НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Факультет компьютерных наук
Образовательная программа бакалавриата
«Прикладная математика и информатика»
Яндекс
Программный проект
Методы машинного обучения (регрессии, классификации) как агрегатные функции

Выполнили студент группы БПМИ-166:
Кожихов А.О.
Конькова М.Н.
Кузнецов М.А.

Научный руководитель:
Руководитель проекта,
Яндекс, руководитель группы
Миловидов А.Н.

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Цель работы

Реализация методов машинного обучения (регрессии, классификации, кластеризации) как агрегатных функций в СУБД ClickHouse.

Задачи работы

1. Реализация линейной регрессии (Кожихов А.О.)
2. Разработка логистической регрессии (Конькова М.Н.)
3. Имплементация инкрементальной кластеризации (Кузнецов М.А.)
4. Написание функции для предсказания на новых данных evalMLMethod (Кожихов А.О.)
5. Проведение тестов для каждого из указанных методов.

НАЗНАЧЕНИЕ ПРОГРАММЫ

Функциональное назначение

Функциональным назначением программы является решение задач регрессии и кластеризации на стриминговых данных на архитектуре агрегатных функций системы ClickHouse.

Эксплуатационное назначение

Эксплуатационным назначением программы является предоставление пользователям ClickHouse возможности решения задач машинного обучения (регрессии, классификации и кластеризации) на их данных в стриминговом режиме.

АКТУАЛЬНОСТЬ РАБОТЫ

ClickHouse предназначена для выполнения аналитических запросов на больших данных. Введение машинного обучения и возможности предсказания исхода по клиентским данным является острой необходимостью сегодня, поэтому реализация данных опций является крайне актуальной задачей.

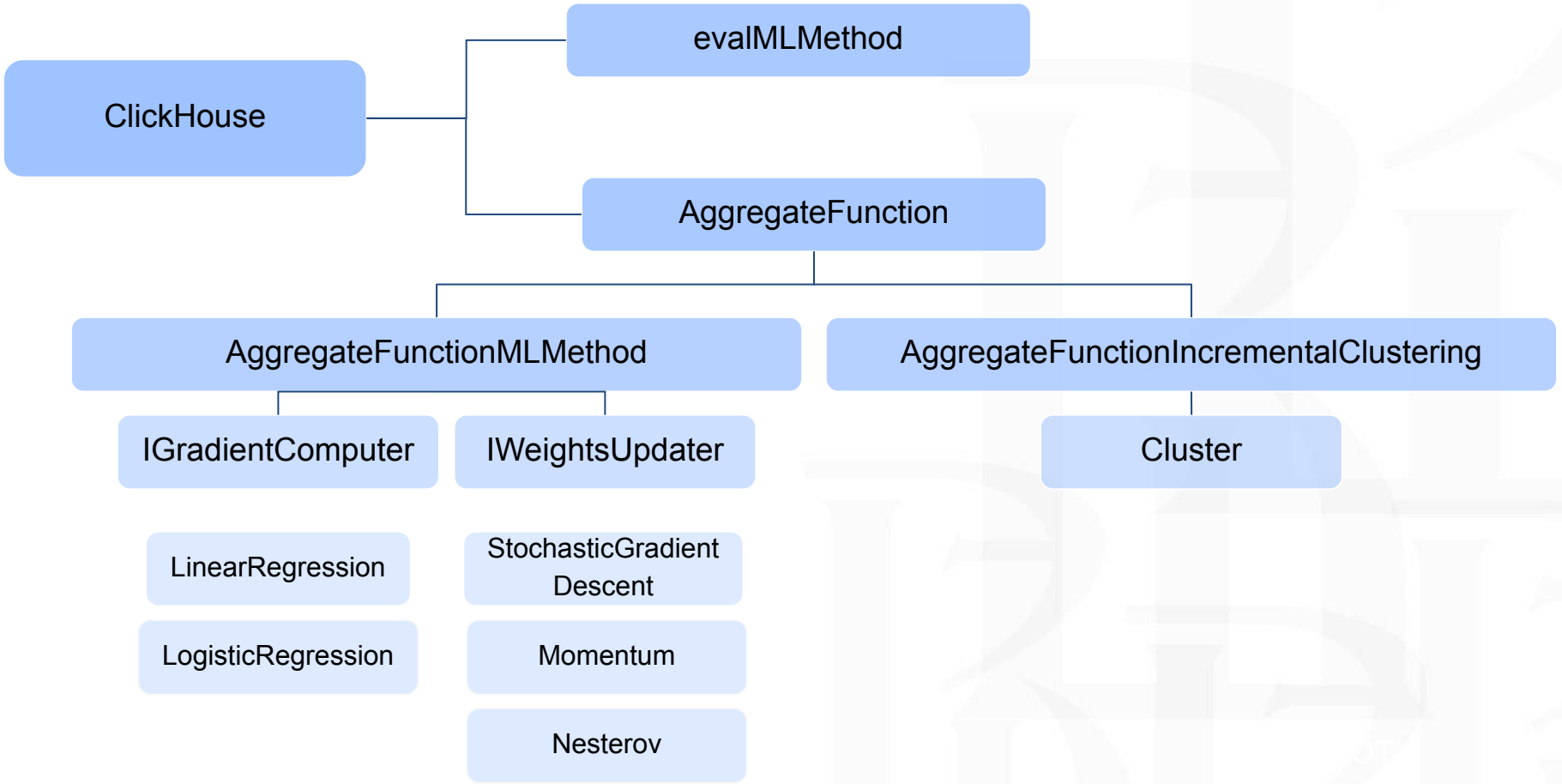
Программа применяется для поиска зависимостей и правил в данных о пользователях в режиме online, позволяет сделать некоторые выводы о статистике и предугадывать поведение очередного пользователя.

ТЕХНОЛОГИИ И ИНСТРУМЕНТЫ РЕАЛИЗАЦИИ

Используется инструментарий, соответствующий системе ClickHouse:

- Язык C++ (17-ый стандарт)
- Сборка проекта в системах CMake, Ninja
- Библиотека STL (Standard Template Library)
- Текстовые редакторы и IDE: CLion, Vim, Sublime Text
- Виртульная машина в Yandex.Cloud

ТЕХНОЛОГИИ И ИНСТРУМЕНТЫ РЕАЛИЗАЦИИ



ТЕХНОЛОГИИ И ИНСТРУМЕНТЫ РЕАЛИЗАЦИИ

Тестирование

Использовалось несколько способов тестирования:

1. Небольшие тестовые запросы для проверки корректности вычислений - обычные тесты в ClickHouse (.../queries/0_stateless)
2. Тесты производительности (performance tests) на больших объемах данных, которые также используются в ClickHouse.
3. Большие тесты на специально выбранных наборах данных для проверки реальной работоспособности методов - не используются во время планового тестирования.
4. Также дополнительно тестировалось качество предсказания после слияния (merge) нескольких состояний.

ВЫБОР МОДЕЛЕЙ, МЕТОДОВ И АЛГОРИТМОВ

Алгоритм

Обычно задачи линейной и логистической регрессии решаются с помощью градиентного спуска, а для больших наборов данных может использоваться метод стохастического градиентного спуска (SGD).

В силу стриминговой архитектуры агрегатных функции в системе ClickHouse, мы реализуем стохастические способы градиентного шага. Это не совсем SGD, т.к. каждый объект используется один раз при потоковой обработке данных.

Согласно архитектуре ClickHouse существовала необходимость осуществления объединения состояний предобученных моделей для дальнейшей работы. Это было реализовано в функциях *merge*.

Проверка реальной работоспособности будет затронута в следующих слайдах.

ОПИСАНИЕ IGradientComputer

Реализованные функции потерь

Две реализованные функции потерь соответствуют задачам линейной и логистической регрессии соответственно

1. Линейная регрессия (класс LinearRegression) использует квадратичную функцию потерь.
2. Логистическая регрессия (класс LogisticRegression) использует логистическую функцию потерь.

В обоих случаях также используется L2-регуляризация с произвольным коэффициентом.

АНАЛИЗ СУЩЕСТВУЮЩИХ РЕШЕНИЙ

В первую очередь мы сравнивали наши методы с соответствующими методами из библиотеки sklearn:

1. SGDRegressor для сравнения с stochasticLinearRegression
2. SGDClassifier для сравнения с stochasticLogisticRegression

В каждом случае настраивались параметры для того, чтобы модель показывала наилучшее качество:

1. learning rate - длина шага при градиентном спуске
2. L2 regularization coefficient - коэффициент компоненты оптимизируемого функционала, отвечающей за l2-регуляризацию.
3. Mini-Batch size - количество элементов, по которым считается градиент прежде чем сделать шаг в стохастическом методе.
4. Способ делать шаг в стохастическом методе.

ОПИСАНИЕ IWeightsUpdater

Используем обозначение $$\nabla J(batch) = \frac{1}{batch\_size} \sum_{i=0}^{batch\_size} \nabla f(x_{k_i})$$

StochasticGradient Descent

$$x_{k+1} = x_k - \alpha \nabla J(batch)$$

Momentum

$$accumulated\_gradient_{k+1} = \gamma \cdot accumulated\_gradient_k + \alpha \nabla J(batch)$$

$$x_{k+1} = x_k - accumulated\_gradient_{k+1}$$

Nesterov

$$accumulated\_gradient_{k+1} = \gamma \cdot accumulated\_gradient_k + \alpha \nabla J(batch - \gamma \cdot accumulated\_gradient_k)$$

$$x_{k+1} = x_k - accumulated\_gradient_{k+1}$$

ОПИСАНИЕ ВЫБРАННЫХ МЕТОДОВ

Пошаговое описание работы методов

Обучение (fit)

1. Добавляем объект к мини-батчу, сразу считая его вклад в градиент
   a. В случае с методом Нестерова и Momentum учитываем accumulated gradient.
   b. Вызываем метод compute класса LinearRegression или LogisticRegression.
2. Если количество добавленных элементов равно размеру мини-батча, то выполняем обновление состояния (функция *update_state*)
   a. Изменяем веса модели (weights, bias)
   b. Сбрасываем градиент, накопленный в мини-батче.
   c. Обновляем сглаженный градиент

1. | x_1 | ... | x_n |

IWeightsUpdater::add_to_batch → *accumulated_gradient*

2. *accumulated_gradient = new_accumulated_gradient*

   weights -= accumulated_gradient

   В зависимости от способа градиентного спуска

ОПИСАНИЕ ВЫБРАННЫХ МЕТОДОВ

Пошаговое описание работы методов

Предсказание (predict)

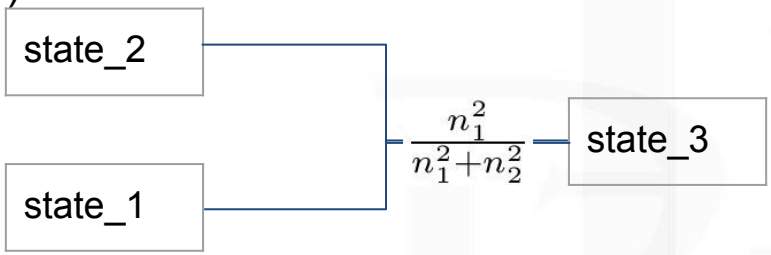
1. Вызываем метод predict класса IGradientComputer.
   a. Проверяем корректность данных в столбцах, соответствующих признакам объектов, на которых строятся предсказания.
   b. Учитывая обученные веса модели получаем предсказание для каждого объекта и заносим его в требуемый столбец (внешний цикл идет по столбцам, когда строим предсказания).

ОПИСАНИЕ ВЫБРАННЫХ АЛГОРИТМОВ *MERGE*

Пошаговое описание работы методов

Слияние (merge)

1. Вызывается функция *update_state* у правого: сбрасывается накопленный минибатч.
2. Итоговое состояние берется как средневзвешенное весов моделей.
3. Вызывается метод *IWeightsUpdater::merge* (сливаются значения *accumulated_gradient*)



РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Тестируем на выборке https://www.kaggle.com/mehdidag/black-friday/home. Оба набора данных были обработаны. Три способа обучения, в каждом случае выполнялся подбор параметров. Для сравнения использовалась метрика MSE. Размер обучающей выборки - 480000 объектов.

Метод	Результат MSE
linearRegression из ClickHouse	22379142
LinearRegression из sklearn	22313874
SGDRegressor из sklearn	22399580 (усредненное по 20 вызовам)

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ MERGE

Выполнялся merge нескольких состояний. По результатам оказалось, что взвешенное слияние состояний для случая, когда веса состояний заметно отличаются, приводит к тому, что новое состояние показывает результат хуже, чем более весомое состояния. Поэтому учитывалось среднеквадратическое весов.

Размеры данных для обучения отдельных состояний перед слиянием (через запятую) тыс. шт.	Результат MSE
50, 50, 50	23788181
30, 50, 70	23870467
120	23078578
120, 50	23280482
90	23563333
90, 90	23540933

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Было проделано несколько экспериментов на данных различных размеров (10000, 50000, 100000, 150000), сгенерированных функцией из библиотеки sklearn sklearn.datasets.make_classification.

№	Размер выборки	accuracy_score на SGDClassifier	accuracy_score на LogisticRegressionState	Параметры обучения LogisticRegressionState
1	10k	0.868	0.7575	[>0.85, (0.01, 0, 16, 'SGD')]
2	50k	0.876	0.7512	[>0.85, (0.01, 0, 32, 'SGD')]
3	100k	0.79825	0.7905	[>0.85, (0.01, 0, 64, 'SGD')]
4	150k	0.864533	0.856866	[>0.85, (0.01, 0, 64, 'SGD')]

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ MERGE

Результаты: 0.86013, 0.8624, 0.86013. Затем была получена модель, как merge трех предыдущих (state1 + state2 + state3).

Оно составило 0.8610, что соответствовало нашим ожиданиям.

Описание алгоритма инкрементальной кластеризации

1. Так как иногда необходимо делать merge, то будем хранить большее число кластеров (N), чем задал пользователь (K). В нынешней реализации N = 30 * K.
2. При добавлении новой точки:
   a. Если нынешнее кол-во кластеров меньше, чем N, то создаем новый кластер из точки
   b. Если кол-во кластеров равно N, то добавляем точку к ближайшему кластеру (с учетом штрафа за размер кластера)
3. При слиянии двух состояний:
   a. Если в одном из состояний кластеров меньше, чем N, то добавим все эти кластеры, как точки, в другое состояние
   b. Иначе разобьем кластера на пары так, чтобы в каждой паре было по кластеру из разных состояний, после чего пары сольем.

Описание метода add

Опишем метод add в виде псевдокода:

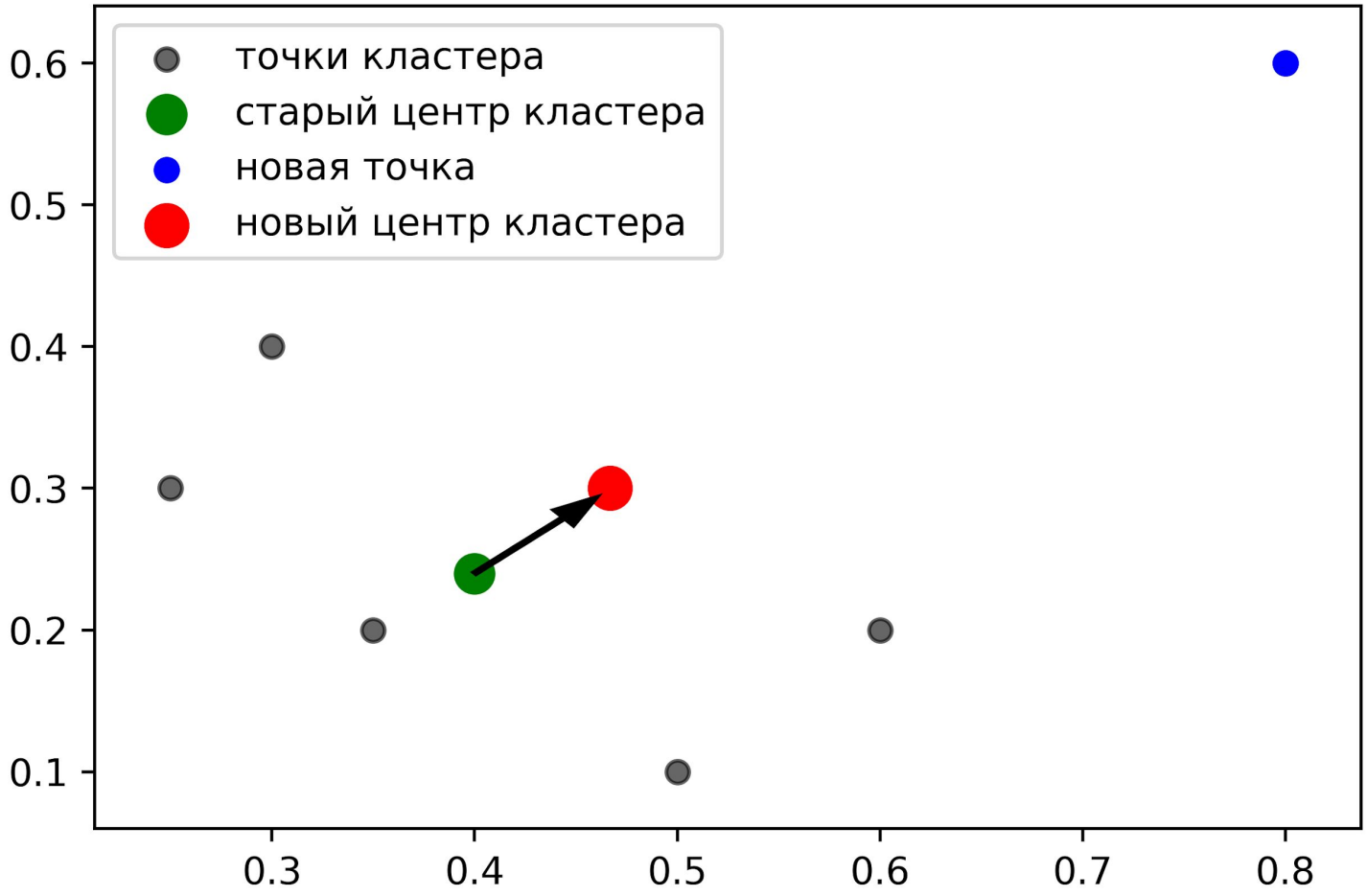
```
void State::add(point):
  if clusters.size() < multiplication_factor * clusters_num:
    clusters.emplace_back(point)
  else:
    closest_cluster = find_closest_cluster_with_penalty(point)
    clusters[closest_cluster].add(point)

Cluster::Cluster(point):
  points_num = 1
  coordinates = point

void Cluster::add(point):
  points_num += 1
  point_weight = 1 / points_num
  cluster_weight = 1 - point_weight
  coordinates = cluster_weight * coordinates + point_weight * point
```

(умножение и сложение векторов происходит покоординатно)

Описание метода add



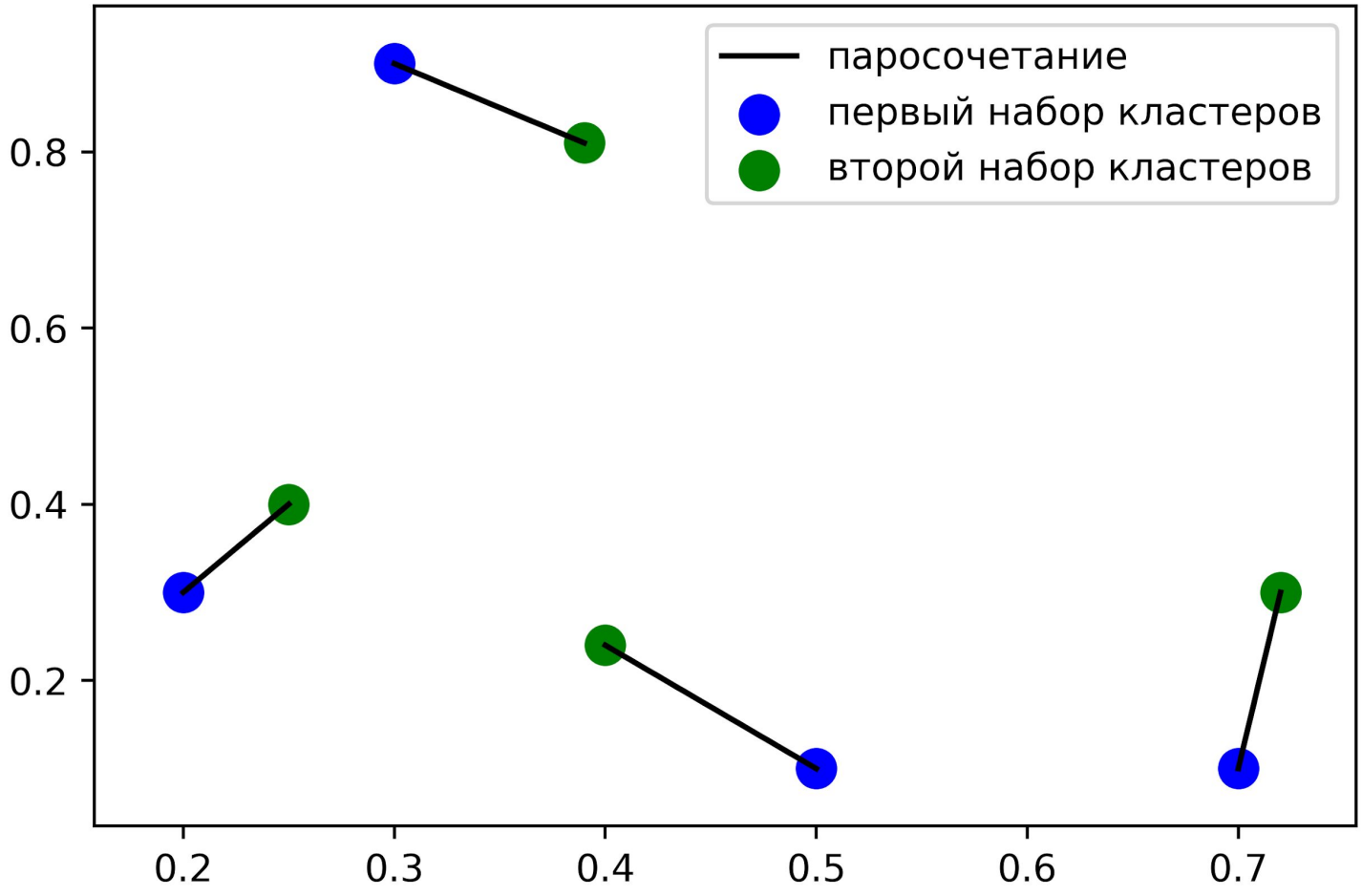
Описание метода merge

Опишем метод merge в виде псевдокода:

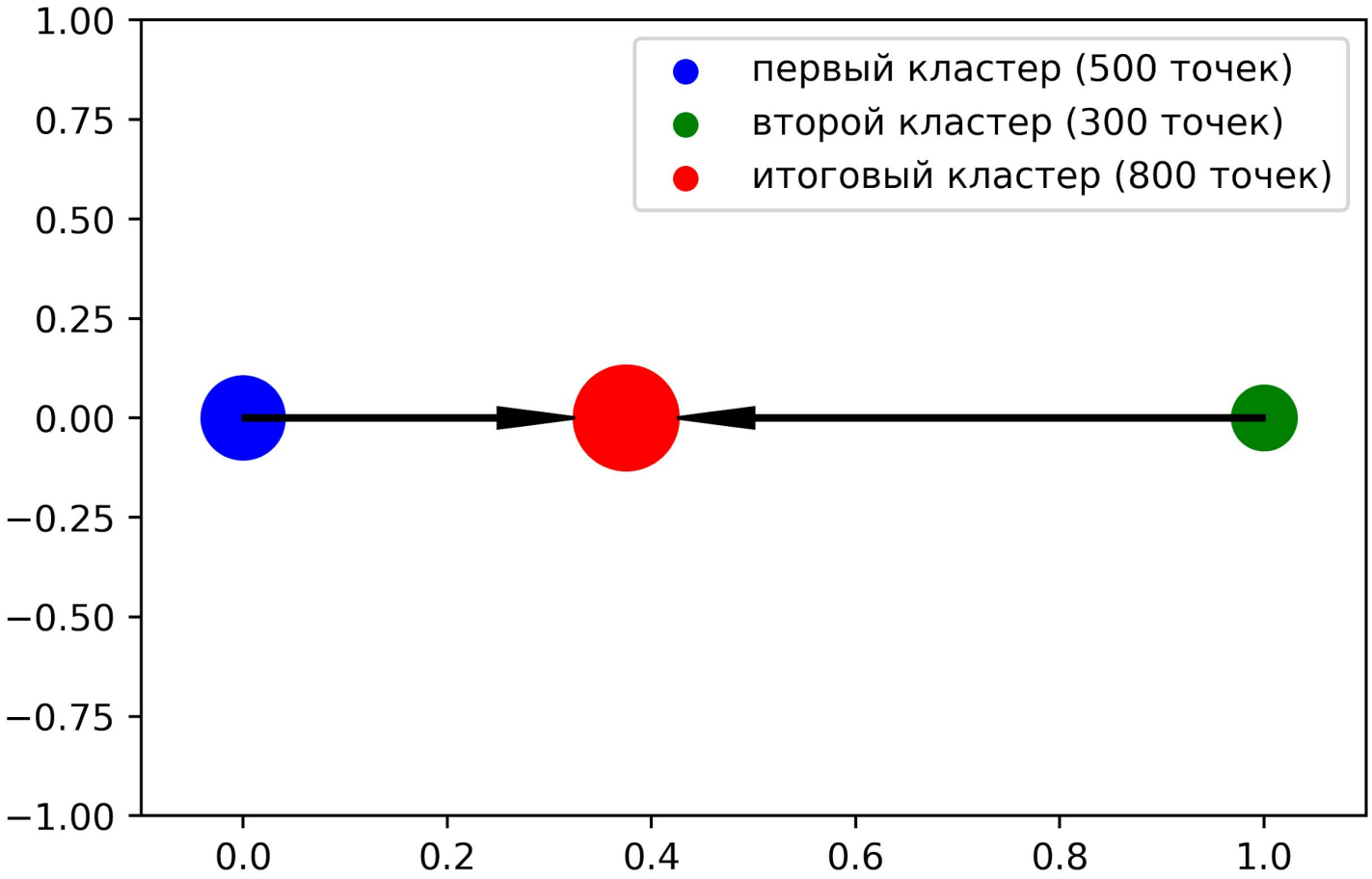
```
void State::merge(State other):
  if clusters.size() < multiplication_factor * clusters_num:
    for cluster in clusters:
      other.add(cluster.center())
  elif other.clusters.size() < multiplication_factor * clusters_num:
    for cluster in other.clusters:
      add(cluster.center())
  else:
    for pair in closest_pairs(clusters, other.clusters):
      pair.first.merge(pair.second)

void Cluster::merge(Cluster other):
  this_weight = points_num / (points_num + other.points_num)
  other_weight = 1 - this_weight
  points_num += other.points_numm
  coordinates *= this_weight
  coordinates += other_weight * other.coordinates
```

Описание метода merge



Описание метода merge



Описание метода predict

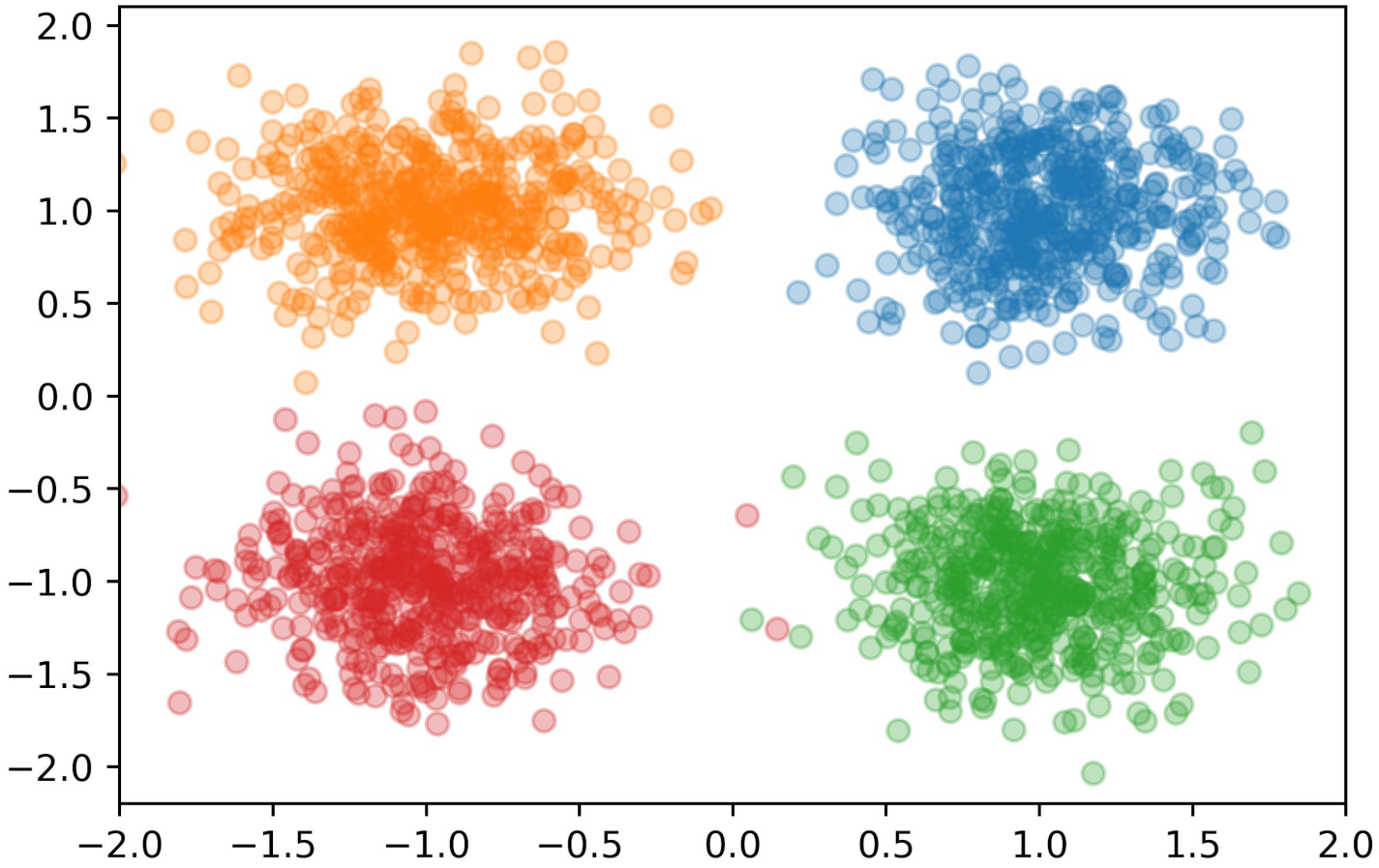
Алгоритм

Чтобы уменьшить кол-во кластеров с полученных N до необходимых K, будем запускать алгоритм K-Means. После чего каждой точке ставится в соответствие индекс ближайшего кластера из полученных K кластеров.

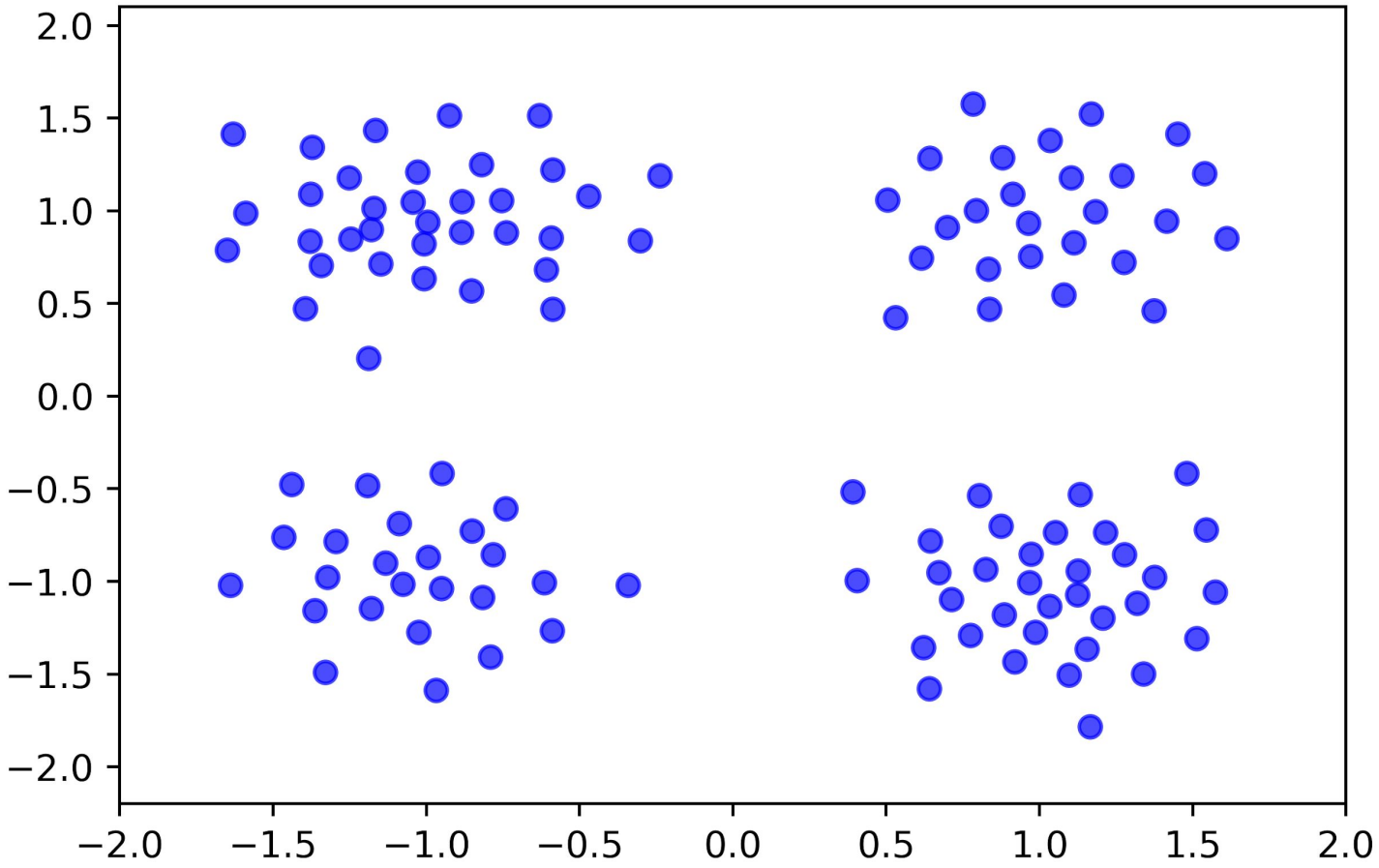
K-Means (Метод k-средних) - итеративный алгоритм кластеризации, который пытается минимизировать суммарное квадратичное отклонение точек кластеров от центров этих кластеров.

На каждой итерации алгоритм вычисляет центр для каждого кластера, после чего разбивает исходные точки на новые кластеры в зависимости от того, какой центр оказался ближе.

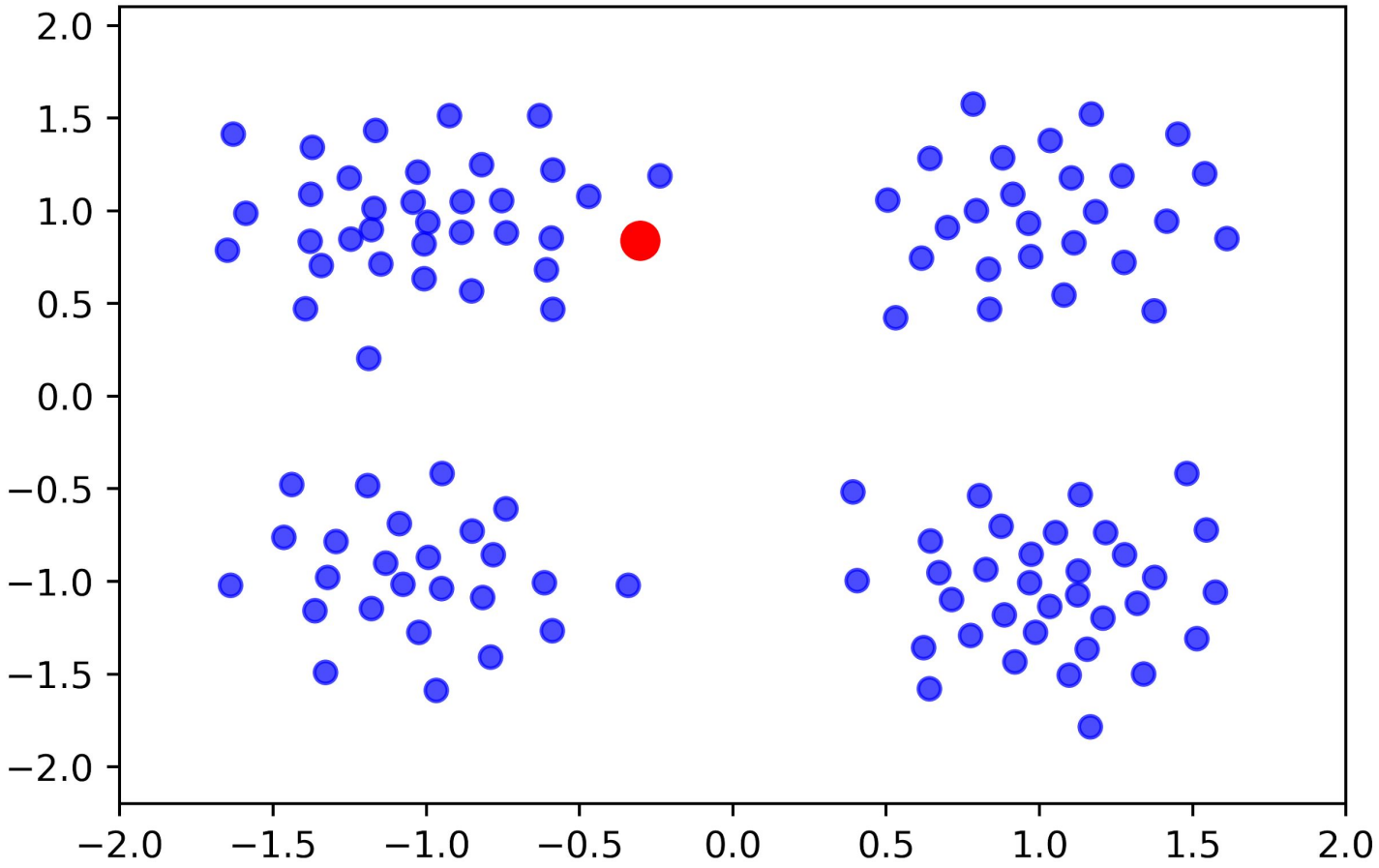
K-Means



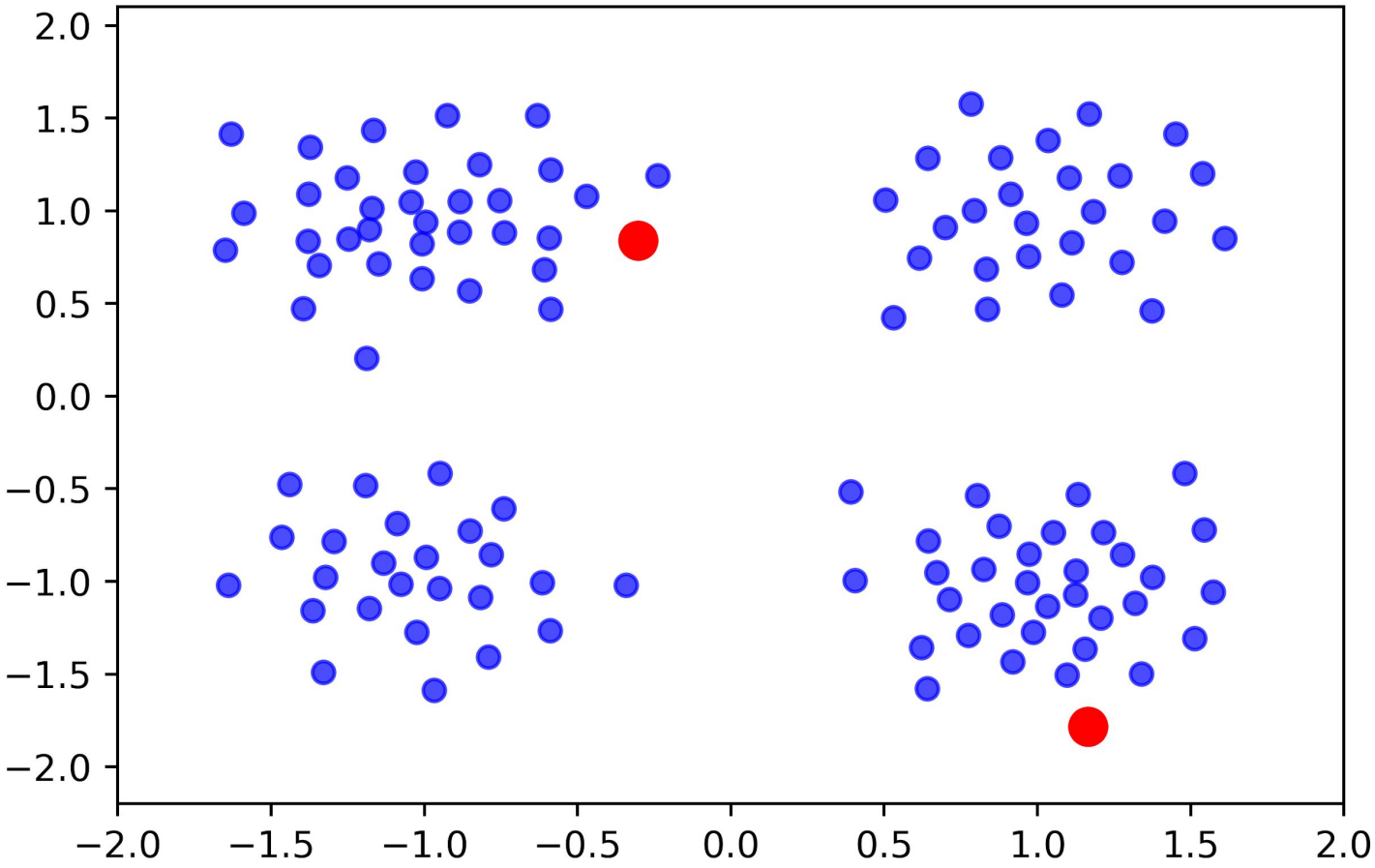
K-Means



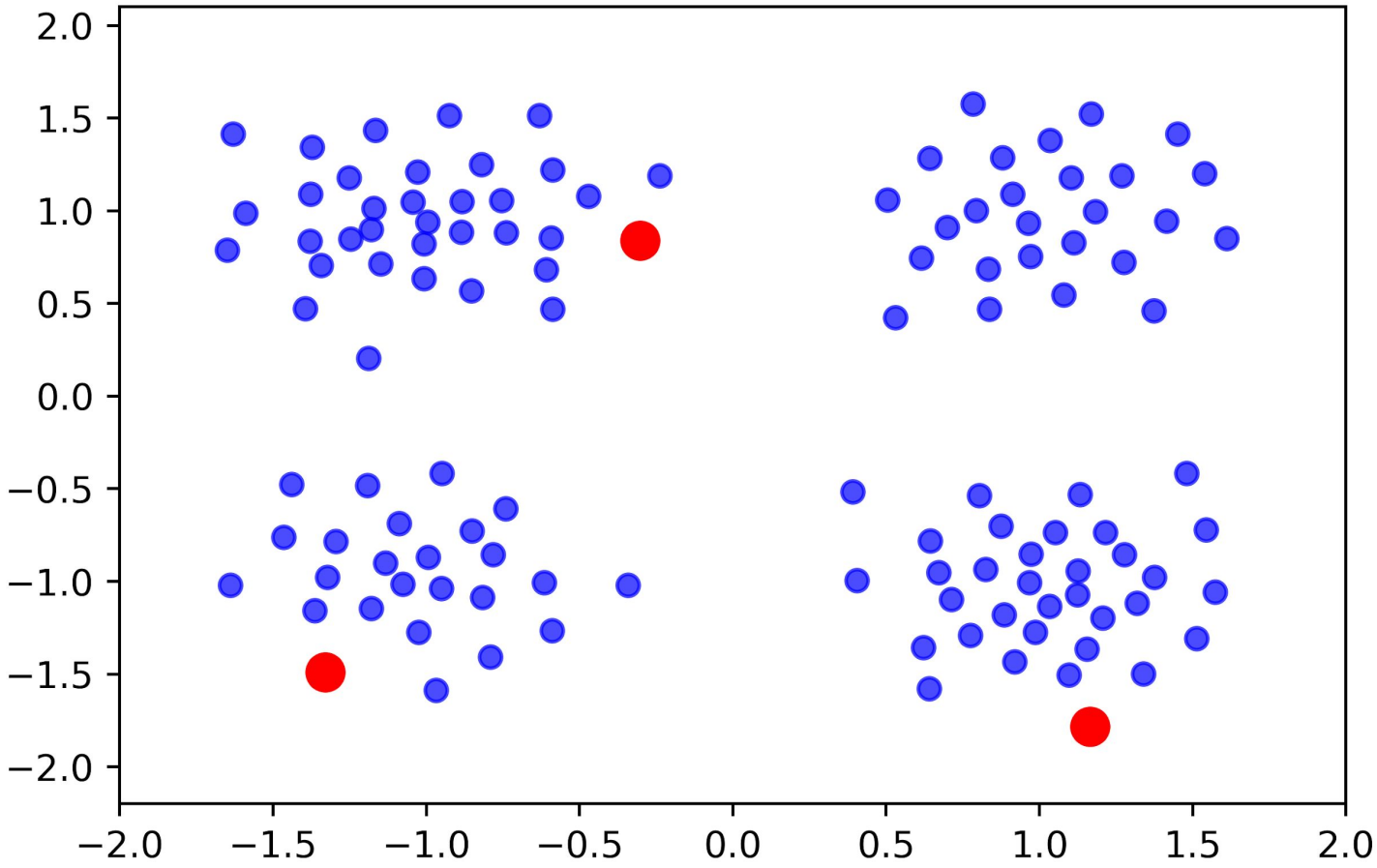
K-Means



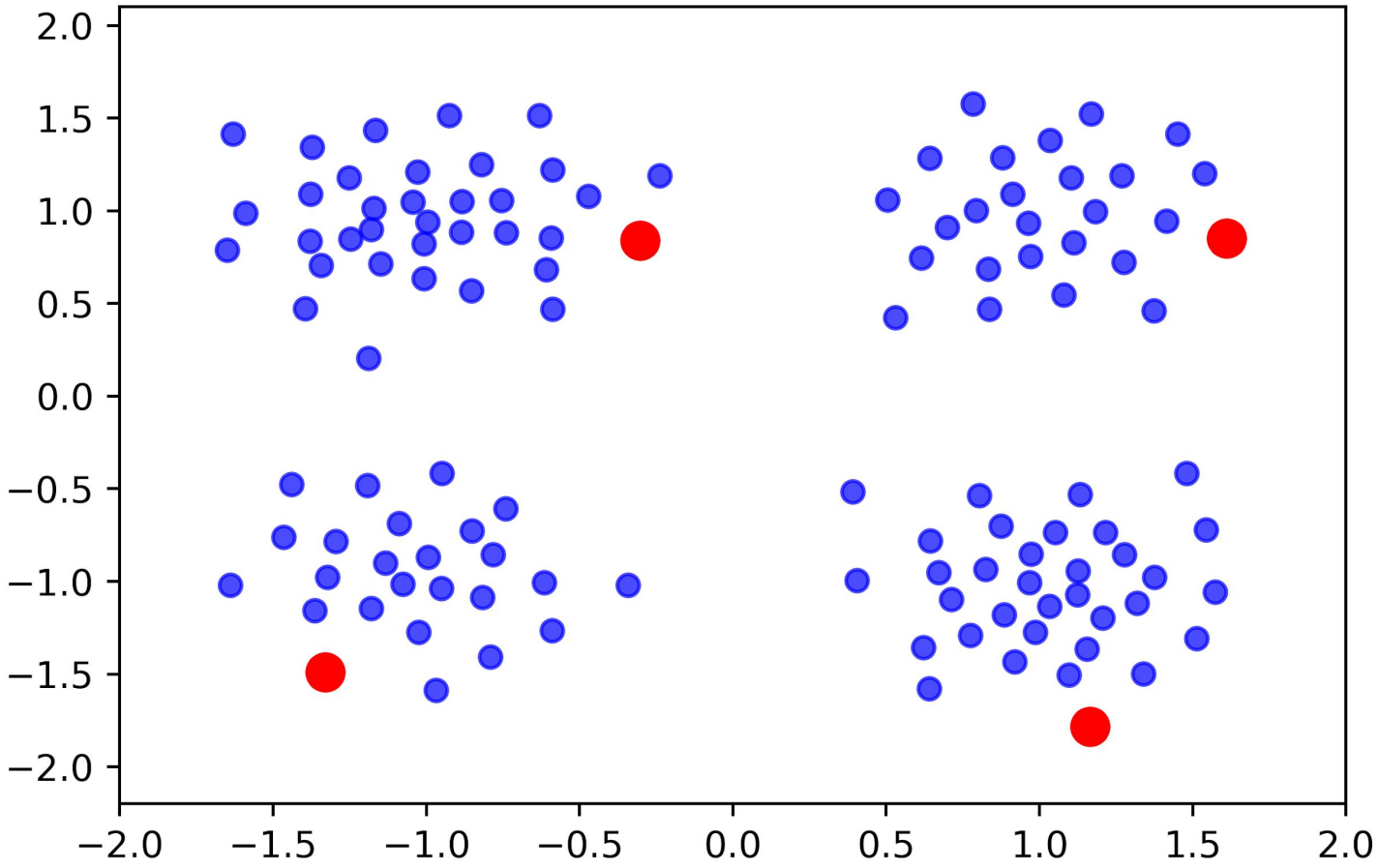
K-Means



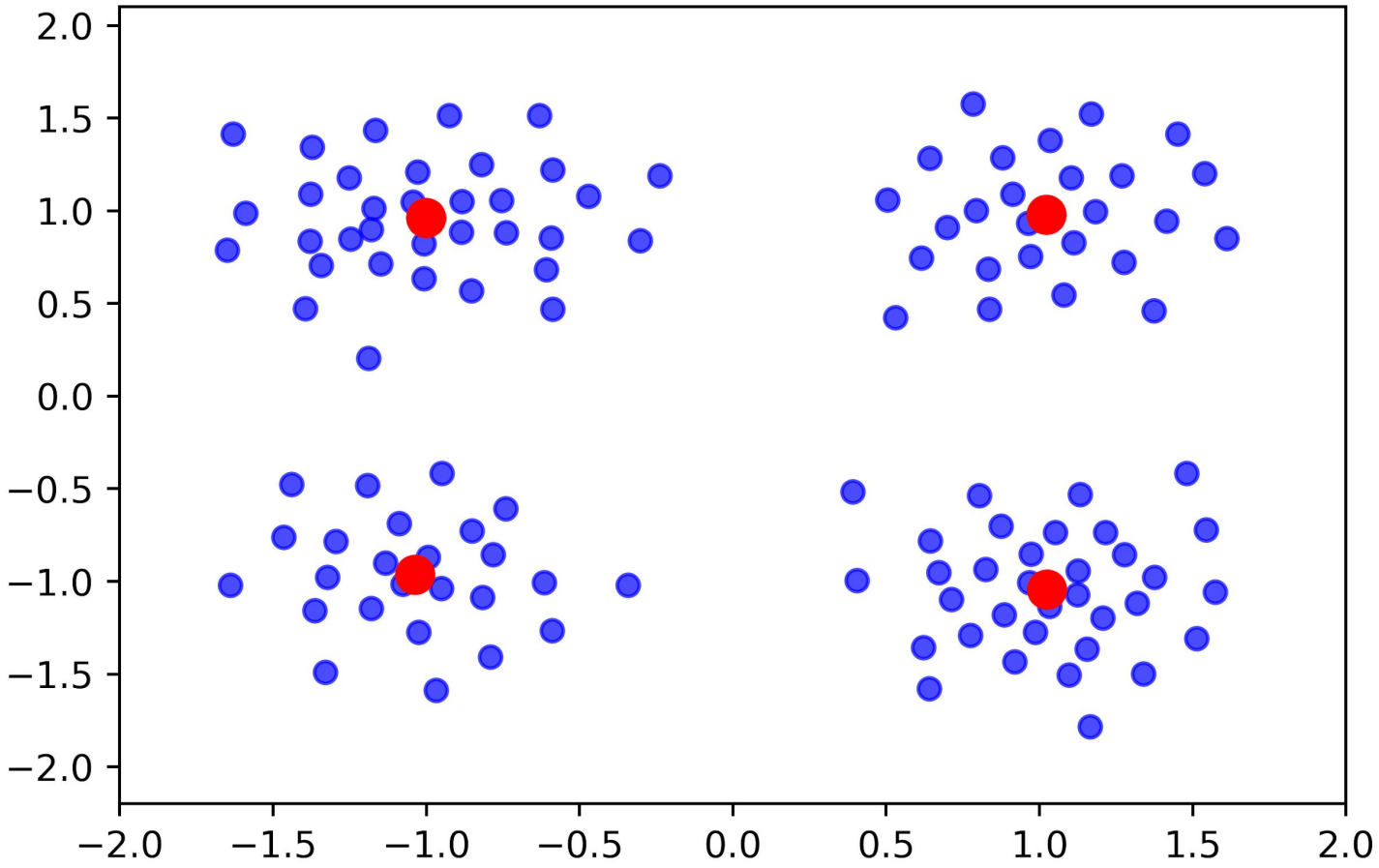
K-Means



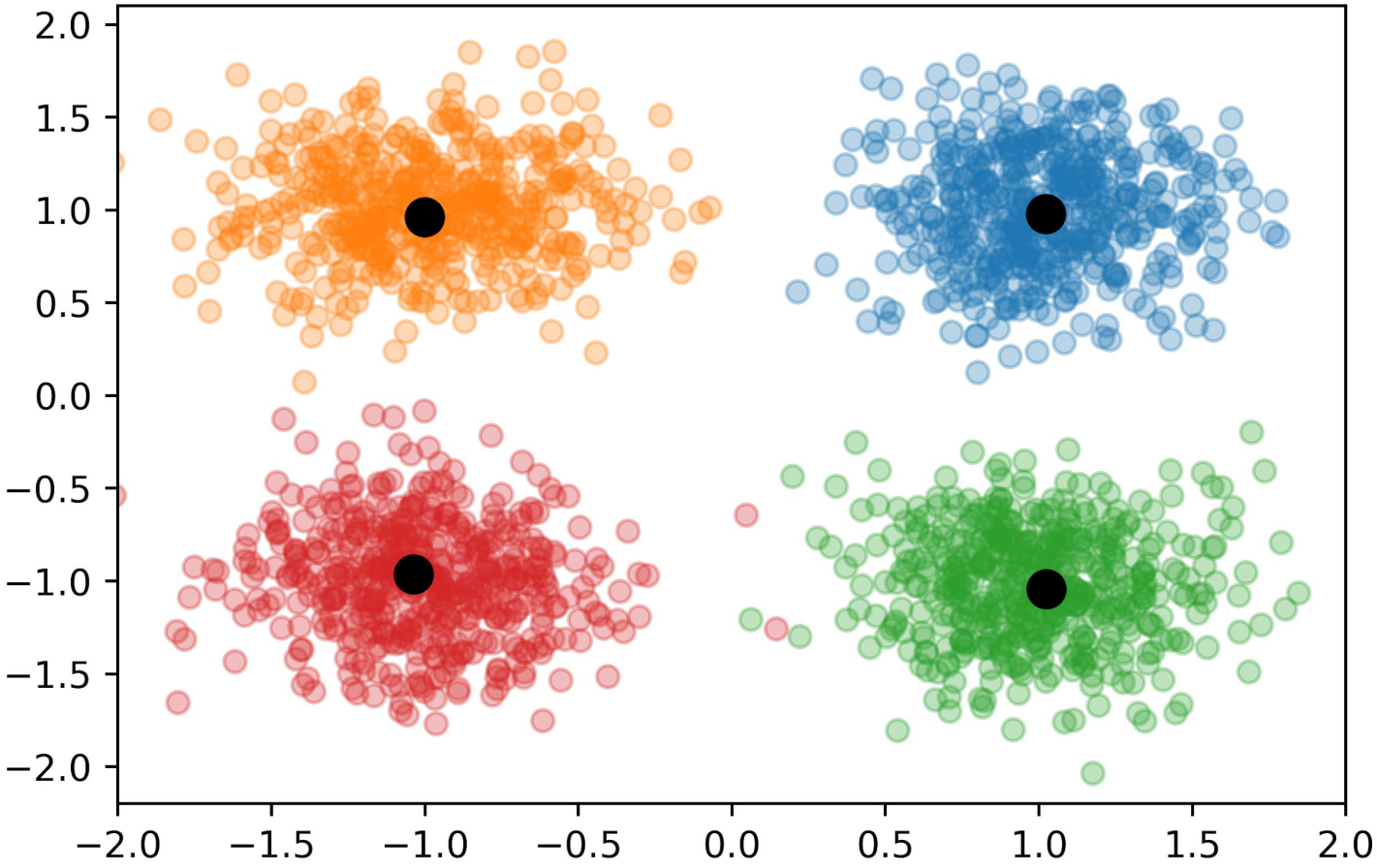
K-Means



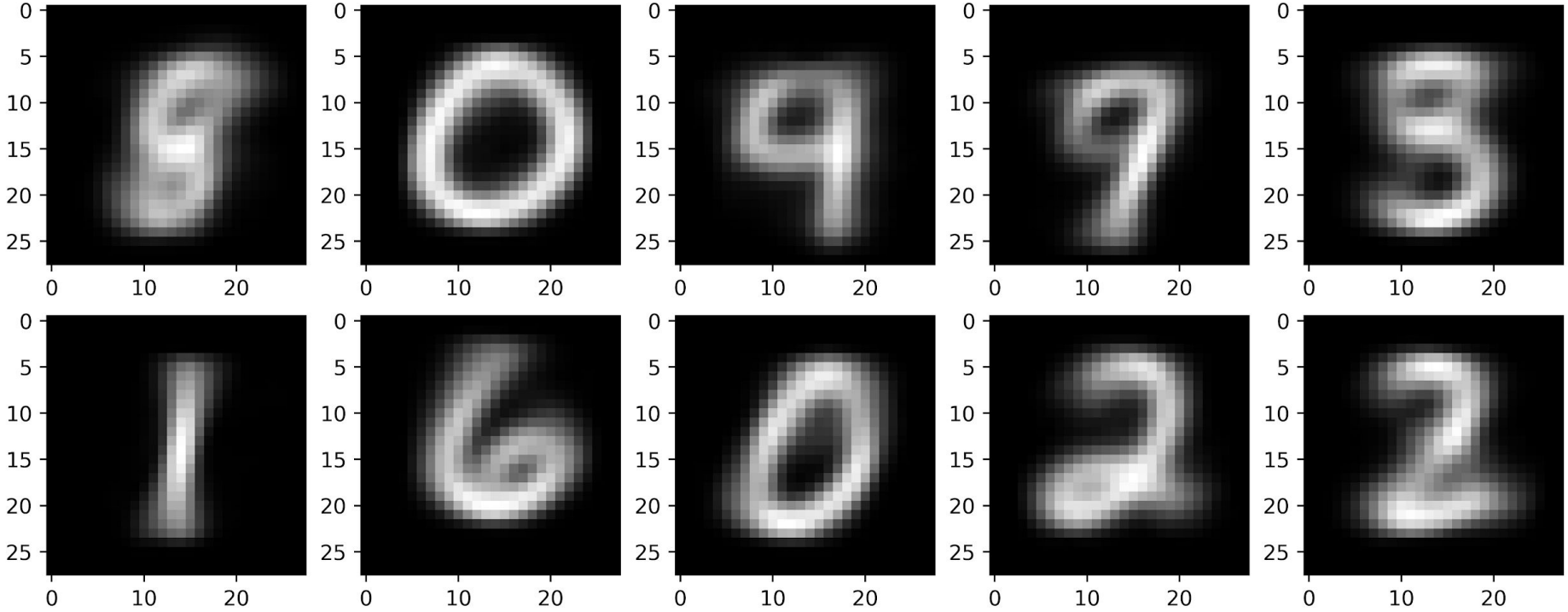
K-Means



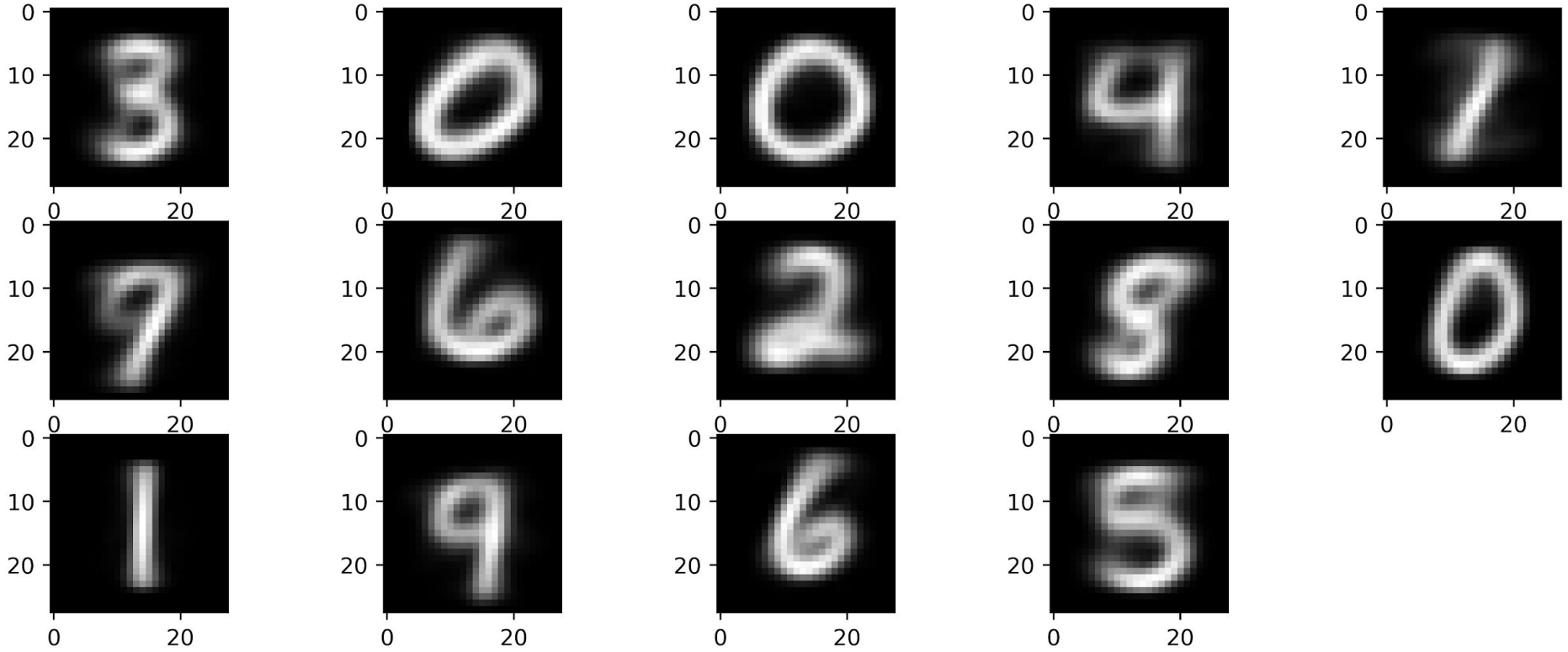
K-Means



MNIST



MNIST



РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Для тестирования было сгенерировано несколько наборов данных, где точки были нормально распределены вокруг центров кластеров, а также был взят датасет mnist из tensorflow.keras.datasets.mnist.

№	**Количество кластеров**	**Размерность**	**Количество точек в кластере**	**adjusted_rand_score для KMeans**	**adjusted_rand_score для IncrementalClusteringState**
1	4	2	40	1.0	1.0
2	4	2	1000	1.0	1.0
3	4	2	10000	0.988	0.987
4	10	2	100	0.868	0.837
5	10	2	1000	0.853	0.846
6	10	2	10000	0.846	0.841
7	10	784	7000	0.367	0.356

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Еще был проведен тест, который проверяет работу функции merge. Взяты были данные из эксперимента №5. Обучающие данные были разделены на 3 равные части. На каждой их этих частей были обучены IncrementalClusteringState и подсчитаны метрики качества: 0.793, 0.856, 0.789. Далее все 3 части были слиты в одну методом merge и подсчитана метрика для получившейся модели: 0.850. В качестве метрики опять бралась adjusted_rand_score.

ОПИСАНИЕ ВЫБРАННЫХ МЕТОДОВ evalMLMethod

Пошаговое описание работы методов

evalMLMethod

Функция реализована как функция Finalize, которая принимает состояние агрегатной функции и завершает ее, возвращая итоговый результат работы агрегатной функции. Отличие в том, что evalMLMethod также принимает параметры объектов, на которых требуется предсказать целевую переменную. Это позволяет использовать обученную модель много раз для предсказания на других данных.

ВЫВОДЫ ПО РАБОТЕ

Практическая значимость

Непосредственная интеграция методов машинного обучения - актуальное направление развития для современных баз данных. Примером может служить система Splunk.

Ссылки на наши пулл реквесты:

1. https://github.com/yandex/ClickHouse/pull/4943
2. https://github.com/yandex/ClickHouse/pull/5337
3. https://github.com/yandex/ClickHouse/pull/5411
4. https://github.com/yandex/ClickHouse/pull/5492

Направления дальнейшей работы

1. Реализация simpleLinearRegression (a.k.a. leastSqr) для векторов произвольного размера.
2. Реализовать уменьшение шага градиентного спуска.
3. Протестировать и внедрить еще один метод градиентного спуска Adam.
5. Показывать значение функции потерь на отложенной выборке во время обучения
6. Протестировать различные методы выбора начальных точек k-means (Например, k-means++)
7. Протестировать различные способы реализации метода merge в кластеризации (Например, минимальное паросочетание)
8. Реализация других методов машинного обучение (Например, стохастический SVM)

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *ГОСТ 19.101-77 Виды программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.*
2. *ГОСТ 19.102-77 Стадии разработки. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.*
3. *Конспекты по курсу “Машинное обучение” ФКН ВШЭ [Электронный ресурс] //URL: https://github.com/esokolov/ml-course-hse (Дата обращения: 29.05.2019, режим доступа: свободный).*

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Спасибо за внимание!

Кожихов А.О.
Конькова М.Н.
Кузнецов М.А.

Москва - 2019