

nグラム超入門 ～ FSNLP §6 を読みながら

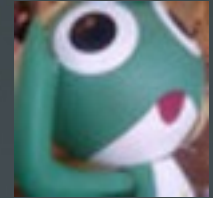
- presented by naoya_t
- 自然言語処理の超基礎テクニックである n -gram について、FSNLP の6章とか奥村本の2章あたりを読みながら超入門的なことを話します
- 超入門です。今からお話しする程度の事は皆さん既にご存知かと思います。シルバーウィークに遊び疲れた皆さん、発表資料作り等で寝不足な皆さんは暫しご休憩(ないしご内職)ください



はじめに



自己紹介



- @naoya_t
- Shibuya.lisp のあまり働かない発起人の人
- PRML 読書会の幹事の人
1周目が読了して只今2周目(復習レーン)
- 競技プログラミング(TopCoder とか)ファンの人
黄色では恐れ多くてファンとしか言えない…
- 仕事では NLP とか ML とかは特に何も。
(Lisp も仕事では何も…)



忘れないうちに告知

- 11/6sat.
PRML 復習レーン #5
次回は4章を読みます。
公式タグ #PRMLrevenge
- 11/27sat.
Shibuya.lisp Technical Talk #5
- ↑ どちらも会場は EC ナビ(ここです!)
@ajiyoshi さんいつもありがとうございます



n グラ超入門



最初に、 n グラムの読み方書き方から

- n -gram 「エヌ グラム」。

n	名称	読み方	百度風
1	unigram	ユニグラム	1 グラム
2	bigram	バイグラム	2 グラム
3	trigram	トライグラム	3 グラム
4	four-gram	フォー・グラム	4 グラム
5	five-gram	ファイブ・グラム	5 グラム

- $n \geq 4$ の場合、数字の英語名 + gram のように読まれる。英語・ラテン語 (uni-, bi-, tri-) ・ギリシャ語 (-gram) が混ざって気持ち悪いですね。

で、 n グラムとは何か

- 連続する n 単語 (ないし n 文字) を1つの塊とみなしたものの。

単語単位	単語 n グラム (word n -gram)
文字単位	文字 n グラム (letter n -gram, character n -gram)

- 語順 (ないし文字の並び) の情報を n 個分だけ保持する性質を利用して
 - あんなことや
 - こんなことを



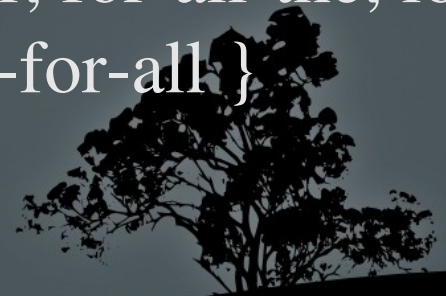
単語 n グラ

- **例題** ” so long, and thanks for all the fish.” という例文を考えてみる。この例文が含んでいる1 グラ, 2 グラ, 3 グラはそれぞれなんであるか。



単語 n グラ

- **例題** ” so long, and thanks for all the fish.” という例文を考えてみる。この例文が含んでいる 1 グラ, 2 グラ, 3 グラはそれぞれなんであるか。
- **答**
 - 1 グラ : { all, and, fish, for, long, so, thanks, the }
 - 2 グラ : { all-the, and-thanks, for-all, long-and, so-long, thanks-for, the-fish }
 - 3 グラ : { all-the-fish, and-thanks-for, for-all-the, long-and-thanks, so-long-and, thanks-for-all }



単語 n グラムの特徴

- $n=1$ (unigram) では語順の情報が完全に失われていて、何の話だったか想像しにくい。それでも特徴的な単語が多めに含まれていれば何とかなるかも
- n が大きくなるにつれ、語順情報が入ってくるので元の文が少しずつ見えてくる。
- 文に含まれる語数が n に等しいとき、 n -gram に元の文がまるごと含まれる。



ダミー単語の挿入

- 単語が文の最初や最後に出現するという情報を n -gram に含めるために、文の先頭と末尾にダミーの単語を入れることができる。
 - 2 グラムの例 : { ^-so, all-the, and-thanks, fish-\$, for-all, long-and, so-long, thanks-for, the-fish }
 - 3 グラムの例 : { ^-so-long, all-the-fish, and-thanks-for, for-all-the, long-and-thanks, so-long-and, thanks-for-all, the-fish-\$ }



文字 n グラム

- 例題 「ごはんはおかず」という日本語文が含む文字 2 グラム、および文字 3 グラムを列挙せよ。



文字 n グラ

- 例題 「ごはんはおかず」という日本語文が含む文字 2 グラ、および文字 3 グラを列挙せよ。
- 答
 - 2 グラ : { おか, かず, ごは, はお, はん, んは }
 - 3 グラ : { おかず, ごはん, はおか, はんは, んはお }



文字 n グラムについて補足

- 空白を文字として扱えば、(単語に限らず)文についても文字 n -gram を考えることができる。
- 単語(文字列)の先頭と末尾にダミーの文字を追加することで、ある文字が単語(文字列)の先頭ないし末尾に出現するという情報を文字 n -gram に取り込むことができる



誰得？何得？n グラ



1) 文章の素性値としての n グラム

- 語彙リストが与えられているとして、ある文章における各語彙の出現情報をベクトルで表現したものをその文章の素性として扱う → bag-of-words
 - 出現頻度を表現 → 頻度ベクトル
 - 出現の有無を表現 → 二値ベクトル
- ここで単語の代わりに n -gram を用いることで、(部分的にはあるが) 語順の情報を素性として取り込むことができる → bag-of-ngrams



bag-of-bigrams

- 文字 n-gram でベクトル化してみる
 - s1: 「ごはんはおかず」
 - s2: 「むしろごはんがおかずだよ」



bag-of-bigrams

- 文字 n-gram でベクトル化してみる
 - s1: 「ごはんはおかず」
 - s2: 「むしろごはんがおかずだよ」
 - // おかず, かずだ, がおか, ごはん, しろご, ずだよ, はおか, はんが, はんは, むしろ, ろごは, んがお, んはお



bag-of-trigrams

- 文字 3-gram でベクトル化してみる
 - s1: 「ごはんはおかず」
 - s2: 「むしろごはんがおかずだよ」
 - // おかず, かずだ, がおか, ごはん, しろご, ずだよ, はおか, はんが, はんは, むしろ, ろごは, んがお, んはお
- $X_{\text{bigrams}}^{(s1)} = (1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1)$
- $X_{\text{bigrams}}^{(s2)} = (1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0)$



bag-of-bigrams

- 文字 n-gram でベクトル化してみる
 - s1: 「ごはんはおかず」
 - s2: 「むしろごはんがおかずだよ」
 - // おかず, かずだ, がおか, ごはん, しろご, ずだよ, はおか, はんが, はんは, むしろ, ろごは, んがお, んはお
- $X_{\text{bigrams}}^{(s1)} = (1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1)$
- $X_{\text{bigrams}}^{(s2)} = (1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0)$
- 余弦類似度 $= 2 / 5\sqrt{2} \doteq 0.2828$



2) n グラムで次に来る単語を予測

- 言語を(文法構造とか無視して、単純に) n -gram でモデル化する。
- ある単語 w_n が出現する確率を、直前の $n-1$ 語 w_1, \dots, w_{n-1} の出現を前提とした条件つき確率 $p(w_n | w_1, \dots, w_{n-1})$ で表す
- 直前の数語 ($n-1$ 語)のみが次の単語(第 n 語)の出現に影響を及ぼすと仮定している。
これは! $n-1$ 次のマルコフモデル!!



n グラム、あるいは $n-1$ 次マルコフ連鎖

- 直前の $n-1$ 語 (w_1, \dots, w_{n-1}) がその先頭 $n-1$ 語にマッチするような n -gram を、(巨大なコーパスから学習した中から) 探す
- 次に来る語 w_n は、 n -gram の最後の語に対応
- 各 n -gram の出現頻度に比例して、 n -gram の最後の語が出現する



なぜ n グラムで言語をモデル化するのか

- 他にいくらでもモデル化の方法がありそうなのに
- 言語に関する知識を利用して、例えば次の単語の予測なら、直前の単語と、直前の「述語動詞(英語の場合)」から推測してはどうか?
→どれがその文のメインの述語動詞かをそう簡単には識別できない等、割とやりにくい
- その点、 n -gram は非常に単純な割に、驚くほどうまく行く(当社比)モデル。
3-gram に勝つのは意外に難しいよ



n グラムの残念なところ

- 語彙数のn乗にほぼ比例してメモリ空間やら CPU 時間やらを消費。
- 語彙数を(控えめに?) 20000としても

2-gram	$20000 \times (20000 - 1) \doteq 4.0 \times 10^8$
3-gram	$20000 \times 20000 \times (20000 - 1) \doteq 8.0 \times 10^{12}$
4-gram	$20000 \times 20000 \times 20000 \times (20000 - 1) \doteq 1.6 \times 10^{17}$

※ 20000-1 というのは(総和が 1 という確率の制約上)最後の値は他の全ての値から求まる為

- 賢くするには超巨大なコーパスが必要。
- 

n グラム 残念解消法

- n -gram の n (次元数) を減らす
実用的な範囲で 2-gram とか 3-gram がよく用いられる
- 語彙数を減らす
stemming: 派生語を同一素性として扱う
lemmatization (見出し語化): 活用語等は原形が共通なら同一素性として扱う
- 計算資源を湯水のように使える環境を構築
- とりあえずコーパスはでかいのを用意



軽く応用編



実際のテキストで n -gram してみよう

- The Project Gutenberg から、みんなが大好き Jane Austen の作品を (Manning & Schütze 両先生が) 落としてきて句読点等を抜いた (その結果を FSNLP のサイトから頂いてきた) 物
- 訓練データ : Emma, Mansfield Park, Northanger Abbey, Pride and Prejudice, Sense and Sensibility
→ $N = 617,091$ [words of text] / $V = 14,585$ [word types]; コーパスとしては小さめ
- テストデータ : Persuasion



実証コード書いてみた

- 最尤推定 (MLE) してみても FSNLP の Table 6.3 (p.200) のような結果になるのか、コードを書いて試してみた
- 資料つくる時間が足りなくてコードをコピペ><
- 結果:再現できたっぽい!



```
(use srfi-1)
(use srfi-13)
```

```
(define (string-list< sl1 sl2)
  (cond [(null? sl1) #t]
        [(null? sl2) #f]
        [(string< (car sl1) (car sl2)) #t]
        [(string> (car sl1) (car sl2)) #f]
        [else (string-list< (cdr sl1) (cdr sl2))]))
```

```
(define (uniq-cnt ls)
  (let1 ht (make-hash-table 'equal?)
    (for-each
      (lambda (elm)
        (hash-table-put! ht elm (+ 1 (hash-table-get ht elm 0))))
      ls)
    (hash-table-map ht cons)))
```

```
(define (n-gram n words)
  (if (= n 1) (map list words)
      (let1 s (append (cons "^" words) ("$$$"))
        (let loop ((l (length s)) (rest s) (result '()))
          (if (< l n) result
              (loop (- l 1) (cdr rest) (cons (take rest n) result)))))))
```

```
(define (n-1-gram n n-gram)
  (let ((n-1 (- n 1))
        (ht (make-hash-table 'equal?))
        (htc (make-hash-table 'equal?)))
    (for-each
      (lambda (p)
        (let* ((ng (car p)) (cnt (cdr p))
              (n-1g (take ng n-1)) (w (car (last-pair ng))))
          (print "- " n-1g " " w " " cnt)
          (let1 ht* (or (hash-table-get ht n-1g #f) (make-hash-table 'equal?))
            (hash-table-put! ht* w cnt)
            (hash-table-put! ht n-1g ht*))
          (hash-table-put! htc n-1g
            (+ cnt (hash-table-get htc n-1g 0)))
          ))
      n-gram)
  (values ht htc)))
```



```

(define (load-file-as-wordlist path)
  (with-input-from-file path
    (lambda ()
      (let loop ((buf '()))
        (let1 line (read-line)
          (cond [(eof-object? line)
                 (apply append (reverse! buf))]
                [else
                 (let1 splitted (string-split line #\Space)
                   (loop (cons splitted buf))))))))))

(define austen (load-file-as-wordlist "austen.txt"))
(define persuasion (load-file-as-wordlist "ja-pers-clean.txt"))

(define (not-blank? w) (not (string=? "" w)))
(define austen-N 617091) ;(length (filter not-blank? austen))) ; 617091
(define austen-V 14585) ;(- (length austen-1-gram) 1) ; 14585

;; n-gram
(let* ((n 3)
      (austen-n-gram (uniq-cnt (n-gram n austen)))
      (persuasion-n-gram (n-gram n persuasion)))
  (receive (ht hcnt) (n-1-gram n austen-n-gram)
    (for-each
     (lambda (ws)
       (let* ((n-1g (take ws (- n 1)))
              (w (car (last-pair ws)))
              (den (hash-table-get hcnt n-1g #f))
              (ht* (hash-table-get ht n-1g #f))
              (num (if ht* (hash-table-get ht* w #f) #f))
              (r (if (and num den) (/ num den) 0)))
         (format #t "p(~a|~a) = ~a\n" w (string-join n-1g ",") r)))
      persuasion-n-gram ))

```



...

$p(\text{My}|\text{lady},) = 0$

$p(|\text{that},\text{lady}) = 0$

$p(\text{lady}|\text{than},\text{that}) = 0$

$p(\text{that}|\text{even},\text{than}) = 0$

$p(\text{than}|\text{enemy},\text{even}) = 0$

$p(\text{even}|\text{my},\text{enemy}) = 0$

$p(\text{enemy}|\text{more},\text{my}) = 0$

$p(\text{my}|\text{person},\text{more}) = 0$

$p(\text{more}|\text{one},\text{person}) = 0$

$p(\text{person}|\text{been},\text{one}) = 0$

$p(\text{one}|\text{have},\text{been}) = 0.0011947431302270011$

$p(\text{been}|\text{not},\text{have}) = 0.15120274914089346$

$p(\text{have}|\text{may},\text{not}) = 0.1276595744680851$

$p(\text{not}|\text{there},\text{may}) = 0.0625$

$p(\text{may}|\text{whether},\text{there}) = 0$

$p(\text{there}|\text{itself},\text{whether}) = 0$

$p(\text{whether}|\text{suggested},\text{itself}) = 0$

$p(\text{itself}|\text{has},\text{suggested}) = 0$

$p(\text{suggested}|\text{question},\text{has}) = 0$

$p(\text{has}|\text{question}) = 0$

$p(\text{question}|\text{a},) = 0$

$p(|\text{and},\text{a}) = 0.0035335689045936395$

$p(\text{a}|\text{past},\text{and}) = 0$

$p(\text{and}|\text{the},\text{past}) = 0.2$

$p(\text{past}|\text{over},\text{the}) = 0.010752688172043012$

$p(\text{the}|\text{thinking},\text{over}) = 1.0$

$p(\text{over}|\text{been},\text{thinking}) = 0$

$p(\text{thinking}|\text{have},\text{been}) = 0.0035842293906810036$

$p(\text{been}|\text{too},\text{have}) = 0$

...



まとめ



(略)



トックス !!

