Swahili verb segmenter

Kevin Donnelly[*]

Abstract

These notes explain how to install the Swahili verb segmenter, and give some information on how it operates.

1 Introduction

This verb segmenter allows Swahili verbforms to be segmented for use in parsers or taggers, or for the examination of verbform instances in corpora. Although some segmenters already exist, they either handle only a few basic forms, or are not available under an open license. This segmenter handles all the one-word tenses in Swahili, and is licensed under the GPLv3[1] and the AGPLv3[2]. The segmenter has been built and tested on GNU/Linux, but it is likely to run also on legacy platforms like Microsoft Windows or Apple Mac OS X.

Two variants of the segmenter exist: a browser-based version,[3] and a version that can be run from the command line against a file listing verbforms to be analysed. It would also be possible to use this latter version as part of an application that would tag connected text.

The segmenter is currently relatively slow - working through a list of 140,000 possible verbforms drawn from Kevin Scannell's 5m-word *Crubadán*[4] Swahili corpus, its average analysis rate was around 87 verbforms per minute.

2 Background

The first iteration of this segmenter was a generating conjugator like the one I did for Welsh,[5] where I would set up verbal forms and affixes in a database, along with shaping or affixation rules, and then PHP scripts would stick all these

[*] kevin@dotmon.com
[1] http://www.gnu.org/licenses/gpl.html
[2] http://www.gnu.org/licenses/agpl.html
[3] http://kevindonnelly.org.uk/swahili/segmenter
[4] http://borel.slu.edu/crubadan/
[5] http://eurfa.org.uk

together. This would have been much easier to do for a Bantu language than for an inflected language like Welsh. Even though many of the forms would have been semantically dubious, even if morphologically possible, that would not have mattered, since they could just sit quietly in the database, offending no-one. The main drawback would be that as new verbs were added to the dictionary, the relevant forms would have to be generated, but that could be done by a script.

However, when I took the first steps of generating forms for the current present (**-na-**) tense, adding subject and object pronouns for all the classes, something emerged that should have been obvious from the start. A first run-through yields about 400 forms for **-ambia** (*say,tell*). And it turns out that of these 400 "possible" forms, only 15 (less than 4%) occur in the *Crubadán* Swahili corpus. Once we do a rough calculation, the enormity of the problem becomes apparent: 20 subject prefixes x 20 object prefixes x 25 tenses x 20 relatives x (say) 2,000 verbs (initially) = 400m entries! Keeping millions of forms in a database when they may never be used was clearly impractical.

So the segmenter now adopts an analytical approach, which takes advantage of the fact that Swahili verbs have specific slots for each type of morphological affix. There is some overlap, particularly where negative markers and the general present tense are concerned, but in general the sequence is fixed. The verbform is segmented slot by slot, and the material in each slot is tagged with all possible affix information before moving on to the next.

For instance, in **anapika** (he/she is cooking), the **a-** is a marker for the third person singular of Class 1. However, in **apika** (he/she cooks), it is a combined marker for the class and also the general present tense. When the segmenter meets **a-**, it gives it the tag **[sp1-3s,sp1-3s+gen]**, reflecting both these possibilities. Later, disambiguation rules strike out affix tags that are not applicable. In this case, if no other tense marker is present the first tag is deleted, but if another tense marker like **-na-** is found, the second tag is deleted.

This on-the-fly analysis has the advantage that you don't need to generate the forms beforehand and add them to the dictionary (though of course you won't get a verb lemma until you put an entry for it into the dictionary!). The main disadvantage is that the analyser trusts you to put in correct forms – it will analyse incorrect forms as best it can, which in some cases may give incorrect results. The beginnings of a checker are therefore included, which means that even though the analyser cannot completely rule out incorrect forms, but it will flag the most obvious. However, this feature needs more development so that a greater number of incorrect forms are dealt with.

Verbal extensions (where -**pika** (cook) can produce -**pikwa**, -**pikia**, -**pikiwa**, -**pikisha**, -**pikika**, and so on) are not handled directly in the analyser. The main reasons for this are:

- they are less productive (of the 8 or so main extensions, many verbs may have only a few in common use);
- the morphology is more variable (often depending on the structure of the

verb root), which makes analysis more complicated.

Instead, they are handled by giving them separate entries in the dictionary, with the extensions tagged there. All the analyser does is replicate these in its analysis. So **zilipelekewa** (they were sent to) will be analysed as:

zi[sp10]li[past]PELEK_prep_pass(be sent to)[_unmarked]

with the prepositional and passive extensions marked in the analysis.

The main drawback here is that the extended verbs need to be added to the dictionary, but on the other hand in many cases the meaning of the extended verb is some distance from that of the root verb anyway, and giving this meaning in the analysis is probably more helpful than simply marking the extensions on-the-fly. An example would be: -**elea** (*be clear*), -**elewa** (*know*), -**elewana** (*reach agreement*), -**elekea** (*face*), -**elekeza** (*direct*).

3 Preparation

Your machine will need to have Apache2, PHP5, PostgreSQL, and (optionally) Git already installed. The Appendix gives a summary of the relevant commands for Ubuntu 9.10, and suggests how to configure your machine to use these applications effectively – note that this is only a suggestion, and other approaches are possible.

By the end of this process, you should have an instance of the segmenter running on your local webserver, for interactive access, and the command-line version will be available for work on files containing verbforms.

4 Usage

To use the browser version, go to your local installation as detailed in Section F of the Appendix, and type in a verb you want to segment.

To use the command-line version, go to the install directory (**/srv/www/segmenter/public_html** if you have followed the steps in the Appendix), open a terminal, and enter:

```
php cli\_segment.php
```

By default, this will parse a small selection of tenses in the **test/verbs.txt** file, depositing the output in **test/parsed_verbs.txt**, as well as echoing it to screen. To segment your own list of verbs, adjust the **$input** line in **cli_segment.php** with the location of that list, or replace the contents of **test/verbs.txt** with your own data.

5 Method

The segmenter uses the data from Beata Wójtowicz's FreeDict Swahili dictionary, so it will not specify the verb unless it is one of the 500-odd which that dictionary already contains. As the dictionary is expanded, the segmenter will recognise more verbs.

The approach used here takes advantage of the fact that Swahili verbs have specific slots for each type of morphological affix. There is some overlap, particularly where negative markers and the general present tense are concerned, but in general the sequence is fixed.

The segmentation and tagging is done in **segment.php** for the web version, and **cli_segment.php** for the command-line version. The only real differences between these two files is that the former allows for more complex handling of the output (**display.php**), and the latter refers to a verbform list instead of taking input from a web page.

The verbform is segmented sequentially slot by slot, using the functions **cutter_end()** for suffixes and **cutter()** for prefixes – these and other functions are in the file **includes/fns.php**. The material in each slot is then tagged with morphological information. Table 1 shows the order in which the segmentation takes place, and gives some examples of the affixes that occur in that slot, with the tags the segmenter assigns to them. The relevant function used is also listed. Tables 2 and 3 list the tags used. The tags assigned can be changed by adjusting the text in the relevant function – for instance, if you want to change the tag for **zi-** (sp10), you can adjust this line in the **prefixes()** function:

```
$text=preg_replace("/^(zi)/u", "$1[sp10]::", $text);
```

to read:

```
$text=preg_replace("/^(zi)/u", "$1[SUBJ_Class10]::", $text);
```

or whatever you wish. This would change the tag for **zi-** from *sp10* to *SUBJ_Class10*. Bear in mind that the disambiguation and correction rules (see below) may also need to be adjusted accordingly.

After each such segmentation, the remainder of the verbform is checked against the dictionary in the **vbdict** table, using the **lookup()** function, to see if it can be interpreted as a verb stem, and whether or not it seems to have a negative (**-i**) or subjunctive (**-e**) verb ending. After each stage, the affix information and the verb information (if any) are written to the **parsed** table.

Once an entry has been found in the dictionary, no further segmentation is done, and the information is read out of the **parsed** table to give the final parsed form.

This form is then examined in order to see whether some tags are inconsistent with others and can therefore be struck out. This is done in **disambiguate.php** for the web version and **disambiguate.php** for the command-line version, although the only real difference between the two is that the former prints out more feedback. The disambiguation is similar to constraint grammar,[6] but work-

[6]http://beta.visl.sdu.dk/constraint_grammar.html

Affixes	Examples	Equivalent tags	Function
suffixes	*-ye, -zo, -pi, -je*	endrel1, endrel10, int-where, int-how	suffixes()
negative *ha-*	*ha-*	initneg	ineg()
subject pronouns, negatives, infinitive, habitual	*ni-, si-, ku-, hu-*	sp1-1s, neg+sp1-1s, infin15, hab	prefixes()
tense and mood markers and negative particles	*-li-, -sipo-, -ki-*	past, part-neg, cons	tenses()
relatives	*-ye-, -zo-*	rel1, rel10	relatives()
objects	*-tu-, -zi-, -ji-*	op2-1p, op10, refl	objects()

Table 1: *Segmentation slots*

Tenses, moods, aspects		Verbal extensions	
comp	completive **-sha-, -isha-, -kwisha-**	*assoc*	associative extension **-an-**
conc	concessional **-nga-**	*caus*	causative extension **-iz-, -ez-, -ish-, -esh-, -y-**
cond	conditional **-nge-**	*cont*	continuative extension **-t-**
cons	consecutive **-ka-**	*conv*	conversive extension **-u-, -o-**
curr	current present **-na-**	*inc*	inceptive extension **-p-**
fut	future **-ta-**	*pass*	passive extension **-w-**
hab	habitual **hu-**	*pos*	positional extension **-m-**
hypo	hypothetical **-japo-**	*prep*	prepositional extension **-i-, -e-**
imp	imperative **-e**	*stat*	stative extension **-ik-, -ek-**
gen	general present **-a-**		
part	participial **-ki-**		
past	past **-li-**		
perf	perfective **-me-**		
subj	subjunctive **-e**		
supp	suppositional **-ngali-**		

Table 2: *Tense and extension tags*

Pronouns		**Miscellaneous**	
sp	subject pronoun - numbers refer to the class	*_unmarked*	unmarked (default) vowel ending
op	object pronoun - numbers refer to the class	*endrel*	end relative
1s	first person singular	*initneg*	initial negative
2s	second person singular	*int-how*	interrogative how **-je?**
3s	third person singular	*int-what*	interrogative what **-ni?**
1p	first person plural	*int-where*	interrogative where **-pi?**
2p	second person plural	*ms*	monosyllabic prop
3p	third person plural	*neg*	negative
refl	reflexive	*pl*	plural
		redup	reduplicated
		rel	relative

Table 3: *Other tags*

ing within the word boundary rather than across it. There are in fact arguments for using CG for this purpose, but on balance I decided that adding an additional dependency to the segmenter was not justified at present - perhaps in the future!

The existing disambiguation rules need to be extended, since there are a few areas where mutually exclusive tags are not resolved.

A related area is error-checking. As noted earlier, initial work has been done on this, with suggestions to the user as to how to correct a few obvious errors, but this needs to be greatly extended. One of the problems here is where to draw the line – there are in fact an infinite number of errors that can be made, and since the segmenter works by *ad hoc* analysis, it has no list of definitively correct forms. Progressive refinement of the error-checking is therefore a long-term project.

6 Future development

It would be interesting to port this segmenter to other Bantu languages like Shona or Zulu. Even though tone is not marked in the orthography of many Bantu languages, the segmenter should ideally deal with tonal distinctions as well.

Appendix: Configuring Ubuntu 9.10

These instructions should also work on Ubuntu 10.04. In either case, they assume a properly-working desktop with network access.

A Install relevant software

Install Apache (webserver), PHP5 (scripting system), and PostgreSQL (database), phpPgAdmin (browser interface to PostgreSQL), and (optionally) Git (versioning system) and pgAdminIII (desktop interface to PostgreSQL):

```
sudo apt-get install git-core, apache2, apache2-utils, libapache2-mod-php5,
php5, php-pear, php5-xcache, php5-pgsql, postgresql, phppgadmin, pgadmin3
```

B Configure Apache

B.1 Configure a virtual host

```
sudo nano /etc/apache2/sites-available/segmenter
```

Place the following in the file:

```
<VirtualHost *:80>
ServerName segmenter
DocumentRoot /srv/www/segmenter/public_html/
</VirtualHost>
```

B.2 Tell the PC about the new virtual host

```
sudo nano /etc/hosts
```

Add the following line:

```
127.0.0.1 segmenter
```

B.3 Enable the site and restart Apache

```
sudo a2ensite segmenter
sudo /etc/init.d/apache2 restart
```

B.4 Give your normal user access to the `/srv` directory

```
sudo chown -R myuser.myuser /srv
```

B.5 Create a directory structure for the virtual host you set up earlier

```
mkdir -p /srv/www/segmenter/public_html
```

B.6 Create a front-page for the virtual host

```
cd /srv/www/segmenter/public_html
```

```
nano index.html
```

Enter the following:

```
<html>
<head>
<title>Segmenter</title>
</head>
<body>
Front page for segmenter virtual host.
</body>
</html>
```

If you now enter **segmenter** in the address bar of your browser you should see a page reading *Front page for segmenter virtual host*.

C Configure PostgreSQL

C.1 Set PostgreSQL to use passwords

```
sudo nano /etc/postgresql/8.4/main/pg_hba.conf
```

Change the line:

```
local all all ident
```

to:

```
local all all md5
```

C.2 Create a database user

```
sudo -i
```

```
su - postgresql
```

```
createuser -P mypguser
```

Enter a password (twice), and enter **y** at the superuser question.

Enter **exit** twice to return to your normal (desktop) user.

You will have to use the username/password you have just entered to replace the default *kevin/kevindbs* near the top of the `.php` scripts in the download.

D Download the segmenter

In a working directory of your choice, run:
`git clone http://thinkopen.co.uk/git/segmenter`
The files will be downloaded into a *segmenter* folder in your working directory.

If you have chosen not to install Git, you can instead download the files by going to **http://thinkopen.co.uk/git/**, clicking on **segmenter**, then on **tree**, and finally on **snapshot**. This will download a tarball containing the files. Uncompress this to create a `segmenter` folder in your working directory.

Copy all the files in the `segmenter` folder to your new web directory at `/srv/www/segmenter/public_html`. Delete the **index.html** file you created earlier at B.6.

E Initialise the database

E.1 Create the database

Go the the `dbs` folder and create a new database:

```
createdb -U mypguser swahili
```

using the PostgreSQL user you created earlier at C.2. Enter your PostgreSQL password when prompted.

E.2 Import the tables

Log in to psql as your user:

```
psql -U mypguser swahili
```

At the `swahili=#` prompt, enter:

```
\i vbdict.sql
```

```
\i parsed.sql
```

Enter `\q` to exit **psql**.

E.3 Configure the database connection

Open a text editor, and replace the username and password in the file **swahili/config.php** with the PostgreSQL username and password you created earlier at C.2.

Move the configuration details out of the web tree:

```
sudo mv siarad /opt
```

F Test access

Open a web browser, and enter **segmenter** on the location line. You should see the front page of the segmenter.

If this does not work, contact me at **kevin@dotmon.com**.