CS2302 - Data Structures

Fall 2017

Lab # 5

Word Similarity

Deadline: Friday, November 3, 11:59 p.m.

Natural Language Processing (NLP) is the sub-field of artificial intelligence that deals with designing algorithms, programs, and systems that can understand human languages in written and spoken forms. Word embeddings are a recent advance in NLP that consists of representing words by vectors in such a way that if two words have similar meanings, their embeddings are also similar. See https://nlp.stanford.edu/projects/glove/ for an overview of this interesting research.

In order to work in real-time, NLP applications such as Siri and Alexa use hash tables to efficiently retrieve the embeddings given their corresponding words. In this lab you will implement a simple version of this.

The web page mentioned above contains links to files that contain word embeddings of various lengths for various vocabulary sizes. Use the file "glove.6B.50d.txt" which contains word embeddings of length 50 for a very large number of words. Each line in the file starts with the word being described, followed by 50 floating point numbers that represent the word's vector description (the embedding). The words are ordered by frequency of usage, so "the" is the first word. Some "words" do not start with an alphabetic character (for example "," and "."); feel free to ignore them.

Your task for this lab is to write a program that does the following:

1. Read the file "glove.6B.50d.txt" and store each word and its embedding in a hash table. Choose a prime number for your initial table size and increase the size to twice the current size plus one every time the load factor exceeds a predefined threshold.
2. Read another file containing pairs of words (two words per line) and for every pair of words find and display the "similarity" of the words. To find the similarity of words w_0 and w_1, with embeddings e_0 and e_1, we use the cosine distance, which ranges from -1 to 1, given by:

$$sim(w_0, w_1) = \frac{e_0 \cdot e_1}{|e_0||e_1|}$$

   where $e_0 \cdot e_1$ is the *dot product* of e_0 and e_1 and $|e_0|$ and $|e_1|$ are the magnitudes of e_0 and e_1.
3. Analyze the distribution of words in your hash table to verify that your hash function distributes the words as evenly as possible among the lists. Compute the following items from your table:
   (a) Load factor
   (b) The percentage of the lists in the table that are empty (lower is better)
   (c) The standard deviation of the lengths of the lists (lower is better)

Since the key used for hashing is a string, you need to convert it to an integer value in a way that would ultimately result in as few collisions as possible. A simple way is to add the int values of all the characters in the string, and then apply the mod operation. A better way is to consider strings as numbers in a base 26 alphabet. Your grade will depend partly on the quality of the hash function you design. You are not allowed to use the *hashCode* function provided in java.

As usual, write a report describing your work. See the appendix for a hypothetical sample run and the class definition for your hash table.

Appendix

Sample run:

```
Word similarities:
barley shrimp 0.5352693296408768
barley oat 0.6695943991361065
federer baseball 0.28697851474308855
federer tennis 0.7167608209373065
harvard stanford 0.8466463229405593
harvard utep 0.06842559024883524
harvard ant -0.026703792920826475
raven crow 0.615012250504612
raven whale 0.32908915459219595
spain france 0.7909148906835685
spain mexico 0.7513763544646808
mexico france 0.5477963415949284

Table stats:
Load factor: 1.45
Number of empty lists: 23
Standard deviation of the lengths of the lists: 1.56
```

Hash table class definition.

```
public class sNode{
   public String word;
   public float[] embedding;
   public sNode next;

   public sNode(String S, float[] E, sNode N){
      word = S;
      embedding = new float[50];
      for (int i=0;i<50;i++)
         embedding[i] = E[i];
      next = N;
   }
}

public class hashTableStrings{
private sNode [] H;

  public hashTableStrings(int n){ // Initialize all lists to null
      H = new sNode[n];
      for(int i=0;i<n;i++)
         H[i] = null;
   }
}
```