N-gram

BLEU 采用一种N-gram的匹配规则，原理比较简单，就是比较译文和参考译文之间n组词的相似的一个占比。

例如：

原文：今天天气不错

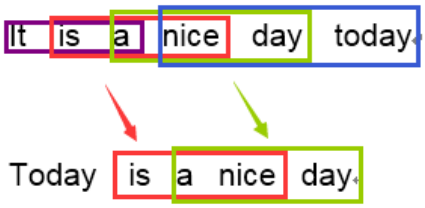
机器译文：It is a nice day today

人工译文：Today is a nice day

如果用1-gram匹配的话：

可以看到机器译文一共6个词，有5个词语都命中的了参考译文，那么它1-gram的匹配度为 5/6

我们再以3-gram举例：



可以看到机器译文一共可以分为四个3-gram的词组，其中有两个可以命中参考译文，那么它3-gram的匹配度为 2/4

依次类推，我们可以很容易实现一个程序来遍历计算N-gram的一个匹配度。一般来说1-gram的结果代表了文中有多少个词被单独翻译出来了，因此它反映的是这篇译文的忠实度；而当我们计算2-gram以上时，更多时候结果反映的是译文的流畅度，值越高文章的可读性就越好。

召回率

上面所说的方法比较好理解，也比较好实现，但是没有考虑到召回率，举一个非常简单的例子说明：

原文：猫站在地上

机器译文：the the the the

人工译文：The cat is standing on the ground

在计算1-gram的时候，the 都出现在译文中，因此匹配度为4/4 ，但是很明显 the 在人工译文中最多出现的次数只有2次，因此BLEU算法修正了这个值的算法，首先会计算该n-gram在译文中可能出现的最大次数：

$$Count_{clip} = min(Count, Max\_Ref\_Count)$$

Count是N-gram在机器翻译译文中的出现次数，Max_Ref_Count是该N-gram在一个参考译文中最大的出现次数，最终统计结果取两者中的较小值。然后在把这个匹配结果除以机器翻译译文的N-gram个数。因此对于上面的例子来说，修正后的1-gram的统计结果就是2/4。

我们将整个要处理的将机器翻译的句子表示为Ci，标准答案表示为 Si=si1,...sim（m表示有m个参考答案）

n-grams表示n个单词长度的词组集合，令Wk第k个n-gram

比如这样的一句话，”I come from china”，*第1个2-gram为：I come; 第2个2-gram为：come from; 第3个2-gram为：from china;*

Hk(Ci) 表示Wk翻译选译文Ci中出现的次数

Hk(Sij) 表示Wk在标准答案Sij中出现的次数

综上所述各阶N-gram的精度都可以按照下面这个公式计算：

$$P_n = \frac{\sum_i \sum_k min(h_k(c_i), max_{j \in m} h_k(s_{ij}))}{\sum_i \sum_k min(h_k(c_i))}$$

maxi∈mhk(sij)表示某n-gram在多条标准答案中出现最多的次数

∑i∑kmin(hk(ci),maxj∈mhk(sij))表示取n-gram在翻译译文和标准答案中出现的最小次数

惩罚因子

上面的算法已经足够可以有效的翻译评估了，然而N-gram的匹配度可能会随着句子长度的变短而变好，因此会存在这样一个问题：一个翻译引擎只翻译出了句子中部分句子且翻译的比较准确，那么它的匹配度依然会很高。为了避免这种评分的偏向性，BLEU在最后的评分结果中引入了长度惩罚因子(Brevity Penalty)。

$$BP = \begin{cases} 1 & if \quad l_c > ls \\ e^{1-\frac{ls}{lc}} & if \quad l_c <= l_s \end{cases}$$

BP的计算公式如上。lc代表表示机器翻译译文的长度，ls表示参考答案的有效长度，当存在多个参考译文时，选取和翻译译文最接近的长度。当翻译译文长度大于参考译文的长度时，惩罚系数为1，意味着不惩罚，只有机器翻译译文长度小于参考答案才会计算惩罚因子。

BLEU

由于各N-gram统计量的精度随着阶数的升高而呈指数形式递减，所以为了平衡各阶统计量的作用，对其采用几何平均形式求平均值然后加权，再乘以长度惩罚因子，得到最后的评价公式：

$$BLEU = BP \times exp(\sum_{n=1}^{N} Wn log P_n)$$

BLEU的原型系统采用的是均匀加权，即Wn=1/N 。N的上限取值为4，即最多只统计4-gram的精度。

实例

译文（Candidate）

```
Going to play basketball this afternoon ?
```

参考答案（Reference）

```
Going to play basketball in the afternoon ?
```

译文gram长度：7　　参考答案gram长度：8

先看1-gram，除了this这个单词没有命中，其他都命中了，因此：

P1 = 6/7 = 0.85714...

其他gram以此类推：

P2 = 4/6 = 0.6666..

P3 = 2/5 = 0.4

P4 = 1/4 = 0.25

再计算logPn，这里用python自带的：

```
>>> import math
>>> math.log(0.857142)
-0.15415167982775835
>>> math.log(0.666666)
-0.4054661081086644
>>> >>> math.log(0.4)
-0.916290731874155
>>> math.log(0.25)
-1.3862943611198906
```

∑logPn和为-2.8622 ； 再乘以Wn，也就是除以4为 0.7156

BP = e^(1-8/7) 约等于 0.867

BLEU = 0.867 * e^((P1 + P2 + P3 + P4)/4) = 0.867*0.4889 = 0.4238

本来打算自己实现一个python的代码，结果发现已经有国外小哥做了，拿下来稍微修改了点内容，这里供大家参考

```
#-*- coding:utf-8 -*-
import sys
import codecs
import os
import math
import operator
import json


# 如果是一份答案的话，务必在答案的后面加上.txt   python Bleu.py Candidate ref.txt
# 如果是多份答案的话，把多份答案放到一个文件夹中  python Bleu.py Candidate 文件夹

def fetch_data(cand, ref):
    """ Store each reference and candidate sentences as a list """
    references = []
    if '.txt' in ref:
        reference_file = codecs.open(ref, 'r', 'utf-8')
        references.append(reference_file.readlines())
    else:
        for root, dirs, files in os.walk(ref):
            for f in files:
                reference_file = codecs.open(os.path.join(root, f), 'r', 'utf-
                references.append(reference_file.readlines())
    candidate_file = codecs.open(cand, 'r', 'utf-8')
    candidate = candidate_file.readlines()
    return candidate, references


def count_ngram(candidate, references, n):
```

```
    clipped_count = 0
    count = 0
    r = 0
    c = 0
    for si in range(len(candidate)):
        # Calculate precision for each sentence
        #print si
        ref_counts = []
        ref_lengths = []
        #print references
        # Build dictionary of ngram counts
        for reference in references:
            #print 'reference' + reference
            ref_sentence = reference[si]
            ngram_d = {}
            words = ref_sentence.strip().split()
            ref_lengths.append(len(words))
            limits = len(words) - n + 1
            # loop through the sentance consider the ngram length
            for i in range(limits):
                ngram = ' '.join(words[i:i+n]).lower()
                if ngram in ngram_d.keys():
                    ngram_d[ngram] += 1
                else:
                    ngram_d[ngram] = 1
            ref_counts.append(ngram_d)
        # candidate
        cand_sentence = candidate[si]
        cand_dict = {}
        words = cand_sentence.strip().split()
        limits = len(words) - n + 1
        for i in range(0, limits):
            ngram = ' '.join(words[i:i + n]).lower()
            if ngram in cand_dict:
                cand_dict[ngram] += 1
            else:
                cand_dict[ngram] = 1
        clipped_count += clip_count(cand_dict, ref_counts)
        count += limits
        r += best_length_match(ref_lengths, len(words))
        c += len(words)
    if clipped_count == 0:
```

```
        pr = 0
    else:
        pr = float(clipped_count) / count
    bp = brevity_penalty(c, r)
    return pr, bp


def clip_count(cand_d, ref_ds):
    """Count the clip count for each ngram considering all references"""
    count = 0
    for m in cand_d.keys():
        m_w = cand_d[m]
        m_max = 0
        for ref in ref_ds:
            if m in ref:
                m_max = max(m_max, ref[m])
        m_w = min(m_w, m_max)
        count += m_w
    return count


def best_length_match(ref_l, cand_l):
    """Find the closest length of reference to that of candidate"""
    least_diff = abs(cand_l-ref_l[0])
    best = ref_l[0]
    for ref in ref_l:
        if abs(cand_l-ref) < least_diff:
            least_diff = abs(cand_l-ref)
            best = ref
    return best


def brevity_penalty(c, r):
    if c > r:
        bp = 1
    else:
        bp = math.exp(1-(float(r)/c))

    return bp


def geometric_mean(precisions):
```

```
    return (reduce(operator.mul, precisions)) ** (1.0 / len(precisions))


def BLEU(candidate, references):
    precisions = []
    for i in range(4):
        pr, bp = count_ngram(candidate, references, i+1)
        precisions.append(pr)
        print 'P'+str(i+1), ' = ',round(pr, 2)
    print 'BP = ',round(bp, 2)
    bleu = geometric_mean(precisions) * bp
    return bleu

if __name__ == "__main__":
    candidate, references = fetch_data(sys.argv[1], sys.argv[2])
    bleu = BLEU(candidate, references)
    print 'BLEU = ',round(bleu, 4)
    out = open('bleu_out.txt', 'w')
    out.write(str(bleu))
    out.close()
```