надо ввести в командной строке

1. pip install spacy
2. spacy download ru_core_news_sm

Lab 5

Задание:

Для произвольного предложения или текста решите следующие задачи:

- Токенизация.
- Частеречная разметка.
- Лемматизация.
- Выделение (распознавание) именованных сущностей.
- Разбор предложения.

Для произвольного набора данных, предназначенного для классификации текстов, решите задачу классификации текста двумя способами:

- Способ 1. На основе CountVectorizer или TfidfVectorizer.
- Способ 2. На основе моделей word2vec или Glove или fastText.

Сравните качество полученных моделей.

```
import gzip
import shutil
import os
```

Text

```
text1 = 'Москва – субъект Российской Федерации, город федерального значения, столица Росси
text2 = 'Компьютер — устройство или система, способная автоматически выполнять заданную, и
text3 = 'Каждая команда самостоятельно описывает свою систему в отдельном спейсе в Conflue
```

```
text1
```

```
'Москва – субъект Российской Федерации, город федерального значения, столица Российс
кой Федерации, '
```

1. Токенизация

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
nltk_tk_1 = nltk.WordPunctTokenizer()
nltk_tk_1.tokenize(text1)
```

```
['Москва',
 '–',
 'субъект',
 'Российской',
 'Федерации',
 ',',
 'город',
 'федерального',
 'значения',
 ',',
 'столица',
 'Российской',
 'Федерации',
 '.']
```

```
# Токенизация по предложениям
nltk_tk_sents = nltk.tokenize.sent_tokenize(text3)
print(len(nltk_tk_sents))
nltk_tk_sents
```

```
2
['Каждая команда самостоятельно описывает свою систему в отдельном спейсе в Confluenc
 'В документировании обязательно участвует техлид, потому что он обязан фиксировать и
```

```
from spacy.lang.ru import Russian
import spacy
nlp = spacy.load('ru_core_news_sm')
spacy_text1 = nlp(text1)
spacy_text1
```

```
Москва – субъект Российской Федерации, город федерального значения, столица Российско
```

```
for t in spacy_text1:
    print(t)
```

```
Москва
–
субъект
Российской
Федерации
```

```
,
город
федерального
значения
,
столица
Российской
Федерации
.
```

```
spacy_text2 = nlp(text2)
spacy_text2
```

```
Компьютер — устройство или система, способная автоматически выполнять заданную, измен
```

```
spacy_text3 = nlp(text3)
spacy_text3
```

```
Каждая команда самостоятельно описывает свою систему в отдельном спейсе в Confluence
```

2. Частеречная разметка (Part-Of-Speech tagging, POS-tagging)

[ ] *Скрыто 3 ячейки.*

3. Лемматизация

[ ] *Скрыто 3 ячейки.*

4. Выделение (распознавание) именованных сущностей, named-entity recognition (NER)

[ ] *Скрыто 4 ячейки.*

5. Разбор предложения

[ ] *Скрыто 6 ячеек.*

Classification models

```
import gensim
from gensim.models import word2vec
```

```
import re
import pandas as pd
import numpy as np
from typing import Dict, Tuple
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from nltk import WordPunctTokenizer
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
datasets_dirname = './datasets/'
if os.path.exists(datasets_dirname) == 0:
    os.mkdir(datasets_dirname)

with gzip.open(datasets_dirname + 'clickbait_data.gz', 'rb') as f_in:
    with open(datasets_dirname + 'clickbait_data.txt', 'wb') as f_out:
        shutil.copyfileobj(f_in, f_out)

with gzip.open(datasets_dirname + 'non_clickbait_data.gz', 'rb') as f_in:
    with open(datasets_dirname + 'non_clickbait_data.txt', 'wb') as f_out:
        shutil.copyfileobj(f_in, f_out)
```

```
import pandas as pd

clickbait = pd.read_csv(datasets_dirname + 'clickbait_data.txt', delimiter='__label__ ', h
clickbait['target'] = 1
clickbait

non_clickbait = pd.read_csv(datasets_dirname + 'non_clickbait_data.txt', delimiter='__labe
non_clickbait['target'] = 0

df = pd.concat([clickbait, non_clickbait], axis=0)
df = df.sample(frac=1)

df.to_csv(datasets_dirname + 'dataset.csv', index=False)
os.remove(datasets_dirname + 'clickbait_data.txt')
os.remove(datasets_dirname + 'non_clickbait_data.txt')

df
```

	text	target
2874	What Do You Actually Deserve For Christmas	1
11511	Mamdouh Habib to be released from Guantanamo B...	0
10350	How Much British History Do You Actually Know	1
4702	How Many Of These "Harry Potter" Characters Ca...	1
9078	Man arrested in connection with July 21 London...	0
...	...	...
6518	Mayor of Flint, Michigan resigns for health re...	0
13895	Freddie From "ICarly" Got Hitched And Your Chi...	1
8124	Australian Vaulter Returns to Millrose Games a...	0

```
df.target.value_counts()
```

```
0    16001
1    15999
Name: target, dtype: int64
```

```
# Подготовим корпус
corpus = []
stop_words = stopwords.words('english')
tok = WordPunctTokenizer()
for line in df['text'].values:
    line1 = line.strip().lower()
    line1 = re.sub("[^a-zA-Z]"," ", line1)
    text_tok = tok.tokenize(line1)
    text_tok1 = [w for w in text_tok if not w in stop_words]
    corpus.append(text_tok1)
```

```
corpus[:5]
```

```
[['actually', 'deserve', 'christmas'],
 ['mamdouh', 'habib', 'released', 'guantanamo', 'bay', 'without', 'charge'],
 ['much', 'british', 'history', 'actually', 'know'],
 ['many', 'harry', 'potter', 'characters', 'name'],
 ['man', 'arrested', 'connection', 'july', 'london', 'bomb', 'attempts']]
```

```
# количество текстов в корпусе не изменилось и соответствует целевому признаку
assert df.shape[0]==len(corpus)
```

```
%time
model_imdb = word2vec.Word2Vec(corpus, workers=4, min_count=10, window=10, sample=1e-3)
```

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 8.58 µs
```

```
# Проверим, что модель обучилась
```

```
print(model_imdb.wv.most_similar(positive=['great'], topn=5))
```

```
[('ellen', 0.9997909069061279), ('still', 0.9997831583023071), ('meet', 0.9997776746
```

word2vec

```
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)


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]))


class EmbeddingVectorizer(object):
    '''
    Для текста усредним вектора входящих в него слов
    '''
    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])


# Обучающая и тестовая выборки
boundary = 700
X_train = corpus[:boundary]
X_test = corpus[boundary:]
y_train = df.target.values[:boundary]
y_test = df.target.values[boundary:]


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

```
 Метка    Accuracy
 0        0.905306174573782
 1        0.871027559306861
```

```
# Сформируем общий словарь для обучения моделей из обучающей и тестовой выборки
vocab_list = df['text'].tolist()
vocab_list[1:10]
```

```
['Mamdouh Habib to be released from Guantanamo Bay without charge',
 'How Much British History Do You Actually Know',
 'How Many Of These "Harry Potter" Characters Can You Name',
 'Man arrested in connection with July 21 London bomb attempts',
 'Gangs Said to Contact Politicians',
 'American tourist killed in Beijing',
 'Do You Know Which TV Show This "B" Is From',
 'People Eat King Cake For The First Time',
 'A "Rugrats" Artist Is "Bugged" By People Thinking The Babies Grew Up To Be Really A
```

```
vocabVect = CountVectorizer()
vocabVect.fit(vocab_list)
```

```
corpusVocab = vocabVect.vocabulary_
print('Количество сформированных признаков - {}'.format(len(corpusVocab)))
```

```
 Количество сформированных признаков - 22761
```

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

```
 do=6160
 you=22629
 actually=641
 deserve=5743
 for=7983
 christmas=4003
 mamdouh=12367
 habib=9116
 to=20630
```

```
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, df['text'], df['target'], scoring='accuracy
            print('Векторизация - {}'.format(v))
            print('Модель для классификации - {}'.format(c))
            print('Accuracy = {}'.format(score))
            print('============================')
```

```
import numpy as np
import pandas as pd
from typing import Dict, Tuple
from scipy import stats
from IPython.display import Image
from sklearn.datasets import load_iris, load_boston
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_erro
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 matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")

import warnings
warnings.filterwarnings('ignore')
```

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

```
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '000th': 2, '00s': 3
                               '04': 5, '05': 6, '08': 7, '08m': 8, '09': 9,
                               '10': 10, '100': 11, '1000': 12, '10000th': 13,
                               '1000blackgirls': 14, '100k': 15, '100m': 16,
                               '100th': 17, '100ºf': 18, '101': 19, '101st': 20,
                               '102': 21, '103': 22, '104': 23, '105': 24,
                               '106': 25, '107': 26, '108': 27, '109': 28,
                               '109th': 29, ...})
Модель для классификации - LogisticRegression(C=3.0)
Accuracy = 0.9717499950618573
============================
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '000th': 2, '00s': 3
                               '04': 5, '05': 6, '08': 7, '08m': 8, '09': 9,
                               '10': 10, '100': 11, '1000': 12, '10000th': 13,
                               '1000blackgirls': 14, '100k': 15, '100m': 16,
                               '100th': 17, '100ºf': 18, '101': 19, '101st': 20,
                               '102': 21, '103': 22, '104': 23, '105': 24,
                               '106': 25, '107': 26, '108': 27, '109': 28,
                               '109th': 29, ...})
Модель для классификации - LinearSVC()
Accuracy = 0.9703750233803694
============================
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '000th': 2, '00s': 3
                               '04': 5, '05': 6, '08': 7, '08m': 8, '09': 9,
                               '10': 10, '100': 11, '1000': 12, '10000th': 13,
                               '1000blackgirls': 14, '100k': 15, '100m': 16,
                               '100th': 17, '100ºf': 18, '101': 19, '101st': 20,
                               '102': 21, '103': 22, '104': 23, '105': 24,
                               '106': 25, '107': 26, '108': 27, '109': 28,
                               '109th': 29, ...})
Модель для классификации - KNeighborsClassifier()
Accuracy = 0.7986253863375131
============================
Векторизация - TfidfVectorizer(vocabulary={'00': 0, '000': 1, '000th': 2, '00s': 3
                               '04': 5, '05': 6, '08': 7, '08m': 8, '09': 9,
                               '10': 10, '100': 11, '1000': 12, '10000th': 13,
                               '1000blackgirls': 14, '100k': 15, '100m': 16,
                               '100th': 17, '100ºf': 18, '101': 19, '101st': 20,
                               '102': 21, '103': 22, '104': 23, '105': 24,
                               '106': 25, '107': 26, '108': 27, '109': 28,
                               '109th': 29, ...})
Модель для классификации - LogisticRegression(C=3.0)
Accuracy = 0.9699999794329509
============================
Векторизация - TfidfVectorizer(vocabulary={'00': 0, '000': 1, '000th': 2, '00s': 3
                               '04': 5, '05': 6, '08': 7, '08m': 8, '09': 9,
                               '10': 10, '100': 11, '1000': 12, '10000th': 13,
                               '1000blackgirls': 14, '100k': 15, '100m': 16,
                               '100th': 17, '100ºf': 18, '101': 19, '101st': 20,
                               '102': 21, '103': 22, '104': 23, '105': 24,
                               '106': 25, '107': 26, '108': 27, '109': 28,
                               '109th': 29, ...})
Модель для классификации - LinearSVC()
Accuracy = 0.9751874696771529
============================
```

```
Векторизация - TfidfVectorizer(vocabulary={'00': 0, '000': 1, '000th': 2, '00s': 3
                               '04': 5, '05': 6, '08': 7, '08m': 8, '09': 9,
```

```
X_train, X_test, y_train, y_test = train_test_split(df['text'], df['target'], test_size=0.
```

```
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)
```

```
sentiment(TfidfVectorizer(), LogisticRegression(C=5.0))
```

```
Метка    Accuracy
0        0.9735957155311994
1        0.960607201104002
```

```
sentiment(TfidfVectorizer(ngram_range=(1,3)),  LogisticRegression(C=5.0))
```

```
Метка    Accuracy
0        0.9447004608294931
1        0.9720235854974282
```

```
sentiment(TfidfVectorizer(ngram_range=(2,3)), LogisticRegression(C=5.0))
```

```
Метка    Accuracy
0        0.9527961140864367
1        0.9055325555137373
```

```
sentiment(TfidfVectorizer(ngram_range=(1,4)), LogisticRegression(C=5.0))
```

```
Метка    Accuracy
0        0.9351102254328061
1        0.974658135742065
```

```
sentiment(TfidfVectorizer(ngram_range=(2,6)), LogisticRegression(C=5.0))
```

```
Метка    Accuracy
0        0.9427076846431686
1        0.9111780203236733
```

```
os.remove('./datasets/dataset.csv')
```