CSCI5510 Big Data Analytics Assignment 1

***Due Date: 23:59 Oct 11, 2013***

Submission Instruction:

For this assignment, please submit electronic version only. We don’t accept hard copy. For the programming questions, you need to submit BOTH your code and your result. Submit code as zipped tar file and the output of your program in plain-text file. For other questions, answer them in a word document. You should place the relevant files in their separate directory (preferable A, B, C, D, and E for each part).

Then compress all your files as one zip named using your student id and submit to our course account at [csci5510@cse.cuhk.edu.hk](mailto:csci5510@cse.cuhk.edu.hk) with email title “CSCI5510 Assignment You name, Your student ID”.

# Part A. Hadoop Programming

*Start working on the following problem after lecture 2.*

1. In this exercise, you will write a Map Reduce program to find out what are the most used words by William Shakespeare. Provided is the complete works of Shakespeare in txt format, by the courtesy of Gutenberg Project. (Hint: You can leave the final sorting step to a UNIX utility called “sort”. Your MapReduce program needs only to count the appearance of each word. Do pay attention to punctuation marks and the stopwords.)

1. Please write a map reduce program to list the top 200 words used by Shakespeare. You should ignore all the punctuation marks and count capital words the same as small words. (It means that you have to take care of converting one to another in your program). Take a look at the list. Is this a representative list? Why?
2. English stop words are those words that are used so common in all kinds of text that it is better to exclude them when doing statistics. Provided are two lists of stop words (stopwords1.txt, stopwords2.txt). Now modify your program to exclude any words from consideration which has appeared in the list. Now what are the top 200 used words by Shakespeare?

For this question, please submit all your code as a zipped tar file and the results for sub-question 1 and 2 in their respective plain-text files.

# Part B. Locality Sensitive Hashing

*Start working on the following problem after lecture 3.*

In this question, you are going to implement nearest-neighbor search for text documents. For your convenience, we only give you bag-of-words based data which have been preprocessed so that you do not need to do shingling. Your task is to implement minhashing, and locally sensitive hashing. We split it into two parts:

1. Implement a module, that given a collection of sets of objects (e.g., strings, or numbers), creates a minhashing based signature for each set.

2. Implement a module that implements the locally sensitive hashing (LSH) technique, so that, given a collection of minhashing signatures of a set of documents and a document ID, it finds top 100 nearest neighbors of this document.

To test the LSH algorithm, also implement a class that given the words of each document and a document ID, it finds the top 100 nearest neighbors of this document by comparing all the word sets with it. Please use the Jaccard similarity as the similarity measure. If a word appears multiple times in one document, you should also count multiple times when computing the similarity.

To apply the algorithm you have the following tasks:

1. Find the nearest neighbors of **Doc. 1** among all the documents using minhashing and LSH. **Sth wrong when I run it**
2. Find the nearest neighbors of **Doc. 1** among all the documents by comparing all the word sets. It’s still running when I commit this program
3. Report the size of intersection of the results in step 2 and 3.
4. Report the time required to compute the near neighbors in step 2 and 3.

3m14.215s in generating the sigmatrix

about 6s in lsh

1. Report the number of hash functions used in minhashing. **100**
2. Report values for r and b (see Section 3.4 in the book) that can give us the desired behavior. To plot the graph that gives the probability as a function of the similarity for different values of r and b you can use, for example, R.

About the data:

We have prepared both .mat and .txt file, you can either one as you want. For each pair *(i, j, val)*, it means that word *j* appears *val* times in document *i*. For the detailed information of this dataset, please refer to <http://www.cad.zju.edu.cn/home/dengcai/Data/TextData.html>

For this question, submit your code as zipped tar file. Submit your program’s output for sub-question 1 and 2 as plain-text file (list the nearest neighbors’ id in decreasing order). Answer sub-question 3-6 in the word file.

# Part C. Mining Data Streams

*Start working on the following problems after lecture 4.*

1. Suppose we have a stream of tuples with the schema: (university, courseID, studentID, grade).

Assume universities are unique, but a courseID is unique only within a university (i.e., different universities may have different courses with the same ID, e.g., “CS101”) and likewise, studentID’s are unique only within a university (different universities may assign the same ID to different students). Suppose we want to answer certain queries approximately from a 1/20th sample of the data. For each of the queries below, indicate how you would construct the sample. That is, tell what the key attributes should be.

1. For each university, estimate the average number of students in a course.

**(university, courseID)**

1. Estimate the fraction of students who have a GPA of 3.5 or more.

**(university, studentID)**

2. Suppose that, for fixed k and n, we want to design a true Bloom filter, using h hash functions. The question I would like to ask is “what value of h, as a function of k and n, produces the smallest probability of a member x of F that is not in S becoming a false positive?” Remember that increasing h gives us more chances of finding a hash function that maps x to a 0, but it also increases the density of 1’s in the bit array, thus making it less likely that any given hash function will find a 0. The above question is very hard, but feels free to work on it if you like, and we’ll be pleased to accept that as your answer to this question. A somewhat easier question, which you should be able to work out without your head exploding, is to find the conditions on k and n such that using two hash functions gives a lower false-positive rate than using only one hash function. For example, if k and n are such that one hash function only puts 1’s in 1/10th of the bit array (i.e., e -k/n = .9), then adding a second hash function lowers the false-positive rate. (With one hash function, the false-positive rate is 1/10, while with two hash functions the false-positive rate is (.19)2 = .0361. However, if k and n put 1’s in half of the bit array (i.e., e -k/n = .5), then adding a second hash function causes there to be more false positives (probability of a false positive = 9/16) than if only one hash function is used (probability of a false positive = ½).

**Answer:**

**K = 10 million**

**n = 100 million**

3. Usually we use DGIM algorithm to count ones in a bit stream. Now we have a length ten-thousand-bit stream (provided in stream\_data.txt). You are required to program to count ones in last one thousand bits for this stream through DGIM algorithm. Output should demonstrate how you set the buckets and your estimate.

Updating rules:

1. If the current bit is 0, no other changes are needed;
2. If the current bit is 1:
3. Create a new bucket of size 1, for just this bit, and end timestamp = current time;
4. If there are now three buckets of size 1, combine the oldest two into a bucket of size 2;
5. If there are now three buckets of size 2, combine the oldest two into a bucket of size 4;
6. And analogize for the rest.

(Hints: you cannot directly count ones; that is wrong answer.)

For this problem, submit your code as a zipped tar file and describe your buckets (the separation of buckets) and your estimate in the word file.

**Answer:**

**==winList===[ts, value]=========**

**[9998, 1] [9994, 2] [9989, 4] [9981, 8] [9948, 16] [9911, 32] [9835, 64] [9663, 128] [9349, 128] [9019, 256]**

**1's in last 1000bits ~= 511 in fact = 391**

# Part D. Scalable Clustering

*Start working on the following problems after lecture 5.*

1. For the three clusters of the following figure, compute

1. The representation of the cluster as in the BFR Algorithm. That is, compute N, SUM, and SUMSQ.

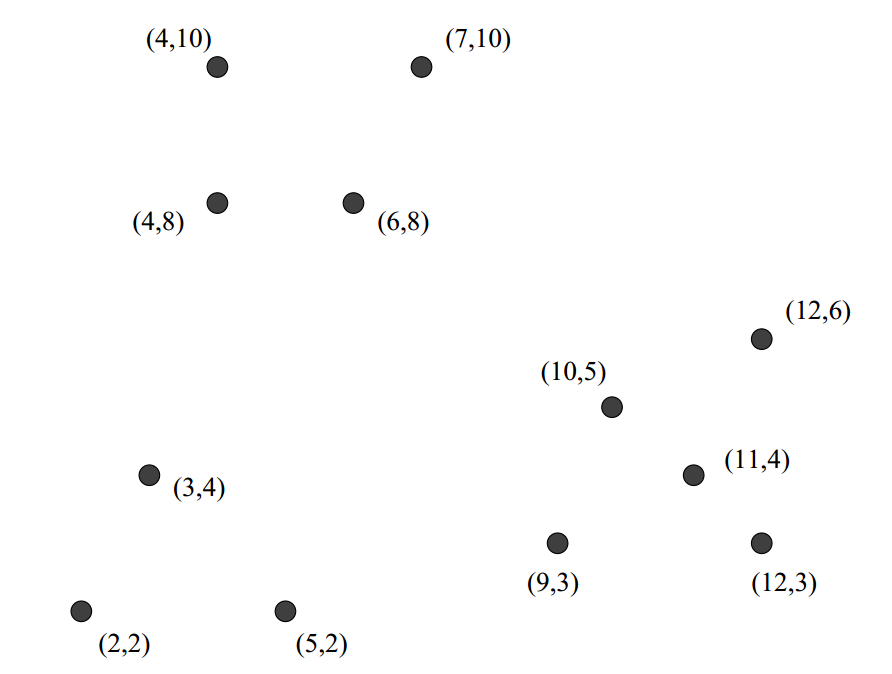
**(N, SUM, SUMSQ)**

**(4, [21, 36], [117, 328])**

**(3, [10, 8], [38, 24])**

**(5, [54, 21], [590, 95])**

1. The variance and standard deviation of each cluster in each of the two dimensions.



cluster: (4, [21, 36], [117, 328])

variance and standard deviation:

[(1.6875, 1.29903), (1.0, 1.0)]

=======

cluster: (3, [10, 8], [38, 24])

variance and standard deviation:

[(1.55555, 1.24721), (0.88888, 0.94280)]

=======

cluster: (5, [54, 21], [590, 95])

variance and standard deviation :

[(1.35999, 1.166190), (1.35999, 1.16619)]

2. Suppose a cluster of three-dimensional points has standard deviations of 2, 3, and 5, in the three dimensions, in that order. Compute the Mahalanobis distance between the origin (0,0,0) and the point (1, −3,4).

**Sqrt(189)/10**