Query Expansion and You:

A cautionary tale

Harry Scells 2015

A brief outline

- Query Expansion
  - Using EMIM / Bo1 probability [1]
  - EMIM $$w(t) = p(a,b) \cdot log \frac{p(a,b)}{p(a)p(b)}$$
  - Bo1 (Bose-Einstein 1) $$w(t) = tfx \cdot log_2 \frac{1 + P_n(t)}{P_n(t)} + log_2(1 + P_n(t))$$
- Query Term Weighting $$P_n(t) = \frac{tfc}{N}$$
  - Using rough estimation of concept weighting [2]
- BM25 used for indexing, first round of retrieval and for retrieving documents used in expanding the queries

Elastic Search

* Treating this like a black box was not the best decision
* Lots of workarounds needed due to REST API
* Calculating TF for the collection requires a call to web server
    * can use memoization, still not very good performance
* Difficult to find out how the documents are being stored

Query Expansion

* Tested using both EMIM and Bo1
    * Take the top `n` documents
    * Expand using a model
    * Take the top `t` terms from the expansion and append them to the original query
    * EMIM was found to perform slightly worse than Bo1

Tuning variables

* At index-time, the BM25 model is tuned to the standard k=0.75 and b=0.5
* The number of documents and the number of terms to expand to can also be tuned
  * documents (n) was manually tuned to 20
  * terms (t) was manually tuned to 5

Query Expansion Cont.

* Bo1 takes into account the top documents, as well as it's overall "importance" in the collection
  * will return more technical/ medically accurate terms than EMIM

Implementation

```clojure
(defn prob
  "calculate the probability of n gram x in terms n"
  ([x n]
    (cond
      (zero? (count n)) 0.1
      :else
        (float (/ (count (filter #{x} n)) (count n)))))
  ([x y n]
    (cond
      (zero? (count n)) 0.1
      :else
        (float (/ (+ (count (filter #{x} n)) (count (filter #{y} n))) (count n))))))

(defn pmi
  "implementation of Pointwise Mututal Information"
  [a b n]
  (cond
    (zero? (* (prob a n) (prob b n))) 0.1
    :else
    (Math/log (/ (prob a b n) (* (prob a n) (prob b n))))))

(defn emim
  "implementation of Expected Mutual Information Measure"
  [a b n]
  (* (prob a b n) (pmi a b n)))
```

```clojure
(defn log2 [n]
  (/ (Math/log n) (Math/log 2)))

(def N (inputs :N))

(defn tfc
  [term]
  "get the tf value for a term in the index"
  (let [conn  (esr/connect (connection/config :host))
        res   (esd/count conn (connection/config :index-name) "document" (q/term :text term))]
      (let [c (get res :count)]
        (cond
          (zero? c) 0.1
          :else c))))

(def tfc-memoize (memoize tfc))

(defn Pn
  [t]
  (/ (tfc-memoize t) N))

(defn bo1
  [_ t n]
  (* (prob t n) (+ (log2 (/ (+ 1 (Pn t)) (Pn t))) (log2 (+ 1 (Pn t))))))
```

Some Examples

* n=10 t=10 bo1
    * “cloudy cornea and vision problem”
    * “corneal transplant cloudy infection surgery vision people keratoplasty eye and cornea tissue problem dystrophy conditions”
    * ([cloudy 1] [vision 1] [and 1] [cornea 1] [problem 1] [keratoplasty 0.1220986463102716 5] [surgery 0.11811875313826738] [corneal 0.10698035201059863] [conditions 0.10469290378014434] [eye 0.09351981921854947] [infection 0.08753013223180833] [transplant 0.08666527104530014] [dystrophy 0.0799956633212258] [tissue 0.07789051199279287] [people 0.0776688980659462 5])

Some Examples

* n=10 t=10 bo1
    * “toddler having squeaky breath”
    * “pregnancy it's padding breath bronchitis squeaky tantrums babycenter baby having holding acute toddler toddlers”
    * ([breath 1] [squeaky 1] [having 1] [toddler 1] [bronchitis 0.2579155664050002] [baby 0.1922489813589645] [babycenter 0.1915944120420775 8] [holding 0.12281693638333344] [acute 0.09334087383031404] [pregnancy 0.08641078868975333] [padding 0.07860283710635275] [toddlers 0.07123382419182371] [tantrums 0.06140846819166672] [it's 0.05895213055349037])

Some Examples

* n=3 t=5 bo1
    * “red itchy eyes”
    * “allergies eyes red itchy allergy drops allergic conjunctivitis”
    * ([allergies 1.6982935384829927] [eyes 1] [red 1] [itchy 1] [allergy 0.5244363195737437] [drops 0.47368441767951047] [allergic 0.29335873052909545] [conjunctivitis 0.2757912644583393])

Some Examples

* n=3 t=10 bo1
    * “red itchy eyes”
    * “story lenses eyes keratoconjunctivitis allergic drops eye conjunctivitis red allergy itchy allergies commonly”
    * ([allergies 1.6982935384829927] [eyes 1] [red 1] [itchy 1] [allergy 0.5244363195737437] [drops 0.47368441767951047] [allergic 0.29335873052909545] [conjunctivitis 0.2757912644583393] [eye 0.24888648840049404] [keratoconjunctivitis 0.21992490820834415] [story 0.15225570568269978] [lenses 0.135338405051288 7] [commonly 0.135338405051288 7])

Query Term Weighting

* After expanding the query, can try to weight each term in the query
* A method of finding a weight for each term given the query, and then boosting each term accordingly
* Highly modified implementation of a previous work involving boosting individual terms in a query [2]
* Found to actually decrease the score of the standard Lucene search
    * p@10 Lucene Search = 0.2323
    * p@10 Query Term Weighting = 0.1508

- Similar to the model used in “Discovering Key Concepts in Verbose Queries”
- Weighting a concept (term) in a query is done using the following formula:
- Where:
  - n is the # of terms in top t documents
  - q is the query
  - ci is the term

$$p(c_i|q) = \frac{h_k(c_i)}{\sum_{c_i \in q} h_k(c_i)}$$

$$h_k(c_i) = log\left[1 + \frac{c_i}{n}\right]$$

* The term and it’s concept weight is fed into an elastic search “function_score” query.
* This is a combination of a query string search and each individual term with it’s own weighting added

```
{:query
  {
   {:function_score
    {
      :query (q/query-string :query "apples and grapes" :default_operator "OR")
      :functions
      [
        {:filter {:term {:text "apples"}} :weight 0.10423}
        {:filter {:term {:text "and"}} :weight 0.00142}
        {:filter {:term {:text "grapes"}} :weight 0.11223}
      ]
    }
   }
  }
}
```

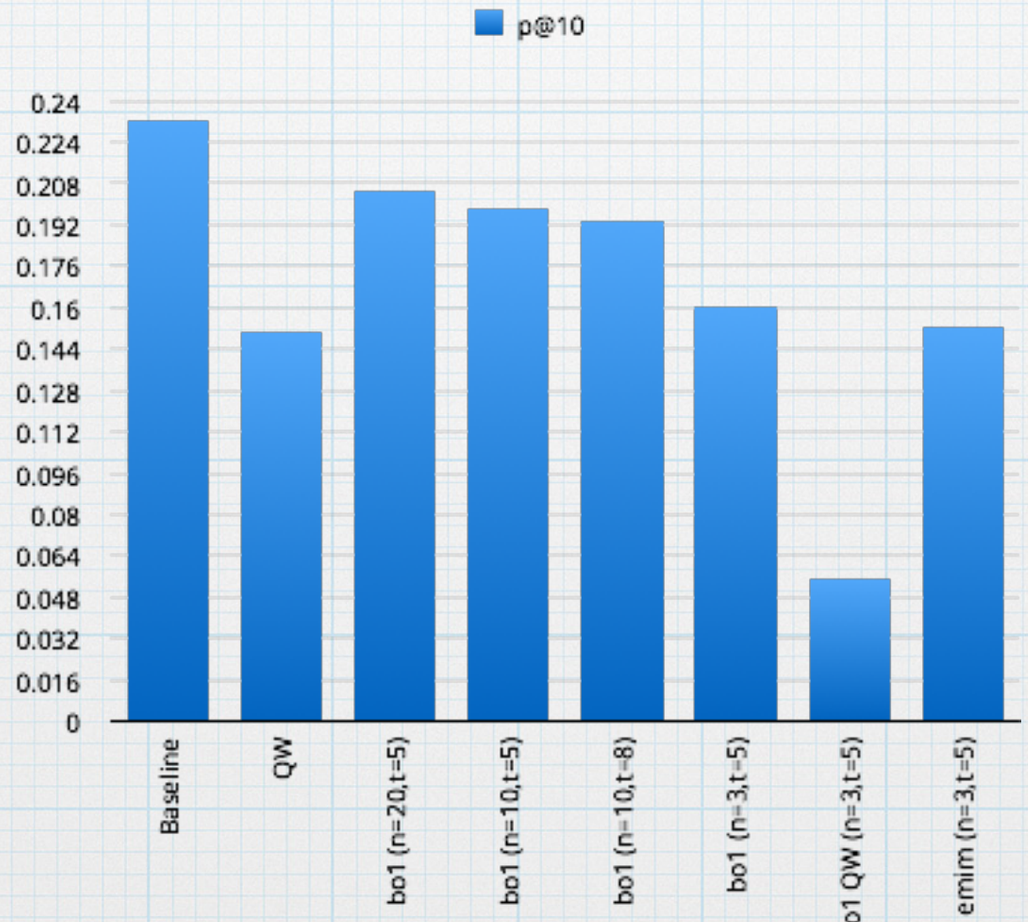
Combining Models

* The bo1 query expansion model combined with the concept weighting model
* Query gets expanded
* Terms are weighted

Results

p@10 results for different methods of querying

	Baseline	QW	bo1 (n=20,t=5)	bo1 (n=10,t=5)	bo1 (n=10,t=8)	bo1 (n=3,t=5)	bo1 QW (n=3,t=5)	emim (n=3,t=5)
p@10	0.2323	0.1508	0.2046	0.1985	0.1938	0.1600	0.0554	0.1523



Why the bad results?

* Most likely due to bad index tuning
* The query expansion is only as good as the baseline search results (which was surprisingly good)
* Tuning Index parameters should provide better query expansion
* Adding more documents for the query to expand on could increase accuracy, but hinders performance dramatically

Where to go from here?

* Look into combining the query expansion weighting scores into the concept weighting model
* Picking a better function to weight query terms on
* Investigate indexing & look into precomputing scores for performance
* Dive deeper into the code
    * Modelling on top of a black box limited the potential of the system
    * Uncertainties about the system
    * Performance improvements

What did we learn?

* Don’t make the same mistakes
    * Get as close as possible to the source code as you can, otherwise there **will** be workarounds
    * Treating a search engine as a black box is great, so long as it has great documentation, you know how the internals work, and your models are fantastic
    * If you want performance, don’t do everything over a REST API 😀 (expensive queries can take many seconds to complete)

Appendix

* Resources cited:
    * [1] UBML participation to CLEF eHealth IR challenge 2015: Task 2
    * [2] Discovering Key Concepts in Verbose Queries
* Further Reading
    * Code documentation
        * http://hscells.github.io/health-search/doc/
    * Source code
        * https://github.com/hscells/health-search

Questions