Assignment A04 - Synonyms

CSCI 242 - Computer Science II :: Version 1.0.0

Due date is posted in the LMS.

Please read the entire assignment description before beginning the project.

Objective

The objective of this assignment is to reinforce the concepts of generics and collections in Java. To accomplish this, you will implement a program that uses semantic similarity to answer questions about synonyms.

Introduction

"The universe may have a purpose, but nothing we know suggests that, if so, this purpose has any similarity to ours."
- *Bertrand Russell*

Background

One type of question encountered in the SAT and the Test of English as a Foreign Language (TOEFL) is the "Synonym Question". In questions like this, students are asked to pick a synonym of a word from a list of alternatives. For example:

1. Vexed
    a. synonym
    b. annoyed
    c. book
    d. spellbound

The correct answer is, b, annoyed.

For this assignment, you will build an intelligent program that can learn to answer questions like the one shown above. In order to do that, the program will approximate the ***semantic similarity*** of any pair of words. The semantic similarity between two words is the measure of the closeness of their meanings. For example, the semantic similarity between "car" and "vehicle" is high, while that between "car" and "flower" is low. In order to answer the question above, you will compute the semantic similarity between the word (vexed) and each possible answers, and pick the answer with the highest semantic similarity.

Semantic Similarity

So how does semantic similarity work? Given a word w and a list of potential synonyms s_1, s_2, s_3, s_4, we compute the similarities of (w, s_1), (w, s_2), (w, s_3), (w, s_4) and choose the word whose similarity to w is the highest. We will measure the semantic similarity of pairs of words by first computing a ***semantic descriptor vector*** for each of the words. We will then measure the similarity by using the ***cosine similarity*** between the two vectors.

Given a text with n words denoted by $(w_1, w_2, \ldots, w_n)$ and a word w, let $desc_w$ be the semantic descriptor vector of w computed using the text. $desc_w$ is an n-dimensional vector where the i-th coordinate of $desc_w$ is the number of sentences in which both w and w_i occur.

A HashMap Implementation

A handy implementation of the semantic descriptor vector is to use a HashMap. Recall that a HashMap is defined as `HashMap<K, V>` where `K` is the key and `V` is the value. The key in the HashMap is the word under consideration. This would be the word "Vexed" in the example above. The value of the HashMap is another HashMap, where the keys are words and values are counts. The declaration for your HashMap will be something like:

```
private HashMap<String, HashMap<String, Integer>> descriptors;
```

For example, suppose we are given the following text (the opening of *Notes from the Underground* by Fyodor Dostoyevsky, translated by Constance Garnett):

> *I am a sick man. I am a spiteful man. I am an unattractive man. I believe my liver is diseased. However, I know nothing at all about my disease, and do not know for certain what ails me.*

The word "man" only appears in the *first three sentences*. Its semantic descriptor vector would be:

```
{
    "i": 3,
    "am": 3,
    "a": 2,
    "sick": 1,
    "spiteful": 1,
    "an": 1,
    "unattractive": 1
}
```

The word "liver" occurs in the *second sentence only*, so its semantic descriptor vector is:

```
{
    "i": 1,
    "believe": 1,
    "my": 1,
    "is": 1,
    "diseased": 1
}
```

We store all words in all-lowercase, since we do not consider case to be a factor. For example, "Man" and "man" are the same words. We do, however, consider, e.g., "believe" and "believes", or "am" and "is" to be different words. We discard all punctuation.

Make sure you understand the data structure and how the data is stored before moving on… The LMS contains an example of how to iterate over a HashMap.

Cosine Similarity

Now that we understand the data structure, we need to use it to calculate the semantic similarity. Researchers have suggested many ways of measuring semantic similarity. We will use a fairly straightforward method known as the ***cosine similarity***.

The cosine similarity between two vectors $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$ and $\mathbf{B} = \{B_1, B_2, \ldots, B_n\}$ is defined as:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\sqrt{\sum\limits_{i=1}^{n} B_i^2}}$$

We cannot apply the formula directly to our semantic descriptors since we do not store the entries which are equal to zero. However, we can still compute the cosine similarity between vectors by only considering the positive entries. For example, to compute the cosine similarity of "man" and "liver" given the semantic descriptors above, we apply the cosine similarity formula to the two vectors:

$$\frac{3 \cdot 1 \text{ (for the word "i")}}{\sqrt{(3^2+3^2+2^2+1^2+1^2+1^2+1^2)(1^2+1^2+1^2+1^2+1^2)}} = 3/\sqrt{130} = 0.2631\ldots$$

Corpus

The final item you need to know is, what is a corpus? In linguistics (which is what we are doing here), a ***corpus*** (plural corpora) or ***text corpus*** is a large and structured set of texts that are used to do statistical analysis and testing, checking occurrences or validating linguistic rules within a specific language. The large set of text give us enough data to do real statistical analysis such as cosine similarity. We will define and use a corpus to test and run our synonym programs.

The Project

You are required to use IntelliJ IDEA for your assignment. The following is your project information:

Package:	`edu.uwp.cs.csci242.assignments.a04.synonyms`
IntelliJ Project Name:	`A04_Synonyms`
Zip filename:	`A04.zip`

The Assignment

Your assignment is to implement a Java program to compute the semantic similarity of words using the cosine similarity method. Ultimately, you will be able to answer the "Synonym question" posed at the beginning of this assignment.

You are free to design your program any way you desire. However, there are some requirements you must follow:

1. Leave your testing code in your `main()`. We want to see that you tested all of the functionality.

To perhaps give you a head start or something to think about, the following collection of methods may prove useful…

`public Synonyms ( URL[] corpus )`

The constructor instantiates the data structures and then populates them by calling `parseCorpus()`.

`public void parseCorpus ( URL[] corpus )`

The corpus will consist of a set of web pages of books at Project Gutenberg. The corpus will be used to build the HashMap.

This method should:

1. Open each URL in turn.
2. Read the URL a sentence at a time. To read a sentence one sentence at a time, you can set the delimiter of your scanner with `useDelimiter ( "[\\.\\?\\!]|\\Z" )`.
3. Split the sentence into words, stripping out punctuation and making the words lowercase.

   Notes:

   - An easy way to strip out punctuation is to use `replaceAll ( \\W+, "" )`.
   - The `String` class has a `split()` method. If you split around `"\\s+"` you will be splitting the sentence around whitespace characters.
   - The LMS has an example of how to read URL's.

4. Update the HashMap.

Obviously, the urls you choose for the corpus make a huge difference. For example, if you want to explore synonyms for computer science terms, you would load a bunch of CS texts. For purposes of this assignment, we will hardcode in some classic literature links.

```
URL[] corpus =
{
    // Pride and Prejudice, by Jane Austen
    new URL ( "https://www.gutenberg.org/files/1342/1342-0.txt" ),

    // The Adventures of Sherlock Holmes, by A. Conan Doyle
    new URL ( "http://www.gutenberg.org/cache/epub/1661/pg1661.txt" ),

    // A Tale of Two Cities, by Charles Dickens
    new URL ( "https://www.gutenberg.org/files/98/98-0.txt" ),

    // Alice's Adventures In Wonderland, by Lewis Carroll
    new URL ( "https://www.gutenberg.org/files/11/11-0.txt" ),

    // Moby Dick; or The Whale, by Herman Melville
    new URL ( "https://www.gutenberg.org/files/2701/2701-0.txt" ),

    // War and Peace, by Leo Tolstoy
    new URL ( "https://www.gutenberg.org/files/2600/2600-0.txt" ),

    // The Importance of Being Earnest, by Oscar Wilde
    new URL ( "http://www.gutenberg.org/cache/epub/844/pg844.txt" ),

    // The Wisdom of Father Brown, by G.K. Chesterton
    new URL ( "https://www.gutenberg.org/files/223/223-0.txt" ),
};
```

double calculateCosineSimilarity (String word1, String word2)

A public method that computes the Cosine Similatity of two words. If you look at the formula above, you see that there are three sums computed: one vector dot product, and two vector magnitudes. The correct way to tackle this is to write two private methods to make these calculations.

There is one special case: `word1` or `word2` might not appear anywhere in the corpus. If this is true, your method should return a similarity value of `-1.0`.

`public static void main ( String [] args )`

The main method should contain a loop, where the user enters the word on the first line of input and the list of choices on the second line. It then calculates the best match and prints it. For purposes of grading this assignment, please print each choice and its cosine similarity before printing the final result. The process repeats until the user enters a blank line for the word.

Output

Here is a sample run from your instructor's program:

```
Enter a word:
vexed
Enter the choices
synonym annoyed book spellbound
   synonym -1.0
   annoyed 0.6924426938376693
   book 0.082729574126462
   spellbound 0.6391785522199395
annoyed
```

```
Enter a word:
provincial
Enter the choices
rural cosmopolitan forested horse
   rural 0.8741068346646507
   cosmopolitan 0.42806939662558796
   forested -1.0
   horse 0.8668406391340332
rural
```

Submission

Zip up your entire project folder into a single zip file named `A04.zip` and submit that one file. Submit the Zip file to the LMS "dropbox" labeled `Assignment A04 – Synonyms >>> DROPBOX`. Your instructor should be able to unzip the entire project and load the entire project their IDE without incident.

Grading

The LMS has a rubric attached to the assignment. This rubric is similar to the one shown in the document titled, "Programming Requirements and Rubric" found in the LMS.

Good software engineering is expected. Remember to follow the Coding Guidelines 1.0.0 found in the LMS. Use lots of comments throughout the code, lots of methods, and appropriate variable names. Make sure you validate your input.

Up to 10 points Extra Credit

The Cosine similarity measure described above is only one possible measure of similarity, and is less than perfect. Looking at the example on the right above, we can scratch our heads and ask, what do "provincial" and "horse" have in common to earn a 0.8668 score? The answer if very little.

Here are some variations on similarity measures that you can try.

1. When we write, we tend to use a lot of short words, e.g. a, is, the, for, was, it, and, do, but, etc. Most of these words are used a lot, but carry very little semantic information. Try calculating the cosine similarity only using words with length greater than or equal to 4. You will probably see all you scores decrease, but the most similar words will decrease less, giving you fewer incorrect answers. To implement, overload your `calculateCosineSimilarity()` method.
2. If you think about each descriptor vector as an n-dimensional point, then we can calculate the Euclidean distance between two vectors. We will let you look up the distance formula. In this case, we will want to pick the words with the minimum distance between them. The easiest way to do this is to negate their distance and still choose the maximum value. Modify your code to use Euclidean distance instead of cosine similarity by implementing a new method.

Ω