COM-35702 Design and Engineering of Intelligent Information Systems
Engineering and Error Analysis with UIMA
ITAM

Tania Patiño
137175
25-04-2014

Task 1.3 Analysis Engine

Introduction

This task is about the implementation and refactor Java code that at first was created when the hw4 project was made following the section about the specification related to Creating Maven project from the archetype.

1. Correctly extract bag of words feature vector from the input text collection

Bag of words, is a model that analyses the occurrence of words that maybe are on paragraphs, long pieces of text, record talks those examples are part of the area (Natural Language Processing). On the other hand, if a classifier wants to analyze images or sounds, then applies bag of word model to organize important characteristics (features), where each part is like a puzzle piece, this corresponds to (Computer Vision Topics). In both cases, the main idea is to find specific features and make some data training and then make some data test to know which the behavior of the data is, when this model is applied.

Description about the files on the hw4-137175:

These are the java files that are part of this project: VectorSpaceRetrieval.java, DocumentVectorAnnotator.java, RetrievalEvaluator.java,DocumentReader.java,Document_Type.java,Document.java,Token_Type.java,Token.java, Utils.java. Others classes are Document_Type.java, Document.java, Token_Type.java, Token.java that extends from Annotation and Annotation_Type class. Each of those ones analyses in the document: frequence, features, relevanceValue, about words.

The next ones are the Java classes that were modified:

RetrievalEvaluator.java: methods that were constructed: global word dictionary, cosine similarity and ***MMR (Mean***

Reciprocal Rank) metric but this one was not finished so I do not added in this version.

DocumentVectorAnnotator.java: a vector of tokens was constructed and update the tokenList in CAS.

DocumentReader.java: read and parser all document, gives a value to each keyword that finds with a possible

feature.

VectorSpaceRetrieval.java: contains the main of the program which implements the aggregate analysis engine

VectorSpaceRetrieval.xml on a text corpus. This file contains the definition type system where is located a “Document”, “Token” type.

This class was created into the project, that shows the bag of words:

Bag of Words.java: this contains the description about a bag of words, the bag was stored into:

```
//bag
ArrayList<String> bag = new ArrayList<String>();
```

Testing **VectorSpaceRetrieval.java:**

Bag of words Implementation from the input text collection: **document.txt**

Experiments with Bag of Words and document.txt files, to know what and which data is located on the Bag and in the textFile. In the file **Bag_Words.java** are the code lines, related to how to load a file.txt, in a structure like Array List where many queries could be made finding the number of sentences, the file length, how many words are on the file, make some test searching in an specific position of the Array and know which sentence is located, like in document.txt returns the sentence:

Test:qid=3 rel=0 The best antiques are old friends

Extract bag of words features vector from the input text collection there are two options one is with Cleatk and the other one is with Lucene.snowball many test were made but finally the second option was applied.

A toolCleartk , was found to make the extraction bag of words features from document.txt file. Cleartk-release-1.4.1-bin.zip, this folder contains the next jar files:

Add this line in pom.xml file

```
<!--clearTK -->
<!-- Declare cleartk-release as your parent -->
<parent>
<groupId>org.cleartk</groupId>
<artifactId>cleartk-release</artifactId>
<version>1.4.1</version>
</parent>
<!-- Add cleartk modules, without versions, under your dependencies -->
<dependencies>
<dependency>
<groupId>org.cleartk</groupId>
<artifactId>cleartk-ml</artifactId>
</dependency>
<dependency>
<groupId>org.cleartk</groupId>
<artifactId>cleartk-ml-opennlp-maxent</artifactId>
</dependency>
<!--clearTK -->
```

Other option is to use the lucene.snowball-2.0.0.jar: to make the extraction bag of words features from document.txt

References

[1] https://code.google.com/p/cleartk/downloads/detail?name=cleartk-release-1.4.1-bin.zip&can=2&q=
[2] http://mvnrepository.com/artifact/org.cleartk
[3] http:// make the extraction bag of words features from document.txt make the extraction bag of words features from document.txt
[4] lucene.snowball-2.0.0.jar

2. Compute the cosine similarity between two sentences in the text collection.

OPTION#1: This is a fragment about how to compute the cosine similarity measure, some functions were called that are part of the import Jama.Matrix library like norm1, normF.

```
import Jama.Matrix;


// TODO :: compute the cosine similarity measure


public doublecosineSimilarityMeasure(Matrix inputDocu, Matrix outputDocu){
double dotProduct = inputDocu.arrayTimes(outputDocu).norm1();
double euclidianDist = inputDocu.normF()* outputDocu.normF();
returndotProduct / euclidianDist;
}
```

The next functions were called in the computation of the cosine similarity between two sentences in the text collection: **document.txt**

```
/** One norm
@return maximum column sum.
*/
publicdouble norm1 () {
double f = 0;
for (int j = 0; j < n; j++) {
double s = 0;
for (int i = 0; i < m; i++) {
s += Math.abs(A[i][j]);
}
f = Math.max(f,s);
}
return f;
}
/** Infinity norm
@return maximum row sum.
*/
publicdouble normInf () {
double f = 0;
for (int i = 0; i < m; i++) {
double s = 0;
for (int j = 0; j < n; j++) {
s += Math.abs(A[i][j]);
}
f = Math.max(f,s);
}
```

```
return f;
}
/** Frobenius norm
@return sqrt of sum of squares of all elements.
*/
publicdouble normF () {
double f = 0;
for (int i = 0; i < m; i++) {
for (int j = 0; j < n; j++) {
f = Maths.hypot(f,A[i][j]);
}
}
return f;
}
```

OPTION#2: Cosine Similarity

RetrievalEvaluator.java

```
private double computeCosineSimilarity(
                        Map<String, Integer> queryVector,
                        Map<String, Integer> docVector) {
                double cosine_similarity=0.0;

                // TODO :: compute cosine similarity between two sentences
                Map<String, Integer> minMap, otherMap;
                if(queryVector.size() < docVector.size()) {
                        minMap = queryVector;
                        otherMap = docVector;
                } else {
                        minMap = docVector;
                        otherMap = queryVector;
                }
                String s;
                Iterator itMin = minMap.values().iterator(),
                                itOther = otherMap.values().iterator();
                double normMin = 0, normOther = 0;
                for(Map.Entry<String, Integer> e: minMap.entrySet()) {
                        s = e.getKey();
                        if(otherMap.containsKey(s))
                                cosine_similarity += e.getValue() * otherMap.get(s);
                        normMin += Math.pow((Integer)itMin.next(), 2);
                        normOther += Math.pow((Integer)itOther.next(), 2);
                }
                while(itOther.hasNext())
                        normOther += Math.pow((Integer)itOther.next(), 2);

                return cosine_similarity/(normMin*normOther);
        }
```

Making some test in Maven build in the file pom.xml, the next dependencies were added: EJML library (efficient-java-matrixlibrary) and Jama (A Java Matrix Package) which is better? Which one is more apt for this task?. Finally I choose Jama, for making the experiment about cosine similarity. BUT I added a dependency related to EJML-Efficient Java Matrix Library.

```
<!-- Adding EJML dependency -->
<!-- Efficient-Java-Matrix-Library -->
<dependency>
<groupId>com.googlecode.efficient-java-matrix-library</groupId>
<artifactId>ejml</artifactId>
<version>0.24</version>
</dependency>
<!-- Adding JAMA dependecy -->
<!-- A Java Matrix Package -->
<dependency>
<groupId>gov.nist.math</groupId>
<artifactId>jama</artifactId>
<version>1.0.2</version>
</dependency>
```

References

[1] http://en.wikipedia.org/wiki/Cosine_similarity
[2] http://ant.apache.org/ivy/ivyde/download.cgi
[3] http://ntwrkanlystlkit.sourceforge.net/java2html/Jama/Matrix.java.html
[4] http://math.nist.gov/javanumerics/jama/
[5] https://code.google.com/p/efficient-java-matrix-library/wiki/EjmlManual

DocumentVectorAnnotator.java:a vector of tokens was constructed and updatethe tokenList in CAS. DocumentReader.java

Making the preprocess about the text paragraph: lower, trimming, stopwords and stemming

```
private FSList createTermFreqVector(JCas jcas, Document doc) {

        //replace the whitespaces
        String stripWs = doc.getText().trim().replace("[ ]+", " ");

        //lowercase letters from strings
        String lower = stripWs.toLowerCase();

        //remove from stopWords
        String[] words = lower.split(" ");
        ArrayList<String> wordList = new ArrayList<String>();

        for(String s: words) {
                if(!stopwords.contains(s))
                        wordList.add(s);
        }
        //calculating the stemm
        for(int i=0; i<wordList.size(); i++)
                wordList.set(i, stem(wordList.get(i)));
```

```
		//calculating tokens
		HashMap<String, Integer> tokens = new HashMap<String, Integer>();
		for(String s: wordList )
			if(tokens.containsKey(s))
				tokens.put(s, tokens.get(s) + 1);
			else
				tokens.put(s, 1);

		//transform  to the tokenList
		ArrayList<Token> sal = new ArrayList<Token>();
		Token token = null;
		for(Entry<String, Integer> e: tokens.entrySet()) {
			token = new Token(jcas);
			token.setText(e.getKey());
			token.setFrequency(e.getValue());
			sal.add(token);
		}
		//append
		return Utils.fromCollectionToFSList(jcas, sal);
}

private String stem(String string) {
		return null;
}

public static HashSet<String> loadStopwords() {
		HashSet<String> res = new HashSet<String>();
		try {

			System.out.println(System.getProperty("user.dir"));

			URL url = VectorSpaceRetrieval.class.getResource("/data/stopwords.txt");

			BufferedReader br = new BufferedReader(new
InputStreamReader(url.openStream()));
			String line = null;
			while((line = br.readLine()) != null)
				if(!line.startsWith("#"))
					res.add(line);
			br.close();

		} catch(Exception ex) {
			System.err.println("Issue reading stopwords.txt: " +ex);
			System.exit(1);
			return null;
		}

		return res;
}
```

```
        //Print the text that is in the Document called doc
        private void printDoc(Document doc)

                System.out.println("Document Text: " +doc.getText());
                StringBuffer sb = new StringBuffer();
                doc.getTokenList().prettyPrint(0, 1, sb, true);
                System.out.println(sb.toString());
        }

        //Calculating the stemmer in the string
        private String stemString(String s) {
                englishStemmer stemmer = new englishStemmer();
                stemmer.setCurrent(s);
                stemmer.toString();
                return stemmer.getCurrent();
        }
}
```

Bag of Words.java, were words are located and with and arrayList each sentence of the paragraph is located.

```
public class Bag_Words<Item> implements Iterable<Item> {

        private static final String StdIn = "";
        private int N;           // elements in bag
   private Node<Item> first;    // beginning of bag
        //private static Object stdOut;


   public static class Node<Item> {
      private Item item;
      private Node<Item> next;
   }


    //Empty bag.
   public Bag_Words() {
      first = null;
      N = 0;
   }


    // Check what is in the bag?
    // true if this bag is empty; false otherwise
   public boolean isEmpty() {
      return first == null;
   }

   //The number of items in the bag
   public int size(Object stdOut) {
      return N;
   }
```

```
  // Adds the item to this bag
  public boolean add(String st) {
     Node<Item> oldfirst = first;
     first = new Node<Item>();
     first.item = (Item) st;
     first.next = oldfirst;
     N++;
                        return false;
  }


   // Returns an iterator that iterates over the items in the bag
  public Iterator<Item> iterator()  {
     return new ListIterator<Item>(first);
  }

    // Iterator
     private class ListIterator<Item> implements Iterator<Item> {
     private Node<Item> current;

     public ListIterator(Node<Item> first) {
        current = first;
     }

     public boolean hasNext()  { return current != null;                  }
     public void remove()      { throw new UnsupportedOperationException();  }

     public Item next() {
        if (!hasNext()) throw new NoSuchElementException();
        Item item = current.item;
        current = current.next;
        return item;
     }
  }
```

This class called Pair.java, is applied to make comparisons between two strings:

```
package edu.cmu.lti.f13.hw4.hw4_137175.utils;

import java.util.Arrays;
import java.util.Comparator;

public class Pair<T1 extends Comparable<T1>, T2 extends Comparable<T2>> {
            protected T1 t1;
            protected T2 t2;

            public Pair() {}

            public Pair(T1 t1, T2 t2) {
                        this.t1 = t1;
                        this.t2 = t2;
```

```
}

public T1 getT1() {
        return t1;
}

public void setT1(T1 t1) {
        this.t1 = t1;
}

public T2 getT2() {
        return t2;
}

public void setT2(T2 t2) {
        this.t2 = t2;
}

public  Comparator<Pair<T1, T2>> getComparatorT1() {
        return new Comparator<Pair<T1, T2>>() {
```

Conclusion

There were made some modifications to the initial code where the analysis and classification about strings contained in the paragraph was done to find tokens, remove whiteSpaces, calculate stemm, create a bag of words and know what is located into the bag searching by the position into the array.

The calculation about the features was made making some experiments with ClearTK.jar and with Lucene.snowball.version.jar and with the implementation of the method:

```
// Calculating the stemmer in the string

private String stemString(String s) {
        englishStemmer stemmer = new englishStemmer();
        stemmer.setCurrent(s);
        stemmer.toString();
        return stemmer.getCurrent();
}
```