语言模型

定义

将一段自然语言文本看作一串离散的时间序列，假设对于一段长度为 T 的文本中的词以此为$w_1, w_2, \dots, w_T$ ，则对于该离散的时间序列，w_t表示在 time step = t 的时候的输出或 label。则有定义，对于给定一个长度为 T 的词序列$w_1, w_2, \dots, w_T$ ，语言模型（即该序列的概率）为

$$P(w_1, w_2, \dots, w_T)$$

N-gram

当取 T=4 的时候，我们可以根据联合概率的计算方法得到

$$P(w_1, w_2, w_3, w_4)=P(w_1)P(w_2|w_1)P(w_3|w_1,w_2)P(w_4|w_1,w_2,w_3)$$

更一般的，对于序列$w_1, w_2, \dots, w_T$ ，其概率计算可以表示为

$$P(w_1, w_2,\dots, w_T)=\prod_{t=1}^TP(w_t|w_1, w_2, \dots, w_{t-1})$$

引入马尔可夫链

由于当序列长度非常长的时候，计算和存储的复杂度会指数级别增长，且数据会十分稀疏从而导致概率失真。通过引入马尔可夫链来简化计算。

假设：一个词的出现只与其前面 n 个词相关，也就是**n 阶马尔可夫链**

$$P(w_1, w_2,\dots, w_T) \approx\prod_{t=1}^TP(w_t|w_{t-n+1}, \dots, w_{t-1})$$

n 的取值

- 当 n 太小的时候，模型无法准确的学习到真正相关词之间的关系，欠拟合，high bias
- 当 n 太大的时候，退化成原始计算方法，导致计算存储复杂度很大，数据很稀疏概率失真。high variance
- 根据 n 的取值不同，主要有 unigram(n=1)，bigram(n=2)，trigram(n=3)
    - unigram：假设每个单词都是独立的
    - bigram：只依赖于前面一个词

平滑法

Lidstone smoothing $\alpha=k$

某些词可能未出现在 vocab 中或频次很小很小，则对于分母(denominator)可能出现零概率问题，因此与 Naive Bayers 中类似可以采用平滑技术

$$P_{smooth}(w_i|w_{i-n+1}, \dots, w_{i-1})=\frac{C(w_{i-n+1},\dots,w_i)+\alpha}{C(w_{i-n+1},\dots,w_{i-1})+\alpha|V|}$$

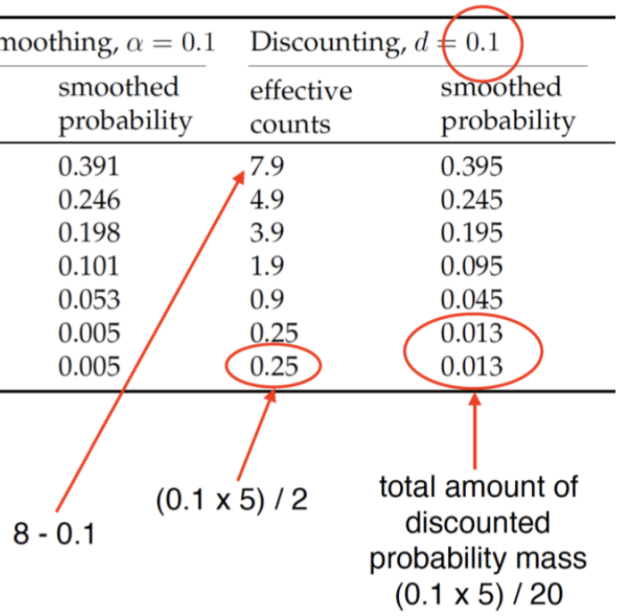
- 我们将分子称为**有效计数（effective counts）**

- 当$\alpha=1$的时候是我们常见的**拉普拉斯法(Laplace Smoothing)**
- 当$\alpha=0.5$的时候称为**Jeffreys-Perks law **
- 我们可以发现此时对于每个可能的w，有效计数与真实计数之间的差值是不固定的

Absolute Discounting smoothing

- 从每个**观测到的** n-gram 计数中“借”一个**固定的概率质量（a fixed probability mass）**d
- 将其**重新分配（redistribute）到未观测到的** n-gram
- e.g.:
  - context = 'alleged'
  - 5 observed bi-grams: i.e., C('alleged'|char $\in${'impropriety', 'offense', 'damage', 'deficiencies', 'outbreak'}) > 0
  - 2 unobserved bi-grams: i.e., C('alleged' | 'infirmity') = 0 and C('alleged' | 'cephalopods') = 0
    - context $= alleged$
    - 5 observed bi-grams
    - 2 unobserved bi-grams

	counts	unsmoothed probability	Lidstone smoothing, $\alpha = 0.1$ effective counts	Lidstone smoothing, $\alpha = 0.1$ smoothed probability	Discounting, $d = 0.1$ effective counts	Discounting, $d = 0.1$ smoothed probability
impropriety	8	0.4	7.826	0.391	7.9	0.395
offense	5	0.25	4.928	0.246	4.9	0.245
damage	4	0.2	3.961	0.198	3.9	0.195
deficiencies	2	0.1	2.029	0.101	1.9	0.095
outbreak	1	0.05	1.063	0.053	0.9	0.045
infirmity	0	0	0.097	0.005	0.25	0.013
cephalopods	0	0	0.097	0.005	0.25	0.013



  - 对于

$$ P_{ADS}(w_i|w_{i-1})=\left{ \begin{array}{l} \frac{C(w_{i-1},w_i)-d}{C(w_{i-1})}, & \text{if } C(w_{i-1}, w_i) > 0\

```
       \frac{d\times C(observed\; n-grams)}{C(unobserved\; n-
grams)}=\frac{\alpha(w_{i-1})}{|\{w_j:C(w_{i-1},w_j)=0\}|}, &
\text{otherwise}.
         \end{array}
```

\right. $$

```
def get_ads_prob(prev_word, word, unigram_counts, bigram_counts, d):
    """Compute a single word's smoothing probability using absolute
discounting
    Args:
        prev_word: str, the previous word
        word: str, current word
        unigram_counts: the Counter of the whole sentences using unigram
model
        bigram_counts: the Counter of the whole sentences using bigram
model
        d: float, the discounting rate
    """
    if bigram_counts[prev_word][word] > 0: # observed
        prob = (bigram_counts[prev_word][word] - d) /
unigram_counts[prev_word]
    else: # unobserved
        observed_prob = len([val for val in
bigram_counts[prev_word].values() if val > 0])
        unobserved_prob = len([val for val in
bigram_counts[prev_word].values() if val == 0])
        prob = d * observed_prob / unobserved_prob
    return prob
```

```
unigram_counts = {'alleged': 20}
bigram_counts = Counter({'alleged': Counter({'impropriety': 8, 'offense':
5, 'damage': 4, 'deficiencies': 2, 'outbreak': 1, 'infirmity':
0,'cephalopods': 0})})
```

```
for char in ['impropriety', 'offense', 'damage', 'deficiencies',
'outbreak', 'infirmity', 'cephalopods']:
    print(char + ':' + str(get_ads_prob('alleged',char, unigram_counts,
bigram_counts, 0.1)))
```

```
impropriety:0.395
offense:0.24500000000000002
damage:0.195
deficiencies:0.095
outbreak:0.045
infirmity:0.25
cephalopods:0.25
```

Backoff Smoothing

Katz backoff

我们可以通过使用 Absolute Discounting Smoothing 得到的结果看出，对于 unobserved word，它们的概率是一样的。然而真实情况中并非如此，因此我们引入 Katz backoff，对每个不同的 unobserved word 的 discounting 加入权重比。

我们可以考虑使用降低模型阶数来避免稀疏数据的问题。

对于 bigram 模型，Katz backoff 的公式可以表示为如下形式：

$$ P_{Katz}(w_i|w_{i-1})=\left{ \begin{array}{l} \frac{C(w_{i-1}, w_i)-d}{C(w_{i-1})}, & \text{if } C(w_{i-1}, w_i) > 0\

```
        \alpha(w_{i-1})\frac{P(w_i)}{\sum_{w_j:C(w_{i-1}, w_j)=0}P(w_j)},
& \text{otherwise}.
         \end{array}
```

\right. $$

- 其中对于 observed bi-gram（也就是$C(w_{i-1}, w_i)>0$）： Katz backoff 的平滑概率与 Absolute Smoothing 概率相同
- 而对于 unobserved bi-gram（也就是$C(w_{i-1}, w_i)=0$）： Katz backoff 通过退回一步（即回到 unigram），考虑当前单词w_i在 unigram 中的概率$P(w_i)$，分母表示为在上下文(corpus)中与单词w_{i-1}没有共同出现(co-occurrence)的单词的 unigram 概率之和
- e.g., 仍用以上例子，我们假设已知'infirmity'在文中出现的次数比'cephalopods'多，也就是$P('infirmity')>P('cephalopods')$，而剩余两个部分两者保持一致【其中$\alpha(w_{i-1})=\alpha('alleged')=5\times 0.1=0.5$， 分母都是$P('infirmity')+P('cephalopods')$】
- **Katz Backoff 存在的问题**
  - 由于我们只是单独考虑某个单词在上下文中出现的概率，而忽略了某些单词是否经常出现与不同的其他单词组合或是仅和少有不同其他单词组合
  - e.g., 在语料库(corpus)中知道$C('Francisco') > C('glasses')$，又假设 $C('reading'|'Francisco')=0,C('reading'|'glasses')=0$.
    - 此时当我们运用 Katz backoff 来对'reading'后的单词进行填词的时候，会偏向于选择填写'Francisco'
    - 也就是说**Katz Backoff Smooth 偏向于选择在低阶模型中拥有更高出现概率的单词**
    - **Katz Backoff Smooth is prone to predict the word with higher probability in lower order gram model.**

```
def katz_backoff_prob(prev_word, candidate_words, unigram_counts,
bigram_counts, d):
    """Compute the smoothing probability using Katz backoff smoothing
    Args:
        prev_word: str, the previous word
        candidate_words: list, store all possible candidate words
```

```
        unigram_counts: the Counter of the whole sentences using unigram
model
        bigram_counts: the Counter of the whole sentences using bigram
model
        d: float, the discounting rate
    """
    unobserved_words = []
    words_prob = {}
    for word in candidate_words:
        if bigram_counts[prev_word][word] > 0: # observed
            prob = (bigram_counts[prev_word][word] - d) /
unigram_counts[prev_word]
            words_prob[word] = prob
        else: # store unobserved words
            unobserved_words.append(word)
    for word in unobserved_words:
        observed_count = len(candidate_words) - len(unobserved_words)
        weighted = unigram_counts[word] / sum([unigram_counts[char] for
char in unobserved_words])
        prob = d * observed_count * weighted
        words_prob[word] = prob
    return words_prob
```

Kneser-Ney 平滑

用于解决 Katz Backoff 的问题，引入**多功能性（versatility）又称为延续概率（continuation probability）**

- High versatility: 与很多单独单词一起成对出现（co-occur with **a lot of** unique words）
  - e.g., glasses. (black glasses, man's glasses, ...)
- Low versatility：与仅有的单独单词一起出现（co-occur with **few** unique words）
  - e.g., Francisco. (San Francisco)

对于 bi-gram, Kneser-Ney Smoothing 的公式可以表示成如下：

$$ P_{KN}(w_i|w_{i-1})=\left{ \begin{array}{**lr**} \frac{C(w_{i-1}, w_i)-d}{C(w_{i-1})}, & \text{if } C(w_{i-1}, w_i) > 0\

```
        \alpha(w_{i-1})P_{cont}(w_i), & \text{otherwise}.
         \end{array}
```

\right. $$

- 对于 observed bigram, 仍然没有区别
- 对于 unobserved bigram，保留从 observed bigram 借来并进行 discount 的部分，其中$P_{cont}(w_i)$表示当前单词w_i的延续概率(continuation probability)
  - $$P_{cont}(w_i)=\frac{|{w_{i-1}:C(w_{i-1},w_i)>0}|}{\sum_{w'i|*in W_i}|{w'*{i-1}:C(w'_{i-1},w'_i)>0}|}$$， 其中W_i表示所有候选可能的单词的集合(candidaters)（e.g., glasses & Francisco 或对于上面 alleged 的例子是指那 7 个 words）

    - 分子表示与当前单词w_i在 corpus 中一起出现过的 unique 单词数量（即 previous word 的数量），即$P_{continuation}$
    - 分母表示所有与可能单词（candidaters) 在 corpus 中一起出现过的 unique 单词数量之和（也就是 candidaters 的所有 bigram 类型数量之和）
    - 注意分子分母都是集合的大小，也就是对于重复出现的 co-occur 单词只能算一次

- 此时对于 Low versatility word，分子会很小，就比如 Francisco，它的分子很有可能就是 1

RNN 语言模型

$$x_m\triangleq \phi_{w_m}$$ **embedding layer/lookup layer**

$$h_m=RNN(x_m, h_{m-1}) \triangleq g(\Theta h_{m-1}+x_m)$$ **Elman unit**

$$p(w_{m+1}|w_1, w_2, \dots, w_m)=\frac{\exp(O_{w_{m+1}}.h_m)}{\sum_{w'\in }\exp(O_{w'}.h_m)}$$

- ϕ是一个 word embeddings 矩阵，x_m 表示词w_m对应的 embedding matrix
- Θ 是 recurrence matrix
- g是非线性激活函数(tanh)用于压缩数据到[-1,1] ,**squashing function**
- O_{w_m}表示词w_m对应的输出值

评估语言模型 evaluate language models

focus on **Instrinsic evaluation**

Held-out likelihood

计算未参加训练的数据集**held-out data**的 log-likelihood。

$$l(w)=\sum_{m=1}^M\log p(w_m|w_{m-1},\dots, w_1)$$

w 表示 sentences, documents

- prefer to **higher** likelihoods

Perplexity

$$Perplex(w)=n^{-\frac{l(w)}{M}}$$

w 表示 sentences, documents，**M** refers to the **total number of tokens in the held-out data**

- Lower perplexities -> higher likelihoods, thus expect to **lower** perplexities
- In **unigram model**, assume a uniform, where $p(w_i)=\frac{1}{V}$, then
    - $$l(w) = \sum_{m=1}^M\log \frac{1}{V}=-M\log V$$
    - $Perplex(w)=n^{-\frac{1}{M}\times (-M\log_n V)}=n^{\log_n V}$$

  当 log 以 n 为底，则**Perplex(w) = V**，其中 V 表示 the **total number of tokens in the w**

- 理论取值范围：[1, $\infin$]
    - according to the range of $p(w_m|w_{m-1},\dots, w_1)$, that is from 0 to 1

- practical 取值范围：[1, V]
- 通常我们取 n=e

Text Classification

Tasks

- Topic classification
- Sentiment analysis
- Native-language identification
- Natural language inference
- Automatic fact-checking
- Paraphrase

流程

- Identify a task of interest
- Collect an appropriate corpus
- Carry out annotation
- Select features
- Choose a machine learning algorithm
- Train mdel and tune hyperparameters using held-out development data
- Repeat earlier steps as needed
- Train final model
- Evaluate model on held-out test data

Algorithms

Naive Bayes

- Finds the class with the highest likelihood under Bayes lay

pros

- Fast to train and classify
- robust, low-variance → good for low data situations
- optimal classifier if independence assumption is correct
- extremely simple to implement

cons

- Independence assumption rarely holds
    - e.g., the scenoria in NLP, where considering prefix.
        - As usual, $$P(word='unfit',prefix='un-'|y)=P(prefix='un-',word='unfit'|y)P(word='unfit'|y)=1\times P(word='unfit'|y)$$
        - if use NB, $$P(word='unfit',prefix='un-'|y)=P(word='unfit'|y)P(prefix='un-'|y)$$
- low accuracy compared to similar methods in most situations

- smoothing required for unseen class/feature combinations

Logistic Regression

Cons

‣ 训练速度慢 Slow to train;

‣ 需要标准化或归一化处理 Feature scaling needed

‣ 需要大量的数据 Requires a lot of data to work well in practice

‣ 容易导致 overfitting Choosing regularisation strategy is important since overfitting is a big problem

Support Vector Machines

- Finds hyperplane which separates the training data with maximum margin
- be popular for NLP, mainly because it works well with huge feature sets and NLP problems often involve large feature sets

Pros

• 快且准 Fast and accurate linear classifier

• 能够支持非线性 kernel Can do non-linearity with kernel trick

• 在大特征集中表现良好 Works well with huge feature sets

Cons

• 多分类的效果不佳 Multiclass classification awkward

• 需要特征标准化 Feature scaling needed

• 难处理类不平衡问题 Deals poorly with class imbalances

• 可解释性差 Interpretability

K-Nearest Neighbour

- Classify based on majority class of *k*-nearest training examples in feature space

Pros

• 简单且有效 Simple but surprisingly effective

• **不需要训练** No training required

• 适用于多分类 Inherently multiclass

• 在无穷大的数据集中能够找到 optimal Optimal classifier with infinite data

Cons

• 难选择 k Have to select *k*

• 难处理类不平衡问题 Issues with imbalanced classes

• 速度慢 Often slow (for finding the neighbours)

• 特征选取难 Features must be selected carefully

Decision tree

Pros

• 训练测试速度快 Fast to build and test

• 不需要特征归一化 Feature scaling irrelevant

• 在小特征集中效果好 Good for small feature sets

• 能处理非线性可分的情况 Handles non-linearly-separable problems

Cons

• 可解释性差 In practice, not that interpretable

• 子树集合高度冗余 Highly redundant sub-trees

• 在大特征集中效果不佳 Not competitive for large feature sets

Random Forests

Pros

• Usually more accurate and more robust than decision trees

• Great classifier for medium feature sets

• Training easily parallelised

Cons:

• Interpretability

• Slow with large feature sets

Neural Networks

Pros

• Extremely powerful, dominant method in NLP and vision

• Little feature engineering

Cons

• Not an off-the-shelf classifier

• Many hyper-parameters, difficult to optimise

• Slow to train

• Prone to overfitting

参数调优 Hyper-parameter Tuning

Dataset for tuning

- development set
- not the training set or the test set
- k-fold cross-validation

Grid Search

for multiple hyper-parameters

Part of Speech Tagging

常用词性标记集

Penn Treebank corpus,

Brown

构建词性标注模型

- 特征构建
    - Word itself
    - Lowercased word
    - prefixes
    - suffixes
    - capitalization
    - word shapes
    -