Урок 2. Д/з.

Ввод [1]:
```
import pandas as pd
```

Новости, пользователи со списками последних прочитанных новостей

Ввод [2]:
```
news = pd.read_csv("articles.csv")
users = pd.read_csv("users_articles.csv")
```

Нужно получить векторные представления пользователей на основе прочитанным ими новостей и самих новостей

1. Получаем векторные представления новостей

Ввод [3]:
```
from gensim.corpora.dictionary import Dictionary

#предобработка текстов
import re
import numpy as np
from nltk.corpus import stopwords
from razdel import tokenize
import pymorphy2
```

Ввод [4]:
```
stopword_ru = stopwords.words('russian')
morph = pymorphy2.MorphAnalyzer()
```

Ввод [5]:
```
with open('stopwords.txt') as f:
    additional_stopwords = [w.strip() for w in f.readlines() if w]
stopword_ru += additional_stopwords
```

Ввод [6]:
```
def clean_text(text):
    '''
    очистка текста

    на выходе очищеный текст

    '''
    if not isinstance(text, str):
        text = str(text)

    text = text.lower()
    text = text.strip('\n').strip('\r').strip('\t')
```

```
    text = re.sub("-\s\r\n\|-\s\r\n|\r\n", '', str(text))

    text = re.sub("[0-9]|[-—.,:;_%©«»?*!@#№$^•·&()]|[+=]|[[]|[]]|[/]|", '', text)
    text = re.sub(r"\r\n\t|\n|\\s|\r\t|\\n", ' ', text)
    text = re.sub(r'[\xad]|[\s+]', ' ', text.strip())

    #tokens = list(tokenize(text))
    #words = [_.text for _ in tokens]
    #words = [w for w in words if w not in stopword_ru]

    #return " ".join(words)
    return text

cache = {}

def lemmatization(text):
    '''
    лемматизация
        [0] если зашел тип не `str` делаем его `str`
        [1] токенизация предложения через razdel
        [2] проверка есть ли в начале слова '-'
        [3] проверка токена с одного символа
        [4] проверка есть ли данное слово в кэше
        [5] лемматизация слова
        [6] проверка на стоп-слова

    на выходе лист отлемматизированых токенов
    '''

    # [0]
    if not isinstance(text, str):
        text = str(text)

    # [1]
    tokens = list(tokenize(text))
    words = [_.text for _ in tokens]

    words_lem = []
    for w in words:
        if w[0] == '-': # [2]
            w = w[1:]
        if len(w)>1: # [3]
            if w in cache: # [4]
                words_lem.append(cache[w])
            else: # [5]
                temp_cach = cache[w] = morph.parse(w)[0].normal_form
                words_lem.append(temp_cach)

    words_lem_without_stopwords=[i for i in words_lem if not i in stopword_ru] # [6]
```

```
    return words_lem_without_stopwords
```

Ввод [7]:
```
%%time
#Запускаем очистку текста. Будет долго...
news['title'] = news['title'].apply(lambda x: clean_text(x), 1)
```

```
<ipython-input-6-7ee348d9b386>:15: FutureWarning: Possible nested set at position 39
  text = re.sub("[0-9]|[--.,:;_%©«»?*!@#№$^•·&()]|[+=]|[[]|[]]|[/]|", '', text)

Wall time: 21.5 s
```

Ввод [8]:
```
%%time
#Запускаем лемматизацию текста. Будет очень долго...
news['title'] = news['title'].apply(lambda x: lemmatization(x), 1)
```

```
Wall time: 2min 19s
```

Обучим нашу модель

Ввод [9]:
```
#сформируем список наших текстов, разбив еще и на пробелы
texts = [t for t in news['title'].values]

# Create a corpus from a list of texts
common_dictionary = Dictionary(texts)
common_corpus = [common_dictionary.doc2bow(text) for text in texts]
```

Запускаем обучение

Ввод [10]:
```
from gensim.models import LdaModel
```

Ввод [11]:
```
%%time
# Train the model on the corpus.
lda = LdaModel(common_corpus, num_topics=25, id2word=common_dictionary)#, passes=10)
```

```
Wall time: 25 s
```

Ввод [12]:
```
from gensim.test.utils import datapath
# Save model to disk.
temp_file = datapath("model.lda")
lda.save(temp_file)

# Load a potentially pretrained model from disk.
lda = LdaModel.load(temp_file)
```

Обучили модель. Теперь 2 вопроса:

1. как выглядят наши темы
2. как получить для документа вектор значений (вероятности принадлежности каждой теме)

Ввод [13]:
```
# Create a new corpus, made of previously unseen documents.
other_texts = [t for t in news['title'].iloc[:3]]
other_corpus = [common_dictionary.doc2bow(text) for text in other_texts]

unseen_doc = other_corpus[2]
print(other_texts[2])
lda[unseen_doc]
```

```
['форвард', 'авангард', 'томаш', 'заборский', 'прокомментировать', 'игра', 'свой', 'команда', 'матч', 'чемпионат', 'кхл', 'пр
отив', 'атланта', 'nnnn', 'провести', 'плохой', 'матч', 'нижний', 'новгород', 'против', 'торпедо', 'настраиваться', 'первый',
'минута', 'включиться', 'работа', 'сказать', 'заборский', 'получиться', 'забросить', 'быстрый', 'гол', 'задать', 'хороший', '
темп', 'поединок', 'мочь', 'играть', 'ещё', 'хороший', 'сторона', 'пять', 'очко', 'выезд', 'девять', 'это', 'хороший']
```

Out[13]:
```
[(10, 0.24978022),
 (11, 0.023781622),
 (13, 0.15819567),
 (14, 0.46783817),
 (23, 0.08249416)]
```

Ввод [14]:
```
x=lda.show_topics(num_topics=25, num_words=7,formatted=False)
topics_words = [(tp[0], [wd[0] for wd in tp[1]]) for tp in x]

#Below Code Prints Only Words
for topic,words in topics_words:
    print("topic_{}: ".format(topic)+" ".join(words))
```

```
topic_0: компания фильм улица клиент предприниматель реконструкция египетский
topic_1: млн составить население тыс подчёркивать год общий
topic_2: сша ракета россия который год это американский
topic_3: который год сша взрыв человек пострадать это
topic_4: год правительство президент страна также россия проект
topic_5: вирус девочка больной фрагмент компенсировать устойчивость явиться
topic_6: гражданин ребёнок научный территория который граница участок
```

Давайте напишем функцию, которая будет нам возвращать векторное представление новости

Ввод [15]:
```
#text = news['title'].iloc[0]

def get_lda_vector(text):
    unseen_doc = common_dictionary.doc2bow(text)
    lda_tuple = lda[unseen_doc]
    not_null_topics = dict(zip([i[0] for i in lda_tuple], [i[1] for i in lda_tuple]))

    output_vector = []
    for i in range(25):
        if i not in not_null_topics:
            output_vector.append(0)
        else:
            output_vector.append(not_null_topics[i])
    return np.array(output_vector)
```

Ввод [16]:
```
topic_matrix = pd.DataFrame([get_lda_vector(text) for text in news['title'].values])
topic_matrix.columns = ['topic_{}'.format(i) for i in range(25)]
topic_matrix['doc_id'] = news['doc_id'].values
topic_matrix = topic_matrix[['doc_id']+['topic_{}'.format(i) for i in range(25)]]
topic_matrix.head(5)
```

Out[16]:

	doc_id	topic_0	topic_1	topic_2	topic_3	topic_4	topic_5	topic_6	topic_7	topic_8	...	topic_15	topic_16	topic_17	topic_18	topic_19	topic_20	topic_21	topic_
0	6	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	...	0.000000	0.040156	0.0	0.000000	0.0	0.0	0.0	(
1	4896	0.0	0.0	0.0	0.649187	0.000000	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.0	0.000000	0.0	0.0	0.0	(
2	4897	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.0	0.000000	0.0	0.0	0.0	(
3	4898	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.0	0.000000	0.0	0.0	0.0	(
4	4899	0.0	0.0	0.0	0.000000	0.393512	0.0	0.0	0.0	0.0	...	0.038076	0.353346	0.0	0.191681	0.0	0.0	0.0	(

5 rows × 26 columns

Следующий шаг - векторные представления пользователей

Ввод [17]:
```
users.head(3)
```

Out[17]:

	uid	articles
0	u105138	[293672, 293328, 293001, 293622, 293126, 1852]
1	u108690	[3405, 1739, 2972, 1158, 1599, 322665]
2	u108339	[1845, 2009, 2356, 1424, 2939, 323389]

Ввод [18]:
```
doc_dict = dict(zip(topic_matrix['doc_id'].values, topic_matrix[['topic_{}'.format(i) for i in range(25)]].values))
```

Ввод [19]:
```
doc_dict[293622]
```

Out[19]:
```
array([0.        , 0.04340571, 0.        , 0.        , 0.09570687,
       0.        , 0.12378234, 0.        , 0.        , 0.12674166,
       0.03850625, 0.        , 0.03037902, 0.        , 0.25161475,
       0.        , 0.06706324, 0.11507846, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.09861792, 0.        ])
```

Ввод [20]:
```
#

def get_user_embedding(user_articles_list, agr_func):
    user_articles_list = eval(user_articles_list)
    user_vector = np.array([doc_dict[doc_id] for doc_id in user_articles_list])
    user_vector = agr_func(user_vector, 0)
    return user_vector
```

Теперь получим эмбединги для всех пользователей и проверим их качество на конкретной downstream-задаче

Ввод [21]:
```
#

user_embeddings_list = []
for agr_func in [np.mean, np.median, np.max]:
    user_embeddings = pd.DataFrame([i for i in users['articles'].apply(lambda x: get_user_embedding(x, agr_func), 1)])
    user_embeddings.columns = ['topic_{}'.format(i) for i in range(25)]
    user_embeddings['uid'] = users['uid'].values
    user_embeddings = user_embeddings[['uid']+['topic_{}'.format(i) for i in range(25)]]
    user_embeddings_list.append(user_embeddings)
```

Датасет готов - можно попробовать обучить модель. Загрузим нашу разметку

Ввод [22]:
```
target = pd.read_csv("users_churn.csv")
```

Ввод [23]:
```
#
```

```
X_list = []
for user_embeddings in user_embeddings_list:
    X = pd.merge(user_embeddings, target, 'left')
    X_list.append(X)
```

Ввод [24]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline
```

Ввод [25]:
```
preds_list = []
for X in X_list:
    #разделим данные на train/test
    X_train, X_test, y_train, y_test = train_test_split(X[['topic_{}'.format(i) for i in range(25)]],
                                                        X['churn'], random_state=0)

    #обучим модель
    logreg = LogisticRegression()
    logreg.fit(X_train, y_train)

    #прогнозы для тестовой выборки
    preds = logreg.predict_proba(X_test)[:, 1]

    preds_list.append(preds)
```

Рассчитаем Precision, Recall, F_score

Ввод [26]:
```
import itertools
from sklearn.metrics import f1_score, roc_auc_score, precision_score
```

Ввод [27]:
```
print(f'\tF-Score\t\tPrecision\tRecall\troc_auc_score')
for preds, agr_func_name in zip(preds_list, ['mean', 'median', 'max', 'min']):

    precision, recall, thresholds = precision_recall_curve(y_test, preds)
    fscore = (2 * precision * recall) / (precision + recall)
    # locate the index of the largest f score
    ix = np.argmax(fscore)
    print(f'{agr_func_name}\t{fscore[ix]:.4f}\t\t{precision[ix]:.4f}\t\t{recall[ix]:.4f}\t{roc_auc_score(y_test, preds):.4f}')
```

```
        F-Score         Precision       Recall  roc_auc_score
mean    0.7110          0.6655          0.7633  0.9542
median  0.7549          0.7212          0.7918  0.9648
max     0.7842          0.7975          0.7714  0.9693
```

Домашнее задание

1. Самостоятельно разобраться с тем, что такое tfidf (документация https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html) и еще - https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction (https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction))
2. Модифицировать код функции get_user_embedding таким образом, чтобы считалось не среднее (как в примере np.mean), а медиана. Применить такое преобразование к данным, обучить модель прогнозирования оттока и посчитать метрики качества и сохранить их: roc auc, precision/recall/f_score (для 3 последних - подобрать оптимальный порог с помощью precision_recall_curve, как это делалось на уроке)
3. Повторить п.2, но используя уже не медиану, а max
4. (опциональное, если очень хочется) Воспользовавшись полученными знаниями из п.1, повторить пункт 2, но уже взвешивая новости по tfidf (подсказка: нужно получить веса-коэффициенты для каждого документа. Не все документы одинаково информативны и несут какой-то положительный сигнал). Подсказка 2 - нужен именно idf, как вес.
5. Сформировать на выходе единую таблицу, сравнивающую качество 3 разных метода получения эмбедингов пользователей: mean, median, max, idf_mean по метрикам roc_auc, precision, recall, f_score
6. Сделать самостоятельные выводы и предположения о том, почему тот или ной способ оказался эффективнее остальных

Ссылки

1. http://www.machinelearning.ru/wiki/images/d/d5/Voron17survey-artm.pdf (http://www.machinelearning.ru/wiki/images/d/d5/Voron17survey-artm.pdf)
2. https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation (https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation)

Ввод [ ]: