Course Work

Course: Information Retrieval

Name: Guangjin Zhang

ID Number: 2136849

1. Introduction

This course work aims to develop an IR system and evaluate it on a small data set. It need use P-R graph, weighting method and vector space to query and compare.

The P-R graph shows the average performance over a large set of queries with each system. By calculating the average precision at each standard recall level across all queries, we can draw the graph precision-recall curves to evaluate overall system performance. When precision is equal to 1, it returns mostly relevant documents but misses many useful information too. When recall is equal to 1, it returns all the relevant documents but includes lots of junk.

There are two demands on the weight: one is the degree to which a particular document about a topic or a particular keyword; another is the degree to which a keyword discriminates documents in the collection. The Inverse Document Frequency (IDF) means the queries using rather broadly defined, frequently occurring terms, which is the more specific terms that are particularly important in identifying relevant material.

Vector space model represents both query and documents using term sets (term vectors) and computes similarity between documents and query. As documents and queries are represented in a high dimensional, each dimension of the space corresponds to a word in the document collection.

2. Report on tokenizer

Add two description paths in `flair.descriptions.file`:

```
1 testDescriptions/medlineDescription.txt
2 testDescriptions/wt2gDescription.txt
```

The Integer at the start of each line represents the Type Identifier and ensure the correct type is used for each document in the '`flair.documents.file`'. The post type of Medline description is 1 and the post type of WT2G description is 2.

Add two collection paths in `flair.documents.file`:

```
1 testCollection/MED.ALL
2 testCollection/B01.txt
```

The Integer at the start of each line represents the Type Identifier and ensure the correct type is used for each document. The post type of Medline collection is 1 and the post type of WT2G collection is 2.

Add index in `flair.index.spec` without stop and stemming:
Token type is TEXT_TOKEN in `uk.ac.gla.mir.flair.irmodel.filter.TextTokenizer`:

```
<INDEX dirname = IdentifiersMedLine tokentype = IDENTIFIER_TOKEN posttype = 1><id>1</id></INDEX>
<INDEX invname = TextIndexMedLine tokentype = TEXT_TOKEN posttype = 1><id>2</id></INDEX>
<INDEX dirname = IdentifiersB01 tokentype = IDENTIFIER_TOKEN posttype = 2><id>1</id></INDEX>
<INDEX invname = TextIndexB01 tokentype = TEXT_TOKEN posttype = 2><id>5</id></INDEX>
```

Token type is NEW_TEXT_TOKEN in `uk.ac.gla.mir.flair.irmodel.filter.NewTextTokenizer`:

```
<INDEX dirname = IdentifiersMedLine tokentype = IDENTIFIER_TOKEN posttype = 1><id>1</id></INDEX>
<INDEX invname = TextIndexMedLine tokentype = NEW_TEXT_TOKEN posttype = 1><id>2</id></INDEX>
<INDEX dirname = IdentifiersB01 tokentype = IDENTIFIER_TOKEN posttype = 2><id>1</id></INDEX>
<INDEX invname = TextIndexB01 tokentype = NEW_TEXT_TOKEN posttype = 2><id>5</id></INDEX>
```

A Direct Index will link the internal retrieved document to its identified, which decreases lookup time for DOCNO. An Inverted Index will link the terms to the internal document IDs.

For `dirname = IdentifiersMedLine, posttype = 1` will link to `medlineDescription.txt` and `<id>1</id>` will specify the DOCNO, as we numbered this field at 1 in the document description.
For `invname = TextIndexMedLine, posttype = 1` will link to `medlineDescription.txt` and `<id>2</id>` will link specify the BODY, as we numbered this field at 2 in the document description.
For `dirname = IdentifiersB01, posttype = 2` will link to `wt2gDescription.txt` and `<id>1</id>` will specify the DOCNO, as we numbered this field at 1 in the document description.
For `invname = TextIndexMedLine, posttype = 1` will link to `wt2gDescription.txt` and `<id>5</id>` will link specify the BODY, as we numbered this field at 5 in the document description.

The Fig.1_1 shows the index of MED.ALL with `TEXT_TOKEN` Token type and the Fig.1_2 shows the statistics about the collection. The time to index is 484 ms.

```
C:\Users\Guangjin\Desktop\course\IR\flair\Flair>bin\flair.bat -i
Set FLAIR_HOME to be C:\Users\Guangjin\Desktop\course\IR\flair\Flair
**********
* Assertion Info:
* Properties Loaded OK
**********
**********
* Assertion Info:
* Flair Home = C:\Users\Guangjin\Desktop\course\IR\flair\Flair
* Flair ETC = C:\Users\Guangjin\Desktop\course\IR\flair\Flair\etc
* Flair Descriptions File = etc/flair.descriptions
* Flair Input File = etc/flair.documents
* Flair Stopwords File = etc/stopfile.txt
* Flair Index Path = index
**********
**********
* Assertion Info:
* Flair Index Spec File = etc/flair.index.spec
**********
Creating Inverted Index : "TextIndexMedLine" : null : TEXT_TOKEN : 1 : <2>
Using :irmodel.filter.TextTokenizer
Creating Direct Index : "IdentifiersMedLine" : null : IDENTIFIER_TOKEN : 1 : <1>
**********
* Assertion warning:
* Document Position Index index\docPosition.index already exists, it will be overwritten.
**********
Do you want to continue?y
1 testDescriptions/medlineDescription.txt
Reading Description File 1 = testDescriptions/medlineDescription.txt
Indexing testCollection/MED.ALL <0>
**********
* Assertion Info:
* Indexing Complete. Taking 484ms.
**********
```

Fig.1_1

```
C:\Users\Guangjin\Desktop\course\IR\flair\Flair>bin\flair.bat --print-stat 1
Set FLAIR_HOME to be C:\Users\Guangjin\Desktop\course\IR\flair\Flair
**********
* Assertion Info:
* Properties Loaded OK
**********
**********
* Assertion Info:
* Flair Home = C:\Users\Guangjin\Desktop\course\IR\flair\Flair
* Flair ETC = C:\Users\Guangjin\Desktop\course\IR\flair\Flair\etc
* Flair Descriptions File = etc/flair.descriptions
* Flair Input File = etc/flair.documents
* Flair Stopwords File = etc/stopfile.txt
* Flair Index Path = index
**********
**********
* Assertion Info:
* Flair Index Spec File = etc/flair.index.spec
**********
Loading Inverted Index : "TextIndexMedLine" : null : TEXT_TOKEN : 1 : <2>
Using :irmodel.filter.TextTokenizer
Loading Direct Index : "IdentifiersMedLine" : null : IDENTIFIER_TOKEN : 1 : <1>
Direct Index for 1 : IdentifiersMedLine
Inverted Index for 1 : TextIndexMedLine
Total Number of Documents = 1033
Average Document Length = 155.0329138431752
Total Number of Uniq Terms = 13300
```

Fig.1_2

The Fig.2_1 shows the index of MED.ALL with NEW_TEXT_TOKEN Token type and the Fig.2_2 shows the statistics about the collection. The time to index is 516 ms.

```
C:\Users\Guangjin\Desktop\course\IR\flair\Flair>bin\flair.bat -i
Set FLAIR_HOME to be C:\Users\Guangjin\Desktop\course\IR\flair\Flair
**********
* Assertion Info:
* Properties Loaded OK
**********
**********
* Assertion Info:
* Flair Home = C:\Users\Guangjin\Desktop\course\IR\flair\Flair
* Flair ETC = C:\Users\Guangjin\Desktop\course\IR\flair\Flair\etc
* Flair Descriptions File = etc/flair.descriptions
* Flair Input File = etc/flair.documents
* Flair Stopwords File = etc/stopfile.txt
* Flair Index Path = index
**********
**********
* Assertion Info:
* Flair Index Spec File = etc/flair.index.spec
**********
Creating Inverted Index : "TextIndexMedLine" : null : NEW_TEXT_TOKEN : 1 : <2>
Using :irmodel.filter.NewTextTokenizer
Creating Direct Index : "IdentifiersMedLine" : null : IDENTIFIER_TOKEN : 1 : <1>
**********
* Assertion warning:
* Document Position Index index\docPosition.index already exists, it will be overwritten.
**********
Do you want to continue?y
1 testDescriptions/medlineDescription.txt
Reading Description File 1 = testDescriptions/medlineDescription.txt
Indexing testCollection/MED.ALL <0>
**********
* Assertion Info:
* Indexing Complete. Taking 516ms.
**********
```

Fig.2_1

```
C:\Users\Guangjin\Desktop\course\IR\flair\Flair>bin\flair.bat --print-stat 1
Set FLAIR_HOME to be C:\Users\Guangjin\Desktop\course\IR\flair\Flair
**********
* Assertion Info:
* Properties Loaded OK
**********
**********
* Assertion Info:
* Flair Home = C:\Users\Guangjin\Desktop\course\IR\flair\Flair
* Flair ETC = C:\Users\Guangjin\Desktop\course\IR\flair\Flair\etc
* Flair Descriptions File = etc/flair.descriptions
* Flair Input File = etc/flair.documents
* Flair Stopwords File = etc/stopfile.txt
* Flair Index Path = index
**********
**********
* Assertion Info:
* Flair Index Spec File = etc/flair.index.spec
**********
Loading Inverted Index : "TextIndexMedLine" : null : NEW_TEXT_TOKEN : 1 : <2>
Using :irmodel.filter.NewTextTokenizer
Loading Direct Index : "IdentifiersMedLine" : null : IDENTIFIER_TOKEN : 1 : <1>
Direct Index for 1 : IdentifiersMedLine
Inverted Index for 1 : TextIndexMedLine
Total Number of Documents = 1033
Average Document Length = 153.75217812197482
Total Number of Uniq Terms = 20219
```

Fig.2_2

The Fig.3_1 shows the index of B01.txt with TEXT_TOKEN Token type and the Fig.3_2 shows the statistics about the collection. The time to index is 649 ms.

```
C:\Users\Guangjin\Desktop\course\IR\flair\Flair>bin\flair.bat -i
Set FLAIR_HOME to be C:\Users\Guangjin\Desktop\course\IR\flair\Flair
**********
* Assertion Info:
* Properties Loaded OK
**********
**********
* Assertion Info:
* Flair Home = C:\Users\Guangjin\Desktop\course\IR\flair\Flair
* Flair ETC = C:\Users\Guangjin\Desktop\course\IR\flair\Flair\etc
* Flair Descriptions File = etc/flair.descriptions
* Flair Input File = etc/flair.documents
* Flair Stopwords File = etc/stopfile.txt
* Flair Index Path = index
**********
**********
* Assertion Info:
* Flair Index Spec File = etc/flair.index.spec
**********
Creating Inverted Index : "TextIndexB01" : null : TEXT_TOKEN : 2 : <5>
Using :irmodel.filter.TextTokenizer
Creating Direct Index : "IdentifiersB01" : null : IDENTIFIER_TOKEN : 2 : <1>
**********
* Assertion warning:
* Document Position Index index\docPosition.index already exists, it will be overwritten.
**********
Do you want to continue?y
2 testDescriptions/wt2gDescription.txt
Reading Description File 2 = testDescriptions/wt2gDescription.txt
Indexing testCollection/B01.txt <0>
**********
* Assertion Info:
* Indexing Complete. Taking 649ms.
**********
```

Fig.3_1

```
C:\Users\Guangjin\Desktop\course\IR\flair\Flair>bin\flair.bat --print-stat 2
Set FLAIR_HOME to be C:\Users\Guangjin\Desktop\course\IR\flair\Flair
**********
* Assertion Info:
* Properties Loaded OK
**********
**********
* Assertion Info:
* Flair Home = C:\Users\Guangjin\Desktop\course\IR\flair\Flair
* Flair ETC = C:\Users\Guangjin\Desktop\course\IR\flair\Flair\etc
* Flair Descriptions File = etc/flair.descriptions
* Flair Input File = etc/flair.documents
* Flair Stopwords File = etc/stopfile.txt
* Flair Index Path = index
**********
**********
* Assertion Info:
* Flair Index Spec File = etc/flair.index.spec
**********
Loading Inverted Index : "TextIndexB01" : null : TEXT_TOKEN : 2 : <5>
Using :irmodel.filter.TextTokenizer
Loading Direct Index : "IdentifiersB01" : null : IDENTIFIER_TOKEN : 2 : <1>
Direct Index for 2 : IdentifiersB01
Inverted Index for 2 : TextIndexB01
Total Number of Documents = 296
Average Document Length = 1051.4527027027027
Total Number of Uniq Terms = 19640
```

Fig.3_2

The Fig.4_1 shows the index of B01.txt with NEW_TEXT_TOKEN Token type and the Fig.4_2 shows the statistics about the collection. The time to index is 686 ms.

```
C:\Users\Guangjin\Desktop\course\IR\flair\Flair>bin\flair.bat -i
Set FLAIR_HOME to be C:\Users\Guangjin\Desktop\course\IR\flair\Flair
**********
* Assertion Info:
* Properties Loaded OK
**********
**********
* Assertion Info:
* Flair Home = C:\Users\Guangjin\Desktop\course\IR\flair\Flair
* Flair ETC = C:\Users\Guangjin\Desktop\course\IR\flair\Flair\etc
* Flair Descriptions File = etc/flair.descriptions
* Flair Input File = etc/flair.documents
* Flair Stopwords File = etc/stopfile.txt
* Flair Index Path = index
**********
**********
* Assertion Info:
* Flair Index Spec File = etc/flair.index.spec
**********
Creating Inverted Index : "TextIndexB01" : null : NEW_TEXT_TOKEN : 2 : <5>
Using :irmodel.filter.NewTextTokenizer
Creating Direct Index : "IdentifiersB01" : null : IDENTIFIER_TOKEN : 2 : <1>
**********
* Assertion warning:
* Document Position Index index\docPosition.index already exists, it will be overwritten.
**********
Do you want to continue?y
2 testDescriptions/wt2gDescription.txt
Reading Description File 2 = testDescriptions/wt2gDescription.txt
Indexing testCollection/B01.txt <0>
**********
* Assertion Info:
* Indexing Complete. Taking 686ms.
**********
```

Fig.4_1

```
C:\Users\Guangjin\Desktop\course\IR\flair\Flair>bin\flair.bat --print-stat 2
Set FLAIR_HOME to be C:\Users\Guangjin\Desktop\course\IR\flair\Flair
**********
* Assertion Info:
* Properties Loaded OK
**********
**********
* Assertion Info:
* Flair Home = C:\Users\Guangjin\Desktop\course\IR\flair\Flair
* Flair ETC = C:\Users\Guangjin\Desktop\course\IR\flair\Flair\etc
* Flair Descriptions File = etc/flair.descriptions
* Flair Input File = etc/flair.documents
* Flair Stopwords File = etc/stopfile.txt
* Flair Index Path = index
**********
**********
* Assertion Info:
* Flair Index Spec File = etc/flair.index.spec
**********
Loading Inverted Index : "TextIndexB01" : null : NEW_TEXT_TOKEN : 2 : <5>
Using :irmodel.filter.NewTextTokenizer
Loading Direct Index : "IdentifiersB01" : null : IDENTIFIER_TOKEN : 2 : <1>
Direct Index for 2 : IdentifiersB01
Inverted Index for 2 : TextIndexB01
Total Number of Documents = 296
Average Document Length = 878.4459459459459
Total Number of Uniq Terms = 39086
```

Fig.4_2

Table 1

Token type	Collection	Index time	No. of DOC	Average Length of DOC	No. of Unique Terms
TEXT_TOKEN	Medline	484ms	1033	155	13300
	HTML DOC	649ms	296	1051	19640
NEW_TEXT_TOKEN	Medline	516ms	1033	154	20219
	HTML DOC	686ms	296	878	39086

Medline vs. html documents:

Table 1 shows that with the same Token type, the index time of Medline is less than HTML documents, however, the number of documents of the former is much more than that of the latter. Furthermore, the average length of documents and number of unique terms of Medline are smaller than those of HTML documents. The reason is that Medline documents have fixed format and main useful content is all in BODY part. It is easy to index by searching the BODY. However, the HTML documents have many different types of content in body such as table and specific symbol. It is hard to identify whole of them in the description. Therefore there are many useless information in searching part of HTML documents. So that it will increase the length of documents and number of unique terms, which will increase the index time.

TEXT_TOKEN vs. NEW_TEXT_TOKEN:

According to the Table 1, the average index time used TEXT_TOKEN is less than that used NEW_TEXT_TOKEN. Basically, the number and length of document of the Medline and HTML documents have no change. However, the number of unique terms of them have huge difference between two types. The number with NEW_TEXT_TOKEN is much larger than the one of TEXT_TOKEN. The reason is that TEXT_TOKEN use char method to identify terms straightly. However NEW_TEXT_TOKEN use split method to split word based on blank. The terms may contain useless information such symbol and functional mark. Therefore the number of unique terms are increased and meanwhile the index time is also raise. Especially, as the HTML documents have uncertain structure, there are many special symbol and functional mark. So the difference between two types tokenizer of HTML documents is huger.

Create Query Spec file:

Fig.5_1, Fig.5_2 and Fig.5_3 show the process of creating Query Spec from Medline Topic file.
Enter 1 to set the filename to the MED.QRY file.
Enter 2 to change the weighting type TFIDF.
Enter 3 to set the post type and it will likely be 1 for Medline documents.
Enter 4 to set the field ID and for Medline this should be 2 which is BODY part.

Enter 5 and the '`etc/query.spec.file`' will be created.

```
C:\Users\Guangjin\Desktop\course\IR\flair\Flair>bin\createQuerySpec.bat
Set FLAIR_HOME to be C:\Users\Guangjin\Desktop\course\IR\flair\Flair
**********
* Assertion Info:
* Properties Loaded OK
**********

*********************************************************
*                                                       *
*   This simple program will convert TREC style         *
*   Topic files to the Flair query specification.       *
*   The file 'etc/flair.query.spec' will be created.    *
*                                                       *
*   Current Settings;                                   *
*   TREC Topic FileName = unknown
*   Weighting Model     = TFIDF
*   Post Type           = unknown
*   Field ID            = 2
*                                                       *
*   1.) Get TREC Topics File Name                       *
*   2.) Set Weighting Mode                              *
*   3.) Set Post Type                                   *
*   4.) Set Field ID                                    *
*   5.) Create flair.query.spec                         *
*                                                       *
*   9.) Display This Help Message                       *
*                                                       *
*                                                       *
*   0.) Exit                                            *
*                                                       *
*********************************************************


Enter option number ->
1
Enter the TREC Topic File Name ->
MED.QRY
File Name = MED.QRY
Correct Y/N [Y] -?
y
```

Fig.5_1

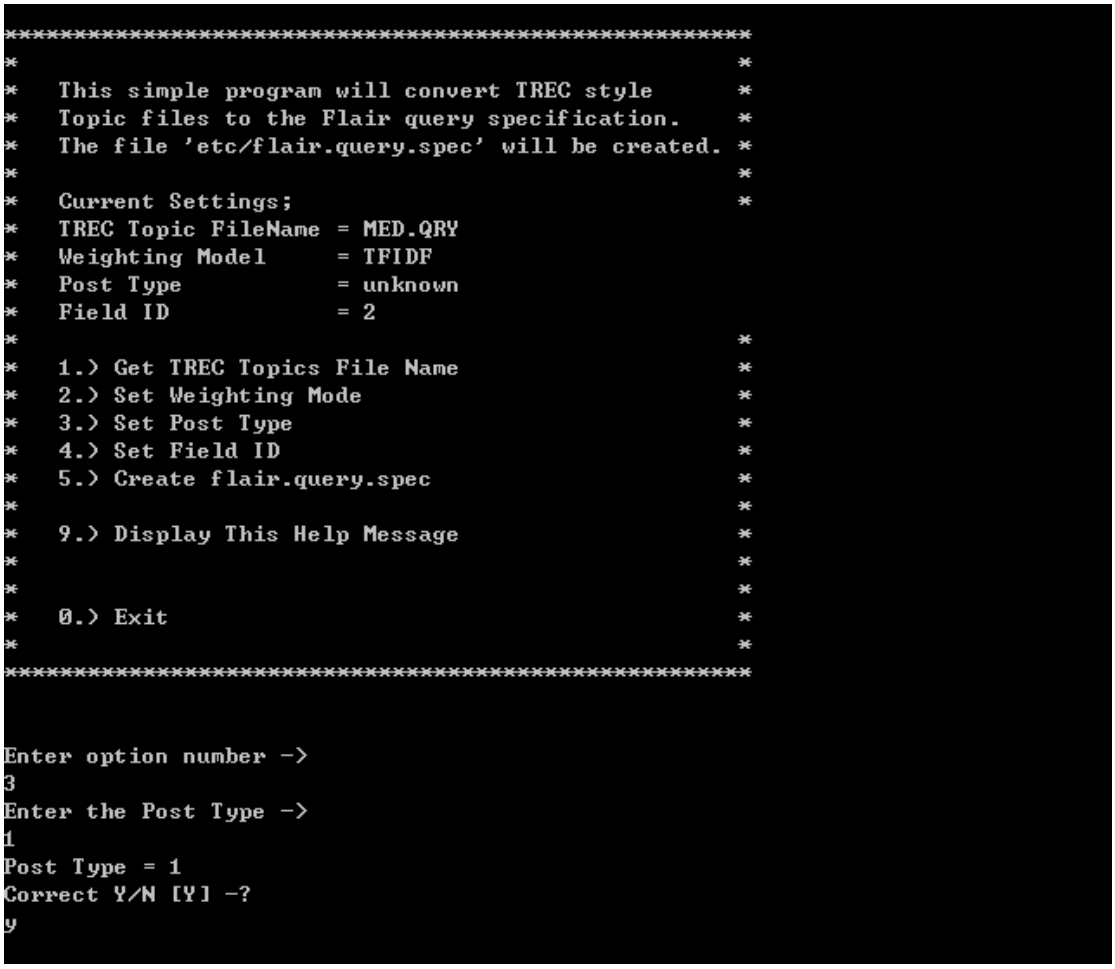

This simple program will convert TREC style
Topic files to the Flair query specification.
The file 'etc/flair.query.spec' will be created.
Current Settings;
TREC Topic FileName = MED.QRY
Weighting Model = TFIDF
Post Type = unknown
Field ID = 2
1.) Get TREC Topics File Name
2.) Set Weighting Mode
3.) Set Post Type
4.) Set Field ID
5.) Create flair.query.spec
9.) Display This Help Message
0.) Exit
Enter option number ->
3
Enter the Post Type ->
1
Post Type = 1
Correct Y/N [Y] -?
y

Fig.5_2

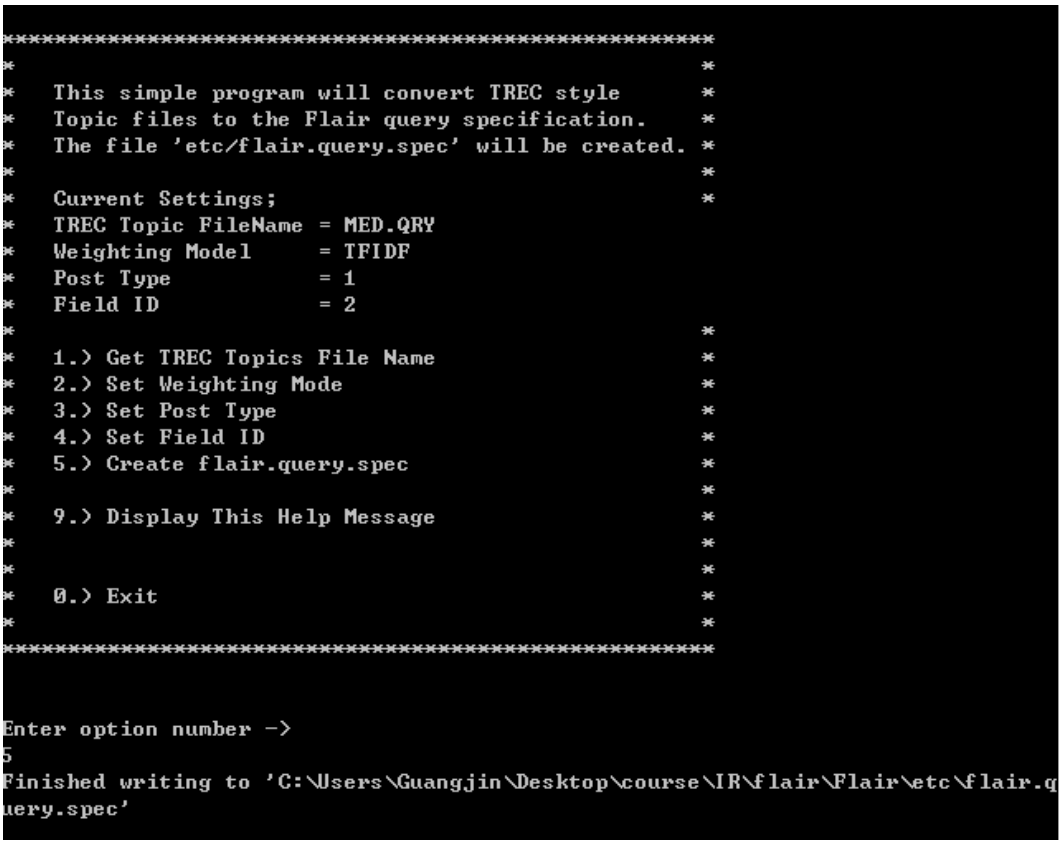
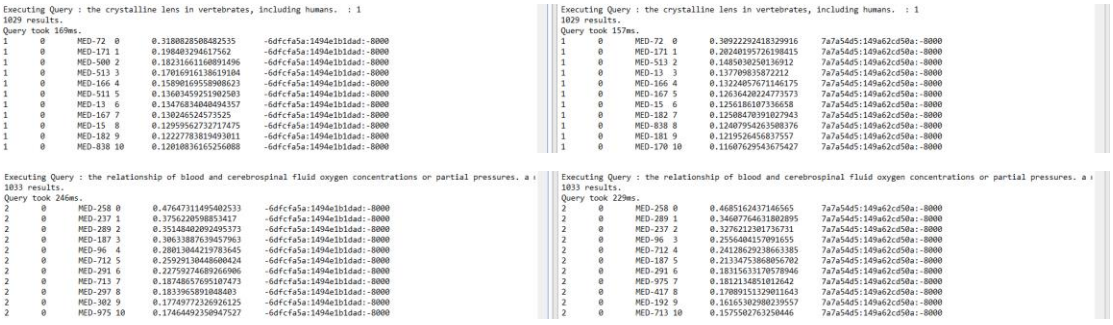


Fig.5_3

Query the collection of MED.ALL with TEXT_TOKEN and NEW_TEXT_TOKEN

By putting in ***bin\flair.bat –q*** and storing the results in two texts, we can compare the results in Fig.5_4 as follow:



TEXT_TOKEN **NEW_TEXT_TOKEN**

Fig.5_4

The result shows that the query time are nearly same. However, the TF-IDF weighting score of each row with **TEXT_TOKEN** is larger than the score with **NEW_TEXT_TOKEN**. It is because the token results of former is more accuracy than the latter one which contains more noise results.

P-R graph for medline collection used TF-IDF without stop and stemming.

The P-R graph Fig.5_5 is as follow:

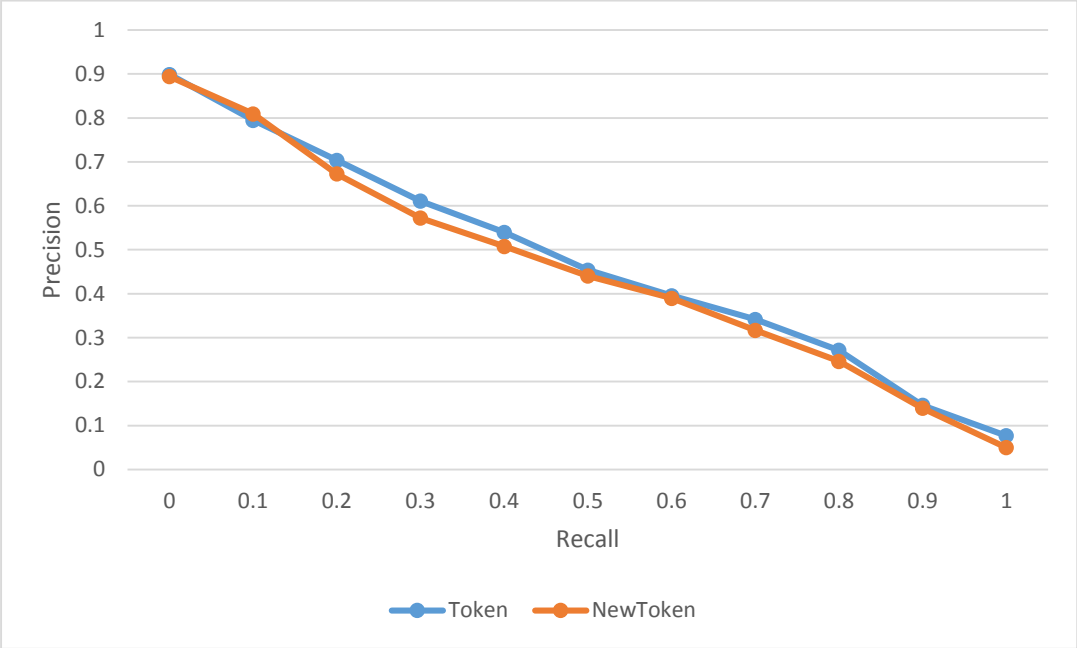


Fig.5_5

The graph shows the performance of system with **TEXT_TOKEN** token method and **NEW_TEXT_TOKEN** method. According to the graph, basically the average precision in each recall with the former is larger than the one with the latter. The curve of the former is more close to the right–hand corner of the graph. It means the system with **TEXT_TOKEN** indicated better performance than the one with **NEW_TEXT_TOKEN.**

This is because the former use char method to tokenize document, which contains less noise terms than the latter that splits word based on blank. For example, if the query term is “Glasgow”, but the term in the document is “(Glasgow)”, the system cannot add this document as relevant document when retrieving.

3. Stemming vs. No stemming:

I used TEXT_TOKEN tokenizer with char method.

Add index in `flair.index.spec` with stemming:

```
<INDEX dirname = IdentifiersMedLine tokentype = IDENTIFIER_TOKEN posttype = 1><id>1</id></INDEX>
<INDEX invname = TextIndexMedLine tokentype = TEXT_TOKEN stoptype = STOP stemtype = PORTER
posttype = 1><id>2</id></INDEX>

<INDEX dirname = IdentifiersB01 tokentype = IDENTIFIER_TOKEN posttype = 2><id>1</id></INDEX>
<INDEX invname = TextIndexB01 tokentype = TEXT_TOKEN stoptype = STOP stemtype = PORTER posttype
= 2><id>5</id></INDEX>
```

Add index in `flair.index.spec` without stemming:

```
<INDEX dirname = IdentifiersMedLine tokentype = IDENTIFIER_TOKEN posttype = 1><id>1</id></INDEX>
<INDEX invname = TextIndexMedLine tokentype = TEXT_TOKEN stoptype = STOP posttype = 1><id>2</id></INDEX>

<INDEX dirname = IdentifiersB01 tokentype = IDENTIFIER_TOKEN posttype = 2><id>1</id></INDEX>
<INDEX invname = TextIndexB01 tokentype = TEXT_TOKEN stoptype = STOP posttype = 2><id>5</id></INDEX>
```

The token type is `TEXT_TOKEN`, stop type is `STOP`, Stem type is `PORTER` and weighting scheme is `TFIDF`.

The Fig.6_1 shows the index of MED.ALL with Stemming and the Fig.6_2 shows the statistics about the collection. The time to index is 482 ms.

```
C:\Users\Guangjin\Desktop\course\IR\flair\Flair>bin\flair.bat -i
Set FLAIR_HOME to be C:\Users\Guangjin\Desktop\course\IR\flair\Flair
**********
* Assertion Info:
* Properties Loaded OK
**********
**********
* Assertion Info:
* Flair Home = C:\Users\Guangjin\Desktop\course\IR\flair\Flair
* Flair ETC = C:\Users\Guangjin\Desktop\course\IR\flair\Flair\etc
* Flair Descriptions File = etc/flair.descriptions
* Flair Input File = etc/flair.documents
* Flair Stopwords File = etc/stopfile.txt
* Flair Index Path = index
**********
**********
* Assertion Info:
* Flair Index Spec File = etc/flair.index.spec
**********
Creating Inverted Index : "TextIndexMedLine" : PORTER : TEXT_TOKEN : 1 : (2)
Using :irmodel.filter.TextTokenizer
Using :irmodel.filter.Stopper
Using :irmodel.filter.PorterStemmer
Creating Direct Index : "IdentifiersMedLine" : null : IDENTIFIER_TOKEN : 1 : (1)
**********
* Assertion warning:
* Document Position Index index\docPosition.index already exists, it will be overwritten.
**********
Do you want to continue?y
1 testDescriptions/medlineDescription.txt
Reading Description File 1 = testDescriptions/medlineDescription.txt
Indexing testCollection/MED.ALL (0)
**********
* Assertion Info:
* Indexing Complete. Taking 482ms.
**********
```

Fig.6_1

```
C:\Users\Guangjin\Desktop\course\IR\flair\Flair>bin\flair.bat --print-stat 1
Set FLAIR_HOME to be C:\Users\Guangjin\Desktop\course\IR\flair\Flair
**********
* Assertion Info:
* Properties Loaded OK
**********
**********
* Assertion Info:
* Flair Home = C:\Users\Guangjin\Desktop\course\IR\flair\Flair
* Flair ETC = C:\Users\Guangjin\Desktop\course\IR\flair\Flair\etc
* Flair Descriptions File = etc/flair.descriptions
* Flair Input File = etc/flair.documents
* Flair Stopwords File = etc/stopfile.txt
* Flair Index Path = index
**********
**********
* Assertion Info:
* Flair Index Spec File = etc/flair.index.spec
**********
Loading Inverted Index : "TextIndexMedLine" : PORTER : TEXT_TOKEN : 1 : <2>
Using :irmodel.filter.TextTokenizer
Using :irmodel.filter.Stopper
Using :irmodel.filter.PorterStemmer
Loading Direct Index : "IdentifiersMedLine" : null : IDENTIFIER_TOKEN : 1 : <1>
Direct Index for 1 : IdentifiersMedLine
Inverted Index for 1 : TextIndexMedLine
Total Number of Documents = 1033
Average Document Length = 83.58470474346564
Total Number of Uniq Terms = 8758
```

Fig.6_2

The Fig.7_1 shows the index of MED.ALL without Stemming and the Fig.7_2 shows the statistics about the collection. The time to index is 445 ms.

```
C:\Users\Guangjin\Desktop\course\IR\flair\Flair>bin\flair.bat -i
Set FLAIR_HOME to be C:\Users\Guangjin\Desktop\course\IR\flair\Flair
**********
* Assertion Info:
* Properties Loaded OK
**********
**********
* Assertion Info:
* Flair Home = C:\Users\Guangjin\Desktop\course\IR\flair\Flair
* Flair ETC = C:\Users\Guangjin\Desktop\course\IR\flair\Flair\etc
* Flair Descriptions File = etc/flair.descriptions
* Flair Input File = etc/flair.documents
* Flair Stopwords File = etc/stopfile.txt
* Flair Index Path = index
**********
**********
* Assertion Info:
* Flair Index Spec File = etc/flair.index.spec
**********
Creating Inverted Index : "TextIndexMedLine" : null : TEXT_TOKEN : 1 : <2>
Using :irmodel.filter.TextTokenizer
Using :irmodel.filter.Stopper
Creating Direct Index : "IdentifiersMedLine" : null : IDENTIFIER_TOKEN : 1 : <1>
**********
* Assertion warning:
* Document Position Index index\docPosition.index already exists, it will be overwritten.
**********
Do you want to continue?y
1 testDescriptions/medlineDescription.txt
Reading Description File 1 = testDescriptions/medlineDescription.txt
Indexing testCollection/MED.ALL <0>
**********
* Assertion Info:
* Indexing Complete. Taking 445ms.
**********
```

Fig.7_1

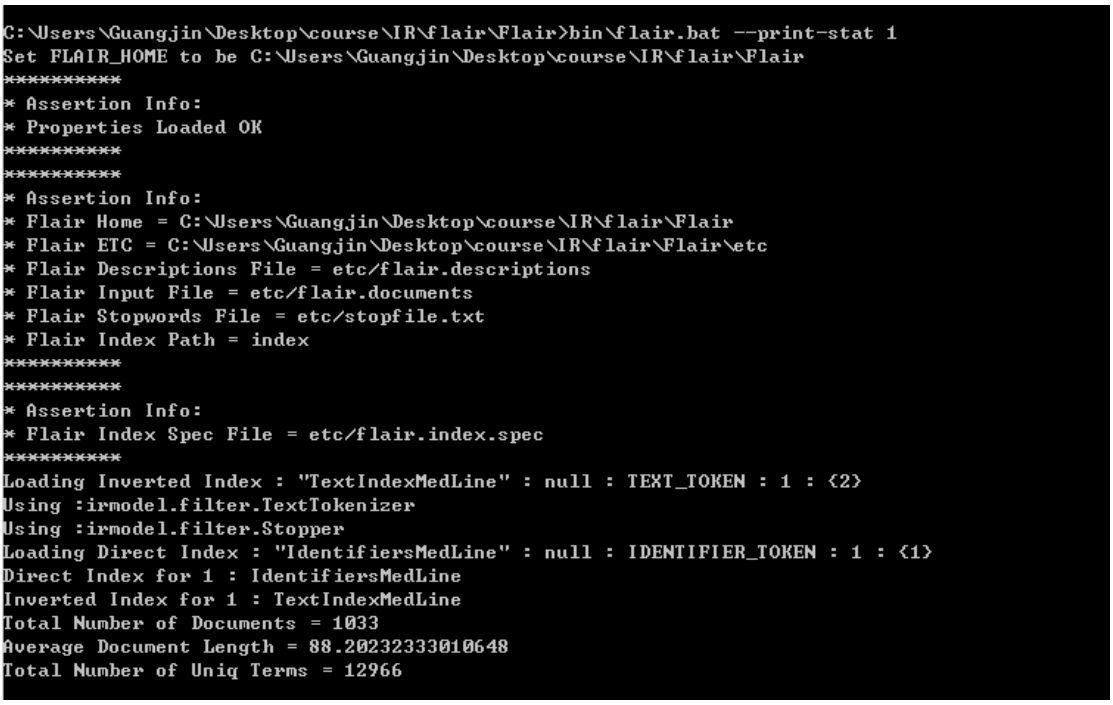


Fig.7_2

The Fig.8_1 shows the index of B01.txt with Stemming and the Fig.8_2 shows the statistics about the collection. The time to index is 773 ms.

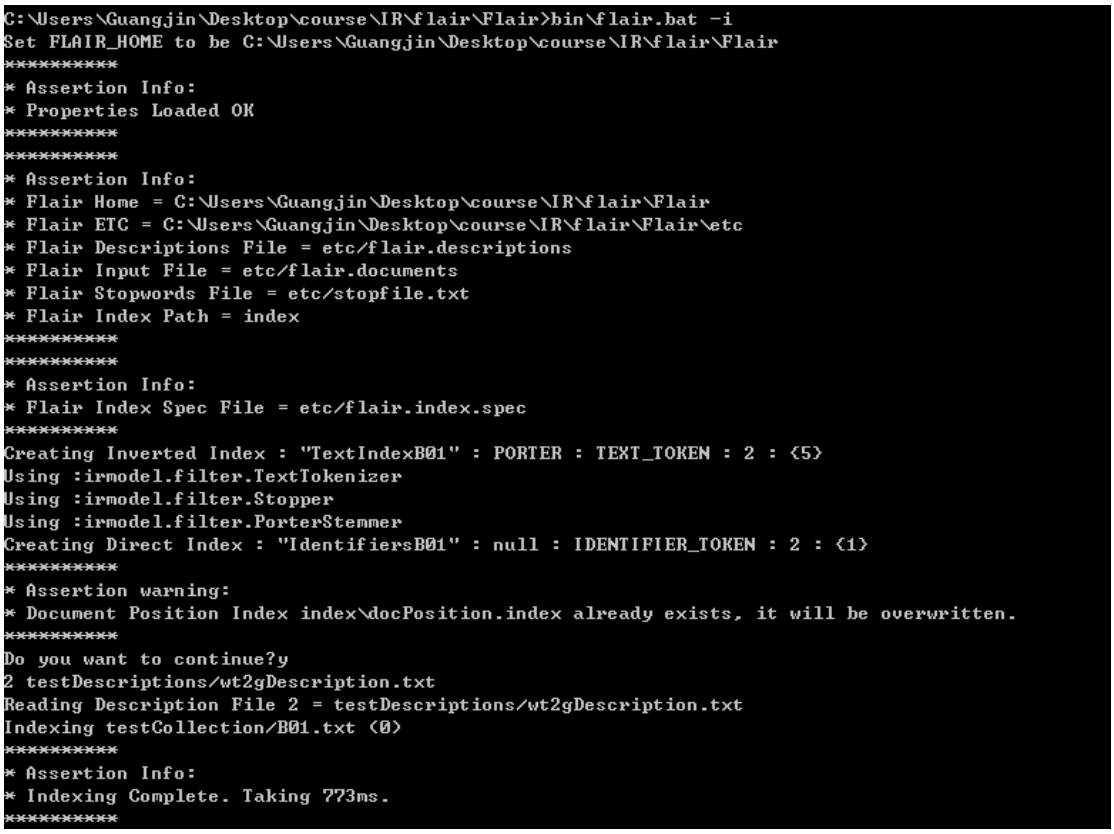


Fig.8_1

```
C:\Users\Guangjin\Desktop\course\IR\flair\Flair>bin\flair.bat --print-stat 2
Set FLAIR_HOME to be C:\Users\Guangjin\Desktop\course\IR\flair\Flair
**********
* Assertion Info:
* Properties Loaded OK
**********
**********
* Assertion Info:
* Flair Home = C:\Users\Guangjin\Desktop\course\IR\flair\Flair
* Flair ETC = C:\Users\Guangjin\Desktop\course\IR\flair\Flair\etc
* Flair Descriptions File = etc/flair.descriptions
* Flair Input File = etc/flair.documents
* Flair Stopwords File = etc/stopfile.txt
* Flair Index Path = index
**********
**********
* Assertion Info:
* Flair Index Spec File = etc/flair.index.spec
**********
Loading Inverted Index : "TextIndexB01" : PORTER : TEXT_TOKEN : 2 : <5>
Using :irmodel.filter.TextTokenizer
Using :irmodel.filter.Stopper
Using :irmodel.filter.PorterStemmer
Loading Direct Index : "IdentifiersB01" : null : IDENTIFIER_TOKEN : 2 : <1>
Direct Index for 2 : IdentifiersB01
Inverted Index for 2 : TextIndexB01
Total Number of Documents = 296
Average Document Length = 621.7635135135135
Total Number of Uniq Terms = 12216
```

Fig.8_2

The Fig.9_1 shows the index of B01.txt without Stemming and the Fig.9_2 shows the statistics about the collection. The time to index is 615 ms.

```
C:\Users\Guangjin\Desktop\course\IR\flair\Flair>bin\flair.bat -i
Set FLAIR_HOME to be C:\Users\Guangjin\Desktop\course\IR\flair\Flair
**********
* Assertion Info:
* Properties Loaded OK
**********
**********
* Assertion Info:
* Flair Home = C:\Users\Guangjin\Desktop\course\IR\flair\Flair
* Flair ETC = C:\Users\Guangjin\Desktop\course\IR\flair\Flair\etc
* Flair Descriptions File = etc/flair.descriptions
* Flair Input File = etc/flair.documents
* Flair Stopwords File = etc/stopfile.txt
* Flair Index Path = index
**********
**********
* Assertion Info:
* Flair Index Spec File = etc/flair.index.spec
**********
Creating Inverted Index : "TextIndexB01" : null : TEXT_TOKEN : 2 : <5>
Using :irmodel.filter.TextTokenizer
Using :irmodel.filter.Stopper
Creating Direct Index : "IdentifiersB01" : null : IDENTIFIER_TOKEN : 2 : <1>
**********
* Assertion warning:
* Document Position Index index\docPosition.index already exists, it will be overwritten.
**********
Do you want to continue?y
2 testDescriptions/wt2gDescription.txt
Reading Description File 2 = testDescriptions/wt2gDescription.txt
Indexing testCollection/B01.txt <0>
**********
* Assertion Info:
* Indexing Complete. Taking 615ms.
**********
```

Fig.9_1

```
C:\Users\Guangjin\Desktop\course\IR\flair\Flair>bin\flair.bat --print-stat 2
Set FLAIR_HOME to be C:\Users\Guangjin\Desktop\course\IR\flair\Flair
**********
* Assertion Info:
* Properties Loaded OK
**********
**********
* Assertion Info:
* Flair Home = C:\Users\Guangjin\Desktop\course\IR\flair\Flair
* Flair ETC = C:\Users\Guangjin\Desktop\course\IR\flair\Flair\etc
* Flair Descriptions File = etc/flair.descriptions
* Flair Input File = etc/flair.documents
* Flair Stopwords File = etc/stopfile.txt
* Flair Index Path = index
**********
**********
* Assertion Info:
* Flair Index Spec File = etc/flair.index.spec
**********
Loading Inverted Index : "TextIndexB01" : null : TEXT_TOKEN : 2 : <5>
Using :irmodel.filter.TextTokenizer
Using :irmodel.filter.Stopper
Loading Direct Index : "IdentifiersB01" : null : IDENTIFIER_TOKEN : 2 : <1>
Direct Index for 2 : IdentifiersB01
Inverted Index for 2 : TextIndexB01
Total Number of Documents = 296
Average Document Length = 670.6959459459459
Total Number of Uniq Terms = 19263
```

Fig.9_2

Table 2

	Collection	Index time	No. of DOC	Average Length of DOC	No. of Unique Terms
Stemming	Medline	482ms	1033	84	8758
	HTML DOC	773ms	296	622	12216
No Stemming	Medline	445ms	1033	88	12966
	HTML DOC	615ms	296	671	19263

Medline vs. html documents:

Table 2 shows that with the same statement of stemming, the index time of Medline is less than HTML documents, however, the number of documents of the former is much more than that of the latter. Furthermore, the average length of documents and number of unique terms of Medline are smaller than those of HTML documents. The reason is that Medline documents have fixed format and main useful content is all in BODY part. It is easy to index by searching the BODY. However, the HTML documents have many different types of content in body such as table and specific symbol. It is hard to identify whole of them in the description. Therefore there are many useless information in searching part of HTML documents. So that it will increase the length of documents and number of unique terms, which will increase the index time. Moreover, due to the complex of HTML documents, it need more time to stemming than Medline.

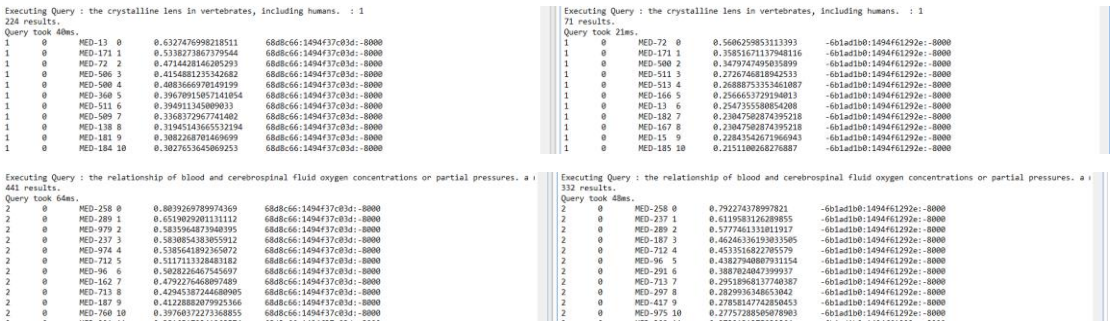
Stemming vs. No Stemming:

According to the Table 2, the average index time with stemming is larger than that without stemming. Basically, the number and length of document of the Medline and HTML documents have no change. However, the number of unique terms of them are different in two cases. The number with stemming is smaller than the one without stemming. It is because that stemming algorithm is a conflation procedure which can reduce all words with same root into a single root. It need time to stem the document so that it can compress the index to reduce the index size.

According to the results by querying the collection (`MedlineQueryStemming.txt` and `MedlineQueryNoStemming.txt` in attachment), the average query time with stemming is larger than that without stemming. Meanwhile, the number of relevant documents with stemming is more than that without stemming. Furthermore, for the same relevant document by querying, the score of weighting model tf-idf with stemming is bigger than that without stemming. The reason is that as two keywords that were initially treated independently are interchangeable, stemming can increases retrieval of all possibly relevant documents.

Query the collection of MED.ALL with and without stemming

By putting in ***bin\flair.bat –q*** and storing the results in two texts, we can compare the results in Fig.10 as follow:



Stemming **non-Stemming**

Fig.10

The query time of each query and TF-IDF weighting score of each row with stemming are both larger than those without stemming. This is because stemming process need spend much more time, and it can reduce all words with same root into a single root. So the term frequency will be increased and value of TF will be raise. Due to the TF-IDF weighting method is TF*IDF, the score with stemming is bigger than the one without stemming. Furthermore, it is also easier for query to match document by terms because there are less noise terms with stemming.

P-R graph for medline collection used TF-IDF with and without stemming.

The P-R graph Fig.11 is as follow:

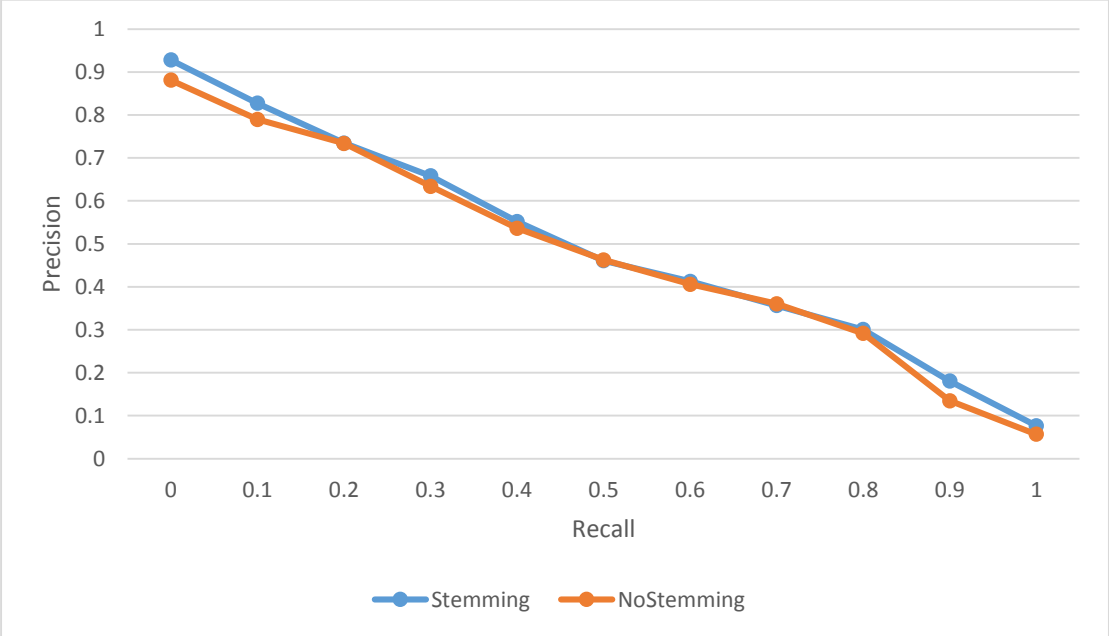


Fig.11

The graph shows the performance of system with stemming and without stemming. According to the graph, basically the average precision in each recall with stemming is larger than the one without stemming. The curve of the former is more close to the right–hand corner of the graph. It means the system with stemming indicated better performance than the one without stemming.

This is because two keywords that were initially treated independently are interchangeable so stemming can increase retrieval of all possibly relevant documents. Due to the algorithm of precision is the number of relevant document which is retrieved over the whole retrieved number, the score of precision will be increased. For example, the query contain the term “university”, but the document just has the terms “universities”, the system cannot retrieval it as a relevant document without stemming.

4. Ranking Algorithms

I use **TEXT_TOKEN**, stop and stemming in this section. Fig.12 shows creating query with TF weighting and that one for TFIDF is the same as Fig.5_3.

```
*********************************************************
*                                                       *
*   This simple program will convert TREC style         *
*   Topic files to the Flair query specification.       *
*   The file 'etc/flair.query.spec' will be created.    *
*                                                       *
*   Current Settings;                                   *
*   TREC Topic FileName = MED.QRY
*   Weighting Model     = RAWTF
*   Post Type           = 1
*   Field ID            = 2
*                                                       *
*   1.) Get TREC Topics File Name                       *
*   2.) Set Weighting Mode                              *
*   3.) Set Post Type                                   *
*   4.) Set Field ID                                    *
*   5.) Create flair.query.spec                         *
*                                                       *
*   9.) Display This Help Message                       *
*                                                       *
*                                                       *
*   0.) Exit                                            *
*                                                       *
*********************************************************


Enter option number ->
5
Finished writing to 'C:\Users\Guangjin\Desktop\course\IR\flair\Flair\etc\flair.query.spec'
```

Fig.12

P-R graph for medline collection used TF and TF-IDF with and without normalization.
The P-R graph Fig.13 is as follow:

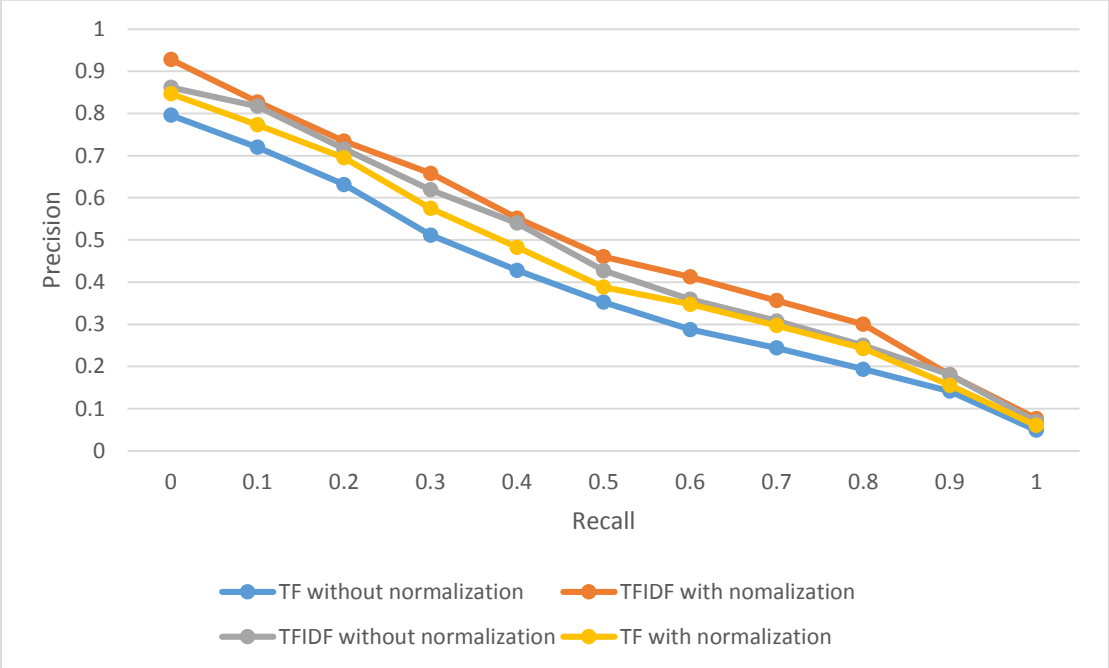


Fig.13

The graph shows the performance of system used TF and TF-IDF with and without normalization. According to the graph, basically the average precision in each recall

ranked by TF-IDF is larger than the one by TF whatever it with or without normalization. The curve of the former is more close to the right–hand corner of the graph. It means the system ranked by TF-IDF indicated better performance than the one by TF.

This is because the TF algorithm is based on the terms frequency of document and TF-IDF uses TF times IDF which is calculate by whole number of documents over the number of documents that contain the term. As calculating frequency of terms in whole documents, the more specific terms that are particularly important in identifying relevant material. Therefore, TF-IDF can reduce the score of normal terms and increase the score of specific terms. Due to rank by TF-IDF, the retrieval will be more effective and more relevant documents can be searched. So the precision will be increased at each recall. For example, a query contain two terms “sport” and “football”, and there are two documents: the first has 3 “sport” and the second has 1 “football”. If using TF weighting to rank, the first document may be in front of the second because the frequency 3 is larger than the frequency 1. However, if using TF-IDF weighting to rank, the second document may be in front of the first one, because the “football” may be a special term of whole document whose topic are sport and the value of IDF is much bigger than the one of term “sport”. So the score of TF-IDF is larger.

The graph also shows that basically the average precision in each recall with normalization is larger than the one without normalization by same weighting method. The curve of the former is more close to the right–hand corner of the graph. It means the system with normalization indicated better performance than the one without normalization.

This is because the normalization can normalize the vector length to 1 by dividing each vector component over the total length of the vector. As avoiding long document has high weighting score when ranking, it can reduce the noise by the different length of each document and make retrieval more effective. For example, a query has term “Glasgow” and there are two documents: the first contains 3 “Glasgow” and the document length is 10000; the second contain 1 “Glasgow” and the document length is 100. If ranking without normalization, the first document will in front of the second one because the term frequency of the first is 3 and the one of the second is 1. So the first document has higher score than the second one. However, if ranking with normalization, the second document may be in front of the first one because the value of term frequency need be divided to the document length. As length 10000 is much larger than 100, the first document has smaller score than the second one.

5. Evaluation

5.1 Query-by-Query performance

There are 28 queries with positive performance improvements and 2 queries with negative performance improvement. They are 10^{th} and 23^{rd} query. According to the

evaluation results, with 10th query, the relevant documents is 24, but the retrieved relevant document is just 9. With 23rd queries, the relevant documents is 39, but the retrieved relevant document is just 19. However the other queries retrieved all relevant document nearly.

This is because the terms “neoplasm immunology” in 10th query and “infantile autism” in 23rd query are very special. Some documents mention the relevant information but not include these terms. So it is hard to have a great retrieved result for them. Moreover, these two queries just contains few terms. The less information also affect the retrieved performance.

5.2 Zipf analysis

According to the all terms frequency result, we can draw a Zipf graph Fig.14 as follow:

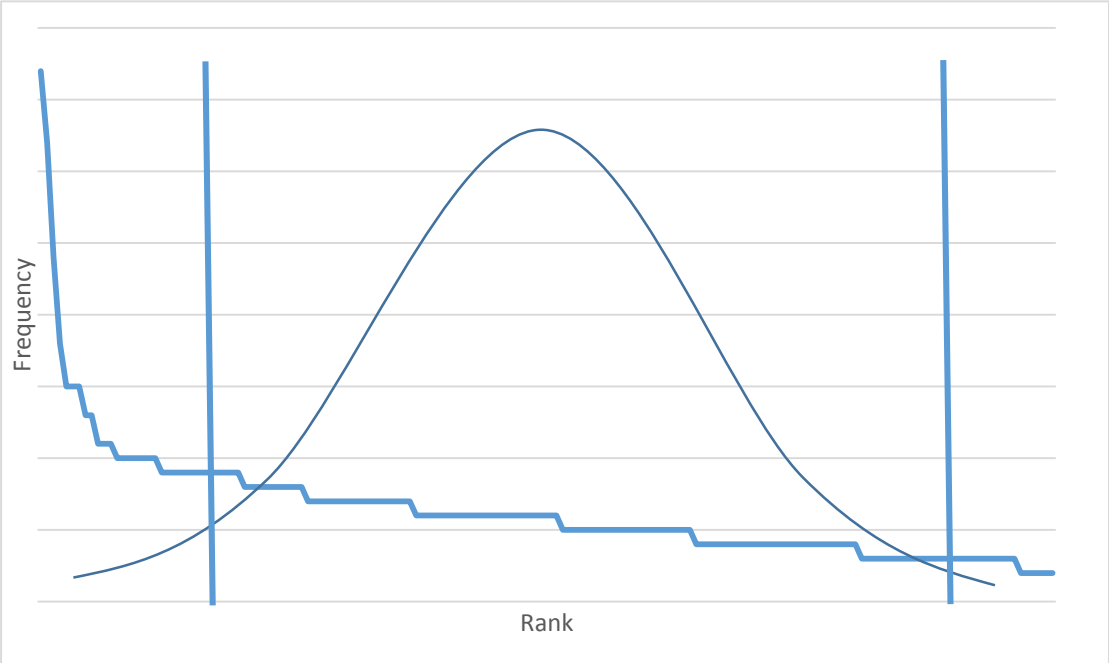


Fig.14

The left line is upper cut-off and the right line is lower cut-off. So at the left side of the former line, the terms is less discriminative and at the right side of the latter line, the terms are likely to be used in queries. It means a few terms occur very frequently and a medium number of terms have medium frequency, however, many elements occur very infrequently. This is because there are always a few very frequent tokens have not good discriminators which are called “stop words” in information retrieval. Usually, they are correspond to linguistic notion of “closed-class” words such as “from” and “to”. Oppositely, there are always a lot of tokens that occur once and can mess up algorithms such “Hmmm”. Therefore medium frequency terms have most descriptive.

5.3 Scalability of the approach

I think the approach will has good performance for large collections. The reason is that the TF-IDF method can reduce the query time when retrieving large collections and increase the effective. However, basically the source need has a fixed format so that it will easy to index the information. Moreover, I think cosine similarity measure can be used to identify the similar documents by query to increase the accuracy.

6. Conclusion

The purpose of this course work is to use IR system to compare and evaluation each method's performance. According to the result, the TEXT_TOKEN, stemming and TF-IDF weighting have better performance.

7. Summary of research paper

The beliefs and biases of human can impact people's judgment, decision and behavior. The bias in web search will also influence the results of information retrieval. For example, there are some bias which impact people's behavior in IR: people will prefer special web domains when searching because of domain bias. They are also like high-ranked information of results due to rank bias. Sometimes as caption bias, people favor captions which have certain words in psychology such as anchoring-and–adjustment, confirmation and availability. So these show an opportunity to link between IR and psychology. For example, such as the question that can tea tree oil treat canker sores? It is yes. In real situation, a health searcher may prefer a certain answer based on their beliefs on the knowledge of the oil. They may favor disaffirming results unconsciously. Therefore when users search or are presented with results which deviates from the truth things observably, bias will be found in Web search.

The authors give an experiment about initial exploratory questionnaire firstly. The purpose is to obtain aware of the possibility about biases in web search at beginning. They focus on simplicity Yes-No questions whose answers along single dimension. The participants are Microsoft employees and 198 respondents recalled recent Yes-No query which contain multi-point scale such as Yes, Lean Yes, Equal, Lean No and No. The experiment have two processes, one is to distribute belief before searching, and another is to distribute belief after searching.

The graph about belief distribution before searching shows a positive deflection in respondents of beliefs that 58% of respondents tend to Yes and only 21% tend to No. The remaining 21% of respondents have equal belief. Actually, the sum of lean yes, equal and lean no is 77%. The graph about belief distribution before searching displays that there are less change about the percentage of yes or no, however, the sum of lean yes, equal and lean no reduced to 48% which is three quarters of the respondents before searching. It means there are double of participants who shifted their beliefs

from lean yes to yes after searching. Furthermore, some parts of the participants who believed lean yes or lean no prefer yes or no when confirm the answers after searching on web.

Therefore there are two findings, one is the respondents confirm their beliefs to choose yes or no when they are strongly believed. Initially, they prefer some information to support their belief so they select lean yes or lean no, and once they found the answer, they confirm beliefs to tend to yes or no. Another finding is that when people have no idea about the answer, they prefer choose a positive answer rather than a negative one before and after searching.

The next step is to adopt log-based study of Yes-No queries. The data is based on the queries clicks and the results from Bing logs in two weeks. They collected yes-no questions which start with “can”, “is” and so on. Moreover, they focused on the health information as it is very important and can obtain truth result. Two physicians gave the reviews answer which contain Yes, No, 50/50 and don’t know. Then they use Cohen’s free-marginal kappa (k) inter-rater agreement to analyze the result. They found that it showed substantial agreement with the physician answers, and 55% Yes and 45% No of distribution used as truth in their experiment.

Then they use caption and result content from crowd sourced to analyze labeling content and truth. There are 3 to 5 judges or caption and four assign label of yes, no, both and neither. It shows that the agreement is on 96% captions and according to top ten search results, the one is on 92% of pages with crowd sourced judges. Based on the physician answers as truth, they collect 680 Yes-No health question from logs and the truth from physicians’ judgments for each one which has top 10 search result and click through behavior from the web.

By analyzing ranking of results, they found that more Yes content is in top 10 than No content. Moreover Yes content shows higher rank than No content usually. By analyzing behavior of users, more users prefer to click on the captions with Yes content and skip No to click Yes content. Furthermore, according to the accuracy results of answer, the accurate of top result is only 45% and it is less when truth is No, and users improve the accuracy slightly. Finally, the answer transitions result shows that no one transitioned from Yes to No. It means people prefer confirmatory information rather than changing their hypothesis when searching.

The article found that the web engines prefer rank Yes content above No content and display more results about positive. Meanwhile people also prefer to click Yes content than No content. Sometimes, the engines have wrong answer of top results. It mean the engines adopt behavior, which influence the accuracy of the answer. As the weakness query match of rank algorithm, it will also impact the performance of web search.

I think the article display a great study result to explain the beliefs and biases in web search. However, there are some limited points in the experiment and analysis. Firstly, in initial exploratory questionnaire, the participants are Microsoft employees who have high level of education. They may actually know the answer of question rather than prefer Yes answer. So the participants are not normal. Furthermore, the number of questions with Yes answer may larger than that with No answer, so it will cause wrong awareness that people prefer Yes answer. Thirdly, the result shows that double respondents who select Lean Yes tend to Yes after searching. However, more than three times people also shift their beliefs from Lean No to No after searching. Finally, this article just focuses on Yes-No questions, it may be not representative. Actually a real thing is that it is always hard for people to change his mind about one question.