Cluster Rank Demo Harness - Unfortnate Series of Events

Philip Robinson

Oregon Health & Science University

March 12, 2018

Abstract

It is often the case that initial query compositions result in frequent restarts as the user negotiates with their retrieval system. This is likely a product of a incompatible understanding mapping a query formulation against the choice of ranking algorithm, which results in monolithically incorrect results. To reduce user restarts, the proposed model clearly separates the retrieval process into the three steps `relevance`, `clustering`, and `ranking`. Relevance designates the total document set to operate over. Ranked-clusters are presented to the user without pollution from other groups, allowing users to select a document, or a group to recurse the `cluster` and `ranking` steps on. The final product is a demo harness that abstracts these steps, so that others may easily produce, test, and reproduce prototypes against their own corpora; and a likewise an oracle of the system for evaluation.

1 Introduction

Information retrieval systems have long suffered from non-informative query formulations by their users. Many systems employ techniques such as query expansion, domain ontologies, advanced search parameters, to address such difficulties. Unfortunately, most retrieval systems presume user provided queries to be informative, and select to return the most query-relevant documents. We can interpret a query-relevance ranking on documents retrieved by a non-informative query as being an over-fit ranking (to the query). Unfortunately, this over-fitting can happen regardless of query quality. This often results the document set retrieved being nearly identical. Either in the case of non-informative queries or of monolithic document listings, users are often tasked with query reformulation. A side effect of this workflow, query reformulation can make it difficult to assess the general effectiveness of a retrieval system [1].

To prevent over-fit rankings some retrieval systems have proposed document diversification strategies [2]. This can be accomplished by introducing similarity or taxonomic clustering of documents either prior to or after query submission. Additionally, to improve query formulation many search engines provide similar search terms along side retrieved documents. These aid terms are usually assigned by experts to similar vocabulary or to retrieved documents, and can be expensive to curate.

1.1 Proposal

This model harness allows for prototyping information retrieval systems against corpora where clustering and ranking do not inform eachother. This system separates the query processing pipeline into quarantined steps. First identifying documents by relevance, then perform unsupervised clustering of relevant documents, finally rank each cluster by the query provided. We then allow the user to zoom in on a cluster. During this process, every cluster is also tagged with terms automatically, by providing the grouping's most impactful `tfidf` words. These terms don't impact the ranking, but act to provide the user with additional information in selecting a term.

It is hypothesised that a document clustering will allow users to eliminate poor document groupings. Additionally automatic tagging of clusters with relevance terms should allow users to navigate a listing retrieved from non-informative queries. To form queries, sample words are randomly selected documents' `tfidf` distributions and capture the resulting ranking $R_{\texttt{tfidf}}$. For a system with C-way clustering to be considered permanent, the same query should yield our target document within S steps.

$$S < log_C(R_{\texttt{tfidf}})$$

As this is a process greatly empowered by composable components, we also separate out these concerns in code, to hopefully decrease testing time against multiple corpora, ranking, and clustering configurations.

2 Implementation Details

The retrieval systems harness is developed in `python`[1]. The harness is designed to be used from the command line, but hosts results in a manner accessible to a flask server (that is also provided).

In order to reduce complexity[2], `numpy` arrays were used to track, sort, and split document collections. Interfaces that required of designers must yield index arrays. Unfortunately, (presently) this limits corpus scaling to what can be handled by `numpy`. As an additional restriction, corpora are required to be transformed to `numpy` compliant structures.[3]

Provided is the `olib.Handler` class that orchestrates the cluster, ranking, steps as a coroutines. An `olib.Organizer` is passed as a constructor to the `olib.Hanlder`. `olib.Organizers` which are `numpy.ndarrays` extended with interfaces for `.cluster(vocabulary, n_clusters)` and `.rank(*rankargs)`, which raise `NotImplemented` unless provided by designer. Designers implement their own `olib.Organizers` and overload these methods.

For test cases `kmidf.CosKMIDFROrganizer`, a `tfidf` ranker, is coupled with k-means over cosine distance, and `ldaidf.LDAIDFOrganizer`, a `tfidf` ranker, coupled with Latent Dirichlet Allocation max clustering. Classes prepended with `Augmented` have been altered for evaluation purposes.

[1] `https://github.com/probinso/IR-cluster-rank-demo`

[2] and scratch the author's wish to use array languages

[3] examples found in code

Modifications to a handlers allow an oracle into the performance of a system. `olib.AugmentedHandler` tags a cluster's meta-data with ownership of a target document, and it's rank location. This allows inspection and traversal of a guided path through the documents. Discovery of a document is measured by it's being placed in the displayed range (top 3 by default) for a cluster.

```
class IDFRankOrganizer(Organizer):
  def rank(self, queryvec):
    dist = np.sum(self[:, queryvec],axis=1)
    idx_dist = sorted(enumerate(dist),
                      key=itemgetter(1),
                      reverse=True)
    idx  = np.array([key for key, value
                     in idx_dist])
    return idx
```

```
class CosKMOrganizer(Organizer):
  def cluster(self, terms, n_clusters=3):
    dist = 1 - cosine_similarity(self)
    alg  = KMeans(n_clusters=n_clusters)

    # Fit cosine distance kmeans
    results  = alg.fit(dist)
    labels   = results.labels_
    complete = dict()
    ulabels  = np.unique(labels)
    for l in ulabels:
      keys = np.array([i for i, b in
               enumerate(labels==l)
               if b])
      complete[l] = keys

    ... # populate meta

    return complete, meta
```

```
class LDAOrganizer(Organizer):
  def cluster(self, terms, n_clusters=3):
    lda = LatentDirichletAllocation(
      n_topics=n_clusters)
    lda.fit(self)

    topics = lda.transform(self)

    # argmax is a bad assignment choice
    assigm = topics.argmax(axis=1)
    labels = np.unique(assigm)

    complete = dict()

    for l in labels:
      complete[l] = np.where(assigm==l)[0]

    ... # populate meta

    return complete, meta
```

```
class CosKMIDFROrganizer(
    CosKMOrganizer, IDFRankOrganizer):
  pass

class LDAIDFROrganizer(
    LDAOrganizer, IDFRankOrganizer):
  pass
```

3 Evaluation

3.1 Approach

In order to provide repeatable results without depending on expertly relevance and ranked corpora, against traditional `tfidf`. Relevance is determined as any document containing any of the query terms, and it's rank is defined as $R_{\texttt{tfidf}}$ denoting its place in the listing, from `tfidf` score against these documents. As mentioned above, for a system with C-way clustering to be considered proffered, an identical query should yield our target document within S steps.

In this case, the clusters contain unique document lists[4]. This means that success is measured by satisfying bounds for $S < log_C(R_{\texttt{tfidf}})$. Using the augmented structures, described above, clusters are labeled to contain the target document. This however would not be the case in a real environment. It is also important to note that some documents contained the only instance of a word in the corpus. To prevent sampling these words, we eliminated words with `tfidf` scores ≥ 1

[4]It is not a restriction of the system that clusters need be unique

from our sample space.

Tests are generated against two different systems, the `kmidf.CosKMIDFROrganizer` and the `kmidf.LDAIDFROrganizer`, with a collection of tenthousand medical abstracts.

The unit retrieval was a structure of cluster labels and corresponding document labels with meta-data. The image below is an example of a first response on `kmidf.CosKMIDFROrganizer`.

```
{   0: {
    'documents': [
      'Molecular Mechanisms Governing Cooperating Motors',
      'A wrinkle in time: The perception of expressions in the elderly',
      'Computer simulations of populations of mammalian motor units'],
    'meta': [
      'epitheli', 'knockout',
      'overexpress', 'mutant',
      'homeostasi', 'downstream',
      'extracellular', 'kinas',
      'induct', 'stem',
      'HERE!!! :27']},
  1: {
    'documents': [
      "The Cyclical Lower-extremity Exercise for Parkinson's Trial",
      'OLDER BREAST CANCER PATIENTS: RISK FOR COGNITIVE DECLINE',
      'Statewide Intervention to Reduce Early Mortality in Persons with Mental Illness'],
    'meta': [
      'extracellular', 'intracellular',
      'knockout', 'mutant',
      'overexpress', 'transgen',
      'apoptosi', 'transcript',
      'mammalian', 'ligand']},
  2: {
    'documents': [
      'Optic Nerve Head SDOCT Imaging in Glaucoma',
      'Multimodal Connectivity for Surgical Mapping',
      'Molecular Imaging Directed, 3D Ultrasound-guided, Biopsy System'],
    'meta': [
      'apoptosi', 'block', 'cycl',
      'diabet', 'dna', 'elucid',
      'endotheli', 'homeostasi',
      'host', 'immun']}}
>> 0
```

3.2 Results and Discovered Limitations

After selecting many documents, from the corpus, at varying query strengths we were rarely able to satisfy the success criteria. This may have been due to many factors, some of which arose as a serious consequence of pure segregation of clustering from ranking. Ranking that isn't informed by clustering would retain the same top documents, unless you were (randomly) lucky enough split along the top ranked documents. This behavior resulted in stagnating documents at the top of reported clusters, suppressing the target document from being selected.

This document suppression was especially bad in k-means. This was likely because the the scoring metric for document similarity was in the same space as the ranking. Consequently, if a target document was close to a dense centroid, then it would likely never hit the visible documents list; as its competing documents would likely continue to score high from that region. This was a

poor outcome.

Additionally, the soft clustering (found in Latent Dirichlet Allocation) introduced problems with reasonable group assignment. As groups became smaller, topics became less informative and would often break down. Complicated group assignment algorithms (informed by LDA) were used, but resulted in meaningless splits nearing the end of a query session.

The most tested document, `id(9871)`[5], was selected due to the evaluator's ability to understand it's content. The hope was to use a cluster's meta-data, in the form of cluster informed keywords, to guide sub cluster selection. Unfortunately, due to document suppression, dense documents clusters what overlapped with the query results also overloaded the meta search terms. Finally, `tfidf` distributions weren't updated between clustering steps, which resulted in stabilization of terms in groups, (eventually leading to many shared terms across groups). The evaluator was unable to appraise cluster selection without the oracle by use the assisted query terms; likely due to the costs of document suppression. Similar results were found with other documents.

Table 1: Document Retrieval Performance

doc_id	Query	$R_{\texttt{tfidf}}$	LDA (S, R_S)	Kmeans (S, R_S)
9871	motor younger medic vivo	52	0:15, 1:12, 2:12, 3:11	0:31, 1:27, 2:19, 3:10
			0:31, 1:30, 2:30, 3:30	0:31, 1:27, 2:17, 3:5
			0:31, 1:27, 2:27, 3:26	0:31, 1:27, 2:17, 3:5
9871	motor age medic vivo	25	0:16, 1:14, 2:12	0:12, 1:1
			0:9, 1:9, 2:9	0:12, 1:1
			0:6, 2:6, 3:6	0:12, 1:1

Interestingly, since both k-means over cos-distance and LDA provided very stable results on this corpus. It has become apparent, that stable clustering may significantly reduce the effects that ranking may have when selecting a cluster to recurse on. Further work may lead to better strategies for perturbing these clusters, so that ranking can have more interesting effects.

3.3 Limitations

In order to support more sophisticated document clustering and ranking (like those using word2vec) we will need to loosen our restriction to `numpy` data structures. Additionally, significant speedups could be found in the document relevance lookup, resulting in faster testing of configurations to corpora.

Although this was proposed as an evaluation harness, several design decisions weren't made with evaluation in mind. The `AugmentedHandler` behavior should be moved to first priority in the class structure, as to provide sane defaults for potential users of the system.

[5]Molecular Aging of the Human Brain: Genetic Modulation and Functional Outcomes

4 Next Steps

Firstly, we would like to address any concerns raised from the limitations section. We are also interested in providing a full restful API so those who find these systems useful may easily host their corpora without much engineering effort.

We are also interested in extending user search to include selecting multiple clusters and inject new query terms without restarting their search.

5 Conclusions

This project will likely die in its current encarnation. The results weren't very promising, and the discovered fixes violate the original premis. We are still interested in starting clean with a retrieval system prototyping harness, but likely starting fresh with our newly gained knowlege.

References

[1] Jeff Huang and Efthimis N. Efthimiadis. Analyzing and evaluating query reformulation strategies in web search logs. In *Proceeding of the 18th ACM conference on Information and knowledge management - CIKM '09*. ACM Press, 2009.

[2] Yadong Zhu, Yanyan Lan, Jiafeng Guo, Xueqi Cheng, and Shuzi Niu. Learning for search result diversification. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval - SIGIR '14*. ACM Press, 2014.