МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Н. Э. БАУМАНА
Факультет информатики и систем управления
Кафедра теоретической информатики и компьютерных технологий

Лабораторная работа №4
по курсу «Автоматическая обработка текстов»

«Перплексия по униграммам и биграммам»

Выполнил:
студент группы ИУ9-11М
Беляев А. В.

Проверила:
Лукашевич Н. В.

Москва 2018

1 Цель работы

Подсчитать перплексию по униграммам и биграммам, обучив модели на одном отрывке стихотворения «Дом, который построил Джек», и выполнив непосредственный расчет на другом отрывке и сравнить получившиеся результаты.

Перплексия – среднее число вариантов, из которых происходит выбор на каждом шаге

2 Алгоритм

2.1 Униграммы

Считаем вероятности для униграмм из обучающей выборки. Затем для каждого слова w из тестовой выборки выполняем следующее:

- если слово w встречалось в обучающей выборке, то вероятность:

$$\frac{count(w) + 1}{N + V}$$

- если слово w не встречалось, то веротяность:

$$\frac{1}{N + V}$$

Здесь и далее N – общее число слов, V – число уникальных слов (типов слов)

2.2 Биграммы

Аналогично считаем вероятности для биграмм из обучающей выборки. Затем для каждой биграммы bi (состоящей из слов $w1$ и $w2$) из тестовой выборки выполняем:

- если биграмма bi встречалась в обучающей выборке, то вероятность определяется по стандартной формуле рекурсивной интерполяционной модели:

$$P_{interp}(w2|w1) = \lambda * P_{max\_likelihood}(w2|w1) + (1 - \lambda) * P_{interp}(w2)$$

- если биграмма bi не встречалась, но слова $w1$ и $w2$ – встречались, то в формуле выше обнуляется левая часть, отвечающая за биграммы 2го порядка. Используется правая часть, отвечающая за биграммы 1го порядка (слова)

- если биграмма bi не встречалась, слово $w1$ встречалось, а $w2$ – нет, то вычисления проводятся так же, как и в пункте выше

- если биграмма bi не встречалась, и слово $w1$ не встречалось, либо оба слова не встречались, то используется соответствующая формула аддитивного сглаживания:

$$P = \frac{count(w1, w2) + 1}{count(w1) + V^2_{train}} = \frac{1}{V^2_{train}}$$

3 Текст программы

```python
import re
from collections import defaultdict

file_train = 'train.txt'
file_test = 'test.txt'

WORD = 0
COUNT = 1

def read_words(filename: str) -> list:
    contents = open(filename, 'r').read().casefold()
    clean = re.sub(r'[^a-z0-9]', ' ', contents) # add cyrillic to regex!
    splitted = re.compile(r'\s+').split(clean)
    return splitted[:-1] # remove last empty word

def count_bigrams(word_list: list) -> dict:
    bigram_count = {}
    i = 1
    while i < len(word_list):
        bi = (word_list[i - 1], word_list[i])
        if bi in bigram_count:
            bigram_count[bi] += 1
        else:
            bigram_count[bi] = 1
        i += 1
    return bigram_count

def unigram_additive_model(train_words: list, test_words: list) -> (dict, float):
    # ----- train set -----
    word_count = defaultdict(int)
    for w in train_words:
        word_count[w] += 1

    N = len(train_words)
    V = len(word_count)
    print(f'N (num of words): {N}, V (num of types): {V}')

    word_probability = {}
    for w in word_count:
        word_probability[w] = (word_count[w] + 1) / (N + V)

    # ----- test set -----
    product = 1
    for w in test_words:
        if w in train_words:
            product *= word_probability[w]
        else:
            product *= 1 / (N + V)

    perplexity = product ** (-1 / len(test_words))
    return word_probability, perplexity
```

```
52
53
54 def bigram_witten_bell_model(train_words: list, test_words: list,
       unigram_model: dict) -> (dict, float):
55     # ----- train set -----
56     train_bigram_count = count_bigrams(train_words)
57     word_count = defaultdict(int)
58     for w in train_words:
59         word_count[w] += 1
60
61     print(f'num of bigrams: {len(train_bigram_count)}')
62
63     # returns number of N-gram types which start with a given word 'prefix'
64     N = lambda prefix: len(list(filter(lambda bi: bi[0] == prefix, list(
           train_bigram_count.keys()))))
65
66     # Max likelihood: count(bigram) / count (bigram-prefix)
67     Pml = lambda bigram: train_bigram_count[bigram] / word_count[bigram[0]]
68
69     # Number of bigram histories (bigrams that end with given word)
70     def E(word: str) -> int:
71         end_with_word = list(filter(lambda bi: bi[0][1] == word,
               train_bigram_count.items()))
72         return sum(w[1] for w in end_with_word)
73
74     # 1 - lambda = Num-of-prefixes / (Num-of-prefixes + Total-occurrence)
75     Lambda = lambda bigram: 1 - (N(bigram[0]) / (N(bigram[0]) + E(bigram
           [1])))
76
77     bigram_probability = {}
78     for bigram in train_bigram_count:
79         bigram_probability[bigram] = Lambda(bigram) * Pml(bigram) + \
80                                      (1 - Lambda(bigram)) * unigram_model[
                                           bigram[1]]
81
82     # ----- test set -----
83     V = len(word_count)
84     test_bigrams = count_bigrams(test_words)
85     product = 1
86     for bi in test_bigrams:
87         w1 = bi[0]
88         w2 = bi[1]
89         # bigram is present in train set
90         if bi in train_bigram_count:
91             product *= bigram_probability[bi]
92       # no bigram found, but both words are present in train set
93         elif w1 in train_words and w2 in train_words:
94             product *= (1 - Lambda(bi)) * unigram_model[w1]
95       # first word is present, second is absent
96         elif w1 in train_words and w2 not in train_words:
97             product *= (1 - Lambda(bi)) * unigram_model[w1]
98       # first or/and second not found -> use additive smoothing
99         else:
100            product *= 1 / (V ** 2)
101
```

```
102     perplexity = product ** (-1 / len(test_words))
103     return bigram_probability, perplexity
104
105
106 def main():
107     train_set = read_words(file_train)
108     test_set = read_words(file_test)
109
110     uni_train_model, pp_unigram = unigram_additive_model(train_set,
            test_set)
111     bi_train_model, pp_bigram = bigram_witten_bell_model(train_set,
            test_set, uni_train_model)
112     print(f'Unigram perplexity: {pp_unigram}\n'
113           f'Bigram  perplexity: {pp_bigram}')
114
115     with open('out_unigrams.txt', 'w+') as out:
116         for word, prob in uni_train_model.items():
117             print(f'{word} -> {prob}', file=out)
118
119     with open('out_bigrams.txt', 'w+') as out:
120         for word, prob in bi_train_model.items():
121             print(f'{word[0]} {word[1]} -> {prob}', file=out)
122
123 if __name__ == '__main__':
124     main()
```

Листинг 1: Исходный код программы

4 Результаты тестирования

Полученные результаты представлены в Таблицах 1 и 2.
Фактическое значение переменной N (количество слов): 37.
Значение переменной V (количество уникальных слов): 20.
Количество уникальных биграмм – 23.
Далее была подсчитана перплексия.
Перплексия униграмм, вычисленная против тестового набора: 23.392
Перплексия биграмм: 8.373

5 Выводы

В ходе работы были вычислены перплексии для униграмм и биграмм. Перплексия для биграмм показывает результаты в несколько раз лучше перплексии для униграмм. При этом расчет перплексии для биграмм гораздо более трудоемкий.

Таблица 1: Униграммы, аддитивное сглаживание

вот	0.03508771929824561
дом	0.03508771929824561
который	0.07017543859649122
построил	0.07017543859649122
джек	0.07017543859649122
а	0.05263157894736842
это	0.05263157894736842
пшеница	0.03508771929824561
которая	0.07017543859649122
в	0.08771929824561403
тёмном	0.05263157894736842
чулане	0.05263157894736842
хранится	0.05263157894736842
доме	0.05263157894736842
весёлая	0.03508771929824561
птица	0.03508771929824561
синица	0.03508771929824561
часто	0.03508771929824561
ворует	0.03508771929824561
пшеницу	0.03508771929824561

Таблица 2: Биграммы, сглаживание Уиттена-Белла

вот дом	0.5175438596491229
дом который	0.7675438596491229
который построил	0.7675438596491229
построил джек	0.7675438596491229
джек а	0.4619883040935673
а это	0.6842105263157895
это пшеница	0.19005847953216376
пшеница которая	0.7675438596491229
которая в	0.4736842105263158
в тёмном	0.2763157894736842
тёмном чулане	0.6842105263157895
чулане хранится	0.6842105263157895
хранится в	0.8175438596491228
в доме	0.2763157894736842
доме который	0.7675438596491229
это весёлая	0.19005847953216376
весёлая птица	0.5175438596491229
птица синица	0.5175438596491229
синица которая	0.7675438596491229
которая часто	0.13450292397660818
часто ворует	0.5175438596491229
ворует пшеницу	0.5175438596491229
пшеницу которая	0.7675438596491229