Assignment 4
Introduction to Information Retrieval
CS734/834

John Berlin

December 8, 2016

Note

During the course of this class I have found myself having to reusing code from previous assignments and reworking them for use in the current question I am working on at the time. This came to a head in this assignment where the changes to existing code became non-trivial which lead me to create more generalized versions of common operations used throughout this course. In order to keep the answers to questions about the methodology used in answering rather than implementation details using these generalized methods I will explain what they do here. The python3 classes seen in Source Code 9 are to be utilized with the context control statement `with` most commonly used with the `open` statement which opens a file object and automatically closes it when the control flow of the program exits that context. This is en-essence what each of these classes do.

The BeautifulSoupFromFile classes takes the path to a file plus the parser type used when creating the soup i.e html or xml and returns to the context using it the soup produced from opening the file and creating a new BeautifulSoup. AutoSaver and AutoSaveCsv takes the type to be save (python list or dictionary) with the path to the file to be written and returns to the calling context an empty version of the type specified to be populated in the scope of that context finally saving the types content to the file path when the context is exited. They also take optional arguments of functions which are used to format the output and or select what entities in they save type to be written to file. RunCommandSaveOut classes takes a function producing a `subprocess.Popopen` class, the command line arguments to pass to the function and supplies the function with a file object by which the standard output of the spawned process is serialized to. CleanFileLines class reads the contents of a specified file and applies a cleaning function to the contents (for example: $f(\text{“}A, B, C, X, Y, Z\backslash\mathtt{n}\text{”}) \rightarrow \text{“}A, B, C\text{”}$) returning to the context the cleaned lines with the option to save the cleaned lines back to the originating file. FileLinePrinter operates similarly but only prints the raw lines of the specified files. SelectFromFile also applies a transformation function to the files content but also applies a selector function to the cleaned contents of the file returning to the context only what is produced after applying it to the files contents. RandFinderFile operates similarly but the selection function supplied is to randomly select a line from the files content which is then returned to the context.

The other functions used in answering the questions in this assignment follow the convention of the previous assignments defining less intensive helpers in the file util.py seen in Source Code 11. Also the shell script created to run galago commands in assignment 3 Source Code 12 was also utilized with the eval argument which sets up the correct command line arguments to run the galago function eval as the default options for this do not return results required for this assignment. It explicitly asks for all available metrics to be computed the full list of metrics can be seen in Source Code 12. A more specialized utility file was created for use with the context classes described previously for use with the output of galago functions especially for sanatizing the output of the query runs to work with the eval function of galago. These functions can be seen in Source Code 10.

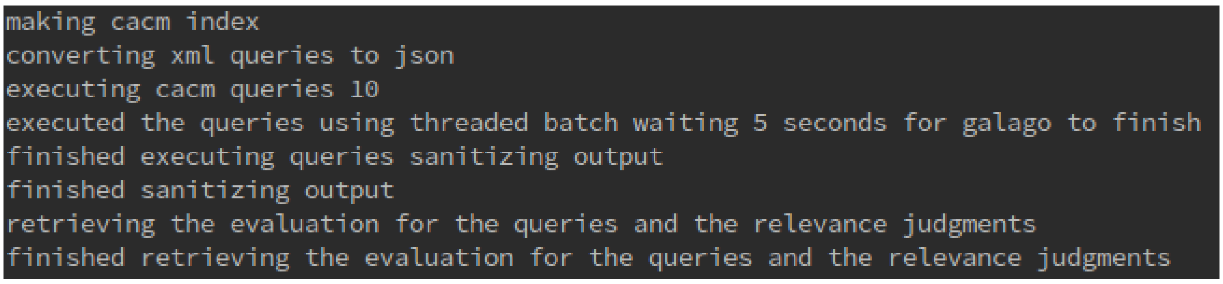
Finally it must also be noted that I will be using an updated version of galago (v3.10) in this report. For a more detailed explanation as to why and the differences in this version from the one used by the textbook please consult this same section in the report for assignment 3 for more details.

Q.1 Question 8.3

```
For one query in the CACM collection (provided at the book website),
generate a ranking using Galago, and then calculate average precision, NDCG at 5
and 10, precision at 10, and the reciprocal rank by hand.
```

Q.1 Answer

Like the previous assignments questions where galago was used to query a collection creation of the index and query creation is the first step. This process was automated thanks to the lessons learned from having used galago before and having the queries were supplied to us. Since the provided queries are in the old xml style they are converted to json after the index has been created. The automation process can be seen in the python file **q1_cacm_query_rel.py**, Source Code 1 along with the rest of the code used in answering this question. Running the file `python3 q1_cacm_query_rel.py` for the first time will create the index, convert the queries and after the running the queries against the CACM collection execute the *eval* function of galago against the runs results automatically. Subsequent executions of this file will not run the galago commands if the files produced by running this file are found to exist. The first time output generated can be seen in Figure 1.



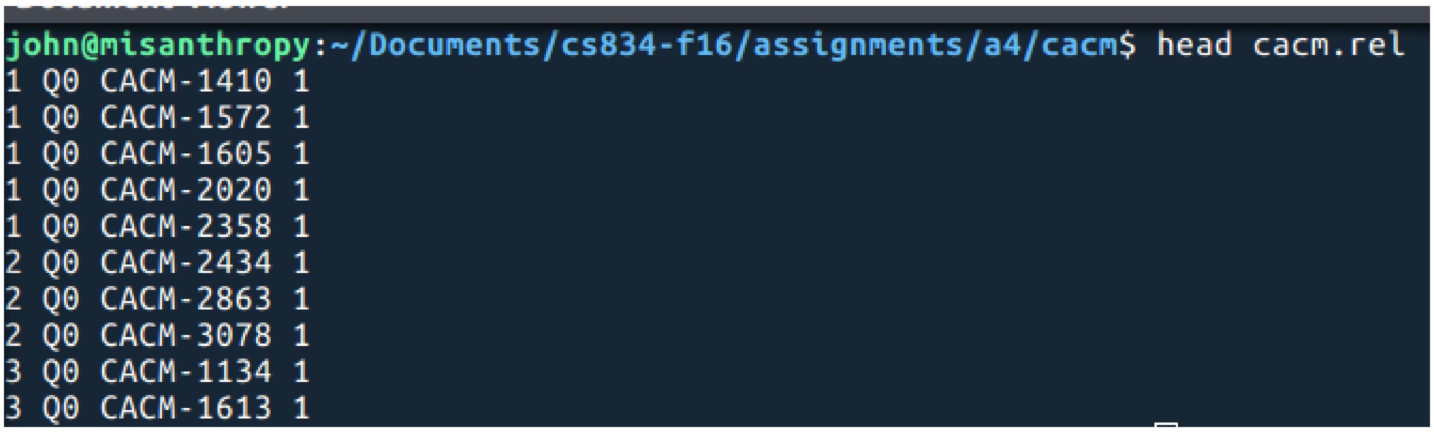
```
making cacm index
converting xml queries to json
executing cacm queries 10
executed the queries using threaded batch waiting 5 seconds for galago to finish
finished executing queries sanitizing output
finished sanitizing output
retrieving the evaluation for the queries and the relevance judgments
finished retrieving the evaluation for the queries and the relevance judgments
```

Figure 1: Building CACM collection index

Their is an extra step noted in the output generated by this python, Figure 1, which is *finished executing queries sanitizing output*. This extra step is required in order for the eval function of galago to work properly. The batch search function of galago as stated in the documentation produces output in the following format

`<qid> Q0 <docid> <rank> <score> <system-name> <begin> <end>`

Running the batch search command made the *docid* part of the output contain the full path to a CACM document on my machine. The full path for the docid caused the **eval** function of galago to not produce any output. This puzzled me for upwards of thirty minutes as I attempted to figure out why by recreating the index and playing around with various parameters to the **batch-search** function. It was not until I looked at the provided *cacm.rel* file provided on the textbooks webcite. Figure 2 shows the first few lines of this file.



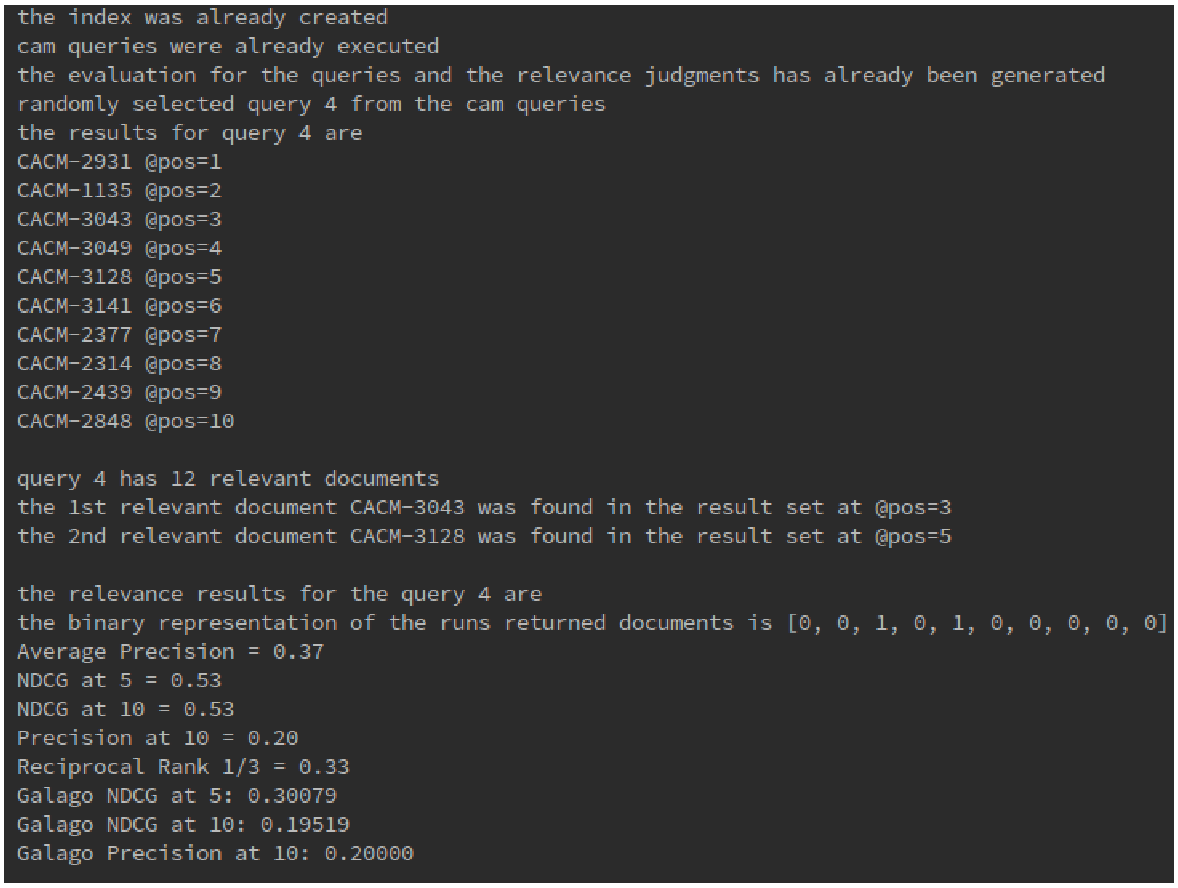
```
john@misanthropy:~/Documents/cs834-f16/assignments/a4/cacm$ head cacm.rel
1 Q0 CACM-1410 1
1 Q0 CACM-1572 1
1 Q0 CACM-1605 1
1 Q0 CACM-2020 1
1 Q0 CACM-2358 1
2 Q0 CACM-2434 1
2 Q0 CACM-2863 1
2 Q0 CACM-3078 1
3 Q0 CACM-1134 1
3 Q0 CACM-1613 1
```

Figure 2: head cacm.rel

As you can see in the output of running the unix command *head* on the file there is only the name of the file contained in the collection with the query number it is relevant for and a weight (Q0 is disregarded by galago). On inspection I realized that galago must not be able to match a queries returned documents to the relevance file if they contain the full path to them. I could not find any option to suppress this in the output for the runs so cleaning the runs results to be in the form $<$ *quid Q*0 *docid rank score* $>$ in order to be matched with the relevance file was the only way forward. After sanitizing the output the the relevance results for the run can was determined. Now the question stated **the reciprocal rank by hand** after a list of metrics to be calculated which could mean just the reciprocal rank or all of them but it also can be roughly coerced into the interpretation of by code utilizing values produced by code you wrote which is how I interpreted its meaning. Plus I wanted to use a python file I found on gist a while back ;) which can be seen in Source Code 8.

The query for the metrics calculation is selected by random from the runs results and then calculated by hand written code along with the output of the same metrics as galago computed them for a sanity check. The by hand calculation is done as following: select a random run from the batch-search runs *eval* results output, select the relevant documents for that run from the *cam.rel* file and then select that runs returned documents from the batch-search output. If the run had relevant documents or the *eval* results for query did not have NaN computed by galago for the metrics do the by hand computation of the metrics. Queries 34 35 41 46 47 50 51 52 53 54 55 56 had NaN values for the metrics when calculated by galago

For each document returned by the query create a list of binary values indicated if the returned document at position n was relevant (1) or not (0) then calculate NDCG@5 and NDCG@10, P@10, Average Precision using the created list plus the Reciprocal Rank. This process was done for three randomly chosen query 4 Figure 3, query 9 Figure 4 and query 11 Figure 5, along with the galago computed scores for NDCG@5 and NDCG@10, P@10. It must be noted that galago computes Average Precision for MAP when consulting the source code to discover the available metrics. As seen in the figures the computed P@10 values for all three queries match the calculation done by galago while the NDCG values for queries 4 and 9 are way off in comparison to galago computed versions whereas query 11 had computed NDCG values off by only .01 for NDCG@5 and .17 for NDCG@10. Consulting the methods used to compute these metrics besides using numpy to calculate the score the methods for all intents and purposes are implemented correctly. The number of returned documents for each each query was set to ten when executed using galago as the Google only 10 results per page in its search results. As seen in the output queries 4,9 and 11 have 12, 9, and 19 relevant documents respectfully according to the provided *cam.rel* used in answering this question. This leads me to believe that galago is considering the full set of relevant documents in the calculations it returns.



```
the index was already created
cam queries were already executed
the evaluation for the queries and the relevance judgments has already been generated
randomly selected query 4 from the cam queries
the results for query 4 are
CACM-2931 @pos=1
CACM-1135 @pos=2
CACM-3043 @pos=3
CACM-3049 @pos=4
CACM-3128 @pos=5
CACM-3141 @pos=6
CACM-2377 @pos=7
CACM-2314 @pos=8
CACM-2439 @pos=9
CACM-2848 @pos=10

query 4 has 12 relevant documents
the 1st relevant document CACM-3043 was found in the result set at @pos=3
the 2nd relevant document CACM-3128 was found in the result set at @pos=5

the relevance results for the query 4 are
the binary representation of the runs returned documents is [0, 0, 1, 0, 1, 0, 0, 0, 0, 0]
Average Precision = 0.37
NDCG at 5 = 0.53
NDCG at 10 = 0.53
Precision at 10 = 0.20
Reciprocal Rank 1/3 = 0.33
Galago NDCG at 5: 0.30079
Galago NDCG at 10: 0.19519
Galago Precision at 10: 0.20000
```

Figure 3: Relevance Metrics For Queries 4

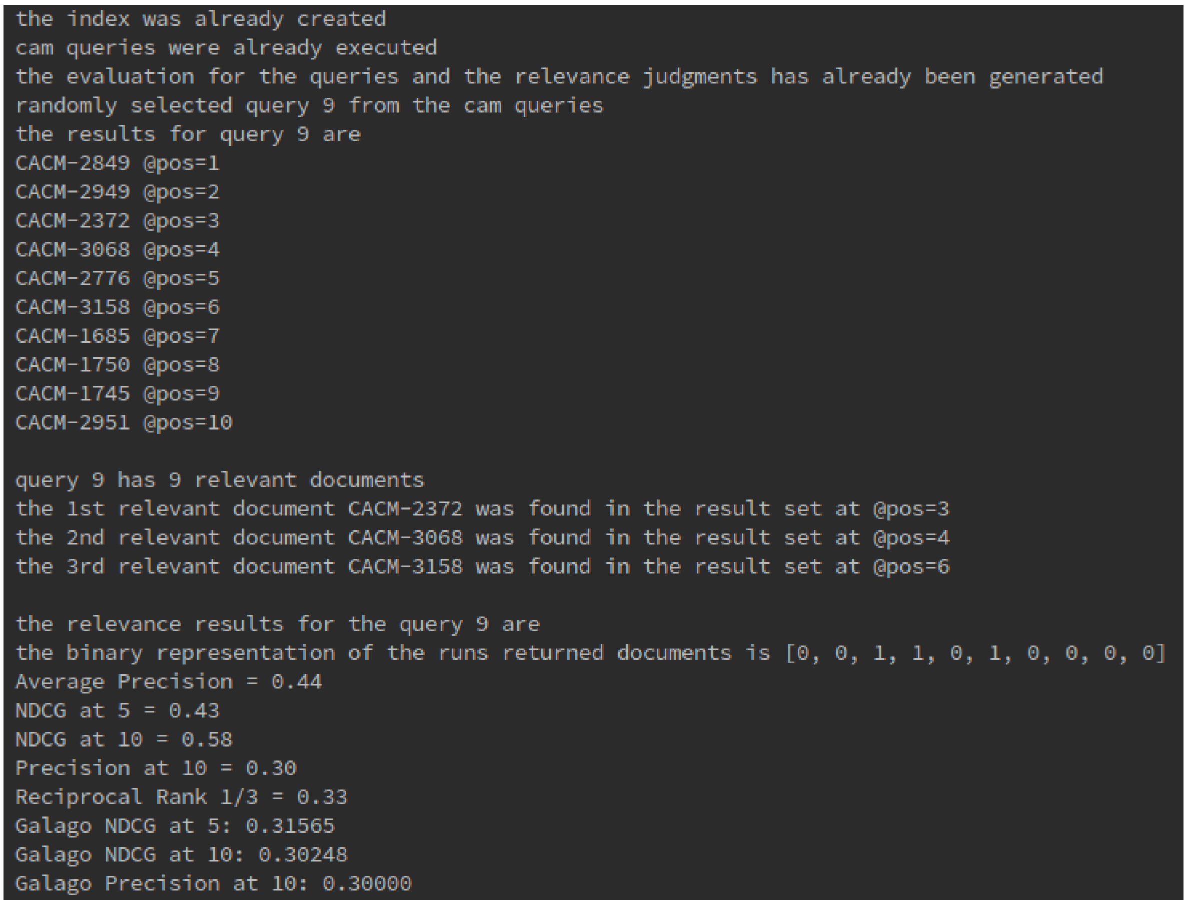
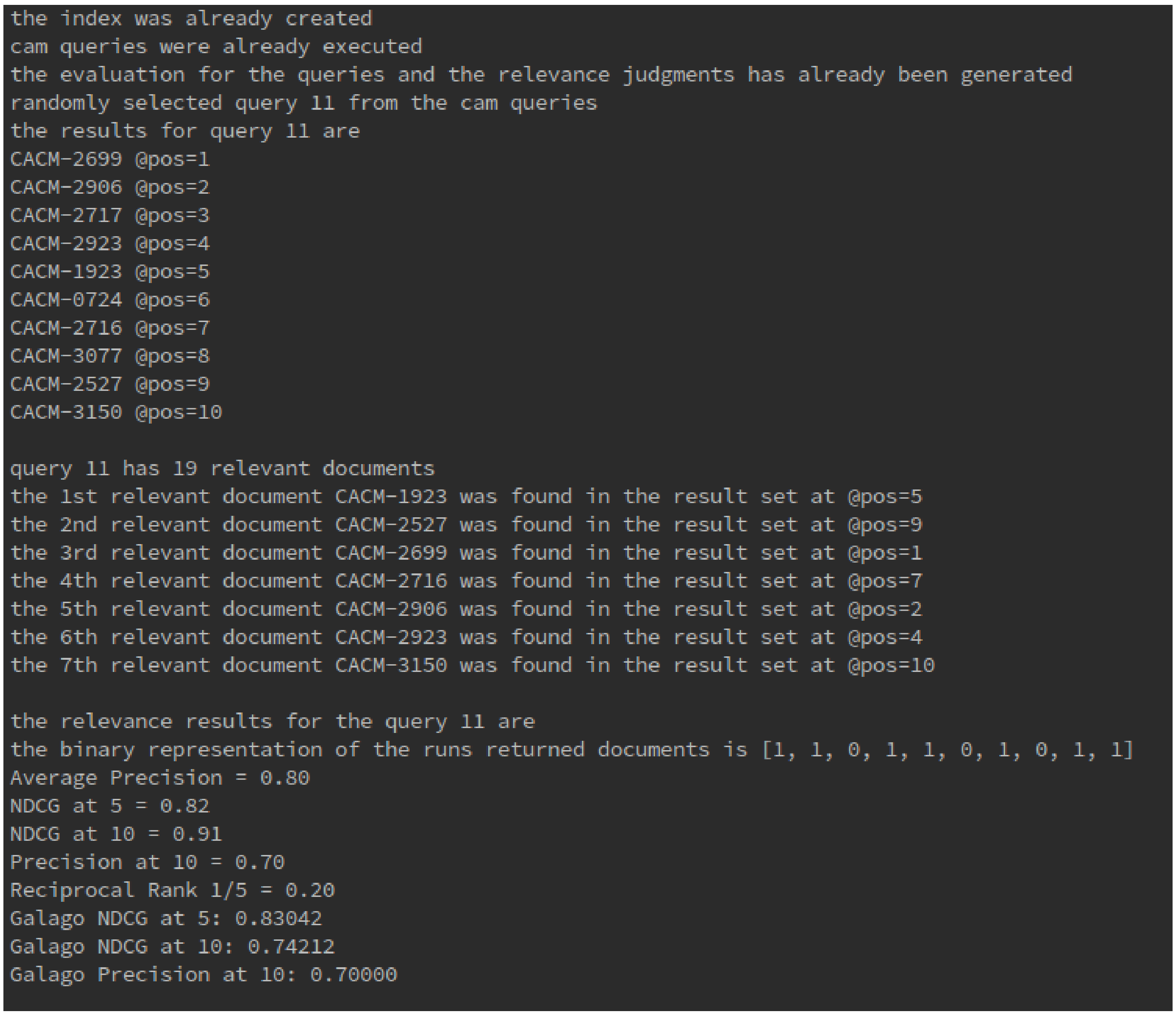

```
the index was already created
cam queries were already executed
the evaluation for the queries and the relevance judgments has already been generated
randomly selected query 9 from the cam queries
the results for query 9 are
CACM-2849 @pos=1
CACM-2949 @pos=2
CACM-2372 @pos=3
CACM-3068 @pos=4
CACM-2776 @pos=5
CACM-3158 @pos=6
CACM-1685 @pos=7
CACM-1750 @pos=8
CACM-1745 @pos=9
CACM-2951 @pos=10

query 9 has 9 relevant documents
the 1st relevant document CACM-2372 was found in the result set at @pos=3
the 2nd relevant document CACM-3068 was found in the result set at @pos=4
the 3rd relevant document CACM-3158 was found in the result set at @pos=6

the relevance results for the query 9 are
the binary representation of the runs returned documents is [0, 0, 1, 1, 0, 1, 0, 0, 0, 0]
Average Precision = 0.44
NDCG at 5 = 0.43
NDCG at 10 = 0.58
Precision at 10 = 0.30
Reciprocal Rank 1/3 = 0.33
Galago NDCG at 5: 0.31565
Galago NDCG at 10: 0.30248
Galago Precision at 10: 0.30000
```

Figure 4: Relevance Metrics For Query 9



```
the index was already created
cam queries were already executed
the evaluation for the queries and the relevance judgments has already been generated
randomly selected query 11 from the cam queries
the results for query 11 are
CACM-2699 @pos=1
CACM-2906 @pos=2
CACM-2717 @pos=3
CACM-2923 @pos=4
CACM-1923 @pos=5
CACM-0724 @pos=6
CACM-2716 @pos=7
CACM-3077 @pos=8
CACM-2527 @pos=9
CACM-3150 @pos=10

query 11 has 19 relevant documents
the 1st relevant document CACM-1923 was found in the result set at @pos=5
the 2nd relevant document CACM-2527 was found in the result set at @pos=9
the 3rd relevant document CACM-2699 was found in the result set at @pos=1
the 4th relevant document CACM-2716 was found in the result set at @pos=7
the 5th relevant document CACM-2906 was found in the result set at @pos=2
the 6th relevant document CACM-2923 was found in the result set at @pos=4
the 7th relevant document CACM-3150 was found in the result set at @pos=10

the relevance results for the query 11 are
the binary representation of the runs returned documents is [1, 1, 0, 1, 1, 0, 1, 0, 1, 1]
Average Precision = 0.80
NDCG at 5 = 0.82
NDCG at 10 = 0.91
Precision at 10 = 0.70
Reciprocal Rank 1/5 = 0.20
Galago NDCG at 5: 0.83042
Galago NDCG at 10: 0.74212
Galago Precision at 10: 0.70000
```

Figure 5: Relevance Metrics For Query 11

Source Code 1: Single CACM Query Relevance

```python
from util import *
from contextClasses import *
from assignmentFilePaths import *
from cam_q_utils import *
from ranking_methods import ndcg_at_k, average_precision, precision_at_k


def build_idx_qs():
    idx_maker = galago_mk_index(cacm_idx, cacm_in)
    if idx_maker is not None:
        print('making cacm index')
        stdout, stderr = idx_maker.communicate()
        rc = idx_maker.returncode
        if rc != 0:
            print('we had failure making the index for cacm')
        else:
            print('converting xml queries to json')
            cam_xmlqs_to_json(cacm_idx)
    else:
        print('the index was already created')


def execute_queries(qresult_path=cacm_q_out_p, rel_outp=cacm_rel_out, num=10):
    if not path_exists(qresult_path):
        print('executing cacm queries', num)
        rc = galago_search(qpath=cacm_qs, idxpath=cacm_idx, retpath=qresult_path, requested=num)
        if rc != 0:
            print(
                'we might have had error while executing queries\nif the ouput does not display '
                'org.lemurproject.galago.core.tools.apps.ThreadedBatchSearch run\nINFO: Still running... \nthere
                ↪  was '
                'an issue')
        import time
        print('executed the queries using threaded batch waiting 5 seconds for galago to finish')
        time.sleep(5)
        print('finished executing queries sanitizing output')
        with CleanFileLines(qresult_path, q_result_cleaner(), sort_output=q_results_sorter, save_back=True):
            print('finished sanitizing output')
    else:
        print('cam queries were already executed')

    if not path_exists(rel_outp):
        print('retrieving the evaluation for the queries and the relevance judgments')
        cline = './rungalago.sh eval cacm/cacm.rel %s %d' % (qresult_path, num)
        with RunCommandSaveOut(cline, rel_outp, command_fun=forward_galago_output, print_file=True) as g_eval:
            rc = g_eval.returncode
            print('finished retrieving the evaluation for the queries and the relevance judgments')
    else:
        print('the evaluation for the queries and the relevance judgments has already been generated')


def run_queries():
    qresult_path = cacm_q_out_p
    rel_outp = cacm_rel_out
    execute_queries(qresult_path=qresult_path, rel_outp=rel_outp, num=10)


def eval_q_relevancej():
    transformer = ltransformer_group_by(lambda line: line.split(' ')[0])
    with RandFinderFile(cacm_q_out_p, transformer=transformer) as (num, qs):
        print('randomly selected query %s from the cam queries' % num)
        query_results_pos = {}
        doc_list = []
        print('the results for query %s are' % num)
        for qline in rand_select_stripper(qs):
            qn, q0, doc, pos, score = qline.split(' ')
            doc_list.append(doc)
            print('%s @pos=%s' % (doc, pos))
            query_results_pos[doc] = pos, score
        transformer1 = rel_line_trans(0)
        transformer2 = rel_line_trans(1)
        with SelectFromFile(cacm_rel, selector=rel_docs_transform(num), transformer=transformer1) as rel, \
                SelectFromFile(cacm_rel_out, selector=rel_results_transform(num), transformer=transformer2) as
                ↪  rel_ret:

            if not is_good(rel, rel_ret):
                print('no relevant documents for query %s' % num)
            else:
                print('\nquery %s has %d relevant documents' % (num, len(rel)))
                reciprocal_rank = None
                reciprocal_ranks = None
                rr_found = False
                count = 1
                for doc in rel:
                    it = query_results_pos.get(doc, None)
                    if it is not None:
                        print('the %s relevant document %s was found in the result set at @pos=%s' % (
                            ordinal(count), doc, it[0]))
                        if not rr_found:
                            reciprocal_rank = '%.2f' % (1 / int(it[0]))
                            reciprocal_ranks = '1/%s' % it[0]
                            rr_found = True
                    count += 1
                calc_list = []
                for ret_doc in doc_list:
                    if ret_doc in rel:
                        calc_list.append(1)
                    else:
                        calc_list.append(0)

                print('\nthe relevance results for the query %s are' % num)
                print('the binary representation of the runs returned documents is', calc_list)
                print('Average Precision = %.2f' % average_precision(calc_list))
                print('NDCG at 5 = %.2f' % ndcg_at_k(calc_list, 5))
                print('NDCG at 10 = %.2f' % ndcg_at_k(calc_list, 10))
                print('Precision at 10 = %.2f' % precision_at_k(calc_list, 10))
                print('Reciprocal Rank %s =' % reciprocal_ranks, reciprocal_rank)
                print('Galago NDCG at 5:', rel_ret['ndcg5'])
                print('Galago NDCG at 10:', rel_ret['ndcg10'])
                print('Galago Precision at 10:', rel_ret['P10'])


if __name__ == '__main__':
    build_idx_qs()
    run_queries()
    eval_q_relevancej()
```

Q.2 Question 8.5

```
Generate the mean average precision, recall-precision graph, average NDCG
at 5 and 10, and precision at 10 for the entire CACM query set
```

Q.2 Answer

The values used in the graph for this question were retrieved from the results generated from *eval* function used in section Q.1 and then saved to a csv file Source Code 2 for plotting via R Source Code 3. Figure 6 shows the metrics plotted against each and it is clear to see that results follow a similar pattern. This plot also brings up the question of does galago consider the entire relevant document for a query in these calculations as was noted in section Q.1.

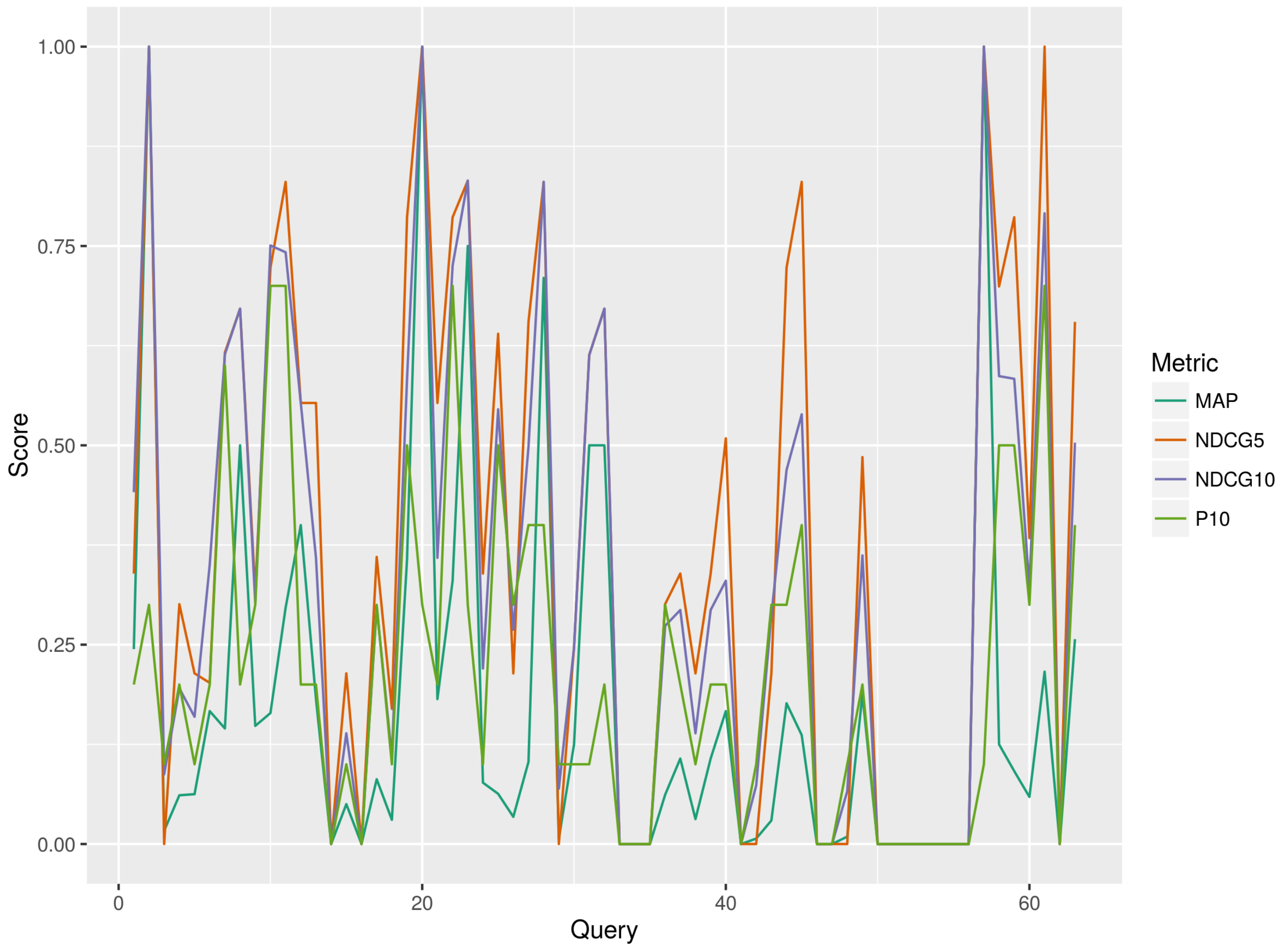


Figure 6: Galago Eval Metrics For All Queries

Considering Figure 7 which plots the Recall and Precision scores for each query as calculated by galago, they as well follow the pattern seen in Figure 6.

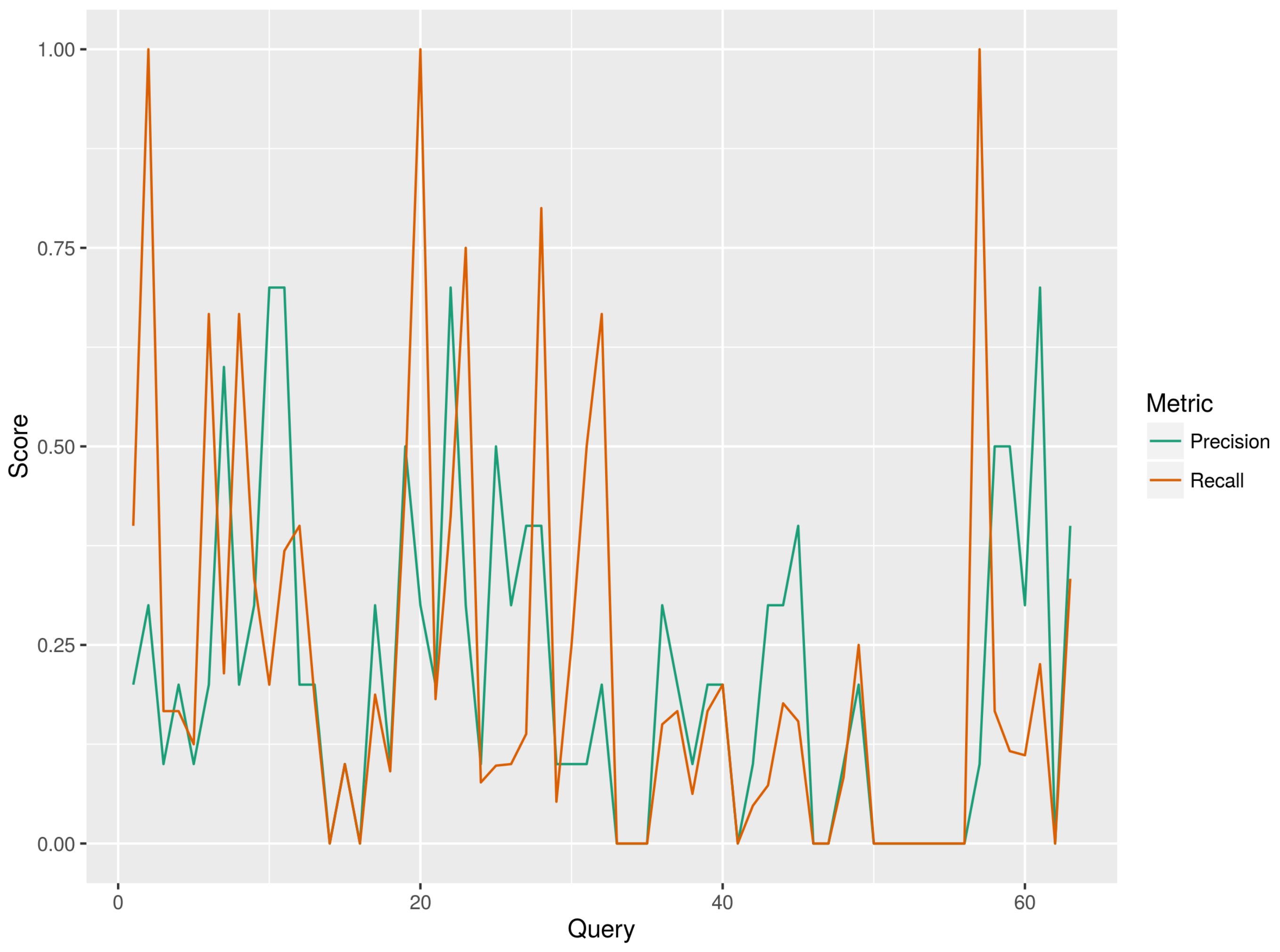


Figure 7: Galago Eval Recall Precision For All Queries

The seemingly eradicate hills and valleys are present in both plots and when considering Figure 7 it becomes clear that the overall performance of the queries was poor due to the number of queries with NaN values and how galago considers all relevant documents in its metrics. Now consider Table 1 which shows the queries with less than 10 relevant documents and Figure 8 showing the number of relevant documents for the all of the CACM queries. Figure 8 shows a relatively similar pattern as is seen in Figure 6 and Figure 7. Table 1 shows 19 out of 64 queries which had relevant document counts less than 10. These queries further confirm my assumption that galago considers all relevant documents as Figure 6 and Figure 7 start out at 1.0 which is perfect. Then a sharp drop as query 2 only had 3 relevant documents and then back up for query 5 which had 8.

Table 1: Relevant Document < 10 Count

Query	Count
1	5
2	3
3	6
5	8
6	3
8	3
9	9
12	5
20	3

Query	Count
23	4
28	5
30	4
31	2
32	3
33	1
49	8
57	1
62	8
64	1

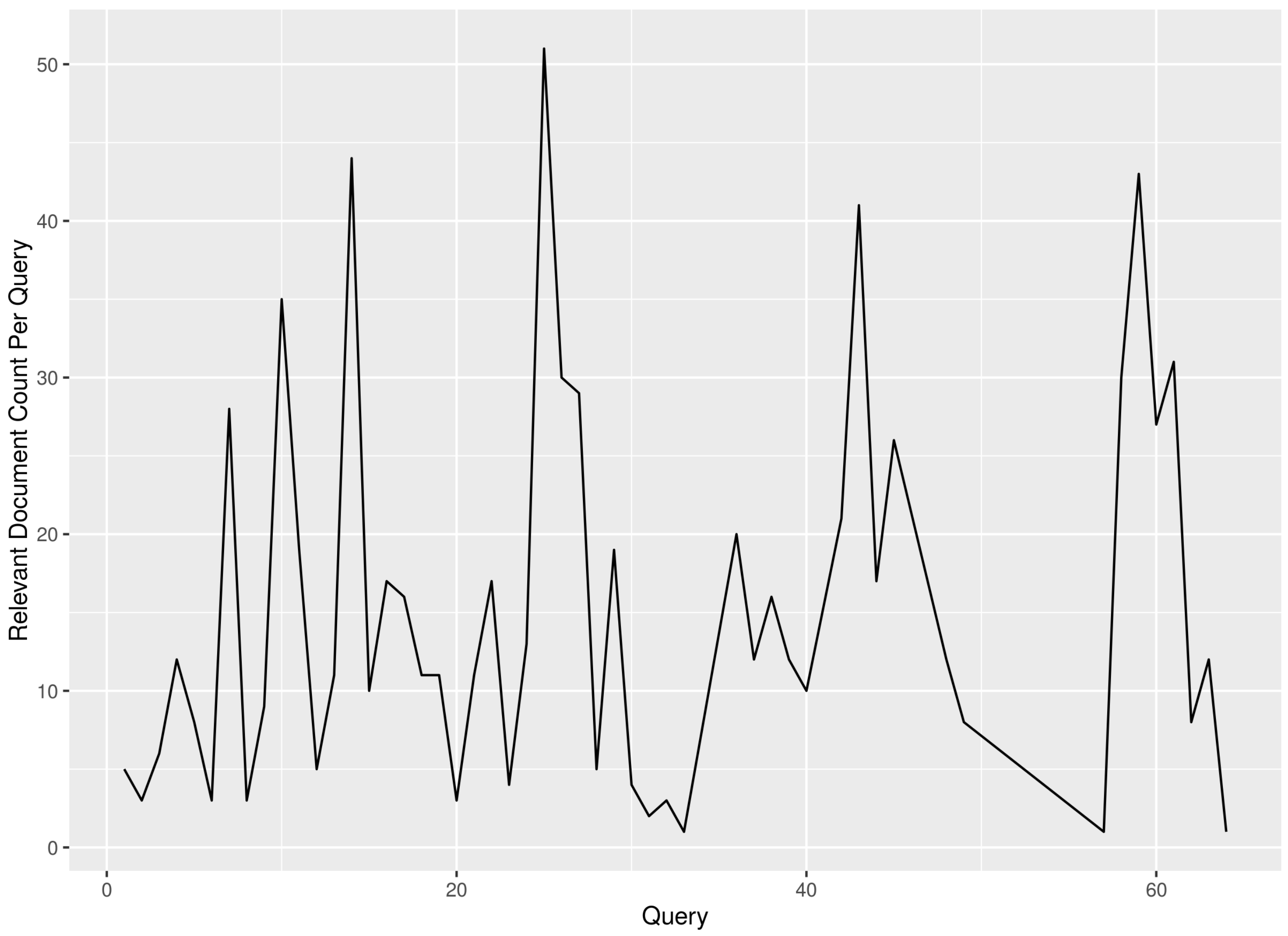


Figure 8: Relevant Document Count For CACM Queries

Source Code 2: Graph Data File Generator

```python
from assignmentFilePaths import *
from cam_q_utils import *
from contextClasses import SelectFromFile, AutoSaveCsv


def select_rel_results_save(relp, csvp):
    with SelectFromFile(relp, selector=rel_all_results_transform, transformer=rel_line_trans(1)) as rel_ret, \
            AutoSaveCsv(list, csvp, ['q', 'MAP', 'R-Prec', 'NDCG5', 'NDCG10', 'P10']) as out:
        for qret in rel_ret:
            out.append({
                'q': qret['q'],
                'MAP': sanitize_score(qret['scores']['map']),
                'R-Prec': sanitize_score(qret['scores']['R-prec']),
                'NDCG5': sanitize_score(qret['scores']['ndcg5']),
                'NDCG10': sanitize_score(qret['scores']['ndcg10']),
                'P10': sanitize_score(qret['scores']['P10'])
            })
            out.sort(key=lambda q: int(q['q']))


def count_relevant_docs():
    transformer1 = rel_line_trans(0)
    with SelectFromFile(cacm_rel, selector=lambda x: x, transformer=transformer1) as rel, \
            AutoSaveCsv(list, 'output_files/reldocs_count.csv', ['q', 'count']) as out:
        for doc, rels in sorted(rel.items(), key=lambda x: int(x[0])):
            print(doc, len(rels))
            out.append({
                'q': doc,
                'count': len(rels)
            })
            out.sort(key=lambda q: int(q['q']))


if __name__ == '__main__':
    print('generating the csv file for 10 requested documents for all cacm queries')
    select_rel_results_save(cacm_rel_out, cacm_all_rel_csv_reqnum % 10)
    count_relevant_docs()
```

Source Code 3: Graph Data File Generator Plotter

```
library(ggplot2)
library(reshape2)
library(RColorBrewer)

colors <- c('#1B9E77','#D95F02','#7570B3','#66A61E','#E6AB02','#A6761D','#666666')
scores <- melt(read.csv('output_files/q2_all_rel.csv'),id.vars = c('q'), measure.vars = c('MAP', 'NDCG5',
 ↪   'NDCG10', 'P10'))
ggplot(scores,aes(q,value,color=variable)) + geom_line() + scale_color_manual(values=colors) + labs(x = 'Query',
 ↪   y='Score',color='Metric')

ggsave('images/q2_plot.png')

# extra
metrics <- c('p','r')
scores <- read.csv('output_files/q3_rcompare_requested10.csv')
scores <- melt(scores,id.vars = c('q'), measure.vars = metrics)
ggplot(scores,aes(q,value,color=variable)) + geom_line() +
  scale_color_manual(values=colors,name ='Metric',breaks=c('p', 'r'),labels=c('Precision', 'Recall')) +
  labs(x = 'Query', y='Score',color='Metric')

ggsave('images/q2_plot_rp.png')

rel_count <-  read.csv('output_files/reldocs_count.csv')
ggplot(rel_count,aes(q,count)) + geom_line() +
  labs(x = 'Query', y='Relevant Document Count Per Query',color='Metric')

out <- capture.output(xtable::xtable(subset(rel_count,count < 10)))
cat(out, file="output_files/lt_10_table.txt", sep="\n", append=TRUE)
ggsave('images/q2_plot_rc.png')
```

Q.3 Question 8.7

```
Another measure that has been used in a number of evaluations is R-precision.
This is defined as the precision at R documents, where R is the number of relevant
documents for a query. It is used in situations where there is a large variation in
the number of relevant documents per query. Calculate the average R-precision
for the CACM query set and compare it to the other measures.
```

Q.3 Answer

Since galago computes the relevance metrics considering all relevant documents it is clear that the R-precision is one of the few metrics that would give an accurate results for all queries. This is due to it having the nice property of the R parameter which is the number of documents in the result set of a query which would be ten in this case. The code used to generate the data and the graphs can be seen in Source Code 4 and Source Code 5. As a more in depth look at what would be causing the graph to behave as it does was done in section Q.2 my answer for this question will be more general. Figure 9 shows the graph comparing R-Precision to the other metrics with loess smoothing applied in order to better compare these metrics. Also shown in this plot are all values for these metrics in the background in order to show the need for the smoothing as without it a comparison would be hard.

As was seen in the plots for section Q.2 the R-Precisions follows the same trend of dipping where the queries had below ten relevant documents or NaN values were computed. Unlike the other methods R-Precision appears to have less of a dramatic increase or decrease in its values.

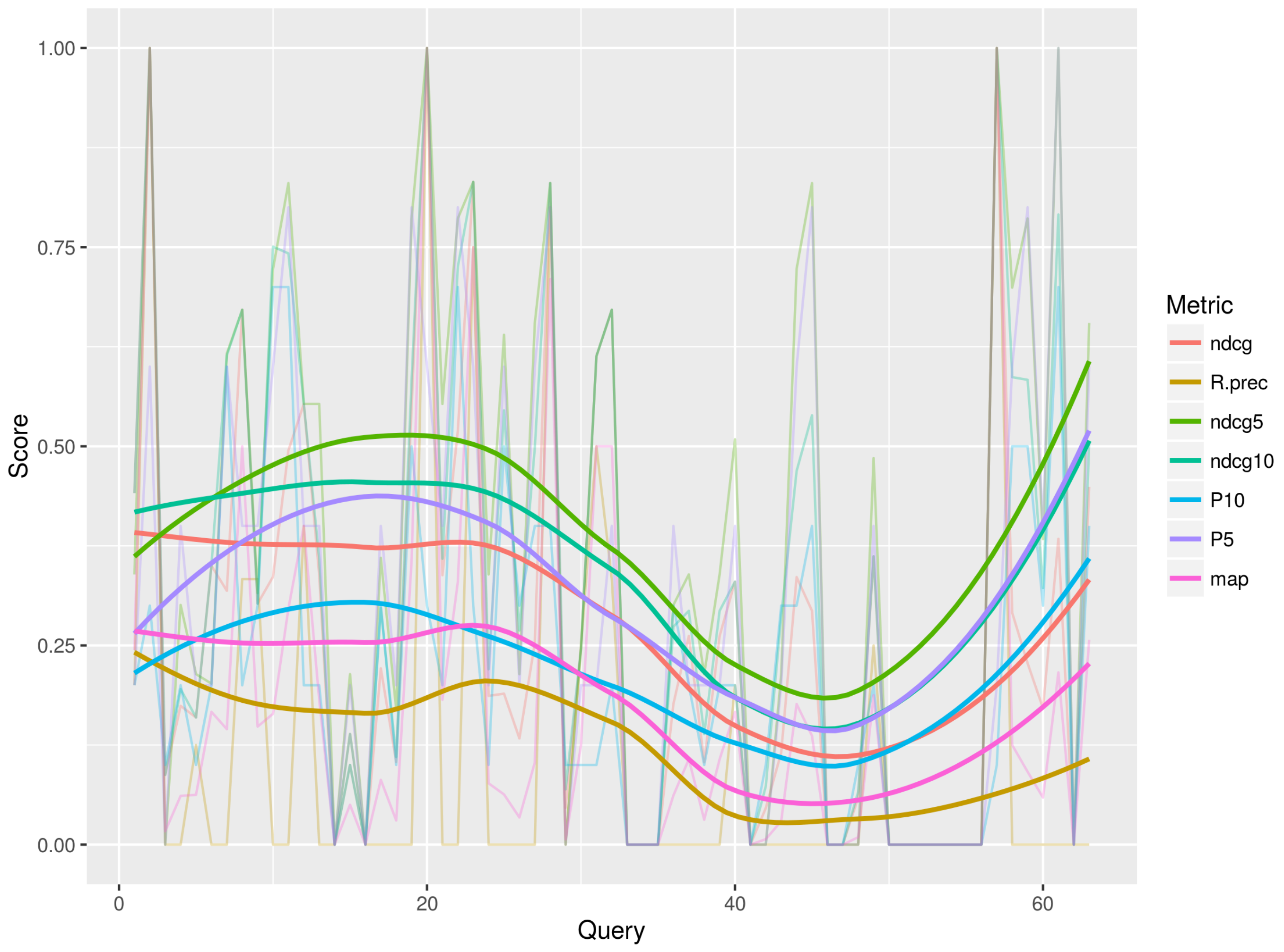


Figure 9: R-precision Comparison

Now to gain a better understanding of what is actually happening consider Figure 10 and Figure 11 which show the r values at 20 and 100 respectively.

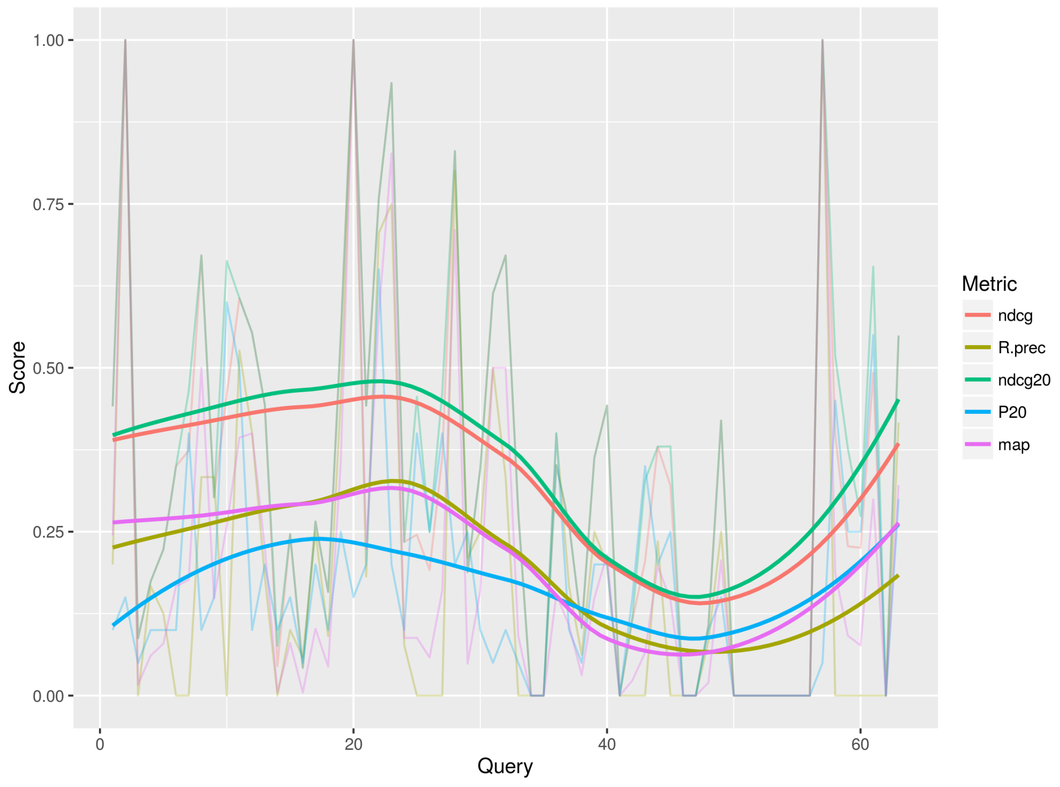


Figure 10: R-precision Comparison r=20

At r=20, Figure 10, the R-Precision increases to be on par with MAP and the P20 is mirrored in compairison to Figure 9. The ndcg20 value is greater than the overall ndcg for all queries which would be expected but why that is so will at the end of this section. At r=100, Figure 11, R-Precision and MAP show only a marginal increase but NDGC100 has the largest increase overall whereas P100 is almost 0.

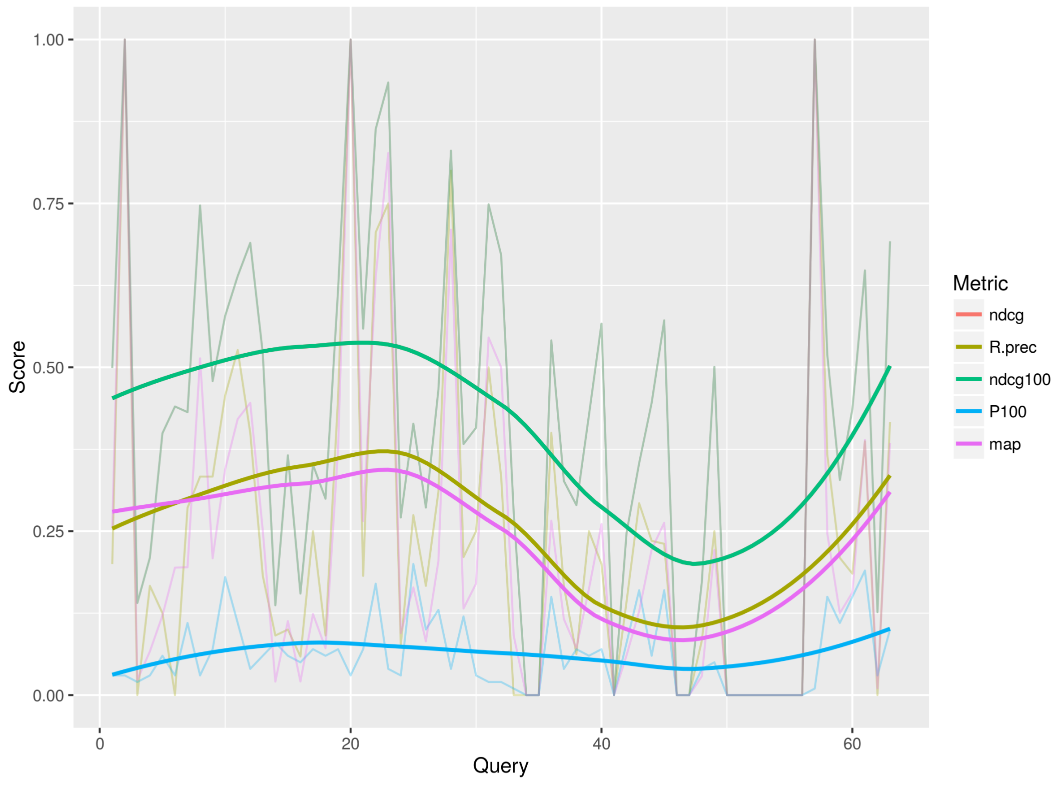


Figure 11: R-precision Comparison r=100

The only explanation for this can be found in the number of relevant documents greater than or equal to ten according to the *cam.rel* file. Table 2 shows these numbers and from the previous discussion about how galago computes these metrics it is clear to see that all relevant documents are taken into consideration for these scores.

Table 2: Relevant Document >= 10 Count

Query	Count
4	12
7	28
10	35
11	19
13	11
14	44
15	10
16	17
17	16
18	11
19	11
21	11
22	17
24	13
25	51
26	30
27	29
29	19
36	20
37	12
38	16
39	12
40	10
42	21
43	41
44	17
45	26
48	12
58	30
59	43
60	27
61	31
63	12

Source Code 4: Plot data Gen

```python
from assignmentFilePaths import *
from cam_q_utils import *
from contextClasses import SelectFromFile, AutoSaveCsv


def select_rel_results_save(relp, csvp):
    with SelectFromFile(relp, selector=rel_all_results_transform, transformer=rel_line_trans(1)) as rel_ret:
        keys = list(rel_ret[0]['scores'].keys())
        fields = ['q']
        fields.extend(keys)
        with AutoSaveCsv(list, csvp, fields) as out:
            for qret in rel_ret:
                qstats = { 'q': qret['q'] }
                for k in keys:
                    qstats[k] = sanitize_score(qret['scores'][k])
                out.append(qstats)
                out.sort(key=lambda q: int(q['q']))


if __name__ == '__main__':
    for requested in cacm_q_nums:
        print(
            'generating the csv file for R-precision comparison for %d requested documents for all cacm queries' %
            requested)
        csvp = cacm_all_rel_csv_rp_compare % requested
        if requested != 10:
            relp = cacm_rel_rnum_out % requested
        else:
            relp = cacm_rel_out
        select_rel_results_save(relp, csvp)
```

Source Code 5: Plot Generator

```r
library(ggplot2)
library(reshape2)
library(RColorBrewer)

metrics_all <-
 ↪  c('ndcg','R.prec','P500','ndcg5','ndcg1000','P30','ndcg10','p','P100','ndcg200','P10','ndcg30','P20','P5','ndcg100',
metrics <- c('ndcg','R.prec','ndcg5','ndcg10','P10','P5','map')
scores <- read.csv('output_files/q3_rcompare_requested10.csv')
scores <- melt(scores,id.vars = c('q'), measure.vars = metrics)
ggplot(scores,aes(q,value,color=variable)) +geom_line(alpha = 0.3) +  geom_smooth(se=F) + labs(x = "Query",
 ↪   y='Score',color='Metric')

ggsave('images/q3_plot.png')

metrics <- c('ndcg','R.prec','ndcg20','P20','map')
scores <- read.csv('output_files/q3_rcompare_requested20.csv')
scores <- melt(scores,id.vars = c('q'), measure.vars = metrics)
ggplot(scores,aes(q,value,color=variable)) +geom_line(alpha = 0.3) +  geom_smooth(se=F) + labs(x = "Query",
 ↪   y='Score',color='Metric')
ggsave('images/q3_plot_r20.png')

metrics <- c('ndcg','R.prec','ndcg100','P100','map')
scores <- read.csv('output_files/q3_rcompare_requested100.csv')
scores <- melt(scores,id.vars = c('q'), measure.vars = metrics)
ggplot(scores,aes(q,value,color=variable)) +geom_line(alpha = 0.3) +  geom_smooth(se=F) + labs(x = "Query",
 ↪   y='Score',color='Metric')
ggsave('images/q3_plot_r100.png')

rel_count <-  read.csv('output_files/reldocs_count.csv')

out <- capture.output(xtable::xtable(subset(rel_count,count >= 10)))
cat(out, file="output_files/gt_10_table.txt", sep="\n", append=TRUE)
```

Q.4 Question 9.6

```
Compare the accuracy of a one versus all SVM classifier and a one versus
one SVM classifier on a multiclass classification data set. Discuss any differences
observed in terms of the efficiency and effectiveness of the two approaches.
```

Q.4 Answer

This questions answer follows the example for SVMs on scikit-learns website as it provides a python interface for both liblinear and libsvm. Dataset used is the publicly available iris dataset which comes packaged with sciki-learn. The code used for this question can be seen in Source Code 6 and uses two different linear SVM implementations SVC(libsvm) and LinearSVC(liblinear). The SVC or SVM class C utilizes a one-vs-one for multiclass and LinearSVC a one-vs-all for multiclass. Also in order to ensure these svm implementations did the correct one-vs-one and one-vs-all classification on the iris dataset they were wrapped in a multiclass classification strategy helper for the respective one-vs-x classification.

Figure 12 shows the partitioning done by the classifiers on the iris dataset where the color represent the iris type. The one-vs-one strategy created partitions that encompassed the iris types evenly whereas the one-vs-all strategy took more from the light blue iris type. Also one-vs-all did not include the single dark blue iris type in its true class rather classified it as light blue.

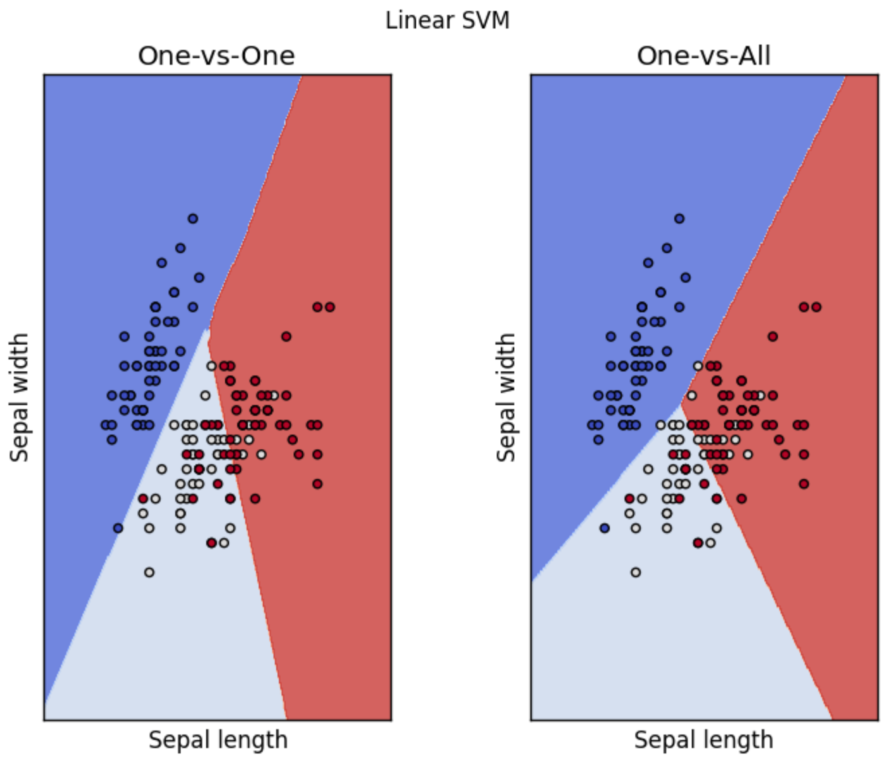


Figure 12: 1v1 and 1vAll Classification Comparison

Figure 13 shows the confusion matrix for the one-vs-one strategy and the color represents the number of iris types classified for a label. As shown in the confusion matrices the one vs one strategy did marginally better than the one vs all strategy for versicolor. Table 3 which shows the classification report for one-vs-one and Table 4 shows the classification report for one-vs-all also reflect that one-vs-one performed better. The support for both strategies are the same which leads me to conclude that using one-vs-one vs one-vs-all depends on the classification task at hand. This toy example highlighted the one-vs-one performance gains as the iris types are distributed close together for the outliers that when using the one-vs-all strategy it would pick up on this and wrongly classify iris types when they are extremely close together.

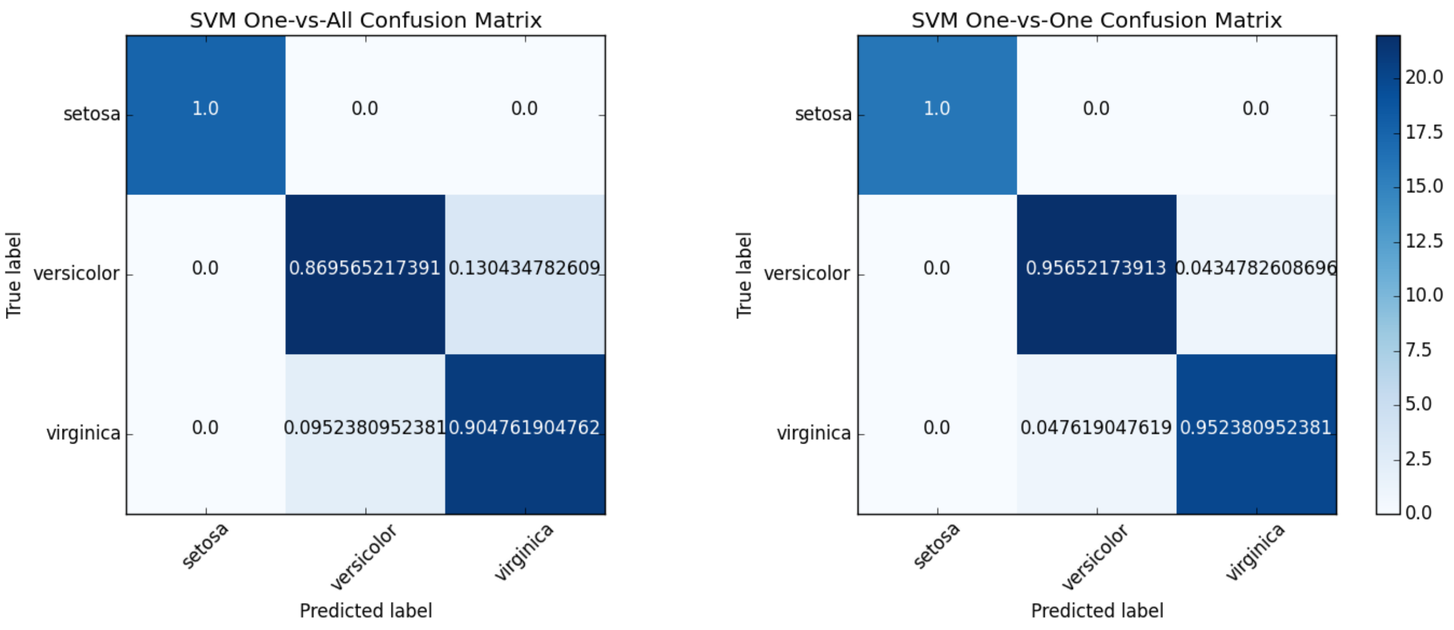


Figure 13: Classification Strategy Confusion Matrices

Class	Precision	Recall	F-score	Support
setosa	1.0	1.0	1.0	16
versicolor	0.96	0.96	0.96	23
virginica	0.95	0.95	0.95	21
avg	0.97	0.97	0.97	60

Table 3: Classification Report for One-vs-One

Class	Precision	Recall	F-score	Support
setosa	1.0	1.0	1.0	16
versicolor	0.91	0.87	0.89	23
virginica	0.86	0.9	0.88	21
avg	0.92	0.92	0.92	60

Table 4: Classification Report for One-vs-All

Source Code 6: svm code

```
import numpy as np
import matplotlib.pyplot as plt
import itertools
from collections import OrderedDict
from sklearn import datasets
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier


def parse_classification_report(clfreport):
    # https://gist.github.com/julienr/6b9b9a03bd8224db7b4f
    """
    Parse a sklearn classification report into a dict keyed by class name
```

```python
    and containing a tuple (precision, recall, fscore, support) for each class
    """
    lines = clfreport.split('\n')
    # Remove empty lines
    lines = list(filter(lambda l: not len(l.strip()) == 0, lines))

    # Starts with a header, then score for each class and finally an average
    header = lines[0]
    cls_lines = lines[1:-1]
    avg_line = lines[-1]

    assert header.split() == ['precision', 'recall', 'f1-score', 'support']
    assert avg_line.split()[0] == 'avg'

    # We cannot simply use split because class names can have spaces. So instead
    # figure the width of the class field by looking at the indentation of the
    # precision header
    cls_field_width = len(header) - len(header.lstrip())

    # Now, collect all the class names and score in a dict
    def parse_line(l):
        """Parse a line of classification_report"""
        cls_name = l[:cls_field_width].strip()
        precision, recall, fscore, support = l[cls_field_width:].split()
        precision = float(precision)
        recall = float(recall)
        fscore = float(fscore)
        support = int(support)
        return cls_name, precision, recall, fscore, support

    data = OrderedDict()
    for l in cls_lines:
        ret = parse_line(l)
        cls_name = ret[0]
        scores = ret[1:]
        data[cls_name] = scores

    # average
    data['avg'] = parse_line(avg_line)[1:]

    return data


def report_to_latex_table(clfreport):
    data = parse_classification_report(clfreport)
    out = ""
    out += "\\begin{tabular}{c | c c c c}\n"
    out += "Class & Precision & Recall & F-score & Support\\\\\n"
    out += "\hline\n"
    out += "\hline\\\\\n"
    for cls, scores in data.items():
        out += cls + " & " + " & ".join([str(s) for s in scores])
        out += "\\\\\n"
    out += "\\end{tabular}"
    return out


def plot_classification():
    # modified
    ↪   http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html#sphx-glr-auto-examples-svm-plot-iris-py
    iris = datasets.load_iris()
    X = iris.data[:, :2] # only want  to classify what type of iris it is
    y = iris.target
    C = 1.0  # SVM regularization parameter
    h = .02  # step size in the mesh

    svm_c = OneVsOneClassifier(SVC(kernel='linear', C=C))
    svm_l = OneVsRestClassifier(LinearSVC(C=C))

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    titles = ['One-vs-One', 'One-vs-All']

    plt.suptitle('Linear SVM')

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    plt.subplot(1, 2, 0 + 1)
    plt.subplots_adjust(wspace=0.4, hspace=0.4)

    Z_c = svm_c.fit(X, y).predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z_c = Z_c.reshape(xx.shape)
    plt.contourf(xx, yy, Z_c, cmap=plt.cm.coolwarm, alpha=0.8)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
    plt.xlabel('Sepal length')
    plt.ylabel('Sepal width')
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xticks(())
    plt.yticks(())
    plt.title(titles[0])

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    plt.subplot(1, 2, 1 + 1)
    plt.subplots_adjust(wspace=0.4, hspace=0.4)

    Z_l = svm_l.fit(X, y).predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z_l = Z_l.reshape(xx.shape)
    plt.contourf(xx, yy, Z_l, cmap=plt.cm.coolwarm, alpha=0.8)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
    plt.xlabel('Sepal length')
    plt.ylabel('Sepal width')
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xticks(())
    plt.yticks(())
    plt.title(titles[1])

    plt.savefig('images/svm_linear_1v1_1va.png')
    plt.close()


def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion matrix', cmap=plt.cm.Blues,showC=True):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    if showC:
        plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print(title)
```

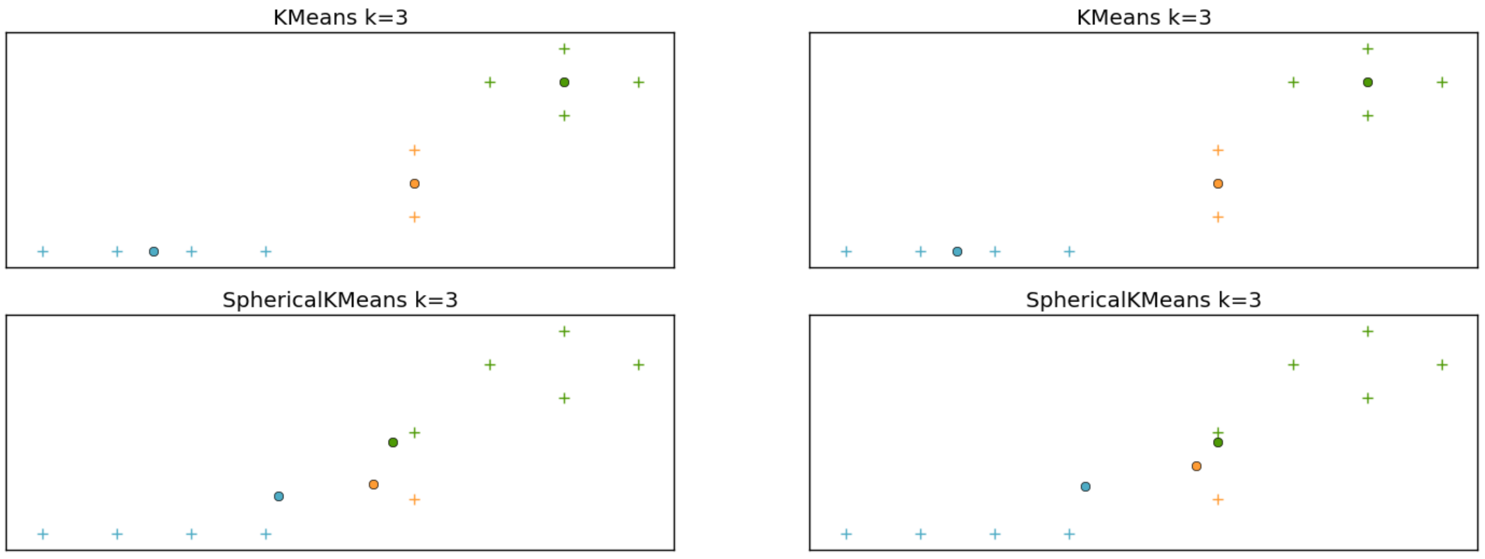
```python
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


def evaluation():
    iris = datasets.load_iris()
    class_names = iris.target_names
    C = 1.0  # SVM regularization parameter
    X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.4, random_state=0)

    svm_c = OneVsOneClassifier(SVC(kernel='linear', C=C))
    y_pred = svm_c.fit(X_train, y_train).predict(X_test)

    with open('output_files/svm_1v1_classification_report.tex', 'w') as out:
        out.write('\\begin{table}\n')
        out.write(report_to_latex_table(classification_report(y_test, y_pred, target_names=iris.target_names)))
        out.write('\n\\end{table}\n')

    # Compute confusion matrix
    cnf_matrix = confusion_matrix(y_test, y_pred)
    np.set_printoptions(precision=2)

    # Plot normalized confusion matrix
    plt.figure()
    plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                          title='SVM One-vs-One Confusion Matrix')
    plt.tight_layout()
    plt.savefig('images/svm_linear_1v1_cm.png')
    plt.close()

    svm_l = OneVsRestClassifier(LinearSVC(C=C))
    y_pred = svm_l.fit(X_train, y_train).predict(X_test)
    with open('output_files/svm_1va_classification_report.tex', 'w') as out:
        out.write('\\begin{table}\n')
        out.write(report_to_latex_table(classification_report(y_test, y_pred, target_names=iris.target_names)))
        out.write('\n\\end{table}\n')

    # Compute confusion matrix
    cnf_matrix = confusion_matrix(y_test, y_pred)
    np.set_printoptions(precision=2)

    # Plot normalized confusion matrix
    plt.figure()
    plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                          title='SVM One-vs-All Confusion Matrix',showC=False)

    plt.tight_layout()
    plt.savefig('images/svm_linear_1va_cm.png')
    plt.close()


if __name__ == '__main__':
    plot_classification()
    evaluation()
```

Q.5 Question 9.8

Use K-means and spherical K-means to cluster the data points in Exercise 9.8. How do the clusterings differ?

Q.5 Answer

Scikit learn was also used to answer this question as well as an extension of its kmeans class SphericalKMeans provided by spherecluster which extends scikit learns version to do the sphericalkmeans. The points from 9.8 are $(-4,-2),(-3,-2),(-2,-2),(-1,-2),(1,-1),(1,1),(2,3),(3,2),(3,4),(4,3)$ and the code used to answer this question can be seen in Source Code 7. The k value chosen was 3 due to the sparse nature of the points which can be seen in Figure 14. Figure 14(a) which shows the kmeans algorithm after the first iteration and Figure 14(b) shows the kmeans algorithm after running to completion. The biggest difference seen in Figure 14 is how the centroids are placed even after the first iteration. Kmeans drops the points directly in the middle of the clusters whereas sphericalkmeans puts them in the middle and works outwards in a spiral. I believe this is happening due to the n_init parameter which runs the algorithm with n different initial seeds after which it chooses the best.



(a) One Iteration

(b) Run To Completion

Figure 14: Classification Strategy Confusion Matrices

Source Code 7: kmeans code

```python
import numpy as np
from matplotlib import pyplot as plt
from sklearn.metrics.pairwise import pairwise_distances_argmin
from sklearn.cluster import KMeans
from spherecluster import SphericalKMeans

def one_iter():
    f, (ax1, ax2) = plt.subplots(2, sharex=True, sharey=True)
    colors = ['#4EACC5', '#FF9C34', '#4E9A06']
    X = np.array([[-4, -2], [-3, -2], [-2, -2], [-1, -2], [1, -1], [1, 1], [2, 3], [3, 2], [3, 4], [4, 3]])
    print('data to cluster =', X, '\n')
    print('executing Kmeans with 3 clusters')
    km = KMeans(n_clusters=3, init='k-means++', n_init=20, max_iter=1)
    km.fit(X)
    print('executing SphericalKMeans with 3 clusters')
    skm = SphericalKMeans(n_clusters=3, init='k-means++', n_init=20, max_iter=1)
    skm.fit(X)
    print('plotting the results')
    k_means_cluster_centers = np.sort(km.cluster_centers_, axis=0)
    skm_means_cluster_centers = np.sort(skm.cluster_centers_, axis=0)
    k_means_labels = pairwise_distances_argmin(X, k_means_cluster_centers)
    skm_means_labels = pairwise_distances_argmin(X, skm_means_cluster_centers)

    # KMeans
    for k, col in zip(range(3), colors):
        my_members = k_means_labels == k
        cluster_center = k_means_cluster_centers[k]
        ax1.scatter(X[my_members, 0], X[my_members, 1], marker='+', c=col, s=60)
        ax1.plot(cluster_center[0], cluster_center[1], marker='o', markerfacecolor=col,
                 markeredgecolor='k', markersize=6)
    ax1.set_title('KMeans k=3')
    ax1.set_xticks(())
    ax1.set_yticks(())

    # SphericalKMeans
    for k, col in zip(range(3), colors):
        my_members = skm_means_labels == k
        cluster_center = skm_means_cluster_centers[k]
        ax2.scatter(X[my_members, 0], X[my_members, 1], marker='+', c=col, s=60)
        ax2.plot(cluster_center[0], cluster_center[1], marker='o', markerfacecolor=col,
                 markeredgecolor='k', markersize=6)
    ax2.set_title('SphericalKMeans k=3')
    ax2.set_xticks(())
    ax2.set_yticks(())
    # plt.show()
    f.savefig('images/kmeans_vs_sphericalkmean_iter.png')
    plt.close(f)
    print('the plot of the results is located at images/kmeans_vs_sphericalkmean.png')


def full_iter():
    f, (ax1, ax2) = plt.subplots(2, sharex=True, sharey=True)
    colors = ['#4EACC5', '#FF9C34', '#4E9A06']
    X = np.array([[-4, -2], [-3, -2], [-2, -2], [-1, -2], [1, -1], [1, 1], [2, 3], [3, 2], [3, 4], [4, 3]])
    print('data to cluster =', X, '\n')
    print('executing Kmeans with 3 clusters')
    km = KMeans(n_clusters=3, init='k-means++', n_init=20, max_iter=3000)
    km.fit(X)
    print('executing SphericalKMeans with 3 clusters')
    skm = SphericalKMeans(n_clusters=3, init='k-means++', n_init=20, max_iter=3000)
    skm.fit(X)

    print('plotting the results')
    k_means_cluster_centers = np.sort(km.cluster_centers_, axis=0)
    skm_means_cluster_centers = np.sort(skm.cluster_centers_, axis=0)
    k_means_labels = pairwise_distances_argmin(X, k_means_cluster_centers)
    skm_means_labels = pairwise_distances_argmin(X, skm_means_cluster_centers)

    # KMeans
    for k, col in zip(range(3), colors):
        my_members = k_means_labels == k
        cluster_center = k_means_cluster_centers[k]
        ax1.scatter(X[my_members, 0], X[my_members, 1], marker='+', c=col, s=60)
        ax1.plot(cluster_center[0], cluster_center[1], marker='o', markerfacecolor=col,
                 markeredgecolor='k', markersize=6)
    ax1.set_title('KMeans k=3')
    ax1.set_xticks(())
    ax1.set_yticks(())

    # SphericalKMeans
    for k, col in zip(range(3), colors):
        my_members = skm_means_labels == k
        cluster_center = skm_means_cluster_centers[k]
        ax2.scatter(X[my_members, 0], X[my_members, 1], marker='+', c=col, s=60)
        ax2.plot(cluster_center[0], cluster_center[1], marker='o', markerfacecolor=col,
                 markeredgecolor='k', markersize=6)
    ax2.set_title('SphericalKMeans k=3')
    ax2.set_xticks(())
    ax2.set_yticks(())
    # plt.show()
    f.savefig('images/kmeans_vs_sphericalkmean.png')
    plt.close(f)
    print('the plot of the results is located at images/kmeans_vs_sphericalkmean.png')

if __name__ == '__main__':
    one_iter()
    full_iter()
```

Source Code 8: Ranking methods found on gist

```python
"""
from https://gist.github.com/bwhite/3726239 circa 2012
"""
import numpy as np


def mean_reciprocal_rank(rs):
    """Score is reciprocal of the rank of the first relevant item

    First element is 'rank 1'.  Relevance is binary (nonzero is relevant).

    Example from http://en.wikipedia.org/wiki/Mean_reciprocal_rank
    >>> rs = [[0, 0, 1], [0, 1, 0], [1, 0, 0]]
    >>> mean_reciprocal_rank(rs)
    0.61111111111111105
    >>> rs = np.array([[0, 0, 0], [0, 1, 0], [1, 0, 0]])
    >>> mean_reciprocal_rank(rs)
    0.5
    >>> rs = [[0, 0, 0, 1], [1, 0, 0], [1, 0, 0]]
    >>> mean_reciprocal_rank(rs)
    0.75

    Args:
        rs: Iterator of relevance scores (list or numpy) in rank order
            (first element is the first item)

    Returns:
        Mean reciprocal rank
    """
    rs = (np.asarray(r).nonzero()[0] for r in rs)
    return np.mean([1. / (r[0] + 1) if r.size else 0. for r in rs])


def r_precision(r):
    """Score is precision after all relevant documents have been retrieved

    Relevance is binary (nonzero is relevant).

    >>> r = [0, 0, 1]
    >>> r_precision(r)
    0.33333333333333331
    >>> r = [0, 1, 0]
    >>> r_precision(r)
    0.5
    >>> r = [1, 0, 0]
    >>> r_precision(r)
    1.0

    Args:
        r: Relevance scores (list or numpy) in rank order
            (first element is the first item)

    Returns:
        R Precision
    """
    r = np.asarray(r) != 0
    z = r.nonzero()[0]
    if not z.size:
        return 0.
    return np.mean(r[:z[-1] + 1])


def precision_at_k(r, k):
    """Score is precision @ k

    Relevance is binary (nonzero is relevant).
    >>> r = [0, 0, 1]
    >>> precision_at_k(r, 1)
    0.0
    >>> precision_at_k(r, 2)
    0.0
    >>> precision_at_k(r, 3)
    0.33333333333333331
    >>> precision_at_k(r, 4)
    Traceback (most recent call last):
        File "<stdin>", line 1, in ?
    ValueError: Relevance score length < k


    Args:
        r: Relevance scores (list or numpy) in rank order
            (first element is the first item)

    Returns:
        Precision @ k

    Raises:
        ValueError: len(r) must be >= k
    """
    assert k >= 1
    r = np.asarray(r)[:k] != 0
    if r.size != k:
        raise ValueError('Relevance score length < k')
    return np.mean(r)


def average_precision(r):
    """Score is average precision (area under PR curve)

    Relevance is binary (nonzero is relevant).

    >>> r = [1, 1, 0, 1, 0, 1, 0, 0, 0, 1]
    >>> delta_r = 1. / sum(r)
    >>> sum([sum(r[:x + 1]) / (x + 1.) * delta_r for x, y in enumerate(r) if y])
    0.7833333333333333
    >>> average_precision(r)
    0.78333333333333333

    Args:
        r: Relevance scores (list or numpy) in rank order
            (first element is the first item)

    Returns:
        Average precision
    """
    r = np.asarray(r) != 0
    out = [precision_at_k(r, k + 1) for k in range(r.size) if r[k]]
    if not out:
        return 0.
    return np.mean(out)


def mean_average_precision(rs):
    """Score is mean average precision

    Relevance is binary (nonzero is relevant).

    >>> rs = [[1, 1, 0, 1, 0, 1, 0, 0, 0, 1]]
    >>> mean_average_precision(rs)
    0.78333333333333333
    >>> rs = [[1, 1, 0, 1, 0, 1, 0, 0, 0, 1], [0]]
    >>> mean_average_precision(rs)
    0.39166666666666666

    Args:
        rs: Iterator of relevance scores (list or numpy) in rank order
            (first element is the first item)

    Returns:
        Mean average precision
    """
    return np.mean([average_precision(r) for r in rs])


def dcg_at_k(r, k, method=0):
    """Score is discounted cumulative gain (dcg)

    Relevance is positive real values.  Can use binary
    as the previous methods.

    Example from
    http://www.stanford.edu/class/cs276/handouts/EvaluationNew-handout-6-per.pdf
    >>> r = [3, 2, 3, 0, 0, 1, 2, 2, 3, 0]
    >>> dcg_at_k(r, 1)
    3.0
    >>> dcg_at_k(r, 1, method=1)
    3.0
    >>> dcg_at_k(r, 2)
    5.0
    >>> dcg_at_k(r, 2, method=1)
    4.2618595071429155
    >>> dcg_at_k(r, 10)
    9.6051177391888114
    >>> dcg_at_k(r, 11)
    9.6051177391888114

    Args:
        r: Relevance scores (list or numpy) in rank order
            (first element is the first item)
        k: Number of results to consider
        method: If 0 then weights are [1.0, 1.0, 0.6309, 0.5, 0.4307, ...]
                If 1 then weights are [1.0, 0.6309, 0.5, 0.4307, ...]

    Returns:
        Discounted cumulative gain
    """
    r = np.asfarray(r)[:k]
    if r.size:
        if method == 0:
            return r[0] + np.sum(r[1:] / np.log2(np.arange(2, r.size + 1)))
        elif method == 1:
            return np.sum(r / np.log2(np.arange(2, r.size + 2)))
        else:
            raise ValueError('method must be 0 or 1.')
    return 0.


def ndcg_at_k(r, k, method=0):
    """Score is normalized discounted cumulative gain (ndcg)

    Relevance is positive real values.  Can use binary
    as the previous methods.

    Example from
    http://www.stanford.edu/class/cs276/handouts/EvaluationNew-handout-6-per.pdf
    >>> r = [3, 2, 3, 0, 0, 1, 2, 2, 3, 0]
    >>> ndcg_at_k(r, 1)
    1.0
    >>> r = [2, 1, 2, 0]
    >>> ndcg_at_k(r, 4)
    0.9203032077642922
    >>> ndcg_at_k(r, 4, method=1)
    0.96519546960144276
    >>> ndcg_at_k([0], 1)
    0.0
    >>> ndcg_at_k([1], 2)
    1.0

    Args:
        r: Relevance scores (list or numpy) in rank order
            (first element is the first item)
        k: Number of results to consider
        method: If 0 then weights are [1.0, 1.0, 0.6309, 0.5, 0.4307, ...]
                If 1 then weights are [1.0, 0.6309, 0.5, 0.4307, ...]

    Returns:
        Normalized discounted cumulative gain
    """
    dcg_max = dcg_at_k(sorted(r, reverse=True), k, method)
    if not dcg_max:
        return 0.
    return dcg_at_k(r, k, method) / dcg_max


if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Source Code 9: Context Helper Classes

```
import csv
from bs4 import BeautifulSoup
from functional import seq
from random import sample, shuffle


def build_lstransformer(func, seq_to='list'):
    if seq_to == 'list':
        return lambda ilist: seq(ilist).group_by(func).to_list()
    else:
        return lambda ilist: seq(ilist).group_by(func).to_dict()


def ltransformer_group_by(func):
    return lambda ilist: seq(ilist).group_by(func).to_list()


def default_ltransformer():
    return lambda ilist: ilist


def default_random_selector():
    return lambda ilist: sample(ilist, 1)[0]


def selected_list_joiner(selected, out):
    out.extend(selected)


def build_selected_joiner(func):
    def joiner(selected, out):
        return func((selected, out))

    return joiner


def default_formatter(item):
    return '%s\n' % item


def int_formatter(i):
    return '%d\n' % i


def build_formatter(format_s):
    return lambda item: format_s % item


class BeautifulSoupFromFile(object):
    def __init__(self, file_p, mode):
        self.file_p = file_p
        self.file_obj = open(file_p, 'r')
        self.mode = mode
        self.soup = None

    def __enter__(self):
        self.soup = BeautifulSoup(self.file_obj.read(), self.mode)
        return self.soup

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.file_obj.close()


class RunCommandSaveOut(object):
    def __init__(self, cline, file_p, print_file=False, command_fun=None):
        self.cline = cline
        self.file_p = file_p
        self.file_obj = open(file_p, 'w')
        self.print_file = print_file
        if command_fun is None:
            raise Exception('RunCommandSaveOut the command_fun was None')
        self.command_fun = command_fun

    def __enter__(self):
        spawn_process = self.command_fun(self.cline, self.file_obj)
        return spawn_process

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.file_obj.close()
        if self.print_file:
            with FileLinePrinter(self.file_p):
                one = 1


class AutoSaver(object):
    def __init__(self, save_type, file_name, formatter=default_formatter, selector=None):
        self.file_name = file_name
        self.save_type = save_type()
        self.formatter = formatter
        self.selector = selector

    def __enter__(self):
        return self.save_type

    def __exit__(self, exc_type, exc_val, exc_tb):
        if self.selector is None:
            outl = self.save_type
        else:
            outl = self.selector(self.save_type)
        with open(self.file_name, 'w') as out:
            for it in outl:
                out.write(self.formatter(it))


class AutoSaveCsv(object):
    def __init__(self, save_type, file_name, field_names):
        self.file_name = file_name
        self.save_type = save_type()
        self.field_names = field_names

    def __enter__(self):
        return self.save_type

    def __exit__(self, exc_type, exc_val, exc_tb):
        with open(self.file_name, 'w') as out:
            writer = csv.DictWriter(out, fieldnames=self.field_names)
            writer.writeheader()
            for it in self.save_type:
                writer.writerow(it)


class SelectFromFile(object):
    def __init__(self, q_file, selector, transformer=default_ltransformer()):
        self.q_file = open(q_file, 'r')
        self.line_transformer = transformer
        self.selector = selector

    def __enter__(self):
        lines = self.line_transformer(self.q_file.readlines())
        return self.selector(lines)

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.q_file.close()


class RandFinderFile(object):
    def __init__(self, q_file, transformer=default_ltransformer(), selector=default_random_selector()):
        self.q_file = open(q_file, 'r')
        self.line_transformer = transformer
        self.random_selector = selector

    def __enter__(self):
        lines = self.line_transformer(self.q_file.readlines())
        return self.random_selector(lines)

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.q_file.close()


class RandFinderSaver(RandFinderFile):
    def __init__(self, q_file, save_p, save_type, out_adder, formatter=default_formatter, sselector=None,
                 transformer=default_ltransformer(), selector=default_random_selector()):
        super(RandFinderSaver, self).__init__(q_file=q_file, transformer=transformer, selector=selector)
        self.selected = None
        self.save_p = save_p
        self.save_type = save_type
        self.formatter = formatter
        self.sselector = sselector
        self.out_adder = out_adder

    def __enter__(self):
        self.selected = super(RandFinderSaver, self).__enter__()
        return self.selected

    def __exit__(self, exc_type, exc_val, exc_tb):
        super(RandFinderSaver, self).__exit__(exc_tb, exc_val, exc_tb)
        with AutoSaver(self.save_type, self.save_p, formatter=self.formatter, selector=self.sselector) as out:
            self.out_adder(self.selected, out)


class FileLinePrinter(object):
    def __init__(self, file_p, line_transformer=lambda line: line.rstrip().lstrip()):
        self.file_p = open(file_p, 'r')
        self.line_transformer = line_transformer

    def __enter__(self):
        for line in map(self.line_transformer, self.file_p):
            print(line)
        return None

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.file_p.close()


class CleanFileLines(object):
    def __init__(self, file_p, remove_fun, save_back=False, sort_output=lambda x: x):
        self.file_p = file_p
        self.file_obj = open(file_p, 'r')
        self.remove_fun = remove_fun
        self.clean_lines = []
        self.save_back = save_back
        self.sort_output = sort_output

    def __enter__(self):
        for line in self.sort_output(self.file_obj):
            self.clean_lines.append(self.remove_fun(line))
        return self.clean_lines

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.file_obj.close()
        if self.save_back:
            with AutoSaver(list, self.file_p) as out:
                out.extend(self.clean_lines)
```

Source Code 10: CACM Helper Functions

```python
import os
import re
from contextClasses import build_lstransformer

cacm_q_nums = [10, 20, 50, 100, 1000]


def sanitize_score(score):
    if score == 'NaN':
        return 0.0
    else:
        return score


def q_result_cleaner():
    camp = os.path.join(os.getcwd(), 'cacm/')
    return lambda fline: fline.rstrip().lstrip().replace(camp, '').replace('galago', '').replace('.html', '')


def q_results_sorter(file_obj):
    for line in sorted(file_obj, key=lambda fline: int(fline.split(' ')[0])):
        yield line


def is_good(rel, rel_ret):
    return not (rel is None or rel_ret['num_rel_ret'] == '0.00000' or rel_ret['num_rel_ret'] == 'NaN')


def rel_line_trans(key_where):
    def split_trans(flines):
        splitter = re.compile('\s+')
        transformer = build_lstransformer(lambda line: splitter.split(line)[key_where], seq_to='dict')
        return transformer(map(lambda x: x.rstrip().lstrip(), flines))

    return split_trans


def rel_docs_transform(num):
    def trans(gdict):
        results = gdict.get(num, None)
        if results is not None:
            rets = []
            for ret in results:
                qn, q0, doc, _ = ret.split(' ')
                rets.append(doc)
            return rets
        else:
            return None

    return trans


def rel_results_transform(num):
    def trans(gdict):
        splitter = re.compile('\s+')
        results = gdict.get(num, None)
        if results is not None:
            rets = {}
            for ret in results:
                method, doc, val = splitter.split(ret)
                rets[method] = val
            return rets
        else:
            return None

    return trans
```

```
def rel_all_results_transform(gdict):
    ret_list = []
    splitter = re.compile('\s+')
    for q, vals in gdict.items():
        ret = {}
        for val in vals:
            method, doc, score = splitter.split(val)
            ret[method] = score
        ret_list.append({'q': q, 'scores': ret})
    return ret_list
```

Source Code 11: General Utility Functions

```
import json
import os
import math
from subprocess import Popen, PIPE
from shlex import split
from itertools import islice
from assignmentFilePaths import cacm_qs, cacm_q_xml
from contextClasses import BeautifulSoupFromFile


def ordinal(n):
    return "%d%s" % (n, "tsnrhtdd"[(math.floor(n / 10) % 10 != 1) * (n % 10 < 4) * n % 10::4])


def join_with_cwd(name):
    return os.path.join(os.getcwd(), name)


def file_name(file_p):
    return os.path.basename(file_p)


def select_n(ilist, n=10):
    return list(islice(ilist, n))


def path_exists(path):
    return os.path.exists(path)


def run_galago(cline, stdout=PIPE, stderror=PIPE):
    return Popen(split(cline), stdout=stdout, stderr=stderror)


def rand_select_stripper(ilist):
    return map(lambda s: s.lstrip().rstrip(), ilist)


def forward_galago_output(cline, outp):
    return run_galago(cline, stdout=outp)


def galago_mk_index(idxp, inp):
    if not path_exists(idxp):
        cline = './rungalago.sh build %s %s' % (idxp, inp)
        return run_galago(cline=cline)
    else:
        return None


def galago_search(qpath, idxpath, requested=10, retpath=None):
    cline = './rungalago.sh threaded-batch-search --requested=%d --index=%s %s' % (requested, idxpath, qpath)
    if retpath is not None:
        with open(retpath, 'w') as retOut:
            runner = run_galago(cline=cline, stdout=retOut)
            stdout, stderr = runner.communicate()
            return runner.returncode
    else:
        return run_galago(cline=cline)


def cam_xmlqs_to_json(idxp):
    with BeautifulSoupFromFile(cacm_q_xml, mode='xml') as soup:
        orginal_queries = soup.find_all('query')
        updated_queries = []
        with open(cacm_qs, 'w') as camqjson:
            for xmlq in orginal_queries:
                updated_queries.append({
                    'number': xmlq.number.text.strip(),
                    'text': xmlq.find('text').text.lstrip().rstrip()
                })
            json.dump({
                'queries': updated_queries,
                'index': idxp,
                'queryType': 'simple'
            }, camqjson, indent=2)
```

Source Code 12: Run Galago Bash File

```
#!/usr/bin/env bash

if [ $# -eq 0 ]; then
   ./galago-3.10/contrib/target/appassembler/bin/galago --help
elif [ "$1" = "help" ]; then
 ./galago-3.10/contrib/target/appassembler/bin/galago --help
elif [ "$1" = "build" ]; then
shift
 ./galago-3.10/contrib/target/appassembler/bin/galago build --nonStemmedPosting=true --stemmedPostings=false \
  --corpus=true --indexPath=$1 --inputPath+$2
elif [ "$1" = "dump-idx" ]; then
shift
./galago-3.10/contrib/target/appassembler/bin/galago dump-index $@
elif [ "$1" = "search" ]; then
shift
./galago-3.10/contrib/target/appassembler/bin/galago search --port=9000 $@
elif [ "$1" = "build-window" ]; then
shift
./galago-3.10/contrib/target/appassembler/bin/galago build-window --width=$1 --ordered=$2 \
 --stemming=false --spaceEfficiecent=true  --inputPath=$3 --indexPath=$4
elif [ "$1" = "eval" ]; then
shift
 ./galago-3.10/contrib/target/appassembler/bin/galago eval --judgments=$1 --baseline=$2  --details=true
  ↪  --summary=false \
 --metrics+num_ret --metrics+num_rel --metrics+num_rel_ret --metrics+num_unjug_ret \
 --metrics+p --metrics+r --metrics+map --metrics+jmap --metrics+muap   \
 --metrics+ndcg --metrics+ndcg5 --metrics+ndcg10 --metrics+ndcg20 --metrics+ndcg30 --metrics+ndcg100  \
 --metrics+ndcg200 --metrics+ndcg500 --metrics+ndcg1000  \
 --metrics+recip_rank --metrics+R-prec  \
 --metrics+P5 --metrics+P10 --metrics+P20 --metrics+P30 --metrics+P100 --metrics+P200 --metrics+P500
  ↪  --metrics+P1000
else
 ./galago-3.10/contrib/target/appassembler/bin/galago $@
fi
```