英文词干提取对布尔检索系统性能影响相关研究

霍超凡

2019 年 12 月 10 日

摘要

布尔检索模型作为最经典的检索模型在经历诸如空间向量模型、概率模型、主题模型等不同检索模型的变革后，生命力仍然经久不衰，它将文档与查询词的匹配规约到词与词之间的相互匹配，它认为文档中涵盖查询词的个数越多，文档与查询词越相关，这种简单的思想使检索模型在一些精确查找的任务中已经表现出独特的优势。然而由于查询词形式的多变，基于精确查找的布尔检索模型往往会表现出召回率较低的现象，为了增加模型的鲁棒性，它往往会结合一些其它文本的预处理操作，其中词干提取就是一个关键的操作。英文单词在实际语料库中的出现形式往往不是原型，而是被添加了一些后缀来表示时态、语态等语法信息，词干提取处理的目的是尽可能正确地通过去掉这些后缀来将这些变化形式归为统一。词干提取加入布尔检索模型后缓解检索模型对单词的变化形式的敏感，由此提高了召回率。然而如何正确的从单词中提取出词干却是一个棘手的问题，单词变化形式的多样，很难从中提取出一个适用于所有单词的规则体系，由此诞生了很多算法。本次实验聚焦于英文单词的词干提取，对比了不同算法，并且通过一些准则来衡量这些算法，大量的实验表明传统的Porter算法仍然具有无与伦比的优势，但是这仅仅限于对于英文的词干提取，对于其他语言来说，人工手动提取规则过于昂贵，基于统计的算法成为我们的首选，基于统计算法中SNG算法由于其巧妙的设计，它在实际检索环境下能够在保持精度的同时提高召回率。

1 概述

英文单词通过后缀来区分单词的语法信息，而单词的前半部分表达单词的实际意义，英文词干提取基于这种构造方式，通过去除后缀使表示同一意义的表现为不同形式的单词归为统一。Porter 词干提取算法[14]是一种基于规则的提取算法，它通过构造一系列规则将后缀去除，从而达到提取出词干的目的。Lovins算法[2]与之类似，它构造了一个后缀词表，查找单词的后缀是否出现在后缀词表中，如果出现则将其去除。以Porter为代表的基于规则的词干提取算法无论是从效率还是性能方面均具有较大的优势，但是这些算法需要人为的提取规则，算法不用迁移到其它语言中。

之后很多人为了克服这个缺陷，提出了基于统计的算法，这些算法需要一个庞大的语料库，从语料库中自动提取出规则，这些算法可以适用于不同的语言。本次实验从中选取了几个算法，这些算法包括HMM词干提取[12]、SNG算法[11]、YASS词干提取[10]。HMM词干提取算法使用隐马尔科夫模型将单词中字符划分到前缀集和后缀集，位于后缀的部分视为后缀，具体来说，单词中字符是已知的被观测序列，单词中的字符位于前缀集合还是后缀集合是隐藏状态，通过计算每一种情况的概率，选取最大的概率作为结果，将位于后缀集合的字符去掉，只保留位于前缀集和的字符作为结果。SNG算法将单词中最具有“区分度”的n-gram作为词干，实验表明相同意义的单词中的n-gram 往往具有独特性，它们往往具有十分相近的n-gram组，而由后缀组成的n-gram 由于频繁在不同单词中出现没有区分度而不会作为最终结果，这种算法可以将形式类似的单词联系在一起。YASS算法利

用相近意义的单词往往具有相同的前缀这个原理，构造了四种不同的公式来计算不同单词之间的距离，最后通过聚类算法将相同意义的单词聚为一类，将类内所有单词的公共前缀作为这些单词的词干。另外基于语料库的算法 [16][13] 也利用了聚类的思想，它们认为意义相近的单词它们会频繁出现在同一篇文章并且出现的位置十分接近，它们使用这样的特点从语料库中计算出单词的关联性，找出关联度较大的单词，并将关联度较大并且具有公共前缀的单词作为一类，由于时间关系仅仅分析了其中的原理，更多的细节并没有讨论。在第二节中，将会具体介绍这些算法。

这些算法思路各异，需要制定一系列统一的标准测评算法，在测评中需要注意的一点的是，我们词干提取的最终目的是为了提高检索系统的性能，评价不同算法时需要将其置于实际的问题中，在信息检索方面我们并不关注算法提取的词干是否正确，而是这个算法能否在提高召回率的同时保持精度。受[8]的启发，实验中使用了不同的准则来衡量这些算法的性能。这些准则将会在第三节中介绍。

本次实验对这些算法从不同角度进行了测评，进行了大量的实验，具体的实验细节将在第四节中介绍。通过大量的实验数据，从中可以得到有用的结论，这些结论最后在第五节中给出。本次实验，并不是我独立完成，而是参考了大量的文献，关于文献细节将会在第六节给出。

2 算法细节

2.1 基于规则的算法

基于规则的词干提取算法是最初的研究热点，不同人制定了不同的规则，其中最有名的当属Porter算法，至今很多系统仍然用Porter算法进行词干提取。Porter算法将英文单词按照元音字母和辅音字母划分成两类，记C为辅音集合、V为元音集合，任何字母都可以表示成为$[C](VC)^m[V]$，即任何单词都可以划分成元音、辅音交叉排列的形式，既可以以辅音开头，也可以以元音结尾，元音在英文单词中起着重要作用，元音是一个实词不可缺少的部分，Porter 这样将单词记为这样的形式是为了区分单词的长度，单词中字母的个数不足以成为我们判断一个单词需要进行词干提取操作的依据，Porter结合英文构词法将单词按照m值的大小划分成几类，m 限制了单词的长度。Porter词干提取算法分为5个阶段，在每个阶段内针对不同的情况制定不同的规则，为了防止提取过度，每条规则对m进行限制。另外Porter还考虑到后缀的顺序，在不同阶段去除不同位置的后缀。Porter在文章[14]中强调，词干提取的最终目的是为了提高检索系统的性能，规则很难涵盖所有的情况，一些规则可能会违背一些单词，但是这些单词可能在语料库中极少出现，并不影响算法的大体性能。Lovins算法相对简单，它构造一个后缀词表，词干提取的过程相当于查表的过程，存在后缀则去除。

2.2 基于统计的算法

基于统计的算法结合单词的构造规则，能够从语料库中学习到什么是后缀，哪些单词会具有相同的词干，这些算法能够较好的适应于其它语言，不需要人为提取规则。本次实验仅讨论其中的三种算法。

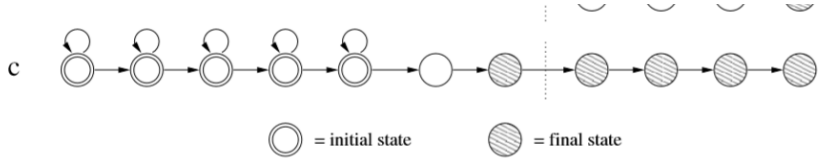


图 1: 用于词干提取的HMM模型(取自[12])

2.2.1 HMM算法

HMM算法通过计算概率来判断应该在哪里截取单词，它将单词中的字符序列视为已知观测状态，而这些字符属于前缀集合还是后缀集合是隐藏状态，我们需要根据已知观测状态得到隐藏状态。隐藏状态分为两类，一类是前缀状态，另外一类是后缀状态，一个单词只能从前缀状态开始，最后在后缀状态结束，一个单词可能没有后缀状态，但是必须存在前缀状态。HMM模型的拓扑结构如图1所示，前缀状态可以维持当前状态，后缀状态必须转移到下一个状态，只能从前缀状态转移到后缀状态，而不能从后缀状态转移到前缀状态。训练HMM模型需要使用MP算法，训练时需要注意模型初始状态的构造和参数更新的时机，更多细节这里不再过多介绍。

HMM算法能够学习出频繁出现在后缀的观测状态，以teaching为例，在语料库中以ing结尾的单词非常多，同时又存在没有ing形式的原型teach，也存在teacher，teached等其他其它形式，经过统计后ing出现在后缀状态的概率会非常大，而teach仅仅出现在前缀又对其它单词teaches，teacher有利，所以模型能够学习到ing，es，er等作为后缀的概率较大。HMM模型对语料库的要求十分苛刻，语料库要大量包含原型，同时又必须包含不同单词的后缀形式。

2.2.2 SNG算法

SNG算法（Single N-gram Stemmer）将单词划分成若干gram，从中选择最具有“区分度”的gram作为这个单词的伪词干。一个具有区分度的gram较少出现在其它单词中，而像后缀频繁出现在不同单词中，没有区分度，它不会作为一个单词的伪词干。算法的大体过程是这样的，将每个单词分成若干gram，统计每个gram在其它单词出现次数来计算这个gram的’idf’，‘idf’较高的gram作为这个单词的伪词干。

2.2.3 YASS算法

YASS算法利用单词的构造规律，一个单词的前缀表达单词的含义，后缀表示单词的形式，这个算法根据这条规律指定了四条公式来衡量不同单词的相似程度，具有相同前缀单词具有较高的相似程度，而两个单词中位于末端相同字符对单词的相似度贡献较少。这个算法统计两个单词公共字符的个数和出现的位置，公共字符对单词相似度的贡献不同，位于词首的公共字符具有最高的贡献度，字符的贡献度随着字符出现的位置的顺序而递减。算法最后通过聚类算法将相似程度较高的单词聚为一类，并且最终构造一个等价类查找表将单词映射到词干。

2.3 基于语料库的算法

基于语料库的算法可以较容易找出相关的单词，一个单词越相关，它们越有可能存在于同一篇文章中，或者越有可能存在于同一篇文章的同一个句子，通过统计两个单词出现在同一篇文章中的比例，来确定两个单词的相似程度，这个算法得到更多的不是具有相同词干的单词，而是相关度较高的单词，比如a lot of、a pair of之类的单词a和of出现在一起的概率非常大，但是它们并不表示同一含义。这种算法可以结合到其它的词干提取算法中微调结果，首先使用传统的词干提取算法计算两个单词的相似度，之后结合基于语料库的算法得到的结果，修正结果。由于时间原因这个算法并没有实现。

3 算法评价准则

受[8]的启发，stem算法评价标准主要按以下三个部分进行，首先是算法的时间效率，算法的时间效率反映算法的计算复杂度。其次是算法的stem效果，词干提取是否准确，词干提取过度还是轻微。最后是算法在布尔检索系统中的性能。

算法的时间效率可以使用每秒钟算法处理了多少token计算，也可以通过每一个token消耗的时间计算，在实验中我们选用后者。关于衡量词干提取的效果，我们使用两个指数：OI和UI，OI（Overstemming Index）是提取结果大于标准词干所占的比例，同理UI（Understemming Index）表示结果词干长度小于标准词干长度的比例，准确提取词干的比例使用RI（Rightstemming Index），同时使用AOL（Average Overstemming Length）表示平均过度提取的长度，使用AUL （Average Understemming Length）衡量欠提取的程度。布尔检索系统中我们关注我们检索的结果中正确的比例和漏掉的比例，使用精度P和R值表示。

4 实验

4.1 实验环境

词干提取算法全部由Java语言编写，词干提取过程中全部在Java环境中展开，测评算法结果的程序使用Python语言编写，所有数据后部处理在Python环境中展开。

4.2 测试数据集

本次实验选用的数据集是BNC（British National Corpus）[3]中的关于英文子数据集。语料库十分庞大，这个数据集收集了20世纪末来自不同领域的书面语和口头语。数据集按照语料的话题来源可以分成8个子集，各个子集的详细情况如表 1。

表 1: BNC数据相关信息

领域	ACPROSE	CONVRSN	FICTION	NEWS	NONAC	OTHERPUB	OTHERSP	UNPUB
字典集大小	190,748	34,315	130,540	152,642	282,768	259,768	50,996	87,392
大小(MB)	109	24	100	62	164	120	37	30

4.3 实验具体情况

从BNC中抽取出单词集和与之对应的标准词干集，为了方便叙述之后使用gold来代表标准词干。使用不同的算法对单词集中每一个单词词干提取得到词干集，之后使用stem来

代表算法提取的词干。将stem和与之对应的gold一一比较计算OI、UI等其它指数。算法可能将错误的单词划分到同一个词干中，亦可能没有将正确的单词划分到同一个词干中，我们需要引入检索词的精度和召回率，对于一个查询词word，我们希望查找到结果能够包含word 的各种变化形式，即这个word的gold所对应的其它所有word，而实际上我们结果词干提取后得到的结果是这个word的stem所对应的其它word，两者之交为我们正确检索出的单词，记为TP，前者减后者为TN，后者减前者为FP，通过TP，TN，FP计算这个检索词的召回率和精度。为了衡量算法在实际布尔检索系统的性能，设计另外的实验，检测词干提取对布尔检索系统的影响，不需要真正的系统中测试，按照[8]的方法，将文档划分成若干个句子，统计各个word出现的句子个数和stem出现的句子个数，对于查询词word，结果词干提取所得到的结果是这个查询词结果词干提取所得到的stem所对应的其它单词所出现的句子，而实际上我们想要得到的这个单词各种变化形式所出现的句子。两者之交除以前者为查询词word在实际语料库的精度，两者之交除以后者是召回率。

4.4 实验结果

结果一：算法的时间效率 在提取语料库中所有单词的词干时，计算统计其token的个数和运行总时间，两者相除得到算法处理一个token所消耗的时间。得到的结果见表2，YASS算法处理一个token的时间为查找词典的时间，具有最高的时间效率，而HMM 算法需要使用维特比算法计算可能状态转移路径的概率，计算量较大，效率较低。SNG和YASS同为查表，但是SNG却明显比YASS所消耗的时间长，这可能与字典结构有关，具体细节没有深究。从大体上看出HMM的其它四种算法时间基本相当。

表 2: 不同算法运行时间对比

算法	Porter	Lovins	HMM-5-5	SNG	YASS
时间(μ s/token)	3.257	3.111	5.874	3.788	2.839

结果二：词干提取算法对不同语料库的敏感程度 我分别测试了各种不同算法在BNC各种子语料库上的性能，不同算法在不同语料库中没有明显的变化，注意到没有经过词干提取NOSTEM变化较为剧烈，而其它算法变化轻微，这说明这些算法对语料库敏感程度较低。

结果三：HMM算法中所使用的模型对结果的影响 我调节HMM模型中前缀状态个数和后缀状态个数得到不同HMM模型，使用HMM-5-4表示具有5个前缀状态和4个后缀状态的HMM模型。我在测试HMM-5-3模型时，结果训练之后得到的参数全部变为NAN，HMM-5-3的结果应该和NOSTEM效果相同，可是它们的召回率却不相同，而我使用的相同的检测程序测试这些算法，应该会得到完全相同的结果，我没有继续进一步找出问题所在，可能在画图时某一个参数弄错了。观察HMM-5-4、HMM-5-5和HMM-5-6，当增加后缀状态数量时，性能不变，这说明后缀状态不是HMM算法性能的决定因素，观察HMM-4-5、HMM-5-5和HMM-6-5，当前缀状态数增多时，精度逐渐提升，召回率略微下降，这说明当提高模型的前缀状态数目时，可以大幅度的提高模型的性能，这个规律是我在全部测试完全部算法所得到的结果，应该测试的HMM-7-5等模型没有测试。

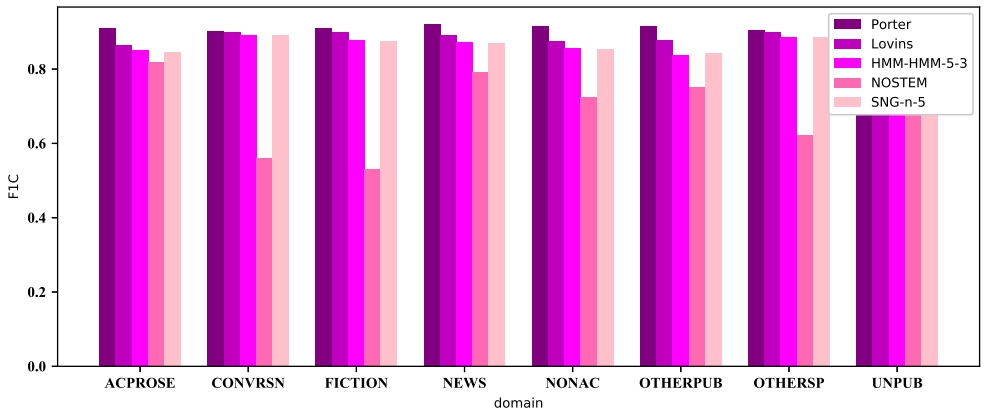


图 2: 不同数据集下算法性能对比

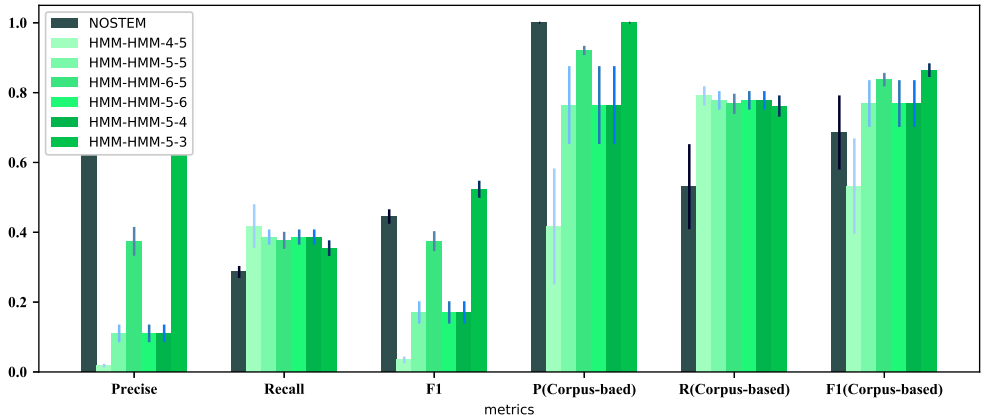
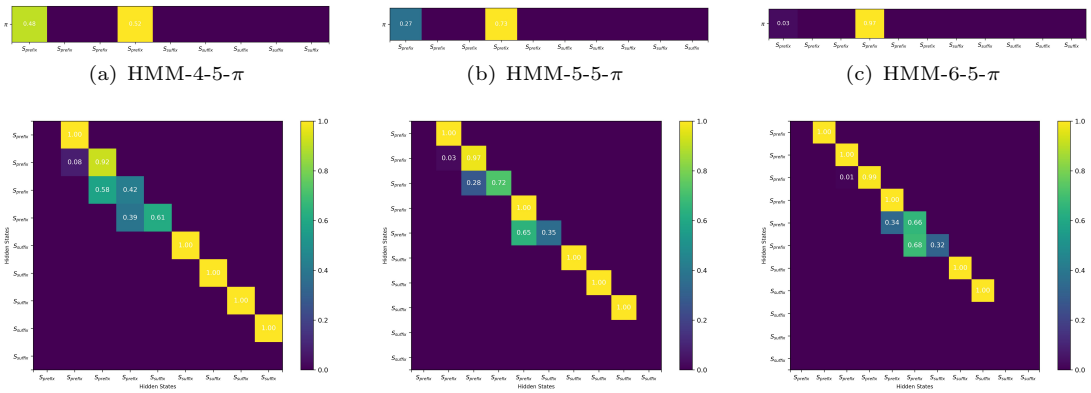


图 3: 不同HMM模型对算法性能的影响

为什么增加前缀状态个数能够大幅度提高模型的性能？我观察模型所得到的参数发现，当提高前缀状态数目时，会使模型中前缀路径延长，更多的单词前缀和后缀的分裂点后移，所去掉的后缀变短，单词越发接近原型，所以精度会逐渐接近于NOSTEM。



(a) HMM-4-5-π (b) HMM-5-5-π (c) HMM-6-5-π

(d) HMM-4-5-TP (e) HMM-5-5-TP (f) HMM-6-5-TP

图 4: HMM模型内部参数的可视化展示，从左往右依次是HMM-4-5、HMM-5-5、HMM-6-5，顶部是初始状态概率 π，底部表示状态转移概率，S_{prefix} 表示前缀隐藏状态，S_{suffix}表示后缀隐藏状态。我们可以发现，在三种模型中第四个前缀隐藏状态具有最高的初始概率，很多单词都将第四状态作为自己的初始状态。当前缀状态数目增多后前缀和后缀分裂点远离第四状态，这使得单词中更多的字符处于前缀状态，从而使得词干提取的效果变弱。

图5是HMM模型的发射概率，从中我们可以发现各个字符在单词中出现的统计数据，比如较多的单词以e，f，t，a，i开头，以e,s,y,d结尾。字母w,j,q,k,x较少出现在单词中，并且z出现在单词中的概率极低，j，q出现基本只出现在单词开头。

结果四：YASS中单词相似度计算方式对算法的影响 我测试了YASS四种距离公式和每一种距离计算方式的聚类过程中不同于阈值带来的影响，得到图 6，从图中可以看到，不同距离计算方式对算法影响的差异很小，对算法影响最大的是聚类过程中阈值的选择。大阈值使召回率提高，而小阈值却可以提高精度。

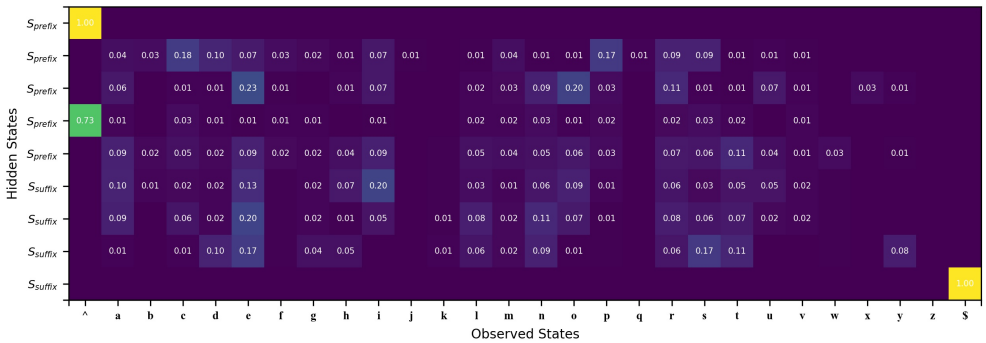
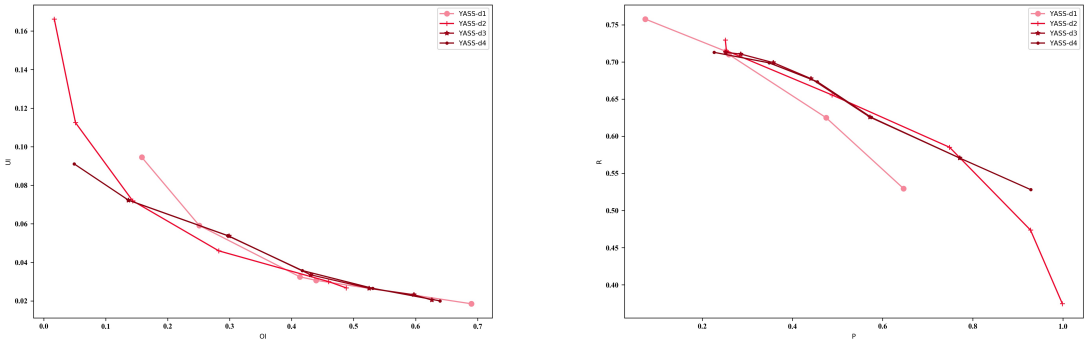


图 5: HMM-5-4模型的发射概率，横坐标表示观测状态，上三角表示词首，dollar符表示词尾，纵坐标表示隐藏状态。后缀状态发射字符e,i,s,n,t,r,l等的概率较大，这些字母频繁出现后缀中，而q,j,x,f字符不会出现在后缀状态中。



(a) YASS-OUI (b) YASS-PR

图 6: 图6(b)是YASS算法P-R曲线，该图展示了YASS算法中使用不同计算方式和按照不同的阈值进行聚类对算法结果的影响。图6(a)YASS的OI-UI曲线，该图展示了YASS 内部参数变化时，词干提取的效果。每一种距离计算方式下，设置6种不同阈值，测试每一个阈值的算法性能，当阈值逐渐增大时，单词与单词之间相似度要求放低，更多的单词被聚为一类，算法的精度提高，但是召回率降低。

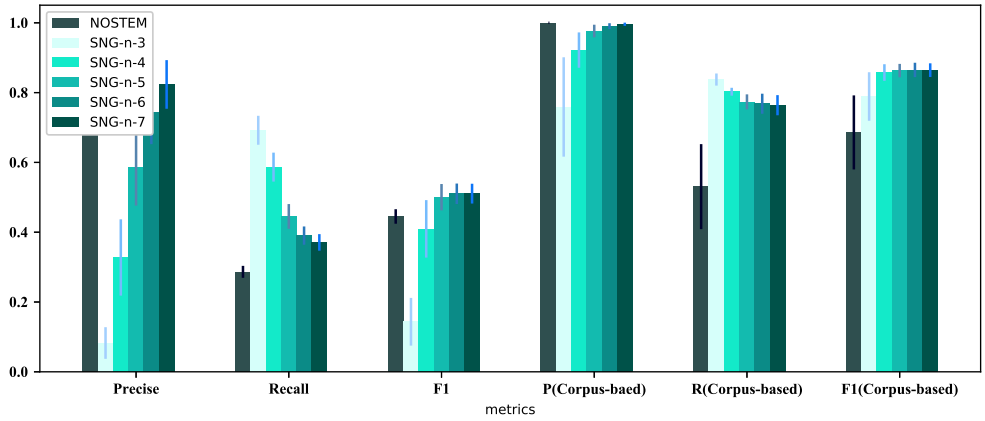


图 7: 使用不同长度gram提取词干，算法性能比较。

结果五：SNG算法中参数N对算法的影响 通过将单词切分成不同长度的gram，计算每一种长度下的算法的性能，所得到的结果如图7所示。使用越长的gram，提取的结果越接近于单词原型，精度越高，但是召回率会降低，值得注意的是这个算法虽然对于单个查询词来说，效果会随着gram长度的变化而显著改变，而在时间语料库的测试中，却保持惊人的稳定性，尤其值得注意的是算法在保持精度的同时又能够提高召回率，这得益于算法的巧妙设计，算法在选择单词的gram时，总是选择在其他单词中出现次数最少的gram，所以能够保持精度，同时，具有相同词干的单词能够较大的概率被提取出相同的gram，所以算法又可以提高召回率，这个算法符合我们设计词干提取算法的基本原则，即在尽可能维持精度的同时提高召回率。

结果六：不同算法词干提取的效果对比 单词提取是否正确虽然不再我们关注的范围内，但是单词提交的结果确会影响检索系统的性能，单词过度提取会导致精度下降而单词的词干提取过轻会导致召回率下降。从图8可以发现Porter和YASS被正确提取的单词所占的比例较大，HMM-5-4明显提取过度。

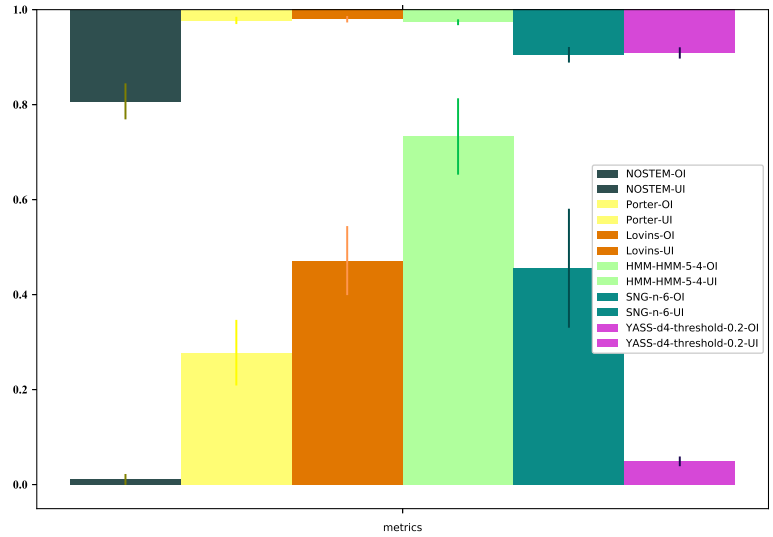


图 8: 不同算法词干提取效果，底部是OI，顶部UI，中间空白区域表示正确词干提取的单词所占的比例。

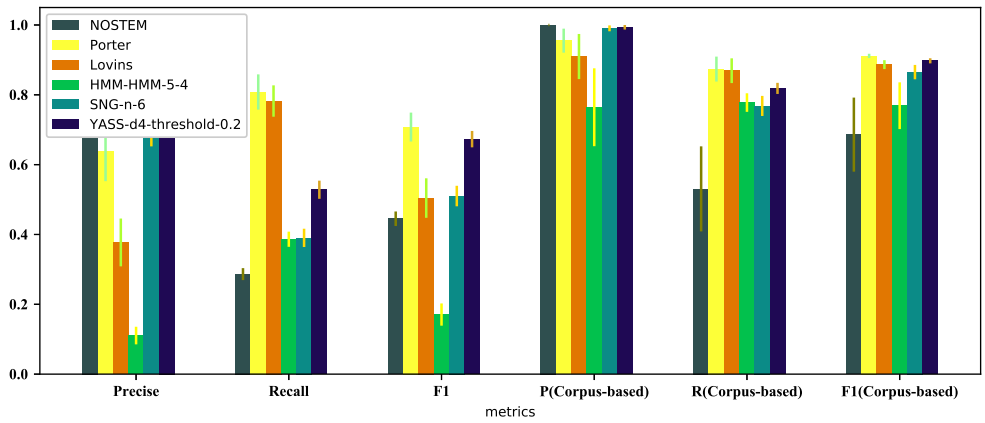


图 9: 不同算法性能对比。

结果七：不同算法在实际布尔检索系统中的性能 以上讨论的各大算法均是以牺牲精度来换取召回率，从F1值上看，传统方法Porter算法仍然是最好的词干提取算法，基于统计的算法中SNG算法在实际检索环境中能够和Porter持平，HMM算法可是是由于模型参数的调节和训练集的缘故，它的性能并不好。

结果八：一些失败的例子 BNC给出的gold并不是我们所认为的词干，这里的gold是在不改变词性的情况下删去一些细枝末节，对于名词来说着重将复数变单数，对于动词来说着重将过去式和过去分词等时态形式变为原型，对于形容词，将比较级变为原型，而对于international来说，它本身就是gold，不需要在进行词干提取，这里着重将词性还原，效果会与预想的不同。

表 3: 一些失败的例子

	Porter		Lovins		HMM-6-5		SNG-n-6		YASS-d4	
	stem	gold	stem	gold	stem	gold	stem	gold	stem	gold
欠提取	hottest	hot	hottest	hot	farthes	far	upped	up	hottest	hot
	biggest	big	biggest	big	uppin	up	bing	b	going	go
	farther	far	furthest	far					putting	put
	maddest	mad	going	go						
过提取	intern	international	nat	national	li	light	ogatio	interrogation		
	rel	relatively	op	openly			adole	adolescence		
	oper	operation	tang	tangentially			ruzi	ruzimatov		
	educ	education	ev	every			ghtenm	enlightenment		

5 结论

本次实验对比了五种算法，并从各个方面对算法进行衡量，词干提取对于信息检索来说是为了将单词的多变化形式归为统一，提高实际检索结果的召回率。一个单词的词干的正确形式很难定义，对于自然语言处理来说在词形还原时不能改变单词的词性，但是信息检索方面，两个意义相近的单词可以具有相同的词干。受数据集的影响，本次实验认为一个单词提取过程中词性不能发生变化，检索一个形容词，不希望出现动词之类的形式，实验根据BNC所给出的单词的原型作为词干提取的正确形式。所得到的结论既有片面性，可能不会适用于其他情形。经过测试了大量的实验数据，我们发现传统的Porter算法仍然既有无与伦比的的性能，但是这仅仅限于英文词干提取，Porter算法不具有普适性，之后诞生了很多的其它基于统计的词干提取算法，这些算法从一个庞大的语料库中自动提取出规则。在测试HMM算法时，我们发现前缀状态集合的个数成为影响模型性能的决定性因素，YASS 算法中的几个衡量单词间距离的公式，都是基于这样的观察，英文单词中靠前的字母决定单词的语义，后单词末端的字母决定单词的形式，所设计出的公式对单词前端相似的字符赋予权重较大，而对后端的字符赋予较小的权重，公式虽然形式上存在差异，都是实际效果上并不是决定因素，影响算法的聚类算法中的阈值选择，阈值选的过大，或导致无关的单词聚为一类，提高召回率的同时降低了精度，反之，阈值过小，精度较高但是召回率较低；SNG算法在实际检索过程中在保持精度的同时能够提高召回率，这得益于算法总是选择最具有区分度的gram作为伪词干。以上各个算法在BNC数据集上的评价标准，均表现为过度提取，提高了召回率，这满足我们设计词干提取的首要要求。Porter算法不具有普适性，基于统计的方法称为学术界研究的重点，本次实验中SNG和YASS的算法表现较好，能够最大程度的不损失精度的前提下，提高召回率，但是YASS的算法存在一些不好的方面，首先是YASS算法的性能受阈值的影响，我们通常喜欢一些不需要调参的算法，其次是聚类过程的计算量非常大，语料库中的单词数是非常庞大的，这对算法的运行空间和时间均有苛刻要求，而SNG算法与之不同，上述实验结果表现gram的长度在实际检索中对结果的影响并不大，更重要的算法设计巧妙，符合我们检索的最终目的，能够检索到最具有区分度的内容。

6 文献评注

本次实验并不是我独立完成的，而是参考大量的其它资料。[9]是关于stemmer 算法的总述，我从这篇文章挖掘出[14][12] [10] [13] [16]，[14]是Porter 算法的文献，[1]是Porter 算法的文章，里面有各种语言版本的Porter算法的实现。[2] 是关于Lovins 算法的网站，里面

有Lovins 算法的实现，关于经典算法的实现均是照搬[1][2]上面的经典代码，我参考[6][7]将这两个算法集成到Lucence 系统中，[12]是介绍将HMM 应用到词干提取方面的文章，[15]这是一篇经典的介绍HMM 模型的文章，里面介绍了各种HMM 模型和训练HMM 的算法。[10] 是关于YASS 的文章，[5]是其的Java 实现，里面使用到聚类算法，使用多线程编程。[13][13]是两个基于语料库算法方面文章，[13]的方法更加清晰，而[16]中计算两个单词之间的相关度的公式比较模糊。[11] 是一篇介绍SNG 算法文章，篇幅短小但意义深刻。[8]给出了评价stemmer 的各种指标， [4] 是它的实现，我参考[4]代码，并做了些改动使其适用于本次实验，[4]使用Java 语言编写的，而本次实验的测评Stemmer 的相关代码是使用Python 语言编写的，本次所使用的语料库来自于[3]。另外，在实现HMM 算法是所使用的模型的代码参考别人，可是出处找不到了。

参考文献

[1] The porter stemming algorithm. `https://tartarus.org/martin/PorterStemmer/`. Accessed December 08,2019.

[2] The lovins stemming algorithm. `http://snowball.tartarus.org/algorithms/lovins/stemmer.html`. Accessed December 08,2019.

[3] British national corpus. `www.natcorp.ox.ac.uk/`. Accessed December 08,2019.

[4] stemmereval-master. `https://github.com/endredy/stemmerEval`. Accessed December 08,2019.

[5] Yass-master. `https://github.com/GiacomoManzoli/YASS`. Accessed December 08,2019.

[6] stemmer-sk-master. `https://github.com/essential-data/stemmer-sk`. Accessed December 08,2019.

[7] lucene-arabic-analyzer. `https://github.com/msarhan/lucene-arabic-analyzer`. Accessed December 08,2019.

[8] Istvá n Endré dy. Corpus based evaluation of stemmers. 05 2015.

[9] Muhammad Haroon. Comparative analysis of stemming algorithms for web text mining. *International Journal of Modern Education and Computer Science*, 10:20–25, 09 2018.

[10] Prasenjit Majumder, Mandar Mitra, Swapan Parui, Gobinda Kole, Pabitra Mitra, and Kalyankumar Datta. Yass: Yet another suffix stripper. *ACM Trans. Inf. Syst.*, 25, 10 2007.

[11] James Mayfield and Paul Mcnamee. Single n-gram stemming. pages 415–416, 01 2003.

[12] Massimo Melucci and Nicola Orio. A novel method for stemmer generation based on hidden markov models. pages 131–138, 01 2003.

[13] Jiaul Paik, Dipasree Pal, and Swapan Parui. A novel corpus-based stemming algorithm using co-occurrence statistics. pages 863–872, 01 2011.

[14] M.F. Porter. An algoritm for suffix stripping. *Program*, 14:130–137, 07 2006.

[15] Lawrence Rabiner. A tutorial on hidden markov models and selected applications. *Proceedings of The IEEE - PIEEE*, 77, 01 1989.

[16] Jinxi Xu and W. Croft. Corpus-based stemming using co-occurrence of word variants. 10 1996.