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

Лабораторная работа №13
по курсу «Информационный поиск»
«Анализ тональности»

Выполнил:
студент группы ИУ9-21М
Беляев А. В.

Проверила:
Лукашевич Н. В.

Москва 2019

1 Цель работы

Даны 2 коллекции цитат - обучающая и тестовая. Необходимо провести анализ тональности цитат и оценить работу классификатора из пакета `scikit-learn` на разных типах векторизации:

- TF-iDF
- частоты
- булевские вектора

В качестве метода классификации был выбран алгоритм AdaBoost (Adaptive Boosting). В основе алгоритма лежат т.н. «weak learners» – слабые классификаторы, которые случайным образом «угадывают» результат. Такие классификаторы объединены в «комитеты классификации». На каждой итерации алгоритма проверяются неверно классифицированные объекты, чтобы на следующей итерации алгоритма коммитет классификации «сфокусирвал свое внимание» на них.

1.1 Ход работы

```
from datetime import datetime
from xml.dom import minidom
import nltk
import numpy as np
from nltk.corpus import stopwords
from sklearn.ensemble import AdaBoostClassifier
from sklearn.feature_extraction.text import TfidfVectorizer,
 ↪  CountVectorizer
from sklearn import metrics
from lab16 import clean_up_sentence, normalize_sentence

TRAIN_DATA = 'news_eval_train.xml'
TEST_DATA = 'news_eval_test.xml'

TONALITY = ['+', '-', '0']
BOOL_VECT_TRUE = 1
BOOL_VECT_FALSE = 0

nltk.download('stopwords')
STOP_WORDS = stopwords.words('russian')
STOP_WORDS_EXCEPTIONS = [
    'все', 'ничего', 'никогда', 'наконец', 'больше', 'хорошо', 'лучше',
    'всегда', 'конечно', 'всю', 'такой', 'впрочем', 'так', 'можно',
     ↪  'даже']
STOP_WORDS = list(set(STOP_WORDS) - set(STOP_WORDS_EXCEPTIONS))

class Cite:
```

```
26      def __init__(self, speech_: str, eval_: str):
27          self.speech = speech_.strip()
28          self.evaluation = eval_.strip()
29          self.tokenized = tokenize(self.speech)
30          self.tokenized_str = ' '.join(self.tokenized)
31
32
33  # [[1, 2, 3], [4], [5, 6]] -> [1, 2, 3, 4, 5, 6]
34  def flatten(lst: list) -> list:
35      flat_list = []
36      for sublist in lst:
37          for item in sublist:
38              flat_list.append(item)
39      return flat_list
40
41
42  def tokenize(text: str) -> list:
43      def remove_stopwords(sentence: str) -> list:
44          words = sentence.split(' ')
45          return list(filter(lambda w: w not in STOP_WORDS, words))
46
47      sentences = nltk.sent_tokenize(text)
48      clean_sents = list(map(lambda s: clean_up_sentence(s), sentences))
49      norm_sents = list(map(lambda s: normalize_sentence(s), clean_sents))
50      no_stopwords_sents = list(map(lambda s: remove_stopwords(s),
        ↪  norm_sents))
51      return flatten(no_stopwords_sents)
52
53
54  def classify(vectorizer, train_cites: list, test_cites: list,
    ↪  boolean_vectorizer=False):
55      tokenized_train = [c.tokenized_str for c in train_cites]
56      x_train = vectorizer.fit_transform(tokenized_train).toarray()
57      if boolean_vectorizer:
58          x_train = np.where(x_train > 0, BOOL_VECT_TRUE, BOOL_VECT_FALSE)
59      y_train = [c.evaluation for c in train_cites]
60
61      tokenized_test = [c.tokenized_str for c in test_cites]
62      x_test = vectorizer.transform(tokenized_test).toarray()
63      y_test = [c.evaluation for c in test_cites]
64
65      classifier = AdaBoostClassifier(n_estimators=10)
66      classifier.fit(x_train, y_train)
67      predicted = classifier.predict(x_test)
68
```

```
    print(metrics.classification_report(y_test, predicted))
    print(f'classified:  {datetime.now()}')


def parse(filename: str) -> list:
    xmldoc = minidom.parse(filename)
    document = xmldoc.getElementsByTagName('document')[0]
    sentences = document.getElementsByTagName('sentence')

    citations = []
    for s in sentences:
        speech =
        ↪  s.getElementsByTagName('speech')[0].childNodes[0].nodeValue
        evaluation =
        ↪  s.getElementsByTagName('evaluation')[0].childNodes[0].nodeValue.strip()
        if evaluation in TONALITY:
            citations.append(Cite(speech, evaluation))
    return citations

def main():
    train_cites = parse(TRAIN_DATA)
    test_cites = parse(TEST_DATA)

    print(f'classify:    {datetime.now()}')
    vectorizer = TfidfVectorizer()
    # vectorizer = CountVectorizer()

    classify(vectorizer, train_cites, test_cites,
     ↪  boolean_vectorizer=False)

if __name__ == '__main__':
    main()
```

2 Результаты

Входные XML файлы были разобраны, цитаты с неоднозначнй тональностью были исключены, а остальные были нормализованы.

Из-за того, что классификаторы, состоящие в комитете классификации не намного лучше слуйчаного гадания (как следует из документации), алгоритм часто используется в сочетании с другими методами. В данной работе AdaBoost был применен отдельно от других алгоритмов. Можно выделить его плюс по сравнению с другими методами - главным настраиваемым параметром метода является n_estimators - количество классификаторов в комитете.

Были проверены результаты работы с комитетом из 10, 100 и 300 классификато-

ров.

На листингах ниже педставлены результаты классификации с использованием разных способов векторизации.

TF-IDF:

```
 ====================== N = 10 ======================
              precision    recall  f1-score   support
           +       0.56      0.30      0.39      1448
           -       0.45      0.89      0.59      1890
           0       0.00      0.00      0.00      1235

   micro avg       0.47      0.47      0.47      4573
   macro avg       0.34      0.40      0.33      4573
weighted avg       0.36      0.47      0.37      4573
====================== N = 100 ======================
           +       0.54      0.48      0.51      1448
           -       0.50      0.76      0.61      1890
           0       0.40      0.14      0.21      1235

   micro avg       0.50      0.50      0.50      4573
   macro avg       0.48      0.46      0.44      4573
weighted avg       0.49      0.50      0.47      4573
====================== N = 300 ======================
           +       0.52      0.50      0.51      1448
           -       0.54      0.66      0.59      1890
           0       0.40      0.28      0.33      1235

   micro avg       0.51      0.51      0.51      4573
   macro avg       0.48      0.48      0.48      4573
weighted avg       0.49      0.51      0.49      4573
```

Частотная векторизация:

```
====================== N = 10  ======================
              precision    recall  f1-score   support
           +       0.57      0.31      0.40      1448
           -       0.45      0.89      0.59      1890
           0       0.00      0.00      0.00      1235

   micro avg       0.47      0.47      0.47      4573
   macro avg       0.34      0.40      0.33      4573
weighted avg       0.36      0.47      0.37      4573
====================== N = 100 ======================
           +       0.53      0.55      0.54      1448
           -       0.52      0.74      0.61      1890
           0       0.44      0.14      0.21      1235
```

```
                                                    
   micro avg       0.52      0.52      0.52      4573
   macro avg       0.50      0.48      0.46      4573
weighted avg       0.50      0.52      0.48      4573
====================== N = 300 ======================
           +       0.55      0.54      0.54      1448
           -       0.57      0.69      0.63      1890
           0       0.41      0.28      0.33      1235

   micro avg       0.53      0.53      0.53      4573
   macro avg       0.51      0.50      0.50      4573
weighted avg       0.52      0.53      0.52      4573
```

Булевская векторизация:

```
====================== N = 10  ======================
              precision    recall  f1-score   support
           +       0.57      0.31      0.40      1448
           -       0.45      0.89      0.59      1890
           0       0.00      0.00      0.00      1235

   micro avg       0.47      0.47      0.47      4573
   macro avg       0.34      0.40      0.33      4573
weighted avg       0.36      0.47      0.37      4573
====================== N = 100 ======================
           +       0.56      0.51      0.53      1448
           -       0.51      0.78      0.62      1890
           0       0.44      0.14      0.21      1235

   micro avg       0.52      0.52      0.52      4573
   macro avg       0.50      0.47      0.45      4573
weighted avg       0.51      0.52      0.48      4573
====================== N = 300 ======================
           +       0.54      0.54      0.54      1448
           -       0.58      0.68      0.63      1890
           0       0.45      0.32      0.37      1235

   micro avg       0.54      0.54      0.54      4573
   macro avg       0.52      0.52      0.52      4573
weighted avg       0.53      0.54      0.53      4573
```

Поскольку комитет из 10 слабых классиикаторов дает посредственные резуьтаты, а комитет из 300 слишком долго работает (порядка 30 минут), считаем комитет из 100 классификаторов оптимальным.

Исключим из `STOP_WORDS` слова, содержащиеся в `STOP_WORDS_EXCEPTIONS`, чтобы по-другому учитывать эмоциональную окраску цитат.

Проверим результаты для Count Vectorizer'a, т.к. он дал наилучший результат по отношению к временным затратам. В комитете 100 классификаторов:

	precision	recall	f1-score	support
+	0.53	0.53	0.53	1448
-	0.52	0.75	0.61	1890
0	0.43	0.12	0.19	1235
micro avg	0.51	0.51	0.51	4573
macro avg	0.49	0.47	0.45	4573
weighted avg	0.50	0.51	0.47	4573

3 Выводы

Был произведен анализ тональности с помощью метода AdaBoost пакета scikit-learn. Выбранный метод показал сравнительно слабые результаты, как и было заявлено в документации к нему. При попытке подогнать параметры метода под результат, ожидаемо выигрываем в одних резльутатах и проигрываем в других.

Тем не менее, этот метод может быть хорошим дополнением к другим методам, т.к. слабо подвержен проблеме переобучения.