CIS 833, Fall 2014 **Name:_____**
Exam 1 – 75 minutes

This test consists of five questions. The number of points for each question is shown below.

- Read all questions carefully before starting to answer them.
- Write all your answers on the space provided in the exam paper.
- The order of the questions is arbitrary, so the difficulty may vary from question to question. Don't get stuck by insisting on doing them in order.
- Show your work. Correct answers without justification will not receive full credit. However, also be concise. Excessively verbose answers may be penalized.
- Clearly state any simplifying assumptions you may make when answering a questions.
- **Be sure to write your name on the test paper.**

Question	1	2	3	4	5	total
Points	20	20	20	20	20	100
Your points						

I. MapReduce [20 points] You have millions of records corresponding to user comments to various blog articles. Each record contains:

(blog-article-ID, comment-article-ID, user-ID, comment-date, comment-message)

i. [7 points] Your first task is to use MapReduce to create a list of users with a per-user count of the number of unique blog articles that the user commented on. The input to the mapper will be files containing tuples having the format described above. The output of the reducer will consist of pairs of the form (user, number-of-unique-blog-articles-the-user-commented-on).

```
class MAPPER

method MAP(record_id, record_content)
    blog-article-ID<-get_blog-article-ID_from_record_content
    user-ID<-get_user-ID_from_record_content
    EMIT (user-ID,blog-article-ID)

class REDUCER

method REDUCE(user-ID, list[blog-article-IDs])
    sort(list[blog-article-IDs])
    sorted_list_unique<-remove_duplicated_from_sorted_list
    count<-count_number_of_articles(sorted_list_unique)
    EMIT(user-ID, count)
```

ii. [6 points] Your second task is to use the tabulation of (user, number-of-unique-blog-articles-the-user-commented-on) from i. to find the users with the highest number of blog articles that they commented on. In your solution, you should use the fact that MapReduce sorts by intermediate keys into the reducer groups.

```
//we are reading the file produced in i. – i.e., each line contains user, count
method MAP(line_id, line_content)
split_line_content_into_user_and_count
EMIT (count, user)

//MapReduce will sort by intermediate keys –> count, user pairs will be sorted by count if
//only one reducer is used
method REDUCE (count, list [user1,user2,…])      //intermediate key,value pairs
        // reduce is the identity function
        for each user in list [user1,user2,…]
                EMIT (user, count)
```

iii. [7 points] Your last task is to use MapReduce to compute the number of pairs of users who both commented on the same blog article (each pair of users should be counted only once). The input will be the original files and the output will be (blog-article-ID, count of user pairs).

```
method MAP(record_id, record_content)
    blog-article-ID<-get_blog-article-ID_from_record_content
    user-ID<-get_user-ID_from_record_content
    EMIT (blog-article-ID, user-ID)


method REDUCE(blog-article-ID, list[user-IDs])
    sort(list[user-IDs])
    sorted_list_unique<-remove_duplicated_from_sorted_list
    N<-length(sorted_list_unique)
    count<-N(N-1)/2
    EMIT(blog-article-ID, count)
```

II. Boolean Retrieval [20 points]

Assume the following collection of short documents.

```
Doc 1: alpha bravo charlie delta echo foxtrot golf
Doc 2: golf golf golf delta alpha
Doc 3: bravo charlie bravo echo foxtrot bravo
Doc 4: foxtrot alpha alpha golf golf delta
```

i. [8 points] Construct a term-document matrix that can be used to perform Boolean retrieval. The index terms have already been arranged for you alphabetically in the following table:

Term	Doc1	Doc2	Doc3	Doc4
alpha	1	1	0	1
bravo	1	0	1	0
charlie	1	0	1	0
delta	1	1	0	1
echo	1	0	1	0
foxtrot	1	0	1	1
golf	1	1	0	1

ii. [8 points] What documents will be returned in response to the following queries?

```
bravo AND charlie AND echo
Doc1 and Doc3
```

```
(NOT alpha) AND foxtrot
```
Doc3

```
golf AND (charlie OR (NOT delta)
```
Doc1

iii. [4 points] Discuss two advantages of the Boolean retrieval model.

Can be very efficiently implemented

Predictable, easy to explain

Structured queries for pinpointing precise documents – very expressive

Works well when you know exactly (or roughly) what the collection contains and what you're looking for

III. Inverted Index and Cosine Similarity [20 points]

i. [4 points] How do stop word removal and stemming reduce the size of an inverted index?

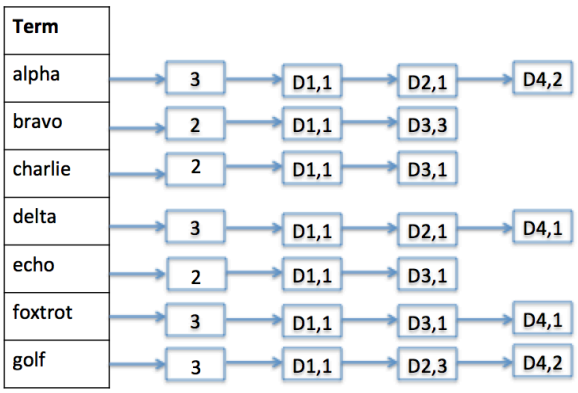
Stop word removal reduces the size of the inverted index by eliminating terms with very long inverted lists.

Stemming reduces the size by consolidating the lists for two or more terms with the same stem

ii. [6 points] Consider again the collection of documents from exercise **II.**

```
Doc 1: alpha bravo charlie delta echo foxtrot golf
Doc 2: golf golf golf delta alpha
Doc 3: bravo charlie bravo echo foxtrot bravo
Doc 4: foxtrot alpha alpha golf golf delta
```

Construct an inverted index for this collection, assuming the same vocabulary terms as in Exercise **II**. Show the index graphically with linked lists.



iii. [4 points] Show how the inverted index can be used to calculate document lengths incrementally using a hashtable (assuming that the number of documents and max term frequency per document are also available).
Document lengths can be calculated incrementally by using using a hashtable for which documents are keys and their lengths are the values. The length hashtable is computed with a pass through the inverted index. As I encounter a document j in the posting list of a term i in the inverted index, I increment the current length (originally 0) for that document with the product w_ij^2 (where w_ij is a TF-IDF weight for the term i in document j).

iv. [6 points] Calculate the cosine similarity between the first two documents, assuming the TF-IDF weighting scheme. Remember that the cosine between two vectors d1 and d2 is given by the following formula:

$$\cos(\vec{d}1,\vec{d}2) = \frac{\vec{d}1 \cdot \vec{d}2}{\left|\vec{d}1\right| \cdot \left|\vec{d}2\right|} = \frac{\sum_{i=1}^{n}(w_{1j} \cdot w_{2j})}{\sqrt{\sum_{i=1}^{n} w_{1j}^{\ 2} \cdot \sum_{i=1}^{n} w_{2j}^{\ 2}}}$$

	alpha	bravo	charlie	delta	echo	foxtrot	golf
DF	3	2	2	3	2	3	3
IDF	log(4/3)	log(4/2)	log(4/2)	log(4/3)	log(4/2)	log(4/3)	log(4/3)
D1 -> TF	1/1	1/1	1/1	1/1	1/1	1/1	1/1
D2-> TF	1/3	0/3	0/3	1/3	0/3	0/3	3/3

$$\cos(\vec{d}1,\vec{d}2) = \frac{w_{1,alpha} * w_{2,alpha} + w_{1,delta} * w_{2,delta} + w_{1,golf} * w_{2,golf}}{\sqrt{(w_{1,alpha}^2 + w_{1,bravo}^2 + w_{1,charlie}^2 + w_{1,delta}^2 + w_{1,echo}^2 + w_{1,foxtrot}^2 + w_{1,golf}^2) * (w_{2,alpha}^2 + w_{2,delta}^2 + w_{2,golf}^2)}}$$

where $w_{i,term}$ is the TF*IDF weight of *term* in document I (obtained by multiplying IDF with TF in the table above)

IV. Latent Semantic Indexing [20 points]

Consider the following term-document matrix corresponding to a collection of six documents represented in terms of four keywords.

i.

Term	Doc1	Doc2	Doc3	Doc4	Doc5	Doc6
Word1	5	15	7	9	7	0
Word2	5	7	1	0	1	0
Word3	1	0	7	4	6	0
Word4	0	1	6	4	0	4

The singular value decomposition of the term-document frequency matrix is: $X=T_0S_0D'_0$ where

$$T_0 = \begin{bmatrix} 0.8817 & 0.1969 & -0.0444 & -0.4264 \\ 0.2887 & 0.4928 & 0.1190 & 0.8122 \\ 0.3033 & -0.6652 & -0.5674 & 0.3790 \\ 0.2173 & -0.5253 & 0.8136 & 0.1222 \end{bmatrix}$$

$$S_0 = \begin{bmatrix} 23.33 & 0 & 0 & 0 \\ 0 & 9.76 & 0 & 0 \\ 0 & 0 & 5.03 & 0 \\ 0 & 0 & 0 & 3.27 \end{bmatrix}$$

$$D_0 = \begin{bmatrix} 0.2638 & 0.6627 & 0.4237 & 0.4293 \\ 0.2850 & 0.6018 & -0.6079 & -0.3061 \\ -0.0385 & 0.1948 & 0.1425 & 0.1162 \\ 0.7038 & -0.1795 & 0.3700 & -0.5590 \\ 0.5557 & -0.3294 & -0.1526 & 0.6077 \\ -0.2090 & 0.1411 & 0.5201 & -0.1635 \end{bmatrix}$$

i. [7 points] We want to reduce the space from four dimensions (word1,word2,word3,word4) to two dimensions (concept1,concept2). How do we accomplish this? Show matrices T, S, D by circling the corresponding entries in matrices T_0, S_0, D_0.

$$T_0 = \begin{bmatrix} 0.8817 & 0.1969 & -0.0444 & -0.4264 \\ 0.2887 & 0.4928 & 0.1190 & 0.8122 \\ 0.3033 & -0.6652 & -0.5674 & 0.3790 \\ 0.2173 & -0.5253 & 0.8136 & 0.1222 \end{bmatrix}$$

$$S_0 = \begin{bmatrix} 23.33 & 0 & 0 & 0 \\ 0 & 9.76 & 0 & 0 \\ 0 & 0 & 5.03 & 0 \\ 0 & 0 & 0 & 3.27 \end{bmatrix}$$

$$D_0 = \begin{bmatrix} 0.2638 & 0.6627 & 0.4237 & 0.4293 \\ 0.2850 & 0.6018 & -0.6079 & -0.3061 \\ -0.0385 & 0.1948 & 0.1425 & 0.1162 \\ 0.7038 & -0.1795 & 0.3700 & -0.5590 \\ 0.5557 & -0.3294 & -0.1526 & 0.6077 \\ -0.2090 & 0.1411 & 0.5201 & -0.1635 \end{bmatrix}$$

ii. [7 points] What is the vector representation for query q = {word1 word3 word1 } in the original vector space (x_q) and in the reduced concept space (d_q)? (Remember that $d_q = x'_q T S^{-1}$).

$x_q = (1,0,1,0)'$

$$d_q = \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0.8817 & 0.1969 \\ 0.2887 & 0.4928 \\ 0.3033 & -0.6652 \\ 0.2173 & -0.5253 \end{bmatrix} \begin{bmatrix} 1/23.33 & 0 \\ 0 & 1/9.76 \end{bmatrix}$$

iii. [6 points] For each of the following two linguistic phenomena, synonymy and polysemy: describe them in a few words; give an example for each; and say if it causes *primarily* a precision or a recall problem.

Synonymy = different words with the same meaning. Example: sick and ill.
Main problem: lowers recall.

Polysemy = same word but different meanings. Example: apple (fruit) and apple (company).
Main problem: lowers precision.

V. IR Evaluation [20 points]

i. [8 points] The table below shows the output of an IR system on two queries. Only top 5 ranks are shown. Crosses correspond to documents which have been judged relevant by a human judge; circles correspond to irrelevant documents. There are no relevant documents in lower ranks (> 5). Compute the MAP.

Rank	Q1	Q2
1	o	x
2	x	o
3	x	o
4	x	o
5	o	x

Average Precision: Average of the precision values at the points at which each relevant document is retrieved.
Mean Average Precision (MAP): Average of the average precision values for a set of queries.

For query 1:

At rank 2, p = ½, r = 1/3

At rank 3, p =2/3, r = 2/3

At rank 3, p = 3/4, r = 3/3 = 1

Avg precision (½ + 2/3 + ¾)/3

For query 2:

At rank 1, p = 1, r = 1/2

At rank 5, p = 2/5, r = 2/2 = 1

Avg precision (1 + 2/5)/2

MAP is [(avg prec for query 1) + (avg prec for query 2)]/2

ii. [7 points] For query Q1 above, plot an exact recall/precision curve and then overlay it with a graph where the precision values are interpolated to the standard 11 points.

The exact recall/precision points are (1/3,1/2), (2/3,2/3), (1,3/4).

Therefore, the interpolated recall/precision points are: (0,3/4), (0.1,3/4), (0.2,3/4), (0.3,3/4), (0.4,3/4), (0.5,3/4), (0.6,3/4), (0.7,3/4), (0.8,3/4), (0.9,3/4), (1,3/4).

iii. [5 points] You have developed a new method for parsing documents that uses semantic information to decide which sentences to index and which to skip. How would you determine whether your method produces better retrieval results than indexing every sentence?

I assume I already have an index that includes every sentence and a query processing engine for that index. Now I will re-parse every document in my collection and re-build the index from scratch. I will use a sample of queries as inputs to both engines, giving me two document rankings for each query—one from the original index, one from the index with the new parsing method. Then I will have assessors judge the relevance of documents, and use those relevance judgments to calculate measures like precision, recall, and average precisions. Whichever engine has the highest precision or recall or average precision is the one that was better.