自然言語処理プログラミング勉強会 2
n-gram 言語モデル

Graham Neubig
奈良先端科学技術大学院大学 (NAIST)

先週の復習：文の確率計算

- 文の確率が欲しい

W = speech recognition system

- 変数で以下のように表す (連鎖の法則を用いて):

$P(|W| = 3, w_1=\text{"speech"}, w_2=\text{"recognition"}, w_3=\text{"system"}) =$

$P(w_1=\text{"speech"} \mid w_0 = \text{"<s>"})$

$* P(w_2=\text{"recognition"} \mid w_0 = \text{"<s>"}, w_1=\text{"speech"})$

$* P(w_3=\text{"system"} \mid w_0 = \text{"<s>"}, w_1=\text{"speech"}, w_2=\text{"recognition"})$

$* P(w_4=\text{"</s>"} \mid w_0 = \text{"<s>"}, w_1=\text{"speech"}, w_2=\text{"recognition"}, w_3=\text{"system"})$

注：
文頭「 <s> 」と文末「 </s> 」記号

注：
$P(w_0 = \text{<s>}) = 1$

確率の漸次的な計算

- 前のスライドの積を以下のように一般化

$$P(W) = \prod_{i=1}^{|W|+1} P(w_i | w_0 \dots w_{i-1})$$

- 以下の条件付き確率の決め方は？

$$P(w_i | w_0 \dots w_{i-1})$$

1-gram モデルは語順を考慮しない

- 以下の確率は同等

P_{uni}(w=speech recognition system) =
P(w=speech) * P(w=recognition) * P(w=system) * P(w=</s>)

=

P_{uni}(w=system recognition speech) =
P(w=speech) * P(w=recognition) * P(w=system) * P(w=</s>)

1-gram モデルは単語の関係性を考慮しない

- 文法的な文：（名詞と活用が一致）

P_{uni}(w=i am) =
P(w=i) * P(w=am) * P(w=</s>)

P_{uni}(w=we are) =
P(w=we) * P(w=are) * P(w=</s>)

- 文法的でない文：（名詞と活用が矛盾）

P_{uni}(w=we am) =
P(w=we) * P(w=am) * P(w=</s>)

P_{uni}(w=i are) =
P(w=i) * P(w=are) * P(w=</s>)

しかし、確率は上記の文と同等

文脈を考慮することで解決！

- 1-gram モデルは文脈を考慮しない

$$P(w_i|w_0 \ldots w_{i-1}) \approx P(w_i)$$

- 2-gram は 1 単語の文脈を考慮

$$P(w_i|w_0 \ldots w_{i-1}) \approx P(w_i|w_{i-1})$$

- 3-gram は 2 単語の文脈を考慮

$$P(w_i|w_0 \ldots w_{i-1}) \approx P(w_i|w_{i-2}w_{i-1})$$

- 4-gram 、 5-gram 、 6-gram などなど

n-gram 確率の最尤推定

- n 単語と n-1 単語からなる文字列の頻度を利用

$$P(w_i|w_{i-n+1}\cdots w_{i-1}) = \frac{c(w_{i-n+1}\cdots w_i)}{c(w_{i-n+1}\cdots w_{i-1})}$$

i live in osaka . </s>
i am a graduate student . </s>
my school is in nara . </s>

n=2 →
P(osaka | in) = c(in osaka)/c(in) = 1 / 2 = 0.5
P(nara | in) = c(in nara)/c(in) = 1 / 2 = 0.5

低頻度 n-gram の問題

- n-gram 頻度が 0→n-gram 確率も 0

P(osaka | in) = c(in osaka)/c(in) = 1 / 2 = 0.5
P(nara | in) = c(in nara)/c(in) = 1 / 2 = 0.5
P(school | in) = c(in school)/c(in) = 0 / 2 = **0**!!

- 1-gram モデルと同じく、線形補間を用いる

2-gram: $$P(w_i|w_{i-1}) = \lambda_2 P_{ML}(w_i|w_{i-1}) + (1-\lambda_2) P(w_i)$$

1-gram: $$P(w_i) = \lambda_1 P_{ML}(w_i) + (1-\lambda_1)\frac{1}{N}$$

補間係数の選択法：グリッド探索

- λ_2 と λ_1 の様々な値を試し、尤度が最も高くなるように選択

$$\lambda_2 = 0.95, \lambda_1 = 0.95$$
$$\lambda_2 = 0.95, \lambda_1 = 0.90$$
$$\lambda_2 = 0.95, \lambda_1 = 0.85$$
$$\dots$$
$$\lambda_2 = 0.95, \lambda_1 = 0.05$$
$$\lambda_2 = 0.90, \lambda_1 = 0.95$$
$$\lambda_2 = 0.90, \lambda_1 = 0.90$$
$$\dots$$
$$\lambda_2 = 0.05, \lambda_1 = 0.10$$
$$\lambda_2 = 0.05, \lambda_1 = 0.05$$

問題：

選択肢が多すぎる
→ 選択に時間がかかる！

全ての n-gram に対して同じ λ
→ 尤度が最適とは限らない！

文脈を考慮した補間係数の選択

頻度の高い単語： Tokyo

c(Tokyo city) = 40
c(Tokyo is) = 35
c(Tokyo was) = 24
c(Tokyo tower) = 15
c(Tokyo port) = 10
…

ほとんどの 2-gram が既観測
→ 大きな λ が最適

頻度の低い単語： Tottori

c(Tottori is) = 2
c(Tottori city) = 1
c(Tottori was) = 0

未観測の 2-gram が多い
→ 小さな λ が最適

- 補間係数の選択にも文脈を考慮：

$$P(w_i|w_{i-1}) = \lambda_{w_{i-1}} P_{ML}(w_i|w_{i-1}) + (1-\lambda_{w_{i-1}}) P(w_i)$$

Witten-Bell 平滑化

- $\lambda_{w_{i-1}}$ を選ぶ方法の 1 つ

$$\lambda_{w_{i-1}} = 1 - \frac{u(w_{i-1})}{u(w_{i-1}) + c(w_{i-1})}$$

$u(w_{i-1})$ = w_{i-1} の後に続く単語の異なり数

- 例えば、

c(Tottori is) = 2　c(Tottori city) = 1
c(Tottori) = 3　u(Tottori) = 2

$$\lambda_{Tottori} = 1 - \frac{2}{2+3} = 0.6$$

c(Tokyo city) = 40 c(Tokyo is) = 35 ...
c(Tokyo) = 270　u(Tokyo) = 30

$$\lambda_{Tokyo} = 1 - \frac{30}{30+270} = 0.9$$

言語モデルのための プログラミング技術

配列への挿入

- 文頭・文末記号を考慮するために以下の操作を利用

my_words = [“this”, “is”, “a”, “pen”]

↓

my_words = [“<s>”, “this”, “is”, “a”, “pen”, “</s>”]

- Python で append と insert 関数を利用

```
my_words.append(“</s>”)   # 配列の最後い挿入
my_words.insert(0, “<s>”)   # 配列の最初に挿入
```

配列からの削除

- n-gram w_{i-n+1} … w_i が与えられた場合、文脈 w_{i-n+1} … w_{i-1} を以下のように計算

```
my_ngram = “tokyo tower”
my_words = my_ngram.split(“ “)  # [“tokyo”, “tower”] へ変換
my_words.pop()                  #  最後の要素 (“tower”) を削除
my_context = “ “.join(my_words) #  配列をもう一度文字列へ連結
print my_context
```

演習問題

演習問題

- 2 つのプログラムを作成
  - train-bigram: 2-gram モデルを学習
  - test-bigram: 2-gram モデルに基づいて評価データのエントロピーを計算
- テスト入力： test/02-train-input.txt
- 学習データ： data/wiki-en-train.word
- data/wiki-en-test.word に対してエントロピーを計算（線形補間を用いる場合、様々な λ_2 を試す）
- 上級編：
  - Witten-Bell 平滑化を利用（線形補間の方が簡単）
  - 任意な文脈長が利用可能なプログラムを作成

train-bigram 擬似コード（線形補間）

create **map** *counts, context_counts*

for each *line* **in** the *training_file*
  split *line* into an array of *words*
  append “</s>” to the end and “<s>” to the beginning of *words*
  for each i **in** 1 **to** length(*words*)-1 # 注：<s> の後に始まる
    counts[“w_{i-1} w_i”] += 1 # 2-gram の分子と分母を加算
    context_counts[“w_{i-1}”] += 1
    counts[“w_i”] += 1 # 1-gram の分子と分母を加算
    context_counts[“”] += 1

open the *model_file* for writing
for each *ngram*, *count* **in** *counts*
  split *ngram* into an array of *words* # “w_{i-1} w_i” → {“w_{i-1}”, “w_i”}
  remove the last element of *words* # {“w_{i-1}”, “w_i”} → {“w_{i-1}”}
  join *words* into *context* # {“w_{i-1}”} → “w_{i-1}”
  probability = *counts*[*ngram*]/context_*counts*[*context*]
  print *ngram*, *probability* **to** *model_file*

test-bigram 擬似コード（線形補間）

λ_1 = ???, λ_2 = ???, V = 1000000, W = 0, H = 0

load model into *probs*

for each *line* **in** *test_file*
  split *line* into an array of *words*
  append “</s>” to the end and “<s>” to the beginning of *words*
  for each i **in** 1 **to** length(*words*)-1 # 注： <s> の後に始まる
    P1 = λ_1 *probs*[“w_i”] + (1 – λ_1) / V # 1-gram の平滑化された確率
    P2 = λ_2 *probs*[“w_{i-1} w_i”] + (1 – λ_2) * P1 # 2-gram の平滑化された確率
    H += -$\log_2$(2)
    W += 1

print “entropy = ”+*H/W*