# ПРИЛОЖЕНИЕ 5 ВАРИАНТ КОДА НА PYTHON3 ДЛЯ АНАЛИЗА ТЕКСТОВ

Для анализа использован парсер и код на python3, представленный сотрудником Санкт-Петербургской школы социальных наук и востоковедения НИУ ВШЭ Аленой Пестовой на открытом семинаре, используемый с разрешения автора. Часть кода заимствована из книги М. Рассела и М. Классена «Data Mining»[[1]](#endnote-1).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232 | Для получения ключа доступа к VK API необходимо:   * создать свое Standalone приложение вк <https://dev.vk.com/> * перейти в настройки в приложении -> сервисный ключ доступа и можем теперь использовать этот ключ для запросов к API   # для того, чтобы иметь доступ к файлам на своем гугл диске  from google.colab import drive  drive.mount('/content/drive')  PATH = '/content/drive/MyDrive/vk\_analysis/'  import requests  import pandas as pd  import numpy as np  from tqdm import tqdm  Парсинг данных  Для начала нам нужно скачать посты ВК. Это можно сделать с помощью VK Api. Сначала посмотрим на то, как делать запросы к VK API по одному  # пример запроса, в файле лежит код доступа  with open(PATH+'key.txt', 'r') as f:      KEY\_ACCESS = f.read().strip()  # раскомментируйте строку и вставьте свой код доступа  # KEY\_ACCESS = '….'  r = requests.get("https://api.vk.com/method/wall.get",                       params={"domain": 'example', "count": 20, "offset": 0, "access\_token": KEY\_ACCESS, "v": 5.131})  r.json()  r.json()['response']['items'][0]  Теперь обернем все в функции для автоматического парсинга постов заданных групп  # отправляем запрос к VK API  def send\_request(domain, offset, key\_access, count):      r = requests.get("https://api.vk.com/method/wall.get",                       params={"domain": domain, "count": count, "offset": offset, "access\_token": key\_access, "v": 5.131})      if r.status\_code == 200:          return r.json()      return None  # отправляем n запросов, запрашвая по count постов  def send\_n\_requests(n\_requests, group\_domain, key\_access, count):      responses = []      offset = 0      for i in tqdm(range(n\_requests)):          response = send\_request(group\_domain, offset, key\_access, count)          if response == None:              break          responses.append(response)          offset += count      return responses  # распаковка ответа от сервера в датафрейм  def unpack\_response(response, group\_domain):      data = {"group\_domain":[], "ads": [], "post\_type": [], "text": [], "likes": []}      for item in response['response']['items']:          data['group\_domain'].append(group\_domain)          data['ads'].append(item['marked\_as\_ads'])          data['post\_type'].append(item['post\_type'])          data['text'].append(item['text'])          data['likes'].append(item['likes']['count'])      return pd.DataFrame(data)  data = pd.DataFrame(columns = ['group\_domain','ads', 'post\_type', 'text', 'likes'])  n\_requests = 100  count = 100  # пример ссылок на группы  group\_domains = ["example"," example"]  # ключ доступа из приложения вк  key\_access = KEY\_ACCESS # здесь ваш ключ  for group\_domain in group\_domains:      print(f'parsing {group\_domain}...')      responses = send\_n\_requests(n\_requests, group\_domain, key\_access, count)      for response in responses:          new\_data = unpack\_response(response, group\_domain)          data = pd.concat([data, new\_data], axis=0)  data  #сохранениедаатфрейма на диск  data.to\_csv(PATH + "groups.csv", index=False)  #оценить лексическое разнообразие  def lexical\_diversity(tokens):    return len(set(tokens))/len(tokens)  def average\_words(statuses):      total\_words = sum([len(s.split()) for s in statuses])      return total\_words/len(statuses)  print(lexical\_diversity(data.text))  print(average\_words(data.text)) Подготовка данных для анализа data = pd.read\_csv(PATH + 'groups.csv')  data.head()  Подготовим текст для будущего анализа:   * удалим строчки где нет текста - NaN или пустые строки * удалим посты, которые являются рекламой * посмотрим на распределение длины текстов - удалим слишком короткие и слишком длинные * токенизируем и лемматизируем тексты * удалим стоп-слова (предлоги и другие очень часто встречаемые слова)   #удаление NA  data.dropna(inplace=True)  # создание столбца с длиной текста в символах  data['len\_text'] = data['text'].apply(lambda x: len(x))  data.head()  # значение статистик для длины текста  data['len\_text'].describe()  data['len\_text'].quantile(0.25)  #оставим тексты в 2 и 3 квантилях длины  data = data[data['len\_text'] >= data['len\_text'].quantile(0.25)][data['len\_text'] < data['len\_text'].quantile(0.75)]  #удалим рекламные посты, если они есть  data = data[data['ads'] == 0]  #первое значение -строки, второе-колонки  data.shape  **#Токенизация**  import nltk  from nltk.tokenize import word\_tokenize  nltk.download('punkt')  text = 'Кот скушал рыбу, которая лежала на столе.'  print(word\_tokenize(text))  print([word for word in word\_tokenize(text) if word.isalpha()])  **#Лемматизация**  !pip install pymorphy3  import pymorphy3  morph = pymorphy3.MorphAnalyzer(lang='ru')  morph.parse('рыбу')  morph.parse('рыбу')[0].normal\_form  # часть речи  morph.parse('рыбу')[0].tag  **#Стоп-слова**  import nltk  nltk.download('stopwords')  from nltk.corpus import stopwords  stop\_words = stopwords.words('russian')  stop\_words += ["год", "улица", "район", "дом", "проспект", "петербург",                 "день", "остров", "человек", "савушкина", "кораблестроитель",                 "василеостровец"]  stop\_words  **#**Теперь объеденим токенизацию, лемматизацию и удаление стоп-слов в одну функцию + оставляем только существительные  def preprocess\_text(line):      tokenized = [word for word in word\_tokenize(line) if word.isalpha()]      lemmatized = [morph.parse(word)[0].normal\_form for word in tokenized if str(morph.parse(word)[0].tag).startswith('NOUN')]      lemmatized\_without\_stopwords = [word for word in lemmatized if word not in stop\_words]      return lemmatized\_without\_stopwords  # пример работы функции  preprocess\_text(text)  # применим функцию к текстам постов  data['clean\_text'] = data['text'].apply(preprocess\_text)  data.head() Анализ наиболее частотных слов в целом и по группам def word\_freq(sent\_lists):      words = dict()      for sent in sent\_lists:          for word in sent:              if word in words:                  words[word] += 1              else:                  words[word] = 1      return words  freq = word\_freq(data['clean\_text'])  sorted(freq.items(), key=lambda x: x[1], reverse=True)[:20]  По группам  group\_domains = ['example', 'example']  freq1 = word\_freq(data[data['group\_domain'] == group\_domains[0]]['clean\_text'])  sorted(freq1.items(), key=lambda x: x[1], reverse=True)[:20]  freq2 = word\_freq(data[data['group\_domain'] == group\_domains[1]]['clean\_text'])  sorted(freq2.items(), key=lambda x: x[1], reverse=True)[:20]  **#**Облако слов  from wordcloud import WordCloud  import matplotlib.pyplot as plt  %matplotlib inline  full\_text = " ".join([" ".join(x) for x in data['clean\_text']])  wordcloud = WordCloud().generate(full\_text)  plt.figure(figsize =(12, 12))  plt.imshow(wordcloud)  Для первой группы  full\_text2 = " ".join([" ".join(x) for x in data[data['group\_domain'] == 'example']['clean\_text']])  wordcloud = WordCloud().generate(full\_text2)  plt.figure(figsize =(12, 12))  plt.imshow(wordcloud)  Для второй группы  full\_text2 = " ".join([" ".join(x) for x in data[data['group\_domain'] == ' example ']['clean\_text']])  wordcloud = WordCloud().generate(full\_text2)  plt.figure(figsize =(12, 12))  plt.imshow(wordcloud) Выделение слов, наиболее специфичных для каждой группы def g2(a, b):      c, d = np.sum(a), np.sum(b)      E1 = c \* ((a + b) / (c + d))      E2 = d \* ((a + b) / (c + d))      return 2 \* ((a \* np.log(a / E1 + 1e-7)) + (b \* np.log(b / E2 + 1e-7))), E1, E2  top = sorted(freq1.items(), key=lambda x: x[1], reverse=True)[:100] + sorted(freq2.items(), key=lambda x: x[1], reverse=True)[:100]  top\_words = list(set([w for w, \_ in top]))  top\_words\_count = [freq[w] for w in top\_words]  ll\_data = pd.DataFrame(columns=['words', 'group1', 'group2', 'g2'])  ll\_data['words'] = top\_words  ll\_data['group1'] = ll\_data['words'].apply(lambda x: freq1[x] if x in freq1 else 0)  ll\_data['group2'] = ll\_data['words'].apply(lambda x: freq2[x]if x in freq2 else 0)  ll\_data['g2'], ll\_data['Expected1'], ll\_data['Expected2'], = g2(ll\_data.group1, ll\_data.group2)  ll\_data['more'] = ll\_data.apply(lambda x: 2 if x.Expected1 > x.group1 else 1, axis=1)  Слова, более специфичные для первой группы  ll\_data[ll\_data['more'] == 1].sort\_values(by='g2', ascending=False)[:20]  Слова, более специфичные для второй группы  ll\_data[ll\_data['more'] == 2].sort\_values(by='g2', ascending=False)[:20]  G2 - статистика, теперь посчитаем Logratio - effect size. Позволяет посчитать, насколько большая разница в частотах слов. Чем больше по модулю, тем более слово специфично для определенной группы.  def logratio(a, b):      return np.log((a/np.sum(a)/((b+1e-7)/np.sum(b))) + 1e-7)  ll\_data['logratio'] = logratio(ll\_data.group1, ll\_data.group2)  ll\_data.head()  v\_top = ll\_data[ll\_data['g2'] > 3.84].sort\_values(['logratio'], ascending=False)[['words', 'logratio']][:20]  p\_top = ll\_data[ll\_data['g2'] > 3.84].sort\_values(['logratio'], ascending=True)[['words', 'logratio']][:20]  Построим график наиболее специфичных слов  plt.figure(figsize=(12, 10))  plt.barh(v\_top['words'], v\_top['logratio'],color='#6C8AD5')  plt.barh(p\_top['words'].loc[::-1], p\_top['logratio'].loc[::-1], color='#00A779')  plt.xlabel('logratio - 'group\_domain'/'group\_domain', fontsize=15)  **#Named entities recognition**  pip install natasha  from natasha import (      Segmenter,      MorphVocab,        NewsEmbedding,      NewsMorphTagger,      NewsSyntaxParser,      NewsNERTagger,        PER,      NamesExtractor,      Doc  )  segmenter = Segmenter()  morph\_vocab = MorphVocab()  emb = NewsEmbedding()  morph\_tagger = NewsMorphTagger(emb)  syntax\_parser = NewsSyntaxParser(emb)  ner\_tagger = NewsNERTagger(emb)  names\_extractor = NamesExtractor(morph\_vocab)  def returnnames(data):      doc=Doc(data)      doc.segment(segmenter)      doc.tag\_ner(ner\_tagger)      jan=list()      for item2 in doc.spans:        if item2.type == 'PER':          print(item2.text)          jan.append(item2.text)          return str(jan)  import pandas as pd  df = pd.DataFrame(data)  df.text.apply(returnnames) |

1. Рассел, М. Data Mining [Текст] = Mining the social web: извлечение информации из Facebook, Twitter, Linkedin, Instagram, Github / Мэтью Рассел, Михаил Классен ; [пер. с англ.: А. Киселев]. - [3-е изд.]. - Санкт-Петербург [и др.]: Питер, 2020. - 459, [3] с.: ил. - (IT для бизнеса). [↑](#endnote-ref-1)