# Теоретическое приложение к документации

В этом файле предоставлены краткие теоретические описания алгоритмов кластеризации и мер качества кластеризации. Все они были реализованы на языке программирования Scala. Пользователь может обращаться к ним как напрямую, так и через Python методы, которые обращаются к .jar файлу и используют те же реализации.

# Алгоритмы кластеризации

## Модуль *heaven*

Для каждого алгоритма *Algorithm*: \* Внутри *heaven* лежит *Algorithm.scala* и *AlgorithmModel.scala.* \*

Ниже каждый алгоритм кластеризации рассмотрен подробнее, описан принцип работы каждого алгоритма.

### KMeans

**Расположение:** https://gitlab.com/rainifmo/sparkling/-/tree/master/heaven/src/main/scala/ru/ifmo/rain/algorithms/kmeans

Выбор начальных центроидов производится с помощью процедуры *k-means++*. За *initSteps* шагов выбирается порядка 2\* *initSteps* \* *k* кандидатов начальных центроидов. На этих центроидах запускается *k-means++*. На следующих *maxIterations* шагах производится стандартная для алгоритма *k*-средних процедура, реализованнная с помощью парадигмы map-reduce. В каждом блоке данных вычисляетмся количество объектов и их “сумма” до ближайших из *k* кандидатов. После этого каждый *k*-мерный массив “суммируется”. В итоге полученный массив на позиции *i* содержит “сумму” векторов данных и количество объектов, относящихся к каждому центроиду. Поделив одно на другое получаем координаты новых центроидов. После исчерпания количества ходов или выполнения условия останова получаем массив из координат полученных центроидов. Метки кластера для каждого объекта определяются номером ближайшего центроида.

**Параметры:**

* *k* —количество кластеров
* *maxIterations* — максимальное количество шагов, по достижении которых выполняется условие останова
* *initSteps* — количество шагов инициализации
* *convergence* — минимальное расстояние, которое считается значимым. Если центроиды переместились меньше, чем значение параметра, то условие останова выполнено
* *seed* — параметр генератора псевдослучайных чисел, необходимый для выбора начального решения

### Meanshift

**Расположение:** https://gitlab.com/rainifmo/sparkling/-/tree/master/heaven/src/main/scala/ru/ifmo/rain/algorithms/meanshift

Начальные средние выбираются из числа объектов случайным образом. На каждом шаге для каждого среднего находятся “соседи” из окрестности радиусом *radius*, после чего среднее сдвигается в сторону центроида объектов внутри этой окрестности. По окончании итерационного этапа нужно удалить столько средних, чтобы осталось не более указанной верхней границы. Для начала удаляются “дубликаты”, расстояние между которыми пренебрежимо мало. Из оставшихся оставляем те, у которых больше соседей-объектов.

**Параметры:**

* *radius* —радиус окрестности объекта
* *maxClusters* — верхняя граница количества кластеров
* *maxIterations* — максимальное количество шагов, по достижении которых выполняется условие останова
* *initial* — количество первоначальных кандидатов. Должно вписаться в память драйвера
* *convergence* — минимальное расстояние, которое считается значимым. Если центроиды переместились меньше, чем значение параметра, то условие останова выполнено
* *seed* — параметр генератора псевдослучайных чисел, необходимый для выбора начального решения

### DBSCAN

**Расположение:** https://gitlab.com/rainifmo/sparkling/-/tree/master/heaven/src/main/scala/ru/ifmo/rain/algorithms/dbscan

Привычный алгоритм *DBSCAN* не может быть реализован в парадигме map-reduce. Распределенная реализация первым шагом разбивает пространство объектов на блоки таким образом, чтобы в каждом блоке количество объектов было примерно одинаковым. На следующем шаге каждый из блоков обрабатывается параллельно: для каждого из объектов находим его “соседей” внутри блока, затем находим элементы, находящиеся за границами блока на расстоянии не превышающем *epsilon* и ищем для них “соседей” внутри блока. Следующим шагом для каждого объекта запускается стандартный *DBSCAN* внутри блока. Для того, чтобы из набора локальных разбиений получить “общую картину” нужно найти пары объектов из разных блоков, но на расстоянии не большем *epsilon*. Если количество объектов в обоих кластерах больше, чем *minPoints*, то кластеры сливаются. На заключительном шаге для каждого объекта определяется метка класса или идентификатор шума.

**Параметры:**

* *epsilon* — радиус окрестности
* *minPoints* — количество объектов в окрестности точки, чтобы она считалась “базовой” точкой.
* *borderNoise* — true, если точки на границах следует рассматривать как шум и не относить к кластерам
* *maxClusters* — максимальное количество кластеров
* *pointsInBox* — минимальное количество точек в одном разбиении пространства. Не влияет на результат кластеризации, а влияет только на производительность
* *axisSplits* — количество разбиений на одно измерение. Не влияет на результат кластеризации, а влияет только на производительность
* *levels* — количество процедур, разделяющих пространство по некоторому измерению. Не влияет на результат кластеризации, а влияет только на производительность

### Спектральные алгоритмы

**Расположение:** https://gitlab.com/rainifmo/sparkling/-/tree/master/heaven/src/main/scala/ru/ifmo/rain/algorithms/spectral

Особенностью спектральных алгоритмов является высокая вычислительная сложность: они требуют подсчета собственных значений матриц. Более того, существует ограничение по памяти: собственные векторы имеют размерность *n* (количество объектов) и должны помещаться в памяти локально. В рамках библиотеки реализованы две версии алгоритма: спектральный алгоритм с матрицей схожести и спектральный алгоритм с матрицей смежности.

**Параметры:**

* *eigen* — количество собственных значений
* *k* — количество кластеров
* *maxIterations* — максимальное количество шагов, по достижении которых выполняется условие останова
* *seed* — параметр генератора псевдослучайных чисел, необходимый для выбора начального решения

#### Спектральный алгоритм с матрицей схожести.

Для каждой моды строится матрица схожести по формуле

Для каждой полученной матрицы вычисляется *eigen* собственных значений и собственных векторов. Для каждой из мод полученные собственные векторы объединяются в одну матрицу размером *n* \* *eigen*, образуя спектральный эмбеддинг. Из *m* эмбеддингов образуется новый набор данных, который кластеризуется описанным выше алгоритмом *KMeans*.

#### Спектральный алгоритм с матрицей смежности

Алгоритм отличается от предыдущей модификации способом задания матрицы. Здесь требуется вычислить лапласиан графа , где – диагональная матрица со значениями *neighbours*, – матрица смежности каждой из m мод. Каждый ряд *i* матрицы строится следующим образом: подсчитываются расстояния от объекта до всех остальных: , выбираются *neighbours* индексов *j* с наименьшим значением , для них , для остальных .

### BIRCH

**Расположение:** https://gitlab.com/rainifmo/sparkling/-/tree/master/heaven/src/main/scala/ru/ifmo/rain/algorithms/spectral

Строится сбалансированное CF-дерево с элементами . После чего на вход описанному выше *KMeans* подаются центроиды листьев.

**Параметры:**

* *k* — количество кластеров
* *maxIterations* — максимальное количество шагов, по достижении которых выполняется условие останова
* *threshold* — после слияния радиус образованного подкластера должен быть меньше этого значения
* *maxBranches* — максимальное количество подкластеров на одном узле

### Bisecting K-Means

**Расположение:** https://gitlab.com/rainifmo/sparkling/-/tree/master/heaven/src/main/scala/ru/ifmo/rain/algorithms/bisecting

Алгоритм на каждом шаге разбивает объекты на подкластеры с помощью *KMeans*, начиная со всего множества объектов. Процесс продолжается, пока количество кластеров не достигнет заданного.

**Параметры:**

* *k* —количество кластеров
* *maxIterations* — максимальное количество шагов, по достижении которых выполняется условие останова
* *minClusterSize* — минимальное количество объектов (в процентах от общей выборки) в кластере
* *convergence* — минимальное расстояние, которое считается значимым. Если центроиды переместились меньше, чем значение параметра, то условие останова выполнено
* *seed* — параметр генератора псевдослучайных чисел, необходимый для выбора начального решения

## Модуль sparkling

Внутри *sparkling* лежит *Algorithm.py* (где *Algorithm* – название алгоритма кластеризации), внутри которого прописаны два класса, отвечающие за работу алгоритма и пространства конфигураций гиперпараметров. **Гиперпараметры каждого из алгоритмов не отличаются от тех, что были описаны выше.**

Каждый реализованный Python метод вызывает один из scala-методов, которые описаны выше.

# Внутренние меры качества кластеризации

Пример вызова вычисления меры:

from sparkling.opt.measures import Internal  
  
Internal.CHOSEN\_MEASURE.evaluate(sparkling\_df)

Где sparkling\_df - датафрейм после препроцессинга и кластеризации (присутствует колонка *label*).

### Индекс Calinski–Harabasz:

Идентификатор: *calinski\_harabasz*

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/Internals.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

Индекс должен возрастать. Компактность понимается как расстояние от точек до их центроида, а разделимость как расстояние от центроид кластеров до глобального центроида.

### Индекс Дэвиcа-Болдуина

Идентификатор: *davies\_bouldin*

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/Internals.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

Компактность понимается как расстояние от объектов кластеров до их центроидов, а отделимость как расстояния между центроидами.

При улучшении качества кластеризации убывает.

### Индекс Дэвиcа-Болдуина (Альтернативный)

Идентификатор: *davies\_bouldin\**

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/Internals.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

При улучшении качества кластеризации убывает.

### Индекс Данна

Идентификатор: *dunn*

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/Internals.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

Индекс должен возрастать.

### Обобщенный Индекс Данна (вкл. gD31, gD33, gD41, gD43, gD51, gD53)

Идентификаторы: *gd31*, *gd33*, *gd41*, *gd43*, *gd51*, *gd53*

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/Internals.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

Использует вариации значний и :

Индекс должен возрастать.

### Апроксимация Индекса Данна (вкл. gD31, gD33, gD41, gD43, gD51, gD53)

Идентификаторы: *dunn\**, *gd31\**, *gd33\**, *gd41\**, *gd43\**, *gd51\**, *gd53\**

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/Internals.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

Использует апроксимацию , где вычисляется как расстояние от дальнейшей от центроида точки до самой удалённой от неё точки. Это позволяет значительно снизить вычислительную сложность.

Индекс должен возрастать.

### Критерий Силуэта

Идентификатор: *silhouette*

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/Internals.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

Среднее внутрикластерное расстояние вычисляется по формуле:

Среднее межкластерное расстояние вычисляется по формуле:

Индекс должен возрастать.

### Апроксимация Критерия Силуэта

Идентификатор: *silhouette\**

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/Internals.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

Для уменьшения вычислительной сложности *a* и *b* вычисляются через центроиды, а не через все попарные расстояния.

### Индекс SF

Идентификатор: *sf*

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/Internals.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

Индекс должен возрастать.

# Внешние меры качества кластеризации

Пример вызова вычисления меры:

from sparkling.opt.measures import External  
  
External.CHOSEN\_MEASURE.evaluate(sparkling\_df)

Где sparkling\_df - размеченный датафрейм после препроцессинга и кластеризации (присутствуют колонки *label* и *class*).

### Индекс Rand

Идентификатор: *rand*

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/ExtPairwise.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

Ограничен значениями 0 и 1, где 1 обозначает полное совпадение результатов кластеризации с разметкой.

### Индекс Adjusted Rand

Идентификатор: *adjustedRand*

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/ExtConjugacy.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

Ограничен значениями -1 и 1, где 1 обозначает полное совпадение результатов кластеризации с разметкой.

### Индекс Жаккара

Идентификатор: *jaccard*

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/ExtPairwise.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

Ограничен значениями 0 и 1, где 1 обозначает полное совпадение результатов кластеризации с разметкой.

### Индекс Фоулкса – Мэллова

Идентификатор: *fowlkesMallows*

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/ExtPairwise.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

Индекс должен возрастать.

### Индекс Phi

Идентификатор: *phi*

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/ExtPairwise.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

Индекс должен возрастать.

### Мера F1

Идентификатор: *f1*

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/ExtConjugacy.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

Индекс должен возрастать.

### Purity

Идентификатор: *purity*

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/ExtConjugacy.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

Ограничен значениями 0 и 1, где 1 обозначает полное совпадение результатов кластеризации с разметкой.

### Entropy

Идентификатор: *entropy*

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/ExtConjugacy.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

### Minkowski Score

Идентификатор: *minkowski*

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/ExtConjugacy.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

### Индекс Гудмэна-Крускала

Идентификатор: *goodmanKruskal*

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/ExtConjugacy.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py

### Variation of Information

Идентификатор: *varInformation*

Расположение:

<https://gitlab.com/rainifmo/sparkling/-/blob/master/heaven/src/main/scala/ru/ifmo/rain/measures/ExtConjugacy.scala>

https://gitlab.com/rainifmo/sparkling/-/blob/master/sparkling/opt/measures.py