**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №3

по дисциплине «Прикладные интеллектуальные системы и экспертные системы»

Предварительная обработка текстовых данных

Студент Осипов А.А.

Группа М-ИАП-22

Руководитель Кургасов В.В.

Липецк 2022 г.

Задание кафедры

Вариант 10

1) В среде Jupiter Notebook создать новый ноутбук (Notebook);

2) Импортировать необходимые для работы библиотеки и модули;

3) Загрузить обучающую и экзаменационную выборку в соответствие с вариантом;

4) Вывести на экран по одному-два документа каждого класса;

5) Применить стемминг, записав обработанные выборки (тестовую и обучающую) в новые переменные;

6) Провести векторизацию выборки:

a. Векторизовать обучающую и тестовую выборки простым подсчетом

слов (CountVectorizer) и значеним max\_features = 10000

b. Вывести и проанализировать первые 20 наиболее частотных слов всей выборки и каждого класса по-отдельности.

c. Применить процедуру отсечения стоп-слов и повторить пункт b.

d. Провести пункты a – c для обучающей и тестовой выборки, для которой проведена процедура стемминга.

e. Векторизовать выборки с помощью TfidfTransformer (с использованием TF и TF-IDF взвешиваний) и повторить пункты b-d.

7) По результатам пункта 6 заполнить таблицы наиболее частотными терминами обучающей выборки и каждого класса по отдельности.

Всего должно получиться по 4 таблицы для выборки, к которой применялась операция стемминга и 4 таблицы для выборки, к которой операция стемминга не применялась

8) Используя конвейер (Pipeline) реализовать модель Наивного Байесовского классификатора и выявить на основе показателей качества (значения полноты, точности, f1-меры и аккуратности), какая предварительная обработка данных обеспечит наилучшие результаты классификации. Должны быть исследованы следующие характеристики:

• Наличие - отсутствие стемминга

• Отсечение – не отсечение стоп-слов

• Количество информативных терминов (max\_features)

• Взвешивание: Count, TF, TF-IDF

9) По каждому пункту работы занести в отчет программный код и результат вывода.

10) По результатам классификации занести в отчет выводы о наиболее подходящей предварительной обработке данных (наличие стемминга, взвешивание терминов, стоп-слова, количество информативных терминов).

Ход работы

Импортируем необходимые для работы библиотеки и модули.

- pandas — программная библиотека на языке Python для обработки и анализа данных;

- numPy (сокращенно от Numerical Python)— библиотека с открытым исходным кодом для языка программирования Python. Возможности: поддержка многомерных массивов (включая матрицы); поддержка высокоуровневых математических функций, предназначенных для работы с многомерными массивами;

- matplotlib — библиотека на языке программирования Python для визуализации данных двумерной и трёхмерной графикой;

- библиотека NLTK — пакет библиотек и программ для символьной и статистической обработки естественного языка, написанных на языке программирования Python. Содержит графические представления и примеры данных;

- itertools стандартизирует основной набор быстрых эффективных по памяти инструментов, которые полезны сами по себе или в связке с другими инструментами;

scikit-learn – это библиотека Python, которая является одной из самых полезных библиотек Python для машинного обучения. Она включает все алгоритмы и инструменты, которые нужны для задач классификации, регрессии и кластеризации. Она также включает все методы оценки производительности модели машинного обучения.

1) Импортировать необходимые для работы библиотеки и модули;

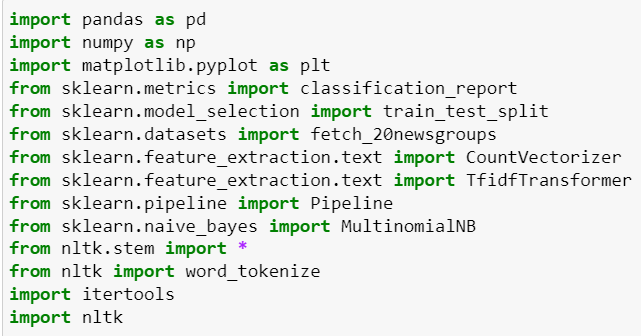


Рисунок 1 – Импорт библиотек

2) Загрузить обучающую и экзаменационную выборку в соответствие с вариантом;

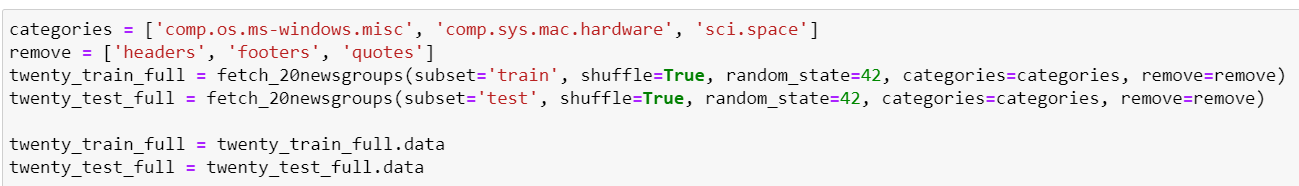


Рисунок 2 – Загрузка выборки

3) Вывести на экран по одному-два документа каждого класса;



Рисунок 3 – Документ для класса comp.os.ms-windows.misc

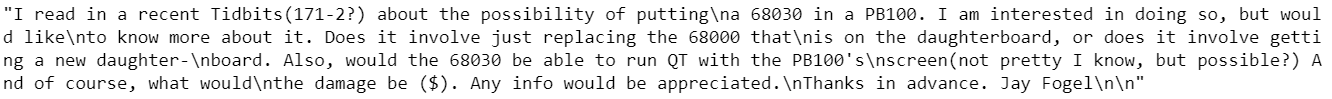


Рисунок 4 – Документ для класса comp.sys.mac.hardware

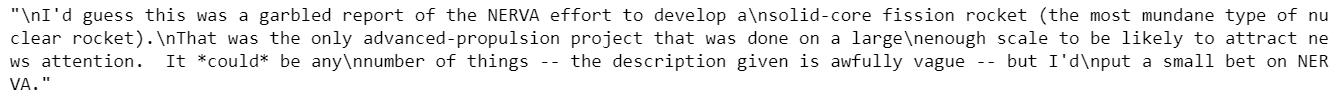


Рисунок 5 – Документ для класса sci.space

4) Применить стемминг, записав обработанные выборки (тестовую и обучающую) в новые переменные;

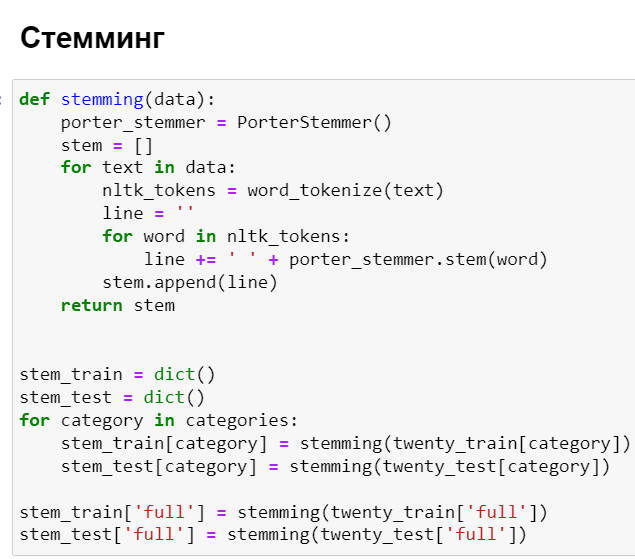


Рисунок 6 – Процедура стемминга

5) Провести векторизацию выборки:

a. Векторизовать обучающую и тестовую выборки простым подсчетом слов (CountVectorizer) и значеним max\_features = 10000

b. Вывести и проанализировать первые 20 наиболее частотных слов всей выборки и каждого класса по-отдельности.

c. Применить процедуру отсечения стоп-слов и повторить пункт b.

d. Провести пункты a – c для обучающей и тестовой выборки, для которой проведена процедура стемминга.

e. Векторизовать выборки с помощью TfidfTransformer (с использованием TF и TF-IDF взвешиваний) и повторить пункты b-d.



Рисунок 7 – Со стеммингом для comp.os.ms-windows.misc



Рисунок 8 – Со стеммингом для всех категорий



Рисунок 9 – Со стеммингом для comp.sys.mac.hardware



Рисунок 10 – Со стеммингом для sci.space



Рисунок 11 – Без стемминга для comp.os.ms-windows.misc



Рисунок 12 – Без стемминга для всех категорий



Рисунок 13 – Без стемминга comp.sys.mac.hardware



Рисунок 14 – Без стемминга для sci.space

6) Используя конвейер (Pipeline) реализовать модель Наивного Байесовского классификатора и выявить на основе показателей качества (значения полноты, точности, f1-меры и аккуратности), какая предварительная обработка данных обеспечит наилучшие результаты классификации. Должны быть исследованы следующие характеристики:

• Отсечение – не отсечение стоп-слов

• Количество информативных терминов (max\_features)

• Взвешивание: Count, TF, TF-IDF



Рисунок 15 – Пример работы программы со следующими параметрами (max\_features = 1000, со стоп словами, без TF, TF-IDF)



Рисунок 16 – Пример работы программы со следующими параметрами

(max\_features=1000, со стоп словами без tf, с idf)



Рисунок 17 – Пример работы программы со следующими параметрами

(max\_features=1000, со стоп словами с tf, без idf)



Рисунок 18 – Пример работы программы со следующими параметрами

(max\_features=1000, со стоп словами, с tf и idf)



Рисунок 19 – Пример работы программы со следующими параметрами

(max\_features=1000, без стоп слов без tf и idf)



Рисунок 20 – Пример работы программы со следующими параметрами

(max\_features=1000, без стоп слов, без tf, с idf)



Рисунок 21 – Пример работы программы со следующими параметрами

(max\_features=1000, без стоп слов, с tf, без idf)



Рисунок 22 – Пример работы программы со следующими параметрами

(max\_features=1000, без стоп слов, с tf и idf)



Рисунок 23 – Пример работы программы со следующими параметрами

(max\_features=5000, со стоп словами без tf и idf)



Рисунок 24 – Пример работы программы со следующими параметрами

(max\_features=5000, со стоп словами, без tf, с idf)



Рисунок 25 – Пример работы программы со следующими параметрами

(max\_features=5000, со стоп словами с tf, без idf=False)



Рисунок 26 – Пример работы программы со следующими параметрами

(max\_features=5000, со стоп словами с tf и idf)



Рисунок 27 – Пример работы программы со следующими параметрами

(max\_features=5000, без стоп слов, без tf и idf)



Рисунок 28 – Пример работы программы со следующими параметрами

(max\_features=5000, без стоп слов без tf, с idf)



Рисунок 29 – Пример работы программы со следующими параметрами

(max\_features=5000, без стоп слов, с tf, без idf)



Рисунок 30 – Пример работы программы со следующими параметрами

(max\_features=5000, без стоп слов, с tf и idf)



Рисунок 31 – Пример работы программы со следующими параметрами

(max\_features=10000, со стоп словами, без tf и idf)



Рисунок 32 – Пример работы программы со следующими параметрами

(max\_features=10000, со стоп словами, без tf, с idf)



Рисунок 33 – Пример работы программы со следующими параметрами

(max\_features=10000, со стоп словами, с tf, без idf)



Рисунок 34 – Пример работы программы со следующими параметрами

(max\_features=10000, со стоп словами, с tf и idf)



Рисунок 35 – Пример работы программы со следующими параметрами

(max\_features=10000, без стоп слов, без tf и idf)



Рисунок 36 – Пример работы программы со следующими параметрами

(max\_features=10000, без стоп слов, без tf, с idf)



Рисунок 37 – Пример работы программы со следующими параметрами

(max\_features=10000, без стоп слов с tf, без idf)



Рисунок 38 – Пример работы программы со следующими параметрами

(max\_features=10000, без стоп слов, с tf и idf)

По результатам классификации наиболее подходящая предварительная обработка данных является со следующими параметрами:

- с tf и tf-idf;

- max\_features = 10000;

- со стоп словами.

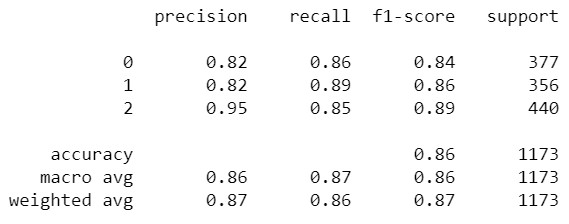


Рисунок 39 – Результат работы программы

Код программы

#!/usr/bin/env python

# coding: utf-8

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import classification\_report

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import fetch\_20newsgroups

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfTransformer

from sklearn.pipeline import Pipeline

from sklearn.naive\_bayes import MultinomialNB

from nltk.stem import \*

from nltk import word\_tokenize

import itertools

import nltk

# ## Загрузка выборки

categories = ['comp.os.ms-windows.misc', 'comp.sys.mac.hardware', 'sci.space']

remove = ['headers', 'footers', 'quotes']

twenty\_train\_full = fetch\_20newsgroups(subset='train', shuffle=True, random\_state=42, categories=categories, remove=remove)

twenty\_test\_full = fetch\_20newsgroups(subset='test', shuffle=True, random\_state=42, categories=categories, remove=remove)

twenty\_train\_full = twenty\_train\_full.data

twenty\_test\_full = twenty\_test\_full.data

twenty\_train = dict()

twenty\_test = dict()

for category in categories:

twenty\_train[category] = fetch\_20newsgroups(subset='train', shuffle=True, random\_state=42, categories=[category], remove=remove)

twenty\_test[category] = fetch\_20newsgroups(subset='test', shuffle=True, random\_state=42, categories=[category], remove=remove)

twenty\_train[category] = twenty\_train[category].data

twenty\_test[category] = twenty\_test[category].data

twenty\_train['full'] = twenty\_train\_full

twenty\_test['full'] = twenty\_test\_full

# ## Стемминг

def stemming(data):

porter\_stemmer = PorterStemmer()

stem = []

for text in data:

nltk\_tokens = word\_tokenize(text)

line = ''

for word in nltk\_tokens:

line += ' ' + porter\_stemmer.stem(word)

stem.append(line)

return stem

stem\_train = dict()

stem\_test = dict()

for category in categories:

stem\_train[category] = stemming(twenty\_train[category])

stem\_test[category] = stemming(twenty\_test[category])

stem\_train['full'] = stemming(twenty\_train['full'])

stem\_test['full'] = stemming(twenty\_test['full'])

# ## Векторизация

def SortbyTF(inputStr):

return inputStr[1]

def top\_list(vect, data, count):

x = list(zip(vect.get\_feature\_names(),np.ravel(data.sum(axis=0))))

x.sort(key=SortbyTF, reverse = True)

return x[:count]

# ## Итоговая таблица

def process(train, categories):

cats = categories[:]

cats.append('full')

mux = pd.MultiIndex.from\_product([['Count','TF','TF-IDF'], ['Без стоп-слов','С стоп-cловами']])

summary = dict()

for category in cats:

summary[category] = pd.DataFrame(columns=mux)

stop\_words = [None, 'english']

idf = [False, True]

indx\_stop = {

'english': 'Без стоп-слов',

None: 'С стоп-cловами'

}

indx\_tf = {

False: 'TF',

True: 'TF-IDF'

}

for category in cats:

for stop in stop\_words:

vect = CountVectorizer(max\_features=10000, stop\_words=stop)

vect.fit(train[category])

train\_data = vect.transform(train[category])

summary[category]['Count', indx\_stop[stop]] = top\_list(vect, train\_data, 20)

for tf in idf:

tfidf = TfidfTransformer(use\_idf = tf).fit(train\_data)

train\_fidf = tfidf.transform(train\_data)

summary[category][indx\_tf[tf], indx\_stop[stop]] = top\_list(vect, train\_fidf, 20)

return summary

summ\_without\_stem = process(twenty\_train, categories)

summ\_with\_stem = process(stem\_train, categories)

for cat in ['full'] + categories:

summ\_without\_stem[cat].to\_excel('without\_stem\_' + cat + '.xlsx')

summ\_with\_stem[cat].to\_excel('with\_stem\_' + cat + '.xlsx')

# ## Pipelines

import os

def print\_classification\_score(clf, data):

print(classification\_report(gs\_clf.predict(data.data), data.target))

categories = ['comp.os.ms-windows.misc', 'comp.sys.mac.hardware', 'sci.space']

remove = ['headers', 'footers', 'quotes']

twenty\_train\_full = fetch\_20newsgroups(subset='train', shuffle=True, random\_state=42, categories=categories, remove=remove)

twenty\_test\_full = fetch\_20newsgroups(subset='test', shuffle=True, random\_state=42, categories=categories, remove=remove)

def prespocess(data, max\_features, stop\_words, use\_tf, use\_idf):

tf = None

cv = CountVectorizer(max\_features=max\_features, stop\_words=stop\_words).fit(data)

if use\_tf:

tf = TfidfTransformer(use\_idf=use\_idf).fit(cv.transform(data))

return cv, tf

def models\_grid\_search(data\_train, data\_test):

max\_features = [1000,5000,10000]

stop\_words = ['english', None]

use\_tf = [True, False]

use\_idf = [True, False]

res = dict()

for param in itertools.product(max\_features, stop\_words, use\_tf, use\_idf):

cv, tf = prespocess(data\_train.data, param[0], param[1], param[2], param[3])

if tf:

clf = MultinomialNB().fit(tf.transform(cv.transform(data\_train.data)), data\_train.target)

prep\_test = tf.transform(cv.transform(data\_test.data))

else:

clf = MultinomialNB().fit(cv.transform(data\_train.data), data\_train.target)

prep\_test = cv.transform(data\_test.data)

name = f'max\_features={param[0]}\_stop\_words={param[1]}\_use\_tf={param[2]}\_use\_idf={param[3]}'

res[name] = pd.DataFrame(classification\_report(clf.predict(prep\_test), data\_test.target, output\_dict=True))

return res

scores = models\_grid\_search(twenty\_train\_full, twenty\_test\_full)

if not os.path.exists('scores'):

os.makedirs('scores')

for name, score in scores.items():

score.to\_excel('scores/' + name + '.xlsx')

from sklearn.model\_selection import GridSearchCV

parameters = {

'vect\_\_max\_features': (1000,5000,10000),

'vect\_\_stop\_words': ('english', None),

'tfidf\_\_use\_idf': (True, False),

}

text\_clf = Pipeline([

('vect', CountVectorizer()),

('tfidf', TfidfTransformer()),

('clf', MultinomialNB())

])

gs\_clf = GridSearchCV(text\_clf, parameters, n\_jobs=-1, cv=3)

gs\_clf.fit(X = twenty\_train\_full.data, y = twenty\_train\_full.target)

print\_classification\_score(gs\_clf, twenty\_test\_full)

gs\_clf.best\_params\_

Вывод

В ходе выполнения данной лабораторной работы мы получили базовые навыки работы с языком python и набором функций для анализа и обработки данных.

Контрольные вопросы

1. Особенности задачи классификации текстовых данных.

Анализе текстовых данных в машинном обучении используется методы регрессии, классификации и кластеризации. Данные методы были описаны в этой работе ранее. Но стоит отметить что есть главная отличие в анализе текстовых данных, так как сама обработка текста является очень сложной задачей в машинном обучении. Главная отличие – это интеллектуальный анализ текстовых данных. Так как текстовый документ для человека ‒ это набор слов, который несет смысл, для машины – это просто битовые данные. И задача интеллектуального анализа текстовых данных состоит в том, чтобы машина смогла понимать смысл текстового документа. Перед тем как использовать алгоритмы машинного обучение, нужно также применить методы обработки текстовых данных.

Классификация текстовых документов, так же как и в случае классификации объектов, заключается в отнесении документа к одному из заранее известных классов. Часто классификацию применительно к текстовым документам называют категоризацией или рубрикацией. Очевидно, что данные названия происходят от задачи систематизации документов по каталогам, категориям и рубрикам. При этом структура каталогов может быть как одноуровневой, так и многоуровневой (иерархической).

1. Этапы предварительной обработки данных.

Этап подготовки и фильтрации данных может занять много времени. Предварительная подготовка данных включает в себя:

- очистку;

- отбор экземпляров;

- нормализацию;

- преобразование данных;

- выделение признаков;

- отбор признаков;

- прочие манипуляции с данными.

3) Алгоритм и особенности Наивного Байесовского метода.

Алгоритм применения;

1. Для каждого класса вычисляется апостериорная вероятность;

2. Выбирается тот класс, для которого значение максимально.

Особенности:

- алгоритм легко и быстро предсказывает класс тестового набора данных. Он также хорошо справляется с многоклассовым прогнозированием;

- производительность наивного байесовского классификатора лучше, чем у других простых алгоритмов, таких как логистическая регрессия. Более того, вам требуется меньше обучающих данных;

- он хорошо работает с категориальными признаками(по сравнению с числовыми). Для числовых признаков предполагается нормальное распределение, что может быть серьезным допущением в точности нашего алгоритма.

4) Как влияет размер словаря терминов на точность классификации?

При увеличении размера словаря точность оценок увеличивается.

5) Как влияет способ взвешивания терминов на точность классификации?

Способ взвешивания терминов влияет прямо пропорционально на точность классификации.