МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное   
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

(Самарский университет)

Институт информатики, математики и электроники

Факультет информатики  
Кафедра технической кибернетики

**Отчет по лабораторной работе № 2**

Дисциплина: «Анализ социальных сетей»

Тема: «Предобработка и классификация данных социальных сетей»

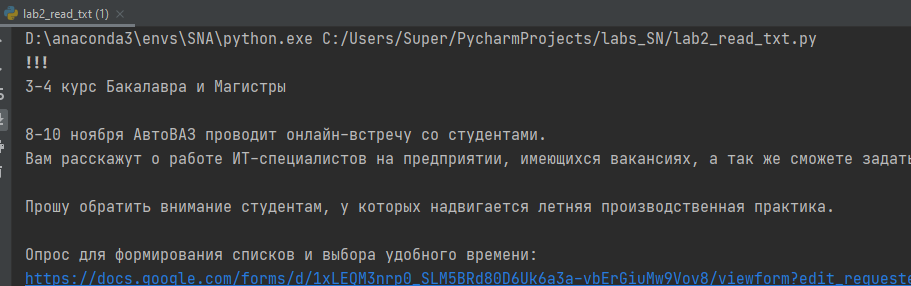
Выполнил: Овчинников И.В.

Группа: 6233-010402D

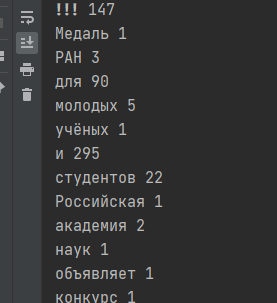
Самара 2021

Задание № 1. Предварительная обработка данных

Загрузка скачанного в ЛР 1 файла и извлечение поля “text”:



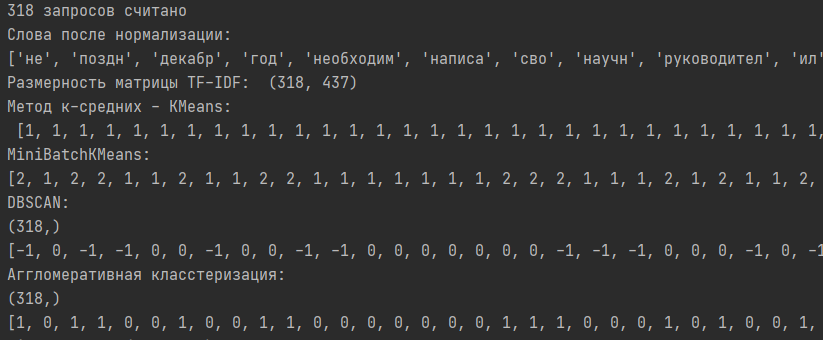
Задание № 2. Подсчет слов в собранных текстовых данных



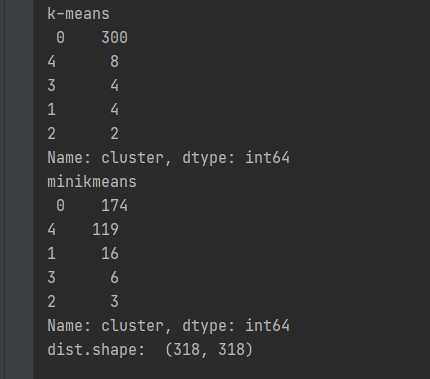
Код программы для задания №1 и 2: Приложение А.

Задание № 3. Кластеризация текстовых данных

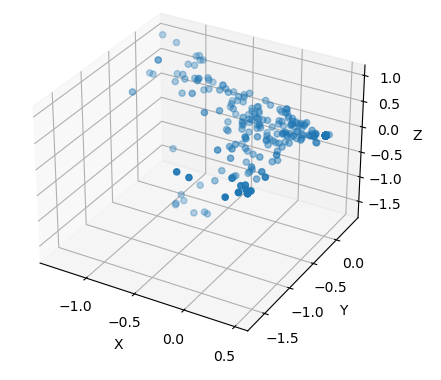
В этом пункте необходимо считать данные в массив, привести слова к начальной форме, создать матрицу весов TF-IDF и применить различные методы кластеризации над полученной матрицей:



Количество запросов, попавших в каждый кластер:



Построение трехмерного графика взаимного расположения запросов друг относительно друга:



Код программы для задания №3 представлен в приложении Б.

**Приложение А**

**Код программы для задания №1 и 2**

import json  
  
# считыввание и парсинг файла  
all\_wall = []  
file = open(r"wall\_PMI.txt")  
for line in file.readlines():  
 string = line  
 wall = json.loads(string)  
 json\_decode = json.JSONDecoder()  
 parsed\_response = json\_decode.decode(json.dumps(wall))  
 nodes = parsed\_response.get('items')  
 for node in nodes:  
 all\_wall.append(node.get("text"))  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print(all\_wall[20])  
 # подсчет слов  
 wordcount = {}  
 for wall in all\_wall:  
 for word in wall.split():  
 if word not in wordcount:  
 wordcount[word] = 1  
 else:  
 wordcount[word] += 1  
 try:  
 for wc in wordcount:  
 print(wc, wordcount[wc])  
 except Exception:  
 k = 1

**Приложение Б**

**Код программы для задания №3**

import pandas as pd  
import nltk  
import re  
import matplotlib.pyplot as plt  
from lab2\_read\_txt import \*  
from nltk.stem.snowball import SnowballStemmer  
  
print(str(len(all\_wall)) + ' запросов считано')  
stemmer = SnowballStemmer("russian")  
# nltk.download()  
  
def token\_and\_stem(text):  
 tokens = [word for sent in nltk.sent\_tokenize(text) for word in nltk.word\_tokenize(sent)]  
 filtered\_tokens = []  
 for token in tokens:  
 if re.search('[а-яА-Я]', token):  
 filtered\_tokens.append(token)  
 stems = [stemmer.stem(t) for t in filtered\_tokens]  
 return stems  
  
  
def token\_only(text):  
 tokens = [word.lower() for sent in nltk.sent\_tokenize(text) for word in nltk.word\_tokenize(sent)]  
 filtered\_tokens = []  
 for token in tokens:  
 if re.search('[а-яА-Я]', token):  
 filtered\_tokens.append(token)  
 return filtered\_tokens  
  
  
# Создаем словари (массивы) из полученных основ  
totalvocab\_stem = []  
totalvocab\_token = []  
for i in all\_wall:  
 allwords\_stemmed = token\_and\_stem(i)  
 totalvocab\_stem.extend(allwords\_stemmed)  
  
 allwords\_tokenized = token\_only(i)  
 totalvocab\_token.extend(allwords\_tokenized)  
  
print("Слова после нормализации:")  
print(totalvocab\_stem[100:150])  
  
  
stopwords = nltk.corpus.stopwords.words('russian')  
#можно расширить список стоп-слов  
stopwords.extend(['что', 'это', 'так', 'вот',  
'быть', 'как', 'в', 'к', 'на', 'бол', 'оп', 'больш', 'будт', 'быт', 'вед', 'впроч', 'всег', 'всегд', 'даж', 'друг', 'е',  
'ег', 'ем', 'есл', 'ест', 'ещ', 'зач', 'зде', 'ил', 'иногд', 'когд', 'конечн', 'куд', 'лучш', 'межд', 'мен', 'мног',  
'мо', 'можн', 'нег', 'нельз', 'нибуд', 'никогд', 'нич', 'опя', 'посл', 'пот', 'почт', 'разв', 'сво', 'себ',  
'совс', 'теб', 'тепер', 'тог', 'тогд', 'тож', 'тольк', 'хорош', 'хот', 'чег', 'чут', 'эт'])  
from sklearn.feature\_extraction.text import TfidfVectorizer, CountVectorizer  
n\_featur=200000  
tfidf\_vectorizer = TfidfVectorizer(max\_df=0.8, max\_features=10000, min\_df=0.01, stop\_words=stopwords, use\_idf=True,  
 tokenizer=token\_and\_stem, ngram\_range=(1,3))  
tfidf\_matrix = tfidf\_vectorizer.fit\_transform(all\_wall)  
print("Размерность матрицы TF-IDF: ", tfidf\_matrix.shape)  
  
num\_clusters = 5  
# Метод к-средних - KMeans  
from sklearn.cluster import KMeans  
km = KMeans(n\_clusters=num\_clusters)  
km.fit(tfidf\_matrix)  
idx = km.fit(tfidf\_matrix)  
clusters = km.labels\_.tolist()  
print("Метод к-средних - KMeans:\n", clusters)  
  
# MiniBatchKMeans  
from sklearn.cluster import MiniBatchKMeans  
mbk = MiniBatchKMeans(init='random', n\_clusters=num\_clusters) #(init='k-means++', ‘random’ or an ndarray)  
mbk.fit\_transform(tfidf\_matrix)  
mbk.fit(tfidf\_matrix)  
miniclusters = mbk.labels\_.tolist()  
print("MiniBatchKMeans:")  
print (mbk.labels\_.tolist())  
# DBSCAN  
from sklearn.cluster import DBSCAN  
db = DBSCAN(eps=0.3, min\_samples=10).fit(tfidf\_matrix)  
labels = db.labels\_  
print("DBSCAN:")  
print(labels.shape)  
print(labels.tolist())  
# Аггломеративная класстеризация  
from sklearn.cluster import AgglomerativeClustering  
agglo1 = AgglomerativeClustering(n\_clusters=num\_clusters, affinity='euclidean')  
  
answer = agglo1.fit\_predict(tfidf\_matrix.toarray())  
print("Аггломеративная класстеризация:")  
print(answer.shape)  
print(answer.tolist())  
  
#k-means  
clusterkm = km.labels\_.tolist()  
#minikmeans  
clustermbk = mbk.labels\_.tolist()  
#dbscan  
clusters3 = labels  
 #agglo  
#clusters4 = answer.tolist()  
frame = pd.DataFrame(all\_wall, index = [clusterkm])  
#k-means  
out = { 'title': all\_wall, 'cluster': clusterkm }  
frame1 = pd.DataFrame(out, index = [clusterkm], columns = ['title', 'cluster'])  
#mini  
out = { 'title': all\_wall, 'cluster': clustermbk }  
frame\_minik = pd.DataFrame(out, index = [clustermbk], columns = ['title', 'cluster'])  
print("k-means\n", frame1['cluster'].value\_counts())  
print("minikmeans\n", frame\_minik['cluster'].value\_counts())  
  
from sklearn.metrics.pairwise import cosine\_similarity  
dist = 1 - cosine\_similarity(tfidf\_matrix)  
print("dist.shape: ", dist.shape)  
  
# Метод главных компонент - PCA  
from sklearn.decomposition import IncrementalPCA  
icpa = IncrementalPCA(n\_components=2, batch\_size=16)  
icpa.fit(dist)  
demo2 = icpa.transform(dist)  
xs, ys = demo2[:, 0], demo2[:, 1]  
# PCA 3D  
from sklearn.decomposition import IncrementalPCA  
icpa = IncrementalPCA(n\_components=3, batch\_size=16)  
icpa.fit(dist)  
ddd = icpa.transform(dist)  
xs, ys, zs = ddd[:, 0], ddd[:, 1], ddd[:, 2]  
#Можно сразу примерно посмотреть, что получится в итоге  
from mpl\_toolkits.mplot3d import Axes3D  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')  
ax.scatter(xs, ys, zs)  
ax.set\_xlabel('X')  
ax.set\_ylabel('Y')  
ax.set\_zlabel('Z')  
plt.show()