Evaluation of text retrieval systems

- Assess the actual utility of a TR system
- Compare different systems and methods

What to measure
- Effectiveness/Accuracy: How accurate are the search results?
- Efficiency: how quickly can a user get the results? How much computing resources are needed to answer a query?
- Usability: How useful is the system for real user tasks?

The Cranfield Evaluation Methodology
- build a reusable test collections and define measures

Test Collection Evaluation
- need queries
- need documents
- have a list of relevance judgments

The if you have two systems and want to run them then run the systems on the documents and get the results of the running.

Which system is better?
How to quantify?

———-
Precision - percentage of correct predictions relevance
Recall - the total number of relevant documents

Evaluating a set of retrieved documents
- retrieved or not retrieved
- relevant or not relevant

Precision = relevant retrieved / (relevant retrieved + retrieved not relevant)
Recall = relevant retrieved / (relevant retrieved + relevant not retrieved)

Precision at 10 docs (the first page of the web results)

Combine Precision and Recall: F-measure
- harmonic mean of precision and recall
- F_b = ((B^2 + 1)P * R) / ((B^2*P) + R)
- F_1 = 2PR / (P+R)
  - rewards a case were P and R are similar
- B: parameter (often set to 1)
- Why not .5*P + .5*R?
  - sum so don't care about the other variable

Summary
- precision: are the retrieved results all relevant?
- recall: have all the relevant documents been retrieved?
- F measure combines Precision and Recall
- Tradeoff between Precision and Recall depends on the user's search task

— —
Evaluation of Ranked Lists
- precision recall curve
- precision the Y axis and recall on the x axis
- compare precision and recall curves to obtain which system is better
  - what if the curves cross?
    - depends if the user cares about whether the recall is high
    - if the user is examining a low number of results then the curve on top is better before it crosses the curve on the bottom
    - if the user is examining a high number of results then the curve on the top after the cross is better than the curve on the bottom after the cross.
- how to summarize ranking
  - measure the area under the curve
- average precision
  - combine all the precision numbers then divide by the total number of documents

— —-
Mean average precision (MAP)
- average precision:
  - the average of precision at every cutoff where a new relevant document is retrieved
  - normalizer = the total number of relevant docs in the collection
  - sensitive to the rank of each relevant document
- mean average precisions (MAP)
  - MAP = arithmetic mean of average precision over a set of queries
  - gMAP = geometric mean of average precision over a set of queries
  - which is better MAP or gMAP?

Special Case: Mean Reciprocal Rank
- when there's only one relevant document in the collection (e.g., known item search)
  - average precision = reciprocal rank = 1/r, where r is the rank position of the single relevant doc
    - r indicates how much effort a user would have to make to find the relevant document
  - mean average precision -> mean reciprocal rank
  - why not simply using r?

Summary
- precision-recall curve characterizes the overall accuracy of a ranked list
- the actual utility of a ranked list depends on how many top-ranked results a user would examine
- average precision is the standard measure for comparing two ranking methods
  - combines precision and recall
  - sensitive to the rank of every relevant document

— —

Multi-level relevance judgments?
- relevance level: r = 1 (non-relevant), 2 (marginally relevant), 3 (very relevant)
- gain is the measure of the relevance of the documents
  - how gain a relevant information can a user gain (1,2,3 points etc.)

Cumulative Gain
- the sum of the relevant documents don't care were they are ranked

Discounted Cumulative Gain
- divide the gain by the log of position
- discounting the documents based upon there position in the listing

Ideal Document Cumulative Gain
- ideal lister would rank the highest rated documents first (ideal ranked list)
  - compute DCG for ideal ranked list

Normalized DCG
- document cumulative gain / ideal document cumulative gain
- maps the values between 0 and 1

If we don't normalize different topics will have different scales of document cumulative values

Normalized Discounted Cumulative Gain
- applicable to multi-level judgments in a scale of [1,r], r>2
- measure the total utility of the top k documents to a user
- normalized for comparability across queries

———-
Practical Issues
- Challenges in creating a test collection
- judgments: completeness vs. minimum human work
- measures: capture the perceived utility by users

Statistical Significance Tests
- How sure can you be that an observed difference doesn't simply result from the particular queries you chose?
- assess the variance across the tests
- wilcoxon look at the signs and the magnitude of the difference

Pooling: Avoid Judging all Documents
- If we can't afford judging all the documents in the collection which subset should we judge?
- Pooling strategy
  - choose a diverse set of ranking methods (TR systems)
  - Have each to return top-K documents
  - Combine all the top-k sets to form a poll for human assessors to judge
  - other (unjudged) documents are usually assumed to be non-relevant (though they don't have to)
  - okay for comparing systems that contributed to the pool, but problematic for evaluation new systems

Summary of TR Evaluation
- TR is an empirically defined problem

==========
WEEK 4
==========
Probabilistic models: f(d,q) = p(R=1|d,q), R element of {0,1}
- classic probabilistic model -> BM25
- Language model -> query likelihood
- Divergence-from-randomness model -> PL2

p(R=1|d,q) equivalent to p(q|d,R=1)
- if a user likes document d, how likely would the user enter query q (in order to retrieve d)?

General idea
- f(q,d) = p(R=1|d,q) = count(q,d,R=1)/count(q,d)
  - estimate the probability of relevance
- can score the document for the query using the probabilistic model
  - using the probabilities of the relevance
- this wont work for unseen queries or unseen documents

Query Likelihood Retrieval Model
- f(q,d) = p(R=1|d,q) similar p(q|d,R=1)
  - p(q | how likely the user enters q
  - | d, R = 1) how likely the user likes d
- Assumption: a user formulates a query based on an “imaginary relevant document”

Summary
- Relevance(q,d) = p(R=1|q,d) -> p(q|d,R=1)
- Query likelihood ranking function: f(q,d) = p(q,d)
  - Probability that a user who likes d would pose query q
- How to compute p(q|d)?
  - How to compute probability of text in general? -> Language Model

Statistical Language model
- What is a language model
  - probability distribution over word sequences
    - p(“Today is Wednesday”) approximately equal to .001
  - context dependent
  - can also be regarded as probabilistic mechanism for “generating” text, thus also called a “generative” model
    - generate a sequence of text
- Why is a LM useful
  - Quantify the uncertainties in natural language
  - Allows us to answer questions like:
    - Given that we see “John” and “feels”, how likely will we see “happy” as opposed to “habit” as the next word? (speech recognition)
    - Given that we observe “baseball” three times and “game” once in a news article, how likely is it about “sports”? (text categorization, information retrieval)

    - Given that a user is interested in sports news, how likely would the user use “baseball” in a query? (information retrieval)
- The simplest language model: unigram LM
  - generate text by generating each word independently
  - Thus, p(w1,w1,…,wn) = p(w1)p(w2)…
  - Parameters: {p(wi)} p(w1) + p(wN) = 1 (N is voc.size)
  - Text - sample drawn according to this word distribution
- Unigram LM p(w|theta) -> sampling Document = ?
  - Topic 1: texting mining
    - words: text, mining, association, clustering (generate with probabilities)
    - can generate a text mining paper with a very small probability
  - Topic 2: health
    - words: food, nutrition, healthy, diet
- Estimation of Unigram LM
  - unigram LM p(w|theta) = ? <- estimation
    - Text mining paper d
    - total number of words = 100
    - generate a paper with the probabilities of the words
  - Maximum Likelihood (ML) Estimator:
    - p(w|theta) = p(w|d) = c(w,d) / |d| (is this the best estimate ?)
    - parameter settings are those that give are observed data the maximum probability
- LMs for Topic Representation
  - General Background English Text
    - Background LM: p(w|B)
  - Computer Science Papers
    - Collection LM: p(w|C)
  - Text mining paper
    - Document LM: p(w|d)
- LMs for Association Analysis
  - What words are semantically related to “computer”?
  - Topic LM: p(w|”computer”)
  - Documents containing the word “computer”
  - remove the words that are not semantically related to computer
    - Background model tells us what words are common in general
  - Normalized Topic LM:
    - p(w|”computer”) / p(w|B)
- Summary
  - Language Model = probability distribution over text
  - Unigram Language Model = word distribution
  - Uses of a Language Model
    - Representing topics
    - Discovering word associations

Query Likelihood Retrieval Function
- Query Generation by Sampling Words from Dic
- If the user is thinking of this doc, how likely would she pose this query?

Unigram Query Likelihood
- p(q = “”| d = “”)
  - p(q = w1 | d) * p(q = w2 | d)
  - (c(“presidential”,d) / |d|) * (c(“campaign”,d) / |d|)
  - each word is generated independently

Does Query Likelihood Make Sense?
- how to score the documents
  - simple count and then use the formula (above)

Try a Different Query?
- what if the query contains a word that is not in any of the documents

Improved Model: Sampling Words from a Doc Model
- How likely would we observe this query from this doc model?

Computation of Query Likelihood
- compute the document LM
- document -> document LM
  - generate a query then obtain the probability of the query using the Document LM

Summary: Ranking based on Query Likelihood
- query = w1w2..wn
- p(q|d) = p(w1|d) * p(w2|d) * … * p(wn|d)
- f(q,d) = log p(q|d) = summation i to n log p(w_i|d) = summation c(w,q) log p(w|d)
  - do this to avoid multiplying a lot of small probabilities together
  - maintain the order of the documents and avoid the underflow problem
  - the product becomes the sum
- retrieval problem -> estimation of p(w_i|d)
- different estimation methods -> different ranking functions

Statical Language Model Part 1
- Ranking function based on query likelihood (above)
- How to estimate p(w|d)
- Max Likelihood Estimate
  - P_ml(w|d) = (c(w,d) / |d|)
- normalize the word frequencies in the document
- represent the frequencies of the words by
  - p(w|d) vertical axis, word horizontal axis
  - step function - which means all the words that have the same frequencies counts will have the same probabilities
  - words that have not occurred in the document will have zero probability
- Smoothed LM
  - p(w|d) > 0 even if c(w,d) = 0
  - attempt to recover the model for the whole article

How to smooth a LM
- what probability should be assigned to an unseen word?
- Let the probability of an unseen word be proportional to its probability by a reference LM
- one possibility: Reference LM = Collection LM
  - p(w|d)
    - P_seen(w|d) if w is seen in d (discounted ML estimate)
    - alpha_d p(w|C) otherwise (Collection language model)

Rewriting the ranking function with smoothing
- log p (q|d) = summation c(w,q) log p (w|d)
  - this is sum over all the query words
- = summation c(w,q) log p_seen(w|d) + summation c(w,q) log alpha_d p(w|C)
  - decomposed into two parts query words matched in document
  - words not matched in document
- rewriting the ranking function
  - summation of the c(w,q) log p_seen(w|d) / (alpha_d * p(w|C)) + |q| log alpha_d
  - + summation c(w,q) log p(w|C)

Benefit of Rewriting
- better understanding of the ranking function
- smoothing with p(w|C) -> TF-IDF weighting + length norm.
- enable efficient computation

Statistical Language Model Part 2
- the main part of the formula is over the matched query terms
- matched query terms TF weighting / IDF weighting + Doc length normalization + Ignore for ranking
  - logarithm is a fixed form of the formula
  - if drop the logarithm the model won't work as well
- summary
  - smoothing of p(w|d) is necessary for query likelihood
  - general idea: smoothing with p(w|c)
    - the probability of an unseen word in d is assumed to be proportional to p(w|c)
    - leads to a general ranking formula for query likelihood with TF-IDF weighting and document length normalization
    - scoring is primarily based on sum of weights on match query terms
  - However, how exactly should we smooth?
    - which unseen word should have a higher probability

Smoothing methods Part 1
- Query Likelihood + Smoothing with p(w|c)
  - how to smooth p(w|d)
  - how to set alpha

- Linear Interpolation Smoothing
  - p(w|d) = (1- lambda) * (c(w,d) / (|d|)) + lambda * p(w|C)
    - (c(w,d) / (|d|)) is the maximum likelihood estimation
    - smoothing parameter is lambda [0,1]
      - the larger the value the more smoothing takes place
    - collection LM p(w|c)
- Dirichlet Prior (Bayesian) Smoothing
  - p(w|d) =
    - (|d| / (|d| + u)) * (c(w,d) / |d|) + ((u) / (|d| + u)) * p(w|C)
    - u element of [0,+infinity]
    - p(w|d) = (c(w;d) + up(w|C)) / (|d| + u)

Smoothing methods part 2
- Ranking function for JM Smoothing
  - (P_seen(w|d)) / alpha_d * P(w|C) =
    - ((1-lambda) * p_ml(w|d) + lambda * p(w|C)) / (lambda * p(w|C)) =
    - 1 + ((1 - lambda) / lambda) * c(w,d) / |d| * p(w|C)

Summary
- two smoothing methods
- both lead to state of the art retrieval functions with assumptions clearly articulated (less heuristic)
  - also implementing TF-IDF weighting and doc length normalization
  - has precisely one (smoothing) parameter

Summary of query likelihood probabilistic model
- effective ranking functions obtained using pure probabilistic modeling
  - assumption 1: relevance(q,d) = p(R=1|q,d) approximately equal to p(q|d, R = 1) approximately equal to p(q|d)
  - assumption 2: Query words are generated independently
  - assumption 3: smoothing with p(w|C)
  - assumption 4: JM or Dirichlet prior smoothing
- less heuristic compared with VSM
- many extensions have been made