ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО Научный руководитель, Доцент департамента больших данных и информационного поиска, заведующая лабораторией «Научно-учебная лаборатория биоинформатики» факультета компьютерных наук	УТВЕРЖДАЮ Академический руководитель образовательной программы «Программная инженерия», канд. техн. наук, профессор департамента программной инженерии факультета компьютерных наук
_________________ М. С. Попцова «___» _____________ 2020 г.	__________________ В. В. Шилов «___» _____________ 2020г.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

ПРОГРАММА ДЛЯ ИССЛЕДОВАНИЯ ПРИМЕНИМОСТИ МЕТОДОВ КЛАСТЕРИЗАЦИИ И СНИЖЕНИЯ РАЗМЕРНОСТИ ДЛЯ АВТОМАТИЧЕСКОГО СРАВНЕНИЯ ЭКСПЕРИМЕНТОВ ОДНОКЛЕТОЧНОГО СЕКВЕНИРОВАНИЯ СЕРВЕР

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.13-01 12 01-1-ЛУ

Исполнитель
Студент группы БПИ 182
___________/И. С. Егоров. /
«___»__________2020 г.

Москва 2020

УТВЕРЖДЕН
RU.17701729.04.13-01 12 01-1-ЛУ

ПРОГРАММА ДЛЯ ИССЛЕДОВАНИЯ ПРИМЕНИМОСТИ МЕТОДОВ КЛАСТЕРИЗАЦИИ И СНИЖЕНИЯ РАЗМЕРНОСТИ ДЛЯ АВТОМАТИЧЕСКОГО СРАВНЕНИЯ ЭКСПЕРИМЕНТОВ ОДНОКЛЕТОЧНОГО СЕКВЕНИРОВАНИЯ СЕРВЕР

Текст программы

RU.17701729.04.13-01 12 01-1

Листов 19

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

Москва 2020

Содержание

1. Текст программы 4

1.1 10x_mtx_cutter.py 4

1.2 database_prepare.py 5

1.3 lda_theme_proportion.py 8

1.4 learn_new_model.py 10

1.5 bio/__init__.py 10

1.6 bio/admin.py 10

1.7 bio/apps.py 10

1.8 bio/models.py 11

1.9 bio/test.py 11

1.10 bio/urls/py 11

1.11 bio/views.py 11

1.12 bio_settings/setting/__init__.py 14

1.13 bio_settings/asgi.py 14

1.14 bio_settings/setting.py 15

1.15 bio_settings/urls.py 18

1.16 bio_settings/wsgi.py 18

1. Текст программы

Текст программы доступен к просмотру по ссылке: https://github.com/Igor-SeVeR/ReductionMethodsForComparingUnicellularSequencingExperiments-Coursework2/tree/master/djagnoBioInformaticsProject

1.1 10x_mtx_cutter.py

```
import sys
import random
#Uncomment when working on prod
path_to_files = sys.argv[1]
path_to_data = sys.argv[2]

file = open(path_to_data + "/matrix.mtx")
input_str = file.readline()
input_str = file.readline()
arr = input_str.split(" ")
file.close()

number_of_barcodes = int(arr[1])
number_of_genes = int(arr[0])

#Working with barcodes

print("Starting barcode reducing!")
barcodes = open(path_to_files + "/barcodes.tsv", "r")
number_of_barcodes = len(barcodes.readlines())
barcodes.close()


#Working with genes

genes = open(path_to_files + "/genes.tsv", "r")
genes_data = genes.readlines()
number_of_genes_in_file = len(genes_data) - 1
genes.close()
genes = open(path_to_files + "/genes.tsv", "w")
if (number_of_genes_in_file >= number_of_genes):
    genes.writelines(genes_data[1:number_of_genes + 1])
```

```
else:
    genes.writelines(genes_data[1:number_of_genes_in_file + 1])
    for i in range (number_of_genes - number_of_genes_in_file):
        genes.write("ENSG00001111159        AAAAAA" + '\n')
genes.close()

#Reducing_the_matrix

def cut_line(line):
    numbers = line.split()
    first_number = int(numbers[0])
    second_number = int(numbers[1])
    if (first_number > number_of_genes or second_number > number_of_barcodes):
        return False
    return True

matrix = open(path_to_files + "/matrix.mtx", "r")
data1 = matrix.readline()
data2 = matrix.readline()
cnt = 0
data = []
next_line = matrix.readline()
while (next_line):
    if (cut_line(next_line)):
        cnt += 1
        data.append(next_line)
    next_line = matrix.readline()
matrix.close()
matrix = open(path_to_files + "/matrix.mtx", "w")
matrix.writelines(data1)
matrix.writelines(str(number_of_genes) + ' ' + str(number_of_barcodes) + ' ' + str(cnt) + '\n')
matrix.writelines(data)
matrix.close()
```

1.2 database_prepare.py

```
import sys

# Library to save LDA model
import joblib

# Library to read files format
import scanpy as sc

# Library to build pyLDAvis graph
import pyLDAvis
```

```
from pyLDAvis import sklearn as sklearn_lda
from sklearn.feature_extraction.text import CountVectorizer

# Path to database
path_to_database = sys.argv[1]

# Path to lda model
path_to_lda_model = sys.argv[2]

# Path to data, on which lda model was learnt
path_to_data = sys.argv[3]

# Path to data, on which lda model was learnt
path_to_output_data = sys.argv[4]

file_name = sys.argv[5]

# Numer of genes, to get from each model topic
number_of_genes_in_topics = 200

# Opening database
data_base = open(path_to_database, "r")

# Reading database and parsing it into dictionary
lines_in_database = data_base.readline()
lines_in_database = data_base.readline()
base_dict = {}
data = []
while (lines_in_database):
    data.append(lines_in_database.split('\t'))
    lines_in_database = data_base.readline()

for i in range(len(data)):
    cell_type = data[i][4]
    cell_name = data[i][5]
    base_dict[cell_name, cell_type] = []

for i in range(len(data)):
    genes_array = data[i][8].replace(" ", "").split(',')
    for j in range(len(genes_array)):
        genes_array[j] = genes_array[j].replace("[", "")
        genes_array[j] = genes_array[j].replace("]", "")
    cell_type = data[i][4]
    cell_name = data[i][5]
    for j in range(len(genes_array)):
        base_dict[cell_name, cell_type].append(genes_array[j])
```

```
# End of reading database

# Readind lda model
lda_model = joblib.load(path_to_lda_model)
model_data = sc.read_10x_mtx(path_to_data + '/')

# Getting top-genes for each theme

genes_names = []
for row in sc.get.var_df(model_data).index:
    genes_names.append(row)

genes_by_theme = []
for topic_idx, topic in enumerate(lda_model.components_):
    topic_genes_names = ""
    topic_genes_names += " ".join([genes_names[i]
                            for i in topic.argsort()[:-number_of_genes_in_topics - 1:-1]])
    genes_by_theme.append(topic_genes_names.split(" "))

# Genes per topic are now in genes_by_theme array

answer = []
for i in range(lda_model.n_components):
    genes_intersection = {}
    step_answer = []
    for elem in (base_dict):
        cnt = 0
        for k in range(len(base_dict[elem])):
            for j in range(len(genes_by_theme[i])):
                if (base_dict[elem][k] == genes_by_theme[i][j]):
                    cnt += 1
        name_and_type = []
        name_and_type.append(elem[0])
        name_and_type.append(elem[1])
        genes_intersection[float(cnt) / float(len(base_dict[elem]))] = name_and_type
    sorted_values = sorted(genes_intersection, reverse=True)
    number = 0
    for j in range(5):
        step_answer.append(genes_intersection[sorted_values[j]])
    answer.append(step_answer)

file = open(path_to_output_data + "/" + file_name + ".txt", "w")
file.write("Here are predicted cells by genes significance in each topic." + '\n')
for i in range(len(answer)):
    file.write("Topic " + str(i) + '\n')
    for j in range(len(answer[i])):
        file.write("Cell name: " + answer[i][j][0] + "; Cell type: " + answer[i][j][1] + '\n')
file.close()

# Building pyLDAvis graph

data_for_pyLDA = []

for row in sc.get.var_df(model_data).index:
    data_for_pyLDA.append(row)

vectorizer_n = CountVectorizer(lowercase=False, token_pattern=r"(?u)\w+\.\w+|\w+\-
\w+|\w+|\.\w+|\w+\.|\w+\-|\-\w+")
count_data = vectorizer_n.fit_transform(data_for_pyLDA)

LDAvis_prepared = sklearn_lda.prepare(lda_model, count_data, vectorizer_n)
pyLDAvis.save_html(LDAvis_prepared, path_to_output_data + '/' + file_name + '.html')
```

1.3 lda_theme_proportion.py

```
import sys

#Library to save LDA model
import joblib

#Library to read files format
import scanpy as sc

#Library, which contains LDA model trainer
from sklearn.decomposition import LatentDirichletAllocation as LDA

#Libary for plotting
import pandas as pd
import matplotlib.pyplot as plot
import numpy as np

#Reading data
path_to_model = sys.argv[1]
path_to_model_data = sys.argv[2]
path_to_users_data = sys.argv[3]
path_to_images = sys.argv[4]
file_name = sys.argv[5]

#Reading LDA model
lda_model = joblib.load(path_to_model)
model_data = sc.read_10x_mtx(path_to_model_data + '/')
data_out = model_data.X
```

```
lda_output = lda_model.transform(data_out)

#Creating a bar plot of percentage of cell divisions by topic in trained model
df_cell_topic = pd.DataFrame(np.round(lda_output, 2))
df_cell_dominant_topic = np.argmax(df_cell_topic.values, axis=1)
df_cell_topic['dominant_topic'] = df_cell_dominant_topic
number_of_dominant_topics = df_cell_topic['dominant_topic'].value_counts()
topics = []
for i in range (lda_model.n_components):
    topics.append(i)
values = []
for i in range (lda_model.n_components):
    try:
        values.append(number_of_dominant_topics[i] / float(data_out.shape[0]) * 100)
    except Exception:
        values.append(0)

# Dictionary loaded into a DataFrame
dataFrame = pd.DataFrame(data={"Topic":topics, "Percentage":values })

# Draw a vertical bar chart
dataFrame.plot.bar(x="Topic", y="Percentage", rot=70, title="")
plot.savefig(path_to_images + "/" + file_name + "1.png", dpi = 300)

#Reading data
data = sc.read_10x_mtx(path_to_users_data + '/')

#Applying data on a trained model to get cells by topics
data_test = data.X
topic_probability_scores = lda_model.transform(data_test)
df_cell_topic_user = pd.DataFrame(np.round(topic_probability_scores, 2))
df_cell_dominant_topic_user = np.argmax(df_cell_topic_user.values, axis=1)
df_cell_topic_user['dominant_topic'] = df_cell_dominant_topic_user
number_of_dominant_topics_user = df_cell_topic_user['dominant_topic'].value_counts()
topics = []
for i in range (lda_model.n_components):
    topics.append(i)
values = []
for i in range (lda_model.n_components):
    try:
        values.append(number_of_dominant_topics_user[i] / float(data_test.shape[0]) * 100)
    except Exception:
        values.append(0)

# Dictionary loaded into a DataFrame
dataFrame = pd.DataFrame(data={"Topic":topics, "Percentage":values })
```

```
# Draw a vertical bar chart
dataFrame.plot.bar(x="Topic", y="Percentage", rot=70, title="")
plot.savefig(path_to_images + "/" + file_name + "2.png", dpi = 300)
```

1.4 learn_new_model.py

```
import sys

#Library to save LDA model
import joblib

#Library to read files format
import scanpy as sc

#Library, which contains LDA model trainer
from sklearn.decomposition import LatentDirichletAllocation as LDA

#Reading data
lda_model_name = sys.argv[1]
path_to_files = sys.argv[2]
number_of_themes = int(sys.argv[3])
a_Data = sc.read_10x_mtx(path_to_files + '/')
path_to_model = sys.argv[4] + '/'

#Learning new model
data = a_Data.X
lda_model = LDA(n_components=number_of_themes, learning_method='online')
lda_output = lda_model.fit_transform(data)

#Saving new model
joblib.dump(lda_model, path_to_model + lda_model_name + '.jl')
```

1.5 bio/__init__.py

1.6 bio/admin.py

```
from django.contrib import admin

# Register your models here.
```

1.7 bio/apps.py

```
from django.apps import AppConfig


class BioConfig(AppConfig):
    name = 'bio'
```

1.8 bio/models.py

```
from django.db import models

# Create your models here.
```

1.9 bio/test.py

```
from django.test import TestCase

# Create your tests here.
```

1.10 bio/urls/py

```
from django.urls import path

from .views import UseBuiltModel, BuildModel

app_name = "bio"
# app_name will help us do a reverse look-up latter.
urlpatterns = [
    path('get_mtx_data', UseBuiltModel.as_view()),
    path('build_model_and_get_mtx_data', BuildModel.as_view())
]
```

1.11 bio/views.py

```
import os
import zipfile
import subprocess

from django.core.files.base import ContentFile
from django.core.files.storage import default_storage
from rest_framework.response import Response
from rest_framework.views import APIView
from bio_settings.settings import SERVER_URL

import sys

#Library to save LDA model
import joblib

#Library to read files format
import scanpy as sc

#Library, which contains LDA model trainer
from sklearn.decomposition import LatentDirichletAllocation as LDA

#Libary for plotting
import pandas as pd
import matplotlib.pyplot as plot
import numpy as np
import pyLDAvis
import random

def clearData(filePath):
    folder = filePath
    for the_file in os.listdir(folder):
        next_file_path = os.path.join(folder, the_file)
        try:
            if os.path.isfile(next_file_path):
                os.unlink(next_file_path)
        except Exception as e:
            print()

def useBloodModel(file):
    clearData("media/plots")
    clearData("media/OutputData")
    z = zipfile.ZipFile("media/" + file, 'r')
    z.extractall(path="media/archives")
    data = sc.read_10x_mtx("media/archives/")
    input_number = open("media/systemFiles/number_of_questions.txt", "r")
    number = int(input_number.readline()) + 1
    input_number.close()
    input_number = open("media/systemFiles/number_of_questions.txt", "w")
    input_number.write(str(number))
    input_number.close()
    file_name = "model_data" + str(number)
    p = subprocess.Popen("python Scripts/10x_mtx_cutter.py media/archives
media/TrainData/blood", stdout=subprocess.PIPE, shell=True)
    p.wait()
    p = subprocess.Popen("python Scripts/lda_theme_proportion.py
media/models/blood_lda_model.jl media/TrainData/blood media/archives media/plots " +
file_name, stdout=subprocess.PIPE, shell=True)
    p.wait()
    p = subprocess.Popen("python Scripts/database_prepare.py
media/DataBase/Human_cell_markers.txt media/models/blood_lda_model.jl
media/TrainData/blood media/OutputData " + file_name,
        stdout=subprocess.PIPE, shell=True)
    p.wait()
    return {"Predicted cells by clusters:": 'media/OutputData/' + file_name + '.txt', "Interactive
pyLDAvis graph:": 'media/OutputData/' + file_name + '.html', "The distribution of barcodes in
model.": 'media/plots/' + file_name +'1.png', "The distribution of barcodes in users data due to
model.": 'media/plots/' + file_name + '2.png'}

def buildModelAndUseIt(file):
    clearData("media/plots")
    clearData("media/OutputData")
    z = zipfile.ZipFile("media/" + file, 'r')
    z.extractall(path="media/archives")
    data = sc.read_10x_mtx("media/archives/")
    input_number = open("media/systemFiles/number_of_questions.txt", "r")
    number = int(input_number.readline()) + 1
    input_number.close()
    input_number = open("media/systemFiles/number_of_questions.txt", "w")
    input_number.write(str(number))
    input_number.close()
    file_name = "users_data" + str(number)
    p = subprocess.Popen("python Scripts/learn_new_model.py users_lda_model media/archives
10 media/models", stdout=subprocess.PIPE, shell=True)
    p.wait()
    p = subprocess.Popen("python Scripts/lda_theme_proportion.py
media/models/users_lda_model.jl media/archives media/archives media/plots " + file_name,
stdout=subprocess.PIPE, shell=True)
    p.wait()
    p = subprocess.Popen("python Scripts/database_prepare.py
media/DataBase/Human_cell_markers.txt media/models/users_lda_model.jl media/archives
media/OutputData " + file_name, stdout=subprocess.PIPE, shell=True)
    p.wait()
    return {"Predicted cells by clusters:": 'media/OutputData/' + file_name + '.txt', "Interactive
pyLDAvis graph:": 'media/OutputData/' + file_name + '.html', "The distribution of barcodes in
model.": 'media/plots/' + file_name + '1.png'}


def saveFile(file, pathToSave):
    data = file.read()
    path = default_storage.save(pathToSave, ContentFile(data))
    return path


def formUrlOnServer(pathes):
    for key in pathes:
        pathes[key] = SERVER_URL + pathes[key]
    return pathes


class UseBuiltModel(APIView):
    def post(self, request):
        clearData("media/archives")
        file = request.FILES['photo']
        pathToArch = saveFile(file, 'archives/arch.zip')
        pathes = useBloodModel(pathToArch)
        pathes = formUrlOnServer(pathes)
        print(pathToArch)
        return Response(pathes)

class BuildModel(APIView):
    def post(self, request):
        clearData("media/archives")
        file = request.FILES['photo']
        pathToArch = saveFile(file, 'archives/arch.zip')
        pathes = buildModelAndUseIt(pathToArch)
        pathes = formUrlOnServer(pathes)
        print(pathToArch)
        return Response(pathes)
```

1.12 bio_settings/setting/__init__.py

1.13 bio_settings/asgi.py

```
"""
ASGI config for bio_settings project.

It exposes the ASGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/3.0/howto/deployment/asgi/
"""
```

```
import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'bio_settings.settings')

application = get_asgi_application()
```

1.14 bio_settings/setting.py

```
"""
Django settings for bio_settings project.

Generated by 'django-admin startproject' using Django 3.0.5.

For more information on this file, see
https://docs.djangoproject.com/en/3.0/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.0/ref/settings/
"""

import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'tjp*cv6np97pabz1j^8w37q4#@5i-60626ivu^uc&t2f7%!1j0'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'corsheaders',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'bio'
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
]

CORS_ORIGIN_ALLOW_ALL = True  # If this is used then `CORS_ORIGIN_WHITELIST`
will not have any effect
CORS_ALLOW_CREDENTIALS = True
CORS_ORIGIN_WHITELIST = [
    'http://localhost:63344',
]  # If this is used, then not need to use `CORS_ORIGIN_ALLOW_ALL = True`
CORS_ORIGIN_REGEX_WHITELIST = [
    'http://localhost:53344',
]

ROOT_URLCONF = 'bio_settings.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')]
        ,
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.media'
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'bio_settings.wsgi.application'

# Database
# https://docs.djangoproject.com/en/3.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}

# Password validation
# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/
```

```
STATIC_URL = '/static/'
MEDIA_DIR = os.path.join(BASE_DIR, 'media')
MEDIA_ROOT = MEDIA_DIR
MEDIA_URL = '/media/'

SERVER_URL = 'http://localhost:8000/'
```

1.15 bio_settings/urls.py

```
"""bio_settings URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.conf.urls.static import static
from django.contrib import admin
from django.urls import path, include

from bio_settings import settings

urlpatterns = [
                  path('admin/', admin.site.urls),
                  path('api/', include('bio.urls'))
              ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

1.16 bio_settings/wsgi.py

```
"""
WSGI config for bio_settings project.

It exposes the WSGI callable as a module-level variable named ``application``.
```

```
For more information on this file, see
https://docs.djangoproject.com/en/3.0/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'bio_settings.settings')

application = get_wsgi_application()
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.507140-01 ТЗ 01-1				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата