wordaxe Silbentrennung

Henning von Bargen, Juni 2009, Release 0.3.3

Übersicht

Der frühere Name der wordaxe-Bibliothek war "deco-cow" und stand für "decomposition of compound words".

Die Bibliothek besteht aber genau genommen aus drei verschiedenen Programmteilen:

1. eine allgemeine (erweiterbare) Klassenbibliothek zur Unterstützung von Silbentrennung in Python-Programmen

2. ein spezieller Silbentrennungsalgorithmus, basierend auf der Zerlegung von zusammengesetzten Wörtern

3. eine Silbentrennungserweiterung zur ReportLab PDF Bibliothek

Bezugsquelle

Die wordaxe Bibliothek wird auf SourceForge verwaltet (http://deco-cow.sourceforge.net).

Das jeweils aktuelle Release der Software kann aber über die entsprechende SourceForge Download-Seite heruntergeladen werden. Die allerneueste in Entwicklung befindliche Version wird im Sourceforge Subversion-Repository verwaltet.

Lizenz

Die wordaxe Silbentrennungsbibliothek kann wahlweise unter einer der beiden Open-Sourcen Lizenzen "Apache 2.0 License" oder "2-Clauses BSD-License" verwendet werden.

Der genaue Text liegt der Bibliothek bei (Datei license.txt).

Zu den Lizenzen für die pyHnj Bibliothek von Danny Yoo zur HNJ-Silbentrennung, und für die ReportLab PDF Bibliothek siehe die entsprechenden Webseiten.

Die "dictionary files" mit der Endung .dic sind von der OpenOffice Distribution übernommen, sie stehen unter der GNU LGPL Lizenz.

Installation

ReportLab 2.3

wordaxe Release 0.3.3 wurde mit Python 2.6 und ReportLab 2.3 getestet, sollte jedoch problemlos auch mit Python 2.4 oder 2.5 funktionieren, da keine neuen Features von Python 2.5 verwendet werden.

ReportLab 2.3 kann von www.reportlab.org heruntergeladen und installiert werden (hier wird nicht beschrieben, wie das geht).

Die Version wordaxe 0.3.0 funktioniert auch mit dem etwas älteren ReportLab 2.2 oder 2.1.

Hinweise zu noch älteren ReportLab-Versionen:

Möglicherweise funktioniert wordaxe Release 0.3.0 auch mit ReportLab 2.0. Ansonsten kann in diesem Fall auch Release 0.2.2 verwendet werden, was aber bei der Installation etwas schwieriger ist (man musste da noch einige Dateien in der ReportLab-Installation überschreiben und die Installationsanleitung war nicht korrekt).

Für ReportLab 1.19 sei auf Release 0.1.1 verwiesen (nicht empfohlen). Bei einer Umstellung von ReportLab 1.x auf 2.x ist (unabhängig von wordaxe) zu beachten, dass bestehender Code evtl. angepasst werden muss, da als "paragraph encoding" immer UTF8 benutzt werden muss.

Installation wordaxe Schritt für Schritt

1. Das ZIP-Archiv wordaxe-0.3.3.zip von der SourceForge Seite (siehe Bezug) herunterladen.

2. Das ZIP-Archiv wordaxe-0.3.3.zip im Wurzelverzeichnis C:\ entpacken (innerhalb des Archivs liegen alle Dateien in einem Verzeichnis "wordaxe-0.3.3"). Dadurch wird die folgende Verzeichnisstruktur angelegt:

Falls bereits eine ältere Version von wordaxe installiert war und die Datei DEhyph.py erweitert wurde, dann muss von dieser Datei unbedingt eine **Sicherheitskopie** angelegt werden!

```
C:
    wordaxe-0.3.3
        docs
        htdocs
            css
            examples
            icons
            images
        tests
        wordaxe
            dict
            plugins
            rl
```

3a. Auf der Kommandozeile ausführen:

```
cd /d c:\wordaxe-0.3.3
setup.py install
```

3b. Alternativ: Für ReportLab 2.2 oder älter: eine Sicherheitskopie der Datei reportlab\pdfbase\rl_codecs.py von der ReportLab Installation anlegen; anschließend die Datei ersetzen durch die mitgelieferte angepasste Version in c:\wordaxe-0.3.3\wordaxe\rl\rl_codecs.py.

Beachte: Dadurch werden nur zwei Zeilen in der Datei geändert, die das "scheue Minuszeichen" SHY betreffen.

Die neue Bibliothek zum Python-Path hinzufügen, zum Beispiel durch Anlegen einer entsprechenden Datei wordaxe.pth im Verzeichnis c:\python26\lib\site-packages, die nur die folgende Textzeile enthält:

```
c:\wordaxe-0.3.3
```

4. Anschließend sicherstellen, dass "import wordaxe" keine Fehlermeldung erzeugt.

5. ReportLab funktioniert genau so wie vorher; Unterschiede können höchstens auftreten, wenn innerhalb der *eigenen* Programme oder Texte das SHY-Zeichen verwendet wird.

Verwendung

Um die Silbentrennung mit dem DCW-Algorithmus (Zerlegung von zusammengesetzten Wörtern) für deutschsprachige Texte in Aktion zu sehen, kann man zum Beispiel das Skript "test_hyphenation.py" im Unterverzeichnis rl aufrufen. Es erzeugt dann zwei PDF-Dateien, test_hyphenation-plain.pdf und test_hyphenation_-styled.pdf.

Auch dieses Dokument selbst wurde mit automatischer Silbentrennung erzeugt (siehe Skript buildDoku.py).

Um die Silbentrennung in eigenen Programmen zu verwenden (am Beispiel des DCW-Algorithmus für Deutsch), genügt es, wenige sehr einfache Änderungen am vorhandenen Programm vorzunehmen:

Die folgenden Zeilen hinzufügen:

```
from wordaxe import hyphRegistry
from wordaxe.DCWHyphenator import DCWHyphenator
hyphRegistry['DE'] = DCWHyphenator('de',5)
```

Die folgenden Strings suchen und ersetzen:

```
Suchen                            Ersetzen durch
reportlab.platypus.paragraph      wordaxe.rl.paragraph
reportlab.platypus.xpreformatted  wordaxe.rl.xpreformatted
reportlab.lib.styles              wordaxe.rl.styles
```

Die Silbentrennung einschalten. Dazu im verwendeten ParagraphStyle zwei Attribute setzen:

```
stylesheet = getSampleStyleSheet()
myStyle = stylesheet["BodyText"]
myStyle.language = 'DE'
myStyle.hyphenation = True
```

Verwendung eines Hyphenators

Selbstverständlich kann die Silbentrennung auch unabhängig von ReportLab verwendet werden.

Beim Konstruktor muss neben einem Sprachcode vor allem eine minimale Wortlänge angegeben werden. Kürzere Wörter werden gar nicht betrachtet.

```
from wordaxe.DCWHyphenator import DCWHyphenator
hyphenator = DCWHyphenator('de',5)
```

Nun können Wörter (Unicode) getrennt werden. Zurückgeliefert wird entwender None (unbekanntes Wort) oder ein HyphenatedWord, d.h. ein Wort mit Angabe der möglichen Trennstellen und ihrer Qualität.

```
hword = hyphenator.hyphenate(u"Donaudampfschiffahrt")
print "Mögliche Trennstellen", hword.hyphenations
# Trenne an der 2. möglichen Trennstelle:
left,right = hword.split(hword.hyphenations[1])
# liefert: (u'Donau\xad', HyphenatedWord(u'dampfschiffahrt'))
# Der linke Teil ist wieder ein Unicode-Objekt (hier: Donau-),
# der rechte Teil ist das übriggebliebene HyphenatedWord, das
# in die nächste Zeile kommen soll.
print left
print right.hyphenations
```

Klassen für die Silbentrennung

Zur Verwendung der Klassen siehe auch die jeweiligen Quelltexte, die jeweils Testcode mit Aufruf des Konstruktors enthalten. Der Testcode kann aufgerufen werden, um zu sehen, wie die jeweilige Klasse mit Worten umgeht. Die Worte können auf der Kommandozeile eingegeben werden. Es bietet sich an, dabei außerdem die Option -v anzugeben.

Beispiel:

```
c:\python26\python wordaxe\DCWHyphenator.py -v Silbentrennung
```

DCWHyphenator

Diese Klasse basiert auf der Zerlegung von zusammengesetzten Wörtern, inspiriert durch die Publikationen der TU Wien zur "sicheren sinnentsprechenden Silbentrennung für die deutsche Sprache", siehe http://www.-ads.tuwien.ac.at/research/SiSiSi/.

Die Implementierung hat jedoch nichts zu tun mit dem Closed Source Produkt "SiSiSi".

Der Algorithmus funktioniert wie folgt: Ein gegegebenes zusammengesetztes Wort wird zunächst in die Einzelwörter zerlegt.

Dazu wird die Datei DE_hyph.ini verwendet. Sie enthält Wortstämme, teilweise versehen mit zusätzlichen Annotationen wie NEED_SUFFIX, NO_SUFFIX etc. Außerdem sind dort mögliche Vorsilben und Suffixe hinterlegt.

Aufgrund der Komplexität des Zerlegungsverfahrens ist es vergleichsweise langsam:

Das Wort wird von links nach rechts betrachtet. Es wird zerlegt in ein Tupel (L,R), wobei natürlich verschiedene Aufteilungen möglich sind, z.B. bei "Trennung" ("T", "rennung"), ("Tr", "ennung"), ("Tre", "nnung"), usw. Immer wird geprüft, ob der linke Teil zu einer der bekannten Vorsilben, Wortstämme bzw. Endungen aus der Datei DE_hyph.ini passt. Falls ja, wird mit dem Restwort analog weitergemacht. Andern-falls, oder wenn die Kombination keinen Sinn ergibt (z.B. Vorsilbe + Suffix ohne Wortstamm dazwischen) wird abgebrochen.

Im Prinzip handelt es sich dabei um ein rekursives Verfahren, was aber im Programm nicht rekursiv, sondern mit Hilfe von Listen umgesetzt wurde.

Die besonderen Eigenschaften dieses Verfahrens sind:

Es werden nur dem Programm bekannte Wortstämme erkannt.

Dadurch werden mögliche Trennstellen evtl. übersehen, denn unbekannte Wörter werden grundsätzlich nicht getrennt.

Andererseits kann es so auch nicht zu falschen Trennungen kommen.

Wenn ein zusammengesetztes Wort auf mehr als eine Art interpretiert werden kann, werden nur die Trennstellen berücksichtigt, die bei allen Zerlegungen gleich sind. Dadurch kann es nicht zu sinnentstellenden Trennungen kommen.

Trennstellen werden mit einer Priorität versehen, die dann vom aufrufenden Programm berücksichtigt werden kann, um gute Trennstellen (an Wortgrenzen) zu bevorzugen.

Diese Klasse unterstützt auch alle Features von ExplicitHyphenator.

ExplicitHyphenator

Diese Klasse unterstützt alle Features von BaseHyphenator, plus:

Bei dieser Klasse kann man für jedes Wort einzeln vorgeben, wie es getrennt werden soll. Damit eignet sich diese Klasse allenfalls für bestimmte Anwendungsbereiche, bei denen der Wortschatz so klein ist, dass die Trennung für alle anfallenden Worte (zumindest die langen) vorab festgelegt werden kann (z.B. wenn praktisch nur feste Textbausteine verwendet werden).

Mit den Methoden add_entry, add_entries sowie add_entries_from_file können solche Trennungen genau vorgegeben werden (siehe Testskript special_words.py).

PyHnjHyphenator

Diese Klasse funktioniert so wie die Silbentrennung in TeX, basierend auf Mustern (vergleiche libhnj, pyhnj). Die Implementierung kann die pyhnj C-Bibliothek verwenden (das ist die Voreinstellung) oder aber Pure Python, wenn das Argument purePython=True beim Konstruktor angegeben wird.

wordaxe.plugins.PyHyphenHyphenator

Diese Klasse funktioniert ebenfalls so ähnlich wie in TeX, verwendet jedoch eine andere Implementierung und *funktioniert wesentlich besser*! Um diese Klasse verwenden zu können, muss die pyhyphen-Bibliothek installiert sein (siehe http://pypi.python.org/pypi/PyHyphen/).

Diese Klasse unterstützt auch alle Features von ExplicitHyphenator.

BaseHyphenator

Diese Klasse funktioniert mit jeder Sprache. Getrennt wird nur nach den folgenden Zeichen:

```
'-'   Minuszeichen (45, '\x2D')
'.'   Punkt (46, '\x2E') (aber z.B. nicht bei Punkten zwischen Ziffern)
'_'   Unterstrich (95, '\x5F')
'-'   SHY hyphenation character (173, '\xAD')
```

Wenn beim Konstruktor das Argument CamelCase=True angegeben wird, dann werden auch CamelCase-Wörter getrennt.

Anmerkungen

Performance

Der DCWHyphenator benötigt aufgrund des verwendeten (im Prinzip rekursiven) Algorithmus relativ lange für die Trennung eines Wortes.

Da die Wortlänge in der Praxis begrenzt ist, ergibt sich beim Einsatz mit ReportLab eine Laufzeit proportional zur Anzahl der (Zeilen minus Absätze), denn es wird jeweils das letzte Wort einer Zeile geprüft (außer bei der letzten Zeile im Absatz).

Speziell für den DCWHyphenator bietet es sich an, diesen nicht direkt zu verwenden, sondern die Ergebnisse wie folgt zu cachen:

```
import wordaxe
from wordaxe.DCWHyphenator import DCWHyphenator
hyph = DCWHyphenator("DE")
wordaxe.hyphRegistry ["DE"] = wordaxe.Cached(hyph, 1000)
```

Erweiterungsmöglichkeiten

Andere Silbentrennungsbibliotheken, z.B. von Duden, können mit Hilfe von ctypes oder mit SWIG leicht eingebaut werden.

Dazu muss die Member-Funktion "hyphenate" überschrieben werden, die ein Unicode-Wort als Eingabe erhält und eine HyphenatedWord-Instanz zurückliefern muss.

Hinweise zu ReportLab

ReportLab macht uns das Leben nicht leicht, da platypus.paragraph.py nicht gerade mit Rücksicht auf spätere Erweiterungen geschrieben wurde.

Alles wäre *viel* einfacher, wenn anstelle der vielen internen Routinen (z.B. "_leftDrawParaLine" etc.) Methoden in der Paragraph-Klasse verwendet würden. Eine Erweiterung der Funktionalität könnte dann im Grunde durch einfaches Schreiben einer abgeleiteten Klasse erfolgen, bei der im Wesentlichen nur "breakLines" überschrieben werden müsste.

Das zweite Problem bei ReportLab 2.x ist die nicht durchgängige Verwendung von Unicode. Stattdessen wird mal mit Unicode, mal mit UTF8 und mal mit noch anderen Byte-Codierungen gearbeitet.

Aus diesem Grunde wird nun eine eigene Paragraph Implementierung genutzt, bei der gegenüber ReportLab große Teile des Codes von Grund auf neu geschrieben wurden.