Noname manuscript No.
(will be inserted by the editor)

A comprehensive performance analysis of various Information Retrieval algorithms for treccar experiments

Internal documentation

Sumanta Kashyapi · Dr. Laura Dietz

Received: date / Accepted: date

Abstract Abstract goes here with purpose, approaches and results.

Keywords Topic modeling · LDA · Information retreival · KMeans clustering · Unigram topic model

1 Introduction

Categorization of large set of textual materials is a well traversed track of research in the field of information retrieval. Typically in these problems we have a collection of text documents as our input. Using the textual information contained in each document we are to determine to which pre-defined category it belongs. In our case, we have a set of M articles $(A_m \mid m \in \mathbb{N}, m \leq M)$. For each article $\mathrm{A_m}$ we have a set of $\mathrm{K_m}$ section labels $(S_{mk} \mid k \in \mathbb{N}, k \leq K_m)$ and a set of $\mathrm{N_m}$ textual content or paragraphs $(P_{mn} \mid n \in \mathbb{N}, n \leq N_m)$ relevant to the article. For each article we have to map it's section labels to paragraphs based on the information we extract from the set of paragraphs. The assignment results for all the articles are compared with the correct assignment provided in a ground truth file. An evaluation framework is developed to quantify this comparison which in turn can be utilized to determine which categorization are better than the rest. Various categorization and classification algorithms can be used for the assignment task and for each approach we can measure it's effectiveness using the evaluation framework. For the experiments described here, we have used LDA topic modeling, KMeans, Unigram topic modeling and

F. Author
first address
Tel.: +123-45-678910
Fax: +123-45-678910
E-mail: fauthor@example.com

S. Author
second address

some of their variations. Our aim will be to identify which components of these algorithms are responsible for good results from the performance measures calculated by our evaluation framework. We will also carry out a feasibility study of a hybrid approach consisting of the best components for this task.

2 LDA topic modeling

LDA theory and mallet implementation details goes here.

Implementation mallet implementation details.

3 KMeans algorithm

KMeans theory and mallet implementation details goes here.

Implementation mallet implementation details.

4 Unigram topic model

In his paper, author Blei describes how his LDA model extended from the simple Unigram mixture model. According to the unigram mixture model the probability of a document $p(\mathbf{w})$ in a corpus can be expressed as:

$$p(\mathbf{w}) = \sum_{z} p(z) \prod_{n=1}^{N} p(w_n \mid z)$$

However, in this paper we considered modifying the existing LDA implementation to have an additional constraint that will enforce unigram behaviour to typical LDA model. The constraint we tried to implement here is each token in a single document has to agree upon a single topic in a single iteration. So we simply would not allow a state during any itrerations in which some token is assigned a topic different than it's adjacent ones. We did this by assigning topics to an entire document rather than individual tokens. However, we still considered individual tokens for probability calculations of the next topic assignment. Calculation of the distribution $p'(m)$, from where $\widetilde{k}$ is drawn, dominates the training phase. According to unigram model a topic is assigned to a whole document m and therefore all tokens t = 1 to T_m inside that document m takes the same topic $\widetilde{k}$ or $z_t = \widetilde{k}$ for all t = 1 to T_m. Hence each probability in the distribution $p'(m)$ can be expressed as:

$$p'(m_i \mid \overrightarrow{m_{-i}}, \overrightarrow{w}) = \prod_{t=1}^{T_m} p_t(z_t = k \mid \overrightarrow{z_{-t}}, \overrightarrow{w})$$

Data: Pre-processed corpus
Result: Trained Unigram topic model for input corpus
for *each doc m in corpus where* $m \in [1, M]$ **do**
  for *each topic i=1 to k* **do**
    calculate $p'(m_i \mid \overrightarrow{m_{-i}}, \overrightarrow{w})$;
  end
  sample a new topic $\widetilde{k} \sim p'(m)$;
  for *each token* $t \in [1, T_m]$ *in doc m* **do**
    decrement count from all data structures involving old topic assignment;
    assign new topic $\widetilde{k}$ to all data structures;
  end
end

Algorithm 1: Unigram topic model training phase

Being an extension of original LDA algorithm, we can still use the proportional relation

$$p_t(z_t = k \mid \overrightarrow{z_{-t}}, \overrightarrow{w}) \propto \frac{n_{k,-t}^{(v)} + \beta_v}{\sum_{v=1}^{V}(n_{k,-t}^{(v)} + \beta_v)}(n_{m,-t}^{(k)} + \alpha_k)$$

But $n_{m,-t}^{(k)} = 0$ for all m and k, so our proportional equation simplifies to:

$$p'(m_i \mid \overrightarrow{m_{-i}}, \overrightarrow{w}) \propto \prod_{t=1}^{T_m} \frac{n_{k,-t}^{(v)} + \beta_v}{\sum_{v=1}^{V}(n_{k,-t}^{(v)} + \beta_v)}$$

ignoring proportionality constant and assuming symmetrical β distribution over vocabulary

$$= \prod_{t=1}^{T_m} \frac{n_{k,-t}^{(v)} + \beta}{\sum_{v=1}^{V} n_{k,-t}^{(v)} + \beta Sum}$$

$$= \exp\left[\sum_{t=1}^{T_m} \log\left(n_{k,-t}^{(v)} + \beta\right) - T_m \log\left(\sum_{v=1}^{V} n_{k,-t}^{(v)} + \beta Sum\right)\right]$$

in log space

While implementing unigram model we computed $p'(m_i)$ in log space to avoid underflows. Also instead of directly sampling from p', we used the smoothed version (Jelinek-Mercer smoothing) p'' which is calculated as:

$$p''(m_i) = \lambda p'(m_i) + (1 - \lambda)$$

Here λ is inverse temperature which controls randomness during sampling. In other words, low λ leads to more *exploration* from wide range of topics and high λ will *exploit* from narrow range of topics. The topic inferencer for unigram model, which is used during the section to paragraph assignment, performs similar calculations except, instead of paragraphs in corpus it takes the text content related to section headings.

Implementaion implementation details.

5 Evaluation Framework

Before we try to experiment with different combinations of topic modeling and clustering modules with our treccar corpus, we must have a reliable framework to compare our results. A robust evaluation framework for our experiments is a key element which will help us to be confident about our findings. We will evaluate two aspects of our results. First, we have a set of measures which will determine the accuracy of the clustering that we have done with our set of data elements, which is in this case a set of paragraph objects identified by unique paragraph ids. Note that while measuring how good our clusters are compared to that of the ground truth, we do not utilize the assignment mappings or more precisely the section objects under which each cluster is mapped. In our second set of measure we do just that by comparing our section to paragraph mappings with that of ground truth.

5.1 Clustering measures

In our current evaluation framework we are using two basic statistical measures for evaluating our clustering results.

5.1.1 Adjusted RAND index

RAND index is a popular statistical measure used to quantify the similarity between two clusterings done on the same dataset. In our case those two clusterings are resulting clusters from our designed model and the clusters made according to the ground truth file. Although RAND index is often associated with the accuracy of the results but in our case we are using it without the class labels. Typically RAND index,R is calculated as:

$$R = \frac{a+b}{a+b+c+d} = \frac{a+b}{\binom{n}{2}} \tag{1}$$

where $a, b =$ pairs of paragraphs present in either same or differnt cluster for both clustering,
$c, d =$ pairs of paragraphs present in same cluster in one but in differnt cluster in other,
$n =$ no. of paragraphs But this simple RAND index expects the two partitions to have same number of clusters. However, we want the framework to be robust enough so that we can compare our results even when our resulting partition is of different size from that of the ground truth. Hence adjusted RAND index or ARI is used which is calculated as:

$$ARI = \frac{\sum_{ij}\binom{n_{ij}}{2} - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}}{1/2[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}] - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}} \tag{2}$$

where n_{ij} = contingency table values from ith row and jth column
a_i = contingency table ith row sum
b_j = contingency table jth column sum

5.1.2 Purity

Purity theory goes here

5.2 Assignment measures

After we finish clustering the paragraphs and assign them with their respective cluster labels, we have to assign them to sections. The quality of the section to paragraph assignment is measured using an existing framework, TREC eval, which is designed specifically for this set of problems where we have to evaluate rankings of retrieved information.

Different measures of TREC eval theory and usage of numq measures, map and rprec with background of precision

6 Implementation details

The algorithms, models and evaluation framework are developed in java. Mallet API has been used for basic text retrieval tasks and as a starting point for developing all the models used in the experiments.

6.1 Important classes

RunExperiment: It is the starting point of the evaluation framework. It contains most of the variables that control the workflow of the experiment including which algorithm and which version of it is to be used. It takes all the values for one set of experiments and constructs the topmost loop that calls the appropriate algorithm with it's respective values. Details of the varibles are in Table

SingleRun: At each iteration of *RunExperiment*, *SingleRun* is called with appropriate arguments. This is the class which decides which version of which algorithm is to be called. Depending on the RUN_BY_PAGE varibale of $RunExperiment$ class, $runExperiment()$ or $runExperimentWholeCorpus()$ method of $SingleRun$ is called from $RunExperiment$. Some important method details are discussed in Table.

```
Data: Pre-processed corpus and necessary parameters
Result: Performance results for each page
for each page p in corpus where i ∈ [1, P] do
    get Instance Lists of paragraphs and queries for p;
    call modelAndAssign() to get cluster and assignment results for p;
    check for any duplicate paragraph assigned;
    store results as p → results;
end
call MeasureExperiment methods with all p → results to obtain performance
measures for each p;
if SAVE then
    save measures to output file
end
```

Algorithm 2: *runExperiment*() algorithm

```
Algorithm modelAssign()
    switch model do
        case 1
            getNumTopics()
            train LDA model with numTopics, αSum, βSum, list of paragraph
            elements;
            call assignUsingLDA();
        end
        case 2
            getNumTopics()
            train KMeans model with numTopics, list of paragraph elements;
            call assignUsingKMeans();
        end
        case 3
            getNumTopics()
            train UMM model with numTopics, βSum, list of paragraph elements;
            call assignUsingUMM();
        end
        case 98
            call assignUsingAllCorrect();               // baseline measures
        end
        case 99
            call assignUsingRandom();                   // baseline measures
        end
    endsw
Procedure getNumTopics()
    if k == 0 then
        numTopics = size of query list;
    end
    else
        if k ≥ sizeofparagraphlist then
            numTopics = size of paragraph list;
        end
        else
            numTopics = k
        end
    end
```

Algorithm 3: *modelAndAssign*() algorithm

```
foreach paragraph p in list of paragraphs do
    foreach query q in list of queries do
        calculate paraQueryMatrix[p][q] = assignVal for p and q
        /* assignVal is the KL-div value for topic modeling algorithms and
           metric distance for clustering algorithms so that the value
           becomes proportional to the quality of p to q assignment      */
    end
end
initialize isParaAssigned[] = false
foreach paragraph p in list of paragraphs do
    bestQueryForPara = q where
    paraQueryMatrix[p][q] = min paraQueryMatrix[p]
end
while checkIfDone() do
    /* checkIfDone() will return true when all values in paraQueryMatrix
       is 0                                                              */
    get index (p, q) of minimum positive value in paraQueryMatrix
    assign para[p] → query[q]
    put −1 in place of row p and column q of paraQueryMatrix
    isParaAssigned[p] = true
end
foreach paragraph p in list of paragraphs do
    if isParaAssigned[p] = false then
        assign para[p] → bestQueryForPara[q]
    end
end
```

Algorithm 4: $assignUsing[modelName]()$ algorithm

```
foreach page pg in corpus do
    obtain clusters from ResultForPage[pg]
end
foreach key in gtMap do
    /* gtMap is the mapping between pageID and paragraph clusters
       according to ground truth file                                    */
    if WHOLE_CORPUS_MODE then
        get the list of lists of paragraphs as gtClusters
    end
    else if key starts with pageID then
        add gtMap[key] to gtClusters
    end
    candClusters = clusters[pg]
    intialize contMat[gtClusters[pg].size()][candClusters.size()]
                                                   // Contingency Matrix
    foreach row r in contMat do
        foreach column c in contMat do
            correct = gtClusters[r]
            candidate = candClusters[c]
            match =no. of matches between correct and candidate
            contMat[r][c] = match
        end
    end
end
```

Algorithm 5: Algorithm to form contingency matrix

Table 1 RunExperiment variables

Variable	Purpose
SAVE_RESULT	Whether we want to save our results
RUN_BY_PAGE	Whether we want to run by page or take the whole corpus as a single page. In case it is false, the problem changes from section to paragraph assignment to page to paragraph assignment. In other words, instead of calculating measures for each page, we now replace all the section titles with page titles and we try to assign paragraphs to it's respective pages. Consequently we calculate a single measure for the entire corpus.
CLUSTERING_MEASURE_FILENAME	The name of the file which will store the clustering measures.
TRECEVAL_ASSIGN_FILENAME	The name of the file which will store the section to paragraph assignments in trec_eval format which will be the input for trec_eval script.
SMOOTHED_UMM	Whether we are using smoothing in Unigram topic model.
model	Which algorithm/model is to be used. Currently we have five valid strings for this: $1 = LDA, 2 = KMeans, 3 = Unigram, 98 = Correct, 99 = Random$
[meta][Param]	These are the parameters for algorithms that we can vary. There are currently four parameters available which are k value, number of iterations for a single call to the respective algorithm, αSum and βSum two hyperparameters for topic modeling. Also for each parameter, we have three bounding values $[meta]$ to calculate different values of the parameter thoroughout the iterations which are start, stop and step. For a particular parameter, we start off the iterations with start value of the parameter until we get to the stop value adding the step value to the current value after each iteration.
isVar	Each boolean value in this array represents one variable. Currently it is of size four and from 0 to 3 it represents $k, iterations, \alpha Sumand\beta Sum$ respectively. For example, if we have the array as $[true, false, false, true]$ then it means we will consider $stopandstep$ values of $kand\beta Sum$ and iterate through different values of $kand\beta Sum$.

ResultForPage: This class is generated for each page and holds the clusters of paragraph ids ($paraClusters$) and the mapping of section ids to list of paragraph ids ($queryParaAssignment$).

MeasureExperiment: This class calculates the cluatering measures for a single page. Currently it calculates two measures, adjusted RAND index ($calculateRANDPerPage$) and Purity ($calculatePurityPerPage$) from the contingency matrix formed

Table 2 Important SingleRun methods

Method	Purpose
runExperiment	Performs experiment on per page basis. This method is implemented using algorithm 2.
runExperimentWholeCorpus	Performs experiment on the whole corpus.
modelAndAssign	Based on the *model* parameter passed from *RunExperiment* different algorithms are called. This method is based on algorithm 3.
convInsAssignToIDAssign	Coverts mappings from Section Instance object to list of Paragraph instances to mappings from Section id to list of Paragraph ids.
assignUsing[algo]	These methods takes the clusters of paragraphs generated by some algorithm and assigns them to section instances using [*algo*] algorithm ($assignUsingKMeans$ uses KMeans). Ideally any changes made to these methods will not affect the results of clustering measures because section to paragraph assignment information is not utilized while calculating those measures. Algorithm 4 is implemented in this method.
matrixAssignment	This is called from $assignUsing[algo]$ to do the assignment using the KL divergence matrix (for topic modeling algorithms) or the distance matrix (for clustering algorithms).

using the algorithm 5. Later on in $SingleRun$ these values are used to get the mean and standard error for the current experiment.

6.2 Sequence of execution

7 Experimental results

7.1 Baseline experiments

Two sets of experiments are carried out to form a baseline for other algorithms. In first set, we randomly assigned paragraphs to sections for each article. For this random assignment we got mean RAND index to be 0.0032. When it is ensured that every section got at least one paragraph, it comes down to -0.0014. more to add here purity, map, rprec

7.2 Minimum no. of iterations

Three sets of experiments were carried out for each algorithm to get the minimum number of iterations needed of the respective algorithm to reach sufficiently close to the highest result it can achieve. Each set of experiments were done using 4 set of measures; Adjusted RAND index, Purity, four numq

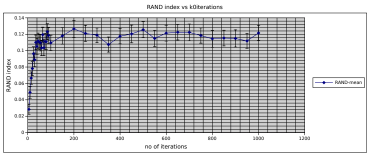

Fig. 1 Minimum no. of iterations of LDA topic modeling for RAND

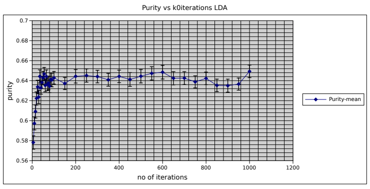

Fig. 2 Minimum no. of iterations of LDA topic modeling for Purity

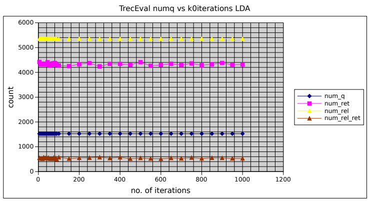

Fig. 3 Minimum no. of iterations of LDA topic modeling for numq measures

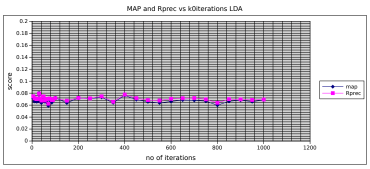

Fig. 4 Minimum no. of iterations of LDA topic modeling for MAP and Rprec

Table 3 Minimum no. of iterations

Algorithm	min. iterations for RAND	purity	numq measures	MAP	Rprec	chosen iteration
LDA	> 80	> 40	NE	NE	NE	100
Unigram	NE	NE	NE	NE	NE	100

NE = No Effect

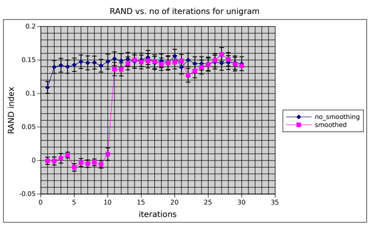

Fig. 5 Minimum no. of iterations of Unigram topic modeling for RAND

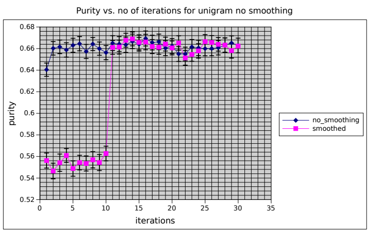

Fig. 6 Minimum no. of iterations of Unigram topic modeling for Purity

measures from trec_eval tool and precision measures. The plots of this study for LDA algorithm are shown in figures 3 to 6 and for Unigram topic model algorithm are shown in figures 7 to 11.

7.3 Hyper parameter learning

LDA algorithm has two hyperparameters, α and β. We have to learn the optimum value for both of these parameters for our particular corpus using the evaluation framework. For Unigram topic model we have to learn only for β. To get the optimum values of these variables we run the algorithm over a

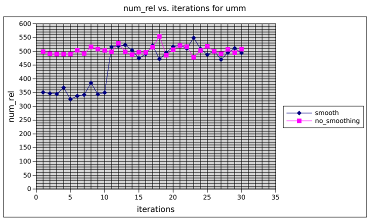

Fig. 7 Minimum no. of iterations of Unigram topic modeling for numq measures

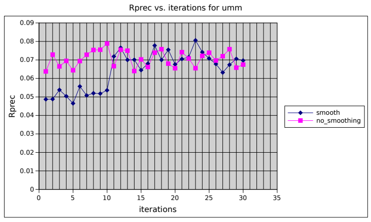

Fig. 8 Minimum no. of iterations of Unigram topic modeling for Rprec

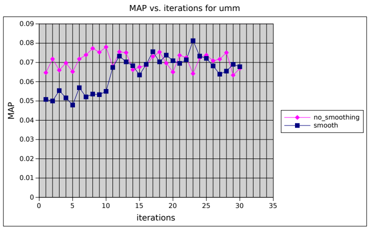

Fig. 9 Minimum no. of iterations of Unigram topic modeling for MAP

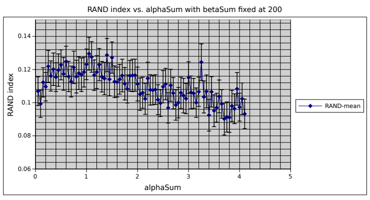

Fig. 10 Learning optimum αSum of LDA for RAND with βSum, iterations fixed

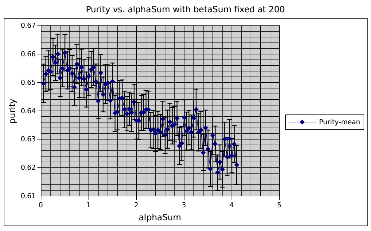

Fig. 11 Learning optimum αSum of LDA for Purity with βSum, iterations fixed

sufficiently large range of a parameter while keeping rest at a constant value. Then we measure various evaluation metrics and find out a narrow range of parameter values for which we are getting sufficiently good measures.

Optimum αSum and βSum for LDA Figures from 9 to 12 plots result obtained from the experiments to learn optimum αSum and βSum for LDA algorithm. From these charts we found optimum range for αSum to be 0.8 to 1.2 and optimum range for βSum to be 100 to 300. For subsequent experiments with LDA algorithm, we chose αSum and βSum values to be 1.0 and 260.

Optimum βSum for Unigram topic model As we have figued out from the iterations experiments, adding smoothing factor to Unigram model did not affect any measures when compared to it's no-smoothing counterpart. Hence, from here onwards we will only use no-smoothing version of the Unigram

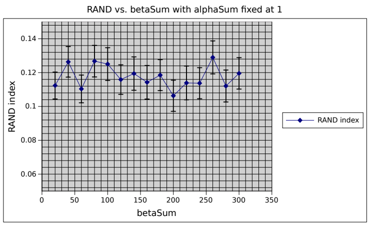

Fig. 12 Learning optimum βSum of LDA for RAND with αSum, iterations fixed

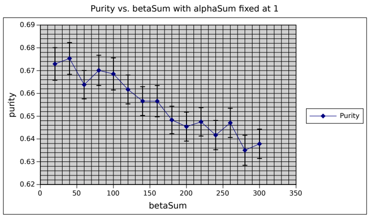

Fig. 13 Learning optimum βSum of LDA for Purity with αSum, iterations fixed

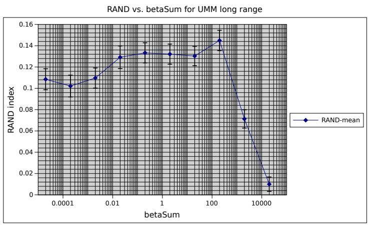

Fig. 14 Learning optimum βSum of UMM for RAND with 100 iterations (long range)

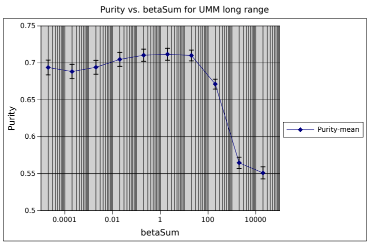

Fig. 15 Learning optimum βSum of UMM for Purity with 100 iterations (long range)

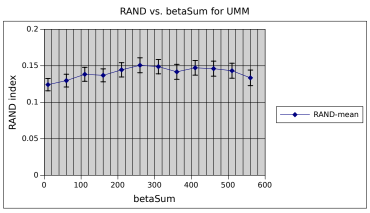

Fig. 16 Learning optimum βSum of UMM for RAND with 100 iterations (short range)

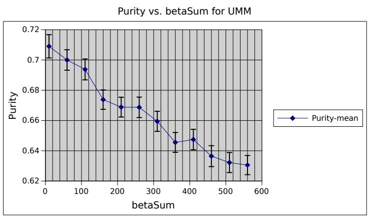

Fig. 17 Learning optimum βSum of UMM for Purity with 100 iterations (short range)

model. Figure to presents the RAND index and purity plots for range of βSum values for Unigram model. Based on these plots, we have chosen $\beta Sum = 260$ as the optimum value for subsequent experiments with Unigram model.

7.4 KMeans clustering results

plots go here

7.5 Comparison of algorithms with varying k

After we have decided the optimum values of hyperparameters for all the algorithms, we compare the measures from different algorithms by varying

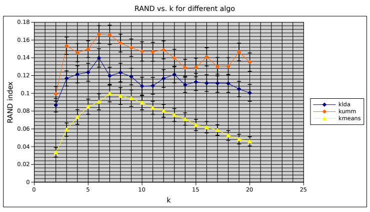

Fig. 18 Comparison of RAND index vs. k

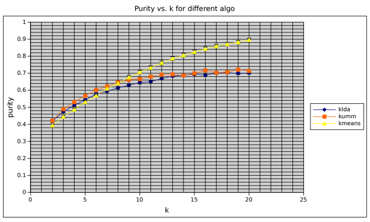

Fig. 19 Comparison of purity vs. k

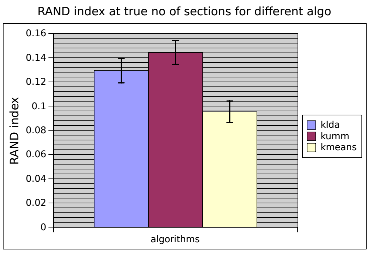

Fig. 20 Comparison of RAND index at true no. of sections

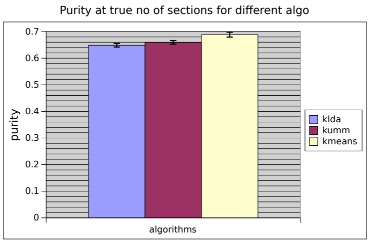

Fig. 21 Comparison of Purity at true no. of sections

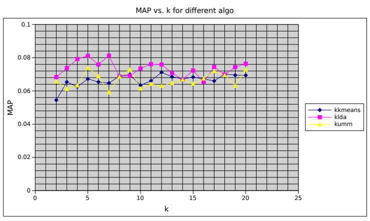

Fig. 22 Comparison of MAP vs. k

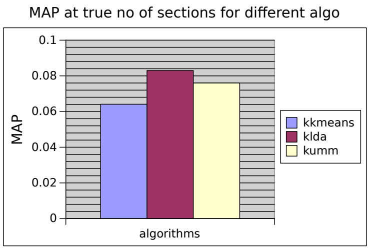

Fig. 23 Comparison of MAP at true no. of sections

k and also at k =true no. of sections. This will give us an idea about the effectiveness of each algorithm for the task.

Figure 7.5 to 7.5 presents the comparison of 3 different algorithms based on clustering measures. Figure 7.5 to 7.5 presents the same comparison based on treceval measures.

7.6 Comparison of results with variations of algo

plots go here

7.7 Difficulty analysis of the corpus

plots go here

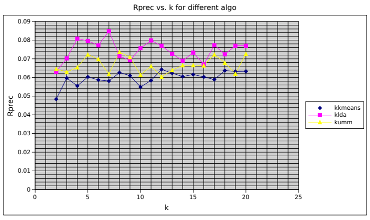

Fig. 24 Comparison of Rprec vs. k

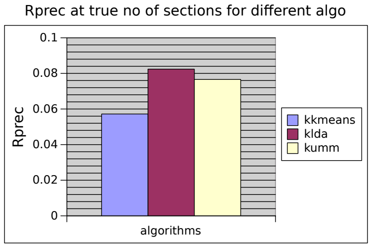

Fig. 25 Comparison of Rprec at true no. of sections

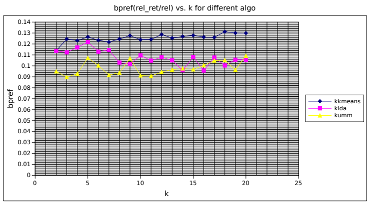

Fig. 26 Comparison of bpref vs. k

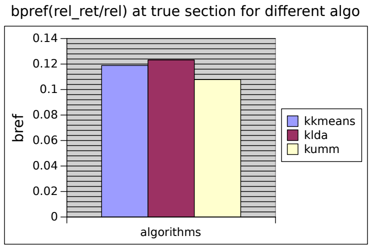

Fig. 27 Comparison of bpref at true no. of sections

8 Analysis of the results

In previous section we have presented results from three different implementation of clustering and assignment models which are LDA topic modeling, KMeans clustering and Unigram mixture model. Based on these results we can gain following insights in context of our specific experimental setup.

From RAND index comparisons in Figure 7.5 and 7.5 it becomes evident that in terms of cluster similarity with that from the ground truth, Unigram model performs better than the rest. Although the purity comparison shows that KMeans algorithm constantly achieves high purity for high k value but we have to remember that it only means that the clusters are more specific or pure and does not signify anything regarding the overall quality of the clustering.

9 Future scope

future scope goes here.

10 Conclusion

conclude the paper.

References

1. Author, Article title, Journal, Volume, page numbers (year)
2. Author, Book title, page numbers. Publisher, place (year)