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

Лабораторная работа №12
по курсу «Автоматическая обработка текстов»

«Методы автоматического разрешения лексической многозначности»

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

Проверила:
Лукашевич Н. В.

Москва 2018

1 Цель работы

Изучить способы разрешения лексической многозначности – выбора значения слова из набора описанных значений. Предлагается использовать подход, основанный на машинном обучении – метод «Наивного байесовского классификатора».

Необходимо выбрать многозначное слово, собрать контекст из 10 предложений, содержащих это слово в определенном значении и использовать его в качестве обучающей выборки.

В качестве признаков следует использовать сами слова (лексические признаки) и части речи близлежащих к целевому слову слов («частеречные» признаки).

2 Ход работы

На Рис. 1 и 2 приведены конкретные алгоритмы обучения и применения байесовского классификатора. Необходимо согласовать терминологию с предметной областью задачи в приведенных алгоритмах:

- Vocabulary – словарь (уникальных) признаков
- Doc – предложение
- Class – значение слова
- Term – признак

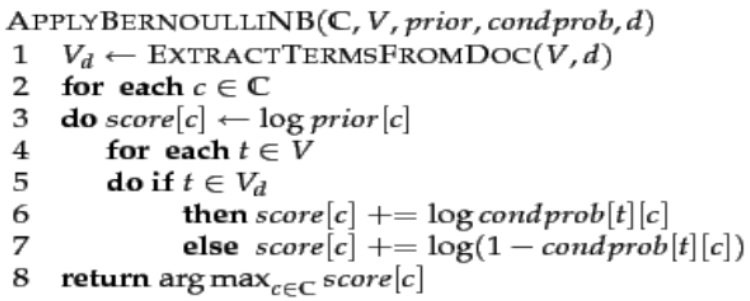
Под признаками в данном алгоритме понимаются предикаты.

TRAINBERNOULLINB($\mathbb{C}, \mathbb{D}$)
1 $V \leftarrow$ EXTRACTVOCABULARY($\mathbb{D}$)
2 $N \leftarrow$ COUNTDOCS($\mathbb{D}$)
3 **for each** $c \in \mathbb{C}$
4 **do** $N_c \leftarrow$ COUNTDOCSINCLASS($\mathbb{D}, c$)
5 $prior[c] \leftarrow N_c/N$
6 **for each** $t \in V$
7 **do** $N_{ct} \leftarrow$ COUNTDOCSINCLASSCONTAININGTERM($\mathbb{D}, c, t$)
8 $condprob[t][c] \leftarrow (N_{ct}+1)/(N_c+2)$
9 **return** $V, prior, condprob$

Рис. 1:

3 Текст программы

```
1 from collections import defaultdict
2 import re
3 from math import log
4 from typing import Optional
5 import pymorphy2
6
7 morph = pymorphy2.MorphAnalyzer()
8
```



```
APPLYBERNOULLINB(C, V, prior, condprob, d)
1  V_d ← EXTRACTTERMSFROMDOC(V, d)
2  for each c ∈ C
3  do score[c] ← log prior[c]
4     for each t ∈ V
5     do if t ∈ V_d
6           then score[c] += log condprob[t][c]
7           else score[c] += log(1 − condprob[t][c])
8  return arg max_{c∈C} score[c]
```

Рис. 2:

```
AMBIGUOUS_WORD = morph.parse('rak')[0].normal_form # make cyrillic


class Sentence:
    def __init__(self, sentence: str, vocabulary: list, klass: str = None):
        self.sent = sentence
        self.words = string_to_words(sentence)
        self.norm = normalize_words(self.words)
        self.klass = klass
        self.vector = {}
        self.apply_pos_features()                  # use part-of-speech fts
        self.apply_lexical_features(vocabulary) # use lexical features

    # apply part-of-speech features
    def apply_pos_features(self):
        amb_word_index = self.norm.index(AMBIGUOUS_WORD)
        word_left_2 = item_at_index_or_none(self.norm, amb_word_index - 2)
        word_left_1 = item_at_index_or_none(self.norm, amb_word_index - 1)
        word_right_1 = item_at_index_or_none(self.norm, amb_word_index + 1)
        word_right_2 = item_at_index_or_none(self.norm, amb_word_index + 2)
        self.vector = {
            # word to the left is present
            'l_exists': ft_word_exists(word_left_1),
            # word to the right is present
            'r_exists': ft_word_exists(word_right_1),
            # pos features
            # left   -2
            'l_2_noun': ft_noun(word_left_2),
            'l_2_adjf': ft_adjf(word_left_2),
            'l_2_verb': ft_verb(word_left_2),
            'l_2_comp': ft_comp(word_left_2),
            'l_2_numr': ft_numr(word_left_2),
            'l_2_advb': ft_advb(word_left_2),
            'l_2_prtf': ft_prtf(word_left_2),
            # left   -1
            'l_1_noun': ft_noun(word_left_1),
            'l_1_adjf': ft_adjf(word_left_1),
            'l_1_verb': ft_verb(word_left_1),
            'l_1_comp': ft_comp(word_left_1),
            'l_1_numr': ft_numr(word_left_1),
            'l_1_advb': ft_advb(word_left_1),
```

```
            'l_1_prtf': ft_prtf(word_left_1),
            # right  +1
            'r_1_noun': ft_noun(word_right_1),
            'r_1_adjf': ft_adjf(word_right_1),
            'r_1_verb': ft_verb(word_right_1),
            'r_1_comp': ft_comp(word_right_1),
            'r_1_numr': ft_numr(word_right_1),
            'r_1_advb': ft_advb(word_right_1),
            'r_1_prtf': ft_prtf(word_right_1),
            # right  +2
            'r_2_noun': ft_noun(word_right_2),
            'r_2_adjf': ft_adjf(word_right_2),
            'r_2_verb': ft_verb(word_right_2),
            'r_2_comp': ft_comp(word_right_2),
            'r_2_numr': ft_numr(word_right_2),
            'r_2_advb': ft_advb(word_right_2),
            'r_2_prtf': ft_prtf(word_right_2)
        }

    # takes dictionary and turns every word 'x' into feature: 'sentence
        contains "x"'
    def apply_lexical_features(self, vocabulary: list):
        for word in set(vocabulary):
            self.vector[word] = word in self.norm

    def satisfies_feature(self, feature_name: str) -> bool:
        return True == self.vector[feature_name]

    # extracts features that evaluate to true(1) for given sentence
    def extract_active_features(self) -> list:
        fts = dict(filter(lambda ft: True == ft[1], self.vector.items()))
        return list(fts.keys())

    def __repr__(self):
        return self.__str__()

    def __str__(self):
        return self.sent


# ========== Feature predicates ==========
ft_noun = lambda w: 'NOUN' == tag(w)
ft_adjf = lambda w: 'ADJF' == tag(w)
ft_verb = lambda w: 'VERB' == tag(w)
ft_comp = lambda w: 'COMP' == tag(w)
ft_numr = lambda w: 'NUMR' == tag(w)
ft_advb = lambda w: 'ADVB' == tag(w)
ft_prtf = lambda w: 'PRTF' == tag(w)
ft_word_exists = lambda w: w is not None
# ========================================


def tag(w: str) -> Optional[str]:
    return morph.parse(w)[0].tag.POS if w is not None else None
```

```
104 def item_at_index_or_none(lst: list, index: int) -> Optional[str]:
105     try:
106         return lst[index]
107     except IndexError:
108         return None
109 
110 def read_sentences(filename: str) -> list:
111     contents = open(filename, 'r').read()
112     # remove empty strings
113     return list(filter(lambda s: '' != s, contents.split('\n')))
114 
115 def string_to_words(s: str) -> list:
116     clean = re.sub(r'[^a-z]', ' ', s.casefold()) # make cyrillic
117     return re.compile(r'\s+').split(clean)
118 
119 # normalizes words and removes functional words
120 def normalize_words(words: list) -> list:
121     tags_to_remove = ['NPRO', 'PRED', 'PREP', 'CONJ', 'PRCL', 'INTJ']
122     normalized = []
123     for w in words:
124         parsed = morph.parse(w)[0]
125         if parsed.tag.POS not in tags_to_remove:
126             normalized.append(parsed.normal_form)
127     return normalized
128 
129 def extract_unique_features(sentences: list) -> list:
130     return sentences[0].vector.keys()
131 
132 def count_sents_in_class(sentences: list, klass: str) -> int:
133     return len(list(filter(lambda s: klass == s.klass, sentences)))
134 
135 
136 def count_sents_in_class_satisfying_feature(sentences: list, klass: str,
        feature: str) -> int:
137     sent_of_klass = list(filter(lambda s: klass == s.klass, sentences))
138     sent_satisfying_ft = list(filter(lambda s: s.satisfies_feature(feature)
            , sent_of_klass))
139     return len(sent_satisfying_ft)
140 
141 
142 def train_bernoulli(klasses: list, sents: list) -> (dict, dict, dict):
143     V = extract_unique_features(sents)
144     N = len(sents)
145     cond_prod = defaultdict(dict)
146     prior = {}
147 
148     for klass in klasses:
149         Nc = count_sents_in_class(sents, klass)
150         prior[klass] = Nc / N
151 
152         for feature in V:
153             Nct = count_sents_in_class_satisfying_feature(sents, klass,
                    feature)
154             cond_prod[feature][klass] = (Nct + 1) / (Nc + 2)
155     return V, prior, cond_prod
```

```
156
157 def apply_bernoulli(klasses: list, V: list, prior: dict, cond_prob: dict,
        new_sent: Sentence) -> dict:
158     Vd = new_sent.extract_active_features()
159     score = {}
160
161     for klass in klasses:
162         score[klass] = log(prior[klass])
163         for feature in V:
164             if feature in Vd:
165                 score[klass] += log(cond_prob[feature][klass])
166             else:
167                 score[klass] += log(1 - cond_prob[feature][klass])
168     return score
169
170
171 def main():
172     # make dictionary of normalized words from all train files
173     train_1 = read_sentences('c1.txt')
174     train_2 = read_sentences('c2.txt')
175     vocabulary = set()
176     for s in train_1 + train_2:
177         normalized = normalize_words(string_to_words(s))
178         vocabulary.update(normalized)
179     vocabulary = list(vocabulary)
180
181     # make sentences (Docs) for class 1
182     klass_1 = 'cancer'
183     sent_1 = list(map(lambda s: Sentence(s, vocabulary, klass_1), train_1))
184
185     # make sentences (Docs) for class 2
186     klass_2 = 'crayfish'
187     sent_2 = list(map(lambda s: Sentence(s, vocabulary, klass_2), train_2))
188
189     V, prior, cond_prod = train_bernoulli([klass_1,klass_2], sent_1+sent_2)
190
191     test_sentences = list(map(lambda s: Sentence(s, vocabulary),
            read_sentences('test.txt')))
192     for sent in test_sentences:
193         scores = apply_bernoulli([klass_1, klass_2], V, prior, cond_prod,
                sent)
194
195         res = klass_1 if max(scores.values())==scores[klass_1] else klass_2
196         print(f'{res}: {sent}: cancer:{scores[klass_1]:.3f}, crayfish:{
                scores[klass_2]:.3f}')
197
198 if __name__ == '__main__':
199     main()
```

Листинг 1: Исходный код программы

4 Результаты тестирования

В качестве многозначного слова было выбрано слово *Рак*. Первое значение – бо-

Таблица 1: Результаты работы

Предложение	Cancer, без частеречн	Crayfish, б/чр.	Cancer, частеречн	Crayfish, частеречн.
В скандинавских страна питаются вареными раками	-23.178	**-21.516**	-33.980	**-31.258**
Известно, что рак может вызываться некоторыми профессиональными вредностями	-18.383	**-18.019**	**-25.765**	-26.125
Длина тела широкопалого рака может достигать 25 см и более	-18.383	**-18.019**	-27.980	**-26.462**
Употребление экологически чистых продуктов уменьшает вероятность появления рака	-20.781	**-20.417**	**-26.882**	-28.213
Понятие рак распространено среди игроков многопользовательских игр, в частности Dota	**-18.383**	-18.808	**-23.211**	-24.206

лезнь. Обучающая выборка представлена в Таблице 2. Второе значение – животное. Обучающая выбора - в Таблице 3.

Тестовая выборка и результаты прдставлены в Таблице 1. Выборка составлена следующим образом:

- 2 предложения, в которых есть слова из соотвествующих обучающих выборок.
- 1 предложение, в котором нет слов из обучающих выборок.
- 1 предложение, в котором есть слова из обеих обучающих выборок.
- 1 предложение из другой предметной области (впрочем, может быть отнесено сразу в обе области)

В первом случае (без учета частеречных признаков) алгоритм верно «угадал» одного рака и один рак.

Во втором случае (с учетом) алгоритм верно справился во всех случаях – как простых, так и сложных (без пересечения по словам).

Можно сделать вывод о том, что использование частеречных признаков улучшает результаты как в абсолютном так и относительом значении: значения без учета частеречных признаков ближе друг к другу и сильно подвержены колебаниям. Значения с учетом чр. признаков подвержены им в меньшей степени.

Слово вне обоих контекстов было определено одинаково.

5 Выводы

В ходе работы были изучены методы машинного обучения для разрешения лексической неоднозначности. Ожидаемые результаты совпали с реальными.

Таблица 2: Обучающая выборка класса 1

Основной упор в разрабатываемой BostonGene системе делается на интеграцию исследований терапии рака
Точные причины развития рака неизвестны
Общими симптомами рака являются резкое снижение веса, потеря аппетита, плохое самочувствие
Самым распространенным является рак легких
группа экспертов по изучению рака составляла классификацию веществ, входящих в продукты питания
В клетках рака с высокой частотой возникают разломы
Рак может поражать практически любую систему, орган или ткань организма
Проблема рака имеет огромное значение
Показатели смертности от рака в разных странах сильно варьируют
Курение является наиболее значимой причиной рака, определяя его возникновение

Таблица 3: Обучающая выборка класса 2

Цвет рака зависит от условия обитания и состава воды
В пищу раков стали употреблять очень много лет назад
Молодые раки питаются только растениями
Поймать раков в их привычной среде обитания довольно просто
Ловить раков башмаком придумали достаточно давно
Цвет вареных раков должен быть равномерный и насыщенно красный
Регулярное использование мяса раков улучшает самочувствие людей после затяжной простуды
Употребление раков может причинить вред организму только в случае, если злоупотреблять ими
В состав мяса раков входят различные микроэлементы, которые насыщают организм
Очень вкусными получаются раки вареные в пиве