УТВЕРЖДАЮ

Заведующий кафедрой В. А. Камаев

_____ _____

(подпись) (расшифровка подписи)

«_____»_____ 2016г.

ЗАДАНИЕ

на производственную практику

Студенту Чечеткину И. А. группы САПР-1.1п

1) Реализация алгоритма Mean Shift с использованием библиотеки scikit-learn.
2) Реализация и тестирование метрики, основанной на прокладывании маршрутов на карте OpenStreetMap.
3) Внедрение метрики в модули с алгоритмами K-Means и Mean Shift.

Руководитель практики _____ / _____

Министерство образования и науки РФ
Государственное образовательное учреждение
Высшего профессионального образования
«Волгоградский государственный технический университет»
Кафедра «САПР и ПК»

ДНЕВНИК

по производственной практике 2016г.

Студента
Фамилия Чечеткина Имя Ильи
Отчество Александровича
Факультет ФЭВТ курс 1 группа САПР-1.1п

Специальность
Информационное и программное обеспечение
автоматизированных систем
База проведения практики _____

КАЛЕНДАРНЫЕ СРОКИ ПРАКТИКИ:
По учебному плану: начало _____ конец _____
Дата прибытия на практику: «____» _____ 2016г.
Дата выбытия: «____» _____ 2016г.

Индивидуальное задание: _____
_____

РУКОВОДИТЕЛЬ ПРАКТИКИ
Кафедра _____ Должность _____
Фамилия _____ Имя _____
Отчество _____

Волгоград, 2016

Дата	Работа, выполненная студентом	Отметка руководителя, подпись
29.06	Получение задания	
30.06 – 3.07	Реализация алгортима Mean Shift	
4.07 – 6.07	Установка, настройка Project OSRM	
7.07 – 9.07	Реализация метрики, основанной на прокладывании маршрутов на карте OpenStreetMap	
10.07	Внедрение метрики в алгоритм K-Means, тестирование	
11.07 – 21.07	Внедрение метрики в алгоритм Mean Shift, тестирование	
22.07 – 24.07	Реализация кэширования рассчитанных дистанций	
25.07 – 27.07	Тестирование работы алгоритма Mean Shift с внедренной метрикой	

«____» _____ 2016г.

Студент Чечеткин И. А. _____ (_____)

Руководитель _____ (_____)

Министерство образования и науки РФ
Государственное образовательное учреждение
Высшего профессионального образования
«Волгоградский государственный технический университет»
Кафедра «САПР и ПК»

ОТЧЁТ

по производственной практике 2016г.

Студента
Фамилия Чечеткина Имя Ильи
Отчество Александровича
Факультет ФЭВТ курс 1 группа САПР-1.1п

Тема дипломной работы: Метод кластеризации предпочтений жителей города по перемещению.

РУКОВОДИТЕЛЬ ПРАКТИКИ
Кафедра _____ Должность _____
Фамилия _____ Имя _____
Отчество _____

«____»_____ 2016г.

Волгоград 2016

СОДЕРЖАНИЕ

1 ВВЕДЕНИЕ

Цель данной работы заключается в реализации метрики, основанной на прокладывании маршрутов на картах OpenStreetMap, а также внедрение этой метрики в работу алгоритмов K-Means и Mean Shift. В данной работе для реализации прокладывания маршрутов на карте используется фреймворк Project OSRM.

Фреймворк OSRM (*Open Source Routing Machine*) является свободным и работает с картами сообщества OpenStreetMap. Фреймворк широко используется для написания приложений для различных навигаторов, поскольку позволяет не только находить кратчайший маршрут между двумя точками, но и выдавать инструкции по передвижению.

Алгоритм Mean Shift (среднего сдвига) является центроидным алгоритмом кластеризации, зачастую применяется в задачах низкоуровневого компьютерного зрения вследствие легкости и эффективности. Алгоритм среднего сдвига основан на ядре оценки плотности.

Алгоритм K-Means (K-средних) является центроидным алгоритмом кластеризации. Широко применяется в задачах классификации и кластеризации, а также для задач машинного зрения и глубокого обучения. Алгоритм стремится минимизировать суммарное квадратичное отклонение точек кластеров от центров этих кластеров. Проблемой алгоритма является то, что результат кластеризации сильно зависит от выбора начальных центров кластеров, а также то, что число кластеров необходимо знать заранее.

Scikit-learn – открытая python-библиотека машинного обучения. В ней содержатся различные алгоритмы классификации, кластеризации и регрессионного анализа.

2 АЛГОРИТМ MEAN SHIFT

2.1 Суть алгоритма

Алгоритм кластеризации Mean Shift направлен на обнаружение сгустков плотности среди заданного набора образцов. Он является центроидным, то есть работает посредством обновления положения центров плотности. На каждой итерации алгоритма высчитывается среднее взвешенное значение плотности образцов в заданной области с использованием особой функции, называемой ядром (2.1):

$$m(x) = \frac{\sum_{x_i \in N(x)} K(x_i - x) x_i}{\sum_{x_i \in N(x)} K(x_i - x)}, \tag{2.1}$$

где $m(x)$ – среднее взвешенное значение плотности в заданной области, $N(x)$ – соседние к x образцы, причем для них $K(x) \neq 0$, $K(x_i - x)$ – ядро.

В ядре содержится единственный параметр, используемый в данном алгоритме (2.2):

$$K(x_i - x) = k\left(\frac{\|x_i - x\|^2}{h^2}\right). \tag{2.2}$$

Параметр h, называемый «пропускной способностью» (*bandwidth*), в данном алгоритме регулирует размер области, в которой ищутся соседние образцы и рассчитывается среднее значение плотности. Функция k здесь – профиль ядра.

Обычно в качестве ядра используют гауссианы (2.3):

$$K(x_i - x) = e^{-c\|x_i - x\|^2}. \tag{2.3}$$

После расчета $m(x)$ алгоритм заменяет положение центроидов x на рассчитанные $m(x)$, и переходит к следующей итерации.

В период пост-обработки центроиды фильтруются, отбрасываются дубликаты и близлежащие центроиды, после чего формируется окончательный набор центроидов. [2–4]

Сильные стороны алгоритма:

- весь процесс кластеризации зависит только от одного параметра h,

имеющего реальный физический смысл;

- нет зависимости от размерности пространства и конкретной метрики расстояний;
- не предполагает конкретных форм кластеров.

Слабые стороны алгоритма:

- выбор параметра h весьма нетривиален, зачастую требует адаптивной подстройки во время работы алгоритма; а неправильный выбор этого параметра может приводить к схопыванию кластеров или генерации дополнительных кластеров, в которых плотность образцов невелика;
- алгоритм является очень ресурсоемким: в общем он требует $O(kN^2)$ операций, где N – число образцов, k – среднее количество итераций на один образец.

На рисунках 2.1 и 2.2 для сравнения приведены соответственно результаты кластеризации одной и той же выборки алгоритмами K-Means и Mean Shift.

2.2 Реализация

Алгоритм Mean Shift был реализован на языке программирования Python с использованием библиотек numpy и scikit-learn [1] Также при написании используются ранее написанные модули ClusteringMachine (прил. 2) и DataCollector (прил. 1). Реализация представляет собой подключаемый модуль (прил. 3), который можно запускать и как обычный Python-скрипт.

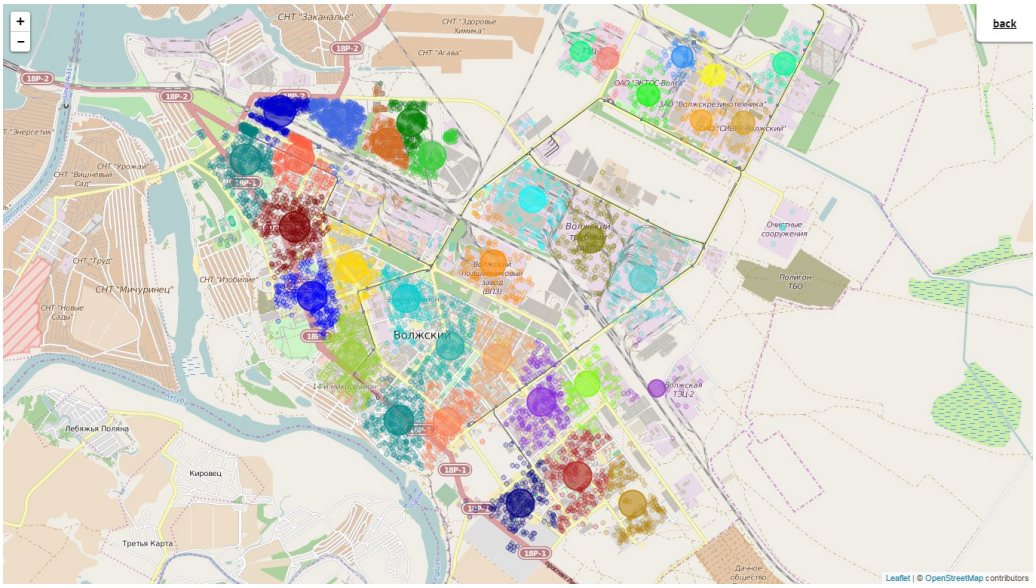

Рисунок 2.1 — Результаты кластеризации алгоритмом K-Means

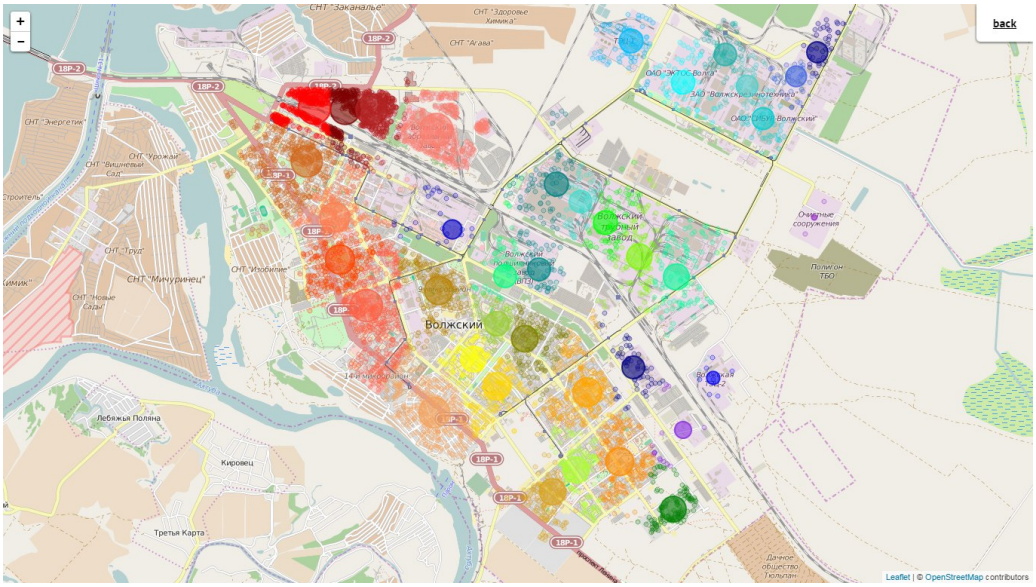

Рисунок 2.2 — Результаты кластеризации алгоритмом Mean Shift

3 ПОСТРОЕНИЕ МАРШРУТОВ

Для нахождения расстояния между двумя точками с помощью построения маршрута между ними используется фреймворк Project-OSRM (`http://project-osrm.org`).

3.1 Установка и настройка Project-OSRM

Подробная информация об установке и настройке OSRM на различные ОС находится в документации проекта [5].

Для ОС Fedora. Произведите установку git, cython, cmake и gcc-c++:

```
$ sudo yum install git cython cmake gcc-c++
```

Установите необходимые зависимости:

```
$ sudo yum install libxml2-devel boost-devel boost-regex bzip2-devel \
  libzip-devel stxxl-devel protobuf-devel protobuf-lite protobuf-lite-devel \
  lua.x86_64 lua-devel.x86_64 luajit.x86_64 luajit-devel.x86_64 \
  luabind.x86_64 luabind-devel.x86_64 expat expat-devel tbb tbb-devel
```

Скачайте и установите библиотеку для чтения и записи PBF файлов:

```
$ git clone https://github.com/scrosby/OSM-binary.git
$ cd OSM-binary
$ cmake .
$ sudo make install
```

Скачайте и установите Project-OSRM:

```
$ git clone https://github.com/DennisOSRM/Project-OSRM.git
$ mkdir -p Project-OSRM/build
$ cd Project-OSRM/build
$ cmake ..
$ make
$ sudo make install
```

Скачайте и установите серверную часть Project-OSRM:

```
$ git clone https://github.com/Project-OSRM/osrm-backend.git
$ cd osrm-backend
$ mkdir -p build
$ cd build
$ cmake ..
$ make
```

Не выходя из директории *build* сделайте символическую ссылку на скоростной профиль и директорию с библиотеками для скоростных профилей. Стандартным скоростным профилем является профиль автомобиля:

```
$ ln -s ../profiles/car.lua profile.lua
$ ln -s ../profiles/lib/
```

Распакуйте необходимую OpenStreetMap-карту:

```
$ osrm-extract map.osm
```

Результатом выполнения команды будет файл *map.osrm*. Выполните обработку данных карты:

```
$ osrm-prepare map.osrm
```

Результатом будет набор из 9 файлов, каждый из которых содержит определенную часть загруженной карты.

Запускаем сервер Project-OSRM:

```
$ osrm-routed map.osrm
```

3.2 Реализация метрики

Реализуемая метрика была названа *route* (прил. 4). Модуль содержит класс *route*, в котором есть три функции: *start*, *stop* и *route_distance*.

Первая функция запускает процесс *osrm-routed* с нужным файлом карты.

Вторая функция останавливает запущенную OSRM машину.

Третья функция возвращает дистанцию между двумя точками, переданных ей в качестве параметров.

Было реализовано локальное хранение данных о построенных маршрутах в виде текстового файла, подгружаемого при старте OSRM машины и перезаписываемого при ее остановке. Это позволяет сократить время на кластеризацию, поскольку при повторении запроса происходит извлечения значения расстояния из памяти без повторного построения маршрута.

4 ВНЕДРЕНИЕ МЕТРИКИ

4.1 Внедрение метрики в алгоритм K-Means

Для внедрения метрики в модуль с алгоритмом K-Means необходимо импортировать написанный модуль метрики; добавить поле *route_* классу, который отвечает за расчет расстояний; прописать этот расчет и прописать запуск и остановку OSRM машины (прил. 5).

4.2 Внедрение метрики в алгоритм Mean Shift

Для внедрения метрики в модуль с алгоритмом Mean Shift также необходимо импортировать написанный модуль метрики; добавить поле *route_* классу, который отвечает за расчет расстояний; прописать этот расчет и прописать запуск и остановку OSRM машины:

```
...
from route import route

class MeanShiftClusteringMachine(ClusteringMachine):

...
    route_ = route()
    def __init__(self, X, useEstimateBandwidth = True, binSeeding = True):
        self.X = X
        self.route_.start()
        self.fit_time = time.time()

...
    def fit(self):
        """ Perform clustering.

        """
        # perform clustering
        self.cluster_instance.fit(self.X)
        # stop osrm machine
        self.route_.stop()

...
```

Теперь при запуске кластеризации с помощью модуля Mean Shift будет производится запуск и остановка OSRM машины, однако будет использоваться прежняя метрика – декартова, поскольку сам алгоритм находится внутри подключаемой библиотеки Scikit-learn.

Скачиваем библиотеку Scikit-learn:

```
$ git clone https://github.com/scikit-learn/scikit-learn.git
```

Копируем в директорию модуль метрики, переходим в директорию *sklearn*:

```
$ cp route.py scikit-learn/sklearn/
$ cd scikit-learn/sklearn/
```

Алгоритм Mean Shift находится в файле *cluster/mean_shift_.py*. Добавляем выбор метрики: параметром прописываем *metric = 'route'*.

Теперь нужно написать саму метрику, которую алгоритм мог бы выбрать. В файле *metrics/pairwise.py* добавляем новую метрику *'route'* и прописываем в списках доступных метрик новую.

Для того, чтобы алгоритм поиска соседей также мог пользоваться метрикой, прописываем метрику в файл *neighbors/dist_metric.pyx*. Для сохранения сделанных изменений в этом файле необходимо скомпилировать с-библиотеку:

```
$ cd neighbors
$ cython dist_metric.pyx
```

Добавляем прописанную метрику в список доступных метрик в файле с алгоритмом поиска соседей *neighbors/ball_tree.pyx*.

Если при первом запуске начальные центры кластеров становятся в точку с координатами `(0, 0)`, то в файле с алгоритмом Mean Shift (*cluster/mean_shift_.py*) необходимо изменить строчку:

```
binned_point = np.round(point / bin_size)
```

В этом месте для упрощения расчетов координаты делятся на размер рассматриваемой области и округляются. Для задачи кластеризации географических координат размер области на 1-2 порядка превышает значения координат. Поэтому необходимо оставить определенное количество цифр после запятой:

```
binned_point = np.round(point / bin_size, decimals = 5)
```

Собираем и устанавливаем измененную библиотеку:

```
$ cd ..
$ python setup.sh build
$ sudo python setup.sh install
```

Теперь вместо библиотеки *scikit-learn* установлена ее измененная версия, в которую добавлена новая метрика; таким образом, модуль с алгоритмом Mean Shift теперь также поддерживает метрику *route*.

5 ВЫВОДЫ

При использовании реализованной метрики заметно повысилась точность кластеризации географических данных – учитывается ландшафт местности и естественные препятствия: железные дороги, реки, овраги и балки. Однако, существенно понизилась скорость выполнения кластеризации: на один запрос расчета дистанции между двумя точками требуется примерно 10 мсек. Таким образом, на то, чтобы рассчитать только расстояния между всеми точками тестовой выборки (6000 точек) необходимо потратить:

$$t = \frac{6000!}{2 * 5998!} \cdot 10 = 179970000 \text{ мсек} \approx 50 \text{ ч.}$$

Использование локального хранения рассчитанных значений ускоряет процесс: на полную кластеризацию 200 точек (от 25 до 50 итераций) алгоритм K-Means тратит от 60 до 180 секунд, тогда как только на одну итерацию (при 40 кластерах) должно уходить около 80 секунд.

На рисунках 5.1 – 5.3 приведены результаты кластеризации 200 точек алгоритмами K-Means и Mean Shift.

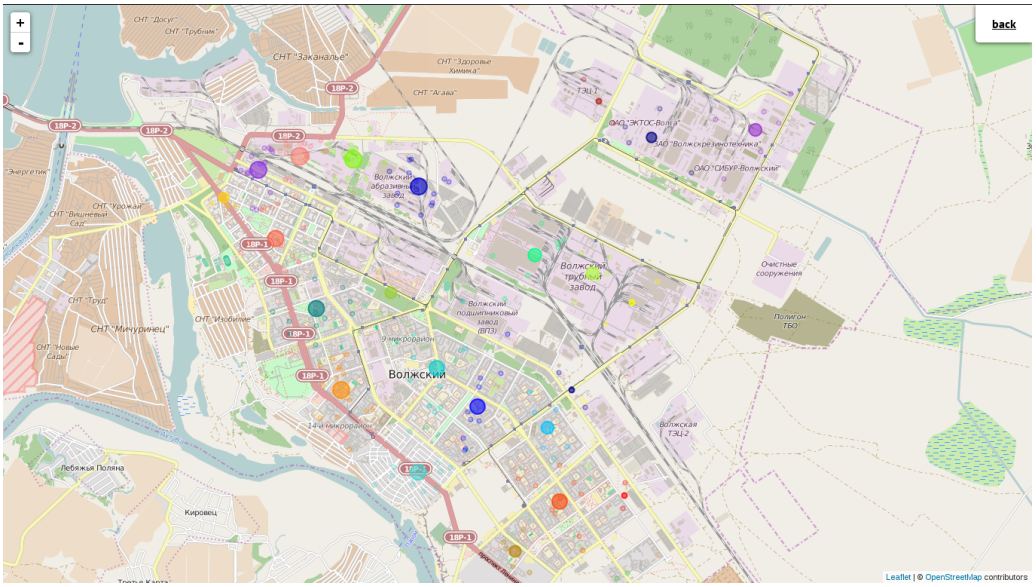

Рисунок 5.1 — Результаты кластеризации алгоритмом K-Means с евклидовой метрикой

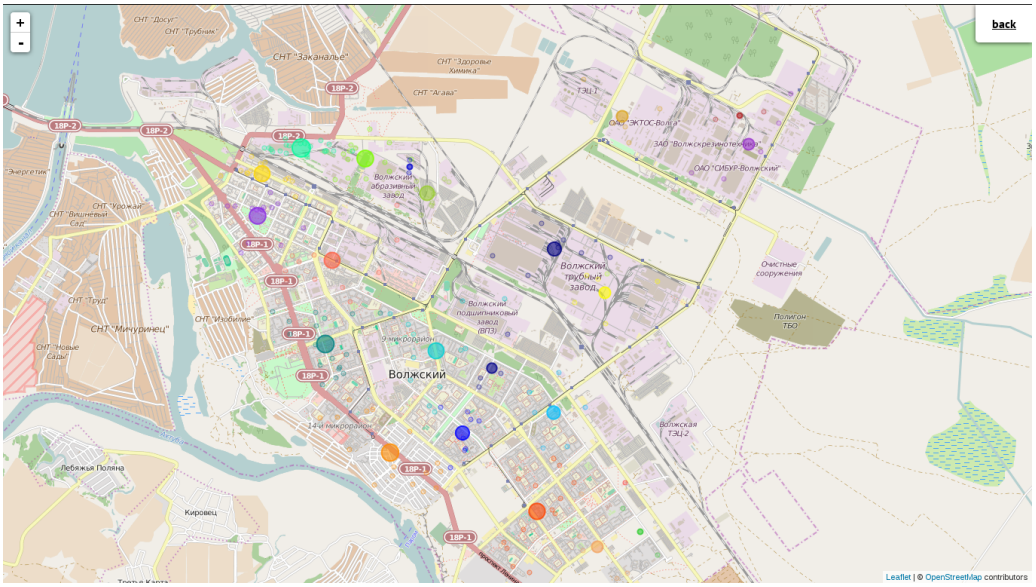

Рисунок 5.2 — Результаты кластеризации алгоритмом K-Means с метрикой route

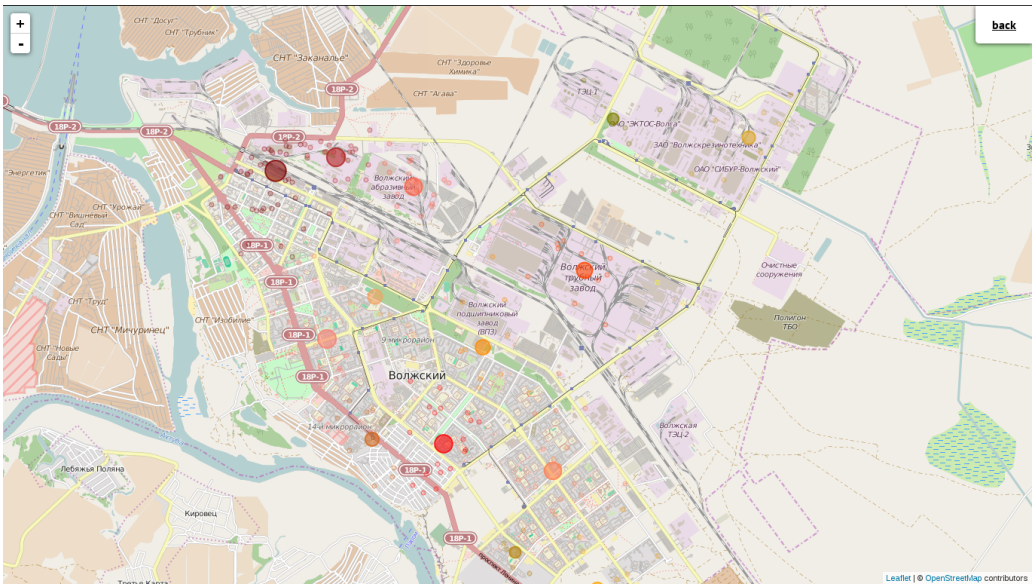

Рисунок 5.3 — Результаты кластеризации алгоритмом Mean Shift с евклидовой метрикой

6 РЕЗУЛЬТАТЫ

В результате выполнения можно заключить следующее:

1) Проведен анализ различных источников информации по теме работы.
2) Произведена реализация модуля кластеризации алгоритмом Mean Shift.
3) Произведена установка фреймворка Project OSRM.
4) Произведена реализация модуля метрики, основанной на построении маршрутов.
5) Произведено внедрение метрики в модуль с алгоритмом K-Means и его тестирование.
6) Произведено внедрение метрики в модуль с алгоритмом Mean Shift и начато его тестирование.

Тестирование работы библиотеки Scikit-learn с внедренной метрикой *route* на данный момент не окончено.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1 Documentation scikit-learn: machine learning in Python — scikit-learn documentation [Electronic resource] — Available at:
http://scikit-learn.org/stable/documentation.html

2 Машинное обучение (курс лекций, К.В.Воронцов) [Электронный ресурс] — Режим доступа:
http://www.machinelearning.ru/wiki/index.php?title=Машинное_обучение_(курс_лекций%2C_К.В.Воронцов)

3 Цветная сегментация изображения с помощью глобальной информации и локальной однородности [Электронный ресурс] — Режим доступа:
http://www.uran.donetsk.ua/~masters/2006/fvti/poltava/library/article2.htm

4 Comaniciu, D. Mean Shift: A Robust Approach Toward Feature Space Analysis [Electronic resource] / D. Comaniciu, P. Meer. — Available at:
https://courses.csail.mit.edu/6.869/handouts/PAMIMeanshift.pdf

5 Home · Project-OSRM/osrm-backend Wiki [Electronic resource] — Available at:
https://github.com/Project-OSRM/osrm-backend/wiki/

ПРИЛОЖЕНИЕ

1 Модуль DataCollector

```
import json
import numpy as np
import matplotlib.pyplot as plt

class DataCollector():
    """ Collects points from various sources.

    Attributes
    ----------
    data : array, [n_points, n_dimensions]
        Coordinates of points.

    header : string
        File header. May contain borders used in some clustering algorithms.
    """

    data = np.array([])
    header = ''

    def uploadFromTextFile(self, filename, header = False, delimiter = ','):
        """ Upload data from text file.

        Parameters
        ----------
        filename : string path
            Name of data source file.
        header : boolean, default False
            If true, first line of file will be used as header.
        delimiter : string, default ','
            Sets source file data delimiter.
        """
        try:
            with open(filename) as file_:
                # create an empty array for points
                data_ = np.empty((0, 2), float)
                # if header set to True read first line to self.header
                if header:
                    self.header = file_.readline()
                # for each line in file read latitude and longitude of point
                # and record them to data array
                for line in file_:
                    lat, lon = [float(n) for n in line.split(delimiter)[1:]]
                    data_ = np.append(data_, [[lat, lon]], axis=0)
```

```
            self.data = data_
        # if error while reading file, print error message and clear data array
        except IOError as e:
            print('{}'.format(e))
            self.data = np.array([])

    def getData(self):
        """ Get collected data.

        Returns
        -------
        data : array, [n_points, n_dimensions]
            Coordinates of points
        """
        return self.data

    def plotData(self):
        """ Plot collected data.

        Notes
        -----
        Uses matplotlib.pyplot for showing data.
        """
        plt.figure()
        plt.scatter(self.data[:, 0], self.data[:, 1])
        plt.show()

    def getHeader(self):
        """ Get collected header.

        Returns
        -------
        header : string
            File header.
        """
        return self.header

    def exportToTextFile(self, filename):
        """ Record data to text file.

        Parameters
        ----------
        filename : string path
            Recording file name.

        Notes
        -----
        Uses JSON data format.
```

```
        """
        try:
            with open(filename, 'w') as file_:
                json.dump(self.data.tolist(), file_)
        except IOError as e:
            print('{}'.format(e))
```

2 Модуль ClusteringMachine

```
import json
import numpy as np
import matplotlib.pyplot as plt

class ClusteringMachine():
    """ General class for clustering machine.

    Parameters
    ----------
    X : array, [n_points, n_dimensions]
        Coordinates of points.

    Attributes
    ----------
    X : array, [n_points, n_dimensions]
        Coordinates of points.
    labels : array, [n_points]
        Labels of each point.
    cluster_centers : array, [n_clusters, n_dimensions]
        Coordinates of cluster centers.
    n_cluster : int
        Number of clusters.
    cluster_instance : class
        Class used for clustering.
    fit_time : float
        Time of clustering.
    """
    X = None
    labels = None
    cluster_centers = None
    n_cluster = None
    cluster_instance = None
    fit_time = 0

    def __init__(self, X):
        self.X = X

    def fit(self):
        """ Perform clustering.
```

```
    """
    self.cluster_instance.fit(self.X)

def plotClusters(self, plotCenters = True):
    """ Plot the results of clustering.

    Parameters
    ----------
    plotCenters : boolean, default True
        If true, mark centers of clusters on plot.

    Notes
    -----
    Uses matplotlib.pyplot for plotting data.
    """
    # set colors of clusters
    colors = 10 * ['r.', 'g.', 'm.', 'c.', 'k.', 'y.', 'b.']
    # create a new figure
    plt.figure()
    # for each point in X array
    for i in range(len(self.X)):
        # plot it on figure with specified color
        plt.plot(self.X[i][0], self.X[i][1],
                 colors[self.labels[i]], markersize = 5)
    # if plotCenters set to True, plot cluster centers as "X" marks
    if plotCenters:
        plt.scatter(self.cluster_centers[:, 0],
                    self.cluster_centers[:, 1], marker = "x",
                    s = 150, linewidths = 2.5, zorder = 10)
    # showing result
    plt.show()

def exportCentersToTextFile(self, filename):
    """ Record cluster centers to text file.

    Parameters
    ----------
    filename : string path
        Recording file name.

    Notes
    -----
    Uses JSON data format.
    """
    try:
        with open(filename, 'w') as file_:
            json.dump(self.cluster_centers.tolist(), file_)
    except IOError as e:
```

```
            print('{}'.format(e))

    def exportPointsToTextFile(self, filename):
        """ Record points with labels to text file.

        Parameters
        ----------
        filename : string path
            Recording file name.

        Notes
        -----
        Uses JSON data format.
        """
        # create new array with one more dimension for points
        X_ = np.empty((len(self.X), 3))
        # for each point in X array
        for i in range(len(X_)):
            # set X_[i][0] and X_[i][1] to point coordinates
            X_[i][0], X_[i][1] = self.X[i][0], self.X[i][1]
            # set X_[i][2] to point label
            X_[i][2] = self.labels[i]

        try:
            with open(filename, 'w') as file_:
                json.dump(X_.tolist(), file_)
        except IOError as e:
            print('{}'.format(e))
```

3 Модуль MeanShift (эвклидова метрика)

```
import time
import numpy as np
from sklearn.cluster import MeanShift
from sklearn.cluster import estimate_bandwidth
from DataCollector import DataCollector
from ClusteringMachine import ClusteringMachine

class MeanShiftClusteringMachine(ClusteringMachine):
    """ A derived class from ClusteringMachine.

    Performs clustering with using Mean Shift algorithm.

    Parameters
    ----------
    X : array, [n_points, n_dimensions]
        Coordinates of points.
    useEstimateBandwidth : boolean, default True
        If true, use sklearn.cluster.estimate_bandwidth for bandwidth
```

```
        calculation.
    binSeeding : boolean, default True
        If true, speed up algorithm by initializing fewer seeds.

    """
    def __init__(self, X, useEstimateBandwidth = True, binSeeding = True):
        self.X = X
        self.fit_time = time.time()

        if useEstimateBandwidth:
            bandwidth_ = estimate_bandwidth(self.X, quantile = 0.3)
        else:
            bandwidth_ = None
        # set cluster instanse to sklearn.cluster.MeanShift with
        # given parameters
        self.cluster_instance = MeanShift(bandwidth = bandwidth_,
            bin_seeding = binSeeding)

    def fit(self):
        """ Perform clustering.

        """
        # perform clustering
        self.cluster_instance.fit(self.X)
        # get points labels
        self.labels = self.cluster_instance.labels_
        # get cluster centers
        self.cluster_centers = self.cluster_instance.cluster_centers_
        # get clusters number
        self.n_cluster = len(np.unique(self.labels))
        # calculate time
        self.fit_time = time.time() - self.fit_time

if __name__ == '__main__':
    # if running as script

    # create DataCollector object
    dc = DataCollector()
    # upload data from 'data.txt' file
    dc.uploadFromTextFile('data.txt', delimiter = ', ')
    # plot collected data
    # dc.plotData()
    # get data from dc object
    X = dc.getData()[::30]

    # create MeanShiftClusteringMachine object with default parameters
    ms = MeanShiftClusteringMachine(X)
    # perform clustering
```

```
    ms.fit()
    # print info
    time.sleep(.5)
    print('Fit time: {}, clusters: {}'.format(ms.fit_time, ms.n_cluster))
    # plot results
    # ms.plotClusters()
    # export centers to 'centers.js'
    ms.exportCentersToTextFile('c_centers.js')
    # export points to 'points.js'
    ms.exportPointsToTextFile('p_points.js')
```

4 Модуль Route

```
import time
import subprocess
import json
from urllib import urlopen
import numpy as np
import warnings

class route():
    """ A class for get distance between points by finding a route.

    Uses OSRM-Project API for routing. It sends requests to local OSRM machine
    and gets distance from json-formatted responce.

    Possible errors:
        - Cannot find route between points:
            OSRM machine can't find route between given points. In this
            case it will try to locate given points to closest OSM node,
            and try to find route between located points.

    Attributes
    ----------
    osrm: subprocess
        Local OSRM machine.

    Notes
    -----
    See https://github.com/Project-OSRM/osrm-backend/wiki for OSRM-Project API.

    """
    osrm = None
    cache = {}

    def start(self):
        """ Start local OSRM machine.

        Needs time to start, contains a 5 seconds sleep statement.
```

```
    You can have one running machine per instance of class.

    """
    if self.osrm is None:
        osrm = subprocess.Popen('osrm-routed ~/map/map.osrm', shell=True)
        time.sleep(5)
        self.osrm = osrm
    try:
        with open('route_cache.txt', 'r') as file_:
            self.cache = json.load(file_)
    except:
        pass

def route_distance(self, a, b):
    """ Get distance between points.

    Get distance between points a and b by finding route on OSM map.

    Parameters
    ----------
    a: array-like, shape=[2]
        Coordinates of first point.
    b: array-like, shape=[2]
        Coordinates of second point.

    Returns
    -------
    dist: int
        Route distance between given points.

    """
    # if shape of given arrays isn't [2]
    if a.shape[0] == 1:
        a = np.array([a[0][0], a[0][1]])
    if b.shape[0] == 1:
        b = np.array([b[0][0], b[0][1]])

    # trying to find distance between points in cache
    c = map(lambda i: np.round(i, decimals=5), a)
    d = map(lambda i: np.round(i, decimals=5), b)
    key1 = '{}{}{}{}'.format(*np.append(c, d))
    key2 = '{}{}{}{}'.format(*np.append(d, c))
    try:
        dist = self.cache[key1]
    except KeyError:
        pass
    try:
```

```
        dist = self.cache[key2]
    # if not found
    except KeyError:
        if np.array_equal(c, d):
            # if points are the same return zero
            dist = 0.0
        else:
            # send request to find route between points
            url = "http://localhost:5000/viaroute?loc={},{}&loc={},{}" \
                "&geometry=false&alt=false".format(*np.append(a, b))
            # get responce
            responce = urlopen(url)
            # parse json
            data = json.load(responce)

            # if route isn't found
            if data['status'] is not 0:
                # show warning
                warnings.warn('Cannot find route between {},{} ' \
                    'and {},{}. Trying to locate points.' \
                    ''.format(*np.append(a, b)))
                # locate first point
                loc = 'http://localhost:5000/locate?loc={},{}'.format(*a)
                l_resp = urlopen(loc)
                l_data = json.load(l_resp)
                # if can't locate
                if l_data['status'] is not 0:
                    # stop osrm machine
                    self.stop()
                    # throw error
                    raise ValueError(l_data['status_message'])
                else:
                    first = l_data['mapped_coordinate']

                # locate second point
                loc = 'http://localhost:5000/locate?loc={},{}'.format(*b)
                l_resp = urlopen(loc)
                l_data = json.load(l_resp)
                if l_data['status'] is not 0:
                    self.stop()
                    raise ValueError(l_data['status_message'])
                else:
                    second = l_data['mapped_coordinate']

                e = map(lambda i: np.round(i, decimals=5), first)
                f = map(lambda i: np.round(i, decimals=5), second)
                key3 = '{}{}{}{}'.format(*np.append(e, f))
                key4 = '{}{}{}{}'.format(*np.append(f, e))
```

```
                    try:
                        dist = self.cache[key3]
                        return dist
                    except KeyError:
                        pass
                    try:
                        dist = self.cache[key4]
                        return dist
                    except KeyError:
                        pass

                    # try to find route between located points
                    url = "http://localhost:5000/viaroute?loc={},{}&loc={},{}" \
                        "&geometry=false&alt=false".format(*np.append(first,
                        second))
                    responce = urlopen(url)
                    data = json.load(responce)
                    if data['status'] is not 0:
                        self.stop()
                        raise ValueError(data['status_message'])
                # if route is found return distance
                dist = data['route_summary']['total_distance']
            self.cache[key1] = dist
            self.cache[key2] = dist
        return dist

    def stop(self):
        """ Stop local OSRM machine.

        """
        if self.osrm is not None:
            self.osrm.terminate()
            self.osrm = None
        with open('route_cache.txt', 'w') as file_:
            json.dump(self.cache, file_)
```

5 Модуль K-Means с метрикой route

```
from __future__ import division
import json
import time
import random
import numpy as np
from DataCollector import DataCollector
from ClusteringMachine import ClusteringMachine
from route import route

class InitMachine():
    """ Initializes centers of clusters.
```

```
    Attributes
    ----------
    centers : array, [n_clusters, n_dimensions + 1]
    """
    centers = None

    def __init__(self):
        self.centers = np.empty([0, 3])

    def grid(self, gridSize, bounds):
        """ Initialize centers by grid.

        Parameters
        ----------
        gridSize : array {size_x, size_y}
            Set grid size.
        bounds : array {bottom_border, left_border, top_border, right_border}
            Specify borders.
        """
        delta_lt = (bounds[2] - bounds[0]) / gridSize[0]
        delta_ln = (bounds[3] - bounds[1]) / gridSize[1]
        curr_lt = bounds[0] + delta_lt / 2
        curr_ln = bounds[1] + delta_ln / 2

        self.centers = np.append(self.centers,
            [[curr_lt, curr_ln, 0]], axis = 0)
        for i in range(gridSize[0] * gridSize[1]):
            if curr_ln + delta_ln > bounds[3]:
                if curr_lt + delta_lt > bounds[2]:
                    break
                else:
                    curr_lt += delta_lt
                    curr_ln -= delta_ln * (gridSize[0] - 1)
            else:
                curr_ln += delta_ln
            self.centers = np.append(self.centers,
                [[curr_lt, curr_ln, i + 1]], axis = 0)

    def random(self, count, bounds):
        """ Initialize centers by random.

        Parameters
        ----------
        count : int
            Set clusters count.
        bounds : array {bottom_border, left_border, top_border, right_border}
            Specify borders.
```

```
        """
        for i in range(count):
            self.centers = np.append(self.centers,
                [[random.uniform(bounds[0], bounds[2]),
                  random.uniform(bounds[1], bounds[3]), i]],
                axis = 0)

    def file(self, filename):
        """ Initialize centers by random.

        Parameters
        ----------
        filename : int
            Specify source file name.
        """
        i = 0
        for line in open(filename):
          if not '#' in line:
            lat, lon = line.split(' ')[:2]
            self.centers = np.append(self.centers,
                [[float(lat), float(lon), i]], axis = 0)
            i += 1

    def getCenters(self):
        """ Get centers of clusters.

        """
        return self.centers

class KMeans():
    """ K-Means clustering.

    Attributes
    ----------
    max_iteration_ : int
        Maximum iteration number. After reaching it, clustering is
        considered as completed.
    cluster_centers_ : array, [n_clusters, n_dimensions + 1]
        Centers of clusters.
    labels_ : array, [n_points]
        Labels of points.
    route_ : class route

    Parameters
    ----------
    max_iteration : int
        Set maximum iteration number.
    """
```

```
max_iteration_ = None
cluster_centers_ = None
labels_ = None
route_ = route()

def __init__(self, max_iteration):
    self.max_iteration_ = max_iteration

def dist(self, a, b, metric):
    """ Calculate distance between two points.

    Parameters
    ----------
    a : array, [n_dimensions]
        First point.
    b : array, [n_dimensions]
        Second point.
    metric : string
        Used metric.
        Known metrics are 'euclid', 'route'.

    Returns
    -------
    r : float
        Distance between points.
    """
    if metric == 'route':
        r = self.route_.route_distance(a, b)
    elif metric == 'euclid':
        r = np.sqrt((b[0] - a[0]) ** 2 + (b[1] - a[1]) ** 2)
    else:
        raise ValueError('Unknown metric: {}'.format(metric))
    return r

def stop(self, iter, old, new):
    """ Check whenever clustering needs to be stopped.

    Parameters
    ----------
    iter : int
        Number of current iteration.
    old : array, [n_clusters, n_dimensions + 1]
        Centers of clusters on previous iteration.
    new : array, [n_clusters, n_dimensions + 1]
        Centers of clusters on current iteration.

    Returns
    -------
```

```
        stop : boolean
            If true, clustering is completed.
        """
        if iter >= self.max_iteration_:
            return True
        return np.array_equal(old, new)

    def fit(self, X, centroids, metric):
        """ Perform clustering.

        Parameters
        ----------
        X : array, [n_points, n_dimensions]
            Coordinates of points.
        centroids : array, [n_clusters, n_dimensions + 1]
            Centers of clusters.
        """
        # set initial parameters
        iteration = 0
        c_old = None

        c_curr = centroids
        l_curr = np.empty([len(X)])

        if metric == 'route':
            self.route_.start()
        # while clustering isn't completed
        while not self.stop(iteration, c_old, c_curr):
            # show iteration number
            print 'iteration {}'.format(iteration + 1)
            # for each point
            for i in range(len(X)):
                # calculate distance between point and all the clusters centers
                distances = [(self.dist(X[i], c_curr[each], metric), each) for
                    each in range(len(c_curr))]
                # sort distances ascending
                distances.sort(key=lambda x: x[0])
                # pick number of cluster, which center has the smallest
                # distance to point
                m = distances[0][1]
                # set label of point
                l_curr[i] = c_curr[m][2]
            # equate the previous and current centers of clusters
            c_old = c_curr

            # create empty python array
            # each item will contain all the points belongs to specific cluster
            arrays = [np.empty([0, 2]) for each in c_curr]
```

```
            # array for calculated centers of clusters
            mu = np.empty([len(c_curr), 3])
            # for each point
            for i in range(len(X)):
                # append it to array contained points of specific cluster
                arrays[int(l_curr[i])] = np.append(arrays[int(l_curr[i])],
                    [X[i]], axis = 0)

            # for each cluster
            for i in range(len(arrays)):
                # if it contains points
                if arrays[i] != np.empty([0, 2]):
                    # calculate center of cluster
                    mu[i] = np.append(np.mean(arrays[i], axis = 0), i)
                else:
                    # if it doesn't: pick the old coordinates of center
                    mu[i] = c_curr[i]
            # equate current centroids to calculated
            c_curr = mu

            # increment iteration counter
            iteration += 1
        # record results
        self.cluster_centers_ = c_curr
        self.labels_ = l_curr
        if metric == 'route':
            self.route_.stop()

class KMeansClusteringMachine(ClusteringMachine):
    """ A derived class from ClusteringMachine.

    Performs clustering with using K-Means algorithm.

    Parameters
    ----------
    X : array, [n_points, n_dimensions]
        Coordinates of points.
    init : string, default 'random'
        Specify the initializing type.
    count : int, default 40
        Set clusters count (in random initializing).
    gridSize : array {size_x, size_y}, default [7, 7]
        Set size of grid (in grid initializing).
    filename : string path, default 'init.txt'
        Specify the name of initialize file (in file initializing).
    max_iteration : int, default 50
        Set maximum iteration number.
    header : string, default None
```

```
    Set header that contains info about borders.

Attributes
----------
initM : InitMachine object
    Initializes centers of clusters.
bounds : array {bottom_border, left_border, top_border, right_border}
    Bounds of initial centers generation.
"""
initM = None
bounds = None

def __init__(self, X, init = 'random', count = 40, gridSize = [7, 7],
             filename = 'init.txt', max_iteration = 50, header = None):
    self.X = X

    self.initM = InitMachine()
    self.bounds = self.getBounds(header_ = header, points_ = X)
    if init == 'random':
        self.initM.random(count = count, bounds = self.bounds)
    elif init == 'grid':
        self.initM.grid(gridSize = gridSize, bounds = self.bounds)
    elif init == 'file':
        self.initM.file(filename = filename)
    else:
        print('Unrecognized init type: {}'.format(init))
    self.cluster_centers = self.initM.getCenters()
    self.cluster_instance = KMeans(max_iteration = max_iteration)

def getBounds(self, header_ = None, points_ = None):
    """ Calculate bounds of initial centers generation.

    Parameters
    ----------
    header_ : string
        String contains info about borders.
    points_ : array [n_points, n_dimensions]
        Coordinates of points

    Returns
    -------
    bounds : array {bottom_border, left_border, top_border, right_border}
        Bounds of initial centers generation.
    """
    bounds_ = None
    # if both header_ and points_ are not setted show error
    if header_ == None and points_ == None:
        print('No source to get bounds')
```

```
        else:
            # if header_ is setted, split it to get bounds
            if header_:
                bounds_ = [float(n) for n in header_.split()]
            else:
                # if points are setted, choose four coordinates
                # of different points: most bottom, most left,
                # most top and most right, as bounds
                t, r, b, l = points_[0], points_[1], points_[0], points_[1]
                for p in points_[1:]:
                    if p[0] > t[0]:
                        t = p
                    if p[0] < b[0]:
                        b = p
                    if p[1] > r[1]:
                        r = p
                    if p[1] < l[1]:
                        l = p
                bounds_ = [b[0], l[1], t[0], r[1]]
        return bounds_

    def fit(self, metric="route"):
        """ Perform clustering.

        """
        t_start = time.time()
        # perform clustering
        self.cluster_instance.fit(self.X, self.cluster_centers, metric)
        # calculate time
        self.fit_time = time.time() - t_start
        # get points labels
        self.labels = self.cluster_instance.labels_
        # get cluster centers
        self.cluster_centers = self.cluster_instance.cluster_centers_
        # get clusters number
        self.n_cluster = len(np.unique(self.labels))

if __name__ == '__main__':
    # if running as script

    # create DataCollector object
    dc = DataCollector()
    # upload data from 'data.txt' file
    dc.uploadFromTextFile('data.txt', delimiter = ', ')
    # plot collected data
    # dc.plotData()
    # get data from dc object
    X = dc.getData()
```

```
# create KMeansClusteringMachine object with specified parameters
km = KMeansClusteringMachine(X, init = 'random', max_iteration = 100, count
= 40)
# perform clustering
km.fit('route')
# print info
print('Fit time: {}, clusters: {}'.format(km.fit_time, km.n_cluster))
# export centers to 'centers.js'
km.exportCentersToTextFile('k_centers.js')
# export points to 'points.js'
km.exportPointsToTextFile('k_points.js')
```