey.

- Let's now consider the following values for the rank r: $\{c|D|, 2c|D|, 3c|D|, \ldots, |D|\}$, where $|D|$ is the cardinality of the

set D. If $c = 0.1$, we will consider the following ranks: $\{\frac{|D|}{10}, \frac{2|D|}{10}, \frac{3|D|}{10}, \ldots, |D|\}$.

- Consider the first in the list of rank values shown above: $r = c|D|$. At this rank, we can expect that, on the average, there will be c fraction of the r documents that will be relevant to the query. That gives us a total of $c^2|D|$ relevant retrievals up to rank r. So we can set $R@r = \frac{c^2|D|}{c|D|} = c$ since $c|D|$ is the total number of documents relevant to any query.

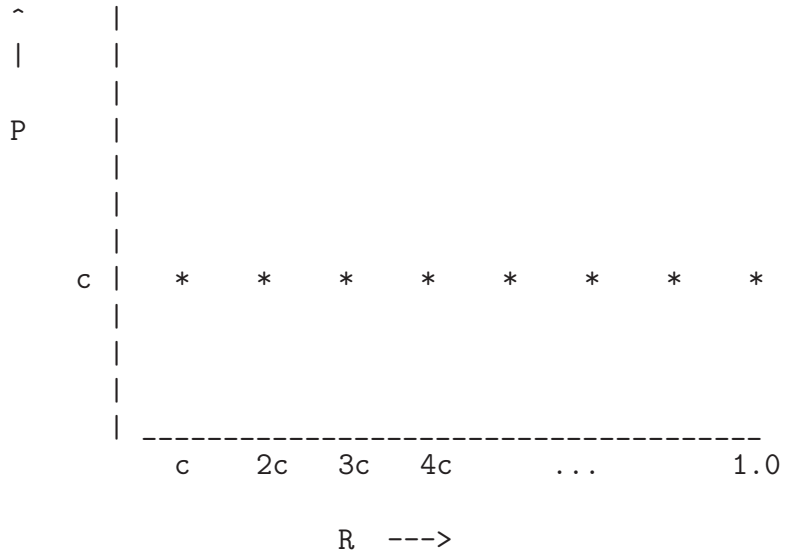
- For obvious reasons, when rank r is $c|D|$, $P@r$ will be given by $P@r = \frac{c^2|D|}{c|D|} = c$ since, up to this rank the total number of documents that are retrieved is $c|D|$ and since the numerator remains the same in the precision and recall calculations.

- We can carry out similar reasoning at rank $r = 2c|D|$. On account of the uniform distribution of the relevant documents in the ranked retrieved list of $|D|$ documents, we expect that, on the average, there will be c fraction of the $2c|D|$ documents that will be relevant. Therefore, at the new rank we are considering, the number of retrieved documents that will be relevant will, on the average, be $2c^2|D|$. Now we can use this as the numerator in our $P@r$ and $R@r$ calculation at this new rank.

- Continuing our reasoning for rank $r = 2c|D|$, since the

denominator for $P@r$ is the rank itself, we get $P@r = \frac{2c^2|D|}{2c|D|} = c$, which is the same as the precision at the previous rank we considered. And since the denominator for $R@r$ is the total number of relevant documents, which is $c|D|$, we get $R@r = \frac{2c^2|D|}{c|D|} = 2c$.

- If you continue this line of reasoning, you will see the following sort of Recall-Precision curve for retrieval by a monkey:



Back to TOC

3: Precision vs. Recall for Retrieval by an Oracle

- We again assume that D represents the entire database and that, on the average, fraction c of the database is relevant to each query q in the set Q of queries.

- As before, let's examine retrievals at the ranks: $\{c|D|, 2c|D|, 3c|D|, \ldots, |D|\}$. If $c = 0.1$, we will consider as before the following ranks: $\{\frac{|D|}{10}, \frac{2|D|}{10}, \frac{3|D|}{10}, \ldots, |D|\}$.

- Consider the first in the list of rank values shown above: $r = c|D|$. At this rank, we can expect that the oracle, being an oracle after all, will recover all the relevant $c|D|$ documents. That is, of all the $c|D|$ retrieved documents, every one of them will be relevant.

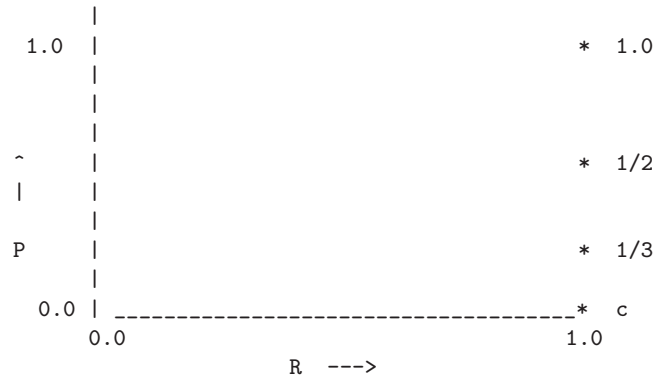
- This implies that, for any query q_i in Q, both the precision and the recall at rank $r = c|D|$ will, on the average, be equal to 1. In other words, if it were possible to somehow average out the precision values in a small neighborhood around the rank $r = c|D|$, we would see $P@r = 1$ and $R@r = 1$ at that rank. Note that we are already at the end of the Recall axis.

- Consider next the rank $r = 2c|D|$. At this rank, for any of the queries, $R@r$ will again be 1. On the other hand, on the average considering all the queries, the $P@r$ will now be 0.5 since only half the retrieved documents will be relevant for any query. In other words, when r is $2c|D|$, $P@r = 0.5$ and $R@r = 1$.

- If we carry out the same kind of reasoning at rank $r = 3c|D|$, we get $P@r = \frac{1}{3}$ and $R@r = 1$ for that value of r.

- Finally, at rank $r = |D|$, we get $P@r = \frac{c|D|}{|D|} = c$ and $R@r = 1$.

- If we plot the values obtained, we get



- We see that for the oracle, if we only consider the ranks $\{c|D|, 2c|D|, 3c|D|, \ldots, |D|\}$, all calculated precision vs. recall values fall on a vertical line at the end point of the Recall axis.

- The "problem" that all of the P-R values fall on a vertical at the end of the R axis is caused by the fact that after the rank $r = c|D|$, the first $c|D|$ retrievals will always be the same for any query — these would be all the documents that are relevant to the query.

- Now consider the following additional ranks that are within the first of the rank values considered so far. That is, we will consider ranks within the interval $[1, c|D|]$. In particular, we will consider the ranks $\{\frac{1}{4}c|D|, \frac{1}{2}c|D|, \frac{3}{4}c|D|, c|D|\}$.

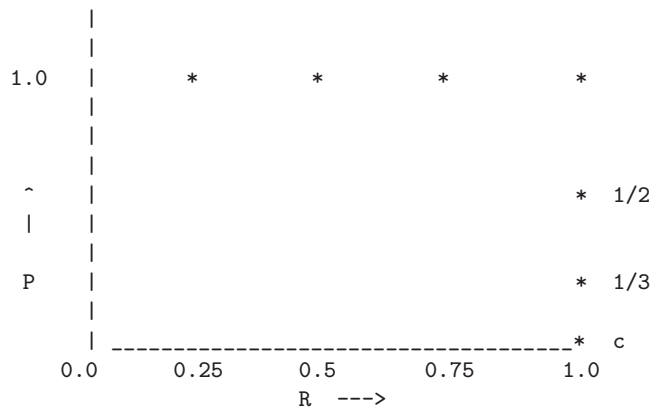
- At rank $r = \frac{1}{4}c|D|$, all of the $\frac{1}{4}c|D|$ documents will be relevant, causing the precision to equal 1. However, the recall at this rank will only be $\frac{1}{4}$ since only a fourth of all the relevant documents will be retrieved. So we have $P@r = 1.0$ and $R@r = 0.25$ when r is equal to $r = \frac{1}{4}c|D|$.

- Similar reasoning leads to the fact that, on the average for any of the queries, $P@r = 1.0$ and $R@r = 0.5$ when rank r is equal to $\frac{1}{2}c|D|$. And $P@r = 1.0$ and $R@r = 0.75$ when r equals $\frac{3}{4}c|D|$.

- We previously calculated the values $P@r = 1.0$ and $R@r = 1.0$ when r equals $c|D|$.

- When we place all of these points on the recall-precision curve, we

get for an oracle:



Back to TOC

4: Average Precision for a Query q_i

- Given a query q_i, let the documents in the database relevant to this query by denoted by the set $\{d_{i_1}, d_{i_2}, d_{i_3}, \ldots, d_{i_m}\}$.

- And let the ranks at which these documents are retrieved be denoted by the set $\{r_{i_1}, r_{i_2}, r_{i_3}, \ldots, r_{i_m}\}$, where we assume that the rank r_{i_k} corresponds to the document d_{i_k} in the list of database documents relevant to query q_i.

- Using the notation presented in the previous section, let $P@r_{i_k}$ and $R@r_{i_k}$ denote the Precision and Recall at rank r_{i_k}.

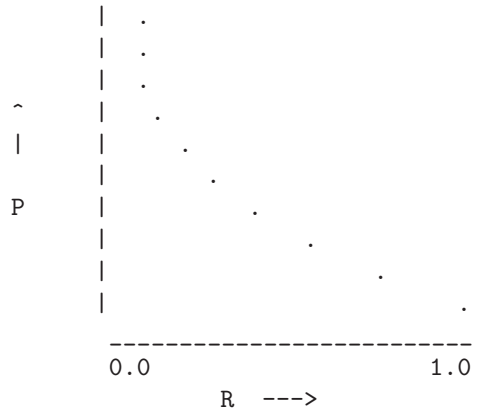
- We can say that the relevant document d_{i_k} for query q_i is retrieved with precision $P@r_{i_k}$.

- The Average Precision associated with all the database documents that are relevant to query q_i is then given by

$$P_{av}(q_i) = \frac{1}{m}\sum_{k=1}^{m} P@r_{i_k}$$

- Very roughly speaking, $P_{av}(q_i)$ equals the area under the

Precision-vs.-Recall curve for query q_i. To justify this claim, let's say that this curve looks like



- Each point on the Recall axis for which we have at least one value for Precision will correspond to one of the relevant documents for query q_i. Considering that the Recall values are confined to the interval $[0, 1]$, the average Precision value calculated at the sampling points for Recall is approximately the area under the curve.

Back to TOC

5: Mean Average Precision (MAP) for a Retrieval Algorithm

- One way to characterize the retrieval power of an information retrieval algorithm (vis-a-vis a given database) **with a single numeric measure** is through what is known as the Mean Average Precision (MAP).

- You have already seen in Section 4 the formula for the Average Precision you can get from a retrieval algorithm for a given query q_i.

- Assume now that we have a set of queries $Q = \{q_1, q_2, \ldots, q_n\}$ for the purpose of evaluating the retrieval power of an algorithm and that we know in advance what database documents are relevant to each query.

- Let's use m_i to denote the number of relevant documents in the database for query q_i.

- The MAP value is obtained by the taking the mean of the Average Precision $P_{av}(q_i)$ over all the queries in the set Q:

$$\begin{aligned} MAP &= \frac{1}{|Q|}\sum_{q_i \in Q} P_{av}(q_i) \\ &= \frac{1}{|Q|}\sum_{q_i \in Q} \frac{1}{m_i}\sum_{k=1}^{m_i} P@r_{i_k} \end{aligned}$$

- The MAP for our monkey retriever is c where c is the fraction of the database relevant to each query in Q and D the database. For a database consisting of, say, 1000 documents and with c equal to 0.1, the MAP for the monkey retriever would be 0.1. On the other hand, the MAP for the oracle retriever equals 1.0.

- One more thing before moving on to the next topic: There is the interesting question of how to look at the Precision-Recall curve. As you saw in the Precision vs. Recall plots shown previously, the curve is obviously not a function with Precision as the range and Recall as the domain — since it is possible to have more than one value of $P@r$ for a given value of $R@r$. It is best to think of this curve as a parametric curve in the $(Recall, Precision)$-plane with rank r as the parameter, in much the same way as the parametric forms for circles, ellipses, etc. As you vary the value of the parameter, you generate various points on the curve.

Back to TOC

6: Algorithm::VSM — A Perl Module for Information Retrieval and Significance Testing

- To become more deeply familiar with the notions related to the characterization of retrieval accuracy, it is good to have access to a retrieval engine. Unless you have one of your own, you might want to download my Perl module `Algorithm::VSM` from the open-source CPAN archive:

  http://search.cpan.org/~avikak/Algorithm-VSM/lib/Algorithm/VSM.pm

- `Algorithm::VSM` is a pure-Perl implementation for constructing a Vector Space Model (VSM) or a Latent Semantic Analysis Model (LSA) of a collection of documents and for using such a model for efficient retrieval of documents in response to search words.

- The rest of this section is devoted to making you familiar with this module. But first let's talk about VSM and LSA:

- VSM and LSA models have been around for a long time in the Information Retrieval (IR) community.

- More recently VSM and LSA models have been shown to be

effective in retrieving files/documents from software libraries. For an account of this research, see the paper by Shivani Rao and Avinash Kak:

http://portal.acm.org/citation.cfm?id=1985451.

- VSM modeling consists of:
  - Extracting the vocabulary used in a corpus.
  - Stemming the words so extracted. Stemming means that closely related words like 'programming' and 'programs' are reduced to the common root word 'program'.
  - Eliminating the designated stop words from the vocabulary. These are the non-discriminating words that can be expected to exist in virtually all the documents.
  - Constructing document vectors for the individual files in the corpus. An element of a document vector is the frequency of occurrence of the word corresponding to that element position.
  - Multiplying the word frequencies in each document vector by the word IDFs. IDF stands for *Inverse Document Frequency*. IDF weighting of the words in the document vectors reduces the importance of the words that are non-discriminatory with respect to the retrieval of the documents. A typical formula for calculating the IDF weight for a word is through the logarithm of the ratio of the total number of documents to the number of documents in which the word appears. So, if a word were to appear in all the documents, its IDF would be zero. We will simply refer to the collection of IDF-modified document vectors taken together as the 'term-frequency' matrix for the corpus.
  - Constructing a query vector for the search query after the query is subject to the same stemming, stop-word elimination, and IDF weighting rules that were applied to the corpus. And, lastly,
  - Using a similarity metric to return the set of documents that are most similar to the query vector. The commonly used similarity metric is one based on the cosine distance between two vectors. Also note that all the vectors mentioned here are of the same size, the size of the vocabulary extracted from the corpus.

- LSA modeling is a small variation on VSM modeling. Now you take VSM modeling one step further by subjecting the term-frequency matrix for the corpus to singular value decomposition (SVD).

- By retaining only a subset of the singular values (usually the N largest for some value of N), you can construct reduced-dimensionality vectors for the documents and the queries.

- In VSM, the size of the document vectors and the query vectors is equal to the size of the vocabulary. For large corpora, this size may involve tens of thousands of elements — this can slow down the VSM modeling and retrieval process. So you are very likely to get faster performance with retrieval based on LSA modeling, especially if you store the model once constructed in a database file on the disk and carry out retrievals using the disk-based model.

- The `Algorithm::VSM` module provides the following functionality:
  - You can construct an instance of the module class by

```
my $vsm = Algorithm::VSM->new(
        break_camelcased_and_underscored  => 1,
        case_sensitive          => 0,

        corpus_directory        => "",
        corpus_vocab_db         => "corpus_vocab_db",
        doc_vectors_db          => "doc_vectors_db",
        file_types              => $my_file_types,
        lsa_svd_threshold       => 0.01,
        max_number_retrievals   => 10,
```

```
        min_word_length         => 4,
        normalized_doc_vecs_db => "normalized_doc_vecs_db",
        query_file              => "",
        relevancy_file          => $relevancy_file,
        relevancy_threshold     => 5,
        save_model_on_disk      => 0,
        stop_words_file         => "",
        use_idf_filter          => 1,
        want_stemming           => 1,
);
```

– The values shown on the right of the big arrows are the default values for the constructor parameters. [The actual constructor parameters you'd need to set would depend on how you want to use the module. For example, if you do not wish to save the model information in disk-based hash tables, you'd not need to set the three '_db' parameters listed above.]

– The twelve scripts that you will find in the 'examples' directory of the module installation directory show how you call up the constructor for different usages of this module.

– After you have constructed a new instance of the Algorithm::VSM class, you must now scan the corpus documents for constructing the corpus vocabulary. This you do by:

```
$vsm->get_corpus_vocabulary_and_word_counts();
```

– The only time you do NOT need to call this method is when you are using a previously constructed disk-stored VSM or LSA model for retrieval.

– If you would like to see corpus vocabulary as constructed by the previous call, make the call

```
$vsm->display_corpus_vocab();
```

– Note that this is a useful thing to do only on small test corpora. If you must call this method on a large corpus, you might wish to direct the output to a file. The corpus vocabulary is shown automatically when debug option is turned on.

– Next you must construct document vectors. This is a necessary step after the vocabulary used by a corpus is constructed. (Of course, if you will be doing document retrieval through a disk-stored VSM or LSA model, then you do not need

– to call this method. You construct document vectors through the following call:

```
$vsm->generate_document_vectors();
```

– If you would like to see the document vectors constructed by the previous call, make the call:

```
$vsm->display_doc_vectors();
```

– After you have constructed a VSM model, you call the following method for document retrieval for a given query. The call syntax is:

```
my $retrievals = $vsm->retrieve_with_vsm(\@query);
```

The argument, `@query`, is simply a list of words that you wish to use for retrieval.

– You can display the retrieved document names by calling this method using the syntax:

```
$vsm->display_retrievals( $retrievals );
```

where `$retrievals` is a reference to the hash returned by a call to one of the retrieve methods.

– If after you have extracted the corpus vocabulary and constructed document vectors, you would do your retrieval with LSA modeling, you need to make the following call:

```
$vsm->construct_lsa_model();
```

– The SVD decomposition that is carried out in LSA model construction uses the constructor parameter `lsa_svd_threshold` to decide how many of the singular values to retain for the LSA model.

– A singular value is retained only if it is larger than the `lsa_svd_threshold` fraction of the largest singular value.

– After you have built an LSA model through the call to `construct_lsa_model()`, you can retrieve the document names most similar to the query by:

```
my $retrievals = $vsm->retrieve_with_lsa(\@query);
```

– Subsequently, you can display the retrievals by calling the `display_retrievals($retrieval)` method described previously.

– If you have set the constructor option `save_model_on_disk`, invoking the methods `get_corpus_vocabulary_and_word_counts()` and `generate_document_vectors()` will automatically deposit the VSM model in database files named with the constructor parameters `corpus_vocab_db` and `doc_vectors_db`. Subsequently, you can carry out retrieval by directly using this disk-based VSM model for speedier performance. In order to do so, you must upload the disk-based model by

```
$vsm->upload_vsm_model_from_disk();
```

– Subsequently you call

```
my $retrievals = $vsm->retrieve_with_vsm(\@query);
$vsm->display_retrievals( $retrievals );
```

for retrieval and for displaying the results.

– The rest of this section goes into the calls you must make for precision and recall calculations if you want to measure the retrieval accuracy that you will get with your constructor options.

– Before you can carry out precision and recall calculations to test the accuracy of VSM and LSA based retrievals from a corpus, you need to have available the relevancy judgments for the queries. (A relevancy judgment for a query is simply the list of documents relevant to that query.) Relevancy judgments are commonly supplied by the humans who are familiar with the corpus. But if such human-supplied relevance judgments are not available, you can invoke the following method to estimate them:

```
$vsm->estimate_doc_relevancies();
```

– For the above method call, a document is considered to be relevant to a query if it contains several of the query words. As to the minimum number of query words that must exist in a document in order for the latter to be considered relevant, that is determined by the `relevancy_threshold` parameter in the VSM constructor.

– But note that this estimation of document relevancies to queries is NOT for serious work. The reason for that is because ultimately it is the humans who are the best judges of the relevancies of documents to queries.

– The humans bring to bear semantic considerations on the relevancy determination problem that are beyond the scope of this module. The generated relevancies are deposited in a file named by the constructor parameter `relevancy_file`.

– If you would like to see the document relevancies generated by the previous method, you can call

```
$vsm->display_doc_relevancies()
```

– After you have created or obtained the relevancy judgments for your test queries, you can make the following call to calculate `Precision@rank` and `Recall@rank`:

```
$vsm->precision_and_recall_calculator('vsm');
  or
$vsm->precision_and_recall_calculator('lsa');
```

depending on whether you are testing VSM-based retrieval or LSA-based retrieval.

– A call to `precision_and_recall_calculator()` will normally be followed by the following call

```
$vsm->display_average_precision_for_queries_and_map();
```

for displaying the Precision@rank and Recall@rank values and the final MAP value for the retrieval algorithm.

– The call shown earlier will also display the Average Precision for each query. As defined in Section 4 of this tutorial, the Average Precision for a query is obtained by averaging the precision values over all the documents relevant to that query. And the value of MAP that is printed out is the mean of the Average Precision values over all the queries.

– When human-supplied relevancies are available, you can upload them into the program by calling

```
$vsm->upload_document_relevancies_from_file();
```

These relevance judgments will be read from a file that is named with the `relevancy_file` constructor parameter.

– If in your own script you want access to the average precision values for the different queries, you call

```
$vsm->get_query_sorted_average_precision_for_queries()
```

You will need to call this method if you are carrying out significance testing for comparing two retrieval algorithms (VSM or LSA with difference choices for some of the constructor parameters).

Back to TOC

7: Convenience Scripts in the examples Directory of the Module Algorithm::VSM

- The '**examples**' directory of the **Algorithm::VSM** module contains the following scripts for you:

For Basic VSM-Based Retrieval: For basic VSM-based model construction and retrieval, run the script:

```
retrieve_with_VSM.pl
```

For a Continuously Running VSM Search Engine: If you want to run an infinite loop for repeated retrievals from a VSM model, run the script:

```
continuously_running_VSM_retrieval_engine.pl
```

You can create a script similar to this for doing the same with the LSA model.

For Storing the Model Information in Disk Files: For storing the model information in disk-based DBM files that can subsequently be used for both VSM and LSA retrieval, run the script:

```
retrieve_with_VSM_and_also_create_disk_based_model.pl
```

For VSM-Based Retrieval with a Disk-Stored Model: If you have previously run a script like `retrieve_with_VSM.pl` and no intervening code has modified the disk-stored VSM model of the corpus, you can run the script

```
retrieve_with_disk_based_VSM.pl
```

This would obviously work faster at retrieval since the VSM model would NOT need to constructed for each new query.

For Basic LSA-Based Retrieval: For basic LSA-based model construction and retrieval, run the script:

```
retrieve_with_LSA.pl
```

For LSA-Based Retrieval with a Disk-Stored Model: If you have previously run a script like `retrieve _with_LSA.pl` and no intervening code has modified the disk-stored LSA model of the corpus, you can run the script

```
retrieve_with_disk_based_LSA.pl
```

The retrieval performance of such a script would be faster since the LSA model would NOT need to be constructed for each new query.

For Precision and Recall Calculations with VSM: To experiment with precision and recall calculations for VSM retrieval, run the script:

```
calculate_precision_and_recall_for_VSM.pl
```

Note that this script will carry out its own estimation of relevancy judgments — which in most cases would not be a safe thing to do.

For Precision and Recall Calculations with LSA: To experiment with precision and recall calculations for LSA retrieval, run the script:

```
calculate_precision_and_recall_for_LSA.pl
```

Note that this script will carry out its own estimation of relevancy judgments — which in most cases would not be a safe thing to do.

For P-R Calculations for VSM with Human-Supplied Relevancies: Precision and recall calculations for retrieval accuracy determination are best carried out with human-supplied judgments of relevancies of the documents to queries. If such judgments are available, run the script:

`calculate_precision_and_recall_from_file_based_relevancies_for_VSM.pl`

This script will print out the average precision for the different test queries and calculate the MAP metric of retrieval accuracy.

For P-R Calculations for LSA with Human-Supplied Relevancies: If human-supplied relevancy judgments are available and you wish to experiment with precision and recall calculations for LSA-based retrieval, run the script:

`calculate_precision_and_recall_from_file_based_relevancies_for_LSA.pl`

This script will print out the average precision for the different test queries and calculate the MAP metric of retrieval accuracy.

To carry out significance tests on retrieval precision: If you wish to compare two retrieval algorithms (say, VSM or LSA with different values for some of the constructor parameters), try either

```
significance_testing.pl randomization
```

or

```
significance_testing.pl t-test
```

Significance testing consists of forming a null hypothesis that the two retrieval algorithms you are comparing are the same from a black-box perspective and then calculating what is known as the **p-value** that goes with the observed precision difference between the two algorithms. If the **p-value** is less than, say, 0.05, you reject the null hypothesis.

You can use either Randomization-based protocol or the Student-t Paired Distribution based protocol for a significance test. The choice between the two is supplied as a command-line argument to the script, as shown earlier.

To calculate a similarity matrix for a set of documents:

If you wish to calculate a similarity matrix for a given set of documents, run:

```
calculate_similarity_matrix_for_all_docs.pl
```

and

```
calculate_similarity_matrix_for_all_normalized_docs.pl
```

Back to TOC

8: Comparing Retrieval Algorithms

- Let's say we are given two different retrieval algorithms, A and B, whose performance is characterized by their respective MAP values. Obviously, if the difference between the MAP values is large, we may assume that one is superior and leave it at that.

- Recall that the largest possible value for the area under the Precision-Recall curve is 1. So, when we say that the difference between the MAP values is large, we mean it is a significant fraction of 1.

- But what if the difference between the MAP values is a small fraction of 1?

- By running a statistical significance test on the results produced by the two algorithms, we can tell which of the two algorithms is superior. [The basic question we want answered is whether the performance of one algorithm is within the "noise range" of the performance of the other algorithm. But how do you do that? Let's say you have a very large number N of queries at your disposal for testing the two algorithms. Now, instead of subjecting each algorithm to all of the N queries all in one go, why not subject the algorithms to randomly chosen subsets of the N queries? Assuming that these subsets are large enough so that you can "trust" the performance numbers yielded by them, now you can create an estimate of the "noise"

associated with the performance of each algorithm. Significance testing gives us statistically sound methods for doing these sorts of calculations.]

- To illustrate further with an example, let's consider two different singular-value acceptance thresholds for retrieval by the LSA algorithm in the `Algorithm::VSM` module.

- Recall that a key step of the LSA algorithm is to carry out a singular value decomposition (SVD) of the term-frequency matrix whose size is $V \times M$, where V is the size of the vocabulary and M the number of documents in the corpus. SVD factorization amounts to expressing the $V \times M$ term-frequency matrix A as $U\Sigma V^T = A$.

- In the SVD factorization of the $V \times M$ matrix A into $U\Sigma V^T$: (1) the columns of the $V \times M$ matrix U are orthogonal (that is, $U^T U = I$); (2) the $M \times M$ matrix Σ is a diagonal matrix of the singular values; and (3) the $M \times M$ matrix V is square orthogonal (that is, $V^{-1} = V^T$). Also note that the columns of U are the eigenvectors of AA^T and the columns of V the eigenvectors of $A^T A$.

- Dimensionality reduction in LSA is accomplished by ignoring the smallest of the singular values in the diagonal matrix Σ. This is accomplished in the LSA code in the module `Algorithm::VSM` by using the threshold `lsa_svd_threshold`.

- In `Algoirthm::VSM`, all singular values smaller than the `lsa_svd_threshold` fraction of the largest singular value are ignored when constructing an LSA model of a corpus.

- When dealing with corpora containing tens of thousands of documents, the retrieval speed can be improved significantly by making the threshold `lsa_svd_threshold` as large as possible. However, one has to keep in the mind the fact that if this threshold is set to too large a value, there could be a significant loss in retrieval precision.

- The good news is that, in general, the retrieval precision does NOT decrease monotonically as the value of the threshold `lsa_svd_threshold` is increased.

- So, as you are playing with different values of `lsa_svd_threshold`, and you find two (or more) that show closely related values for Average Precision for your test set of queries, you are left facing the question as to which version of the LSA algorithm is really better.

- This question of ascertaining the best of a set of algorithms with seemingly closely related values for retrieval precision becomes all the more important in our context because of inherent noise in the calculation of `Precision@rank` and `Recall@rank` values. The

main source of this noise is the human-to-human variability in the relevancy judgments.

- Shown in Table 7 are the `Average Precision` values for a set of queries that you will find in the `test_queries.txt` file in the 'examples' sub-directory of the main installation directory for the `Algorithm::VSM` module. The second column shows the `Average Precision` values for retrieval by LSA with `lsa_svd_threshold` set to 0.02 and the third column with the same threshold set to 0.05.

- Note that the two versions of the LSA algorithm used for producing the results shown in Table 7 possess what seems like comparable overall average precision for the queries. As shown in the fourth column, for some of the queries (those with '+' in the fourth column), the second version of LSA works better, but for others (those with '-' in the fourth column), the first works better. There are also queries (with '=' in the fourth column) for which the two algorithms yield the same retrieval precision.

- Given this query-to-query variation, and in light of the relevancy-related noise associated with the calculation of average precision, how can we say with confidence that one version of the retrieval algorithm is better than the other? The answer to this question is supplied by Significance Testing from statistics.

- A final note about the results shown in Table 7: These `Average Precision` results were obtained with the queries in the file `test_queries.text` and the relevancy judgments in the file `relevancy.txt` of the 'examples' directory of the `Algorithm::VSM` module. **IMPORTANT:** When you first install that module, please make a copy of `relevancy.txt` file for safe storage if you want to reproduce the results shown in Table 7. That file is overwritten each time you execute the script `calculate_precision_and_recall_for_LSA.pl` or the script `calculate_precision_and_recall_for_VSM.pl`. These two scripts are meant to be used when human-supplied relevancy judgments are not available and the module has to do the best it can in order to estimate the relevancies.

Query	Average Precision with `lsa_svd_threshold` = 0.02	Average Precision with `lsa_svd_threshold` = 0.05	Change in Retrieval Precision
q1	0.686	0.835	+
q2	0.931	0.931	=
q3	0.511	0.586	+
q4	0.161	0.293	+
q5	0.091	0.159	+
q6	0.045	0.045	=
q7	0.222	0.142	−
q8	0.100	0.100	=
q9	0.020	0.019	−
q10	0.093	0.192	+
q11	0.147	0.093	−
q12	0.015	0.015	=
q13	0.030	0.278	+
q15	0.212	0.210	−
q16	0.273	0.377	+
q18	0.119	0.227	+
q19	0.213	0.275	+
q20	0.504	0.461	−

Table 7: Average Precision values for 20 queries in the `test_queries.txt` file.

Back to TOC

9: The Three Ingredients of Significance Testing

- Let's refer to the LSA algorithm with the singular-value rejection-threshold, denoted by `lsa_svd_threshold`, set to 0.02 as LSA-1 and with same threshold set to 0.05 as LSA-2. Our question now is whether one can be considered to be superior to the other or whether, given a black-box perspective on the two algorithms, the two are the same.

- The value of MAP for LSA-1 is 0.244 and that for LSA-2 is 0.295. The magnitude of the difference between the two is 0.051. The question we want to answer with significance testing is whether a precision difference of 0.051 is statistically significant?

- The goal of this section is to show how significance testing can be used to answer the above question.

- A significance test involves the following three steps:

  - **(1) Specifying the Null Hypothesis:**
    A typical null hypothesis in our context would be that the two algorithms LSA-1 and LSA-2 are the same. The word "same" here means that, when considered as black boxes, the two algorithms appear to behave the same way vis-a-vis the outside world.

Another way to say the same thing would be that, if the null hypothesis were to be true, the two algorithm would produce performance numbers that are identically distributed *with regard to their statistical properties*.

The null hypothesis will involve something that is known as a *test statistic* whose values can be used to distinguish between the case when the null hypothesis is true and when it is false.

- **(2) Specifying the Test Statistic:** We will use the difference between the MAP values for the two algorithms as our test statistic. Obviously, if the null hypothesis were to be true, the difference between the MAP values produced by the two algorithms would "ordinarily" cluster around 0. As to why only "ordinarily", see Section 13. [You see, we are interested in the different values of the test statistic for different randomizations of what it took to produce the observed value that is being subject to significance testing.]

- **(3) Calculating the p-Value and Comparing It with the Significance Level:** The p-value (short for *probability-value*) associated with a particular *observed* value of the test statistic is the cumulative probability of the test-statistic taking on values in the tails of a two-sided distribution, the tails being specified by the observed value. [If the observed value for the test-statistic is θ, we are interested in the probability mass associated with the test-statistic taking on values in the intervals $(-\infty, \theta)$ and (θ, ∞).] The **significance-level** is a user-defined threshold. If the p-value is less than the significance level, we consider the null hypothesis to be false.

- In the next section, we talk about how to go about calculating the p-value for observed values of the test statistic.

- As mentioned at the beginning of this section, we are interested in the retrieval accuracy difference of 0.051. So we want to calculate the p-value associated with the difference of the two observed MAP values being 0.051.

- The smaller the p-value, the less tenable the null hypothesis.

- A common practice is to set the **significance level** at 0.05. That would imply rejecting the null-hypothesis if the p-value is less than 0.05.

- If the LSA-1 and LSA-2 were to be nearly identical (which we could do by making the `lsa_svd_threshold` parameter settings in the two algorithms to be nearly the same), the p-value associated with a precision difference like 0.051 could be significantly greater than the commonly-used significance level of 0.05.

Back to TOC

10: Calculating p-Values for the Randomization Based Test of the Null Hypothesis

- For the purpose of this test, we put to use the null hypothesis and momentarily forget about which of the two algorithms, LSA-1 or LSA-2, produced the `Average Precision` value in the second column and which produced the value in the third column in each row of the table shown in Table 7.

- So, as we examine each row of Table 7, we randomly assign LSA-1 or LSA-2 to the values in the second column. Obviously, when a row-entry in the second column gets, say, LSA-1, the corresponding entry in the third column will get LSA-2, and vice versa.

- An example of this random assignment of LSA-1 and LSA-2 labels to the data shown Table 7 is presented in Table 8. For the purpose of the test here, we consider the red values to have been produced by LSA-1 and the blue values to have been produced by LSA-2 under the null hypothesis that states that the algorithms LSA-1 and LSA-2 are the same.

- If N is the total number of queries, we will have 2^N ways of assigning the LSA-1 and LSA-2 labels to the values in the second and the third rows of Table 8. In other words, we have 2^N ways of generating the red-blue assignments shown in Table 8. We can also say that we have 2^N permutations of the LSA-1 and LSA-2 labels over the rows in Table 8.

- We next calculate the MAP value for each permutation for the assigned LSA-1 and LSA-2 labels, MAP merely being the mean of the `Average Precision` values over all the queries in our set of test queries.

- The MAP for LSA-1 (the red entries on Table 8) is 0.272 and that for LSA-2 (the blue entries in the same table) is 0.261. Therefore, the value of the test statistic for the LSA-1 and LSA-2 label assignments shown in Table 8 is 0.011.

- In Randomization test, we randomly sample the space of 2^N possibilities for the permutations of the LSA-1 and LSA-2 labels to the average precision data, where N is the number of queries. And for each permutation we calculate the value of the test statistic.

- Let's say we carry out these calculations for P number of permutations. We now count the number of time the test statistic for a permutation is less than the negative of the observed value

q1	0.686	0.835
q2	0.931	0.931
q3	0.511	0.586
q4	0.161	0.293
q5	0.091	0.159
q6	0.045	0.045
q7	0.222	0.142
q8	0.100	0.100
q9	0.020	0.019
q10	0.093	0.192
q11	0.147	0.093
q12	0.015	0.015
q13	0.030	0.278
q15	0.212	0.210
q16	0.273	0.377
q18	0.119	0.227
q19	0.213	0.275
q20	0.504	0.461

Table 8: red entry: Avg. Precision value assumed to be produced by LSA-1. blue entry: Avg. Precision value assumed to be produced by LSA-2.

that is being subject to significance testing or greater than the positive of this value. This is illustrated by the following Perl code snippet.

```
my $OBSERVED_VALUE = $MAP_Algo_1 - $MAP_Algo_2;

foreach (@test_statistic) {
    $count++ if $_ <= -1 * abs($OBSERVED_VALUE);
    $count++ if $_ > abs($OBSERVED_VALUE);
}
my $p_value = $count / @test_statistic;
```

where `$MAP_Algo_1` and `$MAP_Algo_2` are the MAP values that characterize the retrieval performance for a given set of test queries for the algorithms LSA-1 and LSA-2. It is the value of the variable `$OBSERVED_VALUE` that is subject to significance testing.

- As mentioned earlier, a common practice is that if the p-value thus calculated is less than the prescribed significance level of 0.05, we reject the null hypothesis. Obviously, rejection of the null hypothesis means that the two algorithms LSA-1 and LSA-2 are NOT the same.

- Shown below is the code for significance testing with both randomization and Student's t-test. A brief review of the code follows the code itself. This script is taken from the 'examples' directory of the module `Algorithm::VSM`, Version 1.1 or higher.

```
#!/usr/bin/perl -w

#
### significance_testing.pl
#
```

```
### This script is in the 'examples' directory of Version
### 1.1 and higher of the Algorithm::VSM module available at CPAN.
#
use strict;
use Algorithm::VSM;

my $debug_signi = 0;
#
#
die "Only one command-line arg allowed, which must either be " .
    "'randomization' or 't-test'\n" unless @ARGV == 1;
my $significance_testing_method = shift @ARGV;
die "The command-line argument must be either " .
    "'randomization' or " .  "'t-test' for " .
    "this module to be  useful."
       if ($significance_testing_method ne 'randomization') and
          ($significance_testing_method ne 't-test');

print "Proceeding with significance testing based on
                      $significance_testing_method\n";

my $MAX_ITERATIONS = 100000;
my $THRESHOLD_1    = 0.02;              # for LSA-1
my $THRESHOLD_2    = 0.05;              # for LSA-2

my $corpus_dir = "corpus";

my $query_file      = "test_queries.txt";
my $stop_words_file = "stop_words.txt";
my $corpus_vocab_db = "corpus_vocab_db";
my $doc_vectors_db  = "doc_vectors_db";
my $lsa_doc_vectors_db = "lsa_doc_vectors_db";
my $relevancy_file   = "relevancy.txt";

#   Significance testing is applied to the output of two
#   retrieval algorithms.  We want to know if the difference
#   between the MAP values for the two algorithms are
#   statistically significant.  Our example here is based on
#   LSA retrieval algorithms with different values for the
#   singular value acceptance threshold parameter lsa_svd_threshold.
#   For the null hypothesis, we will assume that the two algorithms
#   are the same. Our test statistic will be the difference between
#   the MAP values for the two algorithms.
#
#################  Algorithm LSA-1  ######################
#
my $lsa1 = Algorithm::VSM->new(
                  corpus_directory    => $corpus_dir,
                  corpus_vocab_db     => $corpus_vocab_db,
                  doc_vectors_db      => $doc_vectors_db,
                  lsa_doc_vectors_db  => $lsa_doc_vectors_db,
                  stop_words_file     => $stop_words_file,
                  query_file          => $query_file,
                  want_stemming       => 1,
                  lsa_svd_threshold   => $THRESHOLD_1,
```

```
                    relevancy_file      => $relevancy_file,
          );

$lsa1->get_corpus_vocabulary_and_word_counts();
$lsa1->generate_document_vectors();
$lsa1->construct_lsa_model();
$lsa1->upload_document_relevancies_from_file();
$lsa1->precision_and_recall_calculator('lsa');

my $avg_precisions_1 =
     $lsa1->get_query_sorted_average_precision_for_queries();
my $MAP_Algo_1 = 0;
map {$MAP_Algo_1 += $_} @$avg_precisions_1;
$MAP_Algo_1 /= @$avg_precisions_1;
print "MAP value for LSA-1: $MAP_Algo_1\n";
print "Avg precisions for LSA-1: @$avg_precisions_1\n"
                                if $debug_signi;
#
#
##################  Algorithm LSA-2  ######################
#
#
my $lsa2 = Algorithm::VSM->new(
                    corpus_directory    => $corpus_dir,
                    corpus_vocab_db     => $corpus_vocab_db,
                    doc_vectors_db      => $doc_vectors_db,
                    lsa_doc_vectors_db  => $lsa_doc_vectors_db,
                    stop_words_file     => $stop_words_file,
                    query_file          => $query_file,
                    want_stemming       => 1,
                    lsa_svd_threshold   => $THRESHOLD_2,
                    relevancy_file      => $relevancy_file,
          );

$lsa2->get_corpus_vocabulary_and_word_counts();
$lsa2->generate_document_vectors();
$lsa2->construct_lsa_model();
$lsa2->upload_document_relevancies_from_file();
$lsa2->precision_and_recall_calculator('lsa');

my $avg_precisions_2 =
      $lsa2->get_query_sorted_average_precision_for_queries();

my $MAP_Algo_2 = 0;
map {$MAP_Algo_2 += $_} @$avg_precisions_2;
$MAP_Algo_2 /= @$avg_precisions_2;
print "MAP value for LSA-2: $MAP_Algo_2\n";
print "Average precisions for LSA-2: @$avg_precisions_2\n"
                                        if $debug_signi;

#  This is the observed value for the test statistic that
#  will be subject to significance testing:
my $OBSERVED_t = $MAP_Algo_1 - $MAP_Algo_2;

print "\n\nMAP Difference that will be Subject to
```

```
                        Significance Testing: $OBSERVED_t\n\n";

#
#
#
##################   Significance Testing  ###################
#
#
my @range = 0..@$avg_precisions_1-1;

if ($debug_signi) {
    my $total_number_of_permutations = 2 ** @range;
    print "\n\nTotal num of permuts
                $total_number_of_permutations\n\n";
}
#
#   For each permutation of the algorithm labels over the
#   queries, we will store the test_statistic in the array
#   @test_statistic.
my @test_statistic = ();
#
#  At each iteration, we create a random permutation of the
#  LSA-1 and LSA-2 labels over the queries as explained on
#  pages 48 through and 53 of my tutorial on Significance Testing.
#  For each assignment of average precision values to
#  LSA_1, we calculate the MAP value for LSA-1, and the
#  same for LSA-2.  The difference between the two MAP
#  values is the value of the test_statistic for that
#  iteration.  Our goal is create test_statistic values for,
#  say, 100,000 iterations of this calculation.  In the
#  itself, LSA-1 is represented by algo_1 and LSA-2 by algo_2.

my $iter = 0;
while (1) {
    #  Here is the logic we use for permuting the algo_1 and
    #  algo_2 labels over the average precision values.  We
    #  first create a random permutation of the integers
    #  between 0 and the size of the query set.  We refer to
    #  this permuted list as permuted_range in what follows.
    #  We then walk through the elements of the list
    #  permuted_range and at each position test when the
    #  value at that position is less than or greater than
    #  half the size of the number of queries.  This
    #  determines which of the two avg. precision values for
    #  a given query gets algo_1 label and which gets the
    #  algo_2 label.
    my @permuted_range = 0..@range-1;
    fisher_yates_shuffle( \@permuted_range );
    my @algo_1 = ();
    my @algo_2 = ();
    foreach (0..@range-1) {
        if ($permuted_range[$_] < @range / 2.0) {
            push @algo_1, $avg_precisions_1->[$_];
            push @algo_2, $avg_precisions_2->[$_];
        } else {
```

```
            push @algo_1, $avg_precisions_2->[$_];
            push @algo_2, $avg_precisions_1->[$_];
        }
    }
    my $MAP_1 = 0;
    my $MAP_2 = 0;
    if ($debug_signi) {
        print "\n\nalgo_1 and algo_2 average precisions:\n\n";
        print "\npretend produced by algo 1: @algo_1\n\n";
        print "pretend produced by algo 2: @algo_2\n";
    }
    map {$MAP_1 += $_} @algo_1;
    map {$MAP_2 += $_} @algo_2;
    $MAP_1 /= @range;
    $MAP_2 /= @range;

    if ($debug_signi) {
        print "\nMAP_1: $MAP_1\n";
        print "MAP_2: $MAP_2\n\n";
    }
    $test_statistic[$iter] = $MAP_1 - $MAP_2;
    last if $iter++ == $MAX_ITERATIONS;
    print "." if $iter % 100 == 0;
}

if ($significance_testing_method eq 'randomization') {
    print "test-statistic values for different permutations:
                   @test_statistic\n" if $debug_signi;
    #  This count keeps track of how many of the
    #  test_statistic values are less than and greater than
    #  the value in $OBSERVED_t
    my $count = 0;
    foreach (@test_statistic) {
        $count++ if $_ <= -1 * abs($OBSERVED_t);
        $count++ if $_ > abs($OBSERVED_t);
    }
    my $p_value = $count / @test_statistic;

    print "\n\n\nTesting the significance of the test
                          statistic: $OBSERVED_t\n\n";

    print "\n\np_value for THRESHOLD_1 = $THRESHOLD_1 and
                THRESHOLD_2 = $THRESHOLD_2:   $p_value\n\n";

} elsif ($significance_testing_method eq 't-test') {
    my $mean = 0;
    my $variance = 0;
    my $previous_mean = 0;
    my $index = 0;
    map {    $index++;
             $previous_mean = $mean;
             $mean += ($_-$mean)/$index;
             $variance = $variance*($index-1)+($_-$mean) *
                                        ($_-$previous_mean);
             $variance /= $index;
```

```
        } @test_statistic;
    print "\n\nMAP Difference that will be Subject to \
                        Significance Testing: $OBSERVED_t\n\n";

    my $normalized_bound;
    my $p_value;
    if ($variance > 0.0000001) {
        $normalized_bound = ($OBSERVED_t - $mean) / sqrt($variance);
        print "Normalized bound: $normalized_bound\n\n";
        $p_value = 2*(1-cumulative_distribution_function(abs(
                                              $normalized_bound)));
    } else { $p_value = 1.0; }
    print "\n\n\nTesting the significance of the \
                             test statistic: $OBSERVED_t\n\n";
    print "\n\np_value for THRESHOLD_1 = $THRESHOLD_1 \
                  and THRESHOLD_2 = $THRESHOLD_2:   $p_value\n\n";
}
#
####################  Utility Functions ###################
#
# from perl docs:
sub fisher_yates_shuffle {
    my $arr =  shift;
    my $i = @$arr;
    while (--$i) {
        my $j = int rand( $i + 1 );
        @$arr[$i, $j] = @$arr[$j, $i];
    }
}
#
#  Abramowitz and Stegun's high-quality approximation to the
#  normal CDF:
#
sub cumulative_distribution_function {
    my $x = shift;
    my $PI = 3.14159265358;
    my $normalized_pdf_value = exp(-($x**2)/2.0) / sqrt(2*$PI);
    my $t = 1 / (1 + 0.2316419 * $x);
    my $cdf = 1 - $normalized_pdf_value * (  0.319381530*$t
                                       - 0.356563782*($t**2)
                                       + 1.781477937*($t**3)
                                       - 1.821255978*($t**4)
                                       + 1.330274429*($t**5));
    return $cdf;
}
```

- To explain the organization of the code, shown in the upper half of page 52 are the values for the parameters of the significance testing demonstration here. The variable `$MAX_ITERATIONS` is

the number of permutations from the 2^N possibilities for the assignment of LSA-1 and LSA-2 labels to the rows in Table 8 where N is the number of queries in the `test_queries.txt` file. We have set `$MAX_ITERATIONS` to 100,000. The parameters `$THRESHOLD_1` and `$THRESHOLD_2` are the two values for the constructor parameter `lsa_svd_threshold` that is used to decide how many of the largest singular values to accept for the LSA model.

- The rest of what is in the upper half of page 52 is setting up values for the constructor parameters for the module class `Algorithm::VSM`. See the documentation that comes with the module for an explanation of the constructor parameters.

- What you see in the bottom third of page 52 is a call to the constructor of the `Algorithm ::VSM` class for the LSA-1 model. The corresponding constructor call for LSA-2 is shown in the middle third of page 53. What you see at the top and the bottom of page 53 are the method invocations on the instances of `Algorith::VSM` for actual construction of the corpus models. These method invocations extract the corpus vocabulary, subject the vocabulary to stop words and stemming, construct document vectors, etc.

- Note that the only difference between the constructor calls on pages 72 and 73 is the value for the parameter

`lsa_svd_threshold`. For LSA-1, we set it to `$THRESHOLD_1` and for LSA-2 to `$THRESHOLD_2`.

- At the bottom of page 53, you see the script calculating a value for `$OBSERVED_t`. This is the observed value of the test statistic. It is this value we want to subject to significance testing. What is stored in `$OBSERVED_t` is the difference of the two MAP values, the first being the MAP for LSA-1 and the second being the MAP for LSA-2. The individual MAP values are stored in the variables `$MAP_Algo_1` and `$MAP _Algo_2`.

- The actual significance testing code begins in the middle of page 54. We first create an empty array `@test_statistic`. This array will hold the test statistic values for the different permutations of the assignments of the labels LSA-1 and LSA-2 to the rows of Table 8.

- For generating each permutation, at the bottom of page 54 we first apply the Fisher-Yates shuffle algorithm to create a random permutation of the sequence of integers from 1 though $N - 1$ where N is the number of queries for which we have collected the average precision data. The shuffled sequence of the integers is stored in the array variable `@permuted_range`. Subsequently, we scan through the entries in this array and when an entry exceeds the mean of the integer sequence, we retain the LSA-1/LSA-2 labels for the average precision values for that query. Otherwise,

we switch the labels.

- In the upper half of page 55, we calculate the MAP values for the sequence of precision values that carry the label LSA-1 and the sequence with the label LSA-2.

- Note how the following statement in the middle of page 55 calculates the value of the test statistic by taking a difference of the two MAP values for LSA-1 and LSA-2 for a given permutation of the LSA-1/LSA-2 labels:

```
$test_statistic[$iter]  =   $MAP_1 - $MAP_2;
```

- Having calculated the test-statistic values for each of the permutations, starting in the middle of page 55, I first show the code for p-value calculation with the Randomization method and then show the code for the same but with the Student's Paired t-Test.

- For the method based on randomization, as explained earlier Section 10, we count the number of times the test-statistic value is either less than the negative of the absolute value stored in `$OBSERVED_t` or larger the positive of the absolute value stored in the same variable. When we divide it by the total number of permutations used, we get the p-value.

- What is shown in the lower third of page 55 and the upper half of page 56 is the calculation of the p-value using the Student's Paired t-Test. We will explain this in greater detail in the next section.

- The bottom half of page 56 contains two utility functions, one for the Fisher-Yates random shuffle of an array and the other for calculating a high-quality approximation to the normal cumulative distribution function (CDF) that is needed for the Student's Paired-t Test approach to calculating the p-value.

Back to TOC

11: Calculating the p-Values with the Student's Paired t-Test

- The main difference between the Randomization method and the Student's Paired t-Test method for p-value calculation is as follows: Whereas in the former we make no assumptions regarding the distribution of the test statistic, for the latter we assume the values of the test statistics are distributed according to the Student's t-distribution.

- A t-distributed random variable looks like a normally distributed random variable except for the fact that the former has heavier tails. That makes a t-distributed random variable more likely to capture phenomena where rare events are not as unlikely as for the case of normally distributed random variables.

- Perhaps the easiest way to look at a t-distributed variable is as the ratio of a normally distributed variable and a chi-square distributed variable with ν degrees of freedom. [A chi-square variable of ν degrees of freedom is the square-root of the squares of $\nu + 1$ normally distributed variables.]

- The test statistic for the Student-t test is given by the ratio of the

difference between the MAP values (this is the value of the variable `$OBSERVED_t`) and what is known as the "standard error" of this difference. In our context, the standard error is approximated by $\sqrt{\frac{\sigma^2}{N}}$, where N is the number of queries in the `test_queries.txt` file and σ^2 the variance associated with the MAP values under the null hypothesis. After you have calculated the test statistic, you can find the p-value by looking up a table of values from Student's t-distribution.

- For this tutorial we will take advantage of the fact that as the degrees of freedom increase, the t-distribution approaches the normal distribution.

- For example, when the degrees of freedom are 30 or more, the error in approximating the PDF (probability density function) or the CDF (cumulative distribution function) of a t-distribution by that for a normal distribution is less than 0.005.

- Under the assumption of normal distribution for the test statistic, the p-value can be calculated by finding the total probability mass of the normal distribution that is below the absolute observed value (stored in `$OBSERVED_t`) and that is above this value.

- However, before we can find the above-mentioned probability mass, we must estimate the mean and the variance of the normal

distribution that describes the test statistic.

- The lower half of the code on page 55 shows us using an incremental algorithm for estimating both the mean the variance.

- Having estimated the mean and the variance for the test statistic, we now calculate on 56 the probability mass we mentioned above. This is the p-value as yielded by the Student's Paired t-Test based method.

Back to TOC

12: Comparison of the p-Values Obtained with Randomization and with Student's t-Test

- Shown in Table 9 are the p-values calculated with Randomization and with Student-t. The corpus, the queries, and the relevance judgments used in the results shown here can be found in the `examples` directory of the `Algorithm::VSM` module, version 1.1 or higher. As mentioned earlier, the Student-t results are based on a normal approximation to the t-distribution. This is likely to be a questionable assumption for the experimental parameters used.

`lsa_svd_threshold` for LSA-1	`lsa_svd_threshold` for LSA-2	p-value with Randomization	p-value with Student-t
0.02	0.12	0.00002	0.00044
0.02	0.10	0.00173	0.00438
0.02	0.08	0.00370	0.00154
0.02	0.06	0.00897	0.01710
0.02	0.04	0.00865	0.01984
0.02	0.03	0.07545	0.09548
0.02	0.025	1.000	1.000

Table 9: A comparison of the p-values calculated using Randomization and the Student's t-Test.

- We will use the criterion that a p-value of less than 0.05 means that we can reject the null hypothesis (the null hypothesis says that LSA-1 and LSA-2 can be thought of as being the same algorithm). **So for $p < 0.05$, we consider the two algorithms to be different.**

- With the randomization test we must reject the null hypothesis (implying that the two algorithms are different) when we set LSA-2's `lsa_svd_threshold` to 0.04 or higher. On the other hand, we must accept the null hypothesis (implying that LSA-1 and LSA-2 can be considered to be the same algorithm) when LSA-2's `lsa_svd_threshold` is set to 0.03 or lower. These conclusions apply when we keep LSA-1's `lsa_svd_threshold` set at 0.02.

- The conclusions we draw for the t-test are the same.

- One would think that as we increase the value of `lsa_svd_threshold` for LSA-2 in relation to the value of the same parameter in LSA-1, the p-values would become monotonically smaller since the two retrieval algorithms would become more and more different. That this is not borne out by the results shown here might be attributable to the small number of queries used in the experiments and to the small size of the corpus.

- Note that in all cases, we leave unchanged the value of `lsa_svd_threshold` for LSA-1 for the comparative study.

Back to TOC

13: A Potential Source of Confusion Regarding the Calculation of p-Values

- To explain the source of confusion, let's go back to the loop shown in the first bullet on page 51 where we count the number of times the value of the test statistic is *outside* the interval

$$[\ -abs(\$OBSERVED\_VALUE),\quad abs(\$OBSERVED\_VALUE)\ ]$$

- The confusion arises from the fact that we *accept* the null hypothesis (that is, we consider the two algorithms being compared to be identical) when a large number of the test statistic values fall outside the above range. But, it would seem that when two algorithms being compared are identical, the value of the test statistic would be zero and therefore, seemingly, *within* the interval shown above.

- So what gives?

- Before I answer this question, consider first the case of what the implementation shown earlier does when the two algorithms are identical.

- The case of two algorithms being completely identical is obviously a boundary case.

- As it turns out, the code shown earlier for the calculation of the p-values handles this case correctly. In this case, the variable `$count` in the loop on page 51 is incremented for every iteration through the loop. Note that for `$count` to not get incremented, the value of `$test_statistic` must be strictly within the half-closed interval shown above. But when both the `$OBSERVED_VALUE` and the value for a test statistic are zero, that condition would NOT be satisfied.

- Let's now relax the boundary case described above a bit and consider the case when the two algorithms being compared are nearly identical but not exactly so.

- Now the value of `$OBSERVED_VALUE` will be small but not zero. Let's say that now we want to test the null hypothesis that the two algorithms can be considered to be identical from a black-box perspective. If the two algorithms are nearly identical, then the rationale underlying significance testing is that whereas the two real MAP values for the two algorithms would be close, a random assignment of algorithms to the entries in a table such as the one shown on page 51 would yield a MAP difference that would not be small in relation to the value of `$OBSERVED_VALUE`.

- By the same token, if the two algorithms being compared are significantly different, then the value of `$OBSERVED_VALUE` will be relatively large. But now a random assignment of algorithms to the entries in the table on page 51 would generally produce a MAP difference that could be comparable to the value in `$OBSERVED_VALUE`.

- Another way of saying the same thing would be that when the two algorithms being compared are really close, their MAP difference (which is the value stored in `$OBSERVED_VALUE`) would be very small and that this difference would generally be smaller than what you would get from a random assignment of the algorithms to the retrieval results shown in the different rows of the table on page 51.

Back to TOC

14: Acknowledgments

If you enjoyed this tutorial, much of the credit should go to Mark Smucker, James Allan, and Ben Carterette. They are the authors of the wonderful paper "A Comparison of Statistical Significance Tests for Information Retrieval Evaluation" that was presented at CIKM'07. This paper was brought to my attention by Shivani Rao. Thanks Shivani!

My conversations with Chad Aeschliman, Gaurav Srivastava, and Josiah Yoder were of much help in clarifying the presentation of precision-recall characterization of random retrieval from a database.

During the spring of 2015, my conversations with Naveen Kulkarni of Infosys played an important role in the upgrades I made to my Perl based VSM module described in Sections 6 and 7 of this tutorial. Naveen has packaged the VSM module into a Windows-platform-based standalone executable that incorporates all imported modules.

With regard to the changes made in December 2015, I owe many thanks to Shayan Akbar and Tanmay Prakash for the conversations that resulted in several improvements to my presentation of the material related to Precision-vs.-Recall curves in Section 1 of this tutorial.

During the summer of 2023, Rohan Sarkar's interest in using Deep Metric Learning for image retrieval led me back to this tutorial. Rohan is currently working on his Ph.D in RVL. As a result, I decided to clean up the tutorial and make its formatting a bit more modern.