Лабораторная работа №6: "Классификация текста."

ИУ5-23 Зорин Арсений

Задание:

- Для произвольного набора данных, предназначенного для классификации текстов, решите задачу классификации текста двумя способами:
  - На основе CountVectorizer или TfidfVectorizer.
  - На основе моделей word2vec или Glove или fastText.
  - Сравните качество полученных моделей.

In [25]:

```python
import numpy as np
import pandas as pd
from typing import Dict, Tuple
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_error, median_absolute_er
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSVR
import seaborn as sns
import tensorflow as tf
from collections import Counter
from sklearn.datasets import fetch_20newsgroups
from gensim.models import word2vec
from nltk.corpus import stopwords
import re
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/a.zorin/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[25]: True

In [7]:

```python
categories = ['comp.graphics', 'misc.forsale', 'talk.politics.misc', 'rec.sport.hockey']
groups = fetch_20newsgroups(subset='train', categories=categories)
data = groups['data']
```

In [8]:

```python
def accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray) -> Dict[int, float]:
    """
    Вычисление метрики accuracy для каждого класса
    y_true - истинные значения классов
    y_pred - предсказанные значения классов
    Возвращает словарь: ключ - метка класса,
    значение - Accuracy для данного класса
    """
    # Для удобства фильтрации сформируем Pandas DataFrame
    d = {'t': y_true, 'p': y_pred}
    df = pd.DataFrame(data=d)
    # Метки классов
    classes = np.unique(y_true)
    # Результирующий словарь
    res = dict()
    # Перебор меток классов
    for c in classes:
        # отфильтруем данные, которые соответствуют
        # текущей метке класса в истинных значениях
        temp_data_flt = df[df['t']==c]
        # расчет accuracy для заданной метки класса
        temp_acc = accuracy_score(
            temp_data_flt['t'].values,
            temp_data_flt['p'].values)
        # сохранение результата в словарь
        res[c] = temp_acc
    return res

def print_accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray):
    """
    Вывод метрики accuracy для каждого класса
    """
    accs = accuracy_score_for_classes(y_true, y_pred)
    if len(accs)>0:
        print('Метка \t Accuracy')
    for i in accs:
        print('{} \t {}'.format(i, accs[i]))
```

In [9]:

```python
vectorized = CountVectorizer()
vectorized.fit(data)
vocabulary = vectorized.vocabulary_
print('Количество сформированных признаков - {}'.format(len(vocabulary)))
```

```
Количество сформированных признаков - 34701
```

In [10]:

```python
for i in list(vocabulary)[1:10]:
    print('{}={}'.format(i, vocabulary[i]))
```

```
dwarf=12688
bcarh601=6807
bnr=7381
ca=8258
jim=18501
jordan=18615
subject=30225
re=26291
truly=31992
```

In [11]:

```python
test_features = vectorized.transform(data)
test_features
```

Out[11]:

```
<2234x34701 sparse matrix of type '<class 'numpy.int64'>'
        with 317800 stored elements in Compressed Sparse Row format>
```

In [12]:

```python
# Размер нулевой строки
len(test_features.todense()[0].getA1())
```

Out[12]: 34701

In [16]:

```python
def VectorizeAndClassify(vectorizers_list, classifiers_list):
    for v in vectorizers_list:
        for c in classifiers_list:
            pipeline1 = Pipeline([("vectorizer", v), ("classifier", c)])
            score = cross_val_score(pipeline1, groups['data'], groups['target'], scoring='accuracy', cv=3).mear
            print('Векторизация - {}'.format(v))
            print('Модель для классификации - {}'.format(c))
            print('Accuracy = {}'.format(score))
            print('===========================')
```

In [17]:

```python
vectorizers_list = [CountVectorizer(vocabulary = vocabulary), TfidfVectorizer(vocabulary = vocabulary)]
classifiers_list = [LogisticRegression(C=3.0), LinearSVC(), KNeighborsClassifier()]
VectorizeAndClassify(vectorizers_list, classifiers_list)
```

```
/usr/local/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '000005102000': 2, '000007': 3,
                            '000100255pixel': 4, '000256': 5, '0004': 6,
                            '0007': 7, '000k': 8, '000usd': 9, '001': 10,
                            '0010': 11, '0010580b': 12, '001116': 13,
                            '001200201pixel': 14, '001323': 15, '001338': 16,
                            '00196': 17, '002': 18, '002302': 19, '002339': 20,
                            '0028': 21, '00309': 22, '003221': 23, '0038': 24,
                            '003848': 25, '0039': 26, '004253agrgb': 27,
                            '004325': 28, '004808': 29, ...})
Модель для классификации - LogisticRegression(C=3.0)
Accuracy = 0.9462882778860263
===========================
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '000005102000': 2, '000007': 3,
                            '000100255pixel': 4, '000256': 5, '0004': 6,
                            '0007': 7, '000k': 8, '000usd': 9, '001': 10,
                            '0010': 11, '0010580b': 12, '001116': 13,
                            '001200201pixel': 14, '001323': 15, '001338': 16,
                            '00196': 17, '002': 18, '002302': 19, '002339': 20,
                            '0028': 21, '00309': 22, '003221': 23, '0038': 24,
                            '003848': 25, '0039': 26, '004253agrgb': 27,
                            '004325': 28, '004808': 29, ...})
Модель для классификации - LinearSVC()
Accuracy = 0.9480779870582858
===========================
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '000005102000': 2, '000007': 3,
                            '000100255pixel': 4, '000256': 5, '0004': 6,
                            '0007': 7, '000k': 8, '000usd': 9, '001': 10,
                            '0010': 11, '0010580b': 12, '001116': 13,
                            '001200201pixel': 14, '001323': 15, '001338': 16,
                            '00196': 17, '002': 18, '002302': 19, '002339': 20,
                            '0028': 21, '00309': 22, '003221': 23, '0038': 24,
                            '003848': 25, '0039': 26, '004253agrgb': 27,
                            '004325': 28, '004808': 29, ...})
Модель для классификации - KNeighborsClassifier()
Accuracy = 0.689341728608886
===========================
Векторизация - TfidfVectorizer(vocabulary={'00': 0, '000': 1, '000005102000': 2, '000007': 3,
                            '000100255pixel': 4, '000256': 5, '0004': 6,
                            '0007': 7, '000k': 8, '000usd': 9, '001': 10,
                            '0010': 11, '0010580b': 12, '001116': 13,
                            '001200201pixel': 14, '001323': 15, '001338': 16,
                            '00196': 17, '002': 18, '002302': 19, '002339': 20,
                            '0028': 21, '00309': 22, '003221': 23, '0038': 24,
                            '003848': 25, '0039': 26, '004253agrgb': 27,
                            '004325': 28, '004808': 29, ...})
Модель для классификации - LogisticRegression(C=3.0)
Accuracy = 0.9588204517572346
===========================
Векторизация - TfidfVectorizer(vocabulary={'00': 0, '000': 1, '000005102000': 2, '000007': 3,
                            '000100255pixel': 4, '000256': 5, '0004': 6,
                            '0007': 7, '000k': 8, '000usd': 9, '001': 10,
                            '0010': 11, '0010580b': 12, '001116': 13,
                            '001200201pixel': 14, '001323': 15, '001338': 16,
                            '00196': 17, '002': 18, '002302': 19, '002339': 20,
                            '0028': 21, '00309': 22, '003221': 23, '0038': 24,
                            '003848': 25, '0039': 26, '004253agrgb': 27,
                            '004325': 28, '004808': 29, ...})
Модель для классификации - LinearSVC()
Accuracy = 0.9677726059031536
===========================
Векторизация - TfidfVectorizer(vocabulary={'00': 0, '000': 1, '000005102000': 2, '000007': 3,
                            '000100255pixel': 4, '000256': 5, '0004': 6,
                            '0007': 7, '000k': 8, '000usd': 9, '001': 10,
                            '0010': 11, '0010580b': 12, '001116': 13,
                            '001200201pixel': 14, '001323': 15, '001338': 16,
                            '00196': 17, '002': 18, '002302': 19, '002339': 20,
                            '0028': 21, '00309': 22, '003221': 23, '0038': 24,
                            '003848': 25, '0039': 26, '004253agrgb': 27,
                            '004325': 28, '004808': 29, ...})
Модель для классификации - KNeighborsClassifier()
Accuracy = 0.8325906280820764
===========================
```

In [26]:

```python
# word2vec
vocabular = []
stop_words = stopwords.words('english')
tok = nltk.tokenize.WordPunctTokenizer()
for line in groups['data']:
    line = line.strip().lower()
    line = re.sub("[^a-zA-Z]"," ", line)
    token = tok.tokenize(line)
    token = [w for w in token if not w in stop_words]
    vocabular.append(token)
```

In [28]:

```python
%time model_data = word2vec.Word2Vec(vocabular, workers=2, min_count=10, window=15, sample=1e-3)
```

```
CPU times: user 2.7 s, sys: 53.7 ms, total: 2.76 s
Wall time: 1.54 s
```

In [29]:

```python
def sentiment(v, c):
    model = Pipeline(
        [("vectorizer", v),
         ("classifier", c)])
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print_accuracy_score_for_classes(y_test, y_pred)
```

In [30]:

```python
class EmbeddingVectorizer(object):
    def __init__(self, model):
        self.model = model
        self.size = model.vector_size

    def fit(self, X, y):
        return self

    def transform(self, X):
        return np.array([np.mean(
            [self.model[w] for w in words if w in self.model]
            or [np.zeros(self.size)], axis=0)
            for words in X])
```

In [31]:

```python
boundary = 800
X_train = vocabular[:boundary]
X_test = vocabular[boundary:]
y_train = groups['target'][:boundary]
y_test = groups['target'][boundary:]
```

In [32]:

```python
sentiment(EmbeddingVectorizer(model_data.wv), LogisticRegression(C=5.0))
```

```
/usr/local/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Метка    Accuracy
0        0.9277777777777778
1        0.8564102564102564
2        0.9714285714285714
3        0.9230769230769231
```