Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе № 6

по курсу «Основы глубокого обучения»

Выполнил:  
студент группы 22ВВИм1:

Емелин С.В.

Приняли:

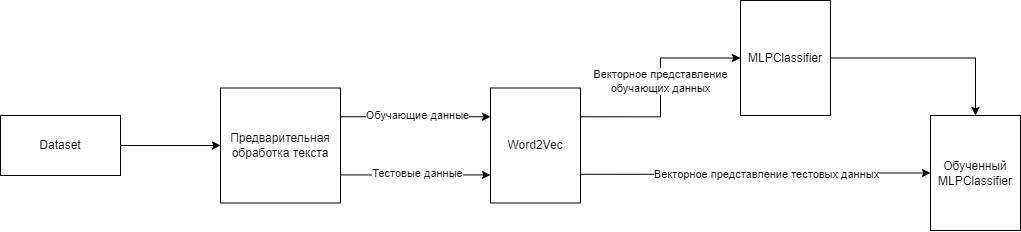
Митрохин М.А  
Панков А.А.

Пенза 2023

# Порядок выполнения работы:

# Задание:

ЗАДАНИЕ 1: создать детектор, решающий задачу классификации токсичных комментариев



Листинг программы:

# #!/usr/bin/env python

# # coding: utf-8

# # In[10]:

# from time import time

# import numpy as np

# import pandas as pd

# import re

# import nltk

# from nltk.corpus import stopwords

# from gensim.models import word2vec

# from sklearn.neural\_network import MLPClassifier

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

# # In[11]:

# df = pd.read\_csv("./data/labeled\_rutoxic.csv", delimiter=',', header=0, names=['sentence', 'label'])

# print('В наборе предложений: \n',df.shape[0])

# # In[12]:

# print('toxic:', df[df['label'] > 0]['label'].count())

# print('not toxic:', df[df['label'] < 1]['label'].count())

# # ## Разбиение на тестовые и обучающие

# # In[13]:

# X = df.iloc[:,0]#

# y = df.iloc[:,1]#

# train , test , y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3) # и отдаем 30% на тест, остальное на обучен

# # ## Обучение модели Word2Vec

# # In[14]:

# # Метод преобразования текста в слова

# def text\_to\_words(raw\_text, remove\_stopwords=False):

# 

# letters\_only = re.sub("[^0-9а-яА-Я]", " ", raw\_text)

# words = letters\_only.lower().split()

# if remove\_stopwords:

# stops = set(stopwords.words("russian"))

# meaningful\_words = [w for w in words if not w in stops]

# words = meaningful\_words

# return words

# sentences\_train = train.apply(text\_to\_words, remove\_stopwords=False)

# sentences\_test = test.apply(text\_to\_words, remove\_stopwords=False)

# print(sentences\_train[:3])

# # In[15]:

# num\_features = 300

# min\_word\_count = 40

# num\_workers = 4

# context = 20

# downsampling = 1e-3

# model = word2vec.Word2Vec(sentences\_train, workers=num\_workers, vector\_size=num\_features, min\_count = min\_word\_count, window = context, sample = downsampling)

# t = time()

# model.init\_sims(replace=True)

# print('Time to build vocab: {} mins'.format(round((time() - t) / 60, 2)))

# # In[16]:

# # получение векторного представления

# def makeFeatureVec(words, model, num\_features):

# featureVec = np.zeros((num\_features,), dtype="float32")

# nwords = 0

# index2word\_set = set(model.wv.index\_to\_key)

# 

# for word in words:

# if word in index2word\_set:

# nwords = nwords + 1

# featureVec = np.add(featureVec, model.wv[word])

# if nwords == 0:

# nwords = 1

# featureVec = np.divide(featureVec, nwords)

# return featureVec

# # получение среднего векторного простнраства для предложения

# def getAvgFeatureVecs(reviews, model, num\_features):

# reviewFeatureVecs = np.zeros((len(reviews), num\_features), dtype="float32")

# counter = 0

# for review in reviews:

# reviewFeatureVecs[counter] = makeFeatureVec(review, model, num\_features)

# counter = counter + 1

# return reviewFeatureVecs

# f\_matrix\_train = getAvgFeatureVecs(sentences\_train, model, num\_features)

# f\_matrix\_test = getAvgFeatureVecs(sentences\_test, model, num\_features)

# # ## Создание обучение MLPClassifier

# # In[17]:

# model = []

# m = MLPClassifier(solver='adam', hidden\_layer\_sizes=(30,40,30), random\_state=1)

# model.append(m)

# print(model)

# # In[33]:

# batch\_size = 200

# total\_rows = f\_matrix\_train.shape[0]

# duration = 0

# start\_train = time()

# pos = 0

# classes = [0.0, 1.0]

# # we use a partial fit approach

# while duration < 10 and pos < total\_rows:

# if pos+batch\_size > total\_rows:

# batch\_size = total\_rows-pos

# X\_p = f\_matrix\_train[pos:pos+batch\_size]

# y\_p = y\_train.values[pos:pos+batch\_size]

# model[0].partial\_fit(X\_p, y\_p, classes)

# pos = pos + batch\_size

# duration = time() - start\_train

# if pos == total\_rows:

# pos = 0

# batch\_size = 10000

# print('done')

# # In[34]:

# ## Сохранение результатов и расчет ошибки

# # In[35]:

# y\_test\_values=y\_test.values

# predicted\_results = model[0].predict\_proba(f\_matrix\_test)

# predicted\_results = np.where(predicted\_results[:,0]>predicted\_results[:,1], 0.0,1.0)

# sum\_errors =sum(y\_test\_values - predicted\_results)

# print('sum\_errors', sum\_errors)

# # In[36]:

# print(len(predicted\_results))

# print(len(y\_test\_values))

# print(len(sentences\_test))

# # In[30]:

# saved\_result = pd.DataFrame({'text':test.values,

# 'expected': y\_test\_values,

# 'predicted': predicted\_results})

# # In[31]:

# saved\_result

# # In[54]:

# saved\_result.to\_csv('result.csv', encoding='utf-8', index=False)

# # In[ ]:

# Вывод:

В ходе выполнения лабораторной работы была реализован детектор классификации токсичных комментариев