POS Tagging Toolbox for UTF-8 Text

Bereket Z. Gichamo
Department of Computer Science, Lund University,Lund,Sweden
bereket-z.gichamo.471@student.lu.se

Ehsan Bouhendi
Department of Computer Scinece , Lund University,Lund , Sweden
eh-san.bouhendi.239@student.lu.se

Abstract

This paper describes a Part-of-speech (POS) tagging program which assigns a POS for a word in UTF-8 format based on its frequency in the training set. It is done as a project in a Language Processing and Computational Linguistics course.

1 Introduction

Annotation of words with grammatical categories is an important part of natural language processing (NLP) system. More complex NLP applications such as information extraction, syntactic parsing, machine translation or semantic field annotation often make use of Corpora tagged with POS as prerequisite. Training of statistical models is also done by making use of such Corpora.

This paper is intended to discuss a POS tagging toolbox (program) that is developed to POS tag words whose characters are encoded in UTF-8 format. The program is implemented using Java as a programming language and it has yielded reasonable outputs when tested with the Corpora of the English, Swedish and Persian languages.

2 POS tagging

POS tagging is an automatic annotation of words with grammatical categories also called POS tags [3]. POS tagger marks up the words in a text as corresponding to a particular part of speech, based on both its definition, as well as its context. For example if a sentence,
I will give you the book.
is POS tagged correctly, it is tagged as:
I/pronoun will/modal give/verb you/pronoun the/determiner book/noun.
But automatic and correct POS tagging is not such easy since words can have more than one POS tags in according to the context they are used. For example the word *will* in the above sentence can be used as a noun in another context like in:
I used my will power.
So any POS tagger should have a way to deal with such ambiguities. Different POS tagging methods have been devised with different suggested methods of marking a word with its correct POS tag.

2.1 Baseline POS tagger

A POS tagger that uses the frequency of a word in a training set to mark it with its corresponding part of speech is called Baseline POS tagger. It is named so for the accuracy obtained using this tagger is the minimal one (Baseline figure [3]). Our POS tagging toolbox is a baseline. The baseline figure can be improved using methods based on either rules or statistics.

2.2 Rule based POS tagger

Rule-based POS tagger is a POS tagger that uses symbolic rules designed by hand or derived automatically from hand-annotated corpora. Rules consider the left and right context of the word to disambiguate, that is, either discard or replace a wrong part of speech. The famous POS tagger that uses this method is the Brill's POS tagger (1995) [3].

2.3 Statistical POS tagger

A Statistical POS tagger is a POS tagger that assigns the most likely tags to words in a sentence based on probabilistic models applied on the sequence statistics which is automatically learned from hand-annotated corpora [3].

3 Character encoding

Words whose POS tags we are concerned about are formed from characters and these characters are encoded in different character encodings based on a format of symbols used in a language. Among the various character encodings, UTF-8 is one of the Unicode Transformation formats which uses 8 bit variable-width (which maximizes compatibility to ASCII) [5].

A POS tagger that is designed to handle texts from different languages need to be designed using a programming/scripting language that has adequate facilities for such demands of encoding. In our case, our POS tagger is designed to handle the Persian language, which uses special symbols as alphabets, in addition to the languages that use Latin alphabets like Swedish and English.

Persian/Farsi is the official Language of Iran and Tajikistan and one of the main languages spoken in Afghanistan. It has 32 letters. Although its script is similar to Arabic, Persian language has four more extra letters than Arabic i.e پ (pe), چ (che) , گ (ge) and ژ (zhe). Its grammatical structure is Subject-Object-Verb(SOV) and doesn't have masculine and feminine variations of words or pronouns[2].

Bijankhan's Corpus from University of Tehran is the only manually POS tagged corpus developed for Persian language. The corpus contains almost 2.6 million of words which are manually tagged with 550 different POS tags [4]. The characters in the corpus are encoded using UTF-8 encoding, and compound words like کسب و کار (kasb o kar), which is 'Business' in its meaning, are constructed by putting the single words separated by single space. The normal word to word separation is double space.

4 Implementation

The POS tagging toolbox is made up of the following four different parts which are discussed in detail in the subsections 4.1 - 4.4. They are

- Graphical user interface: to access all of the functionalities given by the toolbox and display the results.
- Table generator: to generate a table to be used by the tagger to tag input.
- Baseline tagger: to tag the input text based on the table generated by Table Generator.
- Evaluator: to evaluate the result by comparing the manually POS tagged words in the test set with the POS tagged file generated by the baseline tagger.

All of these parts are implemented using Java programming language which has good support of different code pages (especially UTF-8), strong data structures and good regular expression class libraries. The strong data structures in Java have made the generated frequency table easier for saving on a memory of a PC for a subsequent faster search of a word and its corresponding POS tag. The good facilities of regular expressions have also simplified the tokenization of words. The whole code is put under Appendix A.

4.1 Graphical User Interface

The GUI application is selectable between table generator and POS Tagger (Base line tagger). Snap shots of the graphical user interface of the tool box are depicted in figures 1 and 2. Figure 1 illustrates the toolbox when the Table generator is selected while figure 2 illustrates the toolbox when the tagger is selected.

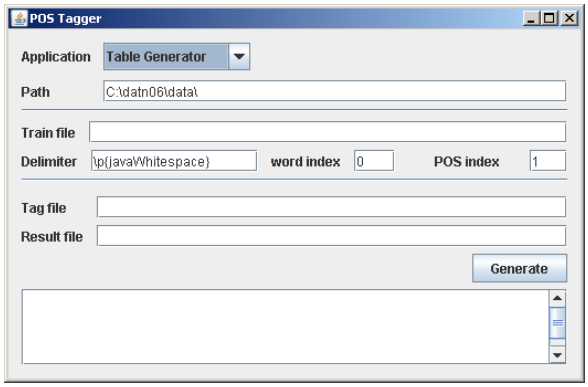

Figure 1: table generator selected

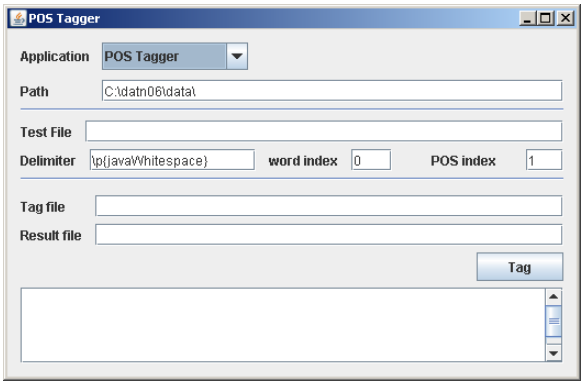

Figure 2: tagger is selected

4.2 Table Generator

The table generator collects all the word-POS combinations and their corresponding frequency of their existence in the training set, and then it assigns for each word a POS tag, eliminates less used POS tag and assigning the maximum used POS tag. It will store the result with a tab delimited text format into a file which can be used later by baseline tagger.

Another feature of this application is ignoring less frequent word by using a threshold value for the frequency of a word to be included in the table. Such low frequent word will be tagged by the POS tag defined for \LFW. \LFW, Low Frequent Words is a predefined word which is added by the table generator at the end of the table to tag words which are not in the training set. \LFW will be tagged by the POS that has maximum frequency in the training set. To select a POS tag for \LFW, some other approaches can be used as well, such as selecting the maximum POS tag which is used for eliminated words, or selecting the maximum used POS tag before removing less used POS tag for each word (contrasted to the first way on which the maximum POS tag was selected after removing less used POS tag).

4.3 Base Line Tagger

Base line tagger loads the table generated in the previous phase. The table file can be used for tagging several files without generating the table again. The tagger gives a POS tag for each word based on the POS tag specified for that word in the table file. The tagger uses \LFW POS tag for words which are not found in the table. To read the training/test set, the tagger uses the delimiter defined by the user. This provides seemless adjustment when reading differently formatted training/test sets.

4.4 Evaluator

Evaluator compares the POS tags given by the tagger against their original POS tags in the training/test set and evaluates the resulting accuracy in percentage. The accuracy of the Baseline tagger is calculated as:

$$accuracy = \frac{N_c}{Total} \times 100$$

Where N_c is the number of correctly tagged words.

5 Result

The following results are found when the toolbox is tested with the English, Persian and Swedish languages. For English and Swedish the training and the test sets are taken from the CONLL web site [1].The sizes of the training and the test sets are given in the Table 1 along with the percentage of accuracy the baseline tagger has yielded.

Language	Size of Training set	Size of Test set	Accuracy in %
English	211727	47377	88.81
Persian	2597937	87414	75.96
Swedish	191467	5656	79.66

Table 1

The most frequent POS tags that are used as \LFW based on the given corpora are NNP, N_SIN and NN for English, Persian and Swedish respectively.

6 Conclusion

Studying different kinds of POS tagger and implementing one of them (i.e. the baseline POS tagger) has a plus towards better understanding of one of the basic fields of natural language processing. The requirements such as making the POS tagger be able to handle UTF-8 encoding have surfaced opportunities to learn more the ways of dealing with real problems and examining the possible solutions that have shaped the overall implementations.

The results found in all of the three languages are encouraging and in line with what is expected from such POS tagger. For further improvements of the resulting figures, using the rule-based or statistical POS tagger is generally recommended. For the Persian language, refining the corpus in a way that reduces the number of tags is also believed to improve the obtained result.

Acknowledgment

Our hearty gratitude goes to Richard Johansson for all his support and valuable pieces of advices throughout the duration of the project.

Reference

Conference on Computational Natural Language Learning. 2008. English and Swedish Corpora, http://www.cnts.ua.ac.be/conll. Date: 08/12/20 at 20:00

John Andrew Boyle. 1966. Grammar of modern Persian. WIESB.

Pierre M. Nugues. 2006. An Introduction to Lan guage Processing with Perl and Prolog. Springer.

Web site of Bijankhan corpus. 2008. Persian Corpus, ,http://ece.ut.ac.ir/DBRG/Bijankhan. Date: 08/12/20 at 20:00

Wikipedia, different authers. 2008. Unicode, http://en.wikipedia.org/wiki/Unicode. Date: 08/12/20 at 20:00

Appendix A. Source Code

MostFrequent.java

```
import java.util.*;
import java.io.*;


public class MostFrequent {
    private static Hashtable<String, Integer> frequences;
    public static Hashtable<String, String> taggs;
    private String delimiter;
    private String trainFile;
    private String taggFile;
    private int wordIndex = 0;
    private int posIndex = 1;
    /**
     * Constructor
     * */
    public MostFrequent(String trainFile, String taggFile,String delimeter)
{
      frequences = new Hashtable<String,Integer>();
        taggs = new Hashtable<String,String>();


      this.trainFile = trainFile;
      this.taggFile = taggFile;
      setDelimiter(delimeter);

    }


      public static void main(String argv[]) throws FileNotFoundException,
IOException{

            MostFrequent      instance      =      new      MostFrequent(
"c:\\datn06\\data\\train.txt",

"c:\\datn06\\data\\en.tbl",

"\\p{javaWhitespace}");

            instance.extractTaggs();
            instance.exportTagTables();

      }
      public void extractTaggs()throws FileNotFoundException, IOException {
            extractFrequences();
            fillTaggs(0);
      }
      public void exportTagTables() throws FileNotFoundException,
                  IOException {
            File outputFile = new File(taggFile);
            OutputStreamWriter outStream =
                  new          OutputStreamWriter(new          FileOutput-
Stream(outputFile),"UTF-8");
            for    (Iterator<String>    it    =    taggs.keySet().iterator();
it.hasNext();){
```

```
                String word = it.next();
                outStream.write(word);
                outStream.write("\t");
                outStream.write(taggs.get(word));
                outStream.write("\n");


            }
            outStream.close();
        }

    private void fillTaggs( int eliminationThreshold ) {

        Hashtable<String, Integer> countedWord = new Hashtable<String,Integer>();
        Hashtable<String, Integer> countedPos = new Hashtable<String,Integer>();

        for (Iterator<String> it = frequences.keySet().iterator(); it.hasNext(); ){
            String currentKey = it.next();
            String word = currentKey.substring(0,currentKey.indexOf("^"));
            String pos = currentKey.substring(currentKey.indexOf("^")+1);

            if (countedWord.get(word) == null){
                countedWord.put(word,1);
                taggs.put(word,pos);
            } else {
                if (countedWord.get(word) < frequences.get(currentKey)){
                    countedWord.put(word,frequences.get(currentKey));
                    taggs.put(word,pos);
                }
            }
            if (countedPos.get(pos) == null){
                  countedPos.put(pos,1);
            } else {
                  countedPos.put(pos,countedPos.get(pos) + 1);
            }
        }
        countedWord = null;

        String targetPos = "";

        if (eliminationThreshold == 0) {


            int maxCountedPos = 0;

            for (Iterator<String> it = countedPos.keySet().iterator(); it.hasNext(); ){
                  String key = it.next();
                  if ( countedPos.get(key) > maxCountedPos ){
                        maxCountedPos = countedPos.get(key);
                        targetPos = key;
                  }
            }


        }
        //System.out.print(countedPos.size());
        countedPos = null;
```

```
        if (eliminationThreshold > 0) {
            Iterator<String> i;

            Hashtable<String,Integer> lfPoses = new Hashtable();

            int maxCount = 0;
            for (i = countedWord.keySet().iterator(); i.hasNext();) {
                String word = i.next();

                if (countedWord.get(word) <= eliminationThreshold){
                    if (lfPoses.get(taggs.get(word)) == null){
                        lfPoses.put(taggs.get(word),                countedWord.get(word));
                    } else {
                        lfPoses.put(taggs.get(word),                  lfPoses.get(taggs.get(word)) + countedWord.get(word));
                    }
                    if (lfPoses.get(taggs.get(word)) > maxCount) {
                        maxCount = lfPoses.get(taggs.get(word));
                        targetPos =  taggs.get(word);
                    }
                    taggs.remove(word);
                }

            } // FOR

        }
        taggs.put("\\LFW", targetPos);

    }

    private void extractFrequences() throws IOException, FileNotFoundException {
      frequences = new Hashtable<String,Integer>();
        taggs = new Hashtable<String,String>();

        File inputFile = new File(trainFile);
        InputStreamReader inputStream =
            new InputStreamReader(new FileInputStream(inputFile),"UTF-8");

        Scanner input = new Scanner(inputStream);

      int l = 0;
        while (input.hasNext()){

            String line = input.nextLine();

            String tokens[] = line.split(delimiter);

                if (tokens.length < 2)
                    continue;

            String word = tokens[wordIndex];

            if (word.matches("(\\d+\\S*)+")){
                  word = "00";
            }

            String pos = tokens[posIndex];
```

```
            if (frequences.get(word+"^"+pos) == null){
                  frequences.put(word+"^"+pos,1);
            }else {
                  frequences.put(word+"^"+pos,frequences.get(word+"^"+pos)
+1);
            }// end if*/
         }
      }

      /**
       * @param delimiter the delimiter to set
       */
      public void setDelimiter(String delimiter) {
            this.delimiter = delimiter;
      }

      /**
       * @return the delimiter
       */
      public String getDelimiter() {
            return delimiter;
      }


      /**
       * @param wordIndex the wordIndex to set
       */
      public void setWordIndex(int wordIndex) {
            this.wordIndex = wordIndex;
      }


      /**
       * @return the wordIndex
       */
      public int getWordIndex() {
            return wordIndex;
      }


      /**
       * @param posIndex the posIndex to set
       */
      public void setPosIndex(int posIndex) {
            this.posIndex = posIndex;
      }

      /**
       * @return the posIndex
       */
      public int getPosIndex() {
            return posIndex;
      }

}
```

BaseLineTagger.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.lang.*;
import java.util.Hashtable;
import java.util.Scanner;

public class ArffMaker {
      private static Hashtable<String, String> taggs;
      public static  void main (String argv[]) throws FileNotFoundExcep-
tion, IOException{

            //readTaggsTable();
            makeHeader();



      }
      private static void makeHeader() throws FileNotFoundException, IOEx-
ception {
            File                  inputFile                  =                  new
File("c:\\datn06\\data","persian_train.txt");
        InputStreamReader inputStream = new InputStreamReader(new FileIn-
putStream(inputFile),"UTF-8");
        Scanner input = new Scanner(inputStream);

      File outputFile = new File("c:\\datn06\\data","persian_header.csv");
            OutputStreamWriter outStream = new OutputStreamWriter(new Fi-
leOutputStream(outputFile),"UTF-8");

            outStream.write("Word,POS\n");

            String wPrev = "BOS";
            //input.useDelimiter("((\\s\\s+)|\n)");
                String delimiter = "((\\s\\s+)|\n)";
                int counter = 0;
            while (input.hasNext() && counter <= 100000) {
                        counter++;

                        String line = input.nextLine();
                        String tokens[] = line.split(delimiter);
                        if (tokens.length < 2)
                           continue;
                        //String wCurr = input.next();
                        String wCurr = tokens[0];
                  /*if (taggs.get(wCurr) == null){
                        wCurr = "\\LFW";

                  }
                  */

                  String currPos = tokens[1];
                  //input.next();
                  if (wCurr.contains("\""))
                        continue;
```

```
                    //outStream.write("\"" + wPrev + "\"");
                    //outStream.write(",");
                    outStream.write("\"" + wCurr +  "\"");
                    outStream.write(",");
                    outStream.write("\"" + currPos +  "\"");
                    outStream.write("\n");

                    //if (wCurr.matches("[#.]"))
                    //    wCurr = "BOS";
                    //wPrev = wCurr;


            }
            System.out.println("DONE.");
            outStream.close();
      }
      private static void readTaggsTable() throws FileNotFoundException {
            File inputFile = new File("c:\\datn06\\data","POS_taggs.txt");
        InputStreamReader inputStream = new InputStreamReader(new FileIn-
putStream(inputFile));
        Scanner input = new Scanner(inputStream);

        taggs = new Hashtable <String,String>();
        while (input.hasNext()){
            String word = input.next();
            String pos = input.next();
            taggs.put(word,pos);

        }
      }

}

import java.io.*;
import java.util.Scanner;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.UnsupportedEncodingException;
```

Eval_PosTag. java

```
public class Eval_PosTag {
      private int corrects = 0;
      private int incorrects = 0;
      private int total = 0;
      private String delimiter;
      private int wordIndex = 0;
      private int posIndex = 1;

      public void evaluate(String refPath, String resPath) throws Unsuppor-
tedEncodingException, FileNotFoundException{


            File refFile = new File(refPath);
            File resFile = new File(resPath);
            InputStreamReader refStream = new InputStreamReader(new FileIn-
putStream(refFile),"UTF-8");
        InputStreamReader resStream = new InputStreamReader(new FileInput-
Stream(resFile),"UTF-8");
```

```
        Scanner refInput = new Scanner(refStream);
        Scanner resInput = new Scanner(resStream);
        refInput.useDelimiter(delimiter);
        resInput.useDelimiter("(\t|\n)");

        while (refInput.hasNext() && resInput.hasNext())
        {
            String line = refInput.nextLine();
            String tokens[] = line.split(delimiter);
            if (tokens.length < 2)
                    continue;

            String refWord = tokens[wordIndex];
            String refPos  = tokens[posIndex];

            String word    = resInput.next();
            String pos     = resInput.next();



            if(refWord.compareTo(word)==0 && refPos.compareTo(pos)== 0)
            {
                  corrects++;
            }
            else
            {


                  incorrects++;
            }
            total++;

        }




      }
      public static void main(String [] args) throws FileNotFoundException,
UnsupportedEncodingException
      {
            Eval_PosTag instance = new Eval_PosTag();
                instance.setDelimiter("((\\s\\s+)|\n)");
            instance.evaluate("c:\\datn06\\data\\persian_test.txt",
                                    "c:\\datn06\\data\\persian_result.txt");
                System.out.println(instance.getIncorrects());

      }

    public String getDelimiter() {
        return delimiter;
    }

    public void setDelimiter(String delimiter) {
        this.delimiter = delimiter;
    }

    public int getCorrects() {
        return corrects;
    }
```

```
    public void setCorrects(int corrects) {
        this.corrects = corrects;
    }

    public int getIncorrects() {
        return incorrects;
    }

    public void setIncorrects(int incorrects) {
        this.incorrects = incorrects;
    }

    public int getTotal() {
        return total;
    }

    public void setTotal(int total) {
        this.total = total;
    }

    public int getWordIndex() {
        return wordIndex;
    }

    public void setWordIndex(int wordIndex) {
        this.wordIndex = wordIndex;
    }

    public int getPosIndex() {
        return posIndex;
    }

    public void setPosIndex(int posIndex) {
        this.posIndex = posIndex;
    }

}
```