Introduction to Information Retrieval: IR Basics and Evaluation

Prof. Srijan Kumar
Logistics

• **Class size:** Due to huge demand, class size has been increased to 85

• **Piazza:** Please join
  – [https://piazza.com/class/spring2020/cse6240/](https://piazza.com/class/spring2020/cse6240/) (same link as before)

• **Canvas:** Logistical issues being resolved now

• **Project:**
  – Example datasets and sample projects will be released by Thursday evening
  – Teams due by Jan 20
Today’s Class

• Web is a collection of documents
  – E.g., web pages, social media posts

• Web is a network
  – E.g., the hyperlink network of websites, network of people on social networks

• Web is a set of applications
  – E.g., e-commerce platforms, content sharing, streaming services

Some slides from today’s lecture are inspired from Prof. Hongyuan Zha’s past offerings of this course.
Today’s Class: Part 1

- **Web is a collection of documents**
  1. Process documents for search and retrieval
  2. Quantifying the quality of retrieval
Search and Retrieval are Everywhere

• **Web search engines**: Querying for documents on the web
  – Google, Bing, Yahoo Search

• **E-commerce platforms**: Querying for products on the platform
  – Amazon, eBay

• **In-house enterprise**: Querying for documents internal to the enterprise
  – Universities, Companies
Processing Document Collections

- **Goal**: Index documents to be easily searchable
- **Steps to index documents**:
  1. **Collect** the documents to be indexed
  2. **Tokenize** the text
  3. **Normalize** of the text (linguistic processing)
  4. **Index** the text: Inverted Indexing
Processing Document Collections

Documents to be indexed.

Token stream.

Modified tokens.

Inverted index.

Tokenizer

Linguistic modules

Indexer

Tokenization and linguistic processing determine the terms considered for retrieval.
Processing Document Collections

Tokenization and linguistic processing determine the terms considered for retrieval.
Tokenization

• Tokenization formats the text by chopping it up into pieces, called **tokens**
  – E.g., remove punctuations and split on white spaces
  – Georgia-Tech → Georgia Tech

• However, **tokenization can give unwanted results**
  – San Francisco → “San” “Francisco”
  – Hewlett-Packard → Hewlett Packard
  – **Dates**: 01/08/2020 → 01 08 2020
  – **Phone number**: (800) 111-1111 → 800 111 1111
  – **Emails**: srijan@cs.stanford.edu → srijan cs stanford edu

• Such splits can result in poor retrieval results
Tokenization: What To Do?

• So, what should one do?
• Come up with **regular expression rules**
  – E.g., only split if the next word starts with a lowercase letter
• **Has to be language specific:** English rules not applicable to all other languages
  – E.g., French: *L’ensemble*
  – German: *Computerlinguistik* means ‘computational linguistics’
Tokenization and linguistic processing determine the terms considered for retrieval.
Text Normalization: Why is it Needed?

• The same text can be written in many ways
  – USA vs U.S.A. vs usa vs Usa

• We need some way to create a **unified representation** to match them

• The **same normalization** is required for the query and the documents
Text Normalization: Other Languages

• **Accents:** resume vs résumé

• **Most important criteria:** How are your users likely to write their queries?

• Even in languages where the accents are the norm, users often not type them, or the input device is not convenient

• **German:** Tuebingen vs. Tübingen
  – should be the same

• **Dates:** July 30 vs. 7/30
Text Normalization Step 1: Case Folding

• Reduce all letters to lower case
  – exception: upper case (in mid-sentence?)

• Often best to **lower case everything**, since users tend to use lowercase regardless of the correct capitalization

• **However**, many proper nouns are derived from common nouns
  – General Motors, Associated Press

• We can create **advanced solutions** (later): bigrams, n-grams
Text Normalization Step 2: Remove Stop Words

- With a stop-word list, one excludes from the dictionary the most common words
  - They have little semantic content: the, a, and, to
  - They take a lot of space: 30% of postings for top 30

- Fewer stop words:
  - Can use good compression techniques
  - Good query optimization techniques mean one pays little at query time for including stop words
Text Normalization Step 2: Remove Stop Words

• **However, stop words can be needed for:**
  – Phrase queries: "King of Prussia"
  – (Song) titles etc.: "Let it be", "To be or not to be"
  – Relational queries: "flights to London"
Text Normalization Step 3: Stemming

• **Key idea:** Derive the base form of words, i.e. root form, to standardize their use
  – Reduce terms to their “roots” before indexing

• **Variations of words do not add value for retrieval**
  – **Grammatical** variations: organize, organizes, organizing
  – **Derivational** variations: democracy, democratic, democratization

• **“Stemming” suggest crude suffix chopping**
  – Again, language dependent
  – E.g., organize, organizes, organizing → organiz
Text Normalization Step 3: Stemming

For example, compressed and compression are both accepted as equivalent to compress.

For example, compress and compress are both accept as equivalent to compress.
Porter’s Stemmer

• Most commonly used stemmer for English
  – Empirical evidence: as good as other stemmers

• Conventions + five phases of reductions
  – phases applied sequentially
  – each phase consists of a set of commands
  – sample convention: of the rules in a compound command, select the one that applies to the longest suffix
Porter’s Stemmer: Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSES → SS</td>
<td>caresses → caress</td>
</tr>
<tr>
<td>IES → I</td>
<td>ponies → poni</td>
</tr>
<tr>
<td>SS → SS</td>
<td>caress → caress</td>
</tr>
<tr>
<td>S →</td>
<td>cats → cat</td>
</tr>
</tbody>
</table>
Processing Document Collections

Tokenization and linguistic processing determine the terms considered for retrieval.
Scoring and Ranking Documents

- Ranked list of documents:
  - Order the documents most likely to be relevant to the searcher
  - It does not matter how large the retrieved set is

- How can we rank-order the docs in the collection with respect to a query?

- Begin with a perfect world – no spammers
  - Nobody stuffing keywords into a doc to make it match queries
Techniques For Indexing

1. Term-Document Incidence Matrix
2. Inverted Index
3. Positional Index
4. TF-IDF
### Technique 1: Term-Document Incidence Matrix

#### Documents

<table>
<thead>
<tr>
<th>Terms</th>
<th>Anthony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anthony</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Brutus</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Caesar</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Cleopatra</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>mercy</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>worser</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- For Boolean query "**Brutus AND Caesar AND NOT Calpurnia**"  
  - $110100 \text{ AND } 110111 \text{ AND } 101111 = 100100$

- **Not scalable**: Billions of terms and millions of documents
Technique 2: Inverted Index

- An inverted index consists of a dictionary and postings
- For each term T in the dictionary, we store a list of documents containing T

```
Brutus  →  1  2  4  11  31  45  173  174
Caesar  →  1  2  4  5  6  16  57  ...
Calpurnia →  2  31  54  101
```

Dictionary  Postings
### Building an Inverted Index I

#### Tokenize documents

<table>
<thead>
<tr>
<th>term</th>
<th>docID</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>did</td>
<td>1</td>
</tr>
<tr>
<td>enact</td>
<td>1</td>
</tr>
<tr>
<td>julius</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>was</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>i’</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>capitol</td>
<td>1</td>
</tr>
<tr>
<td>brutus</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>me</td>
<td>1</td>
</tr>
<tr>
<td>so</td>
<td>2</td>
</tr>
<tr>
<td>let</td>
<td>2</td>
</tr>
<tr>
<td>it</td>
<td>2</td>
</tr>
<tr>
<td>be</td>
<td>2</td>
</tr>
<tr>
<td>with</td>
<td>2</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
</tr>
<tr>
<td>noble</td>
<td>2</td>
</tr>
<tr>
<td>hath</td>
<td>2</td>
</tr>
<tr>
<td>told</td>
<td>2</td>
</tr>
<tr>
<td>you</td>
<td>2</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
</tr>
<tr>
<td>ambitious</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Sort alphabetically

<table>
<thead>
<tr>
<th>term</th>
<th>docID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ambitious</td>
<td>2</td>
</tr>
<tr>
<td>be</td>
<td>2</td>
</tr>
<tr>
<td>brutus</td>
<td>1</td>
</tr>
<tr>
<td>brutus</td>
<td>2</td>
</tr>
<tr>
<td>capitol</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>did</td>
<td>1</td>
</tr>
<tr>
<td>enact</td>
<td>1</td>
</tr>
<tr>
<td>hath</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>I’</td>
<td>1</td>
</tr>
<tr>
<td>it</td>
<td>2</td>
</tr>
<tr>
<td>julius</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>let</td>
<td>2</td>
</tr>
<tr>
<td>me</td>
<td>1</td>
</tr>
<tr>
<td>noble</td>
<td>2</td>
</tr>
<tr>
<td>so</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
</tr>
<tr>
<td>told</td>
<td>2</td>
</tr>
<tr>
<td>you</td>
<td>2</td>
</tr>
<tr>
<td>was</td>
<td>1</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
</tr>
<tr>
<td>with</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Compress using counts/term frequency

<table>
<thead>
<tr>
<th>term</th>
<th>docID</th>
<th>freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ambitious</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>be</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>brutus</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>brutus</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>capitol</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>did</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>enact</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>hath</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>I’</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>it</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>julius</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>let</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>me</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>noble</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>so</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>told</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>told</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>you</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>was</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>with</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Building an Inverted Index II

Compress by creating a list of documents that have the term
**Retrieval with Inverted Index**

- **Example query:** Brutus AND Calpurnia
- **Steps:**
  - Locate Brutus in the Dictionary
  - Retrieve its postings
  - Locate Calpurnia in the Dictionary
  - Retrieve its postings
  - Intersect the two postings lists

```
Brutus → 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174
Calpurnia → 2 → 31 → 54 → 101
Intersection → 2 → 31
```
Algorithm to Intersect/Merge Lists

• Postings in sorted order, complexity O(x + y)

INTERSECT(p, q)
1   answer ← ⟨⟩
2   while p ≠ NIL and q ≠ NIL
3     do if docID[p] = docID[q]
4       then ADD(answer, docID[p])
5           p ← next[p]
6           q ← next[q]
7     else if docID[p] < docID[q]
8       then p ← next[p]
9       else q ← next[q]
10   return answer
Introduction to Information Retrieval: IR Basics and Evaluation
Part 2
Prof. Srijan Kumar
Logistics

- **Piazza**: Still some students remaining. Please join.
  - [https://piazza.com/class/spring2020/cse6240/](https://piazza.com/class/spring2020/cse6240/)

- **Canvas**: Available now. Please join for submissions.

- **Project**: Example datasets and sample projects released
  - **Reminder**: Teams due next Monday

- **Hands-on ipython tutorial session**: Tuesday during office hours (3-4 PM, Klaus 3rd Floor Atrium, by the Elevator)

- **Homework**: Details in Wednesday’s class
Recap from Previous Class

• Web is a collection of documents
  1. Process documents for search and retrieval
  2. Quantifying the quality of retrieval
Processing Document Collections

Tokenization and linguistic processing determine the terms considered for retrieval.
Techniques For Indexing

1. Term-Document Incidence Matrix
2. Inverted Index
3. TF-IDF
Complex Full Text Queries

- Long queries pose a problem for the previous techniques
  - Not scalable as it will generate long Boolean queries
  - Very strict: all query terms should be present
    - In practice, query terms may be missing in a document

- Solution: Advanced processing with term weighting
  - If a document talks about a topic more, then it is a better match
  - A document is relevant if it has many occurrences of the term(s)
  - This leads to the idea of term weighting
Bag of Words Model

• **Represent a document as a collection of words** (after cleaning the document)
  – The order of words is irrelevant
  – The document “John is quicker than Mary” is indistinguishable from the doc “Mary is quicker than John”

• **Rank documents according to the overlap between query words and document words**
Term Frequency Vectors

- Consider **Term Frequency** $tf_{t,d} = \text{the number of occurrences of a term } t \text{ in a document } d$
  
  - A document is a vector (a column of a matrix)

<table>
<thead>
<tr>
<th></th>
<th>Antony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>157</td>
<td>73</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Brutus</td>
<td>4</td>
<td>157</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>232</td>
<td>227</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>57</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>worse</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Term Frequency Vectors

- The use of term frequency vectors poses some **problems**:
  - **Long docs are favored** because they are more likely to contain query terms
    - Possible fix: normalize by document length
  - **All words are treated as equal**
    - Which one tells you more about a document?
    - 10 occurrences of *Brutus* or 10 occurrences of *the*

- Would like to attenuate the weights of common terms
  - How to define common?

- **Solution: Document Frequency**
Document Frequency

- **Document Frequency** $df_t = \text{the number of documents in the corpus containing the term}

- **How to use document frequency?**

- **Inverse Document Frequency** $idf_t$
  - A measure of informativeness of a term: rarity across the whole corpus
  - **High $idf$** = the term is unique; **low $idf$** = common words
  - **Formulation 1:** the raw count of number of documents the term occurs in
    - $idf_t = \frac{1}{df_t}$
  - **Formulation 2:** logarithmically scaled inverse fraction
    - $idf_t = \log \left( \frac{N}{df_t} \right)$
    - Where N = number of documents in the corpus
Scoring a Query Against a Document

- **Scoring a term-document pair \((t, d)\)**
  - Tf-idf weight of term \(t\) in a document \(d\)
    \[
    tf - idf(t, d) = tf_{t,d} \times idf_t
    \]

- **Scoring a query-document pair \((q, d)\)**
  - Aggregate across all terms in the query
    \[
    Score(q, d) = \sum_{t \in q} tf - idf(t, d)
    \]
Scoring a Query Against a Document: BM25

- In BM25 or Okapi BM25:
  - **Term Frequency** = \( \frac{tf_t \cdot (K + 1)}{tf_t + K \cdot (1 - b + b \cdot \frac{|D|}{L})} \)
  - **Inverse Document Frequency** = \( \log\left(\frac{N - df_t + 0.5}{df_t + 0.5}\right) \)

where
- the parameters were set empirically: \( b = 0.75 \), \( K \) lies in \([1.2, 2.0]\)
- \( |D| = \) length of document
- \( L = \) average length of all documents in the corpus
- \( N = \) number of documents
Incorporating Web Page Structure

- **Web page structures are complex**
  - Title, Body, Tags, Metadata, Bold vs light

- **Position of terms in different parts has different importance**
  - Presence of a term in title > Presence of the same term in body

- **Solution: Weight positions differently**
  - E.g., 0.6*<term in title> + 0.1*<term in body> + 0.3*<term in tags>
  - Total weights sum to 1.0
Position Weights

• Where do the weights come from? **Machine Learning**
  – Given
    • A document corpus
    • A suite of queries
    • A set of relevance judgements
  – Learn a set of weights such that relevance judgments matched
  – Can be formulated as a regression problem
Today’s Class: Part 2

- **Web is a collection of documents**
  1. Process documents for search and retrieval
  2. Quantifying the quality of retrieval
Measures of a Search Engine

• How fast does it index
  – Number of documents/hour

• How fast does it search
  – Latency as a function of index size

• How frequent is the index refreshed

• Expressiveness of query language
  – Ability to express complex information needs
  – Speed on complex queries

• How satisfied are the users
  – Users will be satisfied if the results are accurate
  – Most tricky to quantify
User Satisfaction in Different Cases

- **Web search engines: Users find what they want and return to engine**
  - Measure rate of returning users, rate of click

- **E-commerce platforms: Users find what they want and make a purchase**
  - Measure time to purchase

- **In-house enterprise: Users find documents fast**
  - Quantify productivity, i.e., how much time do users save

- **In all of the above, the results have to be accurate**
Quantifying User Satisfaction

• **Commonest proxy: relevance of search results**
  – But how do you measure relevance?
• We will detail a methodology here, then examine its issues
• **Relevant measurement requires 3 elements:**
  – A benchmark document collection
  – A benchmark set of queries
  – A binary assessment of either Relevant or Irrelevant for each query-document pair
• In Web search, relevance is more-than-binary, i.e., multi-grade relevance
Evaluating an IR system

• The IR system should satisfy the user’s information need, which is translated into a query

• Relevance is assessed relative to the information need, not the query
  – E.g., information need: I’m looking for information on whether drinking red wine is more effective at reducing your risk of heart attacks than white wine.
  – Query: wine red white heart attack effective
  – You evaluate whether the doc addresses the information need, not whether it has those words

• However, broad-topic queries tend to represent multiple intentions

• For web search, we need detailed guidelines for relevance judgments
  – perfect, excellent, good, fair, bad
Standard Benchmarking

- **TREC - National Institute of Standards and Testing (NIST)**
  has run a large IR test bed for many years

- **Evaluation Setup:**
  - **Input:** a document corpus and a query
  - The IR system returns a subset of documents, in rank-order (most important to least important)
  - Human experts rate each returned document as relevant or irrelevant

- **Remember:** the ground-truth data is unbalanced
  - Most documents are irrelevant to the query
Evaluation Metrics

- Several practical evaluation metrics:
  1. Accuracy
  2. Precision
  3. Recall
  4. F-score
  5. Mean Average Precision
  6. Normalized Discounted Cumulative Gain (nDCG)
Metric 1: Accuracy

- **Accuracy** = Fraction of correct answers
  - \( \text{Accuracy} = \frac{(\text{Number of relevant retrieved documents} + \text{Number of irrelevant non-retrieved documents})}{\text{Number of all documents}} \)
  - \( \text{Accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)} \)

- **Not a useful metric for IR. Why?**
  - Most documents are irrelevant, so keeping TN high can make accuracy high

Don’t return anything, get ~99.99% accuracy!
Metrics 2 and 3: Precision and Recall

- **Precision** = fraction of retrieved docs that are relevant
  \[ P(\text{relevant} \mid \text{retrieved}) \]

- **Recall** = fraction of relevant docs that are retrieved
  \[ P(\text{retrieved} \mid \text{relevant}) \]

- Precision = \( \frac{TP}{TP + FP} \)
- Recall = \( \frac{TP}{TP + FN} \)

- Good IR systems should have high TP and TN, low FP and FN

<table>
<thead>
<tr>
<th></th>
<th>Relevant</th>
<th>Not Relevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieved</td>
<td>True Positive TP</td>
<td>False Positive FP</td>
</tr>
<tr>
<td>Not Retrieved</td>
<td>False Negative FN</td>
<td>True Negative TN</td>
</tr>
</tbody>
</table>
Precision-Recall Tradeoff

- You can increase recall by returning more docs
- Recall is a non-decreasing function of the number of docs retrieved
  - A system that returns all docs has 100% recall!
- The converse is also usually true: It’s easy to get high precision for very low recall
Metric 4: F-measure

• **F-measure = a combination of Precision and Recall**
  
  – Weighted Harmonic Mean

  \[
  F = \frac{1}{\frac{\alpha}{P} + \frac{1 - \alpha}{R}} = \frac{(\beta^2 + 1)PR}{\beta^2P + R}
  \]

  where \( \beta^2 = \frac{1 - \alpha}{\alpha} \)

  • When \( \beta = 1 \), \( F \) becomes a simple harmonic mean of \( P \) and \( R \); also called \( F_1 \)

  \[
  \frac{1}{F} = \frac{1}{2} \left( \frac{1}{P} + \frac{1}{R} \right)
  \]
F-measure: An Example

• Given a corpus of 100 relevant documents for a query, an IR system returns:

<table>
<thead>
<tr>
<th></th>
<th>relevant</th>
<th>not relevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>retrieved</td>
<td>18</td>
<td>2</td>
</tr>
<tr>
<td>not retrieved</td>
<td>82</td>
<td>1,000,000,000</td>
</tr>
</tbody>
</table>

• **Precision** = \( \frac{18}{18+2} = 0.9 \)
• **Recall** = \( \frac{18}{18+82} = 0.18 \)
• **F1** = \( \frac{2PR}{P+R} = \frac{2 \times 0.9 \times 0.18}{0.9 + 0.18} = 0.3 \)
• F1 is a lot lower than avg of P and R = \( \frac{0.9+0.18}{2} = 0.54 \)
• F1 does not factor true negatives (1B in the above case)
Evaluating Ranked Results

• **Search engine returns ranked list of documents**
  – Take first document, interpret as unordered set of size 1, compute unordered evaluation measures for this set.
  – Take top 2 documents, interpret as unordered set of size 2, compute unordered evaluation measures for this set, and so on.

• **Plot individual measures** → precision-recall curve
Given a ranked list of documents, mark each as relevant or irrelevant, in ranking order. Example:

- 1 – relevant. P = 1/1 = 1.0
- 2 – irrelevant. P = 1/2 = 0.5
- 3 – relevant. P = 2/3 = 0.66
- 4 – relevant. P = 3/4 = 0.75
- 5 – irrelevant. P = 3/5 = 0.6
Evaluation: Issues So Far

• Plots are good, but need quantification
  – **Precision at fixed retrieval level k**
    • Perhaps most appropriate for web search: all people want are good matches on the first one or two results pages

• A precision-recall graph for one query isn’t a very sensible thing to look at every time
  – You need to **average performance over a whole bunch of queries**
Mean Average Precision

• MAP = Average precision value for the top documents so far, each time a relevant document is retrieved

• For a query $q_j$ in set of queries $Q$, the set of relevant documents are $\{d_1, \ldots, d_m\}$
  – Macro-averaging: each query counts equally

• $R_k$ is list of ranked results until $d_k$

\[
MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{\frac{|Q|}{m}} \frac{1}{m} \sum_{k=1}^{m} \text{Precision}(R_k)
\]
Discounted Cumulative Gain (DCG)

- **DCG has a finite ordinal grade set**, e.g., {Perfect, Excellent, Good, Fair}
- **Each grade is associated with a gain value** $g_i = g(L_i)$
  - Perfect = 20, Excellent = 10, Good = 5, Fair = 1, Bad = 0
- **Each position has a discount (importance) factor**: $c_1 > c_2 > \ldots > c_k > 0$
- **DCG for a ranking list of documents** $\{d_1, \ldots, d_N\}$:

$$DCG_{g,K} = \sum_{j=1}^{K} c_j g(d_j), \quad K = 1, \ldots, N,$$

where $g(d_j)$ is the gain value for the label of $d_j$
DCG Example

• 10 ranked documents judged on 0-3 relevance scale:
  – 3, 2, 3, 0, 0, 1, 2, 2, 3, 0

• Discount factors = 1/1, 1/1, 1/1.59, 1/1.7, 1/2.0, 1/2.59, …

• Discounted gain = 3, 2/1, 3/1.59, 0, 0, 1/2.59, 2/2.81, 2/3, 3/3.17, 0
  = 3, 2, 1.89, 0, 0, 0.39, 0.71, 0.67, 0.95, 0

• DCG (cumulative sum) = 3, 5, 6.89, 6.89, 6.89, 7.28, 7.99, 8.66, 9.61, 9.61

• To compare algorithms, DCG numbers are averaged across a set of queries at specific rank values: DCG-5, DCG-10
Normalized DCG (nDCG)

- DCG values are normalized by comparing the DCG at each rank with the DCG value for the perfect ranking
  - This makes averaging easier for queries with different numbers of relevant documents

- **Perfect ranking** = 3, 3, 3, 2, 2, 2, 1, 0, 0, 0
- **Ideal DCG values** = 3, 6, 7.89, 8.89, 9.75, 10.52, 10.88, 10.88, 10.88, 10
- **Example DCG values** = 3, 5, 6.89, 6.89, 6.89, 7.28, 7.99, 8.66, 9.61, 9.61
- **nDCG value of ranking** = 1, 0.83, 0.87, 0.76, 0.71, 0.69, 0.73, 0.8, 0.88, 0.88
Evaluation Process

• Inputs needed: **Test queries** and **Relevance assessments**

• **Test queries:**
  – Best designed by domain experts

• **Relevance assessments:**
  – Human judges, time-consuming, may not be perfect, biased

• **Can we avoid human judgment? Not really**
  – Makes experimental work hard, especially on a large scale

• **In practice: use implicit feedback (clicks, bounce rate)**