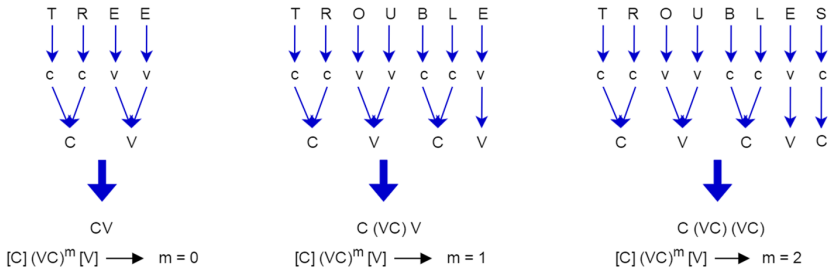

Porter Stemming Algorithm

SSES	→	SS	(m>0) ATIONAL	→	ATE
IES	→	I	(m>0) TIONAL	→	TION
SS	→	SS	(m>0) ENCI	→	ENCE
S	→		(m>0) ANCI	→	ANCE

Natural Language Processing Assignment 2

Implementing the Porter Stemming Algorithm

See on *Jupyter Notebook*

See Code on GitHub

15th August 2020

Anish Sachdeva
Delhi Technological University
DTU/2K16/MC/013
anish_@outlook.com
8287428181

Index

Porter's Stemmer Algorithm

For a running sample of the project see the Jupyter Notebook containing code with detailed explanation.

The Porter Stemmer Algorithm is used very heavily and ubiquitously in the field of Natural Language Processing (NLP) It is used for the morphological analysis of words to find their stems. The stem of a word is the main part also referred to as the root.

E.g Happy → happi, dogs → dog, cats → cat, sunny day → sunni day etc.

eWe find these morphological roots for the tasks of information extraction, information retrieval and also searching data in a big corpus of information.

The porter stemmer algorithm follows the following steps.

Step 1 A

The following word endings (suffixes) are transformed to other word endings.

SSES → SS

IES → I

SS → SS

S → ε

Step 1 B

(m > 0) EED → EE

ED → ε

ING → ε

If the second or the third rule are successful then the following need to be done.

AT → ATE

BL → BLE

IZ → IZE

Step 1 C

Y → I

Step 2

(m > 0) ATIONAL	→ ATE	relational	→ relate
(m > 0) TIONAL	→ TION	conditional	→ condition
		rational	→ rational
(m > 0) ENCI	→ ENCE	valenci	→ valence
(m > 0) ANCI	→ ANCE	hesitanci	→ hesitance
(m > 0) IZER	→ IZE	digitizer	→ digitize
(m > 0) ABLI	→ ABLE	conformabli	→ conformabl
(m > 0) ALLI	→ AL	radicalli	→ radical
(m > 0) ENTLI	→ ENT	differentli	→ different
(m > 0) ELI	→ E	vileli	→ vile
(m > 0) OUSLI	→ OUS	analogousli	→ analogous
(m > 0) IZATION	→ IZE	vietnamization	→ vietnamize
(m > 0) ATION	→ ATE	predication	→ predicate
(m > 0) ATOR	→ ATE	operator	→ operate
(m > 0) ALISM	→ AL	feudalism	→ feudal
(m > 0) IVENESS	→ IVE	decisiveness	→ decisive
(m > 0) FULNESS	→ FUL	hopefulness	→ hopeful
(m > 0) OUSNESS	→ OUS	callousness	→ callous
(m > 0) ALITI	→ AL	formaliti	→ formal
(m > 0) IVITI	→ IVE	sensitiviti	→ sensitive
(m > 0) BILITI	→ BLE	sensibiliti	→ sensible

Step 3

(m > 0) ICATE	→ IC	triplicate	→ triplic
(m > 0) ATIVE	→	formative	→ form
(m > 0) ALIZE	→ AL	formalize	→ formal
(m > 0) ICITI	→ IC	electriciti	→ electric
(m > 0) ICAL	→ IC	electrical	→ electric
(m > 0) FUL	→	hopeful	→ hope
(m > 0) NESS	→	goodness	→ good

Step 4

(m > 1) AL	→	revival	→ reviv
(m > 1) ANCE	→	allowance	→ allow

(m > l) ENCE	→	inference	→ infer
(m > l) ER	→	airliner	→ airlin
(m > l) IC	→	gyroscopic	→ gyroscop
(m > l) ABLE	→	adjustable	→ adjust
(m > l) IBLE	→	defensible	→ defens
(m > 1) ANT	→	irritant	→ irrit
(m > l) EMENT	→	replacement	→ replac
(m > l) MENT	→	adjustment	→ adjust
(m > l) ENT	→	dependent	→ depend
(m > l) and (*S or *T)) ION	→	adoption	→ adopt
(m > 1) OU	→	homologou	→ homolog
(m > l) ISM	→	communism	→ commun
(m > 1) ATE	→	activate	→ activ
(m > l) ITI	→	angulariti	→ angular
(m > l) OUS	→	homologous	→ homolog
(m > l) IVE	→	effective	→ effect
(m > l) IZE	→	bowdlerize	→ bowdler

Step 5

Step 5a			
(m > l) E	→	probate	→ probat
		rate	→ rate
(m = 1 and not *o) E	→	cease	→ ceas
Step 5b			
(m > land *d and *L)	→	single letter	control
	→	controll	→
		roll	→ roll

My Code (See Code + Explanation on GitHub)

```
class PorterStemmer:


    def __init__(self):
        """The word is a buffer holding a word to be stemmed. The letters are
in the range
        [start, offset ... offset + 1) ... ending at end."""


        self.vowels = ('a', 'e', 'i', 'o', 'u')
        self.word = ''
        self.end = 0
        self.start = 0
        self.offset = 0


    def is_vowel(self, letter):
        return letter in self.vowels


    def is_consonant(self, index):
        """:returns True if word[index] is a consonant."""
        if self.is_vowel(self.word[index]):
            return False
        if self.word[index] == 'y':
            if index == self.start:
                return True
            else:
                return not self.is_consonant(index - 1)
        return True


    def m(self):
        """m() measures the number of consonant sequences between start and
offset.
```

```
        if c is a consonant sequence and v a vowel sequence, and <..>
        indicates arbitrary presence,

           <c><v>       gives 0
           <c>vc<v>     gives 1
           <c>vcvc<v>   gives 2
           <c>vcvcvc<v> gives 3
           ....
        """
        n = 0
        i = self.start
        while True:
            if i > self.offset:
                return n
            if not self.is_consonant(i):
                break
            i += 1
        i += 1
        while True:
            while True:
                if i > self.offset:
                    return n
                if self.is_consonant(i):
                    break
                i += 1
            i += 1
            n += 1
            while True:
                if i > self.offset:
                    return n
                if not self.is_consonant(i):
                    break
```

```
                i += 1
            i += 1

    def contains_vowel(self):
        """:returns TRUE if the word contains a vowel in the range [start,
offset]"""
        for i in range(self.start, self.offset + 1):
            if not self.is_consonant(i):
                return True
        return False

    def contains_double_consonant(self, j):
        """:returns TRUE if the word contain a double consonant in the range
[offset, start]"""
        if j < (self.start + 1):
            return 0
        if (self.word[j] != self.word[j - 1]):
            return 0
        return self.is_consonant(j)

    def is_of_form_cvc(self, i):
        """:returns TRUE for indices set {i-2, i-1, i} has the form consonant -
vowel - consonant
        and also if the second c is not w,x or y. this is used when trying to
        restore an e at the end of a short  e.g.

           cav(e), lov(e), hop(e), crim(e), but
           snow, box, tray.
        """
        if i < (self.start + 2) or not self.is_consonant(i) or
self.is_consonant(i - 1) or not self.is_consonant(i - 2):
            return 0
        ch = self.word[i]
```

```
        if ch == 'w' or ch == 'x' or ch == 'y':
            return 0
        return 1

    def ends_with(self, s):
        """:returns TRUE when {start...end} ends with the string s."""
        length = len(s)
        if s[length - 1] != self.word[self.end]:  # tiny speed-up
            return 0
        if length > (self.end - self.start + 1):
            return 0
        if self.word[self.end - length + 1: self.end + 1] != s:
            return 0
        self.offset = self.end - length
        return 1

    def set_to(self, s):
        """sets [offset + 1, end] to the characters in the string s,
readjusting end."""
        length = len(s)
        self.word = self.word[:self.offset + 1] + s + self.word[self.offset +
length + 1:]
        self.end = self.offset + length

    def replace_morpheme(self, s):
        """is a mapping function to change morphemes"""
        if self.m() > 0:
            self.set_to(s)

    def remove_plurals(self):
        """This is step 1 ab and gets rid of plurals and -ed or -ing. e.g.
```

```
            caresses  ->  caress
            ponies    ->  poni
            ties      ->  ti
            caress    ->  caress
            cats      ->  cat

            feed      ->  feed
            agreed    ->  agree
            disabled  ->  disable

            matting   ->  mat
            mating    ->  mate
            meeting   ->  meet
            milling   ->  mill
            messing   ->  mess

            meetings  ->  meet
        """
        if self.word[self.end] == 's':
            if self.ends_with("sses"):
                self.end = self.end - 2
            elif self.ends_with("ies"):
                self.set_to("i")
            elif self.word[self.end - 1] != 's':
                self.end = self.end - 1
        if self.ends_with("eed"):
            if self.m() > 0:
                self.end = self.end - 1
        elif (self.ends_with("ed") or self.ends_with("ing")) and
self.contains_vowel():
            self.end = self.offset
            if self.ends_with("at"):
```

```
            self.set_to("ate")
        elif self.ends_with("bl"):
            self.set_to("ble")
        elif self.ends_with("iz"):
            self.set_to("ize")
        elif self.contains_double_consonant(self.end):
            self.end = self.end - 1
            ch = self.word[self.end]
            if ch == 'l' or ch == 's' or ch == 'z':
                self.end = self.end + 1
        elif self.m() == 1 and self.is_of_form_cvc(self.end):
            self.set_to("e")

    def terminal_y_to_i(self):
        """This defines step 1 c which turns terminal y to i when there is
another vowel in the stem."""
        if self.ends_with('y') and self.contains_vowel():
            self.word = self.word[:self.end] + 'i' + self.word[self.end + 1:]

    def map_double_to_single_suffix(self):
        """Defines step 2 and maps double suffices to single ones.
        so -ization ( = -ize plus -ation) maps to -ize etc. note that the
        string before the suffix must give m() > 0.
        """
        if self.word[self.end - 1] == 'a':
            if self.ends_with("ational"):
                self.replace_morpheme("ate")
            elif self.ends_with("tional"):
                self.replace_morpheme("tion")
        elif self.word[self.end - 1] == 'c':
            if self.ends_with("enci"):
                self.replace_morpheme("ence")
```

```
        elif self.ends_with("anci"):
            self.replace_morpheme("ance")
    elif self.word[self.end - 1] == 'e':
        if self.ends_with("izer"):      self.replace_morpheme("ize")
    elif self.word[self.end - 1] == 'l':
        if self.ends_with("bli"):
            self.replace_morpheme("ble")  # --DEPARTURE--
        # To match the published algorithm, replace this phrase with
        #   if self.ends("abli"):      self.r("able")
        elif self.ends_with("alli"):
            self.replace_morpheme("al")
        elif self.ends_with("entli"):
            self.replace_morpheme("ent")
        elif self.ends_with("eli"):
            self.replace_morpheme("e")
        elif self.ends_with("ousli"):
            self.replace_morpheme("ous")
    elif self.word[self.end - 1] == 'o':
        if self.ends_with("ization"):
            self.replace_morpheme("ize")
        elif self.ends_with("ation"):
            self.replace_morpheme("ate")
        elif self.ends_with("ator"):
            self.replace_morpheme("ate")
    elif self.word[self.end - 1] == 's':
        if self.ends_with("alism"):
            self.replace_morpheme("al")
        elif self.ends_with("iveness"):
            self.replace_morpheme("ive")
        elif self.ends_with("fulness"):
            self.replace_morpheme("ful")
        elif self.ends_with("ousness"):
```

```
            self.replace_morpheme("ous")
        elif self.word[self.end - 1] == 't':
            if self.ends_with("aliti"):
                self.replace_morpheme("al")
            elif self.ends_with("iviti"):
                self.replace_morpheme("ive")
            elif self.ends_with("biliti"):
                self.replace_morpheme("ble")
        elif self.word[self.end - 1] == 'g':  # --DEPARTURE--
            if self.ends_with("logi"):      self.replace_morpheme("log")
        # To match the published algorithm, delete this phrase

    def step3(self):
        """step3() deals with -ic-, -full, -ness etc."""
        if self.word[self.end] == 'e':
            if self.ends_with("icate"):
                self.replace_morpheme("ic")
            elif self.ends_with("ative"):
                self.replace_morpheme("")
            elif self.ends_with("alize"):
                self.replace_morpheme("al")
        elif self.word[self.end] == 'i':
            if self.ends_with("iciti"):     self.replace_morpheme("ic")
        elif self.word[self.end] == 'l':
            if self.ends_with("ical"):
                self.replace_morpheme("ic")
            elif self.ends_with("ful"):
                self.replace_morpheme("")
        elif self.word[self.end] == 's':
            if self.ends_with("ness"):      self.replace_morpheme("")

    def step4(self):
```

```
    """step4() takes off -ant, -ence etc., in context <c>vcvc<v>."""
    if self.word[self.end - 1] == 'a':
        if self.ends_with("al"):
            pass
        else:
            return
    elif self.word[self.end - 1] == 'c':
        if self.ends_with("ance"):
            pass
        elif self.ends_with("ence"):
            pass
        else:
            return
    elif self.word[self.end - 1] == 'e':
        if self.ends_with("er"):
            pass
        else:
            return
    elif self.word[self.end - 1] == 'i':
        if self.ends_with("ic"):
            pass
        else:
            return
    elif self.word[self.end - 1] == 'l':
        if self.ends_with("able"):
            pass
        elif self.ends_with("ible"):
            pass
        else:
            return
    elif self.word[self.end - 1] == 'n':
        if self.ends_with("ant"):
```

```
                pass
            elif self.ends_with("ement"):
                pass
            elif self.ends_with("ment"):
                pass
            elif self.ends_with("ent"):
                pass
            else:
                return
        elif self.word[self.end - 1] == 'o':
            if self.ends_with("ion") and (self.word[self.offset] == 's' or
self.word[self.offset] == 't'):
                pass
            elif self.ends_with("ou"):
                pass
            # takes care of -ous
            else:
                return
        elif self.word[self.end - 1] == 's':
            if self.ends_with("ism"):
                pass
            else:
                return
        elif self.word[self.end - 1] == 't':
            if self.ends_with("ate"):
                pass
            elif self.ends_with("iti"):
                pass
            else:
                return
        elif self.word[self.end - 1] == 'u':
            if self.ends_with("ous"):
```

```
                pass
            else:
                return
        elif self.word[self.end - 1] == 'v':
            if self.ends_with("ive"):
                pass
            else:
                return
        elif self.word[self.end - 1] == 'z':
            if self.ends_with("ize"):
                pass
            else:
                return
        else:
            return
        if self.m() > 1:
            self.end = self.offset

    def step5(self):
        """step5() removes a final -e if m() > 1, and changes -ll to -l if m >
1."""
        self.offset = self.end
        if self.word[self.end] == 'e':
            a = self.m()
            if a > 1 or (a == 1 and not self.is_of_form_cvc(self.end - 1)):
                self.end = self.end - 1
        if self.word[self.end] == 'l' and
self.contains_double_consonant(self.end) and self.m() > 1:
            self.end = self.end - 1

    def stem_document(self, document):
        result = []
```

```
        for line in document.split('\n'):
            result.append(self.stem_sentence(line))
        return '\n'.join(result)

    def stem_sentence(self, sentence):
        result = []
        for word in sentence.split():
            result.append(self.stem_word(word))
        return ' '.join(result)

    def stem_word(self, word):
        self.word = word
        self.end = len(word) - 1
        self.start = 0

        self.remove_plurals()
        self.terminal_y_to_i()
        self.map_double_to_single_suffix()
        self.step3()
        self.step4()
        self.step5()
        return self.word[self.start: self.end + 1]
```

Driver Code (See on GitHub)

```
stemmer = PorterStemmer()

resume = open('resume.txt', 'r').read()

print(stemmer.stem_document(resume))
```

My Input (Resume/CV) (See output in Jupyter Notebook)

My Resume was entered as a plain text present in a text file (resume.txt).

Anish Sachdeva

Software Developer + Clean Code Enthusiast

Phone : 8287428181

email : anish_@outlook.com

home : sandesh vihar, pitampura, new delhi - 110034

date of birth : 7th April 1998

languages : English, Hindi, French

Work Experience

What After College (4 months)

Delhi, India

Creating content to teach Core Java and Python with Data Structures and Algorithms and giving online classes to students

Summer Research Fellow at University of Auckland (2 Months)

Auckland, New Zealand

Worked on Geometry of Mobius Transformations, Differential Grometry under Dr. Pedram Hekmati at the Department of Mathematics, University of Auckland

Software Developer at CERN (14 Months)

CERN, Geneva, Switzerland

Worked in the core Platforms team of the FAP-BC group. Part of an agile team of developers that maintains and adds core functionality to applications used internally at CERN by HR, Financial, Administrative and other departments including Scientific

Worked on legacy applications that comprise of single and some times multiple frameworks such as Java Spring, Boot, Hibernate and Java EE. Also worked with Google Polymer 1.0 and JSP on the client side

Maintained CERN's Electronic Document Handing System application with >1M LOC that comprising of multiple frameworks and created ~20 years ago. Worked on feature requests, support requests and incidents and also release cycles

Teaching Assistant (4 Months)

Coding Ninjas, Delhi

Served as the teaching assistant to Nucleus - Java with DS batch, under Mr. Ankur Kumar. Worked on creating course content and quizzes for online platform of Coding Ninjas for Java. Helped students in core Data Structures and Algorithms concepts in Java

Education

Delhi Technological University (2016 - 2021)

Bachelors of Technology Mathematics and Computing

CGPA: 9.2

The Heritage School Rohini (2004 - 2016)

Physics, Chemistry, Maths + Computer Science with English

Senior Secondary: 94.8%

Secondary: 9.8 CGPA

Technical Skills

Java + Algorithms and Data Structures

MEAN Stack Web Development

Python + Machine Learning

MATLAB + Octave

MySQL, PostgresSQL & MongoDB

Other Skills

MS Office, Adobe Photoshop, LaTeX + MiTeX

University Courses

Applied Mathematics I, II, III

Linear Algebra + Probability & Statistics + Stochastic Processes + Discrete Maths

Computer Organization & Architecture + Data Structures + Algorithm Design and Analysis + DBMS + OS

Computer Vision + NLP

Important Links

https://www.linkedin.com/in/anishsachdeva1998/

https://github.com/anishLearnsToCode

https://www.hackerrank.com/anishviewer

My Output (See in Jupyter Notebook)

Anish Sachdeva

Softwar Develop + Clean Code Enthusiast

Phone : 8287428181

email : anish_@outlook.com

home : sandesh vihar, pitampura, new delhi - 110034

date of birth : 7th April 1998

languag : English, Hindi, French

Work Experienc

What After Colleg (4 months)

Delhi, India

Creat content to teach Core Java and Python with Data Structur and Algorithm and give onlin class to student

Summer Research Fellow at Univers of Auckland (2 Months)

Auckland, New Zealand

Work on Geometri of Mobiu Transformations, Differenti Grometri under Dr. Pedram Hekmati at the Depart of Mathematics, Univers of Auckland

Softwar Develop at CERN (14 Months)

CERN, Geneva, Switzerland

Work in the core Platform team of the FAP-BC group. Part of an agil team of develop that maintain and add core function to applic us intern at CERN by HR, Financial, Administr and other depart includ Scientif

Work on legaci applic that compris of singl and some time multipl framework such a Java Spring, Boot, Hibern and Java EE. Also work with Googl Polym 1.0 and JSP on the client side

Maintain CERN' Electron Document Hand System applic with >1M LOC that compris of multipl framework and creat ~20 year ago. Work on featur requests, support request and incid and also releas cycl

Teach Assistant (4 Months)

Code Ninjas, Delhi

Serv a the teach assist to Nucleu - Java with DS batch, under Mr. Ankur Kumar. Work on creat cours content and quizz for onlin platform of Code Ninja for Java. Help student in core Data Structur and Algorithm concept in Java

Educat

Delhi Technolog Univers (2016 - 2021)

Bachelor of Technolog Mathemat and Comput

CGPA: 9.2

The Heritag School Rohini (2004 - 2016)

Physics, Chemistry, Math + Comput Scienc with English

Senior Secondary: 94.8%

Secondary: 9.8 CGPA

Technic Skill

Java + Algorithm and Data Structur

MEAN Stack Web Develop

Python + Machin Learn

MATLAB + Octave

MySQL, PostgresSQL & MongoDB

Other Skill

MS Office, Adobe Photoshop, LaTeX + MiTeX

Univers Cours

Appli Mathemat I, II, III

Linear Algebra + Probabl & Statist + Stochast Process + Discret Math

Comput Organiz & Architectur + Data Structur + Algorithm Design and Analysi + DBMS + OS

Comput Vision + NLP

Important Link

https://www.linkedin.com/in/anishsachdeva1998/

https://github.com/anishLearnsToCod

https://www.hackerrank.com/anishview

Advantages & Disadvantages of Using the Porter Stemmer Algorithm on a Resume/CV (See Analytics Code on GitHub)

Running a stemming algorithm like the porter stemmer algorithm on a small corpus of information can be advantageous as if a user then wishes to search something then the search string can also be stemmed according to the same algorithm and then the stemmed query string can be looked up (using a lookup table or other data structure) and the results can be retrieved quickly so even for many different strings that do not have exact matches in the corp[ora (resume in this case) there will be many cases despite the corpora being having a small number of textual data.

There are also some very major disadvantages of using the porter stemmer on a Resume. Resume and CV contain some very important information where the exact configuration of the text is very important such as address, email address, phone numbers, University Name and degree name. These information are very precise information and by stemming we are actually losing that precision. If we stem the email address or if we stem the home address or if we stem the birth date, important portfolio links etc. the simply searching for google through the entire corpora might give us back many results but none of those results will be valid email addresses and no result will be a valid google drive link so the information has been rendered useless.

Also word stemming is used where the domain of the words is very high, Here the domain refers to all the possible words that can be used and technically in an English resume the person can use any english word, but normally when describing themselves in a resume people have a fixed vocabulary and skills etc. are also picked from a fixed set of skills and names of famous programming languages and also the possible number of engineering degrees or other diplomas that people can have are a finite number so even if we transform from the original domain to a stemmed version of every word the unique words and identifiers that we have will not decrease drastically but the information loss will definitely be there.

I have performed this statistical analysis on my resume and compared the number of unique words before and after stemming my resume and below is the result.

Number of Words in Resume: 369

Number Of Unique Words In Resume: 251

Number of Unique Words after Stemming: 241

So, we can clearly see from the above figure that stemming has't decreased the number of unique words by a drastic amount.

It should also be mentioned that stemming actually can be very beneficial for smaller corpora as larger corpora have higher probabilities that various forms of the stemmed words will be covered, whereas in a smaller corpora it isn't very likely that the various forms of the same root word will be covered hence by stemming in a smaller corpora we can drastically improve search query as teh search query can also be stemmed and then a match can be found which wouldn't have existed otherwise in the smaller corpora.

So, depending on the application as to what we are trying to achieve with our corpora and also whether we are simply working on our resume or a big data set of resumes stemming can be both advantageous and disadvantageous.

If your application mainly concerns itself with Information extraction and proper information retrieval from a single resume stemming would result in loss of data and is probably not the place to go.

But if we have to search a large corpora of data or all candidates that are familiar with java and workplace management and SAp skills then stemming can help us reduce the unique number of words to some extent and then perform a broad search on these criteria.

Bibliography

1. Natural Language Processing ~Jurafsky
2. NaturaL Language Processing [Wikipedia]
3. Stemming [Wikipedia]
4. Martin Porter [Wikipedia]