МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Колледж Алтайского государственного университета

Отделение экономики и информационных технологий

КУРСОВАЯ РАБОТА

по дисциплине «Основы алгоритмизации и программирование»

РАЗРАБОТКА ПРОГРАММЫ ДЛЯ СЖАТИЯ ИЗОБРАЖЕНИЯ С ПОМОЩЬЮ КЛАСТЕРИЗАЦИИ

Выполнил студент
2 курса 262в-сп группы
Беляев Д.С.
_____
(подпись)
Научный руководитель
Евдокимов Е.А.
_____
(подпись)

Курсовая
работа защищена
«____»_____2018г.
Оценка _____
_____
(подпись)

Барнаул 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Алгоритм кластеризации позволяет находить похожие объекты в изображении, которые впоследствии объединяются в группу. Внедрение алгоритма позволит проанализировать, а также сжать изображение. В области медицины кластеризация помогает идентифицировать центры клеток на изображении группы клеток. Используя GPS-данные мобильного устройства, можно определить наиболее посещаемые пользователем места в пределах определенной территории. Алгоритм будет способствовать профилактике правонарушений.

Объект курсовой работы: кластеризация методом k-средних

Предмет: сжатие изображений с помощью кластеризации

Цель: разработать программный продукт для сжатия изображений с помощью кластеризации

Задачи:

- исследовать кластеризацию и метод k-средних
- провести анализ существующих программ, использующих кластеризацию
- сделать выводы и разработать концепцию своего ПО
- реализовать ПО в среде программирования Python

Структура курсовой работы. Работа состоит из введения, двух глав и заключения. В первой главе будет проведено исследование алгоритмов кластеризации и сжатия изображений с помощью кластеризации методом k-средних, будут проанализированы существующие программы, использующие кластеризацию. Во второй главе будет выбрана методология разработки, описан выбранный язык программирования. Будет подробное описание программы: код программы, назначение, интерфейс, функции, входные и выходные данные, а также тестирование.

1 КЛАСТЕРИЗАЦИЯ

1.1. Основные понятия

Кластеризация (или кластерный анализ) — это задача разбиения множества объектов на группы, называемые кластерами. Внутри каждой группы должны оказаться «похожие» объекты, а объекты разных группы должны быть как можно более отличны. Главное отличие кластеризации от классификации состоит в том, что перечень групп четко не задан и определяется в процессе работы алгоритма.

Применение кластерного анализа в общем виде сводится к следующим этапам:

- Отбор выборки объектов для кластеризации.
- Определение множества переменных, по которым будут оцениваться объекты в выборке. При необходимости – нормализация значений переменных.
- Вычисление значений меры сходства между объектами.
- Применение метода кластерного анализа для создания групп сходных объектов (кластеров).
- Представление результатов анализа.

После получения и анализа результатов возможна корректировка выбранной метрики и метода кластеризации до получения оптимального результата.

Выбор метрики, или меры близости(меры расстояния), является нетривиальным и одним из основных моментом исследования, от которого в значительной степени зависит окончательный вариант разбиения объектов на классы при данном алгоритме разбиения. Метрику можно использовать свою в зависимости от количества переменных, например: рост, возраст, пол, день рождения. В каждом конкретном случае этот выбор должен производиться по-

своему, в зависимости от целей исследования, физической и статистической природы наблюдений, априорных сведений о характере вероятностного распределения.

Основные метрики, используемые в алгоритмах кластеризации:

1) Евклидово расстояние

Наиболее распространенная функция расстояния. Представляет собой геометрическим расстоянием в многомерном пространстве:

$$\rho(x, x') = \sqrt{\sum_{i}^{n} (x_i - x'_i)^2} \quad (1)$$

2) Квадрат евклидова расстояния

Применяется для придания большего веса более отдаленным друг от друга объектам. Это расстояние вычисляется следующим образом:

$$\rho(x, x') = \sum_{i}^{n} (x_i - x'_i)^2 \quad (2)$$

3) Расстояние городских кварталов (манхэттенское расстояние) Это расстояние является средним разностей по координатам. В большинстве случаев эта мера расстояния приводит к таким же результатам, как и для обычного расстояния Евклида. Однако для этой меры влияние отдельных больших разностей (выбросов) уменьшается (т.к. они не возводятся в квадрат). Формула для расчета манхэттенского расстояния(3):

$$\rho(x, x') = \sum_{i}^{n} |x_i - x'_i| \quad (3)$$

4) Расстояние Чебышева

Это расстояние используется, когда нужно определить два объекта как различные, если они различаются по какой-либо одной координате. Расстояние Чебышева вычисляется по формуле:

$$\rho(x, x') = \max(|x_i - x'_i|) \quad (4)$$

5) Степенное расстояние

Применяется в случае, когда необходимо увеличить или уменьшить вес, относящийся к размерности, для которой соответствующие объекты сильно отличаются. Степенное расстояние вычисляется по следующей формуле:

$$\rho(x, x') = \sqrt[r]{\sum_{i}^{n} (x_i - x'_i)^p} \ , \quad (5)$$

где r и p – параметры, определяемые пользователем, x и x’ точки на расстоянии. Параметр p ответственен за постепенное взвешивание разностей по отдельным координатам, параметр r ответственен за прогрессивное взвешивание больших расстояний между объектами. Если оба параметра – r и p — равны двум, то это расстояние совпадает с расстоянием Евклида.

1.2. Алгоритмы кластеризации

Среди алгоритмов кластеризации можно выделить такие виды:

1) Иерархические и плоские.

Иерархические алгоритмы строят не одно разбиение выборки на непересекающиеся кластеры, а систему вложенных разбиений. На выходе мы получаем дерево кластеров, корнем которого является вся выборка, а листьями — наиболее мелкие кластера.

Плоские алгоритмы строят одно разбиение объектов на кластеры.

2) Четкие и нечеткие.

Четкие (или непересекающиеся) алгоритмы каждому объекту выборки ставят в соответствие номер кластера, то есть каждый объект принадлежит только одному кластеру.

Нечеткие (или пересекающиеся) алгоритмы каждому объекту ставят в соответствие набор вещественных значений, показывающих степень отношения объекта к кластерам. Каждый объект относится к каждому кластеру с некоторой вероятностью.

В случае использования иерархических алгоритмов поднимается вопрос о том, как объединять между собой кластера, как вычислять «расстояния» между ними. Существует несколько метрик:

- Одиночная связь (расстояния ближайшего соседа)

В этом методе расстояние между двумя кластерами определяется расстоянием между двумя наиболее близкими объектами (ближайшими соседями) в различных кластерах. Результирующие кластеры имеют тенденцию объединяться в цепочки.

- Полная связь (расстояние наиболее удаленных соседей)

В этом методе расстояния между кластерами определяются наибольшим расстоянием между любыми двумя объектами в различных кластерах (т.е. наиболее удаленными соседями). Этот метод обычно работает очень хорошо, когда объекты происходят из отдельных групп. Если же кластеры имеют удлиненную форму или их естественный тип является «цепочечным», то этот метод непригоден.

- Невзвешенное попарное среднее

В этом методе расстояние между двумя различными кластерами вычисляется как среднее расстояние между всеми парами объектов в них. Метод эффективен, когда объекты формируют различные группы, однако он работает одинаково хорошо и в случаях протяженных («цепочечного» типа) кластеров.

- Взвешенное попарное среднее

Метод идентичен методу невзвешенного попарного среднего, за исключением того, что при вычислениях размер соответствующих кластеров (т.е. число объектов, содержащихся в них) используется в качестве весового коэффициента. Поэтому данный метод должен быть использован, когда предполагаются неравные размеры кластеров.

- Невзвешенный центроидный метод

В этом методе расстояние между двумя кластерами определяется как расстояние между их центрами тяжести.

- Взвешенный центроидный метод (медиана)

Этот метод идентичен предыдущему, за исключением того, что при вычислениях используются веса для учета разницы между размерами

кластеров. Поэтому, если имеются или подозреваются значительные отличия в размерах кластеров, этот метод оказывается предпочтительнее предыдущего.

Среди алгоритмов иерархической кластеризации выделяются два основных типа: восходящие и нисходящие алгоритмы. Нисходящие алгоритмы работают по принципу «сверху-вниз»: в начале все объекты помещаются в один кластер, который затем разбивается на все более мелкие кластеры. Более распространены восходящие алгоритмы, которые в начале работы помещают каждый объект в отдельный кластер, а затем объединяют кластеры во все более крупные, пока все объекты выборки не будут содержаться в одном кластере. Таким образом строится система вложенных разбиений. Результаты таких алгоритмов обычно представляют в виде дерева – дендрограммы. Классический пример такого дерева – классификация животных и растений.

Задачу кластеризации можно рассматривать как построение оптимального разбиения объектов на группы. При этом оптимальность может быть определена как требование минимизации среднеквадратической ошибки разбиения:

$$e^2(X, L) = \sum_{j=1}^{K} \sum_{i=1}^{n_j} \left\| x_i^{(j)} - c_j \right\|^2, \qquad (6)$$

где c_j — «центр масс» кластера j (точка со средними значениями характеристик для данного кластера).

Алгоритмы квадратичной ошибки относятся к типу плоских алгоритмов. Самым распространенным алгоритмом этой категории является метод k-средних. Этот алгоритм строит заданное число кластеров, расположенных как можно дальше друг от друга. Работа алгоритма делится на несколько этапов:

1) Случайно выбрать k точек, являющихся начальными «центрами масс» кластеров.

2) Отнести каждый объект к кластеру с ближайшим «центром масс».

3) Пересчитать «центры масс» кластеров согласно их текущему составу.

4) Если критерий остановки алгоритма не удовлетворен, вернуться к п. 2.

В качестве критерия остановки работы алгоритма обычно выбирают минимальное изменение среднеквадратической ошибки. Так же возможно останавливать работу алгоритма, если на шаге 2 не было объектов, переместившихся из кластера в кластер.

Наиболее популярным алгоритмом нечеткой кластеризации является алгоритм c-средних (c-means). Он представляет собой модификацию метода k-средних.

Шаги работы алгоритма:

1. Выбрать начальное нечеткое разбиение *n* объектов на *k* кластеров путем выбора матрицы принадлежности *U* размера *n x k*.

2. Используя матрицу U, найти значение критерия нечеткой ошибки:

$$E^2(X,U) = \sum_{i=1}^{N}\sum_{k=1}^{K} U_{ik} \left\| x_i^{(k)} - c_k \right\|^2, \quad (7)$$

где c_k — «центр масс» нечеткого кластера k:

$$c_k = \sum_{i=1}^{N} U_k x_i, \quad (8)$$

3. Перегруппировать объекты с целью уменьшения этого значения критерия нечеткой ошибки.

4. Возвращаться в п. 2 до тех пор, пока изменения матрицы U не станут незначительными.

Этот алгоритм может не подойти, если заранее неизвестно число кластеров, либо необходимо однозначно отнести каждый объект к одному кластеру.

Алгоритмы, основанные на теории графов

Суть таких алгоритмов заключается в том, что выборка объектов представляется в виде графа $G=(V, E)$, вершинам которого соответствуют объекты, а ребра имеют вес, равный «расстоянию» между объектами. Достоинством графовых алгоритмов кластеризации являются наглядность, относительная простота реализации и возможность внесения различных усовершенствований, основанные на геометрических соображениях. Основными алгоритмам являются алгоритм выделения связных компонент, алгоритм построения минимального покрывающего (остовного) дерева и алгоритм послойной кластеризации.

В алгоритме выделения связных компонент задается входной параметр R и в графе удаляются все ребра, для которых «расстояния» больше R. Соединенными остаются только наиболее близкие пары объектов. Смысл алгоритма заключается в том, чтобы подобрать такое значение R, лежащее в диапазон всех «расстояний», при котором граф «развалится» на несколько связных компонент. Полученные компоненты и есть кластеры.

Для подбора параметра R обычно строится гистограмма распределений попарных расстояний. В задачах с хорошо выраженной кластерной структурой данных на гистограмме будет два пика – один соответствует внутрикластерным расстояниям, второй – межкластерным расстояниям. Параметр R подбирается из зоны минимума между этими пиками. При этом управлять количеством кластеров при помощи порога расстояния довольно затруднительно.

Алгоритм послойной кластеризации основан на выделении связных компонент графа на некотором уровне расстояний между объектами (вершинами). Уровень расстояния задается порогом расстояния *c*. Например, если расстояние между объектами $0 \leq \rho(x, x') \leq 1$, то $0 \leq c \leq 1$.

Алгоритм послойной кластеризации формирует последовательность подграфов графа *G*, которые отражают иерархические связи между кластерами:

$$G^0 \subseteq G^1 \subseteq \cdots \subseteq G^m, \quad (9)$$

где $G^t = (V, E^t)$ — граф на уровне c^t,

$$E^t = \{e_{ij} \in E : p_{ij} \leq c_t\} \quad (10)$$

c^t – t-ый порог расстояния, m – количество уровней иерархии, $G^0 = (V, o)$, o – пустое множество ребер графа, получаемое при $t^0 = 1$, $G^m = G$, то есть граф объектов без ограничений на расстояние (длину ребер графа), поскольку $t^m = 1$.

Посредством изменения порогов расстояния $\{c^0, \ldots, c^m\}$, где $0 = c^0 < c^1 < \ldots < c^m = 1$, возможно контролировать глубину иерархии получаемых кластеров. Таким образом, алгоритм послойной кластеризации способен создавать как плоское разбиение данных, так и иерархическое.

1.3. Метод k-средних

Алгоритм k-means применяется к объектам, которые представляются точками в d-мерном векторном пространстве. Таким образом, это кластеры набора d-мерных векторов, $D = \{x_i | i = 1, \ldots, N\}$, где $x_i \in R^d$ обозначает i-ый объект или «точку данных». k-means является алгоритмом кластеризации, который разделяет D на k кластеров точек. То есть, алгоритм k-means объединяет все точки данных в D так, что каждая точка x_i попадает в один и только один из k разделов. Можно отследить, какая точка находится в каком кластере, назначив каждой точке номер кластера. Точки с таким же номером кластера находятся в одном и том же кластере, в то время как точки с

различными номерами кластера находятся в разных кластерах. Это можно обозначить как кластерный составной вектор m длиною N, где m_i является номером кластера x_i.

Значение k является основным из входных данных алгоритма. Как правило, значение k основано на критериях, таких как предварительное знание о том, сколько на самом деле кластеров появится в D, как много кластеров требуется для текущего приложения, или типы кластеров, найденные путем изучения/экспериментирования с различными значениями k.

В k-means, каждый из k кластеров представлен одной точкой в R^d. Обозначим этот набор представителей кластера как множество $C = \{c_j | j=1, ….., k\}$. Эти представители k кластеров также называются состояние кластера или центроиды кластера.

В алгоритмах кластеризации точки группируются некоторым понятием «близости» или «подобия». В k-means мера близости по умолчанию Евклидово расстояние. В частности можно с готовностью показать, что k-means пытается минимизировать следующую неотрицательную функцию стоимости:

$$C = \sum_{i=1}^{N} \left(\arg min_j \left\| x_i - c_j \right\|_2^2 \right) \quad (11)$$

Другими словами, k-means пытается минимизировать итоговый квадрат Евклидова расстояния между каждой точкой x_i и ее самым близким представителем кластера c_j. Уравнение 11 часто упоминается как целевая функция k-means.

Алгоритм k-means разделяет D итеративным способом, чередующимся между 2 шагами: переприсваивание номера кластера всем точкам в D и обновление представителей кластера, основанных на точках данных в каждом кластере. Алгоритм работает следующим образом. Сначала представители кластера инициализируются, выбирая k из R^d. Методы для выбора этих начальных источников включают случайную выборку из набора данных,

устанавливая их как решение кластеризации маленького подмножества данных, или нарушая глобальное среднее значение данных k раз. В алгоритме мы инициализируем случайно выбранные k точек. Затем выполняется алгоритм итерации до сходимости за 2 шага.

Каждой точке данных присваивается ее самый близкий представитель притом, что связи нарушаются произвольно. Это приводит к разделению данных.

Каждый представитель кластера перемещается к центру (т. е. среднее арифметическое) всех точек данных, присвоенных ему. Объяснение этого шага основано на наблюдении, что данное множество точек, единственный лучший представитель для этого множества (в смысле минимизации суммы квадрата Евклидова расстояния между каждой точкой и представителем) не что иное, как срединная точка данных. Именно поэтому представитель кластера часто взаимозаменяемо называют срединным элементом кластера или центроидом кластера, отсюда и название этого алгоритма.

Алгоритм сходится, когда присвоение и значения C_j больше не изменяются. Можно показать, что целевая функция k-means, определенная в уравнении 1, будет уменьшаться всякий раз, когда есть изменения в присвоении или шагах измерения, и сближение (сходимость в одной точке) гарантировано за конечное число итераций.

Выбор оптимального значения k может быть трудным. Самое простое решение попробовать несколько различных значений k и выбрать кластеризацию, которая минимизирует целевую функцию k-means (11).

Алгоритм K-means:

Первоначальное распределение объектов по кластерам

1) Выбор случайным образом k точек данных из D как начальное множество представителей кластера C
2) Распределение объектов по кластерам в соответствие с формулой (11)

Перераспределение срединных элементов

1) Вычисление центра для каждого кластера
2) Перераспределение объектов по кластерам

Каждая итерация нуждается в N*k сравнений, которые определяют временную сложность одной итерации. Число итераций, требуемых для сходимости, изменяется и может зависеть от N. Чем больше точек во множестве (N), тем дольше будет работать алгоритм.

Сократить время работы алгоритма можно путем распараллеливания этапа распределения точек по кластерам.

Потенциальная проблема алгоритма – проблема «пустых» кластеров.

1.4. Анализ имеющегося ПО для сжатия изображений

Программных продуктов для сжатия изображений методом k-средних в ходе исследования не найдено, однако для этого есть открытое программное обеспечение, реализованное на разных языках программирования. Например: библиотека scipy написана на Python, OpenCV на C++, ELKI на Java. Все содержат реализацию k-means. Также мною были найдена пользовательская реализация с помощью html и javascript на ресурсе GitHub.

2 ПРОГРАММА ДЛЯ СЖАТИЯ ИЗОБРАЖЕНИЙ

2.1. Выбор методологии разработки программного обеспечения

1. «Waterfall Model» (каскадная модель или «водопад»)

Одна из самых старых, подразумевает последовательное прохождение стадий, каждая из которых должна завершиться полностью до начала следующей. В модели Waterfall легко управлять проектом. Благодаря её жесткости, разработка проходит быстро, стоимость и срок заранее определены. Каскадная модель будет давать отличный результат только в проектах с четко и заранее определенными требованиями и способами их реализации. Нет возможности сделать шаг назад, тестирование начинается только после того, как разработка завершена или почти завершена. Продукты, разработанные по данной модели без обоснованного ее выбора, могут иметь недочеты (список требований нельзя скорректировать в любой момент), о которых становится известно лишь в конце из-за строгой последовательности действий. Стоимость внесения изменений высока, так как для ее инициализации приходится ждать завершения всего проекта. Тем не менее, фиксированная стоимость часто перевешивает минусы подхода. Исправление осознанных в процессе создания недостатков возможно, и, по нашему опыту, требует от одного до трех дополнительных соглашений к контракту с небольшим ТЗ. Каскадную методологию используют:

– В случаях, когда требования известны, понятны и зафиксированы. Противоречивых требований не имеется.

– В случаях, когда нет проблем с доступностью программистов нужной квалификации.

– В относительно небольших проектах.

2. «V-Model»

Унаследовала структуру «шаг за шагом» от каскадной модели. V-образная модель применима к системам, которым особенно важно

бесперебойное функционирование. Например, прикладные программы в клиниках для наблюдения за пациентами, интегрированное ПО для механизмов управления аварийными подушками безопасности в транспортных средствах и так далее. Особенностью модели можно считать то, что она направлена на тщательную проверку и тестирование продукта, находящегося уже на первоначальных стадиях проектирования. Стадия тестирования проводится одновременно с соответствующей стадией разработки, например, во время кодирования пишутся модульные тесты. V-модель используют:

– Если требуется тщательное тестирование продукта.

– Для малых и средних проектов, где требования четко определены и фиксированы.

– В условиях доступности инженеров необходимой квалификации, особенно тестировщиков.

3. «Incremental Model» (инкрементная модель)

В инкрементной модели полные требования к системе делятся на различные сборки. Терминология часто используется для описания поэтапной сборки ПО. Имеют место несколько циклов разработки, и вместе они составляют жизненный цикл «мульти-водопад». Цикл разделен на более мелкие легко создаваемые модули. Каждый модуль проходит через фазы определения требований, проектирования, кодирования, внедрения и тестирования. Процедура разработки по инкрементной модели предполагает выпуск на первом большом этапе продукта в базовой функциональности, а затем уже последовательное добавление 14 новых функций, так называемых «инкрементов». Процесс продолжается до тех пор, пока не будет создана полная система. Инкрементные модели используются там, где отдельные запросы на изменение ясны, могут быть легко формализованы и реализованы. Инкрементную модель используют:

– Когда основные требования к системе четко определены и понятны. В то же время некоторые детали могут дорабатываться с течением времени.

– Когда требуется ранний вывод продукта на рынок.

– Когда есть несколько рискованных целей.

4. «RAD Model» (rapid application development model или быстрая разработка приложений)

RAD-модель — разновидность инкрементной модели. В RAD-модели компоненты или функции разрабатываются несколькими высококвалифицированными командами параллельно, будто несколько мини- проектов. Временные рамки одного цикла жестко ограничены. Созданные модули затем интегрируются в один рабочий прототип. Синергия позволяет очень быстро предоставить клиенту для обозрения что-то рабочее с целью получения обратной связи и внесения изменений. RAD-модель используют только при наличии высококвалифицированных и узкоспециализированных архитекторов. Бюджет проекта большой, чтобы оплатить этих специалистов вместе со стоимостью готовых инструментов автоматизированной сборки. RAD-модель может быть выбрана при уверенном знании целевого бизнеса и необходимости срочного производства системы в течение 2-3 месяцев.

5. «Agile Model» (гибкая методология разработки)

В «гибкой» методологии разработки после каждой итерации заказчик может наблюдать результат и понимать, удовлетворяет он его или нет. Это одно из 15 преимуществ гибкой модели. К ее недостаткам относят то, что из-за отсутствия конкретных формулировок результатов сложно оценить трудозатраты и стоимость, требуемые на разработку. Экстремальное программирование (XP) является одним из наиболее известных применений гибкой модели на практике. Agile используют:

– Когда потребности пользователей постоянно меняются в динамическом бизнесе.

– Потому что изменения на Agile реализуются за меньшую цену из-за частых инкрементов.

– Потому что в отличие от модели водопада, в гибкой модели для старта проекта достаточно лишь небольшого планирования.

6. «Iterative Model» (итеративная или итерационная модель)

Итерационная модель жизненного цикла не требует для начала полной спецификации требований. Вместо этого, создание начинается с реализации части функционала, становящейся базой для определения дальнейших требований. Этот процесс повторяется. Версия может быть неидеальна, главное, чтобы она работала. Итеративную модель используют:

– Когда требования к конечной системе заранее четко определены и понятны.

– Когда проект большой или очень большой.

– Когда основная задача должна быть определена, но детали реализации могут эволюционировать с течением времени.

7. «Spiral Model» (спиральная модель)

«Спиральная модель» похожа на инкрементную, но с акцентом на анализ рисков. Она хорошо работает для решения критически важных бизнес-задач, когда неудача несовместима с деятельностью компании, в условиях выпуска новых продуктовых линеек, при необходимости научных исследований и практической апробации.

Для разработки программы я решил выбрать каскадную модель:

– Модель лучше всего подходит для разработки небольшой программы.

– Требования к программе известны и понятны.

– Нет проблем с доступностью программистов нужной квалификации.

2.2. Описание выбранного языка программирования

Python — высокоуровневый язык программирования с динамической типизацией, поддерживающий объектно-ориентированный, функциональный и императивный стили программирования. Это язык общего назначения, на

котором можно одинаково успешно разрабатывать системные приложения с графическим интерфейсом, утилиты командной строки, научные приложения, игры, приложения для веб и многое другое.

Я выбрал его из-за понятности, простоты. По причине подключения модулей к стандартным библиотекам, Python дает все возможности для осуществления задач.

2.3. Программа для сжатия изображений с помощью кластеризации

Кроме реализации алгоритма кластеризации в программе заявлено не было ничего, поэтому для первой версии программы затраты были маленькие. Программа состоит из нескольких частей: непосредственно код, описывающий алгоритм кластеризации методом k-средних, код графического интерфейса, а также повторяющаяся часть кода, вынесенная в отдельный модуль. Импортированные библиотеки: библиотека интерфейса Tkinter, диалоговые окна Tkinter, библиотека для работы с матрицами numpy, модули Image, ImageTk для работы с изображениями и связи с графическим интерфейсом, matplotlib.pyplot для сборки массивов в изображения, модуль KMeans с реализованным алгоритмом k-means, модуль shuffle, который перемешивает массивы последовательно.

1)Код алгоритма

Код представлен связанными между собой подпрограммами.

а)Подпрограмма compute(event) первая часть

Аргумент event у подпрограммы указывает на привязку данной функции к кнопке. Программа будет выполняться, если поле не было пустым, либо не было со значением 0. Переменной img задается другая переменная, глобальная, из другой подпрограммы. img – ссылка на изображение. Затем изображение преобразуется в массив с помощью функции np.array, где элементы массива превращаются в 8-битовый формат, то есть в формат RGB, затем делятся на 255

для того, чтобы сжатие произошло с меньшими потерями. w,h,d размеры массива, assert – проверка на формат RGB, то есть количество значений в подмассиве, если вдруг оказалось, что там 4 значения, программа откажет в выполнении. reshape переводит массив в размерность 2d. Затем новый массив перемешивается последовательно. random_state замена np.random.seed(), генерирует случайные числа. И собственно сам метод kmeans, принимает количество кластеров и массив, который делится на эти кластеры. labels – случайные метки изображения.

```
def compute(event):
    if clusters()!='' or clusters()!=0:
        img = oh
        img = np.array(img, dtype=np.float64) / 255
        w, h, d = original_shape = tuple(img.shape)
        assert d == 3
        image_array = np.reshape(img, (w * h, d))
        image_array_sample = shuffle(image_array, random_state=0)[:1000]
        kmeans = KMeans(n_clusters=clusters(), random_state=0).fit(image_array_sample)
        labels = kmeans.predict(image_array)
```

б)Подпрограмма compute(event) вторая часть

Сначала создается поле для изображения plt.figure, отключается отображение осей, затем выполняется наша подпрограмма recreate_image, которая принимает центроиды, метки кластеров, и размеры массива, plt.imshow образует читаемый массив изображения. И замена сообщения в интерфейсе при выполнении подпрограммы.

```
        plt.figure(2)
        plt.clf()
        ax = plt.axes([0, 0, 1, 1])
        plt.axis('off')
        q=recreate_image(kmeans.cluster_centers_, labels, w, h)
        plt.imshow(q)
        a['text']='Сжатие произведено,\n сохраните, посмотрите!'
```

в)Подпрограмма recreate_image(codebook, labels, w, h)

Здесь создается сжатое изображение, подаются на вход массивы центроид, меток и размеры массива. Массив центроид преобразуется в одномерный массив, затем создается массив из нулей для заполнения, с принятыми значениями массивов. Выполняется цикл, в котором каждый 0 заменяется на новый элемент кластера.

```
def recreate_image(codebook, labels, w, h):
    d = codebook.shape[1]
    image = np.zeros((w, h, d))
    label_idx = 0
    for i in range(w):
        for j in range(h):
            image[i][j] = codebook[labels[label_idx]]
            label_idx += 1
    return image
```

г)Подпрограмма clusters()

Считываем с помощью метода get из текстового поля число кластеров, возвращаем в типе int.

```
def clusters():
    a=e.get('1.0', END+'-1c')
    return int(a)
```

2)Код графического интерфейса

а)Основной блок

Объявляется переменная для появления графической формы root=Tk(). Называем с помощью title нашу программу, а именно заголовок в самом верху. Меняем иконку iconbitmap, размер окна geometry, чтобы расположенные кнопки выглядели эстетично. Создаем простое контекстное меню из 1 вкладки: «Файл»; добавляем туда опции: «Открыть» с привязанной функцией ownage() и «Выйти» с функцией root.destroy.

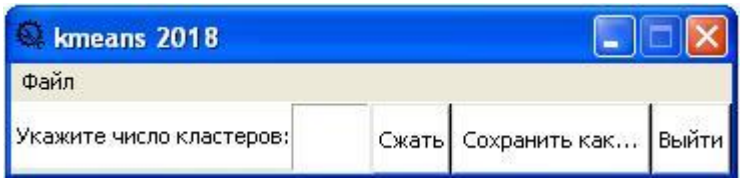

Рисунок 1. Основной блок

Также делаем невозможным изменение размера окна. Добавляем 3 кнопки на окно: «Сжать», к ней с помощью bind привязали основную функцию compute(event), «Выйти»(root.destroy), «Сохранить как…»(команда rampage()). Также на окне расположен элемент Label(метка) с надписью: «Укажите число кластеров» рядом с текстовым полем. Текстовое поле сделано вместо специального поля ввода из-за отсутствия параметра высоты. Дальше идут координаты расположения всех элементов относительно верхнего левого угла программы. root.mainloop() окончательно упаковывает и создает окно программы.

```
root=Tk()

root.title('kmeans 2018')
root.iconbitmap(u'C:\Documents and Settings\Dima B\Мои документы\Downloads\\auto-repair.ico')
root.geometry('360x35')
m=Menu(root)
root.config(menu=m)
u=Menu(m)
m.add_cascade(label='Файл',menu=u)
u.add_command(label='Открыть...', command=ownage)
u.add_command(label='Выйти', command=root.destroy)
root.resizable(width=False, height=False)
a = Label(root, text='Укажите число кластеров:', bg='white', width=23, height=2)
b = Button(root, text='Сжать', bg='white', width=5, height=2)
c = Button(root, text='Выйти', bg='white', command=root.destroy, width=5, height=2)
e = Text(root, width=5, height=2)
ch = Button(root, text='Сохранить как...', width=15, height=2, command=rampage,bg='white')
b.bind("<ButtonRelease-1>",compute)
a.place(x=0,y=0)
b.place(x=180,y=0)
c.place(x=320,y=0)
ch.place(x=220,y=0)
e.place(x=140,y=0)
```

root.mainloop()

б)Подпрограмма ownage()

Выводится диалоговое окно об открытии файла, форматы изображений стоят по умолчанию: jpg, png, gif, bmp.

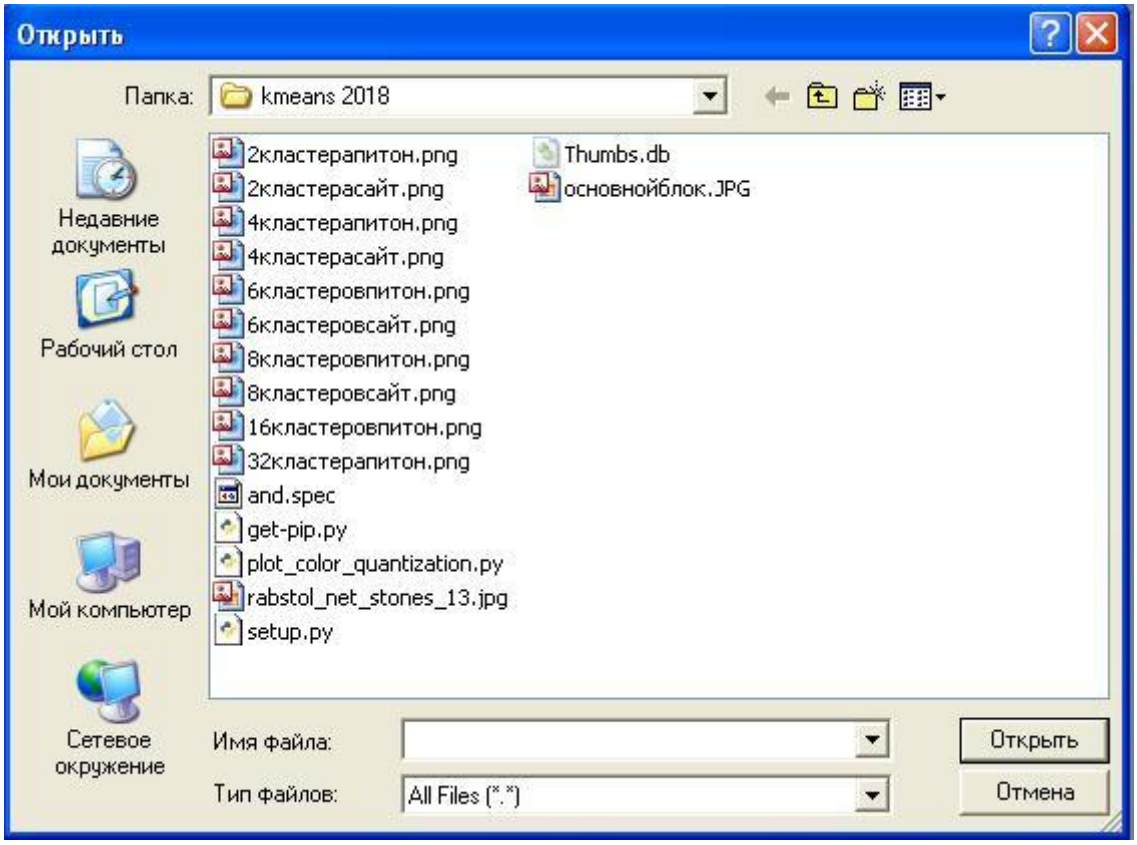

Рисунок 2. Диалоговое окно

При загрузке любого другого файла ничего не произойдет. После открытия изображение помещается на окно. Объявляются глобальные переменные oh, lel, иначе изображение не отобразится на экране. oh передается в функцию compute(event).

```
def ownage():
    zt=askopenfilename()
    b=Toplevel(root)
    b.resizable(width=False, height=False)
    oh=Image.open(zt)
    lel=ImageTk.PhotoImage(oh)
    topimg=Label(b,image=lel)
    topimg.pack(side="bottom", fill="both", expand="yes")
    global lel, oh
```

в)Подпрограмма rampage()

Открывается диалоговое окно для сохранения файлов, по умолчанию сохранит в .png формате.

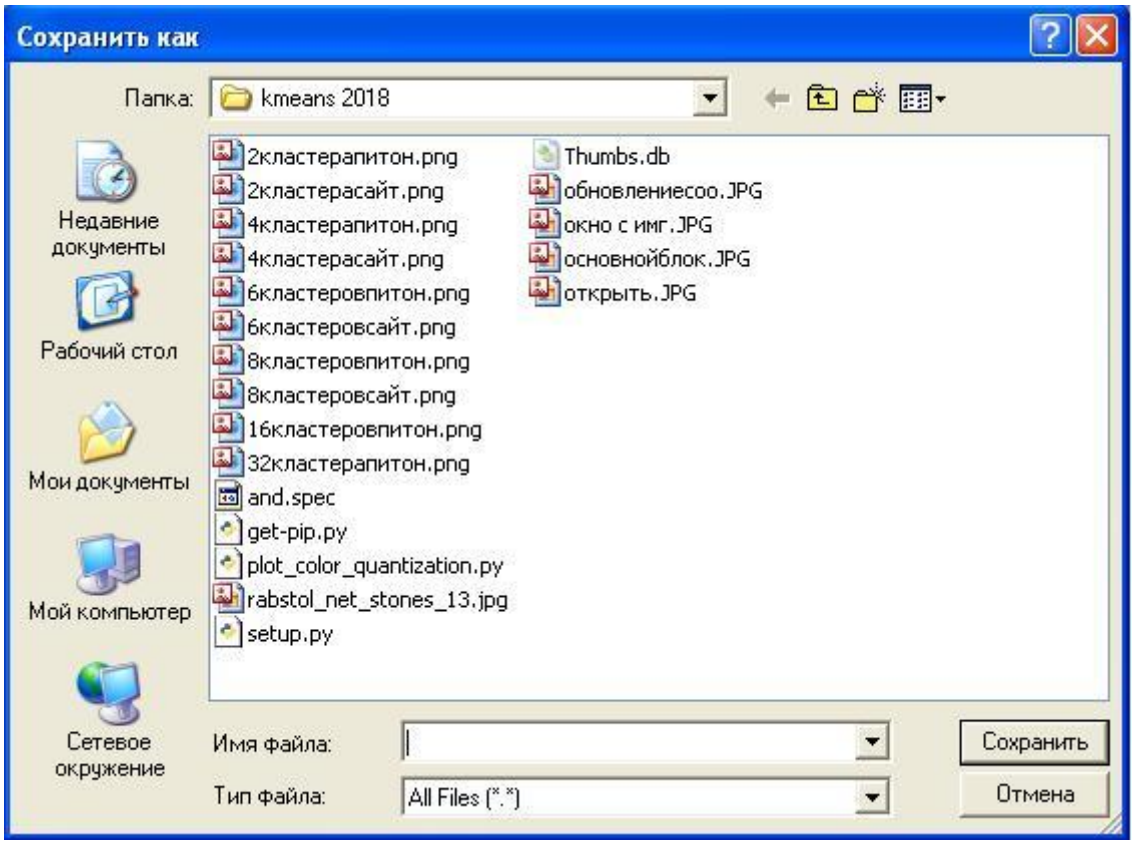

Рисунок 3. Диалоговое окно

Это диалоговое окно является выбором ссылки, в переменной сохранится значение ссылкой, сжатое изображение сохранится по этой ссылке.

```
def rampage():
    t=asksaveasfilename()
    plt.savefig(t)
```

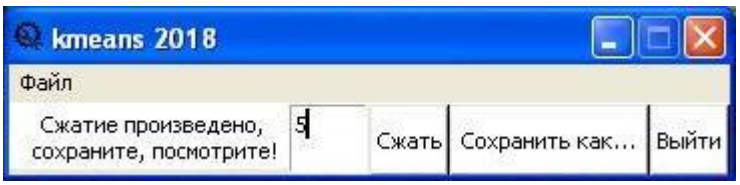

Рисунок 4. Обновление сообщения при выполнении программы

2.4. Тестирование

К тестированию необходимо подготовиться, а именно выбрать подходящее изображение по размеру, так как большие изображения программа обрабатывает долго. К примеру, достаточно изображения с разрешением 800x600. Для сравнения возьмем сайт-программу. Протестируем при k=2,4,6,8.

Рисунок 5. Оригинальное изображение

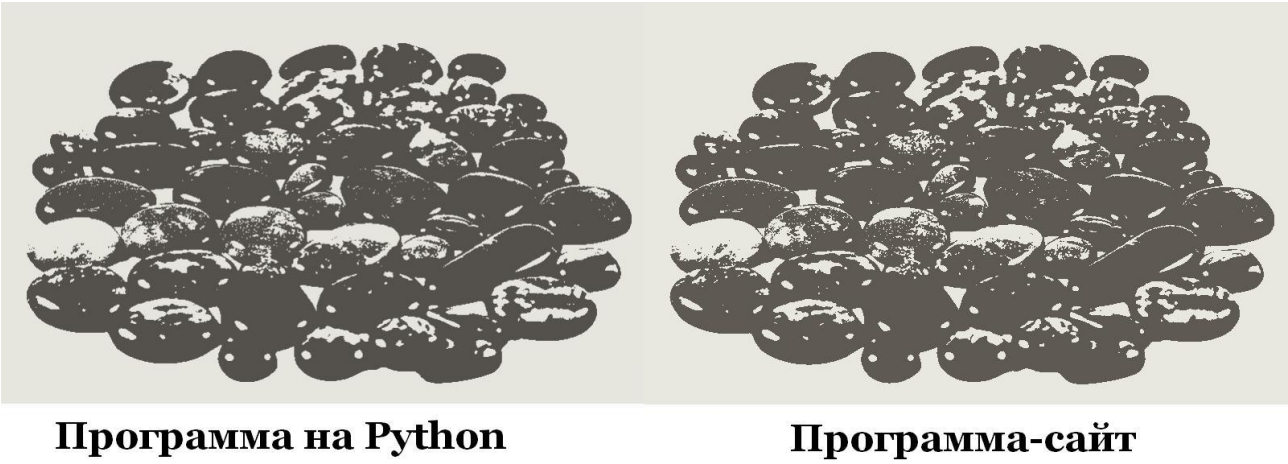

Рисунок 6. Сжатое изображение при k=2

Рисунок 7. Сжатое изображение при k=4

Рисунок 8. Сжатое изображение при k=6

Рисунок 9. Сжатое изображение при k=8

Посмотрев на рисунки, можно увидеть, что при k=2,4 изображения из 2 разных программ более похожие друг на друга, а уже с k=6 начались расхождения по цвету: у программы-сайта код посчитал нужным заполонить

кластер серым или бежевым цветом, у нашей программы этот цвет заполнился коричневым. При k=8 у программы-сайта появился коричневый цвет и близкий к синему, у нашей же программы коричневого стало меньше, но появился отчетливый синий цвет. По результатам тестирования программы выполняют одно и то же, но разнятся они все-таки меньше. Следовательно, программа выполняется верно.

ЗАКЛЮЧЕНИЕ

По ходу курсовой работы я:

1) Исследовал алгоритмы кластеризации и метод k-средних
2) Провел анализ существующих программ, использующих кластеризацию
3) Сделал выводы и разработать концепцию своего ПО
4) Реализовал ПО в среде программирования Python

В перспективе: вывод сжатого изображения на экран, исправление ошибок и багов, улучшение интерфейса, расширение функционала в целом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Князь Д. Анализ основных алгоритмов кластеризации многомерных данных// LAP Lambert Academic Publishing, 2014.
2.Habr[Электронный ресурс]: Обзор алгоритмов кластеризации данных, 2010. URL: https://habr.com/post/101338/ - дата обращения (24.05.18)
3.datascience[Электронный ресурс]: Метод k-средних, 2016. URL: http://datascientist.one/k-means-algorithm/ - дата обращения (25.06.18)
4.scikit-learn[Электронный ресурс]: Color Quantization using K-Means, 2018. URL: http://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html#sphx-glr-auto-examples-cluster-plot-color-quantization-py – дата обращения (23.06.18)

ПРИЛОЖЕНИЕ

Код программы для сжатия изображений с помощью кластеризации:

```
from tkinter import *
from tkinter.filedialog import *
import numpy as np
from PIL import Image, ImageTk
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.utils import shuffle

def rampage():
    t=asksaveasfilename()
    plt.savefig(t)

def ownage():
        zt=askopenfilename()
        b=Toplevel(root)
        b.resizable(width=False, height=False)
        oh=Image.open(zt)
        lel=ImageTk.PhotoImage(oh)
        topimg=Label(b,image=lel)
        topimg.pack(side="bottom", fill="both", expand="yes")
        global lel, oh

def compute(event):
    if clusters()!='' or clusters()!=0:
        img = oh
        img = np.array(img, dtype=np.float64) / 255
        w, h, d = original_shape = tuple(img.shape)
        assert d == 3
        image_array = np.reshape(img, (w * h, d))
        image_array_sample = shuffle(image_array, random_state=0)[:1000]
        kmeans = KMeans(n_clusters=clusters(), random_state=0).fit(image_array_sample)
        labels = kmeans.predict(image_array)
        plt.figure(2)
        plt.clf()
        ax = plt.axes([0, 0, 1, 1])
        plt.axis('off')
        q=recreate_image(kmeans.cluster_centers_, labels, w, h)
        plt.imshow(q)
        a['text']='Сжатие произведено,\n сохраните, посмотрите!'

def recreate_image(codebook, labels, w, h):
    d = codebook.shape[1]
    image = np.zeros((w, h, d))
    label_idx = 0
    for i in range(w):
        for j in range(h):
            image[i][j] = codebook[labels[label_idx]]
            label_idx += 1
```

```
    return image

def clusters():
    a=e.get('1.0', END+'-1c')
    return int(a)

root=Tk()

root.title('kmeans 2018')
root.iconbitmap(u'C:\Documents and Settings\Dima B\Мои документы\Downloads\\auto-
repair.ico')
root.geometry('360x35')
m=Menu(root)
root.config(menu=m)
u=Menu(m)
m.add_cascade(label='Файл',menu=u)
u.add_command(label='Открыть...', command=ownage)
u.add_command(label='Выйти', command=root.destroy)
root.resizable(width=False, height=False)
a = Label(root, text='Укажите число кластеров:', bg='white', width=23, height=2)
b = Button(root, text='Сжать', bg='white', width=5, height=2)
c = Button(root, text='Выйти', bg='white', command=root.destroy, width=5, height=2)
e = Text(root, width=5, height=2)
ch = Button(root, text='Сохранить как...', width=15, height=2, command=rampage,bg='white')
b.bind("<ButtonRelease-1>",compute)
a.place(x=0,y=0)
b.place(x=180,y=0)
c.place(x=320,y=0)
ch.place(x=220,y=0)
e.place(x=140,y=0)

root.mainloop()
```