COMPGI15: Information Retrieval and Data Mining - Part A & B -

Sergiu Tripon
Department of Computer Science
University College London
Gower Street, London, WC1E
`sergiu.tripon.15@ucl.ac.uk`

Part A: Text Mining

Assignment with Terrier

Root path: *sergiu_tripon_irdm1/part-a/assignment-with-terrier/*

Prior to being informed of a new version of the assignment which did not require Terrier, I started the Assignment with Terrier, and completed Questions 1 and 2. Below, you can find instructions on where the scripts are and how to run them.

For Question 1 and 2 of the Assignment with Terrier, I created 2 scripts:

- *terrier-scripts/index_clueweb.sh*
- *terrier-scripts/bm25_terrier.sh*

The two scripts above reside in the terrier-scripts folder which is located within the root path specified above:

sergiu_tripon_irdm1/part-a/assignment-with-terrier/terrier-scripts/

The structure of this folder is as follows:

- ***index_clueweb.sh*** - runs index
- ***bm25_terrier.sh*** - runs batch retrieval
- ***terrier.properties*** - resulting terrier configuration file after running the two scripts above (*index_clueweb.sh* and *bm25_terrier.sh*)
- ***BM25b0.75_0.res*** - resulting bm25 results file after running the *bm25_terrier.sh* script above

The scripts have to be run in the order they appear above. When running the scripts the following folder structure is required:

- terrier-core-4.1 folder
- clueweb folder
- topics folder
- qrels folder
- **terrier-scripts folder**
  - *index_clueweb.sh*
  - *bm25_terrier.sh*

To run the two scripts, the following commands need to be run:

```
$ cd terrier-scripts
$ sh index_clueweb.sh
$ sh bm25_terrier.sh
```

The scripts contain commands that perform a number of actions:

- ***index_clueweb.sh***
  - deletes collection.spec if it exists
  - deletes terrier.properties if it exists
  - deletes index folder if it exists
  - runs trec_setup.sh PATH_TO_DOCUMENT_COLLECTION
  - writes properties to terrier.properties file
  - runs trec_terrier.sh -i -j
- ***bm25_terrier.sh***
  - deletes results folder if it exists
  - writes properties to terrier.properties file
  - runs trec_terrier.sh -r
  - removes path and '.txt' extension from every line in the results file

Please note that the terrier-scripts folder also contains a copy of the *terrier.properties* and *BM25b0.75_0.res* file after the two scripts (*index_clueweb.sh* and *bm25_terrier.sh*) were run.

Assignment without Terrier

Root path: *sergiu_tripon_irdm1/part-a/assignment-without-terrier/*

The following files reside at the root path specified above:

- **input folder**
  - *BM25b0.75_0.res*
  - *document_term_vectors.dat*
  - *query_term_vectors.dat*
  - *qrels.adhoc.txt*
  - *qrels.ndeval.txt*
  - *trec2013-topics.txt*
  - *trec2013-topics.xml*
- **output folder**
  - **final folder** - folder where each final and complete text file is stored, copied and pasted from the temp folder
    - **question-1 folder**
      - *bm25_b_0.75.txt*
    - **question-2 folder**
      - *bm25_ndcg.txt*
      - *bm25_ndcg_alternative_method.txt*
    - **question-3 folder**
      - *mmr_lambda_0.25.txt*
      - *mmr_lambda_0.50.txt*
    - **question-4 folder**
      - *portfolio_b_-4.txt*
      - *portfolio_b_4.txt*
    - **question-5 folder**

        · *mmr_lambda_0.25_ndcg.txt*
        · *mmr_lambda_0.50_ndcg.txt*
        · *portfolio_b_-4_ndcg.txt*
        · *portfolio_b_4_ndcg.txt*
    - **temp folder** - folder where each script dumps its output initially, and when output is deemed final and complete, it is moved to its corresponding question folder at path:
        * *output/final/*

- ***BM25Model.py*** - Question 1 implementation. Dumps output containing 100 results per query at path:
    - *output/temp/bm25_b_0.75.txt*
- ***NDCG.py*** - Question 2 Implementation. Reads BM25 result file provided with the assignment at path:
    - *input/BM25b0.75_0.res*

  Dumps output at path:
    - *output/temp/bm25_ndcg.txt*
- ***MMRScoring.py*** - Question 3 Implementation. Reads BM25 result file generated by BM25 in Question 1 at path:
    - *output/final/question-1/bm25_b_0.75.txt*

  Dumps output at path:
    - *output/temp/mmr_lambda_0.25.txt* or
    - *output/temp/mmr_lambda_0.50.txt*
- ***PortfolioScoring.py*** - Question 4 Implementation. Reads BM25 result file generated by BM25 in Question 1 at path:
    - *output/final/question-1/bm25_b_0.75.txt*

  Dumps output at path:
    - *output/temp/portfolio_b_-4.txt* and
    - *output/temp/portfolio_b_4.txt*
- ***alpha-NDCG.py*** - Question 5 Implementation. Reads MMR or Portfolio re-ordered BM25 result file generated in Question 3 and 4 at path:
    - *output/final/question-3/mmr_lambda_0.25.txt* or
    - *output/final/question-3/mmr_lambda_0.50.txt* or
    - *output/final/question-4/portfolio_b_-4.txt* or
    - *output/final/question-4/portfolio_b_4.txt*

  Dumps output at path:
    - *output/temp/mmr_lambda_0.25_ndcg.txt* or
    - *output/temp/mmr_lambda_0.50_ndcg.txt* or
    - *output/temp/portfolio_b_-4_ndcg.txt* or
    - *output/temp/portfolio_b_4_ndcg.txt*

Implementation Workflow

This subsection describes the implementation workflow I followed throughout the assignment. Each script dumps its output at the following path:

- *output/temp/*

When I have completed the implementation of a specific question and have generated a final and complete output as required, I move the specific file(s) to the following path:

- *output/final/*

The final folder consists of a folder for each question in the format ”question-question number”. Each final and complete output file is copied into its corresponding question folder.

This means you can, if you wish, run the scripts personally and they will not be overwriting any of the final and complete text files submitted for marking.

The final and complete text files required are submitted for marking at path:

output/final/question-1 or 2 or 3 or 4 or 5/

Overall Observations/Assumptions

Duplicates in *document_term_vectors.dat* file

The *document_term_vectors.dat* consists of 4848 lines representing 4848 documents. I assumed that the document ids found on each of the 4848 lines are unique, but it turns out they aren’t. I believe this is wrong, as document ids are intended to identify a unique document within a document collection. In Python, loading the document ids from each line into a list and getting the length of the list will result in 4848. A list allows duplicates. If the list is turned into a set, which does not allow duplicates, and getting the length of the set will result in 4745. This means that there are 103 duplicate document ids in the *document_term_vectors.dat* file. An example of a duplicate file is: *clueweb12-0304wb-36-19053*. When loading the *document_term_vectors.dat* file, I created a list to hold unique document ids. For every line, before doing anything else, I check whether the document id on that line is in the unique document ids list. If it isn’t, the line is processed and loaded. If it is, the line is ignored.

Missing results for query id 219 and 241 in *BM25b0.75_0.res* file

We were provided with a bm25 results file, *BM25b0.75_0.res*, to be used in Question 2. This file doesn’t contain bm25 results for query ids 219 and 241. Because of this, when working on Question 2: NDCG, I had to do some extra checking in order to avoid not considering 50 documents for query ids before 219 and query ids after 241. This is explained further in the form of a comment within the Python script.

Negative relevance scores in *qrels.adhoc.txt* file

To be used in Question 3, we were provided with the *qrels.adhoc.txt file* which consists of adhoc relevance judgements. The relevance scores of the relevance judgements are supposed to range between 0 and 4, 0 meaning non-relevant and 4 meaning key document. However, in the file provided, relevance scores range from -2 to 2. In order to ensure positive relevance scores, I turned the relevance scores into binary form. If greater than or equal to 1, relevance score is scaled to 1. If less than or equal to 0, relevance score is scaled to 0.

BM25Model.py [1]

This file represents the implementation of the BM25 model.

It reads the following files:

- *input/document_term_vectors.dat*
- *input/query_term_vectors.dat*

It creates the following file:

- *output/temp/bm25_b_0.75.txt*

To run this script, enter the following command in a terminal window:

```
  $ python BM25Model.py
```

Below you can find a summary of the BM25 implementation:

Please note that the actual Python script also contains comments throughout.

main function

- Loads *document_term_vectors.dat* file
- Loads *query_term_vectors.dat* file
- Initializes free parameters b and k1
- For every query, calls calc_bm25 function
- Once calc_bm25 function returns, it creates the *bm25_b_0.75.txt* file at path:
  - *output/temp/bm25_b_0.75.txt*

  and writes the first 100 results to the file in standard TREC format.

calc_bm25 function

- for every document
- for every query term id
- gets fqid (the query term id's frequency in the document)
- gets nqi (number of documents containing the query term id)
- uses nqi to calcualte idfqi (inverse document frequency weight of the query term id)
- calculates fraction part of the bm25 score equation
- calculates bm25 score for the current document and current query term id by multiplying idfqi by the fraction part
- sums the obtained bm25 score with existing overall document score: this may be 0 initially or the bm25 score of the previous query term id of the query
- once the query term id for loop completes, it adds the query id, document id and document score to the data list as a Result object
- once the document for loop completes, it sorts the data list by the document score and then returns it back to the main function

```
# excerpt from calc_bm25 function

# get nqi
nqi = doc_freq.get(int(query_term_id))
# get fqid
fqid = doc_vec.get(query_term_id, 0)

# calculate idfqi
idfqi = log2((N - nqi + 0.5) / (nqi + 0.5))
# calculate fraction part
fraction = (fqid * (k1 + 1)) / (fqid + k1 * (1 - b + b * doc_len /
    avg_doc_len))

# calculate bm25 score for current document and current query term qi
bm25_score = idfqi * fraction

# add current query term qi's bm25 score to overall document score
doc_score += bm25_score
```

NDCG.py [2]

This file represents the implementation of NDCG@K.

It takes the following files as input:

- *input/BM25b0.75_0.res* - results file provided with the assignment
- *input/qrels.adhoc.txt*

It outputs the following file:

- *output/temp/bm25_ndcg.txt*

To run this script, enter the following command in a terminal window:

```
$ python NDCG.py
```

Observation

In the NDCG implementation, to calculate DCG and IDCG, I used the first equation on the Wikipedia [2] article provided. However, after doing some further research on the internet, I came across a StackOverflow post [3] which further points to a Microsoft research paper [4]. The equation in this paper differs slightly compared to the one on Wikipedia. I decided to implement both in order to see if the results would turn out to be different. Indeed, I discovered that the results were different as shown below:

bm25

K	NDCG@K
1	0.333
5	0.680
10	0.695
20	0.676
30	0.668
40	0.670
50	0.674

Method 1: Wikipedia

bm25

K	NDCG@K
1	0.333
5	0.555
10	0.589
20	0.615
30	0.631
40	0.644
50	0.654

Method 2: Microsoft research paper

Considering this, I am not sure which equation is correct or whether an equation is better suited for a particular situation. Therefore, I decided to use the equation as it appears on Wikipedia as the default. Therefore, when the script is run, the Wikipedia equation will be calculated. However, I also wanted you to consider the results of the alternative method.

In the submission of the final and complete text files required, I have also included the NDCG output produced when using method 2, the alternative method. It is located at the following path:

output/final/question-2/bm25_ndcg_alternative_method.txt

Below you can find a summary of the NDCG implementation:

Please note that the actual Python script also contains comments throughout.

main function

- Loads *BM25b0.75_0.res* file
- Loads *qrels.adhoc.txt* file
- For every query id
- for k in (1, 5, 10, 20, 30, 40, 50)
- calls calc_ndcg function
- Once calc_ndcg function returns, it creates the *bm25_ndcg.txt* file at path:
  - *output/temp/bm25_ndcg.txt*

  and writes ndcg@k in (1, 5, 10, 20, 30, 40, 50) to the file in the requested format.

calc_ndcg function

- for i in range(start, end) - on first interation, for example, start is 0 and end is either 1, 5, 10, 20, 30, 40 or 50
- gets document's relevance score

- turns relevance scores into binary - if x <= 0 then x is 0, if x >= 1 then x is 1
- creates a relevance scores and a sorted relevance scores list
- for i in range(2, k)
- calculates and sums dcg and idcg fraction
- calculates dcg and idcg using the first relevance score from both the relevance and sorted relevance scores lists
- calculates and returns ndcg

```python
# calc_ndcg function

# calculate ndcg
def calc_ndcg(results, doc_qrel, k, start, end):

   # get document relevance scores for document ids between start and end
   rels = [doc_qrel.get(results[i]) if doc_qrel.get(results[i]) is not
       None else 0 for i in range(start, end)]

   # if relevance score is greater than or equal to 1, rescale to 1,
       else 0 (binary)
   rels = [1 if x >= 1 else 0 for x in rels]

   # sort relevance scores in descending order and assign to sorted
       relevance scores
   sorted_rels = sorted(rels, reverse=True)

   # assign first relevance score to relevance score 1
   rel1 = rels[0]
   # assign first sorted relevance score to sorted relevance score 1
   sorted_rel1 = sorted_rels[0]

   # calculate dcg fraction
   # method 1 - wikipedia
   dcg_fraction = sum([(rels[i] / log2(i)) for i in range(2, k)])
   # method 2 - microsoft research paper
   # dcg_fraction = sum([(rels[i] / log2(i + 1)) for i in range(1, k)])
   # calculate idcg fraction
   # method 1 - wikipedia
   idcg_fraction = sum([(sorted_rels[i] / log2(i)) for i in range(2, k)])
   # method 2 - microsoft research paper
   # idcg_fraction = sum([(sorted_rels[i] / log2(i + 1)) for i in
       range(1, k)])

   # calculate dcg
   dcg = rel1 + dcg_fraction
   # calculate idcg
   idcg = sorted_rel1 + idcg_fraction

   # set ndcg to 0.0
   ndcg = 0.0

   # if dcg and idcg are not equal to 0 or 0.0
   if dcg and idcg != 0 or 0.0:
      # calculate ndcg
      ndcg = dcg / idcg

   # return ndcg
   return ndcg
```

MMRScoring.py [5]

This file represents the implementation of MMR Scoring.

It takes the following files as input:

- *input/document_term_vectors.dat*
- *output/final/question-1/bm25_b_0.75.txt* - file created in Question 1 containing 100 documents per query.

It outputs the following files:

- *output/temp/mmr_lambda_0.25.txt*
- *output/temp/mmr_lambda_0.50.txt*

To run this script, enter the following commands in a terminal window:

```
  $ python MMRScoring.py --lambda 0.25
  $ python MMRScoring.py --lambda 0.50
```

Below you can find a summary of the MMR implementation:

Please note that the actual Python script also contains comments throughout.

main function

- Loads *document_term_vectors.dat* file
- Loads *bm25_b_0.75.txt* file
- Initializes start and end to 0 and 100 respectively
- initializes rq - uses start and end to extract 100 documents from a list containing all 5000 documents
- calls calc_mmr function - it doesn't return anything as the writing to file is done within it
- increments start and end by 100. e.g. start becomes 100 and end becomes 200 and so on

calc_mmr function

- adds first element of rq to dq
- removes first element of dq from rq
- writes the top document result to file in standard TREC format
- for i in range(i, length of rq (initially, 99))
- for rq document id in rq
- for dq document id in dq
- calculates f1 score using the document's normalized bm25 score, normalization used: top bm25 score / current document's bm25 score
- calls calc_sim function which will return the cosine similarity between the rq and dq document id (f2 score)
- calculates mmr
- if current mmr score is larger than highest mmr score recorded
- updates highest mmr score
- records document id that has highest mmr score
- adds document with highest mmr score to dq
- writes the document result to file in standard TREC format
- removes document id with highest mmr score from rq

calc_sim function

- gets the intersection between query term ids of the rq and dq document

- calculates the tf-idf of the rq and dq document
- adds tf-idf result of rq document to rq tf-idf memoize
- adds tf-idf result of dq document to dq tf-idf memoize
- calculates dot product between the rq and dq tf-idf vectors
- calculates sum of rq and dq document term frequencies
- adds sum result of rq document to rq sum memoize
- adds sum result of dq document to dq sum memoize
- memoize is a dictionary which is located outside of the function and caches various results for a specific document id, this prevents computing the same calculation twice or more times
- calculates product of rq and dq sum
- calculates and returns cosine

```
# calc_mmr function

# calculates mmr and ranks 100 given documents based on mmr
def calc_mmr(query_id, qid_did_score, rq, doc_score, doc_vec, idf,
    lambda_weight):

   # assign first token of document score to max score
   max_score = max(doc_score)
   # assign first token of rq to dq
   dq = [rq[0]]
   # remove first token of dq from rq
   rq.remove(dq[0])

   # open file
   with open('output/temp/mmr_lambda_{:.2f}.txt'.format(lambda_weight),
       mode='a') as results_file:
      # write results in standard TREC format
      results_file.write('{} Q0 {} 0 {} bm25_b_0.75\n'.format(query_id,
          dq[0], max_score, lambda_weight))

   # create dictionary of query id document id score and assign it to
       query id document id score
   qid_did_score = dict(qid_did_score)
   # create dictionary of document vector and assign it to document
       vector
   doc_vec = dict(doc_vec)

   # assign length of rq to rq length
   rq_len = len(rq)
   # print progress
   print('query id {} - scored 0 out of {} documents'.format(query_id,
       rq_len + 1))
   # variable to hold document rank set to 1
   doc_rank = 1
   # variable to hold print progress set to 1
   print_progress = 1
   # loop from 0 to rq length
   for i in range(0, rq_len):
      # variable to hold highest mmr set to null
      high_mmr = None
      # variable to hold highest mmr document set to null
      high_mmr_doc_id = None
      # open file
      with
          open('output/temp/mmr_lambda_{:.2f}.txt'.format(lambda_weight),
          mode='a') as results_file:
         # for every rq document id
```

```
        for rq_doc_id in rq:
            # for every dq document id
            for dq_doc_id in dq:
                # assign rq document vector to rq document vector
                rq_doc_vec = doc_vec.get(rq_doc_id)
                # assign dq document vector to dq document vector
                dq_doc_vec = doc_vec.get(dq_doc_id)
                # concatenate query id and rq document id and assign to
                    query id document id
                qid_did = ' '.join([str(query_id), rq_doc_id])
                # assign query id document id score to rq document score
                rq_doc_score = qid_did_score.get(qid_did)
                # calculate f1
                f1 = rq_doc_score / max_score
                # calculate f2
                f2 = calc_sim(rq_doc_id, dq_doc_id,
                    OrderedDict(rq_doc_vec), OrderedDict(dq_doc_vec), idf)
                # calculate mmr
                mmr = lambda_weight * f1 - (1 - lambda_weight) * f2
                # if high mmr is null:
                if high_mmr is None:
                    # assign mmr to high mmr
                    high_mmr = mmr
                    # assign rq document id to high mmr document id
                    high_mmr_doc_id = rq_doc_id
                # if mmr is greater than high mmr:
                if mmr > high_mmr:
                    # assign mmr to high mmr
                    high_mmr = mmr
                    # assign rq document id to high mmr document id
                    high_mmr_doc_id = rq_doc_id

        # add high mmr document id to dq
        dq += [high_mmr_doc_id]
        # write results in standard TREC format
        results_file.write('{} Q0 {} {} {}
            mmr_l_{:.2f}\n'.format(query_id, high_mmr_doc_id, doc_rank,
            high_mmr,
                                                  lambda_weight))
        # increment document rank
        doc_rank += 1
        # print progress
        print('query id {} - scored {} out of {}
            documents'.format(query_id, print_progress, rq_len + 1))
        # increment print progress
        print_progress += 1
        # remove high mmr document id from rq
        rq.remove(high_mmr_doc_id)

    # print progress
    print('\nSaved results to file at path:
        \'{}\'\n'.format(results_file.name))

    # return
    return
```

```
# calc_sim function

# dictionary to hold rq tf-idf memoize
rq_tf_idf_memoize = {}
# dictionary to hold dq tf-idf memoize
dq_tf_idf_memoize = {}
```

```
# dictionary to hold rq sum memoize
rq_sum_memoize = {}
# dictionary to hold dq sum memoize
dq_sum_memoize = {}


# calculates cosine similarity between two given documents
def calc_sim(rq_doc_id, dq_doc_id, rq_doc_vec, dq_doc_vec, idf):

    # assign intersection of rq document vector and dq document vector to
        intersection
    intersection = rq_doc_vec.keys() & dq_doc_vec.keys()
    # sort intersection in ascending order
    intersection = sorted(intersection)

    # if rq document id is not in rq tf-idf memoize or dq document id is
        not in dq tf-idf memoize
    if rq_doc_id not in rq_tf_idf_memoize or dq_doc_id not in
        dq_tf_idf_memoize:
        # calculate rq tf-idf and dq tf-idf
        rq_tf_idf_memoize[rq_doc_id] = [idf.get(x) * rq_doc_vec.get(x) for
            x in intersection]
        dq_tf_idf_memoize[dq_doc_id] = [idf.get(x) * dq_doc_vec.get(x) for
            x in intersection]

    # get rq and dq tf-idf vector
    rq_tf_idf_vector = rq_tf_idf_memoize.get(rq_doc_id)
    dq_tf_idf_vector = dq_tf_idf_memoize.get(dq_doc_id)
    # calculate dot product of rq and dq tf-idf vector
    rq_dq_dot_product = sum([x * y for x, y in zip(rq_tf_idf_vector,
        dq_tf_idf_vector)])

    # if rq document id is not in rq sum memoize or dq document id not in
        dq sum memoize
    if rq_doc_id not in rq_sum_memoize or dq_doc_id not in dq_sum_memoize:
        # assign rq and dq document vector values to rq and dq document
            term frequencies
        rq_doc_term_freq = rq_doc_vec.values()
        dq_doc_term_freq = dq_doc_vec.values()
        # calculate sum of rq and dq document term frequencies
        rq_sum_memoize[rq_doc_id] = sqrt(sum([(x ** 2) for x in
            rq_doc_term_freq]))
        dq_sum_memoize[dq_doc_id] = sqrt(sum([(x ** 2) for x in
            dq_doc_term_freq]))

    # calculate product of rq and dq sum
    rq_dq_product = rq_sum_memoize.get(rq_doc_id) *
        dq_sum_memoize.get(dq_doc_id)

    # calculate cosine
    cos = float(rq_dq_dot_product) / rq_dq_product

    # return cosine
    return cos
```

PortfolioScoring.py [6] [7] [8]

This file represents the implementation of Portfolio Scoring.

It takes the following files as input:

- *input/document_term_vectors.dat*

- *output/final/question-1/bm25_b_0.75.txt* - file created in Question 1 containing 100 documents per query.

It outputs the following files:

- *output/temp/portfolio_b_-4.txt*
- *output/temp/portfolio_b_4.txt*

To run this script, enter the following commands in a terminal window:

```
$ python PortfolioScoring.py --b_param -4
$ python PortfolioScoring.py --b_param 4
```

Below you can find a summary of the Portfolio implementation:

Please note that the actual Python script also contains comments throughout.

main function

- Loads *document_term_vectors.dat* file
- Loads *bm25_b_0.75.txt* file
- Initializes start and end to 0 and 100 respectively
- initializes rq - uses start and end to extract 100 documents from a list containing all 5000 documents
- calls calc_mva function - it doesn't return anything as the writing to file is done within it
- increments start and end by 100. e.g. start becomes 100 and end becomes 200 and so on

calc_mva function

- adds first element of rq to dq
- removes first element of dq from rq
- writes the top document result to file in standard TREC format
- for i in range(i, length of rq (initially, 99))
- for rq document id in rq
- for dq document id in dq
- normalizes bm25 score, normalization used: top bm25 score / current document's bm25 score
- calls calc_pxy function which will return the pearson correlation coefficient between the rq and dq document id
- calculates mva
- if current mva score is larger than highest mva score recorded
- updates highest mva score
- records document id that has highest mva score
- adds document with highest mva score to dq
- writes the document result to file in standard TREC format
- removes document id with highest mva score from rq

calc_pxy function

- gets the intersection between query term ids of the rq and dq document
- gets term frequency of rq document
- gets term frequency of dq document

- adds rq and dq document term frequency to rq dq term frequency memoize
- memoize is a dictionary which exists outside of the function and caches various results for two specific document ids, this prevents computing the same action twice or more times
- calculates covariance between the rq and dq term frequency vectors
- calculates standard deviation between the rq and dq term frequency vectors
- calculates and returns pearson's correlation coefficient

```
# calc_mva function

# calculates mva and ranks 100 given documents at a time
def calc_mva(query_id, qid_did_score, rq, doc_score, doc_vec, b):

   # assign first token of document score to max score
   max_score = max(doc_score)
   # assign first token of rq to dq
   dq = [rq[0]]
   # remove first token of dq from rq
   rq.remove(dq[0])

   # open file
   with open('output/temp/portfolio_b_{}.txt'.format(b), mode='a') as
       results_file:
      # write results in standard TREC format
      results_file.write('{} Q0 {} 0 {} bm25_b_0.75\n'.format(query_id,
          dq[0], max_score, b))

   # create dictionary of query id document id score and assign it to
       query id document id score
   qid_did_score = dict(qid_did_score)
   # create dictionary of document vector and assign it to document
       vector
   doc_vec = dict(doc_vec)

   # assign length of rq to rq length
   rq_len = len(rq)
   # print progress
   print('query id {} - scored 0 out of {} documents'.format(query_id,
       rq_len + 1))
   # variable to hold document rank set to 1
   doc_rank = 1
   # variable to hold print progress set to 1
   print_progress = 1
   # loop from 0 to rq length
   for i in range(0, rq_len):
      # variable to hold highest mva set to null
      high_mva = None
      # variable to hold highest mva document set to null
      high_mva_doc_id = None
      # open file
      with open('output/temp/portfolio_b_{}.txt'.format(b), mode='a') as
          results_file:
         # for every rq document id
         for rq_doc_id in rq:
            # for every dq document id
            for dq_doc_id in dq:
               # assign rq document vector to rq document vector
               rq_doc_vec = doc_vec.get(rq_doc_id)
               # assign dq document vector to dq document vector
               dq_doc_vec = doc_vec.get(dq_doc_id)
               # concatenate query id and rq document id and assign to
                   query id document id
               qid_did = ' '.join([str(query_id), rq_doc_id])
```

```
                # assign query id document id score to rq document score
                rq_doc_score = qid_did_score.get(qid_did)
                # normalize bm25 score
                norm_bm25_score = rq_doc_score / max_score
                # calculate pearson's correlation coefficient
                p_x_y = calc_pxy(rq_doc_id, dq_doc_id,
                    OrderedDict(rq_doc_vec), OrderedDict(dq_doc_vec))
                # calculate mva
                mva = norm_bm25_score - (b * i) - 2 * b * p_x_y
                # if high mva is null:
                if high_mva is None:
                   # assign mva to high mva
                   high_mva = mva
                   # assign rq document id to high mva document id
                   high_mva_doc_id = rq_doc_id
                # if mva is greater than high mva:
                if mva > high_mva:
                   # assign mva to high mva
                   high_mva = mva
                   # assign rq document id to high mva document id
                   high_mva_doc_id = rq_doc_id

         # add high mva document id to dq
         dq += [high_mva_doc_id]
         # write results in standard TREC format
         results_file.write('{} Q0 {} {} {}
             portfolio_b_{}\n'.format(query_id, high_mva_doc_id,
             doc_rank, high_mva, b))
         # increment document rank
         doc_rank += 1
         # print progress
         print('query id {} - scored {} out of {}
             documents'.format(query_id, print_progress, rq_len + 1))
         # increment print progress
         print_progress += 1
         # remove high mva document id from rq
         rq.remove(high_mva_doc_id)

   # print progress
   print('\nSaved results to file at path:
       \'{}\'\n'.format(results_file.name))

   # return
   return
```

```
# calc_pxy function

# dictionary to hold rq dq term frequency memoize
rq_dq_term_freq_memoize = {}


# calculates pearson's correlation coefficient between two given
    documents
def calc_pxy(rq_doc_id, dq_doc_id, rq_doc_vec, dq_doc_vec):

   # concatenate rq and dq document id and assign to rq dq document id
   rq_dq_doc_id = ' '.join([rq_doc_id, dq_doc_id])

   # if rq dq document id is not in rq dq term frequency memoize
   if rq_dq_doc_id not in rq_dq_term_freq_memoize:
      # assign rq and dq document vector values to rq and dq document
          term frequency
```

```
        rq_doc_term_freq = [rq_doc_vec.get(x) for x in rq_doc_vec.keys()
            if x in dq_doc_vec.keys()]
        dq_doc_term_freq = [dq_doc_vec.get(x) for x in dq_doc_vec.keys()
            if x in rq_doc_vec.keys()]
        # add rq and dq document term frequency to rq dq term frequency
            memoize
        rq_dq_term_freq_memoize[rq_dq_doc_id] = (rq_doc_term_freq,
            dq_doc_term_freq)

    # get rq and dq term frequency vector
    rq_term_freq_vector = rq_dq_term_freq_memoize.get(rq_dq_doc_id)[0]
    dq_term_freq_vector = rq_dq_term_freq_memoize.get(rq_dq_doc_id)[1]

    # calculate covariance
    covariance = cov(rq_term_freq_vector, dq_term_freq_vector)[0][1]

    # calculate standard deviation
    standard_deviation = (std(rq_term_freq_vector) *
        std(dq_term_freq_vector))

    # calculate pearson's correlation coefficient
    pxy = covariance / standard_deviation

    # return pearson's correlation coefficient
    return pxy
```

alpha-NDCG.py [9]

This file represents the implementation of alpha-NDCG@K.

It takes the following files as input:

- *input/mmr_lambda_0.25.txt* or
- *input/mmr_lambda_0.50.txt* or
- *input/portfolio_b_-4.txt* or
- *input/portfolio_b_4.txt* and
- *input/qrels.ndeval.txt*

It outputs the following files:

- *output/temp/mmr_lambda_0.25_ndcg.txt* or
- *output/temp/mmr_lambda_0.50_ndcg.txt* or
- *output/temp/mmr_b_-4_ndcg.txt* or
- *output/temp/mmr_b_4_ndcg.txt*

To run this script, enter the following commands in a terminal window:

```
$ python alpha-NDCG.py -mv mmr_0.25
$ python alpha-NDCG.py -mv mmr_0.50
$ python alpha-NDCG.py -mv portfolio_-4
$ python alpha-NDCG.py -mv portfolio_4
```

Below you can find a summary of the alpha-NDCG@K implementation:

Please note that the actual Python script also contains comments throughout.

main function

- Loads:

  - *mmr_lambda_0.25.txt* or
  - *mmr_lambda_0.50.txt* or
  - *portfolio_b_-4.txt* or
  - *portfolio_b_4.txt* file
- Loads *qrels.adhoc.txt* file
- for alpha in (0.1, 0.5, 0.9)
- For every query id
- for k in (1, 5, 10, 20, 30, 40, 50)
- calls calc_alpha_ndcg function
- Once calc_alpha_ndcg function returns, it creates:
  - *mmr_lambda_0.25_ndcg.txt* or
  - *mmr_lambda_0.50_ndcg.txt* or
  - *portfolio_b_-4_ndcg.txt* or
  - *portfolio_b_4_ndcg.txt*

  file at path:
  - *output/temp/mmr_lambda_0.25_ndcg.txt* or
  - *output/temp/mmr_lambda_0.50_ndcg* or
  - *output/temp/portfolio_b_-4_ndcg.txt* or
  - *output/temp/portfolio_b_4_ndcg.txt*

  and writes alpha-NDCG@k in (1, 5, 10, 20, 30, 40, 50) for alpha in (0.1, 0.5, 0.9) to the file in the requested format.

calc_alpha_ndcg function

- for i in range(start, end) - on first interation, for example, start is 0 and end is either 1, 5, 10, 20, 30, 40 or 50
- calculates cumulative gain using alpha
- creates a relevance scores and a sorted relevance scores list
- for i in range(1, k)
- calculates and sums dcg and idcg fraction
- calculates dcg and idcg using the first relevance score from both the relevance and sorted relevance scores lists
- calculates and returns alpha ndcg

```
# calc_alpha_ndcg function

# calculates alpha ndcg
def calc_alpha_ndcg(results, doc_int, doc_rel, k, alpha, start, end):

   # get document relevance scores for document ids between start and end
   rels = [doc_int.get(results[i]) * ((1 - alpha) **
       doc_rel.get(results[i]))
         if doc_rel.get(results[i]) is not None else 0 for i in
           range(start, end)]

   # sort relevance scores in descending order and assign to sorted
       relevance scores
   sorted_rels = sorted(rels, reverse=True)

   # assign first relevance score to relevance score 1
   rel1 = rels[0]
   # assign first sorted relevance score to sorted relevance score 1
   sorted_rel1 = sorted_rels[0]
```

```
    # calculate dcg fraction
    dcg_fraction = sum([(rels[i] / log2(i + 1)) for i in range(1, k)])
    # calculate idcg fraction
    idcg_fraction = sum([(sorted_rels[i] / log2(i + 1)) for i in range(1,
        k)])

    # calculate dcg
    dcg = rel1 + dcg_fraction
    # calculate idcg
    idcg = sorted_rel1 + idcg_fraction

    # set alpha ndcg to 0.0
    alpha_ndcg = 0.0

    # if dcg and idcg are not equal to 0 or 0.0
    if dcg and idcg != 0 or 0.0:
        # calculate alpha-ndcg
        alpha_ndcg = dcg / idcg

    # return alpha ndcg
    return alpha_ndcg
```

Part B: Distributed Computing

Path: *sergiu_tripon_irdm1/part-b/*

Task 1

(a)

Number of unique bigrams: **424462**

(b)

Table 1: Top twenty most frequent bigrams and their counts.

Bigram	Count	Bigram	Count	Bigram	Count	Bigram	Count
of the	**13037**	i will	**3470**	i am	**2603**	and i	**2003**
and the	**7034**	and he	**3020**	for the	**2231**	out of	**1961**
the lord	**7017**	shall be	**3013**	and they	**2185**	my lord	**1869**
in the	**6738**	all the	**2714**	unto the	**2163**	it is	**1833**
to the	**3799**	i have	**2666**	the king	**2062**	of his	**1720**

(c)

sum of counts of the top twenty most frequent bigrams / total number of bigrams
= cumulative frequency of the top twenty bigrams

$$73138/1578220 = 0.04634 = \mathbf{4.63\%}$$

(d)

Number of bigrams that appear only twice: **52967**

(e)

Table 2 presents the job time-based performance of clusters running with mappers and reducers argument values of 1, 5 and 10, respectively. Two aspects are immediately noticeable from the data in Table 2. Firstly, the number of files that are in the output is equivalent to the number of mappers and reducers used. Secondly, the cluster running with the least amount of mappers and reducers, 1, completed in the shortest time, 36.564 seconds. On the other hand, the cluster running with the most amount of mappers and reducers, 10, completed in the longest time, 58.707 seconds. I believe that both aspects discussed support the fact that using 1 mapper and 1 receiver brings benefits such as less files and better time-based performance. However, I also believe that the choice of the amount of mappers and receivers used relies heavily on the amount of the input data among other aspects.

Let's consider, for example, the mapper output to the reducer consists of 50 keys and 100 values per key. In order to process all 50 keys in parallel, 50 reducers are required.

Furthermore, if the mapper output to the reducer consists of 100 keys and 50 values per key instead, then 100 reducers are required in order to process all 100 keys in parallel.

This supports the fact that the choice of how many mappers and reducers are needed is dependent on the size of the data being processed. Any significant amount of data will require more than 1 reducer.

In conclusion, if the data is small, using less mappers and reducers will result in less files and will take less time to complete. This is because 1 mapper and 1 reducer, for example, are able to process the data. If the data is big, more than 1 mapper and reducer are required in order to process more of the data at the same time, in parallel. This way, the job will take less time to complete, but it will however, result in more output files. I believe that finding a suitable balance and priority between the benefits and drawbacks involved is key.

Table 2: Job performance on clusters running 1, 5 and 10 mappers and reducers, respectively.

Mappers	Reducers	Number of Output files	Performance
1	1	1	Job Finished in 36.564 seconds
5	5	5	Job Finished in 44.536 seconds
10	10	10	Job Finished in 58.707 seconds

Java code

```
//BigramCount.java

/*
 * Cloud9: A MapReduce Library for Hadoop
 *
 * Licensed under the Apache License, Version 2.0 (the "License"); you
 * may not use this file except in compliance with the License. You may
 * obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
 * implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

package edu.umd.cloud9.examples;

import java.io.IOException;
```

```java
import java.util.Iterator;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

import org.apache.log4j.Logger;

/**
 * <p>
 * Simple bigram count demo. This Hadoop Tool counts bigrams in flat
     text file, and
 * takes the following command-line arguments:
 * </p>
 *
 * <ul>
 * <li>[input-path] input path</li>
 * <li>[output-path] output path</li>
 * <li>[num-mappers] number of mappers</li>
 * <li>[num-reducers] number of reducers</li>
 * </ul>
 *
 * @author Jimmy Lin
 * @author Marc Sloan
 */
public class BigramCount extends Configured implements Tool {
  private static final Logger sLogger =
      Logger.getLogger(BigramCount.class);

  /**
   * Mapper: emits (token + " " + token, 1) for every bigram occurrence
   *
   */
  private static class MyMapper extends MapReduceBase implements
       Mapper<LongWritable, Text, Text, IntWritable> {

    /**
     * Store an IntWritable with a value of 1, which will be mapped
     * to each bigram found in the test
     */
    private final static IntWritable one = new IntWritable(1);
    /**
     * reuse objects to save overhead of object creation
     */
    //variable to hold bigram
    private Text bigram = new Text();

    /**
     * Mapping function. This takes the text input, converts it into a
        String which is split into
     * words, then each of the bigrams is mapped to the OutputCollector
        with a count of
     * one.
     *
     * @param key Input key, not used in this example
     * @param value A line of input Text taken from the data
     * @param output Map from each bigram (Text) to its count
        (IntWritable)
     */
    public void map(LongWritable key, Text value, OutputCollector<Text,
        IntWritable> output,
        Reporter reporter) throws IOException {

      //Convert input word into String and tokenize to find words
      String line = ((Text) value).toString();
      StringTokenizer itr = new StringTokenizer(line);

      //variable to hold previous word
      Text previous_word = new Text();

      //variable to hold current word
      Text current_word = new Text();

      //For each bigram, map it to a count of one. Duplicate bigrams
         will be counted
      //in the reduce phase.
      while (itr.hasMoreTokens()) {

        //update the current word as the next word
        current_word.set(itr.nextToken());

        //if there is a previous word before the current word
        if (previous_word != null) {

          //form bigram of previous word and current word
          bigram.set(previous_word + " " + current_word);

          //output the bigram
          output.collect(bigram, one);
        }

        //update the previous word as the current word
        previous_word.set(current_word);
      }
    }
  }

  /**
   * Reducer: sums up all the counts
   *
   */
  private static class MyReducer extends MapReduceBase implements
      Reducer<Text, IntWritable, Text, IntWritable> {

    /**
     * Stores the sum of counts for a bigram
     */
    private final static IntWritable SumValue = new IntWritable();
    /**
     * @param key The Text bigram
     * @param values An iterator over the values associated with this
        word
     * @param output Map from each bigram (Text) to its count
        (IntWritable)
     * @param reporter Used to report progress
     */
    public void reduce(Text key, Iterator<IntWritable> values,
        OutputCollector<Text, IntWritable> output, Reporter reporter)
            throws IOException {

      //sum up values
      int sum = 0;
      while (values.hasNext()) {

        sum += values.next().get();
      }

      SumValue.set(sum);
      output.collect(key, SumValue);
    }
  }

  /**
   * Creates an instance of this tool.
   */
  public BigramCount() {
  }

  /**
   * Prints argument options
   * @return
   */
  private static int printUsage() {
    System.out.println("usage: [input-path] [output-path] [num-mappers]
        [num-reducers]");
    ToolRunner.printGenericCommandUsage(System.out);
    return -1;
  }

  /**
   * Runs this tool.
   */
  public int run(String[] args) throws Exception {
    if (args.length != 4) {
      printUsage();
      return -1;
    }

    String inputPath = args[0];
    String outputPath = args[1];

    int mapTasks = Integer.parseInt(args[2]);
    int reduceTasks = Integer.parseInt(args[3]);

    sLogger.info("Tool: BigramCount");
    sLogger.info(" - input path: " + inputPath);
    sLogger.info(" - output path: " + outputPath);
    sLogger.info(" - number of mappers: " + mapTasks);
    sLogger.info(" - number of reducers: " + reduceTasks);

    JobConf conf = new JobConf(BigramCount.class);
    conf.setJobName("BigramCount");
```

```java
    conf.setNumMapTasks(mapTasks);
    conf.setNumReduceTasks(reduceTasks);

    FileInputFormat.setInputPaths(conf, new Path(inputPath));
    FileOutputFormat.setOutputPath(conf, new Path(outputPath));
    FileOutputFormat.setCompressOutput(conf, false);

    /**
     * Note that these must match the Class arguments given in the mapper
     */
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    conf.setMapperClass(MyMapper.class);
    conf.setCombinerClass(MyReducer.class);
    conf.setReducerClass(MyReducer.class);

    // Delete the output directory if it exists already
    Path outputDir = new Path(outputPath);
    FileSystem.get(outputDir.toUri(), conf).delete(outputDir, true);

    long startTime = System.currentTimeMillis();
    JobClient.runJob(conf);
    sLogger.info("Job Finished in " + (System.currentTimeMillis() -
        startTime) / 1000.0
         + " seconds");

    return 0;
  }

  /**
   * Dispatches command-line arguments to the tool via the
   * <code>ToolRunner</code>.
   */
  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new Configuration(), new BigramCount(),
        args);
    System.exit(res);
  }
}
```

```java
//AnalyzeBigramCount.java

/*
 * Cloud9: A MapReduce Library for Hadoop
 *
 * Licensed under the Apache License, Version 2.0 (the "License"); you
 * may not use this file except in compliance with the License. You may
 * obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
 * implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

package edu.umd.cloud9.examples;

import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.StringTokenizer;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;

import edu.umd.cloud9.io.PairOfWritables;

public class AnalyzeBigramCount {
  public static void main(String[] args) {

    //hard coded path to avoid setting arguments
    String[] fixed_path = {"output/bigram-count"};
    args = fixed_path;

    if (args.length != 1) {
      System.out.println("usage: [input-path]");
      System.exit(-1);
    }

    System.out.println("input path: " + args[0]);
    //List<PairOfWritables<Text, IntWritable>> bigrams =
    //SequenceFileUtils.readDirectory(new Path(args[0]));

    List<PairOfWritables<Text, IntWritable>> bigrams;
    try {
      bigrams = readDirectory(new Path(args[0]));

      Collections.sort(bigrams, new Comparator<PairOfWritables<Text,
          IntWritable>>() {
        public int compare(PairOfWritables<Text, IntWritable> e1,
            PairOfWritables<Text, IntWritable> e2) {
          if (e2.getRightElement().compareTo(e1.getRightElement()) ==
              0) {
            return e1.getLeftElement().compareTo(e2.getLeftElement());
          }

          return e2.getRightElement().compareTo(e1.getRightElement());
        }
      });

      int doubletons = 0;
      int sum = 0;
      for (PairOfWritables<Text, IntWritable> bigram : bigrams) {
        sum += bigram.getRightElement().get();

        if (bigram.getRightElement().get() == 2) {
          doubletons++;
        }
      }

      System.out.println("total number of unique bigrams: " +
          bigrams.size());
      System.out.println("total number of bigrams: " + sum);
      System.out.println("number of bigrams that appear only twice: " +
          doubletons);

      System.out.println("\ntwenty most frequent bigrams: ");

      int cnt = 0;
      int x = 0;
      int[] frequencies = new int[20];
      for (PairOfWritables<Text, IntWritable> bigram : bigrams) {
        System.out.println(bigram.getLeftElement() + "\t" +
            bigram.getRightElement());
        frequencies[x] = bigram.getRightElement().get();
        x++;
        cnt++;

        if (cnt >= 20) {
          break;
        }
      }

      int frequency = 0;
      for( int num : frequencies) {
        frequency = frequency + num;
      }

      double total = (double)frequency / (double)sum;

      System.out.println("Cumulative frequency of the top twenty
          bigrams: " + total);

    } catch (IOException e) {
      System.err.println("Couldn't load folder: " + args[0]);
    }
  }

  /**
   * Reads in the bigram count file
   * @param path
   * @return
   * @throws IOException
   */
  private static List<PairOfWritables<Text, IntWritable>>
      readDirectory(Path path)
      throws IOException {

    File dir = new File(path.toString());
    ArrayList<PairOfWritables<Text, IntWritable>> bigrams = new
        ArrayList<PairOfWritables<Text, IntWritable>>();
    for (File child : dir.listFiles()) {
      if (".".equals(child.getName()) || "..".equals(child.getName())) {
        continue; //Ignore the self and parent aliases.
      }
      FileInputStream bigramFile = null;

      bigramFile = new FileInputStream(child.toString());

      //Read in the file
      DataInputStream resultsStream = new DataInputStream(bigramFile);
      BufferedReader results = new BufferedReader(new
          InputStreamReader(resultsStream));

      StringTokenizer rToken;
      String rLine;
      String firstWord;
      String secondWord;
```

```
        String count;

        //iterate through every line in the file
        while ((rLine = results.readLine()) != null) {
          rToken = new StringTokenizer(rLine);
          //extract the meaningful information
          firstWord = rToken.nextToken();
          secondWord = rToken.nextToken();
          count = rToken.nextToken();

          bigrams.add(new PairOfWritables<Text, IntWritable>(new
              Text(firstWord + " " + secondWord), new
              IntWritable(Integer.parseInt(count))));
        }
        if (bigramFile != null)
          bigramFile.close();
      }

      return bigrams;
    }
}
```

Task 2

(a)

Table 3: Five most frequent words following the word *romeo* and their frequency.

Bigram	Frequency	Bigram	Frequency	Bigram	Frequency
(romeo, and)	**0.14130434**	(romeo, i)	**0.061594203**	(romeo, is)	**0.039855074**
(romeo, o)	**0.036231883**	(romeo, what)	**0.028985508**		

(b)

$$\begin{aligned} p(romeo) &= 0.0032 \\ p(is|romeo) &= 0.039855074 \\ p(the|is) &= 0.08581022 \\ p(king|the) &= 0.02218946 \\ p(romeo\ is\ the\ king) &= p(romeo) * p(is|romeo) * p(the|is) * p(king|the) \\ &= 0.0032 * 0.039855074 * 0.08581022 * 0.02218946 = 2.4283952E\text{-}7 \end{aligned}$$

Java code

```
//BigramRelativeFrequency.java

/*
 * Cloud9: A MapReduce Library for Hadoop
 *
 * Licensed under the Apache License, Version 2.0 (the "License"); you
 * may not use this file except in compliance with the License. You may
 * obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
 * implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

package edu.umd.cloud9.examples;

import java.io.IOException;

import java.util.Iterator;
import java.util.StringTokenizer;

//added PairOfStrings package
import edu.umd.cloud9.io.PairOfStrings;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
//added FloatWritable package
import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Partitioner;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

import org.apache.log4j.Logger;

/**
 * <p>
 * Simple bigram relative frequency. This Hadoop Tool counts bigram
    relative frequency in flat text file, and
 * takes the following command-line arguments:
 * </p>
 *
 * <ul>
 * <li>[input-path] input path</li>
 * <li>[output-path] output path</li>
 * <li>[num-mappers] number of mappers</li>
 * <li>[num-reducers] number of reducers</li>
 * </ul>
 *
 * @author Jimmy Lin
 * @author Marc Sloan
 */
public class BigramRelativeFrequency extends Configured implements Tool {
  private static final Logger sLogger =
      Logger.getLogger(BigramRelativeFrequency.class);

  /**
 * Mapper: emits (token + " " + token, 1) for every bigram occurrence
 *
 */

private static class MyMapper extends MapReduceBase implements
     Mapper<LongWritable, Text, PairOfStrings, FloatWritable> {

   /**
    * Store an IntWritable with a value of 1, which will be mapped
    * to each bigram found in the test
    */
   private final static FloatWritable one = new FloatWritable(1);

   /**
    * reuse objects to save overhead of object creation
    */
   //variable to hold bigram
   private final static PairOfStrings bigram = new PairOfStrings();

   /**
    * Mapping function. This takes the text input, converts it into a
       String which is split into
    * words, then each of the bigrams is mapped to the OutputCollector
       with a count of
    * one.
    *
    * @param key Input key, not used in this example
    * @param value A line of input Text taken from the data
    * @param output Map from each bigram (PairOfStrings) to its count
       (IntWritable)
    */
   public void map(LongWritable key, Text value,
       OutputCollector<PairOfStrings, FloatWritable> output,
        Reporter reporter) throws IOException {

     //Convert input word into String and tokenize to find words
     String line = ((Text) value).toString();
     StringTokenizer itr = new StringTokenizer(line);

     //variable to hold previous word
     String previous_word = null;

     //variable to hold current word
     String current_word;

     //For each bigram, map it to a count of one. Duplicate bigrams
         will be counted
     //in the reduce phase.
     while (itr.hasMoreTokens()) {

       //update the current word as the next word
       current_word = itr.nextToken();

       //if there is a previous word before the current word
       if (previous_word != null) {

         //form bigram of previous word and current word
         bigram.set(previous_word, current_word);

         //output the bigram
         output.collect(bigram, one);

         //form bigram of previous word and current word
         bigram.set(previous_word, "***");
```

```
          //output the bigram
          output.collect(bigram, one);
        }

        //update the previous word as the current word
        previous_word = current_word;
      }
    }
  }

  /**
   * Combiner: sums up all the counts
   *
   */
  private static class MyCombiner extends MapReduceBase implements
      Reducer<PairOfStrings, FloatWritable, PairOfStrings,
          FloatWritable> {

    /**
     * Stores the sum of counts for a bigram
     */
    private final static FloatWritable SumValue = new FloatWritable();

    /**
     * @param key The Text bigram
     * @param values An iterator over the values associated with this
         word
     * @param output Map from each bigram (PairOfStrings) to its sum
         (FloatWritable)
     * @param reporter Used to report progress
     */
    @Override
    public void reduce(PairOfStrings key, Iterator<FloatWritable> values,
        OutputCollector<PairOfStrings, FloatWritable> output, Reporter
            reporter) throws IOException {

      //sum up values
      int sum = 0;
      while (values.hasNext()) {

        sum += values.next().get();
      }

      SumValue.set(sum);
      output.collect(key, SumValue);
    }
  }

  /**
   * Reducer: sums up all the counts
   *
   */
  private static class MyReducer extends MapReduceBase implements
      Reducer<PairOfStrings, FloatWritable, PairOfStrings,
          FloatWritable> {

    /**
     * Stores the sum of counts for a bigram
     */
    private final static FloatWritable SumValue = new FloatWritable();
    private float each_frequency = 0.0f;

    /**
     * @param key The Text bigram
     * @param values An iterator over the values associated with this
        word
     * @param output Map from each bigram (PairOfStrings) to its sum
        (FloatWritable)
     * @param reporter Used to report progress
     */
    public void reduce(PairOfStrings key, Iterator<FloatWritable> values,
        OutputCollector<PairOfStrings, FloatWritable> output, Reporter
            reporter) throws IOException {

      //sum up values
      float sum = 0.0f;
      while (values.hasNext()) {

        sum += values.next().get();
      }

      //if the right element of the bigram equals to "***"
      //this will output the frequency of any word followed by the word
         supplied as argument
      if (key.getRightElement().equals("***")) {

        //update sum
        SumValue.set(sum);

        //output bigram and its sum
        output.collect(key, SumValue);

        //update each_frequency as the sum
        each_frequency = sum;

      //if the right element of the bigram doesn't equal to "***"
      //this will output frequencies for every bigram that contains the
         word supplied as argument
      } else {

        //set sum as the sum divided by each_frequency
        SumValue.set(sum / each_frequency);

        //output bigram and its sum
        output.collect(key, SumValue);
      }
    }
  }

  /**
   * Partitioner controls the partitioning of the keys of the
      intermediate map-outputs.
   * The key (or a subset of the key) is used to derive the partition,
      typically by a
   * hash function. The total number of partitions is the same as the
      number of reduce
   * tasks for the job. Hence this controls which of the m reduce tasks
      the intermediate
   * key (and hence the record) is sent for reduction.
   */
  protected static class MyPartitioner extends MapReduceBase implements
      Partitioner<PairOfStrings, FloatWritable> {
    @Override
    public int getPartition(PairOfStrings key, FloatWritable value, int
        numReduceTasks) {
      return (key.getLeftElement().hashCode() & Integer.MAX_VALUE) %
         numReduceTasks;
    }
  }
```

```
/**
 * Creates an instance of this tool.
 */
public BigramRelativeFrequency() {
}

/**
 * Prints argument options
 * @return
 */
private static int printUsage() {
  System.out.println("usage: [input-path] [output-path] [num-mappers]
      [num-reducers]");
  ToolRunner.printGenericCommandUsage(System.out);
  return -1;
}

/**
 * Runs this tool.
 */
public int run(String[] args) throws Exception {
  if (args.length != 4) {
    printUsage();
    return -1;
  }

  String inputPath = args[0];
  String outputPath = args[1];

  int mapTasks = Integer.parseInt(args[2]);
  int reduceTasks = Integer.parseInt(args[3]);

  sLogger.info("Tool: BigramRelativeFrequency");
  sLogger.info(" - input path: " + inputPath);
  sLogger.info(" - output path: " + outputPath);
  sLogger.info(" - number of mappers: " + mapTasks);
  sLogger.info(" - number of reducers: " + reduceTasks);

  JobConf conf = new JobConf(BigramRelativeFrequency.class);
  conf.setJobName("BigramRelativeFrequency");

  conf.setNumMapTasks(mapTasks);
  conf.setNumReduceTasks(reduceTasks);

  FileInputFormat.setInputPaths(conf, new Path(inputPath));
  FileOutputFormat.setOutputPath(conf, new Path(outputPath));
  FileOutputFormat.setCompressOutput(conf, false);

  /**
   * Note that these must match the Class arguments given in the mapper
   */
  conf.setOutputKeyClass(PairOfStrings.class);
  conf.setOutputValueClass(FloatWritable.class);

  conf.setMapperClass(MyMapper.class);
  conf.setCombinerClass(MyCombiner.class);
  conf.setReducerClass(MyReducer.class);
  conf.setPartitionerClass(MyPartitioner.class);

  //Delete the output directory if it exists already
  Path outputDir = new Path(outputPath);
  FileSystem.get(outputDir.toUri(), conf).delete(outputDir, true);

  long startTime = System.currentTimeMillis();
```

```
    JobClient.runJob(conf);
    sLogger.info("Job Finished in " + (System.currentTimeMillis() -
        startTime) / 1000.0
         + " seconds");

    return 0;
  }

  /**
   * Dispatches command-line arguments to the tool via the
   * <code>ToolRunner</code>.
   */
  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new Configuration(), new
        BigramRelativeFrequency(), args);
    System.exit(res);
  }
}
```

```
//AnalyzeRelativeBigramFrequency.java

/*
 * Cloud9: A MapReduce Library for Hadoop
 *
 * Licensed under the Apache License, Version 2.0 (the "License"); you
 * may not use this file except in compliance with the License. You may
 * obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
 * implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

package edu.umd.cloud9.examples;

import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.StringTokenizer;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.FloatWritable;

import edu.umd.cloud9.io.PairOfStrings;
import edu.umd.cloud9.io.PairOfWritables;

public class AnalyzeBigramRelativeFrequency {
  public static void main(String[] args) {

    //hard coded path to avoid setting arguments
```

```
String[] fixed_path = {"output/bigram-relative-frequency"};
args = fixed_path;

if (args.length != 1) {
  System.out.println("usage: [input-path]");
  System.exit(-1);
}

System.out.println("input path: " + args[0]);

//List<PairOfWritables<PairOfStrings, FloatWritable>> pairs =
//SequenceFileUtils.readDirectory(new Path(args[0]));

List<PairOfWritables<PairOfStrings, FloatWritable>> pairs;

try {
  pairs = readDirectory(new Path(args[0]));

  List<PairOfWritables<PairOfStrings, FloatWritable>> list1 = new
      ArrayList<PairOfWritables<PairOfStrings, FloatWritable>>();

  for (PairOfWritables<PairOfStrings, FloatWritable> p : pairs) {
    PairOfStrings bigram = p.getLeftElement();

    //add romeo to list1
    if (bigram.getLeftElement().equals("romeo")) {
      list1.add(p);
    }
  }

  //p of is given romeo
  Collections.sort(list1,
      new Comparator<PairOfWritables<PairOfStrings,
          FloatWritable>>() {
        public int compare(PairOfWritables<PairOfStrings,
            FloatWritable> e1,
             PairOfWritables<PairOfStrings, FloatWritable> e2) {
          if (e1.getRightElement().compareTo(e2.getRightElement())
              == 0) {
            return
                e1.getLeftElement().compareTo(e2.getLeftElement());
          }

          return
              e2.getRightElement().compareTo(e1.getRightElement());
        }
      });

  int i = 0;
  Boolean loop = true;
  for (PairOfWritables<PairOfStrings, FloatWritable> p : list1) {

    PairOfStrings bigram = p.getLeftElement();
    System.out.println(bigram + "\t" + p.getRightElement());
    i++;

    if (i > 5) {
      break;
    }
  }

  //variable to hold list2
  List<PairOfWritables<PairOfStrings, FloatWritable>> list2 = new
      ArrayList<PairOfWritables<PairOfStrings, FloatWritable>>();
```

```
//variable to hold list3
List<PairOfWritables<PairOfStrings, FloatWritable>> list3 = new
    ArrayList<PairOfWritables<PairOfStrings, FloatWritable>>();

//variable to hold list4
List<PairOfWritables<PairOfStrings, FloatWritable>> list4 = new
    ArrayList<PairOfWritables<PairOfStrings, FloatWritable>>();

//value provided in the assignment paper
float romeo = 0.0032f;

//holds p(is) - probability of is
float is = 0.0f;

//holds p(the) - probability of the
float the = 0.0f;

//holds p(king) - probability of king
float king = 0.0f;

for (PairOfWritables<PairOfStrings, FloatWritable> p : pairs) {
  PairOfStrings bigram = p.getLeftElement();

  //add romeo to list1
  if (bigram.getLeftElement().equals("romeo")) {
    list2.add(p);
  }

  //add is to list2
  if (bigram.getLeftElement().equals("is")) {
    list3.add(p);
  }

  //add the to list3
  if (bigram.getLeftElement().equals("the")) {
    list4.add(p);
  }
}

//p of is given romeo
Collections.sort(list2,
     new Comparator<PairOfWritables<PairOfStrings,
         FloatWritable>>() {
       public int compare(PairOfWritables<PairOfStrings,
           FloatWritable> e1,
            PairOfWritables<PairOfStrings, FloatWritable> e2) {
         if (e1.getRightElement().compareTo(e2.getRightElement())
             == 0) {
           return
               e1.getLeftElement().compareTo(e2.getLeftElement());
         }

         return
             e2.getRightElement().compareTo(e1.getRightElement());
       }
     });

i = 0;
loop = true;
for (PairOfWritables<PairOfStrings, FloatWritable> p : list2) {

  PairOfStrings bigram = p.getLeftElement();
  i++;

  //if loop is false
  if (loop == false) {

    //stop looping
    break;

  //if the word to the right of the left word is "is"
  } else if (bigram.getRightElement().equals("is")) {

    //get right element
    String right_element = p.getRightElement().toString();

    //assign value of right element to variable
    is = Float.valueOf(right_element);

    //stop the loop
    loop = false;
  }
}

//p of the given is
Collections.sort(list3,
     new Comparator<PairOfWritables<PairOfStrings,
         FloatWritable>>() {
       public int compare(PairOfWritables<PairOfStrings,
           FloatWritable> e1,
            PairOfWritables<PairOfStrings, FloatWritable> e2) {
         if (e1.getRightElement().compareTo(e2.getRightElement())
             == 0) {
           return
               e1.getLeftElement().compareTo(e2.getLeftElement());
         }

         return
             e2.getRightElement().compareTo(e1.getRightElement());
       }
     });

i = 0;
loop = true;
for (PairOfWritables<PairOfStrings, FloatWritable> p : list3) {

  PairOfStrings bigram = p.getLeftElement();
  i++;

  //if loop is false
  if (loop == false) {

    //stop looping
    break;

  //if the word to the right of the left word is "the"
  } else if (bigram.getRightElement().equals("the")) {

    //get right element
    String right_element = p.getRightElement().toString();

    //assign value of right element to variable
    the = Float.valueOf(right_element);

    //stop the loop
    loop = false;
  }
}

//p of king given the
```

```
Collections.sort(list4,
     new Comparator<PairOfWritables<PairOfStrings,
         FloatWritable>>() {
       public int compare(PairOfWritables<PairOfStrings,
           FloatWritable> e1,
            PairOfWritables<PairOfStrings, FloatWritable> e2) {
         if (e1.getRightElement().compareTo(e2.getRightElement())
             == 0) {
           return
               e1.getLeftElement().compareTo(e2.getLeftElement());
         }

         return
             e2.getRightElement().compareTo(e1.getRightElement());
       }
     });

i = 0;
loop = true;
for (PairOfWritables<PairOfStrings, FloatWritable> p : list4) {

  PairOfStrings bigram = p.getLeftElement();
  i++;

  //if loop is false
  if (loop == false) {

    //stop looping
    break;

  //if the word to the right of the left word is "king"
  } else if (bigram.getRightElement().equals("king")) {

    //get right element
    String right_element = p.getRightElement().toString();

    //assign value of right element to variable
    king = Float.valueOf(right_element);

    //stop the loop
    loop = false;
  }
}

//print p(romeo)
System.out.println("The given probability of \"romeo\": " +
    romeo);

//print p(is|romeo)
System.out.println("The probability of \"is\" given \"romeo\": "
    + is);

//print p(the|is)
System.out.println("The probability of \"the\" given \"is\": " +
    the);

//print p(king|the)
System.out.println("The probability of \"king\" given \"the\": "
    + king);

//calculate combined frequency of "romeo is the king"
float romeo_is_the_king = romeo * is * the * king;

//print p(romeo is the king)
```

```
      System.out.println("The probability of \"romeo is the king\": " +
          romeo_is_the_king);

    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }

  /**
   * Reads in the bigram relative frequency count file
   *
   * @param path
   * @return
   * @throws IOException
   */
  private static List<PairOfWritables<PairOfStrings, FloatWritable>>
      readDirectory(Path path)
      throws IOException {

    File dir = new File(path.toString());
    ArrayList<PairOfWritables<PairOfStrings, FloatWritable>>
        relativeFrequencies = new
        ArrayList<PairOfWritables<PairOfStrings, FloatWritable>>();
    for (File child : dir.listFiles()) {
      if (".".equals(child.getName()) || "..".equals(child.getName())) {
        continue; // Ignore the self and parent aliases.
      }
      FileInputStream bigramFile = null;

      bigramFile = new FileInputStream(child.toString());

      // Read in the file
      DataInputStream resultsStream = new DataInputStream(bigramFile);
      BufferedReader results = new BufferedReader(new
          InputStreamReader(resultsStream));

      StringTokenizer rToken;
      String rLine;
      String firstWord;
      String secondWord;
      String frequency;

      // iterate through every line in the file
      while ((rLine = results.readLine()) != null) {
        rToken = new StringTokenizer(rLine);
        // extract the meaningful information
        firstWord = rToken.nextToken();
        //remove leading ( and trailing ,
        firstWord = firstWord.substring(1, firstWord.length() - 1);
        secondWord = rToken.nextToken();
        //remove trailing )
        secondWord = secondWord.substring(0, secondWord.length() - 1);
        frequency = rToken.nextToken();

        relativeFrequencies.add(new PairOfWritables<PairOfStrings,
            FloatWritable>(
            new PairOfStrings(firstWord, secondWord), new
                FloatWritable(Float
                .parseFloat(frequency))));

      }
      if (bigramFile != null)
        bigramFile.close();
    }
```

```
        return relativeFrequencies;
    }
}
```

Task 3

(a)

Table 4: Histogram of *tf* values for ”king”.

how many times	in how many lines
1	4519
2	393
3	66
4	8
5	5
6	1

(b)

Table 5: Histogram of *tf* values for ”macbeth”.

how many times	in how many lines
1	298
2	3
3	2

Table 6: Histogram of *tf* values for ”juliet”.

how many times	in how many lines
1	220

(c)

Postings corresponding to the term ”martino”:

”martino” appears 1 time in 1 line with line number (*docno*): **39154**

Java code

```
//BuildInvertedIndex.java

/*
 * Cloud9: A MapReduce Library for Hadoop
 *
 * Licensed under the Apache License, Version 2.0 (the "License"); you
 * may not use this file except in compliance with the License. You may
 * obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
 * implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */
```

```
package edu.umd.cloud9.examples;

import java.io.IOException;

import java.util.Iterator;
import java.util.StringTokenizer;
//added Collections package
import java.util.Collections;

//added PairOfInts package
import edu.umd.cloud9.io.PairOfInts;
//added PairOfWritables package
import edu.umd.cloud9.io.PairOfWritables;
//added ArrayListWritable package
import edu.umd.cloud9.io.ArrayListWritable;

//added EntryObject2IntFrequencyDistribution package
import edu.umd.cloud9.util.EntryObject2IntFrequencyDistribution;
//added Object2IntFrequencyDistribution package
import edu.umd.cloud9.util.Object2IntFrequencyDistribution;
//added PairOfObjectInt package
import edu.umd.cloud9.util.PairOfObjectInt;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

import org.apache.log4j.Logger;

/**
 * <p>
 * Simple inverted index. This Hadoop Tool builds an inverted index in
   flat text file, and
 * takes the following command-line arguments:
 * </p>
 *
 * <ul>
 * <li>[input-path] input path</li>
 * <li>[output-path] output path</li>
 * <li>[num-mappers] number of mappers</li>
 * <li>[num-reducers] number of reducers</li>
 * </ul>
 *
 * @author Jimmy Lin
 * @author Marc Sloan
 */
```

```
public class BuildInvertedIndex extends Configured implements Tool {
  private static final Logger sLogger =
      Logger.getLogger(BuildInvertedIndex.class);

  /**
   * Mapper: emits (term, term_freq) for every word occurrence
   *
   */
  private static class MyMapper extends MapReduceBase implements
       Mapper<LongWritable, Text, Text, PairOfInts> {

    /**
     * Store an Object2IntFrequencyDistribution, which will be mapped
     * to each word found in the test
     */
    //variable to hold counts
    private final static Object2IntFrequencyDistribution<String> counts
        = new EntryObject2IntFrequencyDistribution<String>();

    /**
     * reuse objects to save overhead of object creation
     */
    //variable to hold term
    private Text term = new Text();

    /**
     * Mapping function. This takes the text input, converts it into a
         String which is split into
     * words, then each of the terms is mapped to the OutputCollector
         with a term frequency.
     *
     * @param docno Input key, not used in this example
     * @param doc A line of input Text taken from the data
     * @param output Map from each term (Text) to its term_freq
         (PairOfInts)
     */
    public void map(LongWritable docno, Text doc, OutputCollector<Text,
        PairOfInts> output,
         Reporter reporter) throws IOException {

      //Convert input word into String and tokenize to find words
      String line = ((Text) doc).toString();
      StringTokenizer itr = new StringTokenizer(line);

      //clear counts
      counts.clear();

      //variable to hold term
      String word = null;

      //For each bigram, map it to a count of one. Duplicate bigrams
          will be counted
      //in the reduce phase.
      while (itr.hasMoreTokens()) {

        //update the word as the next token/word
        word = itr.nextToken();

        //if word exists
        if (word != null) {

          //increment frequency of word
          counts.increment(word);
        }
      }
      //for every count in counts
      for (PairOfObjectInt<String> count : counts) {

        //set term as left element of count
        term.set(count.getLeftElement());

        //set term frequency as the document number + the right element
            of count
        PairOfInts term_freq = new PairOfInts((int) docno.get(),
            count.getRightElement());

        //output term and its frequency
        output.collect(term, term_freq);
      }
    }
  }

  /**
   * Reducer: sums up all the counts
   *
   */
  private static class MyReducer extends MapReduceBase implements
      Reducer<Text, PairOfInts, Text, PairOfWritables<IntWritable,
          ArrayListWritable<PairOfInts>>> {

    /**
     * Stores the sum of counts for a term
     */
    //variable to hold document frequency
    private final static IntWritable doc_freq = new IntWritable();

    /**
     * @param docno The Text term
     * @param values An iterator over the values associated with this
          term
     * @param output Map from each docno (Text) to its inverted_index
          (PairOfWritables)
     * @param reporter Used to report progress
     */
    public void reduce(Text docno, Iterator<PairOfInts> doc,
        OutputCollector<Text, PairOfWritables<IntWritable,
            ArrayListWritable<PairOfInts>>> output,
        Reporter reporter) throws IOException {

      //sum up values
      //variable to hold frequency
      int freq = 0;

      //variable to hold postings
      ArrayListWritable<PairOfInts> postings = new
          ArrayListWritable<PairOfInts>();

      //until there are values left
      while (doc.hasNext()) {

        //add each value to the postings array by cloning it
        postings.add(doc.next().clone());

        //increment frequency
        freq++;
      }

      //sort postings array in ascending order
      Collections.sort(postings);
      //set document frequency
      doc_freq.set(freq);

      //form the inverted index: document frequency + postings
      PairOfWritables<IntWritable, ArrayListWritable<PairOfInts>>
          inverted_index = new PairOfWritables<IntWritable,
          ArrayListWritable<PairOfInts>>(doc_freq, postings);

      //output document number + inverted index = docno, doc freq,
          postings
      output.collect(docno, inverted_index);
    }
  }

  /**
   * Creates an instance of this tool.
   */
  public BuildInvertedIndex() {
  }

  /**
   * Prints argument options
   * @return
   */
  private static int printUsage() {
    System.out.println("usage: [input-path] [output-path] [num-mappers]
        [num-reducers]");
    ToolRunner.printGenericCommandUsage(System.out);
    return -1;
  }

  /**
   * Runs this tool.
   */
  public int run(String[] args) throws Exception {
    if (args.length != 4) {
      printUsage();
      return -1;
    }

    String inputPath = args[0];
    String outputPath = args[1];

    int mapTasks = Integer.parseInt(args[2]);
    int reduceTasks = Integer.parseInt(args[3]);

    sLogger.info("Tool: BuildInvertedIndex");
    sLogger.info(" - input path: " + inputPath);
    sLogger.info(" - output path: " + outputPath);
    sLogger.info(" - number of mappers: " + mapTasks);
    sLogger.info(" - number of reducers: " + reduceTasks);

    JobConf conf = new JobConf(BuildInvertedIndex.class);
    conf.setJobName("BuildInvertedIndex");

    conf.setNumMapTasks(mapTasks);
    conf.setNumReduceTasks(reduceTasks);

    FileInputFormat.setInputPaths(conf, new Path(inputPath));
    FileOutputFormat.setOutputPath(conf, new Path(outputPath));
    FileOutputFormat.setCompressOutput(conf, false);

    /**
     * Note that these must match the Class arguments given in the mapper
     */
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(PairOfInts.class);

    conf.setMapperClass(MyMapper.class);
    conf.setReducerClass(MyReducer.class);

    // Delete the output directory if it exists already
    Path outputDir = new Path(outputPath);
    FileSystem.get(outputDir.toUri(), conf).delete(outputDir, true);

    long startTime = System.currentTimeMillis();
    JobClient.runJob(conf);
    sLogger.info("Job Finished in " + (System.currentTimeMillis() -
        startTime) / 1000.0
         + " seconds");

    return 0;
  }

  /**
   * Dispatches command-line arguments to the tool via the
   * <code>ToolRunner</code>.
   */
  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new Configuration(), new
        BuildInvertedIndex(), args);
    System.exit(res);
  }
}
```

```
//LookupPostings.java

/*
 * Cloud9: A MapReduce Library for Hadoop
 *
 * Licensed under the Apache License, Version 2.0 (the "License"); you
 * may not use this file except in compliance with the License. You may
 * obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
 * implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

package edu.umd.cloud9.examples;

import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;

//added HashMap package
import java.util.HashMap;
//added Map package
import java.util.Map;
//added Set package
```

```java
import java.util.Set;
import java.util.List;
import java.util.StringTokenizer;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;

//added ArrayListWritable package
import edu.umd.cloud9.io.ArrayListWritable;
//added PairOfInts package
import edu.umd.cloud9.io.PairOfInts;
//added PairOfWritables package
import edu.umd.cloud9.io.PairOfWritables;

public class LookupPostings {
  public static void main(String[] args) throws IOException {

    //hard coded path to avoid setting arguments
    String[] fixed_path = {"output/build-inverted-index"};
    args = fixed_path;

    if (args.length != 1) {
      System.out.println("usage: [input-path]");
      System.exit(-1);
    }

    System.out.println("input path: " + args[0]);

    //calls to lookupPosting function
    lookupPosting("king", args[0].toString());
    lookupPosting("macbeth", args[0].toString());
    lookupPosting("juliet", args[0].toString());
    lookupPosting("martino", args[0].toString());

    //List<PairOfWritables<PairOfStrings, FloatWritable>> pairs =
    //SequenceFileUtils.readDirectory(new Path(args[0]));
  }

  /**
   * Reads in the inverted index file
   *
   * @param path
   * @return
   * @throws IOException
   */
  private static List<PairOfWritables<Text, PairOfWritables<IntWritable,
      ArrayListWritable<PairOfInts>>>> readDirectory(Path path)
       throws IOException {

    File dir = new File(path.toString());

    //variable to hold the final inverted index
      ArrayListWritable<PairOfWritables<Text,
          PairOfWritables<IntWritable, ArrayListWritable<PairOfInts>>>>
          inverted_index_final = new
          ArrayListWritable<PairOfWritables<Text,
          PairOfWritables<IntWritable,ArrayListWritable<PairOfInts>>>>();

      for (File child : dir.listFiles()) {
       if (".".equals(child.getName()) || "..".equals(child.getName())) {
         continue; // Ignore the self and parent aliases.
       }
```

```
FileInputStream buildInvertedIndexFile = null;
buildInvertedIndexFile = new FileInputStream(child.toString());

//Read in the file
DataInputStream resultsStream = new
    DataInputStream(buildInvertedIndexFile);
BufferedReader results = new BufferedReader(new
    InputStreamReader(resultsStream));

StringTokenizer rToken;
String rLine;

//variable to hold inverted index
String invertedindex;

//variable to hold document frequency
  String document_frequency;

  //variable to hold posting
  String posting;

  // iterate through every line in the file
while ((rLine = results.readLine()) != null) {
  //variable to hold postings
     ArrayListWritable<PairOfInts> postings = new
         ArrayListWritable<PairOfInts>();

     //split rLine on "\t"/tab
  rToken = new StringTokenizer(rLine, " ");

  //extract the meaningful information
  //extract the term
  Text term = new Text(rToken.nextToken());

  //extract inverted index >> (document frequency, [(docno,
      term_frequency)])
  invertedindex = rToken.nextToken();

  //split inverted index on "["
     rToken = new StringTokenizer(invertedindex, "[");

     //take the next token >> (int,
     document_frequency = rToken.nextToken();

     //extract document frequency >> int
     document_frequency = document_frequency.substring(1,
         document_frequency.length() - 2);

     //take the next token >> (int, int)
     posting = rToken.nextToken();

     //extract posting >> (docno, term_frequency)
     posting = posting.substring(0, posting.length() - 2);

     //split posting on "(" to extract separate values: docno,
         term_frequency
     rToken = new StringTokenizer(posting, "(");

     //until there are tokens left
     while (rToken.hasMoreTokens()) {

       //split rToken on "," to extract separate values: docno,
           term_frequency
         StringTokenizer pToken = new
             StringTokenizer(rToken.nextToken(), ",");
```

```
            //extract docno >> int
            String docno = pToken.nextToken();

            //take the next token >> int)
            String term_frequency = pToken.nextToken();

            //extract term_frequency >> int
            term_frequency = term_frequency.substring(1,
                term_frequency.length() - 1);

            //add docno and term_frequency values to the postings
                array, need to be wrapped using Integer
            postings.add(new PairOfInts(Integer.valueOf(docno),
                Integer.valueOf(term_frequency)));
         }

         //assign document_frequency value to doc_freq
         IntWritable doc_freq = new
             IntWritable(Integer.valueOf(document_frequency));

         //form inverted_index >> (doc freq, [(docno,
             term_frequency)])
         PairOfWritables inverted_index = new
             PairOfWritables<IntWritable,
             ArrayListWritable<PairOfInts>>(doc_freq, postings);

         //add term to inverted_index >> (term, (doc_freq, [(docno,
             term_frequency)]))
         PairOfWritables<Text, PairOfWritables<IntWritable,
             ArrayListWritable<PairOfInts>>> term_inverted_index =
             new PairOfWritables(term, inverted_index);

         //add term and inverted index to array >> [(term, (doc_freq,
             [(docno, term_frequency)]))]
         inverted_index_final.add(term_inverted_index);
     }

     if (buildInvertedIndexFile != null)
       buildInvertedIndexFile.close();
   }

    //return final inverted index array
   return inverted_index_final;
 }

 public static void lookupPosting(String term, String args) throws
     IOException {

   //variable to hold pairs of inverted index >> [(term, (doc_freq,
       [(docno, term_frequency)]))]
   List<PairOfWritables<Text, PairOfWritables<IntWritable,
       ArrayListWritable<PairOfInts>>>> pairs;

   //assign read output to pairs
   pairs = readDirectory(new Path(args));

   //hashmap variable to hold histogram
   HashMap<String, HashMap<Integer, Integer>> histogram = new
       HashMap<String, HashMap<Integer, Integer>>();

   //variable to hold line_number for "martino"
   int line_number = 0;

   //for every pair in pairs
```

```
for (PairOfWritables<Text, PairOfWritables<IntWritable,
    ArrayListWritable<PairOfInts>>> pair : pairs) {

  //variable to hold current term, used to compare to term supplied
      as argument
  String term_to_compare = pair.getLeftElement().toString();

  //variable to hold inverted index
  PairOfWritables inverted_index = pair.getRightElement();

  //if current term equals to term supplied as argument
  if (term_to_compare.equals(term)) {

    //variable to hold entries array
    ArrayListWritable<PairOfInts> elements = (ArrayListWritable)
        inverted_index.getRightElement();

    //for every entry in entries
    for (PairOfInts element : elements) {

      //variable to hold term frequency
      int term_frequency = element.getRightElement();

      //if current term hasn't already been added to hashmap
      if (histogram.get(term) == null) {

        //variable to hold term frequency
        HashMap term_freq = new HashMap();

        //add term_frequency to term_freq and start the count with 1
        term_freq.put(term_frequency, 1);

        //add the term and its frequency to the histogram
        histogram.put(term, term_freq);

      //if current term has already been added to hashmap
      } else {

        //set count to 0
        int count = 0;

        //if current term's term frequency hasn't already been
            added to hashmap
        if (histogram.get(term).get(term_frequency) == null) {

          //start count
          count = 0;

        //if current term's term frequency has already been added
            to hashmap
        } else {

          //update count
          count = histogram.get(term).get(term_frequency);
        }

        //variable to hold term frequency
        HashMap term_freq = new HashMap(histogram.get(term));

        //add term_frequency to term_freq and increment the count
        term_freq.put(term_frequency, count + 1);

        //add the term and its frequency to the histogram
        histogram.put(term, term_freq);
      }
      }

    //if current term equals martino, get the line_number
    } else if (term.equals("martino")) {

      //variable to hold entries array
      ArrayListWritable<PairOfInts> elements =
          (ArrayListWritable<PairOfInts>)
          inverted_index.getRightElement();

      //variable to hold line number (docno)
      line_number = elements.get(0).getLeftElement();
    }
  }

  //print term supplied as argument
  System.out.println("histogram for term: " + '\"' + term + '\"');

  //get entries for term argument
  Set<Map.Entry<String, HashMap<Integer, Integer>>> elements =
      histogram.entrySet();

  //for every entry of the term argument
  for (Map.Entry<String, HashMap<Integer, Integer>> element :
      elements) {

    //get entry value (right element)
    Set<Map.Entry<Integer, Integer>> frequencies =
        element.getValue().entrySet();

    //for every frequency of the term argument
    for (Map.Entry<Integer, Integer> frequency : frequencies) {
      //variable to hold time/times label for keys
      String key_times;

      //variable to hold time/times label for labels
      String val_times;

      //if frequency key equals to 1 (left element)
      if (frequency.getKey() == 1) {

        //singular
        key_times = " time in ";

      //if frequency key doesn't equal to 1 (left element)
      } else {

        //plural
        key_times = " times in ";
      }

      //if frequency key equals to 1 (right element)
      if (frequency.getValue() == 1) {

        //singular
        val_times = " line";

      //if frequency key doesn't equal to 1 (right element)
      } else {

        //plural
        val_times = " lines";
      }

      //variable to hold line number of "martino" occurence
```

```
            String martino = "";

            //if term supplied as argument equal to "martino"
            if (term.equals("martino")) {

              //variable to hold line number of "martino" occurence
              martino = " with line number: " + line_number;
            }

            //print frequencies >> "how many times" term appears in "how
                many lines"
            System.out.println(frequency.getKey() + key_times +
                frequency.getValue() + val_times + martino);
          }

          //print empty line, used as separator
          System.out.println();
        }
      }
    }
```

References

[1] Wikipedia. *Okapi BM25 — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Okapi%20BM25&oldid=711620475. [Online; accessed 05-April-2016]. 2016.

[2] Wikipedia. *Discounted cumulative gain — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Discounted%20cumulative%20gain&oldid=695019989. [Online; accessed 05-April-2016]. 2016.

[3] StackOverflow. *widely used formula for DCG seems to be wrong?* http://stackoverflow.com/questions/33239416/widely-used-formula-for-dcg-seems-to-be-wrong. [Online; accessed 05-April-2016].

[4] Tetsuya Sakai. "Metrics, statistics, tests". In: *Bridging Between Information Retrieval and Databases*. Springer, 2014, pp. 116–163.

[5] Jaime Carbonell and Jade Goldstein. "The use of MMR, diversity-based reranking for reordering documents and producing summaries". In: *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 1998, pp. 335–336.

[6] Harry Markowitz. "Portfolio selection". In: *The journal of finance* 7.1 (1952), pp. 77–91.

[7] Jun Wang and Jianhan Zhu. "Portfolio theory of information retrieval". In: *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2009, pp. 115–122.

[8] Wikipedia. *Pearson product-moment correlation coefficient — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Pearson%20product-moment%20correlation%20coefficient&oldid=713522265. [Online; accessed 06-April-2016]. 2016.

[9] Charles LA Clarke et al. "Novelty and diversity in information retrieval evaluation". In: *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2008, pp. 659–666.