ITI 1121. Introduction to Computing II
Winter 2013

Assignment 1
(Last modified on January 15, 2013)

Deadline: Friday February 1, 2013, 18:00

[PDF]

Learning objectives

- Using arrays and matrices to implement complex types
- Recognize the role of reference variable to create class associations
- Applying basic of object oriented programming principles to solve problems
- Constructing software systems using one-to-many class associations
- Editing, compiling, debugging, and running Java programs
- Raising awareness concerning the university policies for academic fraud

Background information

Analytics[1] is one of "the four key growth areas of Information and Communications Technology (ICT)[2]". A Google search for "analytics jobs" returns over 50 million Web documents! Analytics aims to extract meaningful information (knowledge) from large and complex data-sets. One of its challenges is to effectively and efficiently compare diverse documents for the purpose of finding similarities. This has applications for searching the Web, searching the literature, detecting plagiarism, detecting intrusions in computer systems, searching genomic data bases, etc. For simplicity, we will limit ourselves to strings representing genomic data (genes). However, the methodology can also be applied to other forms of documents, including text, music, video, and stock exchange data.

Problem statement

How does one compare documents whose length and content are different? One approach, which we explore here, is to transform documents into feature vectors.

In "text mining", a k**-gram** (or word) is a contiguous sequence of k symbols over some alphabet. Given a four letter alphabet, A,C,G,T, there is a total $4^2 = 16$ 2-grams. A **profile** gives the number of occurrences of k-grams in a document. Given an input String of length n, there are $n - k + 1$ overlapping strings (k-grams) of length k.

As an illustration, the complete mitochondrial genome of Neanderthal is a string of length 16,565 over a four-letter alphabet (A,C,G,T):

```
>gi|196123578|ref|NC_011137.1| Homo sapiens neanderthalensis mitochondrion
GATCACAGGTCTATCACCCTATTAACCACTCACGGGAGCTCTCCATGCATTTGGTATTTTCGTCTGGGGG...(16565)...T
TGGGGGTAGCTAAAGTGAACTGTATCCGACATCTGGTTCCTACTTCAGGGCCATAAAGCCTAAATAGCCCACACGTTCCCCTTA
AATAAGACATCACGATG
```

Its profile is a vector of counts of size 16:

```
Profile: [words = AA(1589),AC(1490),AG(799),AT(1229),CA(1533),CC(1769),...,TT(1016)]
```

The above profile shows that the 2-gram AA was found 1,589 times in the mitochondrial genome of Neanderthal, whilst AC was found 1,490 times. Any genomic sequence can be transformed into a profile, and compared to that of Neanderthal.

[1] A term akin to "Big Data" and "Data Mining"
[2] IT worker shortage has serious implications for Canada by Dan Oysey, Financial Post, December 11, 2012.

Implementation

This assignment considers two implementations of a profile. In the first implementation, all possible k-grams are enumerated and stored into memory. Accordingly, this implementation will be called **WordProfile**. For the second implementation, each k-gram is interpreted as a base 4 number. These base 4 numbers represent the index of the k-grams in the profile. Accordingly, this implementation will be called **HashProfile**.

1 Rules and regulation (10 marks)

Follow all the directives available on the assignment directives web page, and submit your assignment through the on-line submission system uottawa.blackboard.com.

You must preferably do the assignment in teams of two, but you can also do the assignment individually. Pay attention to the directives and answer all the following questions.

2 Word (10 marks)

The class **Word** will be used for the implementation of **WordProfile**. Here are the characteristics of the class **Word**.

1. A **Word** object stores a k-gram (a String of size k) as well as a count. The initial value of the count is zero.
2. The class **Word** has a single constructor. It has a single formal parameter, **word**.
3. ~~**void count**(): A **Word** object has a method **count** that increments the counter of this object by one.~~
4. ~~**int getCount**(): The method **getCount** returns the current value of the counter.~~
5. ~~**String getWord**(): The method **getWord** returns the k-gram represented by this object.~~
6. ~~Finally, each **Word** has a **toString()** method that returns a String object consisting of the k-gram represented by this object, as well as the count, in between parentheses.~~

3 WordProfile (30 marks)

A **WordProfile** stores the number of occurrences of possible k-grams in a given input string. The k-grams are stored and returned in lexicographic order.

1. **WordProfile** uses a array of objects of the class **Word** to tally the counts of all k-grams for a given input String.
2. Has a constructor with the following signature, **WordProfile(int k, String input)**, where k is the length of the k-grams and **input** is a reference to a **String** object.
3. **int getSize()**: returns the length of the profile.
4. **String getWord(int index)**: returns the word (k-gram) at the designated position.
5. **int getCount(int i)**: returns the count the ith word.
6. **int getCounts()**: returns the total number of k-grams examined from the input.
7. **double getDistance(WordProfile other)**: returns the distance between *this* and the **other** profile. For this assignment, we measure the Euclidean distance of the two profiles. Some care must be taken so that documents of different lengths can be compared. Indeed, longer documents will display higher count values. We therefore normalize each count of a profile by dividing it by the total number of counts. Each value of the profile is now seen as a frequency. Here is the definition of the Euclidean distance for vectors of size d:

$$((x_0 - y_0)^2 + (x_1 - y_1)^2 + \ldots + (x_{d-1} - y_{d-1})^2)^{\frac{1}{2}}$$

where x_0 is the frequency of the first k-gram in this profile, that is the count of the first k-gram divided by the total number of k-grams in the input, y_0 has the same definition for the other profile. Given an alphabet of size 4, the are $d = 4^k$ distinct k-grams.

8. **String getMostFrequentWord()**: returns the word that has the highest count.

9. **String toString()**: returns a String representation of the project. Specifically, it displays all the k-grams together with their counts.

4 HashProfile (30 marks)

A **HashProfile** stores the number of occurrences of k-grams in a given input string. Unlike **WordProfile**, the k-grams are not explicitly stored! Instead, internally, each k-gram is interpreted as a base 4 number, which represents the index of the k-gram in the array of counts. Be careful, the counts for k-grams must be stored in k-grams lexicographic order. Therefore, **getWord(0)** will return AA for both, a **WordProfile** object and a **HashProfile** object, assuming $k = 2$ and the alphabet is A,C,G,T.

1. **HashProfile** uses an array of integers to tally the counts of all k-grams for a given input.

2. Has a constructor with the following signature, **HashProfile(int k, String input)**, where k is the length of the k-grams and **input** is a reference to a **String**.

3. **int getSize()**: returns the length of the profile.

4. **String getWord(int index)**: returns the word (k-gram) at the designated position.

5. **int getCount(int i)**: returns the count of the ith word.

6. **int getCounts()**: returns the total number of k-grams in the input.

7. **double getDistance(WordProfile other)**: returns the distance between *this* and the **other** profile. For this assignment, we measure the Euclidean distance of the two profiles. Some care must be taken so that documents of different lengths can be compared. Indeed, longer documents will display higher count values. We therefore normalize each count of a profile by dividing it by the total number of counts. Each value of the profile is now seen as a frequency. Here is the definition of the Euclidean distance for vectors of size d:

$$((x_0 - y_0)^2 + (x_1 - y_1)^2 + \ldots + (x_{d-1} - y_{d-1})^2)^{\frac{1}{2}}$$

   where x_0 is the frequency of the first k-gram in this profile, that is the count of the first k-gram divided by the total number of k-grams in the input, y_0 has the same definition for the other profile. Given an alphabet of size 4, the are $d = 4^k$ distinct k-grams.

8. **String getMostFrequentWord()**: returns the word that has the highest count.

9. **String toString()**: returns a String representation of the project. Specifically, it displays all the k-grams together with their counts.

5 WordCompareAll and HashCompareAll (15 marks)

WordCompareAll and **HashCompareAll** are two test programs. Their content will be identical, except that **WordCompareAll** uses **WordProfile** objects, whereas **HashCompareAll** uses **HashProfile** objects. The programs are reading the arguments from the command line:

```
java WordCompareAll 2 toxin.txt unknown_a.txt unknown_b.txt
```

where k is the length of the k-grams. This is followed by a list of file names. Here is the behaviour of the test programs.

1. Display the student information.

2. Read and create a profile for each text file given as an argument to the program.

3. Display all the profiles.

4. Display the similarity of all the pairs of profiles.

5. Display the most frequent word for each profile

6 Application (5 marks)

Cases of food poisoning are unfortunately quite common — "Two people are paralyzed after drinking botulism-contaminated carrot juice s(...)" from Botulism-tainted juice paralyzes two in Canada, www.reuters.com, 2006/10/17 — we have been asked to help the investigators.

Background information. Botulism is a rare disease caused by a toxin (a protein) expressed by the bacterium *Clostridium botulinum*. From the above source of information: "Botulism can cause nausea, fatigue, double-vision, paralysis and respiratory failure. In severe cases, the toxin can be fatal."

The investigators suspect that at least one of the following proteins, Unknown A or Unknown B, has a similar domain to that of *Clostridium botulinum*'s toxin. Which protein(s) is (are) related to this toxin? Why?

```
>gi|27867582 (fragment of the known Clostridium botuninum toxin gene)
GTGAATCAGCACCTGGACTTTCAGATGAAAAATTAAATTTAACTATCCAAAATGATGCTT
ATATACCAAAATATGATTCTAATGGAACAAGTGATATAGAACAACATGATGTTAATGAAC
TTAATGTATTTTTCTATTTAGATGCACAGAAAGTGCCCGAAGGTGAAAATAATGTCAATC
TCACCTCTTCAATTGATACAGCATTATTAGAACAACCTAAAATATATACATTTTTTTCAT
CAGAATTTATTAATAATGTCAATAAACCTGTGCAAGCAGC

> Unknown A
TCTATCAAGTAGATTATTAAATACTACTGCACAAAATGATTCTTACGTTCCAAAATATGA
TTCTAATGGTACAAGTGAAATAAAGGAATATACTGTTGATAAACTAAATGTATTTTTCTA
TTTATATGCACAAAAAGCTCCTGAAGGTGAAAGTGCTATAAGTTTAACTTCTTCAGTTAA
TACAGCATTATTAGATGCATCTAAAGTTTATACGTTTTTTTCTTCAGATTTTATTAATAC

> Unknown B
TCCTGGCTCAGGACGAACGCTGGCGGCGTGTGCTTAACACATGCAAGTCGAGCGATGAAG
CTTCCTTCGGGAAGTGGATTAGCGGCGGACGGGTGAGTAACACGTGGGTAACCTGCCTCA
AAGTGGGGGATAGCCTTCCGAAAGGAAGATTAATACCGCATAACATAAGAGAATCGCATG
ATTTTCTTATCAAAGATTTATTGCTTTGAGATGGACCCGCGGCGCATTAGCTAGTTGGTA
```

7 Academic fraud

This part of the assignment is meant to raise awareness concerning plagiarism and academic fraud. Please read the following documents:

- http://web5.uottawa.ca/admingov/regulations.html#r71
- http://web5.uottawa.ca/mcs-smc/academicintegrity/
- http://www.uottawa.ca/plagiarism.pdf

Cases of plagiarism will be dealt with according to the university regulations.

By submitting this assignment, you acknowledge:

1. **having read the above documents, and**
2. **understanding the consequences of plagiarism**

Files

You must hand in the following files (in .jar file).

- README.txt
- `Word.java`
- `WordProfile.java`
- `WordCompareAll.java`
- `HashProfile.java`

- `HashCompareAll.java`
- Utilities.java
- StudentInfo.java
- toxin.txt
- unknown_a.txt
- unknown_b.txt

The JavaDoc of the classes can be found here: Documentation.

A Frequently Asked Questions (FAQ)

1. **"none for now"**

Last Modified: January 15, 2013