The Porter Stemmer

Daniel Waegel
CISC889 / Fall 2011

General Stemming Overview

- Conflates inflected/derived words to a stem (root)
- Intuitive
- Stem is not (necessarily) the morphological root
- Abate, abated, abatement, abatements, abates might all stem to “abat”
- Other stemmers might produce different stems
- Crude, imperfect by nature
- Ambiguity about correctness

Applications for stemmers

- Information retrieval (Search engines)
  - Stem both document indexes and queries
  - Often can increase recall without decreasing precision
  - Any situation where one is interested in grouping words into semantically similar sets.
- Other tasks?

Stemming approaches

- Many different approaches:
  - Brute force look up
  - Suffix, affix stripping
  - Part-of-speech recognition
  - Statistical algorithms (n-grams, HMM)
- Porter stemmer utilizes suffix stripping
- It does not address prefixes

Porter Stemmer Overview

- Algorithm dates from 1980
- Still the default "go-to" stemmer
- Excellent trade-off between speed, readability, and accuracy
- Stems using a set of rules, or transformations, applied in a succession of steps
- About 60 rules in 6 steps
- No recursion

Porter Stemmer Steps

- Step 1: Gets rid of plurals and -ed or -ing suffixes
- Step 2: Turns terminal y to i when there is another vowel in the stem
- Step 3: Maps double suffixes to single ones: -ization, -ational, etc.
- Step 4: Deals with suffixes, -full, -ness etc.
- Step 5: Takes off -ant, -ence, etc.
- Step 6: Removes a final -e

Porter stemmer helpers

- m()

  Returns the number of "consonant sequences" in the current stem:
  <c><v> gives 0 (cry, cry-ing)
  <c>vc<v> gives 1 (care, car-ing, scare, scar-ing)
  <c>vcvc<v> gives 2 (probab-ility)

- r(String str)

  If m-function is > 0 then set the current suffix to "str"

- cvc(int pos)

  `Checks whether the previous 3 characters before pos were consonant, vowel, consonant`

Step 1

Gets rid of plurals and -ed or -ing suffixes

```
if b[k] == 's'
  if ends("sses")
    k -= 2
  else if ends("ies")
    setto("i")
  else if b[k-1] != 's'
    k--
if ends("eed")
  if m() > 0
    k--
else if ends("ed") || ends("ing") &&
vowelinstem()
  k = j
  if ends("at")
    setto("ate")
  else if ends("bl")
    setto("ble")
  else if ends("iz")
    setto("ize")
  else if doublec(k)
     k--
     int ch = b[k]
     if ch=='l' || ch=='s' || ch=='z'
        k++
  else if (m() == 1 && cvc(k))
     setto("e")
```

- Possesses → possess
- Ponies → poni
- Operatives → operative
- **Markedly → markedly**
- Interesting → interest
- **Confess → confess**
- Consumables → consumable
- Realizes → realize
- Infuriating → Infuriate
- Fables → fable
- Fated → fate

Step 2

Turns terminal y to i when there is another vowel in the stem

```
if ends("y") && vowelinstem()
   b[k] = 'i'
```

- Coolly → coolli
- Furry → furri
- Fry → fry
- Grey → grei
- **Interestingly → Interestingli**

Step 3

Maps double suffixes to single ones

```
switch b[k-1]
  case 'a':
    if ends("ational") -> r("ate")
    if ends("tional") -> r("tion")
  case 'c':
    if ends("enci")) -> r("ence")
    if ends("anci")) -> r("ance")
  case 'e':
    if ends("izer") -> r("ize")
  case 'l':
    if ends("bli") -> r("ble")
    if ends("alli") -> r("al")
    if ends("entli") -> r("ent")
    if ends("eli") -> r("e")
    if ends("ousli") -> r("ous")
  case 'o':
    if ends("ization") -> r("ize")
    if ends("ation") -> r("ate")
    if ends("ator") -> r("ate")
  case 's':
    if ends("alism") -> r("al")
    if ends("iveness") -> r("ive")
    if ends("fulness") -> r("ful")
    if ends("ousness") -> r("ous")
  case 't':
    if ends("aliti") -> r("al")
    if ends("iviti") -> r("ive")
    if ends("biliti") -> r("ble")
   case 'g':
    if ends("logi") -> r("log")
```

- **Rational → rational**
- Optional → option
- Operational → operate
- Possibly→ possibli → possible
- Really → realli → realli
- Realization → realize
- Feudalism → feudal
- Playfulness → playful
- **Liveness → liveness**

Step 4

Deals with suffixes, -full, -ness etc.

```
switch b[k]
  case 'e':
    if ends("icate") -> r("ic")
    if ends("ative") -> r("")
    if ends("alize") -> r("al")
  case 'i':
    if ends("iciti") -> r("ic")
  case 'l':
    if ends("ical") -> r("ic")
    if ends("ful") -> r("")
  case 's':
    if ends("ness") -> r("")
```

- Authenticate → authentic
- Predicate → predic
- **Realize → realize**
- Felicity → feliciti → felic
- Practical → practic
- Playful → play
- **Gleeful → gleeful**
- Largeness → large

Step 5

Takes off -ant, -ence, etc.

```
switch b[k-1]
 case 'a':
   if ends("al") -> break
 case 'c':
   if ends("ance") -> break
   if ends("ence") -> break
 case 'e':
   if ends("er") -> break
 case 'i':
   if ends("ic") -> break
 case 'l':
   if ends("able") -> break
   if ends("ible") -> break
 … more rules here...
 default: return
if m() > 1
  k = j;
```

- Precedent → preced
- Operational → operate → oper
-
- Interestingly
- Infuriating
- Fable → fable
- Parable →parable
- Controllable → controll

Step 6

Removes a final -e

```
if b[k] == 'e'
  int a = m()
  if a > 1 || a == 1 && !cvc(k-1)
    k--
if b[k]=='l' && doublec(k) && m() > 1
  k--
```

- Parable → parabl
- Fate → fate (cvc)
- Deflate → deflat
- Bee → bee
- Controllable → controll → control
- Petrol → petrol
- Stall → stall
- Resell → resel

Walkthrough

- Let's see what happens step-by-step with examples: Semantically, recognizing, destructiveness

Step 1

Gets rid of plurals and -ed or -ing suffixes

```
if b[k] == 's'
  if ends("sses")
    k -= 2
  else if ends("ies")
    setto("i")
  else if b[k-1] != 's'
    k--
if ends("eed")
  if m() > 0
    k--
else if ends("ed") || ends("ing") &&
vowelinstem()
  k = j
  if ends("at")
    setto("ate")
  else if ends("bl")
    setto("ble")
  else if ends("iz")
    setto("ize")
  else if doublec(k)
     k--
     int ch = b[k]
     if ch=='l' || ch=='s' || ch=='z'
        k++
  else if (m() == 1 && cvc(k))
     setto("e")
```

- Semantically → ?
- Destructiveness → ?
- Recognizing → ?

Step 1

Gets rid of plurals and -ed or -ing suffixes

```
if b[k] == 's'
  if ends("sses")
    k -= 2
  else if ends("ies")
    setto("i")
  else if b[k-1] != 's'
    k--
if ends("eed")
  if m() > 0
    k--
else if ends("ed") || ends("ing") &&
vowelinstem()
  k = j
  if ends("at")
    setto("ate")
  else if ends("bl")
    setto("ble")
  else if ends("iz")
    setto("ize")
  else if doublec(k)
     k--
     int ch = b[k]
     if ch=='l' || ch=='s' || ch=='z'
        k++
  else if (m() == 1 && cvc(k))
     setto("e")
```

- Semantically → semantically
- Destructiveness → destructiveness
- Recognizing → recognize

Step 2

Turns terminal y to i when there is another vowel in the stem

```
if ends("y") && vowelinstem()
   b[k] = 'i'
```

- Semantically →
  semantically → ?
- Destructiveness →
  destructiveness → ?
- Recognizing →
  recognize → ?

Step 2

Turns terminal y to i when there is another vowel in the stem

```
if ends("y") && vowelinstem()
   b[k] = 'i'
```

- Semantically → semantically → semanticalli
- Destructiveness → destructiveness → destructiveness
- Recognizing → recognize → recognize

Step 3

Maps double suffixes to single ones

```
switch b[k-1]
  case 'a':
    if ends("ational") -> r("ate")
    if ends("tional") -> r("tion")
  case 'c':
    if ends("enci")) -> r("ence")
    if ends("anci")) -> r("ance")
  case 'e':
    if ends("izer") -> r("ize")
  case 'l':
    if ends("bli") -> r("ble")
    if ends("alli") -> r("al")
    if ends("entli") -> r("ent")
    if ends("eli") -> r("e")
    if ends("ousli") -> r("ous")
  case 'o':
    if ends("ization") -> r("ize")
    if ends("ation") -> r("ate")
    if ends("ator") -> r("ate")
  case 's':
    if ends("alism") -> r("al")
    if ends("iveness") -> r("ive")
    if ends("fulness") -> r("ful")
    if ends("ousness") -> r("ous")
  case 't':
    if ends("aliti") -> r("al")
    if ends("iviti") -> r("ive")
    if ends("biliti") -> r("ble")
   case 'g':
    if ends("logi") -> r("log")
```

- Semantically →
  semantically →
  semanticalli → ?

- Destructiveness →
  destructiveness →
  destructiveness → ?

- Recognizing →
  recognize →
  recognize → ?

Step 3

Maps double suffixes to single ones

```
switch b[k-1]
  case 'a':
    if ends("ational") -> r("ate")
    if ends("tional") -> r("tion")
  case 'c':
    if ends("enci")) -> r("ence")
    if ends("anci")) -> r("ance")
  case 'e':
    if ends("izer") -> r("ize")
  case 'l':
    if ends("bli") -> r("ble")
    if ends("alli") -> r("al")
    if ends("entli") -> r("ent")
    if ends("eli") -> r("e")
    if ends("ousli") -> r("ous")
  case 'o':
    if ends("ization") -> r("ize")
    if ends("ation") -> r("ate")
    if ends("ator") -> r("ate")
  case 's':
    if ends("alism") -> r("al")
    if ends("iveness") -> r("ive")
    if ends("fulness") -> r("ful")
    if ends("ousness") -> r("ous")
  case 't':
    if ends("aliti") -> r("al")
    if ends("iviti") -> r("ive")
    if ends("biliti") -> r("ble")
   case 'g':
    if ends("logi") -> r("log")
```

- Semantically →
  semantically →
  semanticalli →
  semantical
- Destructiveness →
  destructiveness →
  destructiveness →
  destructive
- Recognizing →
  recognize →
  recognize →
  recognize

Step 4

Deals with suffixes, -full, -ness etc.

```
switch b[k]
  case 'e':
    if ends("icate") -> r("ic")
    if ends("ative") -> r("")
    if ends("alize") -> r("al")
  case 'i':
    if ends("iciti") -> r("ic")
  case 'l':
    if ends("ical") -> r("ic")
    if ends("ful") -> r("")
  case 's':
    if ends("ness") -> r("")
```

- Semantically →
  semantically →
  semanticalli →
  semantical → ?
- Destructiveness →
  destructiveness →
  destructiveness →
  destructive → ?
- Recognizing →
  recognize →
  recognize →
  recognize → ?

Step 4

Deals with suffixes, -full, -ness etc.

```
switch b[k]
  case 'e':
    if ends("icate") -> r("ic")
    if ends("ative") -> r("")
    if ends("alize") -> r("al")
  case 'i':
    if ends("iciti") -> r("ic")
  case 'l':
    if ends("ical") -> r("ic")
    if ends("ful") -> r("")
  case 's':
    if ends("ness") -> r("")
```

- Semantically → semantically → semanticalli → semantical → semantic
- Destructiveness → destructiveness → destructiveness → destructive → destructive
- Recognizing → recognize → recognize → recognize → recognize

Step 5

Takes off -ant, -ence, etc.

```
switch b[k-1]
 case 'a':
   if ends("al") -> break
 case 'c':
   if ends("ance") -> break
   if ends("ence") -> break
 case 'e':
   if ends("er") -> break
 case 'i':
   if ends("ic") -> break
 case 'l':
   if ends("able") -> break
   if ends("ible") -> break
 case 'v':
   if ends("ive") -> break
 … more rules here...
 default: return
if m() > 1
  k = j;
```

- Semantically →
  - semantically →
  - semanticalli →
  - semantical →
  - semantic → ?
- Destructiveness →
  - destructiveness →
  - destructiveness →
  - destructive →
  - destructive → ?
- Recognizing →
  - recognize →
  - recognize →
  - recognize →
  - recognize → ?

Step 5

Takes off -ant, -ence, etc.

```
switch b[k-1]
 case 'a':
   if ends("al") -> break
 case 'c':
   if ends("ance") -> break
   if ends("ence") -> break
 case 'e':
   if ends("er") -> break
 case 'i':
   if ends("ic") -> break
 case 'l':
   if ends("able") -> break
   if ends("ible") -> break
 case 'v':
   if ends("ive") -> break
 … more rules here...
 default: return
if m() > 1
  k = j;
```

- Semantically →
  - semantically →
  - semanticalli →
  - semantical →
  - semantic → semant
- Destructiveness →
  - destructiveness →
  - destructiveness →
  - destructive →
  - destructive → destruct
- Recognizing →
  - recognize →
  - recognize →
  - recognize →
  - recognize → recogn

Step 6

Removes a final -e

```
if b[k] == 'e'
  int a = m()
  if a > 1 || a == 1 && !cvc(k-1)
    k--
if b[k]=='l' && doublec(k) && m() > 1
  k--
```

- Semantically →
  semantically →
  semanticalli →
  semantical →
  semantic →
  semant → ?
- Destructiveness →
  destructiveness →
  destructiveness →
  destructive →
  destructive →
  destruct → ?
- Recognizing →
  recognize →
  recognize →
  recognize →
  recognize →

Step 6

Removes a final -e

```
if b[k] == 'e'
  int a = m()
  if a > 1 || a == 1 && !cvc(k-1)
    k--
if b[k]=='l' && doublec(k) && m() > 1
  k--
```

- Semantically →
  semantically →
  semanticalli →
  semantical →
  semantic →
  semant → **semant**
- Destructiveness →
  destructiveness →
  destructiveness →
  destructive →
  destructive →
  destruct → **destruct**
- Recognizing →
  recognize →
  recognize →
  recognize →
  recognize →

Porter Mishaps

- Severing vs. several => sever
- University vs. universe => univers
- Iron vs. ironic => iron
- Animal vs. animated

Stemming Shortcomings

- Stemmers are rudimentary
- No word sense disambiguation (“bats” vs “batting”)
- No POS disambiguation (“Batting” could be noun or verb, but “hitting” could only be verb)
- Cannot handle irregular conjungation/inflection (“to be”, etc.)
- However – Lemmatization in practice does not do much better.

Further reading

- "Overview of Stemming Algorithms" Ilia Smirnov
  http://the-smirnovs.org/info/stemming.pdf
- Dr. Martin Porter's stemmer page
  http://tartarus.org/~martin/PorterStemmer/
- Javascript demo
  http://qaa.ath.cx/porter_js_demo.html