СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Анализ данных — это процесс изучения и интерпретации данных с целью выявления скрытых закономерностей, выявления трендов и прогнозирования будущих событий. Он может быть использован в различных областях, таких как бизнес, наука, медицина, образование и т.д. Анализ данных включает в себя сбор, обработку, очистку, преобразование, хранение, визуализацию и интерпретацию данных. Он позволяет выявлять скрытые связи и зависимости между различными переменными, а также прогнозировать будущие события на основе имеющихся данных.

Аффентивный анализ — это метод выявления связей между объектами, основанный на анализе их отношений. Он широко применяется в различных областях, включая социологию, психологию, экономику и политологию. Аффентивный анализ позволяет выявить закономерности и тенденции в данных, а также определить влияние одних факторов на другие. Он может использоваться для прогнозирования будущих событий и принятия решений на основе полученных результатов.

Регрессионные модели — это статистические модели, которые используются для описания взаимосвязи между переменными. Они позволяют прогнозировать значения зависимой переменной на основе значений независимых переменных. Существует несколько типов регрессионных моделей, включая линейную регрессию, множественную регрессию и логистическую регрессию. Каждая из этих моделей имеет свои преимущества и недостатки, и выбор конкретной модели зависит от целей исследования и характера данных.

Методы кластеризации — это методы анализа данных, которые используются для группировки объектов или элементов в группы, называемые кластерами. Кластеризация может использоваться для различных целей, таких как классификация и сегментация данных, обнаружение аномалий, прогнозирование и т.д. Существует множество методов кластеризации, включая

иерархические методы, методы кластеризации на основе плотности, методы кластерного анализа на основе расстояния и методы кластеризации с использованием алгоритмов машинного обучения. Каждый метод имеет свои преимущества и недостатки, и выбор метода зависит от конкретных требований задачи и характеристик данных.

Минимальное остаточное дерево — это дерево, которое содержит все вершины и рёбра графа, но при этом имеет наименьший вес среди всех возможных деревьев, содержащих все вершины и ребра графа. Для построения минимального остаточного дерева можно использовать алгоритм Прима или Крускала.

1 АССОЦИАТИВНЫЕ ПРАВИЛА

1.1 Постановка задачи

Придумать предметную область и на основе выбранной предметной области реализовать алгоритм Apriori.

1.2 Описание алгоритма

Аффинитивный анализ (affinity analysis) – один из распространенных методов Data Mining. Его название происходит от английского слова affinity, которое в переводе означает «близость», «сходство». Цель дангоro метода исследование взаимной связи между событиями, которые происходят совместно. Разновидностью аффинитивного анализа является анализ рыночной корзины, цель которого обнаружить ассоциации между различными событиями, то есть найти правила для количественного описания взаимной связи между двумя или более событиями. Такие правила называются ассоциативными правилами.

Алгоритм Apriori. При практической реализации систем поиска ассоциативных правил используют различные методы, которые позволяют снизить пространство поиска до размеров, обеспечивающих приемлемые вычислительные и временные затраты, например, алгоритм Apriori.

В основе алгоритма Apriori лежит понятие частого набора, который также можно назвать частым предметным набором, часто встречающимся множеством (соответственно, он связан с понятием частоты). Под частотой понимается простое количество транзакций, в которых содержится данный предметный набор. Тогда частыми наборами будут те из них, которые встречаются чаще, чем в заданном числе транзакций.

Примерами приложения ассоциативных правил могут быть следующие задачи:

- выявление наборов товаров, которые в супермаркетах часто покупаются вместе или никогда не покупаются вместе;

- определение доли клиентов, положительно относящихся к нововведениям в их обслуживании;
- определение профиля посетителей веб-ресурса;
- определение доли случаев, в которых новое лекарство оказывает опасный побочный эффект.

Следующее важное понятие – предметный набор. Это непустое множество предметов, появившихся в одной транзакции.

Анализ рыночной корзины – это анализ наборов данных для определения комбинаций товаров, связанных между собой, иными словами, производится поиск товаров, присутствие которых в транзакции влияет на вероятность наличия других товаров или комбинаций товаров.

Современные кассовые аппараты в супермаркетах позволяют собирать информацию о покупках, которая может храниться в базе данных. Затем накопленные данные могут использоваться для построения систем поиска ассоциативных правил.

Поддержка ассоциативною правила – это число транзакций, которые содержат kar условие, так и следствие.

Например, для ассоциации $A \rightarrow B$ можно записать:

$$S(A \rightarrow B) = P(A \cap B) = \frac{\text{количество транзакций, содержащиих A и B}}{\text{общее количество транзакции}} \quad (1.1)$$

Достоверность ассоциативного правила $A \rightarrow B$ представляет собой меру точности правила и определяется как отношение количества транзакций, содержащих и условие, и следствие, к количеству транзакций, содержащих только условие:

$$C(A \rightarrow B) = \frac{P(A \cap B)}{P(A)} = \frac{\text{количество транзакций, содержащиих A и B}}{\text{количество транзакций, содержащих только A}} (1.2)$$

Если поддержка и достоверность достаточно высоки, можно с большой вероятностью утверждать, что любая будущая транзакция, которая включает условие, будет также содержать и следствие.

Лифт – отношение частоты появления условия в транзакциях, которые также содержат и следствие к частоте появления следствия в целом. Значения лифта большие 1 показывают, что условие чаще появляется в транзакциях, содержащих следствие, чем в остальных. Можно утверждать, что лифт является обобщенной мерой связи двух предметных наборов: при значениях лифта больше 1 связь положительная, при 1 она отсутствует, а при значениях меньше 1 – отрицательная.

Лифт (оригинальное название — интерес) вычисляется следующим образом:

$$L(A \to B) = \frac{C(A \to B)}{S(B)} = \frac{\text{Достоверность (A} \to B)}{\text{количество транзакций, содержащиux } B} \qquad (1.3)$$

1.3 Описание предметной области

Исходные данные из магазина автозапчастей представленные (Рисунок 1.1).

```
"receipt_number","product"
15,"Передние тормозные колодки"
15,"Моторное масло"
15,"Масляный фильтр ДВС"
30,"Фильтр салона"
30,"Свечи зажигания на моделях с бензиновым ДВС"
30,"Передние тормозные колодки"
30,"Моторное масло"
30,"Воздушный фильтр ДВС"
30,"Масляный фильтр ДВС"
30,"Топливный фильтр тонкой очистки"
40,"Передние тормозные колодки"
40,"Фильтр салона"
45,"Моторное масло"
45,"Масляный фильтр ДВС"
50,"Передние тормозные колодки"
50,"Масло в коробке передач"
50,"Жидкость ГУР"
50,"Масляный фильтр коробки передач"
55,"Тормозная жидкость"
55,"Передние тормозные колодки"
60,"Свечи зажигания на моделях с бензиновым ДВС"
60,"Задние тормозные колодки"
60,"Моторное масло"
60,"Воздушный фильтр ДВС"
60,"Масляный фильтр ДВС"
60,"Топливный фильтр тонкой очистки"
60,"Фильтр салона"
75,"Приводной ремень"
75,"Передние тормозные колодки"
75,"Основной аккумулятор"
75,"Моторное масло"
75,"Масляный фильтр ДВС"
80,"Рычаги подвески"
80,"Передние тормозные колодки"
80,"Фильтр салона"
90,"Свечи зажигания на моделях с бензиновым ДВС"
90,"Шрусы или их составные части"
90,"Моторное масло"
90,"Воздушный фильтр ДВС"
90,"Масляный фильтр ДВС"
90,"Топливный фильтр тонкой очистки"
```

Рисунок 1.1 – Исходные данные

1.4 Ассоциотивные правила в программе deductor

В программе Deductor Визуализатор "Правила" отображает ассоциативные правила в виде списка правил. Этот список представлен таблицей со столбцами:

"Номер правила", "Условие", "Следствие", "Поддержка, %", "Поддержка, Количество", "Достоверность", "Лифт" (Рисунок 2).

Правил: 595 из 595 | Фильтр: Без фильтрации

№	Номер правила	Условие	Следствие	Поддержка		Достоверность	Лифт
				Кол-во	%		
1	215	Масляный фильтр ДВС Моторное масло Передние тормозные коло	Щетки стеклоочистителя	4	26,67	57,14	2,143
2	87	Моторное масло Передние тормозные коло	Щетки стеклоочистителя	4	26,67	57,14	2,143
3	75	Масляный фильтр ДВС Передние тормозные коло	Щетки стеклоочистителя	4	26,67	57,14	2,143
4	66	Масляный фильтр ДВС Моторное масло	Щетки стеклоочистителя	4	26,67	44,44	1,667
5	19	Передние тормозные коло	Щетки стеклоочистителя	4	26,67	36,36	1,364
6	14	Моторное масло	Щетки стеклоочистителя	4	26,67	44,44	1,667
7	10	Масляный фильтр ДВС	Щетки стеклоочистителя	4	26,67	44,44	1,667
8	18	Передние тормозные коло	Фильтр салона тонкой очи	5	33,33	45,45	1,136
9	96	Передние тормозные коло	Фильтр салона Фильтр салона тонкой очи	5	33,33	45,45	1,136
10	17	Передние тормозные коло	Фильтр салона	5	33,33	45,45	1,136
11	411	Масляный фильтр ДВС Моторное масло Передние тормозные коло	Топливный фильтр тонкой Щетки стеклоочистителя	4	26,67	57,14	2,143
12	247	Моторное масло Передние тормозные коло	Топливный фильтр тонкой Щетки стеклоочистителя	4	26,67	57,14	2,143
13	237	Масляный фильтр ДВС Передние тормозные коло	Топливный фильтр тонкой Щетки стеклоочистителя	4	26,67	57,14	2,143
14	228	Масляный фильтр ДВС Моторное масло	Топливный фильтр тонкой Щетки стеклоочистителя	4	26,67	44,44	1,667
15	95	Передние тормозные коло	Топливный фильтр тонкой Щетки стеклоочистителя	4	26,67	36,36	1,364

Рисунок 1.2 – Анализ данных в программе Deductor

1.5 Расчёт поддержки

Возьмем ассоциацию передние тормозные колодки и моторное масло. Поскольку количество транзакций, содержащих как передние тормозные колодки, так и моторное масло, равно 7, а общее число транзакций 15, то поддержка данной ассоциации будет:

$$S(A \to B) = P(A \cap B) = \frac{\text{количество транзакций, содержащих A и B}}{\text{общее количество транзакции}} \quad (1.4)$$

$$(\text{передние тормозные колодки} \to \text{моторное масло}) = \frac{7}{15} = 0{,}46 \quad (1.5)$$

1.6 Расчёт достоверности

Возьмем ассоциацию передние тормозные колодки и моторное масло.

Поскольку количество транзакций, содержащих только моторное масло (условие), равно 4, то достоверность данной ассоциации будет:

$$C(A \to B) = \frac{P(A \cap B)}{P(A)} = \frac{\text{количество транзакций, содержащиux A и B}}{\text{количество транзакций, содержащих только A}} \quad (1.6)$$

$$C(\text{передние тормозные колодки } \to \text{моторное масло}) = \frac{7}{11} = 0{,}63 \quad (1.7)$$

1.7 Расчёт лифта

Лифт – отношение частоты появления условия в транзакциях, которые также содержат и следствие к частоте появления следствия в целом.

$$L(A \to B) = \frac{C(A \to B)}{S(B)} = \frac{\text{Достоверность (A} \to B)}{\text{количество транзакций, содержащиux } B} \quad (1.8)$$

$$L(A \to B) = \frac{C(\text{передние тормозные колодки} \to \text{моторное масло})}{S(\text{моторное масло})} = \frac{0.63}{0.60} = 1{,}06 \quad (1.9)$$

1.8 Программная реализация

Создадим ассоциативные правила в программной реализации на языке высокого уровня Python [2].

Полученный результат работы программы ассоциативных правил (Рисунок 1.3).

	Suported	Reliability	Lift
Моторное масло | Топливный фильтр тонкой очистки, Свечи зажигания на моделях с бензиновым ДВС, Воздушный фильтр ДВС, Масляный фильтр ДВС	30.0	50.0	1.67
Масляный фильтр ДВС | Свечи зажигания на моделях с бензиновым ДВС, Топливный фильтр тонкой очистки, Воздушный фильтр ДВС	30.0	50.0	1.67
Масляный фильтр ДВС | Топливный фильтр тонкой очистки, Воздушный фильтр ДВС	30.0	50.0	1.67
Масляный фильтр ДВС | Свечи зажигания на моделях с бензиновым ДВС, Моторное масло, Воздушный фильтр ДВС	30.0	50.0	1.67
Масляный фильтр ДВС | Свечи зажигания на моделях с бензиновым ДВС, Моторное масло, Топливный фильтр тонкой очистки	30.0	50.0	1.67
Масляный фильтр ДВС | Свечи зажигания на моделях с бензиновым ДВС, Воздушный фильтр ДВС, Моторное масло	30.0	50.0	1.67
Масляный фильтр ДВС | Свечи зажигания на моделях с бензиновым ДВС, Воздушный фильтр ДВС, Топливный фильтр тонкой очистки	30.0	50.0	1.67
Масляный фильтр ДВС | Свечи зажигания на моделях с бензиновым ДВС, Топливный фильтр тонкой очистки, Моторное масло	30.0	50.0	1.67
Масляный фильтр ДВС | Моторное масло, Свечи зажигания на моделях с бензиновым ДВС, Воздушный фильтр ДВС	30.0	50.0	1.67
Моторное масло | Топливный фильтр тонкой очистки, Свечи зажигания на моделях с бензиновым ДВС, Масляный фильтр ДВС, Воздушный фильтр ДВС	30.0	50.0	1.67
Масляный фильтр ДВС | Моторное масло, Свечи зажигания на моделях с бензиновым ДВС, Топливный фильтр тонкой очистки	30.0	50.0	1.67
Масляный фильтр ДВС | Моторное масло, Воздушный фильтр ДВС, Свечи зажигания на моделях с бензиновым ДВС	30.0	50.0	1.67
Масляный фильтр ДВС | Моторное масло, Воздушный фильтр ДВС, Топливный фильтр тонкой очистки	30.0	50.0	1.67
Масляный фильтр ДВС | Моторное масло, Топливный фильтр тонкой очистки, Свечи зажигания на моделях с бензиновым ДВС	30.0	50.0	1.67
Масляный фильтр ДВС | Моторное масло, Топливный фильтр тонкой очистки, Воздушный фильтр ДВС	30.0	50.0	1.67
Масляный фильтр ДВС | Воздушный фильтр ДВС, Свечи зажигания на моделях с бензиновым ДВС, Моторное масло	30.0	50.0	1.67
Масляный фильтр ДВС | Топливный фильтр тонкой очистки, Моторное масло	30.0	50.0	1.67
Масляный фильтр ДВС | Топливный фильтр тонкой очистки, Свечи зажигания на моделях с бензиновым ДВС	30.0	50.0	1.67
Масляный фильтр ДВС | Воздушный фильтр ДВС, Топливный фильтр тонкой очистки	30.0	50.0	1.67
Масляный фильтр ДВС | Воздушный фильтр ДВС, Моторное масло	30.0	50.0	1.67
Масляный фильтр ДВС | Воздушный фильтр ДВС, Свечи зажигания на моделях с бензиновым ДВС	30.0	50.0	1.67
Масляный фильтр ДВС | Моторное масло, Топливный фильтр тонкой очистки	30.0	50.0	1.67
Масляный фильтр ДВС | Моторное масло, Воздушный фильтр ДВС	30.0	50.0	1.67
Масляный фильтр ДВС | Моторное масло, Свечи зажигания на моделях с бензиновым ДВС	30.0	50.0	1.67
Масляный фильтр ДВС | Свечи зажигания на моделях с бензиновым ДВС, Топливный фильтр тонкой очистки	30.0	50.0	1.67
Масляный фильтр ДВС | Свечи зажигания на моделях с бензиновым ДВС, Воздушный фильтр ДВС	30.0	50.0	1.67
Масляный фильтр ДВС | Свечи зажигания на моделях с бензиновым ДВС, Моторное масло	30.0	50.0	1.67
Масляный фильтр ДВС | Топливный фильтр тонкой очистки	30.0	50.0	1.67
Масляный фильтр ДВС | Воздушный фильтр ДВС	30.0	50.0	1.67
Масляный фильтр ДВС | Свечи зажигания на моделях с бензиновым ДВС	30.0	50.0	1.67
Моторное масло | Топливный фильтр тонкой очистки, Масляный фильтр ДВС, Воздушный фильтр ДВС, Свечи зажигания на моделях с бензиновым ДВС	30.0	50.0	1.67
Моторное масло | Топливный фильтр тонкой очистки, Масляный фильтр ДВС, Свечи зажигания на моделях с бензиновым ДВС, Воздушный фильтр ДВС	30.0	50.0	1.67
Моторное масло | Топливный фильтр тонкой очистки, Воздушный фильтр ДВС, Масляный фильтр ДВС, Свечи зажигания на моделях с бензиновым ДВС	30.0	50.0	1.67
Масляный фильтр ДВС | Воздушный фильтр ДВС, Свечи зажигания на моделях с бензиновым ДВС, Топливный фильтр тонкой очистки	30.0	50.0	1.67
Масляный фильтр ДВС | Воздушный фильтр ДВС, Моторное масло, Свечи зажигания на моделях с бензиновым ДВС	30.0	50.0	1.67
Масляный фильтр ДВС | Воздушный фильтр ДВС, Моторное масло, Топливный фильтр тонкой очистки	30.0	50.0	1.67
Масляный фильтр ДВС | Моторное масло, Воздушный фильтр ДВС, Свечи зажигания на моделях с бензиновым ДВС, Топливный фильтр тонкой очистки	30.0	50.0	1.67
Масляный фильтр ДВС | Топливный фильтр тонкой очистки, Воздушный фильтр ДВС, Свечи зажигания на моделях с бензиновым ДВС, Моторное масло	30.0	50.0	1.67

Рисунок 1.3 – Результат работы программы

2 K-MEANS

2.1 Постановка задачи

Реализовать алгоритм объектов k-средних для произвольных данных при помощи евклидовое расстояние.

2.2 Описание алгоритма

Сегодня предложено несколько десятков алгоритмов кластеризации и еще больше их разновидностей. Несмотря на это, в Data Mining применяются в первую очередь понятные и простые в использовании алгоритмы. К таким относится алгоритм k-means — в русскоязычном варианте k-средних (от англ. mean — «среднее значение»). Его основная идея состоит в том, что для выборки данных, содержащей n записей (объектов), задается число кластеров — k, которое должно быть сформировано. Затем алгоритм разбивает все объекты выборки на k разделов (k < n), которые и представляют собой кластеры.

Алгоритм выполняется в четыре шага:

1) задается число кластеров — k, которое должно быть сформировано из объектов исходной выборки;

2) случайным образом выбирается k записей исходной выборки, которые будут служить начальными центрами кластеров. Начальные точки, из которых потом вырастает кластер, часто называют «семенами» (от англ. seeds — «семена», «посевы»). Каждая такая запись представляет собой своего рода «эмбрион» кластера, состоящий только из одного элемента;

3) для каждой записи исходной выборки определяется ближайший к ней центр кластера. Чтобы определить, в сферу влияния какого центра кластера входит та или иная запись, вычисляется расстояние от каждой записи до каждого центра в многомерном пространстве признаков и выбирается то «семя», для которого данное расстояние минимальное;

4) в анализе данных распространенной оценкой близости между объектами является метрика, или способ задания расстояния. Выбор конкретной метрики зависит от аналитика и конкретной задачи. Наиболее популярные метрики — евклидово расстояние и расстояние Манхэттена.

Евклидово расстояние, или метрика L2, применяется для вычисления расстояний следующее правило по формуле:

$$d_E(X, Y) = \sqrt{\sum(X_i - y_i)^2} \quad (2.1)$$

где X = (x1, x2, …, xm), Y = (y1, y2, ..., ym) — векторы значений признаков двух записей.

5) старый центр кластера смещается в его центроид.

$$\text{Центроид} = \left(1/n \sum(x_i)\,; 1/n \sum(y_i)/n\right) \quad (2.2)$$

Таким образом, центроиды становятся новыми центрами кластеров для следующего итерации алгоритма. Шаги 3 и 4 повторяются до тех пор, пока выполнение алгоритма не будет прервано или пока не будет выполнено условие в соответствии с некоторым критерием сходимости.

2.3 Ручной расчёт

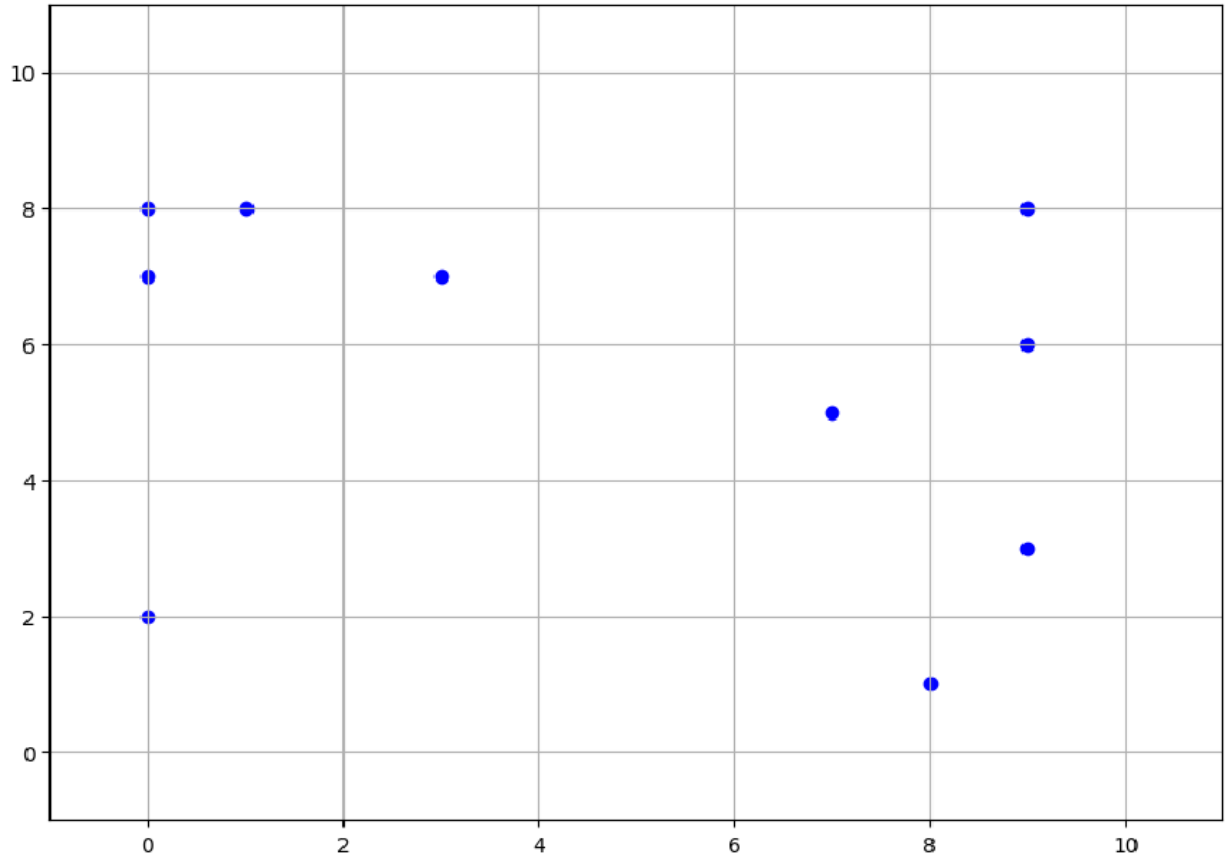

Рисунок 2.1 – Изначальные данные

Шаг 1. Определим число кластеров, на которое требуется разбить исходное множество: k = 2.

Шаг 2. Случайным образом выберем две точки, которые будут начальными центрами кластеров. Пусть это будут точки m1 = (6,32; 1,47) и m2 = (0,0; 7,0). На рис. 5.1 они представлены ромбами.

Шаг 3, для каждой точки определим ближайший к ней центр кластера с помощью евклидова расстояния.

$$d_E(X,Y) = \sqrt{\sum(X_i - y_i)^2} \quad (2.1)$$

расстояния между центрами кластеров m_1 = (1;1) и m_2 = (2:1) и каждой точкой исходного множества и указано, к какому кластеру принадлежит та или иная точка (таблица 2.1).

Таблица 2.1 – Расстояние между точками

Точка	Расстояние от $\boldsymbol{m_1}$	Расстояние от m_2	Принадлежит кластеру
1	5,26	9,06	2
2	8,4	0,0	1
3	3,59	7,28	2
4	7,06	9,06	2
5	8,42	1,41	1
6	3,09	9,85	2
7	9,09	1,0	1
8	1,74	10,0	2
9	6,34	5,0	1
10	6,45	3,0	1

Таким образом, кластер 1 содержит точки *1, 3, 4, 6, 8* а кластер 2 — точки 2, 5, 7, 9, 10.

Шаг 4. Для каждого кластера вычисляется центроид, и в него перемещается центр кластера.

$$\text{Центроид} = \left(\sum (x_i)/n\,; \sum (y_i)/n\right) \tag{2.4}$$

Центроид для кластера 1: [(9 + 0 + 7 + 9 + 1 + 9 + 0 + 8 + 3) / 9, (6 + 7 + 5 + 8 + 8 + 3 + 8 + 1 + 7) / 6] = (4,6; 5,5).

Центроид для кластера 2: [(0) / 1, (2) / 1] = (0; 2).

Итоги первой итерации (Рисунок 2.2).

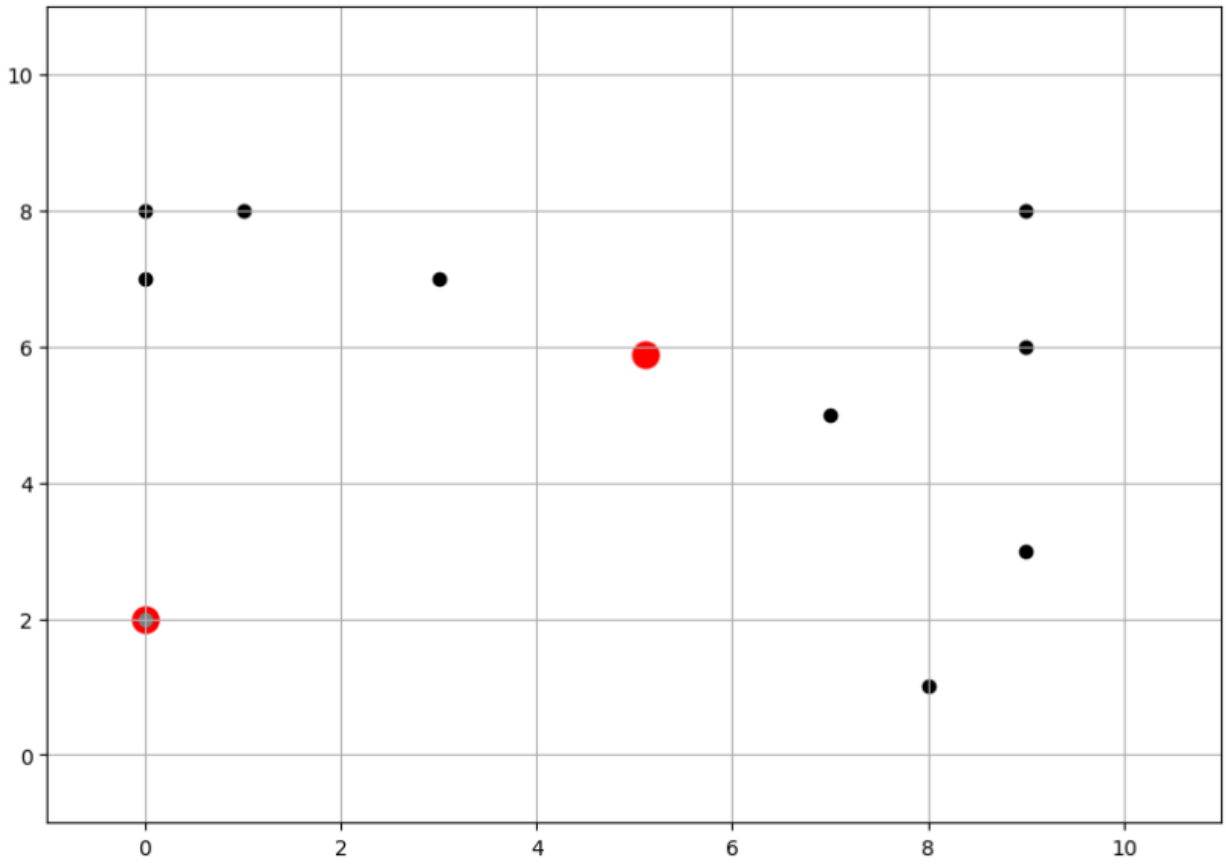

Рисунок 2.2 – Итоги первой итерации

2.4 Программная реализация

Создадим алгоритм k-means в программной реализации на языке высокого уровня Python [2].

Полученный результат работы программы k-means (Рисунок 2.3 – Рисунок 2.4).

```
1 кластер = 33.92
2 кластер = 14.260000000000002
3 кластер = 4.209999999999999
4 кластер = 2.8
5 кластер = 2.73
```

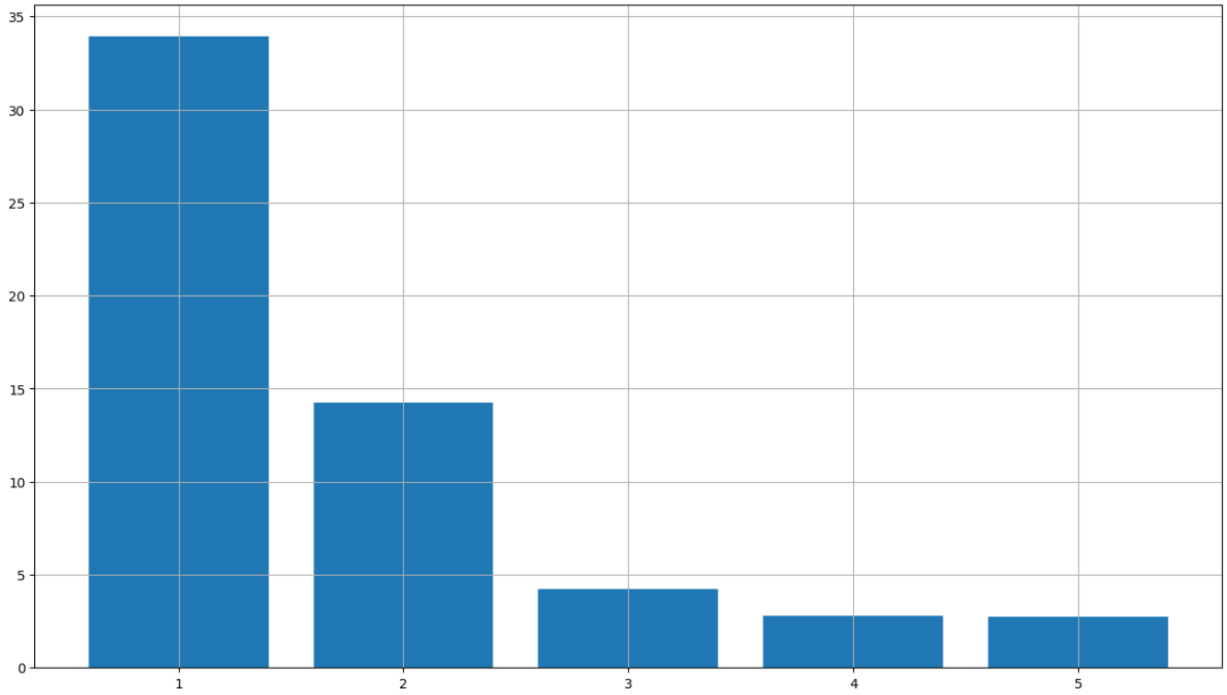

Рисунок 2.3 – Правило локтя для данных

Исходя из правила локтя количество кластеров должно быть 3.

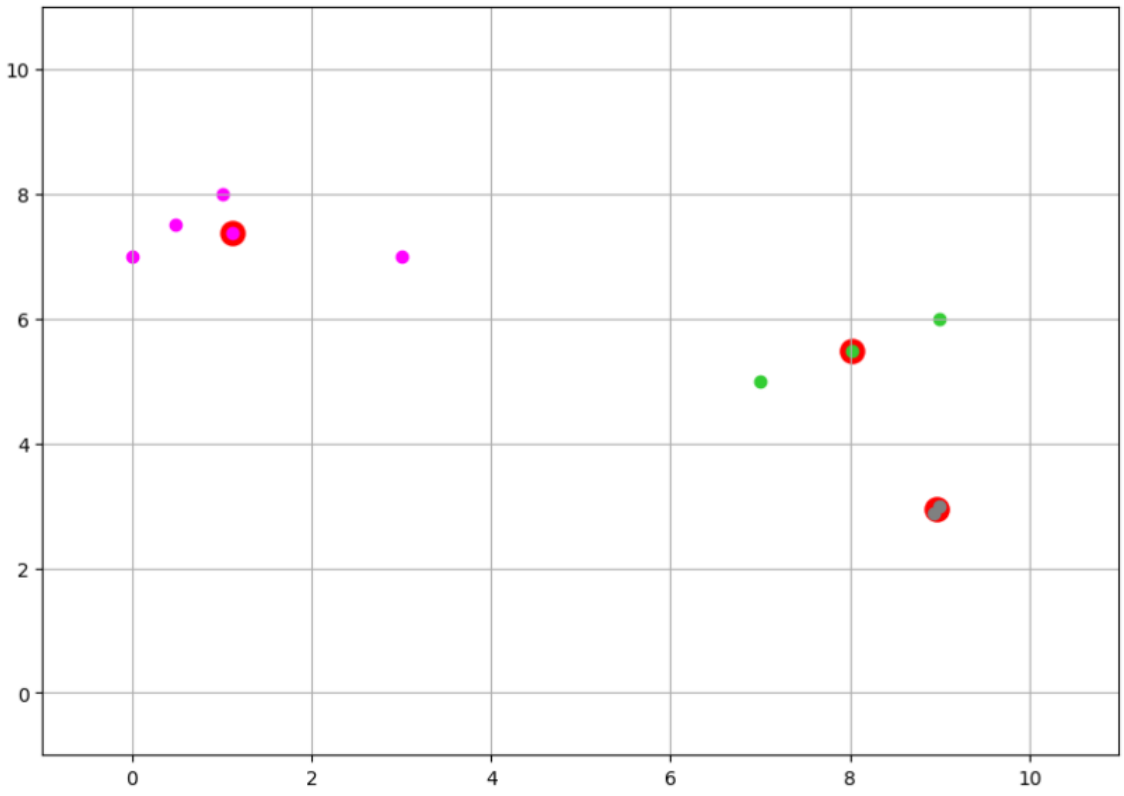

Рисунок 2.4 – Результат работы алгоритма k-means

3 ЛИНЕЙНАЯ РЕГРЕСИЯ

3.1 Постановка задачи

Придумать свою предметную область и реализовать алгоритм линейной регрессии для произвольных данных при помощи евклидовое расстояние.

3.2 Описание предметной области

Результатом собранных наблюдений явилась зависимость ежемесячных продаж картофеля от установленной цены (Таблица. 3.1).

Таблица 3.1. Зависимость объема продаж от цены

№ месяца	Цена за 1 кг, x	Количество проданного картофеля, y, кг	Удалось продать картофель
1	13	1000	0
2	20	600	1
3	17	500	0
4	15	1200	1
5	16	1000	1
6	12	1500	1
7	16	500	0
8	14	1200	1
9	10	1700	0
10	11	2000	1

3.3 Описание алгоритма

Линия регрессия – это прямая наилучшего приближения для множества пар значений входной и выходной переменной (x, y), выбираемая таким образом, чтобы сумма квадратов расстояний от точек (x_i, y_i) до этой прямой, измеренных вертикально (то есть вдоль оси y), была минимальна, Уравнение, описывающее линию регрессии, называется уравнением регрессии :

$$\hat{y} = b_0 + b_1 x + e \quad (3.1)$$

где $\hat{y}$– оценка значения выходной переменной;

b_0 - коэффициент, определяющий точку пересечения линии с осью y, называемый также свободным членом Коэффициент b_1 определяет наклон линии относительно оси x (иногда ero называют *угловым коэффициентом*).

b_1 – это величина, на которую изменяется значение выходной переменной y при изменении входной переменной x на единицу.

е-ошибка.

Тогда сумму квадратов ошибок по всем наблюдениям можно вычислить следующим образом:

$$E = \sum_{i=1}^{n} \varepsilon^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - b_0 - b_i x_i)^2 \quad (3.2)$$

Можно найти значения b_0 и b_1, которые минимизируют $\Sigma_{i=1}^{n} \varepsilon^2$, путем дифференцирования уравнения (3.1) по b_0 и b_1. Частные производные для уравнения (3.2) по $b0$ и $b1$ соответственно будут:

$$\frac{\partial E}{\partial b_0} = -2 \sum_{i=1}^{n} (\hat{y}_i - b_0 - b_i x_i)^2 ; \frac{\partial E}{\partial b_1} = -2 \sum_{i=1}^{n} (\hat{y}_i - b_0 - b_i x_i)^2 \quad (3.3)$$

3.4 Ручной расчёт

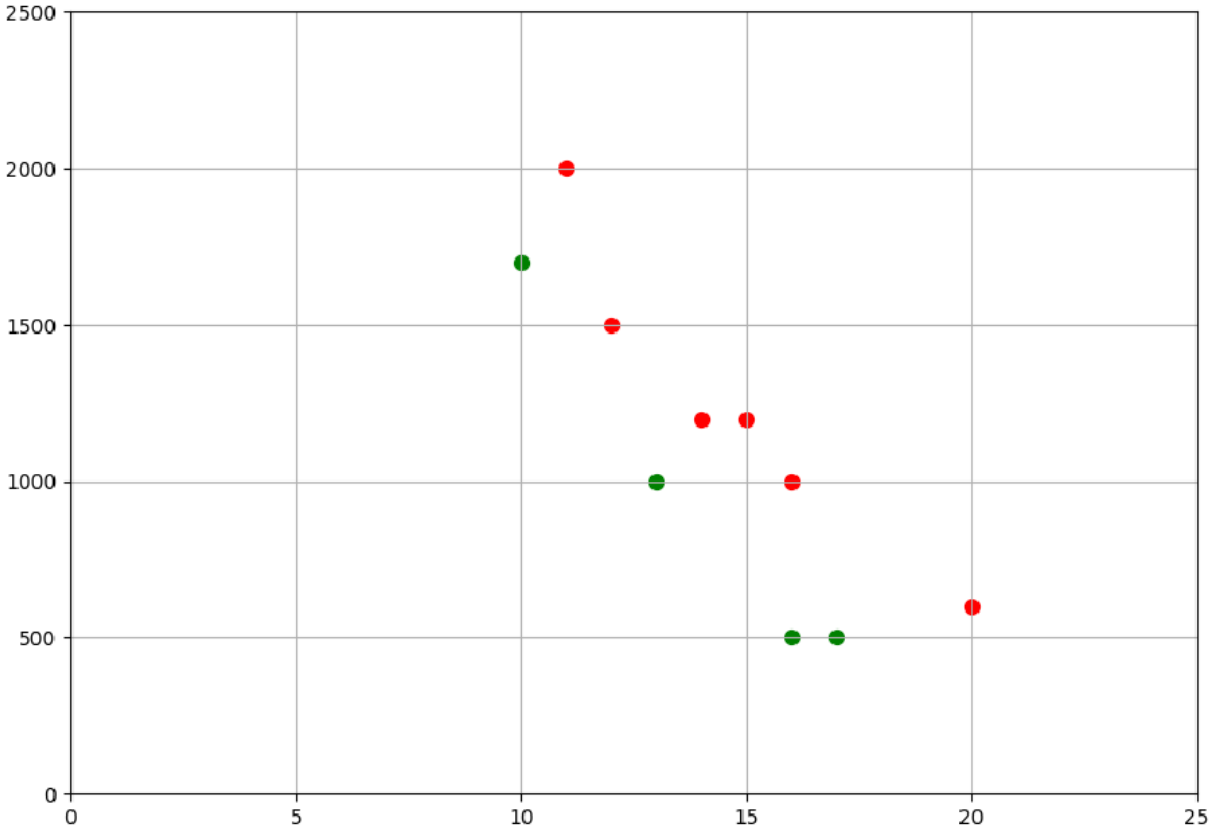

Рисунок 3.1 – Изначальные данные

Система нормальных уравнений.

$$\begin{cases} b0 \cdot n + b1 \cdot \sum x = \sum y \\ b0 \cdot \sum x + b1 \cdot \sum x^2 = \sum (y \cdot x) \end{cases}$$

Для наших данных система уравнений имеет вид

$$10\, b_0 + 144 \cdot b_1 = 11200 \quad (3.4)$$

$$144 \cdot b_0 + 2156 \cdot b_1 = 149300 \quad (3.5)$$

Решим систему уравнений выразим b_0 из формулы (3.4):

$$b_0 = \frac{11200 - 144\, b_1}{10} = 1120 - 14{,}4\, b_1 \quad (3.6)$$

(1.5)

Подставим в формулу (3.6) выражение $b_0 = 1120 - 14{,}4\, b_1$

Получим:

$161280 - 2073{,}6\, b_1 + 2156\, b_1 + 149300$

-2073,6 b_1 + 2156b_1 = 149300 – 161280

82,4 b_1 = -11980

$$b_1 = \frac{-11980}{82,4} = -145,38834$$

Подставим коэффициент b_1 в формулу (3.6) чтобы получить коэффициент b_0

$$b_0 = 1120 - 14,4\, b_1 = 1120 - 14,4(-145,388) = 3213,58$$

Получили коэффициенты регрессии:

$$b_0 = 3213,58$$

$$b_1 = -145,388$$

Подставим коэффициенты b_0 и b_1 в уравнение регрессии в формулы (3.1), получим:

$$\hat{y} = 3213,58 - 145,388x$$

Оценку значения $\hat{y}$ получаем из выражения (1.6).

А также коэффициенты b0 и b1 можно найти другим способом. Они минимизируются $\sum_{i=1}^{n} \hat{e}^2$, путем дифференцирования уравнения (3.1) по b_0 и b_1. Можно использовать метод градиентного спуска или метод Ньютона.

Стандартная ошибка равна корню квадратному среднеквадратической ошибки ($E_{СКО}$), то есть сумме квадратов разностей между реальным и оцененным значениями, вычисленной по всем наблюдениям и отнесенной к числу степеней свободы выборки [1]:

$$E_{СКО} = \frac{\sum_{i=1}^{n}(y_i - \hat{y_i})^2}{n - m - 1} \quad (3.6)$$

где m – количество независимых переменных, которое для простой линейной регрессии равно 1.

$E_{СКО}$ можно рассматривать как меру изменчивости выходной переменной, объясняемую регрессией.

В то время стандартная ошибка оценивания ориентируются следующим способом [1]:

$$E_{\text{ст}} = \sqrt{E_{СКО}} \quad (3.7)$$

Найдем из формулы (3.7) значения:

$\sum_{i=1}^{n}(\widehat{y_i} - y_i)^2 = (1323 - 1000)^2 + (305{,}6 - 600)^2 + (741{,}8 - 500)^2 + (1032{,}6 - 1200)^2 + (887{,}2 - 1000)^2 + (1468{,}8 - 1500)^2 + (887{,}2 - 500)^2 + (1178 - 1200)^2 + (1759{,}6 - 1700)^2 + (1614{,}2 - 2000)^2 = 593\,989{,}28$

$E_{СКО} = \frac{593\,989{,}28}{10-1-1} = 74248{,}66$

Найдем стандартную ошибку из формулы (3.7)

$E_{\text{ст}} = \sqrt{E_{СКО}} = \sqrt{74248{,}66} = 272{,}4860730$

Простая линейная регрессионная модель задается следующим образом. Пускай существует подборка сведений, включающая n исследований, в любом из которых значению самостоятельной величины x_i соответствует зависимой величине y_i, сопряженных с помощью линейной связи:

$$y = b_0 + b_1 x + \varepsilon$$

где b_0 и b_1 – параметры модели, определяющие точку пересечения линии регрессии с осью y и наклон линии регрессии соответственно;

ε – член, определяющий ошибку отклонения реального наблюдения от оценки, полученной с помощью данной модели.

Подставим наши значение b0, b1 и ошибку отклонения в формулу (1.9)

Получим:

$y = 3213{,}58 - 145{,}388x\ (\pm 272{,}4860730)$

Найдем коэффициент корреляции

Еще одной мерой, используемой для количественного описания линейной зависимости между двумя числовыми переменными, является коэффициент корреляции, который определяется следующим образом:

$$r = \frac{\sum(x-\bar{x})(y-\bar{y})}{(n-1)\sigma_x\sigma_y} \qquad (3.9)$$

где σ_x и σ_y стандартные отклонения соответствующих переменных. Значение коэффициента корреляции всегда расположено в диапазоне от -1 до 1

Вычисление коэффициента корреляции по формуле (3.9)

$$r = \frac{\sum xy - (\sum x \sum y)/n}{\sqrt{\sum x^2 - (\sum x)^2/n} \times \sqrt{\sum y^2 - (y)^2/n}}$$

$$= \frac{908 - (50 \times 60)/10}{\sqrt{304 - 50^2/10} \times \sqrt{2788 - 160^2/10}} = 0{,}9733$$

Если коэффициент корреляции близок к 1, то между переменными имеет место сильная положительная корреляция. Иными словами, наблюдается высокая степень зависимости входной и выходной переменных (если значения входной переменной x возрастают, то и значения выходной переменной y также будут увеличиваться).

3.5 Программная реализация

Создадим алгоритм линейной регрессии в программной реализации на языке высокого уровня Python [2].

Полученный результат работы программы линейной регрессии (Рисунок 3.2 – Рисунок 3.4).

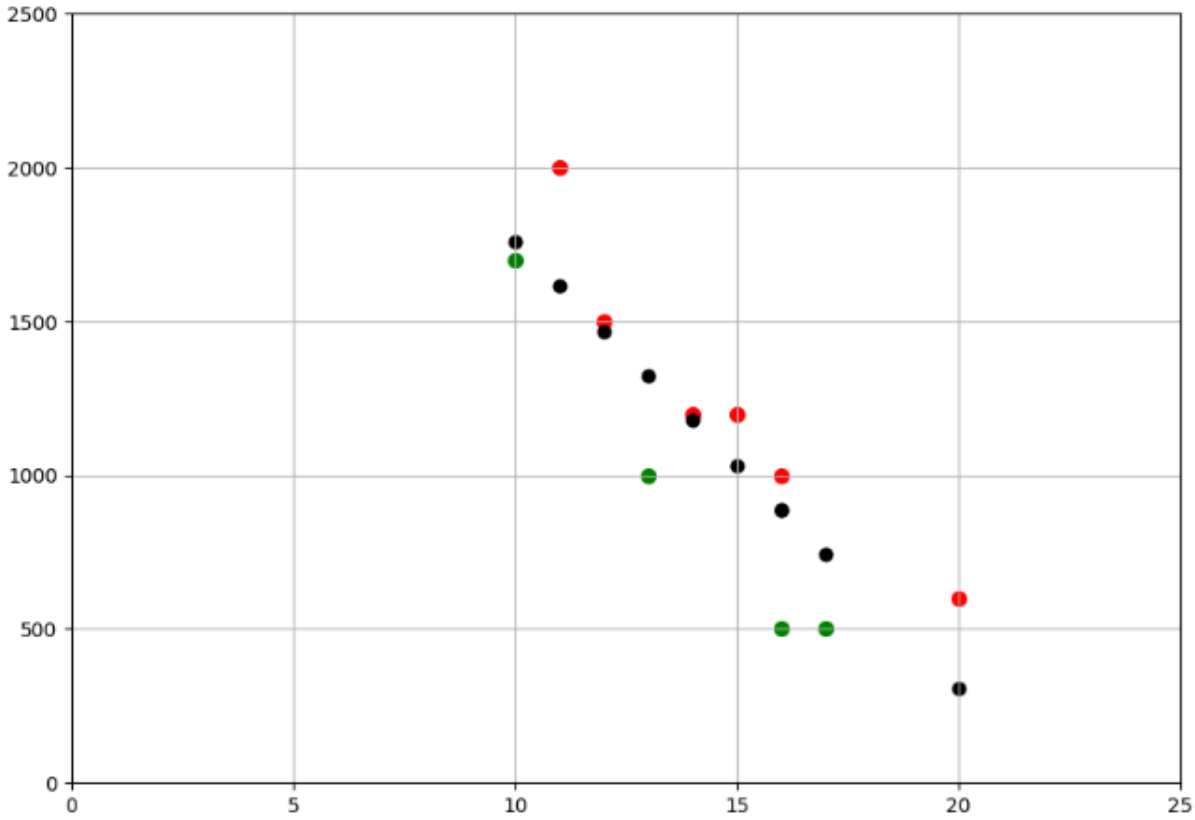

Рисунок 3.2 – График результат работы алгоритма линейной регрессии

```
n = 10
summ x = 144
summ x**2 = 2156
summ y = 11200
summ xy = 149300
b0 * 10 + b1 * 144 = 11200
144 · b0 + 2156 · b1 = 149300 |==| 149300
b0 = 11200/10 - 144·b1 = 1120.0 - 14.4*b1
144 * (1120.0 - 14.4·b1) + 20736·b1 = 145600
-11980.0 / 82.40000000000009
-145.388
3213.587 = 1120.0 - 14.4 * b1
b0 = 3213.587
b1 = -145.388
y^ = 3213.587 + -145.388 * x
1323.543 = 3213.587 + -145.388 * x
```

Рисунок 3.3 – Текстовый результат работы алгоритма линейной регрессии

```
для x = 13 и y = 1000
b0 = -145.388
b1 = 3213.587
dy = 1323.543

для x = 20 и y = 600
b0 = -145.388
b1 = 3213.587
dy = 305.827

для x = 17 и y = 500
b0 = -145.388
b1 = 3213.587
dy = 741.991

для x = 15 и y = 1200
b0 = -145.388
b1 = 3213.587
dy = 1032.767

для x = 16 и y = 1000
b0 = -145.388
b1 = 3213.587
dy = 887.379

для x = 12 и y = 1500
b0 = -145.388
b1 = 3213.587
dy = 1468.931

для x = 16 и y = 500
b0 = -145.388
b1 = 3213.587
dy = 887.379

для x = 14 и y = 1200
b0 = -145.388
b1 = 3213.587
dy = 1178.155
```

Рисунок 3.4 – Текстовый результат работы алгоритма линейной регрессии

```
для x = 11 и y = 2000
b0 = -145.388
b1 = 3213.587
dy = 1614.319

Q = 11199.998
summ_for_Ecko = 594247.574
Ecko = 74280.947
Ect =  272.545
r = 0.8634889672536017
```

Рисунок 3.5 – Текстовый результат работы алгоритма линейной регрессии

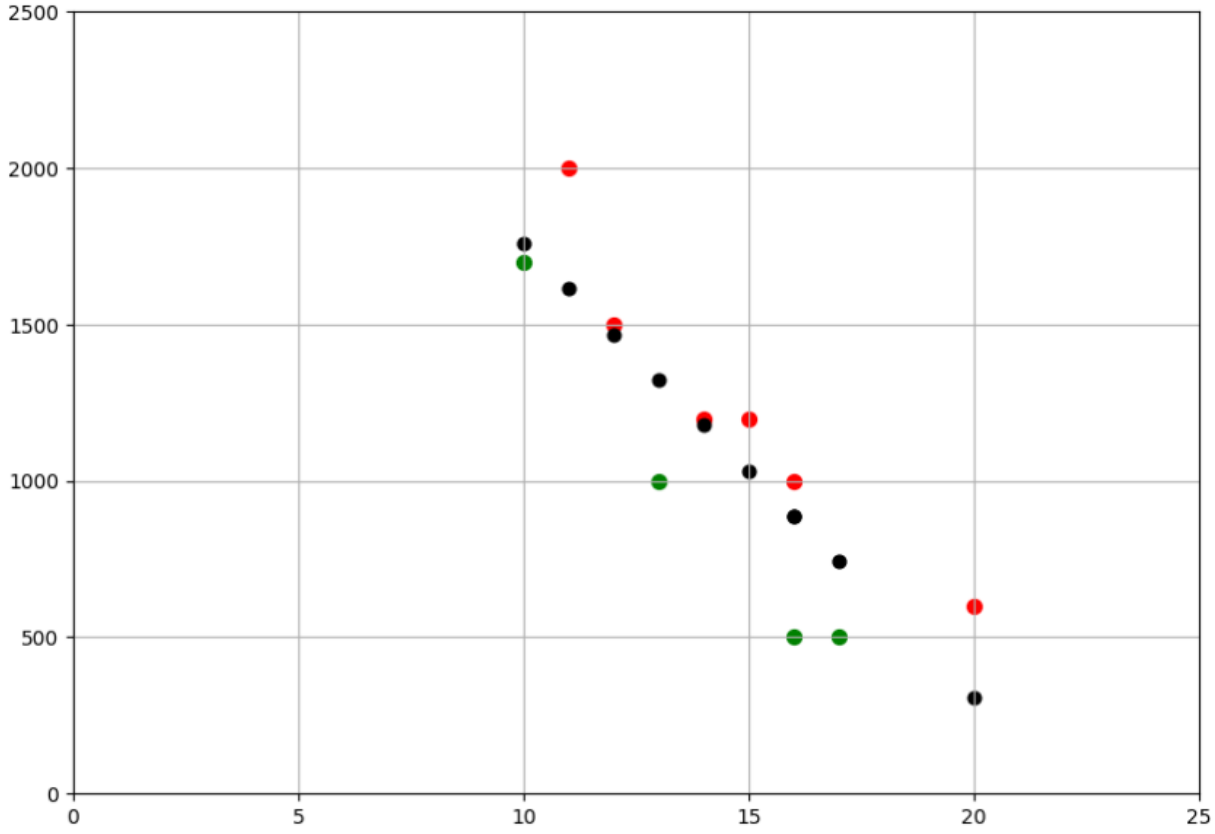

Рисунок 3.6 – Граф результата работы алгоритма линейной регрессии

4. БАЕСОВСКИЙ КЛАСИФИКАТОР

4.1 Постановка задачи

Придумать свою предметную область и реализовать байесовской классификации.

4.2 Описание предметной области

Результатом собранных наблюдений явилась зависимость длины и ширины, от того гусеница это или божья коровка (Таблица. 4.1).

Таблица 4.1. Зависимость объема длины и ширины от класса

Номер	Ширина	Длина	Класс
1	10	50	гусеница
2	20	30	божья коровка
3	25	30	божья коровка
4	20	60	гусеница
5	15	70	гусеница
6	40	40	божья коровка
7	30	45	божья коровка
8	20	45	гусеница
9	40	30	божья коровка
10	7	35	гусеница

4.3 Описание алгоритма

Пусть имеется объект или наблюдение X, класс которого неизвестен. Пусть также имеется гипотеза H, согласно которой X относится к некоторому классу C. Для за дачи классификации можно определить вероятность $P(H|X)$, то есть вероятность того, что гипотеза H для X справедлива. $P(H|X)$ называется условной вероятностью того, что гипотеза H верна при условии, что классифицируется объект X, или апостериорной вероятностью.

(| - множество элементов, удовлетворяющих условию, множество всех… таких, что верно…). Апостериорное распределение – условное распределение вероятностей какой-либо случайной величины при некотором условии,

рассматриваемое в противоположность ее безусловному или априорному распределению)

Предположим, что объектами классификации являются фрукты, которые описываются их цветом и размером, Определим объект X как красный и круглый и выдвинем гипотезу H, что это яблоко. Тогда условная вероятность $P(H|X)$ отражает меру уверенности в том, что объект X является яблоком при условии, что он красный и круглый, Кроме условной (апостериорной) вероятности, рассмотрим так называемую априорную вероятность $P(H)$. В нашем примере это вероятность того, что любой наблюдаемый объект является яблоком, безотносительно к тому, как он выглядит. Таким образом, апостериорная вероятность основана на большей информации, чем априорная, не предполагающая зависимость от свойств объекта X.

Аналогично $P(H|X)$ апостериорная вероятность X при условии H, или вероятность того, что X является красным и круглым, если известно, что это яблоко, $P(X)$априорная вероятность X. В нашем примере это просто вероятность того, что объект является красным и круглым. Вероятности $P(X)$, $P(H)$ и $P(X|H)$ могут быть оценены на основе наблюдаемых данных.

Для вычисления апостериорной вероятности на основе $P(X)$, $P(H)$ и $P(X|H)$ используется формула Байеса:

$$P(H|X) = \frac{P(H|X)P(H)}{P(X)} \quad (4.1)$$

Алгоритм работы простого байесовского классификатора содержит следующие шаги.

1. Пусть исходное множество данных S содержит атрибуты A_1, A_2, ..., A_n Тогда каждый объект или наблюдение $X \in S$ будет представлено своим набором значений этих атрибутов $x_1, x_2, \dots, x_n$ где x_i значение, которое принимает атрибут A_i в данном наблюдении.

2. Предположим, что задано m классов $C = \{C_1, C_2, \dots C_m\}$ и наблюдение X, для которого класс неизвестен. Классификатор должен определить, что X относится к классу, который имеет наибольшую апостериорную вероятность

$P(H|X)$. Простой байесовский классификатор относит наблюдение X к классу $C_x(K = 1, \dots m)$ тогда и только тогда, когда выполняется условие $P(C_k|X) > P(C_k|X) > P(C_k|X)$ для любых $1 \leq j \leq m$:.т.к, $k \neq j$.

По формуле Байеса:

$$P(C_k|X) = \frac{P(X|C_k)P(C_k)}{P(X)} \quad (4.2)$$

3. Поскольку вероятность $P(X)$ для всех классов одинакова, максимизировать требуется только числитель формулы (4.2), Если априорная вероятность класса $P(Ck)$ неизвестна, то можно предположить, что классы равновероятны, $P(C_1)$=$P(C_2)$=...=$P(C_m)$, и, следовательно, мы должны выбрать максимальную вероятность $P(C_k|X)$.

Заметим, что априорные вероятности классов могут быть оценены как $P(C_k)$= s_k/s, где s_k – число наблюдений обучающей выборки, которые относятся к классу C_k, а s – общее число обучающих примеров.

4. Если исходное множество данных содержит большое количество атрибутов, то определение $P(X|C_k)$ может потребовать значительных вычислительных затрат. Чтобы их уменьшить, используется «наивное» предположение о независимости признаков. То есть для набора атрибутов X=$(x_1, x_2, \dots, x_n)$ можно записать:

$$P(X|C_k) = P(x_1|C_k) \times P(x_2|C_k) \times \dots \times P(x_n|C_k) \quad (4.3)$$

Вероятности, стоящие в правой части формулы (4.3), могут быть определены из обучающего набора данных для следующих случаев.

Атрибут A является категориальным, тогда $P(X|Ck)$ =s_{jk}/s_k, где s_{jk} – общее число наблюдений класса C_i, в которых A_i принимает значение x_i, а s_k – общее число наблюдении, относящихся к классу C_k.

Атрибут A является непрерывным, тогда предполагается, что его значения подчиняются закону распределения Гаусса:

$$P(x_i|C_k) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(\frac{(x-m)^2}{2\sigma^2}\right) \quad (4.4)$$

где m и σ^2– математическое ожидание и дисперсия значений атрибута Ai для наблюдений, относящихся к классу C_k.

При классификации неизвестного наблюдения объект X будет относиться к классу, для которого $P(X|C_i)\times P\ (C_i)$ принимает наибольшее значение.

4.4 Ручной расчёт

Рассчитаем математическое ожидание и дисперсию для всех возможных исходов из формулы 4.4.

$$P(x_i|C_k) = \frac{1}{\sqrt{2\pi\sigma}}\exp\left(\frac{(x-m)^2}{2\sigma^2}\right) \quad (4.4)$$

Для критерия ширины:

Если класс гусеница, то:

$$m = \frac{1}{5} * (10 + 20 + 15 + 20 + 7) = 14{,}4$$

$$\sigma^2 = \left(\frac{1}{4}\right) * ((10-14{,}4)^2 + (20-14{,}4)^2 + (15-14{,}4)^2 + (20-14{,}4)^2 + (7-14{,}4)^2) = 34{,}4$$

Если класс божья коровка, то:

$$m = \frac{1}{5} * (20 + 25 + 40 + 30 + 40) = 31$$

$$\sigma^2 = \left(\frac{1}{4}\right) * ((20-31)^2 + (25-31)^2 + (40-31)^2 + (30-31)^2 + (40-31)^2) = 80$$

Для критерия длины:

Если класс гусеница, то:

$$m = \frac{1}{5} * (60 + 50 + 70 + 45 + 35) = 52$$

$$\sigma^2 = \left(\frac{1}{4}\right) * ((50-52)^2 + (60-52)^2 + (70-52)^2 + (45-52)^2 + (35-52)^2) = 182{,}5$$

Если класс божья коровка, то:

$$m = \frac{1}{5} * (30 + 30 + 40 + 45 + 30) = 35$$

$$\sigma^2 = \left(\frac{1}{4}\right) * ((30 - 35)^2 + (30 - 35)^2 + (40 - 35)^2 + (45 - 35)^2 + (30 - 35)^2)$$

$$= 50$$

Подставляем всё в формулу 4.4 и получим:

Вероятность быть божьей коровкой = 0,5

Вероятность быть гусеницей = 0,5

4.5 Программная реализация

Создадим алгоритм байесовского классификатора в программной реализации на языке высокого уровня Python [2].

Полученный результат работы программы байесовского классификатора (Рисунок 4.1 – Рисунок 4.3).

```
[1, 0, 0, 1, 1, 0, 0, 1, 0, 1] [1, 0]
expectation for criterion 0, for value 0 = 1/5 * (72) = 14.4
expectation for criterion 0, for value 1 = 1/5 * (155) = 31.0
expectation for criterion 1, for value 0 = 1/5 * (260) = 52.0
expectation for criterion 1, for value 1 = 1/5 * (175) = 35.0
expectation = [[14.4, 31.0], [52.0, 35.0]]
variance for criterion 0, for value 0 = (1/(5-1)) * (137.2) = 34.3
variance for criterion 0, for value 1 = (1/(5-1)) * (320.0) = 80.0
variance for criterion 1, for value 0 = (1/(5-1)) * (730.0) = 182.5
variance for criterion 1, for value 1 = (1/(5-1)) * (200.0) = 50.0

expectation = [[14.4, 31.0], [52.0, 35.0]]
variance = [[34.3, 80.0], [182.5, 50.0]]
[0.5, 0.5]
```

Рисунок 4.1 –результат работы байесовского классификатора

```
input_data = [10, 50] # задаём данные
```

Рисунок 4.2 – Новые данные для обучения классификатора

```
[-7.606423013120 88e-05, -3.866954471232428e-05]
1 - божья коровка
```

Рисунок 4.3 – Результат классификации

5 ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ

5.1 Постановка задачи

Придумать свою предметную область и реализовать логистическую регрессию.

5.2 Описание предметной области

Результатом собранных наблюдений явилась зависимость баллов за первый и второй экзамен, от того поступил ли человек в вуз (Таблица. 5.1), 1 – если поступи, 0 – если не поступил.

Таблица 5.1. Зависимость объема длины и ширины от класса

№	Баллы за первый экзамен	Баллы за второй экзамен	Класс
1	0,46	0,43	0
2	0,62	0,36	0
3	0,53	0,93	1
4	0,73	0,12	0
5	0,5	0,91	1
6	0,82	0,08	0
7	0,03	1,0	1
8	0,82	0,21	1
9	0,4	0,18	0
10	0,25	0,01	0

5.3 Описание алгоритма

В основе логистической регрессии лежит “сигмоида”. Это монотонная возрастающая нелинейная функция, имеющая форму буквы “S” (формула 1).

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5.1}$$

Логистическая функция определена на бесконечности и изменяется в диапазоне от 0 до 1.

В логистической регрессии используется преобразование вида (формула 2):

$$g(x) = ln\frac{\rho(x)}{1-\rho(x)} = \beta_0 + \beta_1 x \quad (5.2)$$

Оно называется логит-преобразованием и обладает такими полезными свойствами, как линейность, непрерывность и определенность на бесконечности.

Задача обучения логистической регрессии заключается в подборе коэффициентов (весов) для максимизации правдоподобия.

Критерий правдоподобия описывается формулой 3:

$$loss = \sum_i (-y_i \log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i)) \to min \quad (5.3)$$

Для поиска минимума данного выражения применяется метод градиентного спуска.

Для этого на каждой итерации подсчитывается значение градиента (формула 4), который вычитается из значений весов.

$$grad = \frac{\partial loss}{\partial w} = (\hat{y} - y)x \quad (5.4)$$

5.4 Ручной расчёт

Рандомно сгенерируем веса: $a = 0{,}65;\ b_1 = 0.39;\ b_2 = 1{,}15$

Узнаем, как наша программа классифицирует наши данные с с помощью формул (5.12 и 5.13):

$$logit(x) = \langle x, w \rangle \quad (5.5)$$

$$\sigma(x) = \frac{1}{1 + e^{-loggit(x)}} \quad (5.6)$$

Для первого значения

$$logit(x) = \langle x, w \rangle = 0{,}65 + 0{,}46 * 0{,}39 + 0{,}43 * 1.15 = 1{,}33$$

$$\sigma(x) = \frac{1}{1 + e^{-1{,}33}} = 0{,}79 > 0.5$$

Для второго значения

$$logit(x) = \langle x, w \rangle = 0{,}65 + 0{,}62 * 0{,}39 + 0{,}36 * 1.15 = 1{,}33$$

$$\sigma(x) = \frac{1}{1 + e^{-1{,}33}} = 0{,}78 > 0.5$$

Получим таблицу 5.2

Таблица 5.2 – Полученные данные

№	Logit(x)	$\sigma(x)$
1	1.33	0.79
2	1.31	0.79
3	1.94	0.87
4	1.08	0.75
5	1.9	0.87
6	1.07	0.74
7	1.82	0.86
8	1.22	0.77
9	1.02	0.73
10	0.76	0.68

При помощи критерия правдоподобия формула (5.4):

$$logglos = \sum(0 * log(0{,}79) + (1 - 0) + log(1 - 0{,}79)) + (1 * log(0{,}87) + (1 - 1) + log(1 - 0{,}87)) + (0 * log(0{,}75) + (1 - 0) + log(1 - 0{,}75)) + (1 * log(0{,}87) + (1 - 1) + log(1 - 0{,}87)) + (0 * log(0{,}74) + (1 - 0) + log(1 - 0{,}74)) + (1 * log(0{,}86) + (1 - 1) + log(1 - 0{,}86)) + (1 * log(0{,}77) + (1 - 1) + log(1 - 0{,}77)) + (0 * log(0{,}73) + (1 - 0) + log(1 - 0{,}73)) + (0 * log(0{,}68) + (1 - 0) + log(1 - 0{,}68)) = 0{,}90$$

При помощи градиентного спуска по формуле (5.4):

$$grad = \frac{\partial loss}{\partial w} = (\hat{y} - y)x$$

Для $\omega_1 = -(-(0 - 0{,}79)) = -0{,}14$

Для $\omega_2 = -(-(0 - 0{,}79)) = 0{,}03$

Для $\omega_3 = -(-(0 - 0{,}79)) = 0{,}81$

На последней итерации мы получим веса:

$$\omega_1 = -6{,}05, \qquad \omega_2 = 5{,}10, \qquad \omega_3 = 8{,}59$$

5.5 Программная реализация

Создадим алгоритм логистической регрессии в программной реализации на языке высокого уровня Python [2].

Полученный результат работы программы линейной регрессии (Рисунок 5.1 – Рисунок 5.8).

```
[[0.46, 0.43], [0.62, 0.36], [0.53, 0.93], [0.73, 0.12], [0.5, 0.91],
[0, 0, 1, 0, 1, 0, 1, 1, 0, 0]
Изначальные веса: [0.653569605, 0.3950881494525928, 1.153569605]

Итерация 0 |logloss = 0.901186298688469
Результат: [-0.767813031, -0.38617154, 1.77059503] |

Итерация 1 |logloss = 0.4871380551193548
Результат: [-1.024233904, -0.40564432, 2.621207293] |

Итерация 2 |logloss = 0.4365003197946387
Результат: [-1.247235158, -0.346887026, 3.26626717] |

Итерация 3 |logloss = 0.41010157802592906
Результат: [-1.454219652, -0.249257504, 3.770103612] |

Итерация 4 |logloss = 0.3939797623146416
Результат: [-1.647142564, -0.128597116, 4.176065178] |

Итерация 5 |logloss = 0.38288936549236247
Результат: [-1.827619394, 0.006427534, 4.511565537] |

Итерация 6 |logloss = 0.3745847593871715
Результат: [-1.997265452, 0.150565157, 4.794622249] |

Итерация 7 |logloss = 0.36797699118391486
Результат: [-2.15755904, 0.300411722, 5.037572471] |

Итерация 8 |logloss = 0.3624817987267107
Результат: [-2.309774128, 0.453649315, 5.249178254] |

Итерация 9 |logloss = 0.3577604870004619
Результат: [-2.454975963, 0.60863936, 5.435859475] |

Итерация 10 |logloss = 0.3536046367632946
Результат: [-2.594043771, 0.764189519, 5.602444699] |
```

Рисунок 5.1 – консольный вывод результат работы логистической регрессии

```
Итерация 11 |logloss = 0.3498796916291602
Результат: [-2.727700638, 0.919412784, 5.75264603] |

Итерация 12 |logloss = 0.3464951807720808
Результат: [-2.856542193, 1.073638545, 5.889369884] |

Итерация 13 |logloss = 0.34338803352612346
Результат: [-2.981061319, 1.22635438, 6.014926652] |

Итерация 14 |logloss = 0.3405127772604197
Результат: [-3.10166846, 1.377166726, 6.13117607] |

Итерация 15 |logloss = 0.3378355577503067
Результат: [-3.218707941, 1.525773555, 6.239630483] |

Итерация 16 |logloss = 0.3353303795587047
Результат: [-3.332470948, 1.671944854, 6.341529852] |

Итерация 17 |logloss = 0.332976684302005
Результат: [-3.443205821, 1.81550834, 6.437897372] |

Итерация 18 |logloss = 0.3307577602194808
Результат: [-3.551126206, 1.956338725, 6.529581484] |

Итерация 19 |logloss = 0.32865968140149787
Результат: [-3.656417521, 2.094349456, 6.617288196] |

Итерация 20 |logloss = 0.3266705913515624
Результат: [-3.759242101, 2.229486184, 6.701606362] |

Итерация 21 |logloss = 0.32478021389477024
Результат: [-3.859743293, 2.361721498, 6.78302775] |

Итерация 22 |logloss = 0.32297951585042567
Результат: [-3.958048737, 2.491050566, 6.861963215] |

Итерация 23 |logloss = 0.32126047168752125
Результат: [-4.05427299, 2.617487468, 6.938755889] |
```

Рисунок 5.2 – консольный вывод результат работы логистической регрессии

```
Итерация 24 |logloss = 0.31961589687849445
Результат: [-4.148519634, 2.741062056, 7.013692059] |

Итерация 25 |logloss = 0.31803932746465424
Результат: [-4.240882986, 2.861817239, 7.087010225] |

Итерация 26 |logloss = 0.3165249305623666
Результат: [-4.331449468, 2.979806606, 7.1589087] |

Итерация 27 |logloss = 0.3150674354443357
Результат: [-4.42029873, 3.09509234, 7.229552012] |

Итерация 28 |logloss = 0.3136620782072393
Результат: [-4.507504557, 3.207743394, 7.299076342] |

Итерация 29 |logloss = 0.3123045553781789
Результат: [-4.593135618, 3.317833878, 7.367594125] |

Итерация 30 |logloss = 0.3109909834365741
Результат: [-4.67725608, 3.425441654, 7.435197976] |

Итерация 31 |logloss = 0.3097178623472387
Результат: [-4.75992611, 3.530647117, 7.501964014] |

Итерация 32 |logloss = 0.30848204195994705
Результат: [-4.841202305, 3.633532142, 7.567954694] |

Итерация 33 |logloss = 0.307280690633374
Результат: [-4.921138039, 3.73417918, 7.633221202] |

Итерация 34 |logloss = 0.3061112657610539
Результат: [-4.999783767, 3.832670499, 7.697805475] |

Итерация 35 |logloss = 0.3049714860683876
Результат: [-5.077187273, 3.929087555, 7.76174191] |

Итерация 36 |logloss = 0.30385930565302177
Результат: [-5.153393893, 4.023510464, 7.825058786] |
```

Рисунок 5.3 – консольный вывод результат работы логистической регрессии

```
Итерация 36 |logloss = 0.30385930565302177
Результат: [-5.153393893, 4.023510464, 7.825058786] |

Итерация 37 |logloss = 0.3027728897858905
Результат: [-5.228446699, 4.11601759, 7.887779459] |

Итерация 38 |logloss = 0.3017105924985315
Результат: [-5.302386667, 4.206685214, 7.949923351] |

Итерация 39 |logloss = 0.30067093596953726
Результат: [-5.375252815, 4.295587283, 8.011506764] |

Итерация 40 |logloss = 0.2996525916999103
Результат: [-5.447082337, 4.382795228, 8.072543557] |

Итерация 41 |logloss = 0.29865436344085267
Результат: [-5.517910714, 4.468377839, 8.133045685] |

Итерация 42 |logloss = 0.29767517181256664
Результат: [-5.587771815, 4.552401191, 8.193023647] |

Итерация 43 |logloss = 0.2967140405315086
Результат: [-5.656697995, 4.634928611, 8.25248684] |

Итерация 44 |logloss = 0.2957700841473454
Результат: [-5.724720177, 4.716020674, 8.31144384] |

Итерация 45 |logloss = 0.29484249717986843
Результат: [-5.791867929, 4.795735235, 8.369902628] |

Итерация 46 |logloss = 0.29393054453997736
Результат: [-5.858169538, 4.874127476, 8.427870762] |

Итерация 47 |logloss = 0.2930335531169482
Результат: [-5.923652077, 4.951249972, 8.485355509] |

Итерация 48 |logloss = 0.2921509044157381
Результат: [-5.988341468, 5.027152775, 8.542363946] |
```

Рисунок 5.4 – консольный вывод результат работы логистической регрессии

```
Итерация 48 |logloss = 0.2921509044157381
Результат: [-5.988341468, 5.027152775, 8.542363946] |

Итерация 49 |logloss = 0.2912820281322957
Результат: [-6.052262537, 5.101883496, 8.598903036] |

Финальная таблица
№0 data = [0.25, 0.01]| y = 0| y_p = 0.009507421824670506
№1 data = [0.4, 0.18]| y = 0| y_p = 0.08018361710425896
№2 data = [0.73, 0.12]| y = 0| y_p = 0.2152717545824331
№3 data = [0.46, 0.43]| y = 0| y_p = 0.49934132665547104
№4 data = [0.82, 0.08]| y = 0| y_p = 0.23456939665723467
№5 data = [0.62, 0.36]| y = 0| y_p = 0.5507605929212139
№6 data = [0.03, 1.0]| y = 1| y_p = 0.9373114652013924
№7 data = [0.82, 0.21]| y = 1| y_p = 0.4819628893009891
№8 data = [0.5, 0.91]| y = 1| y_p = 0.9865970399936989
№9 data = [0.53, 0.93]| y = 1| y_p = 0.9902476614963216

predict: x = 100.0
1-ый элемент - 0.75
2-ой элемент - 0.25
Вероятность отсенения к 1 классу = 0.4809781459534195
Вероятность отсенения к 0 классу = 0.5190218540465805

predict: x = 198.0
1-ый элемент - 0.99
2-ой элемент - 0.99
Вероятность отсенения к 1 классу = 0.9994535472601087
Вероятность отсенения к 0 классу = 0.00054645273989129

predict: x = 50.0
1-ый элемент - 0.25
2-ой элемент - 0.25
Вероятность отсенения к 1 классу = 0.0674166881714103
Вероятность отсенения к 0 классу = 0.9325833118285897
```

Рисунок 5.5 – консольный вывод результат работы логистической регрессии

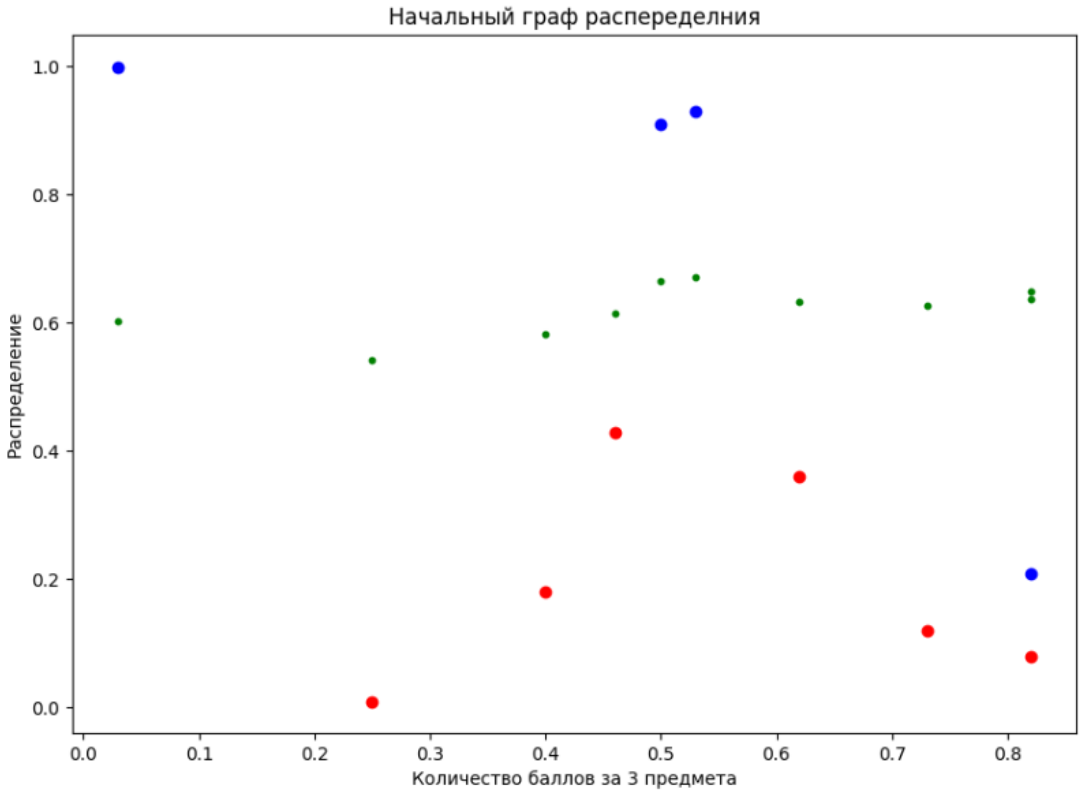

Рисунок 5.6 – Начальный граф распределения

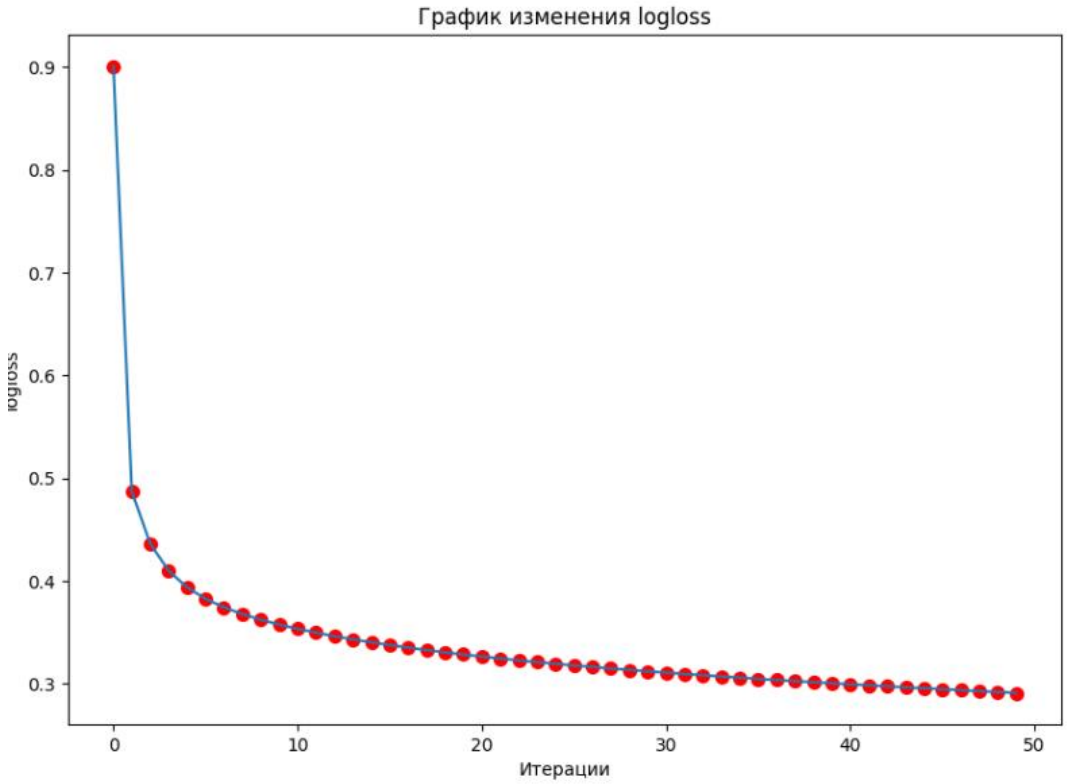

Рисунок 5.7 – График градиентного спуска logloss

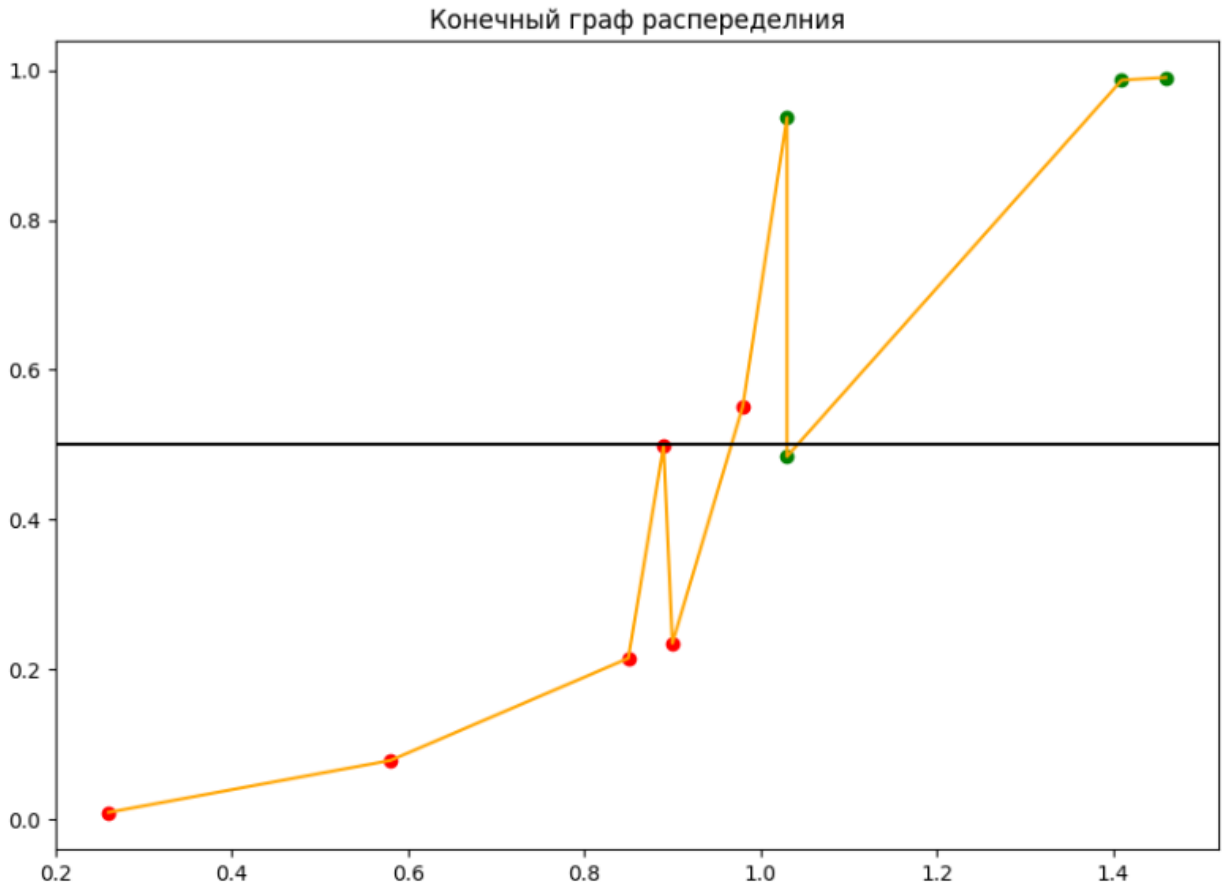

Рисунок 5.8 – Финальный граф

6 PAM

6.1 Постановка задачи

Придумать свою предметную область и реализовать алгоритм PAM.

6.2 Описание предметной области

Рандомно сгенерирован набор данных из (Таблица. 6.1).

Таблица 6.1 – Данные

№	x	y
1	9	6
2	0	7
3	7	5
4	9	8
5	1	8
6	9	3
7	0	8
8	8	1
9	0	2
10	3	7

6.3 Описание алгоритма

Алгоритм PAM очень похож на алгоритм K-means, в основном потому, что оба являются алгоритмами кластеризации, другими словами, оба разделяют множество объектов на группы (кластеры) и работа обоих основана на попытках минимизировать ошибку, но PAM работает с медоидами – объектами, являющимися частью исходного множества и представляющими группу, в которую они включены, а K-means работает с центроидами - искусственно созданными объектами, представляющими кластер.

Введем следующие обозначения для формального описания алгоритма PAM. Пусть $O = \{o_1, o_2, \ldots, o_n\}$ — это множество кластеризуемых объектов, где каждый объект — это кортеж, состоящий из p вещественных чисел. Пусть k

количество кластеров, $k \ll n, C = \{c_1, c_2, \ldots, c_k\}$ множество медоидов, $C \subset O$, и $\rho: O \times C \rightarrow R$ — это метрика расстояния.

На каждой итерации выбирается пара медоид c_i и не-медоид o_j такая, что замена медоида на не-медоид дает лучшую кластеризацию из возможных. Оценка кластеризации выполняется с помощью целевой функции, вычисляемой как сумма расстояний от каждого объекта до ближайшего медоида.

Таким образом, после нахождения набора из k медоидов кластеры строятся путем сопоставления каждого наблюдения с ближайшим медоидом. Затем каждый выбранный медоид n и каждая немедоидная точка данных меняются местами, и вычисляется целевая функция. Целевая функция соответствует сумме отличий всех объектов от их ближайшего медоида.

$$E = \sum_{j=1}^{n} \min_{1 \le i \le k} \rho(c_i, o_j). \tag{6.1}$$

6.4 Ручной расчёт

Возьмём количество кластеров = 1.

Определим начальную медоиду, будем проходить по всем точкам и считать сумму евклидовых расстояний до всех точек формула (6.2).

$$len = \sum \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{6.2}$$

Полученные данные запишем в таблицу 6.2.

Таблица 6.2 – Евклидово расстояние для всех точек

№	len
1	54,79
2	55,65
3	46,49
4	60,63
5	53,02
6	58,91
7	57,92
8	64,93
9	68,31
10	45,89

В качестве медоиды выбираем 10 точку.

Вторую точку выбираем как наиболее удалённую от медоиды. Получим таблицу 6.3 удалённости точек от медоиды, при помощи евклидового расстояния.

$$d_E(X,Y) = \sqrt{\sum(x_i - y_i)^2} \quad (6.3)$$

Таблица 6.3 – Удалённость точек от медоиды

№	Евклидово расстояние
1	6,08
2	3,0
3	4,47
4	6,08
5	2,24
6	7,21
7	3,16
8	7,81
9	5,83

В качестве второй метоиды бедём точку 8.

Определяем какие точки принадлежат 1 и 2 метоиды путём определения евклидового расстояния до каждой из точек, данные представлены в таблице 6.4.

Таблица 6.4 – Определяем какие точки принадлежат 1 и 2 метои

№	len
1	1
2	2
3	1
4	1
5	2
6	1
7	2
8	1
9	2
10	2

Центруем медоиды и снова перераспределяем точки по кластерам получим новую таблицу 6.5.

Таблица 6.5 – Полученные данные

№	len
1	1
2	2
3	1
4	1
5	2
6	1
7	2
8	1
9	2
10	2

6.5 Программная реализация

Создадим алгоритм PAM в программной реализации на языке высокого уровня Python [2].

Полученный результат работы программы PAM (Рисунок 6.1 – Рисунок 6.4).

```
[[3, 6], [0, 4], [6, 8], [4, 4], [3, 1], [2, 7], [8, 3], [0, 7], [4, 9], [5, 3],


[1]
[45.88831077494412]
[1, 2]
[45.88831077494412, 9.414213562373096]
[1, 2, 3]
[45.88831077494412, 9.414213562373096, 6.414213562373095]
[1, 2, 3, 4]
[45.88831077494412, 9.414213562373096, 6.414213562373095, 2.23606797749979]
[1, 2, 3, 4, 5]
[45.88831077494412, 9.414213562373096, 6.414213562373095, 2.23606797749979, 3.0]
Если количество элементов = 1, то
Евклидово расстояние по правилу локтя = 45.88831077494412

Если количество элементов = 2, то
Евклидово расстояние по правилу локтя = 9.414213562373096

Если количество элементов = 3, то
Евклидово расстояние по правилу локтя = 6.414213562373095

Если количество элементов = 4, то
Евклидово расстояние по правилу локтя = 2.23606797749979

Если количество элементов = 5, то
Евклидово расстояние по правилу локтя = 3.0

Введите число кластеров: 2
```

Рисунок 6.1 – консольный вывод результат работы PAM

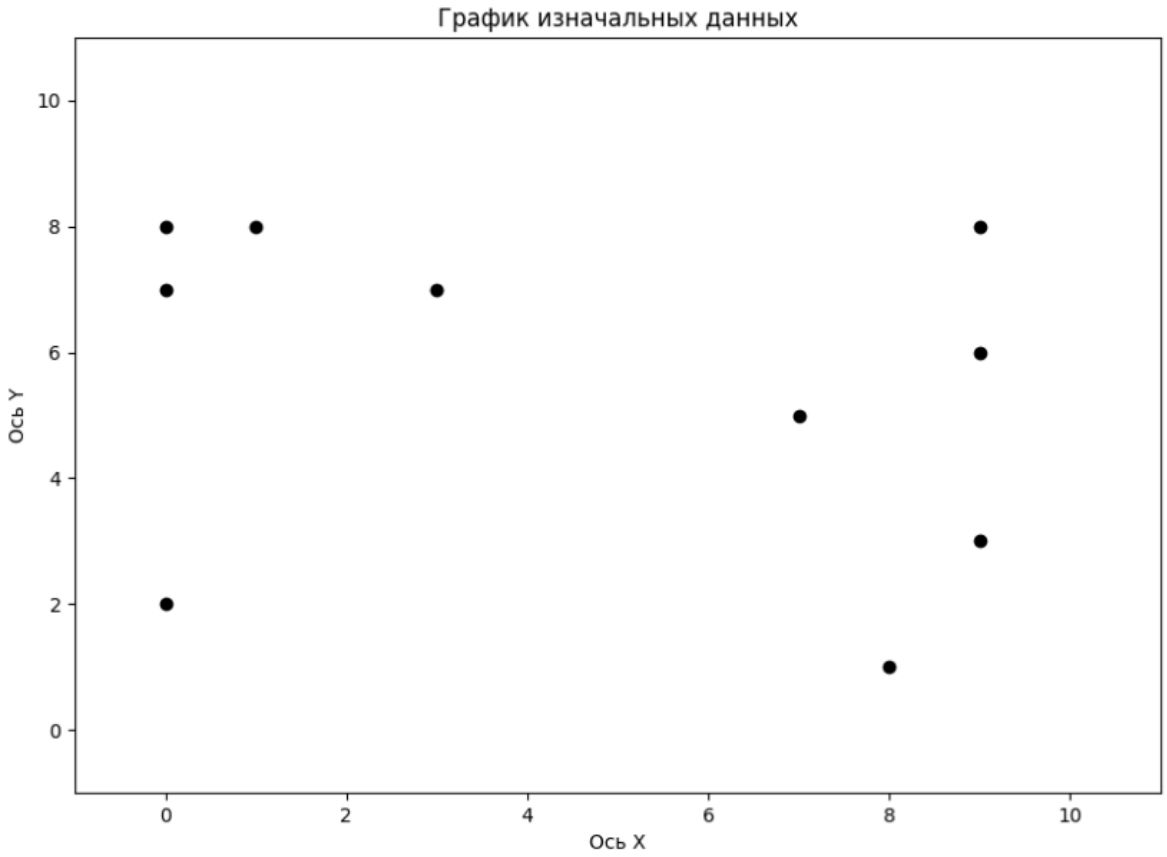

Рисунок 6.2 – Начальные данные

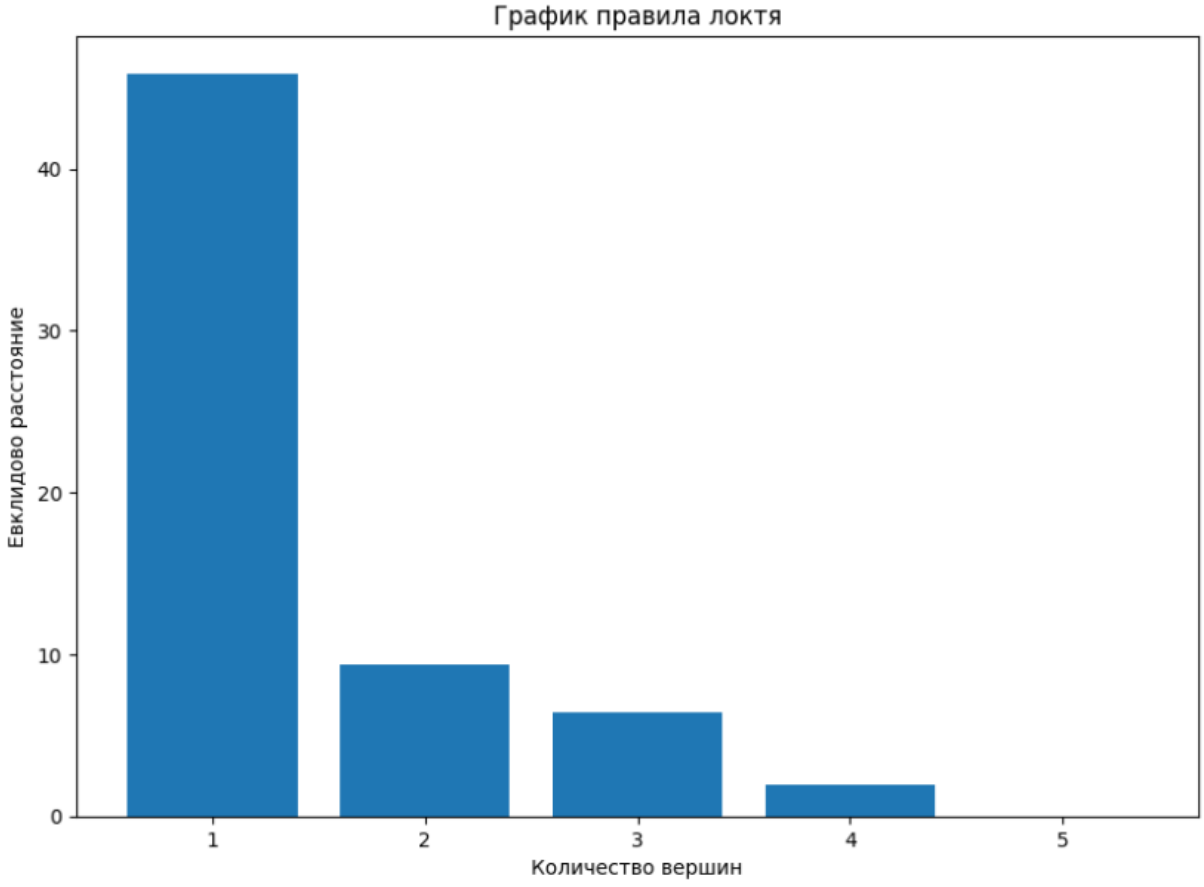

Рисунок 6.3 – Метод локтя

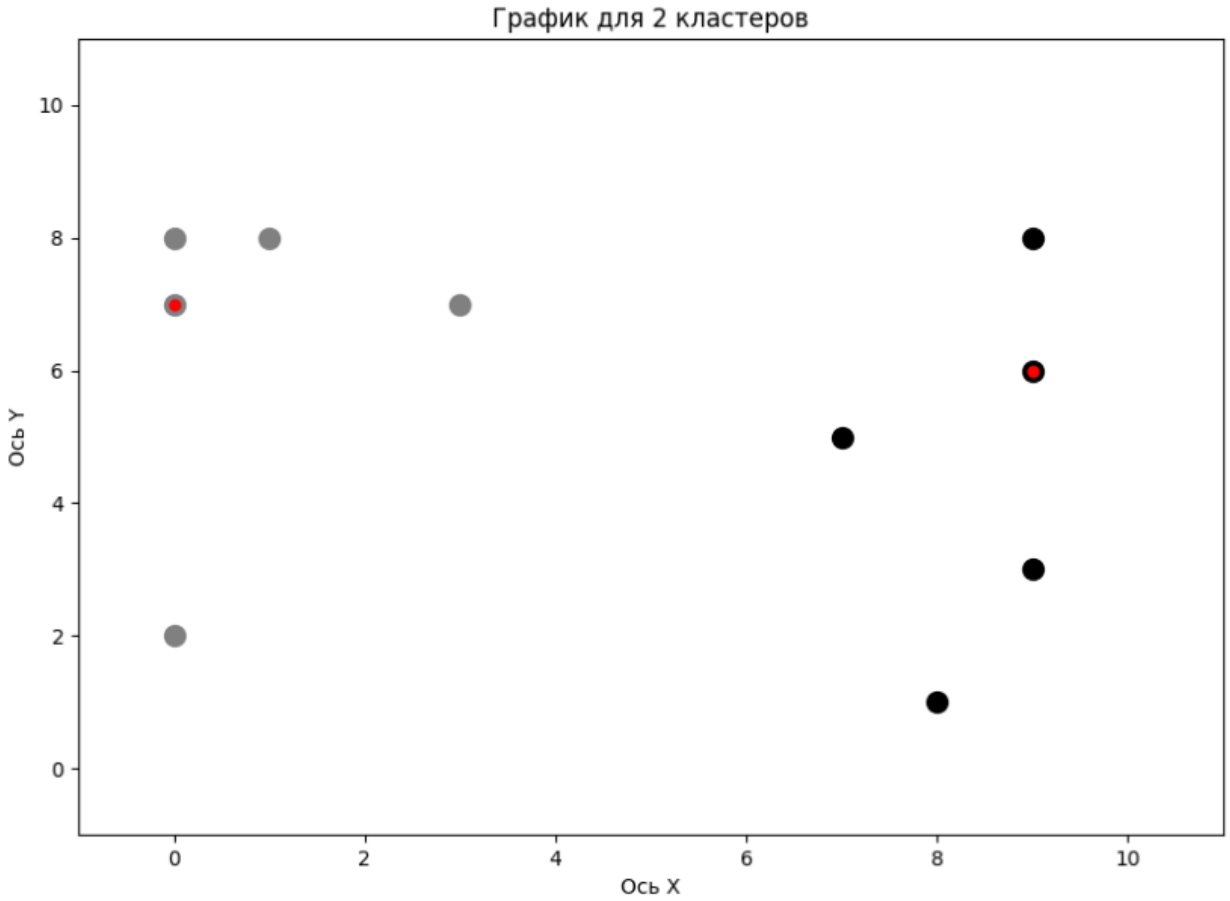

Рисунок 6.4 – Финальный график

7 CURE

7.1 Постановка задачи

Придумать свою предметную область и алгоритм CURE.

7.2 Описание предметной области

Рандомно сгенерирован набор данных из (Таблица 7.1).

Таблица 7.1 – Данные

№	x	y
1	9	6
2	0	7
3	7	5
4	9	8
5	1	8
6	9	3
7	0	8
8	8	1
9	0	2
10	3	7

7.3 Описание алгоритма

Чтобы избежать проблем с неоднородными размерами или формами кластеров, CURE использует алгоритм иерархической кластеризации, который принимает компромиссное решение между центром тяжести и всеми крайностями. В алгоритме CURE выбирается постоянная с точек кластера с хорошим распределением и эти точки стягиваются к центру тяжести кластера на некоторое значение. Точки после стягивания используются как представители кластера. Кластеры с ближайшей парой представителей объединяются на каждом шаге алгоритма иерархической кластеризации CURE. Это даёт возможность алгоритму CURE правильно распознавать кластеры и делает его менее чувствительным к выбросам.

За счет использования точек-представителей алгоритм CURE устойчив к выбросам и может выделять кластеры сложной формы и различных размеров.

Описание шагов

Шаг 1. Если все данные использовать сразу как входные для CURE, то эффективность алгоритма будет низкая, а время выполнения большим. Поэтому на первом шаге мы случайным образом выбираем часть точек, которые помещаются в память, затем группируем наиболее похожие с помощью иерархического метода в заранее заданное число кластеров. Дальше работаем с кластерами.

Шаг 2. Для каждого кластера выбираем с точек-представителей, максимально удаленных друг от друга. Чисто с остается постоянным.

Шаг 3. Объединяем кластеры с наиболее похожими наборами точек-представителей. Если не достигнуто нужное число кластеров, то перейти на шаг 2.

Выбранные точки сдвигаются на следующем шаге на α к центроиду кластера. Алгоритм становится основанным на методе поиска центроида при $\alpha = 1$, и основанным на всех точках кластера при $\alpha = 0$ (рисунок 7.1).

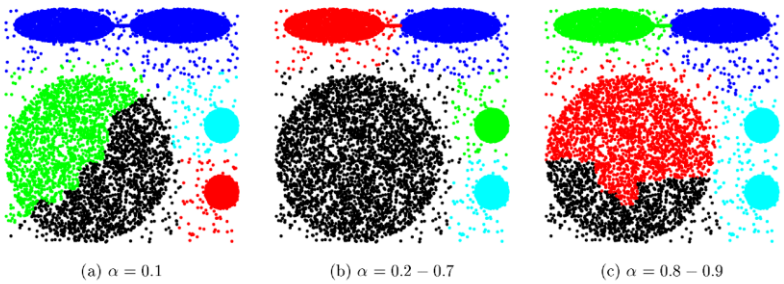

Рисунок 7.1 – Примеры работы алгоритма

Таким образом, для каждого i-го кластера вычисляется центроида μ_i по формуле:

$$\mu_i = \frac{1}{s_i} \sum_{j \in S_i} x^j \tag{7.1}$$

где S_i – номера строк матрицы x, входящие в i-й кластер; s_i – количество элементов в множестве S_i; x^j – j-я строка матрицы x.

Сдвиг представителей к центроиде происходит следующим образом.

Для каждого представителя x^y из k-го кластера выполняется сдвиг к его центроиде μ_k в α раз:

$$\hat{x}^y = \alpha\mu_k + (1-\alpha)x^y \quad (7.2)$$

Получаем множество $\hat{R}_k$ из с точек, которые будут далее использоваться как представители этого кластера.

7.4 Ручной расчёт

С помощью иерархической кластеризации доведём количество кластеров до 6.

Для каждой из точек будем искать наименьшее евклидово расстояние до всех точек и выберем наименьшее и распределим их по кластерам.

В первом круге самое минимальное расстояние будет между точками 2 и 7 $min(len) = \sqrt{(0-0)^2 + (7-8)^2} = 1{,}0$ Объединяем их в один кластер.

Продолжаем объединять пока не достигнем 6 уникальных кластеров таблица 7.2).

Таблица 7.2 – Иерархическое распределение точек по кластерам

№	кластер
1	0
2	1
3	0
4	0
5	1
6	2
7	1
8	3
9	4
10	5

Для каждого кластера создадим точек его представителей, максимально удалённых друг от друга.

Для каждого из кластеров посчитаем расстояние от центра до точек представителей.

$$10{,}31 / 4 * 0{,}75 = 1.9$$

$$5{,}16 / 4 * 0{,}75 = 0.96$$

$$21{,}63 / 4 * 0{,}75 = 4{,}06$$

$$23{,}43 / 4 * 0{,}75 = 4{,}39$$

$$17{,}49 / 4 * 0{,}75 = 3{,}38$$

$$0 / 4 * 0{,}75 = 0{,}0$$

Если расстояние от репрезентативных точек до точек не входящих в кластер меньше, чем расстояние от центра до представителей то присоединяем точку к кластеру.

7.5 Программная реализация

Создадим алгоритм CURE в программной реализации на языке высокого уровня Python [2].

Полученный результат работы программы CURE (Рисунок 7.2 – Рисунок 7.4).

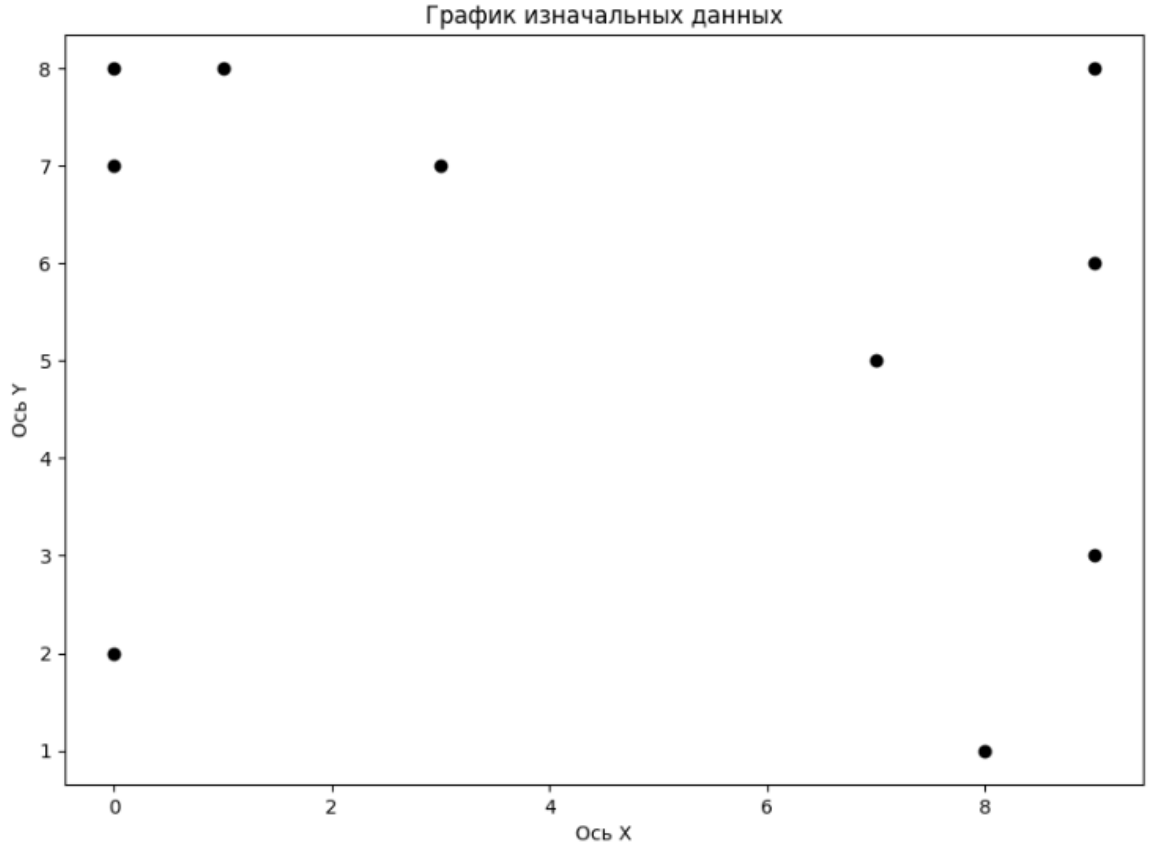

Рисунок 7.2 – Начальные данные

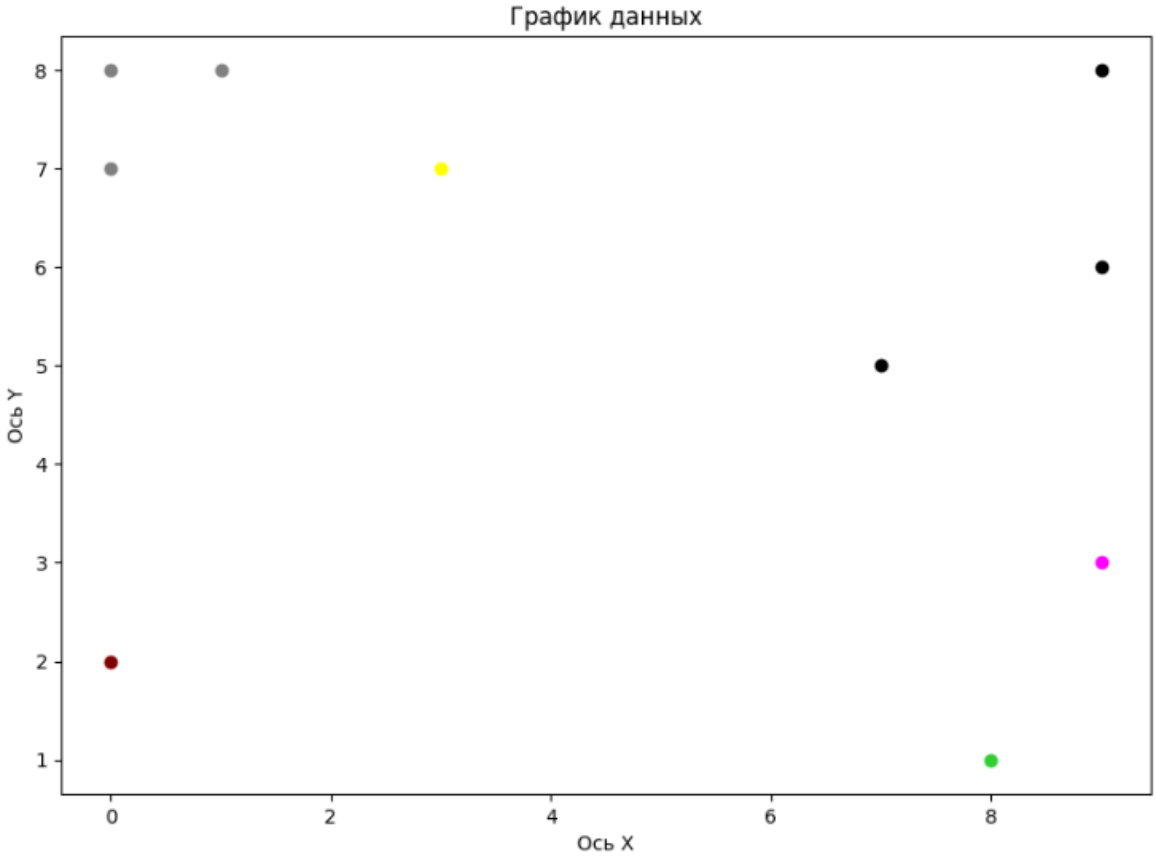

Рисунок 7.3 – График кластеризации после иерархической кластеризации

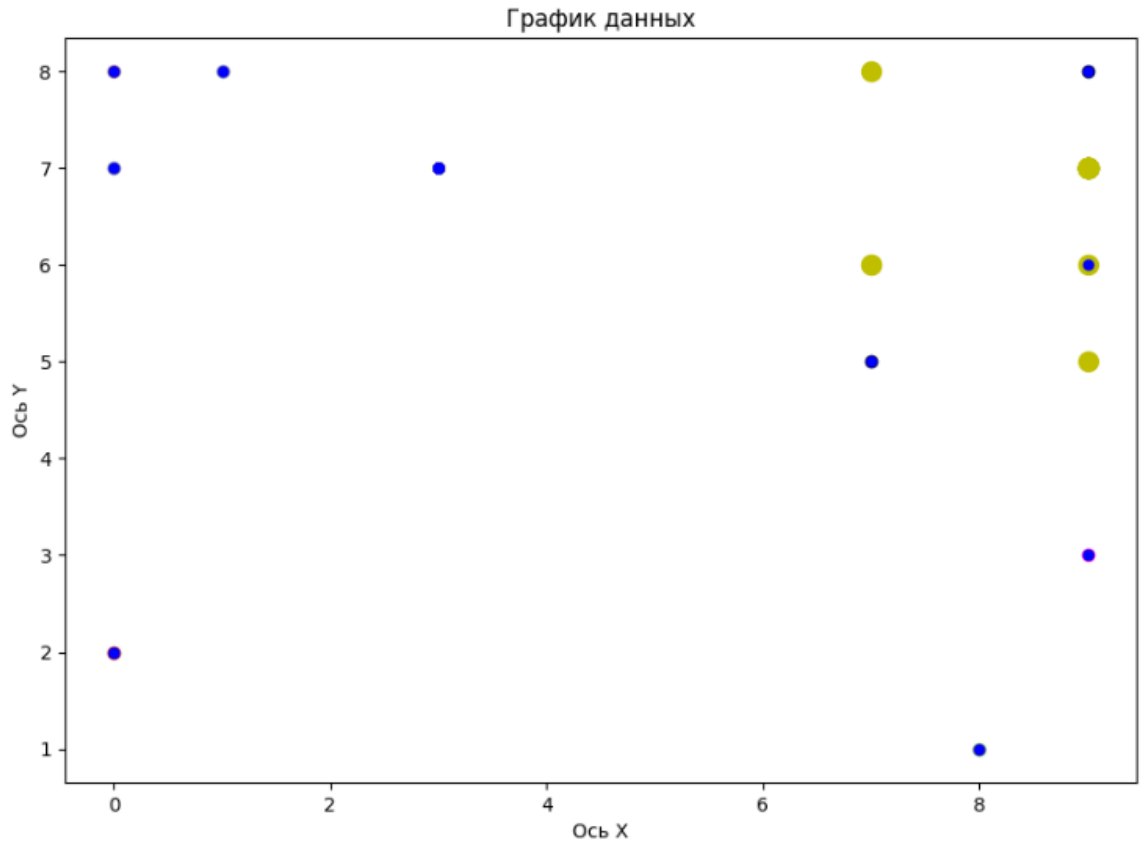

Рисунок 7.4 – График с репрезентативными точками

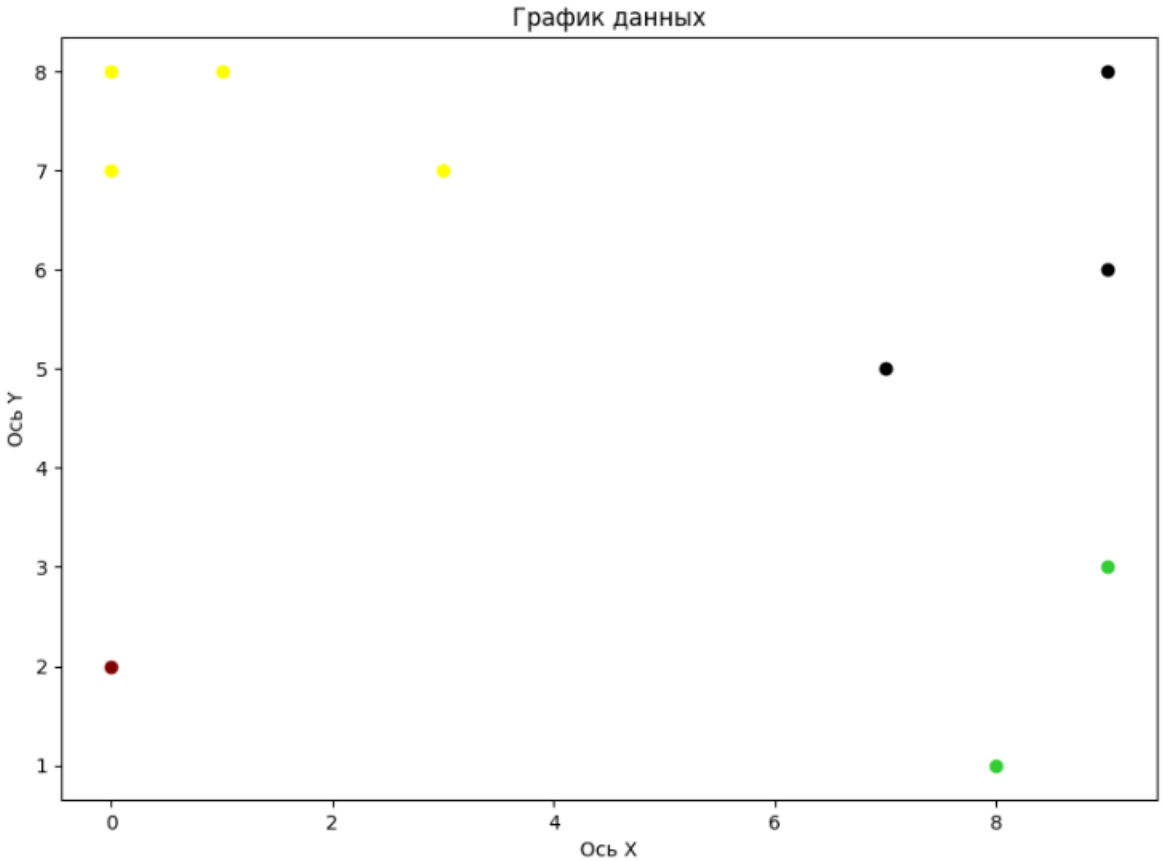

Рисунок 7.5 – Финальный график данных

8 MST

8.1 Постановка задачи

Придумать свою предметную область и реализовать алгоритм MST.

8.2 Описание предметной области

Рандомно сгенерирован набор данных из (Таблица. 6.1).

Таблица 6.1 – Данные

№	1 вершина	2 вершина	Длина пути
1	0	1	38
2	0	2	22
3	0	3	22
4	0	4	24
5	1	2	14
6	1	3	27
7	1	4	32
8	2	3	26
9	2	4	28
10	3	4	39

8.3 Описание алгоритма

Модели графов, в которых с каждым ребром связаны веса или стоимости, используются во многих приложениях. В картах авиалиний, в которых ребрами отмечены авиарейсы, такие веса означают расстояния или стоимости билетов. В электронных схемах, где ребра представляют проводники, веса могут означать длину проводника, его стоимость или время прохода сигнала. В задачах календарного планирования веса могут представлять время или трудоемкость либо выполнения задачи, либо ожидания ее завершения.

Взвешенный неориентированный граф представляет собой множество взвешенных ребер. MST-дерево есть множество ребер минимального общего веса, которые соединяют все вершины. Минимальное остовное дерево (minimal

spanning tree — MST, другие варианты перевода — минимальный остов, минимальный каркас, минимальный скелет) взвешенного графа есть остовное дерево, вес которого (сумма весов его ребер) не превосходит вес любого другого остовного дерева.

Существует несколько алгоритмов для нахождения минимального остовного дерева.

Алгоритм Прима,

Алгоритм Краскала (или алгоритм Крускала),

Алгоритм Борувки,

Алгоритм обратного удаления (получение минимального остовного дерева из связного рёберно взвешенного графа).

Алгоритм Прима и поиск по приоритету

Алгоритм Прима, похоже, наиболее прост для реализации из всех алгоритмов поиска MST и рекомендуется для насыщенных графов. В нем используется сечение графа, состоящее из древесных вершин (выбранных для MST) и недревесных вершин (еще не выбранных в MST-дерево). Вначале мы выбираем в качестве MST-дерева произвольную вершину, затем помещаем в MST минимальное перекрестное ребро (которое превращает недревесную вершину MST-дерева в древесную) и повторяем эту же операцию V-1 раз, пока все вершины не окажутся в дереве.

Здесь главное – найти кратчайшее расстояние от каждой недревесной вершины до дерева. Для реализации этой идеи нам потребуются такие структуры данных, которые предоставляют следующую информацию:

- Ребра дерева.
- Самое короткое дерево, соединяющее недревесную вершину с деревом.
- Длина этого ребра.

8.4 Ручной расчёт

Выберем начальную точку 1.

Ищем минимальное расстояние из точки 1 до любой другой вершины. Минимальное расстояние от вершины 1 до вершины 2 = 14.

Теперь из точек 1 и 2 ищем минимальное расстояние до любой другой вершины. Минимальное расстояние от вершины 2 до вершины 0 = 22

Теперь из точек 1, 2, 0 ищем минимальное расстояние до любой другой вершины. Минимальное расстояние от вершины 0 до вершины 3 = 22

Теперь из точек 1, 2, 0, 3 ищем минимальное расстояние до любой другой вершины. Минимальное расстояние от вершины 0 до вершины 4 = 24

Общая длина = 82

8.5 Программная реализация

Создадим алгоритм MST в программной реализации на языке высокого уровня Python [2].

Полученный результат работы программы MST (Рисунок 8.1 – Рисунок 8.5).

```
Вывод начального графа

Введите начальню точку от 0 до 4: 1
Вывод графа для 0 итерации

14 = long
[1, 2]
Вывод графа для 1 итерации

22 = long
[1, 2, 0]
Вывод графа для 2 итерации

22 = long
[1, 2, 0, 3]
Вывод графа для 3 итерации

24 = long
[1, 2, 0, 3, 4]
Вывод графа для 4 итерации


[1, 2, 0, 3, 4]
[4, 1, 2, 3]
Итерация 0: от 1 до 2
Итерация 1: от 0 до 2
Итерация 2: от 0 до 3
Итерация 3: от 0 до 4
len = 82
```

Рисунок 8.1 – Консольный вывод алгоритм mst

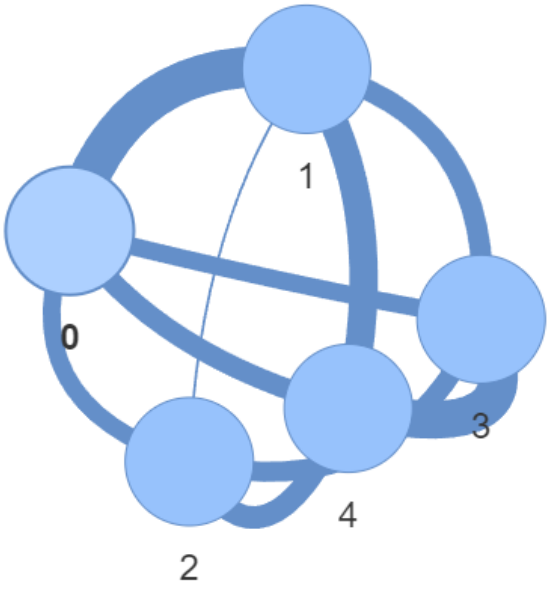

Рисунок 8.2 – Начальный граф

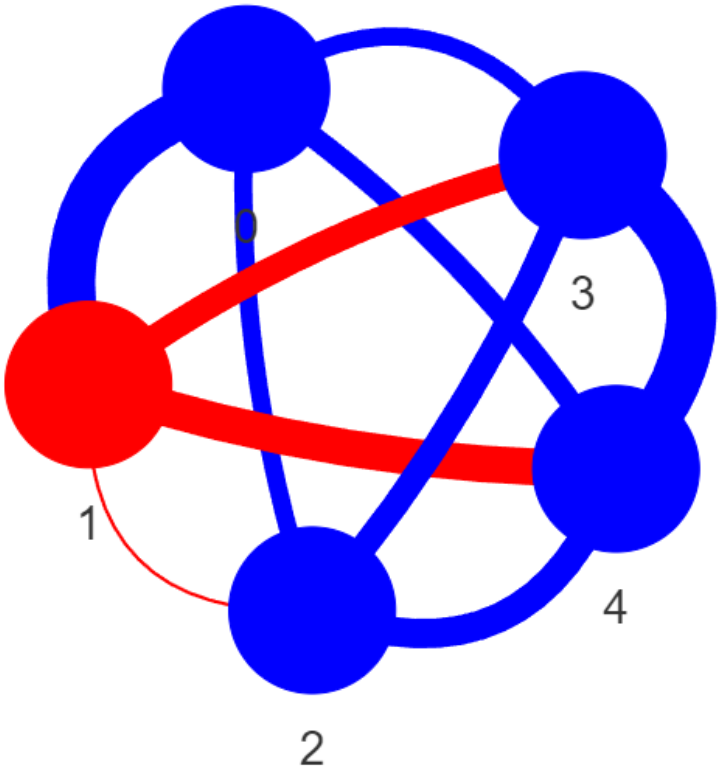

Рисунок 8.3 – Стартовая точка

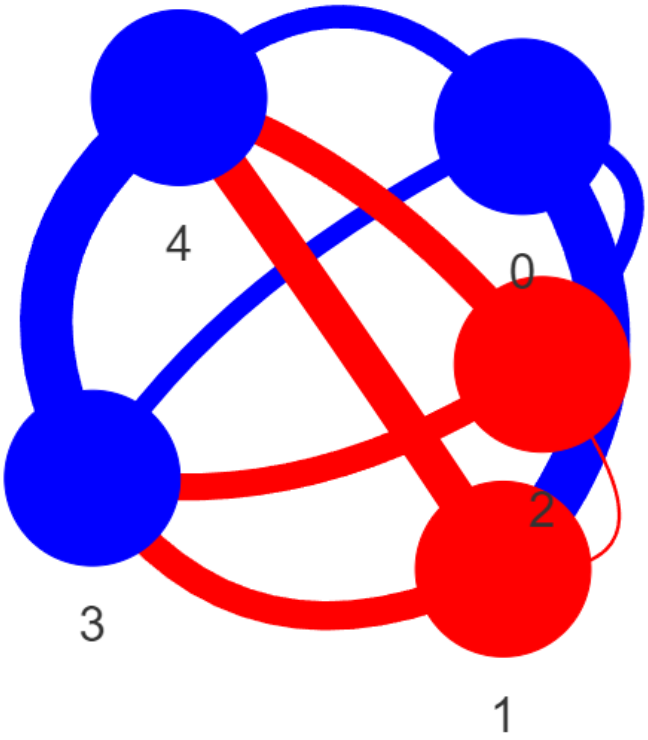

Рисунок 8.4 – Первая итерация

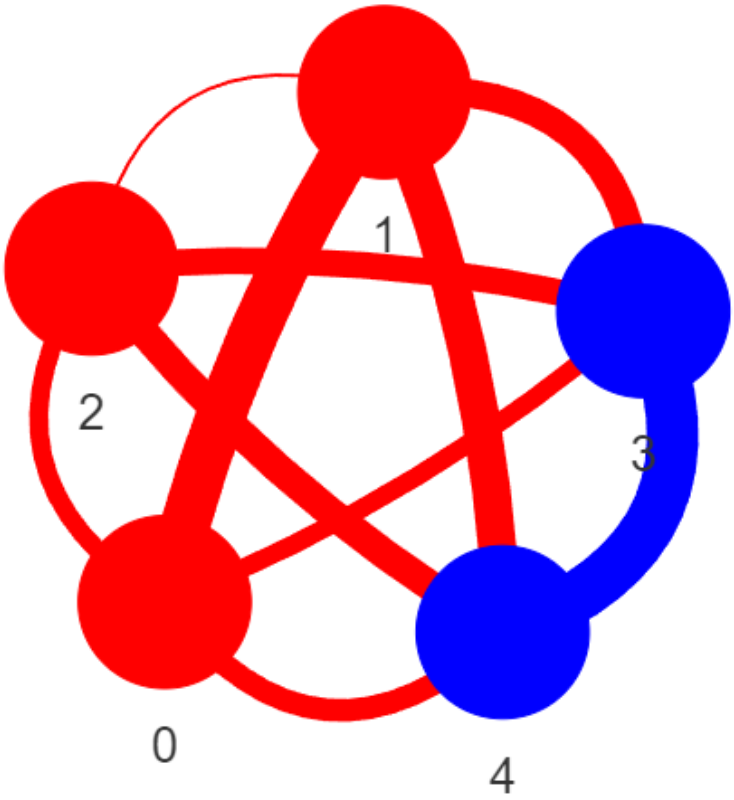

Рисунок 8.5 – Вторая итерация

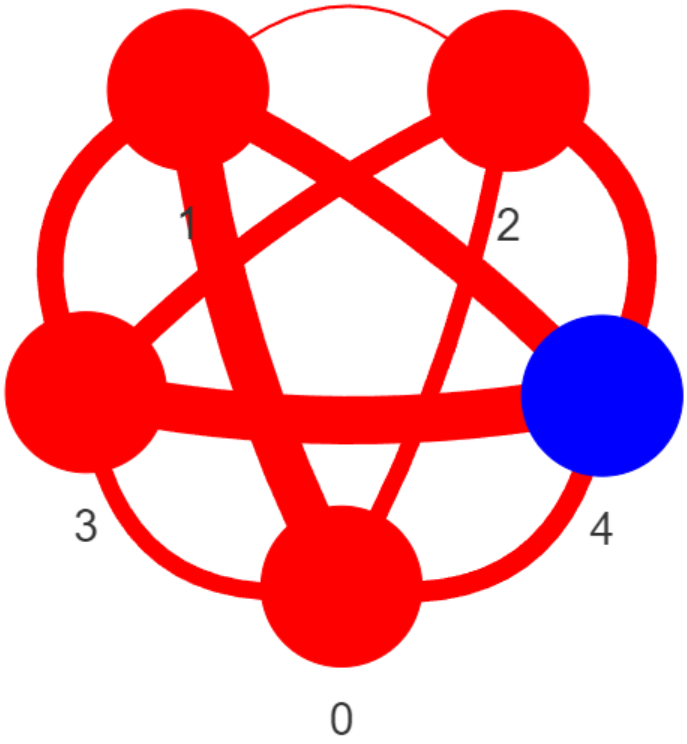

Рисунок 8.6 – Третья итерация

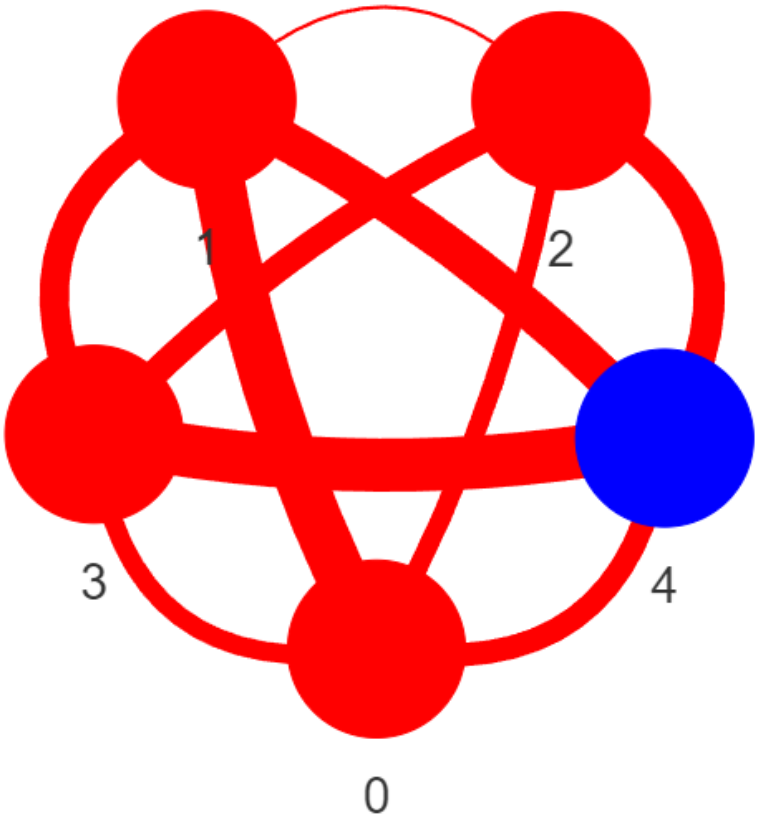

Рисунок 8.7 – Четвёртая итерация

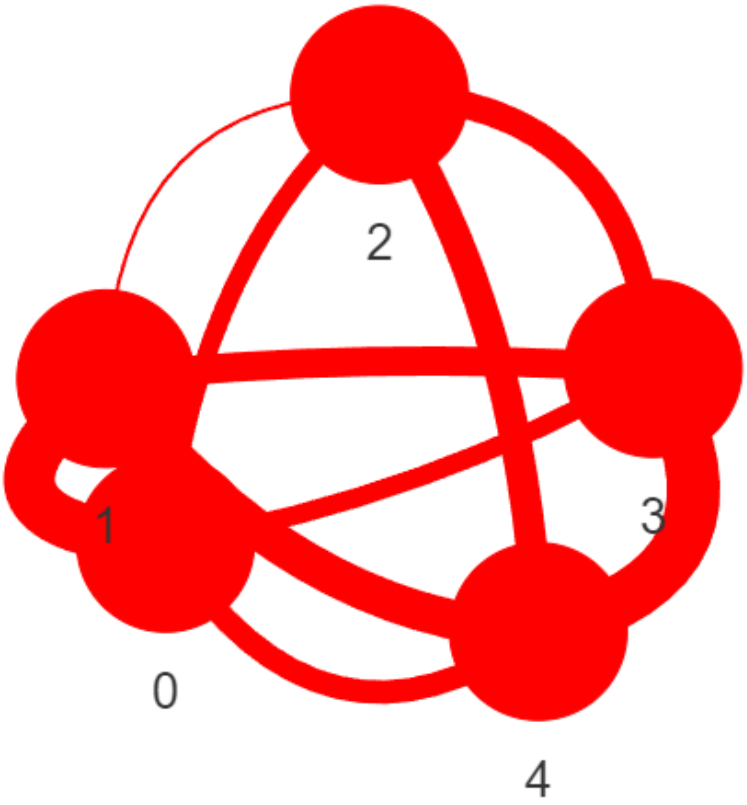

Рисунок 8.8 – Финальный граф

9 ID3

9.1 Постановка задачи

Придумать свою предметную область и реализовать алгоритм ID3.

9.2 Описание предметной области

Данные для обучения (Таблица 9.1).

Таблица 9.1 – Данные

№	1 переменная	2 переменная	Класс
1	1	0	1
2	1	0	1
3	1	0	1
4	0	1	1
5	0	0	0
6	0	0	0
7	0	0	0
8	1	1	0
9	1	0	1
10	1	0	1

9.3 Описание алгоритма

Одними из наиболее популярных алгоритмов построения деревьев решений являются алгоритм ID3 и его модификация C4,5. Алгоритм ID3 начинает работу со всеми обучающими примерами в корневом узле дерева. Для разделения множества примеров корневого узла выбирается один из атрибутов, и для каждого значения, принимаемого этим атрибутом, строится ветвь и создается дочерний узел. Затем все примеры распределяются по дочерним узлам в соответствии со значением атрибута (рисунок 9.1).

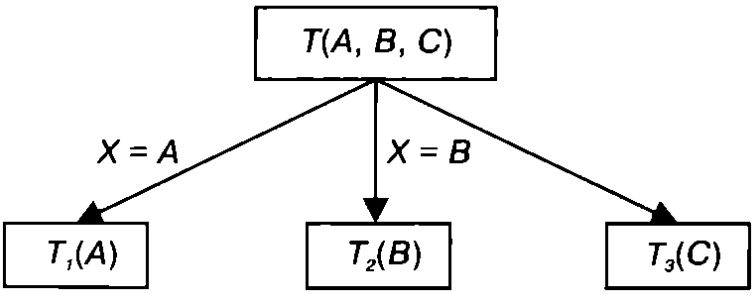

Рисунок 9.1 – Разбиение по атрибуту

Поясним это на рис. 1. Пусть атрибут X принимает три значения: A, B и C. Тогда при разбиении исходного множества T по атрибуту X алгоритм создаст три дочерних узла T_1 (A), T_2 (B) и T_3 (C), в первый из которых будут помещены все записи со значением A, во второй со значением B, а в третий со значением C.

Алгоритм повторяется рекурсивно до тех пор, пока в узлах не останутся только примеры одного класса, после чего узлы будут объявлены листами и разбиение прекратится. Наиболее проблемным этапом алгоритма является выбор атрибута, по которому будет производиться разбиение в каждом узле. Для выбора атрибута разбиения ID3 использует критерий, называемый приростом информации, или уменьшением энтропии.

Введем в рассмотрение меру прироста информации, вычисляемую как (9.1):

$$Gain(S) = Info\,(T) - Info_S(T) \qquad (9.1)$$

Где $Info\,(T)$– энтропия множества T до разбиения;

$Info_S(T)$ – энтропия после разбиения S.

Данная мера представляет собой прирост количества информации, полученный в результате разделения множества T на подмножества $T_1, T_2, \dots, T_k$ с помощью разбиения $Gain(S)$. В качестве наилучшего атрибута для использования вразбиении S выбирается тот атрибут, который обеспечивает наибольший прирост информации Gain(S).

9.4 Ручной расчёт

Определим начальную энтропию по формуле. Количество элементов класса 0 = 4, количество элементов класса 1 = 4.

$$Entropy_{start} = \left(\frac{4}{8}\right) * \log_2\left(\frac{4}{8}\right) - \left(\frac{4}{8}\right) * \log_2\left(\frac{4}{8}\right) = 1$$

Если первая переменная = 1, то есть 3 исхода, где класс 0 и 1 исход, где класс 1.

$$E_{x_1=1} = \left(\frac{3}{4}\right)\log_2\left(\frac{3}{4}\right) + \left(\frac{1}{4}\right)\log_2\left(\frac{1}{4}\right) = 0{,}81$$

Если первая переменная = 0, то есть 3 исхода, где класс 1 и 1 исход, где класс 0.

$$E_{x_1=1} = \left(\frac{3}{4}\right)\log_2\left(\frac{3}{4}\right) + \left(\frac{1}{4}\right)\log_2\left(\frac{1}{4}\right) = 0{,}81$$

Если вторая переменная = 0, то есть 3 исхода, где класс 1 и 3 исход, где класс 0.

$$E_{x_1=1} = \left(\frac{3}{4}\right)\log_2\left(\frac{3}{4}\right) + \left(\frac{3}{4}\right)\log_2\left(\frac{3}{4}\right) = 1{,}0$$

Если вторая переменная = 1, то есть 1 исход, где класс 1 и 1 исход, где класс 0.

$$E_{x_1=1} = \left(\frac{1}{4}\right)\log_2\left(\frac{1}{4}\right) + \left(\frac{1}{4}\right)\log_2\left(\frac{1}{4}\right) = 1{,}0$$

Для выбора какой критерий выбирать первым посчитаем IG по формуле 9.1.

$$IG_{x_1} = 1 - \left(\left(\frac{4}{8}\right) * 0{,}81 + \left(\frac{4}{8}\right) * 0{,}81\right) = 1{,}18$$

$$IG_{x_2} = 1 - \left(\left(\frac{6}{8}\right) * 0{,}81 + \left(\frac{2}{8}\right) * 0{,}81\right) = 1{,}18$$

Так как IG больше у первого критерия, выбираем его в качестве первого для дерева.

9.5 Программная реализация

Создадим алгоритм ID3 в программной реализации на языке высокого уровня Python [2].

Полученный результат работы программы ID3 (Рисунок 9.2 – Рисунок 9.5).

```
Уникальные значения у критериев: [[0, 1], [0, 1]]

Количество значений у критериев: [[4, 4], [6, 2]]

Уникальные классы: [0, 1]

Количество для уникальных классов: [4, 4]

Начальная энтропия = 1.0

Количество элементов для подсчёта энтропии: [[[3, 1], [1, 3]], [[3, 3], [1, 1]]]

mass_entropy [[0.8112781244591328, 0.8112781244591328], [1.0, 1.0]]
IG [0.18872187554086717, 0.0]
[0, 0.18872187554086717]

[[[3, 0], [0, 1]], [[0, 1], [3, 0]]]
[[3, 1], [1, 3]]
mass_entropy [[0.0, 0.0], [0.0, 0.0]]
IG [0.8112781244591328, 0.8112781244591328]


Дерво:
[0]
[1, 1]

Ентропия на всех участках дерева:
1.0
[0.8112781244591328, 0.8112781244591328]
[[0.0, 0.0], [0.0, 0.0]]
```

Рисунок 9.2 – Консольный вывод алгоритм ID3

ЗАКЛЮЧЕНИЕ

В результате выполнения практических заданий, в рамках курсовой работы, были получены знания об основных алгоритмах кластеризации, аффинитивного анализа, регрессионных моделей, а также деревьев. Полученные знания были закреплены путем программной реализации рассмотренных алгоритмов, а именно алгоритма априори, линейной и логистической регрессий, наивного классификатора Байеса, дерева решений, а также алгоритмов кластеризации PAM, CURE, К-средних и MST.

Также, в рамках выполнения практических работ было ознакомление с системой Deductor Studio. Была построена база транзакций по выбранной предметной области, а также применены методы поиска ассоциативных правил в этой системе.

Опыт подкреплён практической реализацией на языке программирования Python.

Что в дальнейшем благополучно скажется на изучении следующих дисциплин, связанных с машинным обучением, искусственным интеллектом и системами поддержки принятия решений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Loginom: Apriori — масштабируемый алгоритм поиска ассоциативных правил — URL: https://loginom.ru/blog/apriori/ (Дата обращения: 25.05.2023)

2. Документация по языку программирования Python: сайт. – URL: https://docs.python.org/3/. – Текст: электронный.

3. Технологии обучения: кластеризация и классификация [Электронный ресурс]: Практикум / А.Б. Сорокин, Л.М. Железняк — М.: РТУ МИРЭА, 2021. — 1 электрон. опт. диск (CD-ROM).

4. UCI Machine Learning Repository — URL: https://archive.ics.uci.edu/ml/datasets/banknote+authentication (Дата обращения: 25.05.2023)

5. Этап проектирования для программной инженерии [Электронный ресурс]: учебно-методическое пособие / А.Б. Сорокин, О.В. Платонова, Л.М. Железняк – Москва: РТУ МИРЭА, 2021.

6. Интеллектуальные системы [Электронный ресурс]: учебно-метод. пособие / А. Б. Сорокин. — М.: МИРЭА, 2016. — Электрон. опт. диск (ISO)

7. Программное обеспечение интеллектуальных систем [Электронный ресурс]: учебно-метод. пособие / В. И. Тихвинский, А. Б. Сорокин. — М.: РТУ МИРЭА, 2019. — Электрон. опт. диск (ISO)

8. Лекции по дисциплине «Проектирование систем поддержки принятия решений» / А. Б. Сорокин.

9. Lecture 62 — The CURE Algorithm (Advanced) | Stanford University: сайт. – URL: https://www.youtube.com/watch?v=JrOJspZ1CUw&t=185s. – Видео: электронный.

10. Логистическая Регрессия | Logistic Regression | Линейная модель для классификации |МАШИННОЕ ОБУЧЕНИЕ: сайт. – URL: https://www.youtube.com/watch?v=9BoVCdedvW8&t=307s. – Видео: электронный.

ПРИЛОЖЕНИЯ

Приложение А – Листинг кода для ассоциативных правил

Приложение Б – Листинг кода для k-means

Приложение В – Листинг кода линейной регрессии

Приложение Г – Листинг кода байесовского классификатора

Приложение Д – Листинг кода логистической регрессии

Приложение Е – Листинг кода PAM

Приложение Ж – Листинг кода CURE

Приложение З – Листинг кода ID3

Приложение А

Листинг кода для ассоциативных правил

Листинг А.1 – Используемые библиотеки

```
import pandas as pd
import numpy as np
import seaborn as sn
import itertools
import time
```

Листинг А.2 – Файл txt с данными для анализа

```
"receipt_number","product"
15,"Передние тормозные колодки"
15,"Моторное масло"
15,"Масляный фильтр ДВС"
30,"Фильтр салона"
30,"Свечи зажигания на моделях с бензиновым ДВС"
30,"Передние тормозные колодки"
30,"Моторное масло"
30,"Воздушный фильтр ДВС"
30,"Масляный фильтр ДВС"
30,"Топливный фильтр тонкой очистки"
40,"Передние тормозные колодки"
40,"Фильтр салона"
45,"Моторное масло"
45,"Масляный фильтр ДВС"
50,"Передние тормозные колодки"
50,"Масло в коробке передач"
50,"Жидкость ГУР"
50,"Масляный фильтр коробки передач"
55,"Тормозная жидкость"
55,"Передние тормозные колодки"
60,"Свечи зажигания на моделях с бензиновым ДВС"
60,"Задние тормозные колодки"
60,"Моторное масло"
60,"Воздушный фильтр ДВС"
60,"Масляный фильтр ДВС"
60,"Топливный фильтр тонкой очистки"
60,"Фильтр салона"
75,"Приводной ремень"
75,"Передние тормозные колодки"
75,"Основной аккумулятор"
75,"Моторное масло"
75,"Масляный фильтр ДВС"
80,"Рычаги подвески"
80,"Передние тормозные колодки"
80,"Фильтр салона"
90,"Свечи зажигания на моделях с бензиновым ДВС"
90,"Шрусы или их составные части"
90,"Моторное масло"
90,"Воздушный фильтр ДВС"
90,"Масляный фильтр ДВС"
90,"Топливный фильтр тонкой очистки"
```

Листинг А.3 – Функция main

```
data = pd.read_csv('static/data//new_data.csv')
data.hist()
unique_receipts = data.receipt_number.unique()
print(len(unique_receipts))
unique_receipts
count_receipt = len(unique_receipts)
unique_products = data['product'].unique()
print(len(unique_products))
unique_products
data_np = data.to_numpy()
group_products_receipts = []
for i in unique_receipts:
    micro_data = []
    for j in data_np:
        if j[0] == i:
            micro_data.append(j[1])
    group_products_receipts.append(micro_data)
```

Листинг А.4 – Функция создание всех сочетаний

```
mass_group_products = []
mass_group_products_str = []
count = 1
last_len = 0
for mass in group_products_receipts:
    start_time = time.time()
    print('-' * 25)
    print(f'Элемент {count} / {len(group_products_receipts)}')
    print('Количество элементов =', len(mass))
    if len(mass) > 4:
        n = 4
    else:
        n = len(mass)
    print(f'Максимум элементов = {n}')
    groups = []
    for count_item in range(1, n + 1):
        permutation = itertools.permutations(mass, count_item)
        comb_not_sort = []
        for comb in permutation:
            groups.append(list(comb))
    for i in range(len(groups)):
        for j in range(len(groups)):
            if i != j and set(groups[j]).isdisjoint(groups[i]) and
set(groups[i]).isdisjoint(groups[j]):
                if sum([groups[i], groups[j]], []) not in
mass_group_products_str:
                    mass_group_products_str.append(sum([groups[i], groups[j]],
[]))
                    mass_group_products.append([groups[i], groups[j]])
    print("Время = %s seconds" % (time.time() - start_time))
    print(f"Получено сочетаний {len(mass_group_products) - last_len}")
    print(f"Всего элементов = {len(mass_group_products)}")
    count += 1
    last_len = len(group_products_receipts)
print('\n' * 1)
print(len(mass_group_products))
```

Листинг А.5 – Функция расчёта поддержки

```
for i in range(len(mass_group_products)):
    count = 0
    for j in group_products_receipts:
        if set(mass_group_products[i][0]).issubset(j) and
set(mass_group_products[i][1]).issubset(j):
            count += 1
    if i == 1:
        print(f'{count*100}/{count_receipt} =
{round((count*100/count_receipt),2)}')
    mass_group_products[i].append((round((count*100/count_receipt),2)))
```

Листинг А.6 – Функция расчёта достоверности

```
for i in range(len(mass_group_products)):
    count_one = 0
    count_two = 0
    for j in group_products_receipts:
        if set(mass_group_products[i][0]).issubset(j) and
set(mass_group_products[i][1]).issubset(j):
            count_one += 1
    for j in group_products_receipts:
        if set(mass_group_products[i][0]).issubset(j):
            count_two += 1
    if i == 1:
        print(f'{count_one}/{count_two} = {round((count_one*100/count_two),2)}')
    if count_two != 0:
        mass_group_products[i].append((round((count_one*100/count_two),2)))
    else:
        mass_group_products[i].append(0)
```

Листинг А.7 – Функция расчёта лифта

```
for i in range(len(mass_group_products)):
    count_one = 0
    for j in group_products_receipts:
        if set(mass_group_products[i][1]).issubset(j):
            count_one += 1
    if i == 1:
        print(f'{count_one}/{count_receipt} =
{round((count_one*100/count_receipt),2)}')

    if (count_one*100/count_receipt) != 0:
        mass_group_products[i].append((round(mass_group_products[i][-
1]/(count_one*100/count_receipt),2)))
    else:
        mass_group_products[i].append(0)
```

Листинг А.8 – Функция получения финальных данных

```
Dataframe = pd.DataFrame(dataframe, columns =['Suported', 'Reliability',
'Lift'], index=name_for_index)
Dataframe = (Dataframe.loc[Dataframe.Suported < 89])
Dataframe = (Dataframe.loc[20 < Dataframe.Suported])
Dataframe = (Dataframe.loc[30 < Dataframe.Reliability])
Dataframe = (Dataframe.loc[Dataframe.Reliability < 70])
Dataframe.sort_values('Lift', ascending=False)
```

Приложение Б

Листинг кода для k-means

Листинг Б.1 – Используемые библиотеки

```
import matplotlib.pyplot as plt
import math
import random
```

Листинг Б.2 – Вывод начального графа

```
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot()
for i in mass:
    ax.scatter(i[0], i[1], color='b')
ax.grid()
plt.xlim([-1, 11])
plt.ylim([-1, 11])
plt.show()
```

Листинг Б.3 – Метод локтя

```
min = 1
max = 3
mass_elbow_value = []
# задаём начальную точку
# cluster_centers = [[round(random.uniform(0.0, 10.0), 2),
round(random.uniform(0.0, 10.0), 2)]]
cluster_centers = [[6.32, 1.47]]
# [[6.32, 1.47]]
print(cluster_centers)
# цикл для определения коилчество кластеров "методом локтя"
for k in range(min, max):
    # для создания точки для определения нового кластера кластера
    if k != 1:
        # считаем евклидово расстояние до каждых точек относительно центра
кластера
        new_mass_point_to_claster = []
        for i in mass:
            new_euclidean_distances = []
            for j in cluster_centers:
                new_euclidean_distances.append(round(math.sqrt((i[0] - j[0]) **
2 + (i[1] - j[1]) ** 2), 2))
            new_mass_point_to_claster.append(new_euclidean_distances)
        print(new_mass_point_to_claster)
        maximum = [-1, -1]
```

Продолжение Листинга Б.3

```
        # выбираем самую дальнюю точку
        for i in range(len(new_mass_point_to_claster)):
            long = 0
            for j in new_mass_point_to_claster[i]:
                long += j
            if long > maximum[0] and mass[i] not in cluster_centers:
                maximum = [long, i]
        # добавляем коодинаты новой точки
        cluster_centers.append(mass[maximum[-1]])
    print(cluster_centers)
    mass_point_to_claster = []

    # расчитываем центры кластеров
    for i in mass:
        euclidean_distances = []
        for j in cluster_centers:
            euclidean_distances.append(round(math.sqrt((i[0] - j[0]) ** 2 +
(i[1] - j[1]) ** 2), 2))
        mass_point_to_claster.append(euclidean_distances)
    for i in range(len(mass_point_to_claster)):
        minimum = 999999
        numer = -1
        for j in range(len(mass_point_to_claster[i])):
            if mass_point_to_claster[i][j] < minimum:
                minimum = mass_point_to_claster[i][j]
                numer = j + 1
        mass_point_to_claster[i] = numer
    #     определяем центры кластеров
    for i in range(len(cluster_centers)):
        print(f"{i} ||||")
        count = 0
        x_ = 0
        y_ = 0
        for j in range(len(mass)):
            if mass_point_to_claster[j] == i + 1:
                count += 1
                x_ += mass[j][0]
                y_ += mass[j][1]
                print(mass[j][0], ' | 0')
                print(mass[j][1], ' | 1')
                print()
        print(count)
        cluster_centers[i][0] = x_ / count
        cluster_centers[i][1] = y_ / count
        print(cluster_centers[i][0])
        print(cluster_centers[i][1])

    # рисуем график и раскрашиваем точки
    fig = plt.figure(figsize=(10, 7))
    ax1 = fig.add_subplot()
    for i in range(k):
        count = 0
        x_ = 0
        y_ = 0
        for j in range(len(mass)):
            if mass_point_to_claster[j] == i + 1:
                count += 1
                x_ += mass[j][0]
                y_ += mass[j][1]
```

Продолжение Листинга Б.3

```
        x_ = x_ / count
        y_ = y_ / count
        cluster_centers[i][0] = x_
        cluster_centers[i][1] = y_
        ax1.scatter(x_, y_, c='r', s=150)
    for i in range(len(mass_point_to_claster)):
        ax1.scatter(mass[i][0], mass[i][1],
c=mass_collor[mass_point_to_claster[i] - 1])
    #     plt.axis([-1, 11, -1, 11])
    ax1.grid()
    print(f'Schedule number {k}')
    plt.xlim([-1, 11])
    plt.ylim([-1, 11])
    plt.show()

    elbow_value = 0
    for i in range(k):
        for j in range(len(mass)):
            if mass_point_to_claster[j] == i + 1:
                elbow_value += round(math.sqrt((cluster_centers[i][0] -
mass[j][0]) ** 2 +
                                               (cluster_centers[i][1] -
mass[j][1]) ** 2), 2)
    mass_elbow_value.append(elbow_value)
```

Листинг Б.4 – Вывод графа метода локтя

```
fig = plt.figure(figsize=(16, 9))
axis = fig.add_subplot()
axis.grid()
mass_k = []
for i in range(1, max):
    mass_k.append(i)
plt.bar(mass_k, mass_elbow_value)
for i in range(len(mass_elbow_value)):
    print(f'{i+1} кластер = {mass_elbow_value[i]}')
```

Листинг Б.5 – Алгоритм k-means

```
k = 3
# cluster_centers = [[random.randint(0, 10), random.randint(0,10)]]
cluster_centers = [[6.32, 1.47]]
for l in range(k-1):
    new_mass_point_to_claster = []
    for i in mass:
        new_euclidean_distances = []
        for j in cluster_centers:
            new_euclidean_distances.append(round(math.sqrt((i[0]-j[0])**2 +
(i[1]-j[1])**2), 2))
        new_mass_point_to_claster.append(new_euclidean_distances)
    maximum = [-1, -1]
```

Продолжение Листинга Б.5

```
    # выбираем самую дальнюю точку
    for i in range(len(new_mass_point_to_claster)):
        long = 0
        for j in new_mass_point_to_claster[i]:
            long += j
        if long > maximum[0] and mass[i] not in cluster_centers:
            maximum = [long, i]
    # добавляем коодинаты новой точки
    cluster_centers.append(mass[maximum[-1]])
cluster_centers[:]
mass_point_to_claster = []
for i in mass:
    euclidean_distances = []
    for j in cluster_centers:
        euclidean_distances.append(round(math.sqrt((i[0]-j[0])**2 + (i[1]-
j[1])**2), 2))
    mass_point_to_claster.append(euclidean_distances)
for i in range(len(mass_point_to_claster)):
    minimum = 999999
    numer = -1
    for j in range(len(mass_point_to_claster[i])):
        if mass_point_to_claster[i][j] < minimum:
            minimum = mass_point_to_claster[i][j]
            numer = j + 1
    mass_point_to_claster[i] = numer
mass_point_to_claster
# старый центр кластера смещается в его центроид
for z in range(4):
    for i in range(k):
        count = 0
        x_ = 0
        y_ = 0
        for j in range(len(mass)):
            if mass_point_to_claster[j] == i+1:
                count += 1
                x_ += mass[j][0]
                y_ += mass[j][1]
        if x_ != 0:
            cluster_centers[i][0] = x_/count
        else:
            cluster_centers[i][0] = 0
        if y_ != 0:
            cluster_centers[i][1] = y_/count
        else:
            cluster_centers[i][1] = 0
    mass_point_to_claster = []
    for i in mass:
        euclidean_distances = []
        for j in cluster_centers:
            euclidean_distances.append(round(math.sqrt((i[0]-j[0])**2 + (i[1]-
j[1])**2), 2))
        mass_point_to_claster.append(euclidean_distances)
    for i in range(len(mass_point_to_claster)):
        minimum = 999999
        numer = -1
        for j in range(len(mass_point_to_claster[i])):
            if mass_point_to_claster[i][j] < minimum:
                minimum = mass_point_to_claster[i][j]
                numer = j + 1
        mass_point_to_claster[i] = numer
```

Листинг Б.6 – Вывод финального графа

```
fig = plt.figure(figsize=(10,7), )
ax1 = fig.add_subplot()
# mass_collor = ['y', 'b', 'g']
for i in range(len(cluster_centers)):
    ax1.scatter(cluster_centers[i][0], cluster_centers[i][1], c='r',s=150)

for i in range(len(mass_point_to_claster)):
    ax1.scatter(mass[i][0],mass[i][1], c=mass_collor[mass_point_to_claster[i]])
ax1.grid()
plt.xlim([-1, 11])
plt.ylim([-1, 11])
plt.show()
```

Приложение В

Листинг кода линейной регрессии

Листинг В.1 – Используемые библиотеки

```
import matplotlib.pyplot as plt
import math
```

Листинг В.2 – Вывод начального графа

```
fig = plt.figure(figsize=(10, 7))
axis = fig.add_subplot()
axis.grid()
for i in mass:
    if i[-1] == 0:
        axis.scatter(i[1], i[2], s=50, c='green')
    else:
        axis.scatter(i[1], i[2], s=50, c='r')
plt.xlim(0,25)
plt.ylim(0,2500)
plt.show()
```

Листинг В.3 – Начальные данные

```
mass = [[1, 13, 1000, 1323.4, 0],
        [2, 20, 600, 305.6, 1],
        [3, 17, 500, 741.8, 0],
        [4, 15, 1200, 1032.6, 1],
        [5, 16, 1000, 887.2, 1],
        [6, 12, 1500, 1468.8, 1],
        [7, 16, 500, 887.2, 0],
        [8, 14, 1200, 1178.0, 1],
        [9, 10, 1700, 1759.6, 0],
        [10, 11, 2000, 1614.2, 1]]
n = len(mass)
summ_x = 0
summ_y = 0
summ_x2 = 0
summ_xy = 0
for i in mass:
    summ_x += i[1]
for i in mass:
    summ_x2 += i[1] ** 2
for i in mass:
    summ_y += i[2]
for i in mass:
    summ_xy += i[1] * i[2]
```

Продолжение Листинга В.3

```
print('n =', n)
print('summ x =', summ_x)
print('summ x**2 =', summ_x2)
print('summ y =', summ_y)
print('summ xy =', summ_xy)

print(f'b0 * {n} + b1 * {summ_x} = {summ_y}')
print(f'{summ_x} · b0 + {summ_x2} · b1 = {summ_xy} |==| 149300')
print(f'b0 = {summ_y}/{n} - {summ_x}·b1 = {summ_y/n} - {summ_x/n}*b1 ')
print(f'{summ_x} * ({summ_y/n} - {summ_x/n}·b1) + {summ_x**2}·b1 =
{mass[0][1]*summ_y}')
b1 = round((-(summ_x * (summ_y/n)) + (summ_xy)) / (summ_x*(-summ_x/n) +
(summ_x2)), 3)
print(f'{(-(summ_x * (summ_y/n)) + (summ_xy))} / {(summ_x*(-summ_x/n) +
(summ_x2))}')
print(b1)
b0 = round(summ_y/n - summ_x/n*b1, 3)
print(f'{b0} = {summ_y/n} - {summ_x/n} * b1')
print(f'b0 = {b0}')
print(f'b1 = {b1}')
print(f'y^ = {b0} + {b1} * x')
dy = round(b0+b1 * mass[0][1], 3)
print(f'{dy} = {b0} + {b1} * x')
```

Листинг В.4 – Вывод коофицентов b0, b1, dy

```
for i in range(len(mass)):
    b1 = round((-(summ_x * (summ_y/n)) + (summ_xy)) / (summ_x*(-summ_x/n) +
(summ_x2)), 3)
    mass[i].append(b1)
    b0 = round(summ_y / n - summ_x / n * b1, 3)
    mass[i].append(b0)
    dy = round(b0 + b1 * mass[i][1], 3)
    mass[i].append(dy)

print()

for i in range(len(mass)):
    print(f'для x = {mass[i][1]} и y = {mass[i][2]}')
    print('b0 =', mass[i][-3])
    print('b1 =', mass[i][-2])
    print('dy =',mass[i][-1])
    print()
```

Листинг В.5 – Расчёт Q, Ecko, r

```
Q = 0
for i in mass:
    Q+= i[-1]
print('Q =', Q)
summ_y2 = 0
for i in mass:
    summ_y2 += i[2] **2

summ_for_Ecko = 0
for i in range(len(mass)):
    summ_for_Ecko += (round((mass[i][-1]-mass[i][2])**2, 3))
print('summ_for_Ecko =',summ_for_Ecko)
#%%
m = 1
Ecko = round(summ_for_Ecko/(n-m-1), 3)
print("Ecko =", Ecko)
Ect = round(math.sqrt(Ecko), 3)
print('Ect = ', Ect)

r = (summ_xy - (summ_x * summ_y)/n) / (math.sqrt (summ_x2-(summ_x**2)/n) *
math.sqrt(summ_y2 - (summ_y**2) /10))
r = math.sqrt(r**2)
print(f"r = {r}")
```

Листинг В.6 – Вывод финального графа

```
plt.close()
axis.clear()
fig = plt.figure(figsize=(10, 7))
axis = fig.add_subplot()
axis.grid()

for i in mass:
    if i[4] == 0:
        axis.scatter(i[1], i[2], s=50, c='g')
    else:
        axis.scatter(i[1], i[2], s=50, c='r')
    axis.scatter(i[1], i[-3], s=40, c='black')
plt.xlim(0,25)
plt.ylim(0,2500)
plt.show()
```

Приложение Г

Листинг кода байесовского классификатора

Листинг Г.1 – Используемые библиотеки

```
import math
```

Листинг Г.2 – Начальные данные и байесовский классификатор

```
data_nature = [
        [10, 50, "гусеница"],
        [20, 30, "божья коровка"],
        [25, 30, "божья коровка"],
        [20, 60, "гусеница"],
        [15, 70, "гусеница"],
        [40, 40, "божья коровка"],
        [30, 45, "божья коровка"],
        [20, 45, "гусеница"],
        [40, 30, "божья коровка"],
        [7, 35, "гусеница"]]

answers = []
for indx, mass in enumerate(data_nature):
    if data_nature[indx][-1] == "гусеница":
        answers.append(1)
    else:
        answers.append(0)
unique_answers = list(set(answers))
unique_answers = [1,0]
print(answers, unique_answers)
expectation = [] #матожидание
variance = [] #дисперсия
for l in range(len(data_nature[0])-1):
    datas_mass = []
    for x in range(len(list(set(answers)))):
        count = 0
        count_value = 0
        for i in range(len(data_nature)):
            if answers[i] == unique_answers[x]:
                count_value += data_nature[i][l]
                count += 1
        print(f'expectation for criterion {l}, for value {x} = 1/{count} *
({count_value}) = {(1/count) * count_value}')
        datas_mass.append((1/count) * count_value)
    expectation.append(datas_mass)
print('expectation =',expectation)
```

Продолжение Листинг Г.2

```
for l in range(len(data_nature[0])-1):
    datas_mass = []
    for x in range(len(list(set(answers)))):
        count = 0
        count_value = 0
        for i in range(len(data_nature)):
            if answers[i] == unique_answers[x]:
                count_value += (data_nature[i][l] - expectation[l][x]) ** 2
                count += 1
        print(f'variance for criterion {l}, for value {x} = (1/({count}-1)) *
({count_value}) = {(1/(count-1)) * count_value}')
        datas_mass.append((1/(count-1)) * count_value)
    variance.append(datas_mass)
print()
print('expectation =',expectation)
print('variance =',variance)

ver = []
for i in range(len(unique_answers)):
    count = 0
    for j in range(len(answers)):
        if unique_answers[i] == answers[j]:
            count += 1
    ver.append(count/len(answers))
print(ver)
```

Листинг Г.3 – Классификация для новых данных

```
input_data = [10, 50] # задаём данные
final_value = []
for i in range(len(unique_answers)):
    value = ver[i] *  (1 / (math.sqrt(2*math.pi* variance[i][0] *
variance[i][1]))) *\
    (((input_data[0] - expectation[i][0])**2)/(2 * variance[i][0] ** 2) -
((input_data[1] - expectation[i][1])**2)/(2 * variance[i][1] ** 2))
    final_value.append(value)
print(final_value)
maximum = [None, -111111]
for i, mass in enumerate(final_value):
    if mass > maximum[-1]:
        maximum = [i, mass]
if maximum[0] == 0:
    print('0 - гусеница')
else:
    print('1 - божья коровка')
```

Приложение Д

Листинг кода логистической регрессии

Листинг Д.1 – Используемые библиотеки

```
import random
import math
import matplotlib.pyplot as plt
```

Листинг Д.2 – Main функция

```
if __name__ == '__main__':
    count_value = 10
    logloss = []
    data, answers = generate_value(count_value)
    # classifier_weights = [random.random(), random.random(), random.random()]
    classifier_weights = [0.653569605, 0.39508814945259281, 1.153569605]
    print(f'Изначальные веса: {classifier_weights}')
    start_graph(data, answers, classifier_weights)
    program_answer = calculation_program_answer(data, answers,
classifier_weights)
    count_interation = 50
    for iteration in range(count_interation):
        print(f'\nИтерация {iteration} |', end='')
        program_answer = calculation_program_answer(data, answers,
classifier_weights)
        logloss.append(count_logloss(data, answers, classifier_weights,
program_answer))
        classifier_weights = count_classifier_weight(data, answers,
classifier_weights)

    stop_graph(data, answers, classifier_weights, program_answer)
    graph_logloss(logloss)

    print('\nФинальная таблица')
    for i in range(len(data)):
        print(
            f'№{i} data = {data[i]}| y = {answers[i]}| y_p =
{program_answer[i]}')
    print()

    predict(0.75, 0.25)
    predict(0.99, 0.99)
    predict(0.25, 0.25)
```

Листинг Д.3 – Функция генерация данных

```
def generate_value(count):  # генерируем данные
    mass_data = []
    mass_answers = []
    for i in range(count):
        x = random.randint(0, 100)
        y = random.randint(0, 100)
        if x + y >= 100:
            mass_data.append([x / 100, y / 100])
            mass_answers.append(1)
        else:
            mass_data.append([x / 100, y / 100])
            mass_answers.append(0)
    print(mass_data, mass_answers)
    return mass_data, mass_answers
```

Листинг Д.4 – Функция начальный граф

```
def start_graph(mass_data, mass_answers, weights):
    mass_color = ['read', 'blue']
    fig = plt.figure(figsize=(10, 7))
    axis = fig.add_subplot()
    for i in range(len(mass_data)):
        axis.scatter(mass_data[i][0], mass_data[i][1],
color=f'{mass_color[mass_answers[i]][0]}')
        sigmoid = 1 / (1 + math.e ** -(mass_data[i][0] * weights[0] + weights[1]
* mass_data[i][1]))
        plt.scatter(mass_data[i][0], sigmoid, s=10, color='green')
    plt.title('Начальный граф распеределния')
    plt.xlabel('Количество баллов за 3 предмета')
    plt.ylabel('Распределение')
    plt.savefig('static//start_graph.png')
```

Листинг Д.5 – Функция расчёта logloss

```
def count_logloss(data, answers, classifier_weights, program_answer):
    logloss = 0
    for i in range(len(data)):
        logloss += math.log(program_answer[i]) * answers[i] + (1 - answers[i]) *
math.log(1 - program_answer[i] + 1e-30)
        # x = f"{round(answers[i], 2)} * log({round(program_answer[i], 2)}) + (1
- {round(answers[i], 2)}) + log(1 - {round(program_answer[i], 2)})"
        # print(f"({x.replace('.', ',')})")
    logloss = (-logloss) / len(data)
    print(f'logloss = {logloss}')
    # exit()
    return logloss
```

Листинг Д.6 – Функция тренировки logloss

```
def count_classifier_weight(data, answers, classifier_weights):
    for i in range(len(data)):
        logit = classifier_weights[0] + data[i][0] * classifier_weights[1] +
data[i][1] * classifier_weights[2]
        sigmoid = 1 / (1 + math.e ** (-logit))
        classifier_weights[0] -= -(answers[i] - sigmoid)
        classifier_weights[1] -= -(answers[i] - sigmoid) * data[i][0]
        classifier_weights[2] -= -(answers[i] - sigmoid) * data[i][1]
    return classifier_weights
```

Приложение Е

Листинг кода PAM

Листинг Е.1 – Используемые библиотеки

```
import matplotlib.pyplot as plt
import math
import random
```

Листинг Е.2 – Main функция

```
if __name__ == '__main__':
    count_point = 20
    data = generate_data(count_point)
    start_graph(data)
    elbow_method(data)
    centroid(data)
```

Листинг Е.3 – Функция генерации данных

```
def generate_data(count_point):
    mass_data = []
    for i in range(count_point):
        mass_data.append([random.randrange(0, 10), random.randrange(0, 10)])
    print(mass_data, "\n")
    return mass_data
```

Листинг Е.4 – Функция выхода начального графа

```
def start_graph(mass):
    fig = plt.figure(figsize=(10, 7))
    axis = fig.add_subplot()
    for i in mass:
        axis.scatter(i[0], i[1], color='black')
    plt.xlim([-1, 11])
    plt.ylim([-1, 11])
    plt.title('График изначальных данных')
    plt.xlabel('Ось X')
    plt.ylabel('Ось Y')
    plt.savefig('static//start_graph.png')
    fig.clear()
    plt.close(fig)
```

Листинг Е.5 – Функция для метода локтя

```
def elbow_method(mass):
    color = ['#000000', '#808080', '#FF00FF', '#32CD32',
             '#FFFF00', '#808000', '#00FF00', '#008000', '#0000FF']
    minimum = 1
    maximum = 5
    mass_elbow_length = []
    start_point = random.randint(0, len(mass) - 1)
    for k in range(minimum, maximum + 1):
        central_points = [random.randint(0, len(mass)-1)]  # начальная точка
        cluster_center_definitions(k, central_points, mass)  # опеределения
центра кластеров
        graph_for_elbow_method(mass, central_points, k)
        mass_to_clusters = []
        distribution_of_points_by_clusters(mass_to_clusters, mass,
central_points)  # распределение точек по кластерам
        redefining_cluster_centers(central_points, mass, mass_to_clusters)  #
переопределение центров
        graph_for_elbow_method_cluster(mass, central_points, k, color,
mass_to_clusters)
        elbow_length = calculation_of_long_elbows(central_points, mass,
mass_to_clusters)

        mass_elbow_length.append(elbow_length)
        graph_elbow_long(mass_elbow_length)
    for index, i in enumerate(mass_elbow_length):
        print(f'Если количество элементов = {index + 1}, то\n'
              f'Евклидово расстояние по правилу локтя = {i}\n')
```

Листинг Е.6 – Функция создания новой центроиды

```
def cluster_center_definitions(k, central_points, mass):  # опеределения центра
кластеров
    if k != 1:  # для метода локтя
        for n in range(k - 1):
            minimum = [0, 0]            for i, element in enumerate(mass):
                if i not in central_points:
                    long = 0
                    for j in central_points:
                        long += math.sqrt((element[0] - mass[j][0]) ** 2 +
(element[1] - mass[j][1]) ** 2)
                    if long > minimum[0]:
                        minimum = [long, i]
central_points.append(minimum[1])
```

Листинг Е.7 – Функция распределение точек по кластерам

```
def distribution_of_points_by_clusters(mass_to_clusters, mass, central_points):
# распределение точек по кластерам
    for i, element in enumerate(mass):
        minimum = [1000000, 0]
        if i not in central_points:
            for index, j in enumerate(central_points):
                long = math.sqrt((element[0] - mass[j][0]) ** 2 + (element[1] -
mass[j][1]) ** 2)
                if long < minimum[0]:
                    minimum = [long, index]
            mass_to_clusters.append(minimum[1])
        else:
            for index, j in enumerate(central_points):
                if element[0] == mass[j][0] and element[1] == mass[j][1]:
                    mass_to_clusters.append(index)
```

Листинг Е.8 – Функция переопределения медоид

```
def redefining_cluster_centers(central_points, mass, mass_to_clusters):  #
переопределение центров
    for class_name in range(len(central_points)):
        mass_for_relocation = []
        for index, j in enumerate(mass):
            if mass_to_clusters[index] == class_name:
                mass_for_relocation.append(index)
        minimum = [1000000, 0]
        for index_i, i in enumerate(mass_for_relocation):
            long = 0
            for index_j, j in enumerate(mass_for_relocation):
                if index_j != index_i:
                    long += math.sqrt((mass[i][0] - mass[j][0]) ** 2 +
(mass[i][1] - mass[j][1]) ** 2)
            if long < minimum[0]:
                minimum = [long, i]
        central_points[class_name] = minimum[1]
```

Листинг Е.9 – Функция вывод графика метод локтя

```
def graph_elbow_long(mass_elbow_length):
    fig = plt.figure(figsize=(10, 7))
    mass_k = []
    for i in range(len(mass_elbow_length)):
        mass_k.append(i + 1)
    print(mass_k)
    print(mass_elbow_length)
    plt.bar(mass_k, mass_elbow_length)
    plt.title('График правила локтя')
    plt.ylabel('Евклидово расстояние')
    plt.xlabel('Количество вершин')
    plt.savefig(f'static./elbow_long.png')
    fig.clear()
    plt.close(fig)
```

Листинг Е.10 – Функция финальный алгоритм pam

```
def centroid(mass):
    color = ['#000000', '#808080', '#FF00FF', '#32CD32',
             '#FFFF00', '#808000', '#00FF00', '#008000', '#0000FF']
    start_point = random.randint(0, len(mass) - 1)
    central_points = [start_point]  # начальная точка
    k = int(input('Введите число кластеров: '))
    cluster_center_definitions(k, central_points, mass)
    graph_for_elbow_method(mass, central_points, k)
    mass_to_clusters = []
    distribution_of_points_by_clusters(mass_to_clusters, mass, central_points)
    for a in range(3):
        for class_name in range(len(central_points)):
            mass_for_relocation = []
            for index, j in enumerate(mass):
                if mass_to_clusters[index] == class_name:
                    mass_for_relocation.append(index)
            minimum = [1000000, 0]
            for index_i, i in enumerate(mass_for_relocation):
                long = 0
                for index_j, j in enumerate(mass_for_relocation):
                    if index_j != index_i:
                        long += math.sqrt((mass[i][0] - mass[j][0]) ** 2 +
(mass[i][1] - mass[j][1]) ** 2)
                if long < minimum[0]:
                    minimum = [long, i]
            central_points[class_name] = minimum[1]
        mass_to_clusters = []
        for i, element in enumerate(mass):
            minimum = [1000000, 0]
            if i not in central_points:
                for index, j in enumerate(central_points):
                    long = math.sqrt((element[0] - mass[j][0]) ** 2 +
(element[1] - mass[j][1]) ** 2)
                    if long < minimum[0]:
                        minimum = [long, index]
                mass_to_clusters.append(minimum[1])
            else:
                for index, j in enumerate(central_points):
                    if element[0] == mass[j][0] and element[1] == mass[j][1]:
                        mass_to_clusters.append(index)
    graph_centroid(mass, central_points, k, color, mass_to_clusters)
```

Листинг Е.11 – Функция финальный граф

```
def graph_centroid(mass, central_points, k, color, mass_to_clusters):
    fig = plt.figure(figsize=(10, 7))
    axis = fig.add_subplot()
    for index, i in enumerate(mass):
        axis.scatter(i[0], i[1], color=color[mass_to_clusters[index]], s=100)
    for i in central_points:
        axis.scatter(mass[i][0], mass[i][1], color='red', s=25)
    plt.title(f'График для {k} кластеров')
    plt.xlabel('Ось X')
    plt.ylabel('Ось Y')
    plt.xlim([-1, 11])
    plt.ylim([-1, 11])
    plt.savefig(f'static//pam_centroid.png')
    fig.clear()
    plt.close(fig)
```

Приложение Ж

Листинг кода CURE

Листинг Ж.1 – Используемые библиотеки

```
import matplotlib.pyplot as plt
import math
import random
```

Листинг Ж.2 – Main функция

```
if __name__ == '__main__':
    count_point = 70
    data = generate_data(count_point)
    start_graph(data)
    massive_path = hierarchy(data)
    cure(data, massive_path)
```

Листинг Ж.3 – Функция генерации данных

```
def generate_data(count):
    mass_data = []
    for i in range(count // 4):
        mass_data.append([round(random.uniform(0, 2), 2),
round(random.uniform(0, 2), 2)])

    for i in range(count // 4):
        mass_data.append([round(random.uniform(0, 2), 2),
round(random.uniform(5, 8), 2)])

    for i in range(count // 4):
        mass_data.append([round(random.uniform(5, 7), 2),
round(random.uniform(5, 10), 2)])

    for i in range(count // 4):
        mass_data.append([round(random.uniform(9, 10), 2),
round(random.uniform(5, 10), 2)])

    for i in range(count // 4):
        mass_data.append([round(random.uniform(2, 3), 2),
round(random.uniform(7, 8), 2)])

    for i in range(count // 4):
        mass_data.append([round(random.uniform(3, 3.5), 2),
round(random.uniform(7, 8), 2)])

    for i in range(count // 4):
        mass_data.append([round(random.uniform(3, 3.5), 2),
round(random.uniform(5, 7), 2)])
    print(mass_data)
    return mass_data
```

Листинг Ж.4 – Функция вывода начальных данных

```
def start_graph(mass):
    fig = plt.figure(figsize=(10, 7))
    axis = fig.add_subplot()
    for i in mass:
        axis.scatter(i[0], i[1], color='black')
    plt.title('График изначальных данных')
    plt.xlabel('Ось X')
    plt.ylabel('Ось Y')
    plt.savefig('static//start_graph.png')
    fig.clear()
    plt.close(fig)
```

Листинг Ж.5 – Функция иерархическая кластеризация

```
def hierarchy(data):
    massive_path = []
    for i in range(len(data)):
        massive_path.append(i)
    count_group = 6
    while len(set(massive_path)) > count_group:
        minimum_count = 1000000
        minimum_index = 0
        for i, i_i in enumerate(data):
            for j, j_j in enumerate(data):
                if i != j and massive_path[i] != massive_path[j]:
                    long = math.sqrt((data[i][0] - data[j][0]) ** 2 +
(data[i][1] - data[j][1]) ** 2)
                    if long < minimum_count:
                        minimum_count = long
                        minimum_index = [i, j]
        if massive_path[minimum_index[0]] > massive_path[minimum_index[1]]:
            massive_path[minimum_index[0]] = massive_path[minimum_index[1]]
        else:
            massive_path[minimum_index[1]] = massive_path[minimum_index[0]]
        print(minimum_index)
        # minimum_count = math.sqrt((data[minimum_index[0]][0] -
data[minimum_index[1]][0]) ** 2 + (data[minimum_index[0]][1] -
data[minimum_index[1]][1]) ** 2)
        print(f' {minimum_count} = math.sqrt(({data[minimum_index[0]][0]} -
{data[minimum_index[1]][0]}) ** 2 + ({data[minimum_index[0]][1]} -
{data[minimum_index[1]][1]}) ** 2)')
    print()
    print('massive_path')
    print('massive_path')
    # a = [0, 1, 2, 3, 4, 5, 6, 7, 8]
    for i in range(count_group):
        if i not in massive_path:
            max = 100
            for j in massive_path:
                if j > i and j < max:
                    max = j
            for j in range(len(massive_path)):
                if massive_path[j] == max:
                    massive_path[j] = i
    graph_hierarchy(data, massive_path)
    return massive_path
```

Листинг Ж.6 – Функция график иерархическческой кластеризации

```
def graph_hierarchy(data, massive_path_):
    fig = plt.figure(figsize=(10, 7))
    axis = fig.add_subplot()
    colors = ['#000000', '#808080', '#FF00FF', '#32CD32', '#800000',
              '#FFFF00', '#808000', '#00FF00', '#008000', '#0000FF']
    for i in range(len(data)):
        axis.scatter(data[i][0], data[i][1], color=colors[massive_path_[i]])

    plt.title('График данных')
    plt.xlabel('Ось X')
    plt.ylabel('Ось Y')
    plt.savefig('static//start_graph_hierarchy.png')
    fig.clear()
```

Листинг Ж.7 – Функция cure кластеризации

```
def central_point(central_cluster):
    central_cluster = []
    cental = list(set(massive_path))
    print(cental, 'central')
    for class_name in cental:
        mass_for_relocation = []
        for index, j in enumerate(data):
            if massive_path[index] == class_name:
                mass_for_relocation.append(index)
        minimum = [1000000, -1]
        for index_i, i in enumerate(mass_for_relocation):
            long = 0
            for index_j, j in enumerate(mass_for_relocation):
                if index_j != index_i:
                    long += math.sqrt((data[i][0] - data[j][0]) ** 2 +
(data[i][1] - data[j][1]) ** 2)
            if long < minimum[0]:
                minimum = [long, i]
        central_cluster.append(minimum[1])
    print(central_cluster)
    graph_c(data, massive_path, central_cluster)

    mass_cluster_contour = []
    for class_name in cental:
        mass_for_relocation = []
        for index, j in enumerate(data):
            if massive_path[index] == class_name:
                mass_for_relocation.append(index)
        mass_four_point = [central_cluster[class_name]]
        for i in range(4):
            index_max = -1
            max_long = 0
            for index, j in enumerate(mass_for_relocation):
                long = 0
                for k in mass_four_point:
                    if j not in mass_four_point:
                        long += (data[k][0] - data[j][0]) ** 2 + (data[k][1] -
data[j][1]) ** 2
                if long > max_long:
                    max_long = long
                    index_max = j
            mass_four_point.append(index_max)
            if i == 0:
                mass_four_point.pop(0)
```

Продолжение – Листинг Ж.7

```
        print(mass_four_point)
        mass_cluster_contour.append(mass_four_point)

    mass_len_a = []
    for ii, i in enumerate(central_cluster):
        len_a = 0
        for j in mass_cluster_contour[ii]:
            len_a += math.sqrt((data[i][0] - data[j][0]) ** 2 + (data[i][1] -
data[j][1]) ** 2)
        print(f"({round(len_a,2)} / 4) * 0.75) = {round((len_a / 4) *
0.75,2)})")
        mass_len_a.append((len_a / 4) * 0.50)
    print(mass_len_a, 'lenn_a')
    mass_to_splete = []
    for k, kk in enumerate(mass_cluster_contour):
        for i, ii in enumerate(kk):
            for jj, j in enumerate(data):
                if ii != jj and massive_path[ii] != massive_path[jj] and \
                        mass_len_a[k] > math.sqrt((data[ii][0] - data[jj][0]) **
2 + (data[ii][1] - data[jj][1]) ** 2):
                    print(ii, jj, massive_path[ii], massive_path[jj], data[ii],
data[jj])
                    print(math.sqrt((data[ii][0] - data[jj][0]) ** 2 +
(data[ii][1] - data[jj][1]) ** 2))
                    mass_to_splete.append([massive_path[ii], massive_path[jj]])

    count_merge_ = 1
    print('-----------')
    print(mass_to_splete)
    print(massive_path)
    if mass_to_splete != 0:
        for k in range(len(mass_to_splete)):
            for i in range(len(massive_path)):
                if massive_path[i] == mass_to_splete[k][1]:
                    massive_path[i] = mass_to_splete[k][0]
    print(massive_path)

    fig = plt.figure(figsize=(10, 7))
    axis = fig.add_subplot()
    colors = ['#000000', '#808080', '#FF00FF', '#32CD32', '#800000',
              '#FFFF00', '#808000', '#00FF00', '#008000', '#0000FF']
    for i in range(len(data)):
        axis.scatter(data[i][0], data[i][1], color=colors[massive_path[i]])
    plt.title('График данных')
    plt.xlabel('Ось X')
    plt.ylabel('Ось Y')
    plt.savefig('static//start_graph_c_end.png')
    fig.clear()
```

Приложение 3

Листинг кода ID3

Листинг 3.1 – Используемые библиотеки

```
import math
```

Листинг 3.2 – Main функция

```
if __name__ == '__main__':
    massive_tree = []
    massive_entropy = []

    x, y = generate_data()
    count_data, unique_data, count_answer, unique_answer =
calculation_of_unique_values(x, y)
    massive_entropy.append(calculation_initial_entropy(count_answer, y, ))
    calculation_first_variable(count_data, unique_data, count_answer,
unique_answer, massive_entropy, massive_tree)
    calculate_next_entropy(massive_tree, massive_entropy, count_data,
unique_data, count_answer, unique_answer, x, y)
    print()
    print(f'Дерво:')
    for i in massive_tree:
        print(i)
    print()
    print(f'Ентропия на всех участках дерева:')
    for i in massive_entropy:
        print(i)
```

Листинг 3.3 – Функция генерации данных

```
def generate_data():
    data = [[1, 0],
            [1, 0],
            [1, 0],
            [0, 1],
            [0, 0],
            [0, 0],
            [0, 0],
            [1, 1]]
    answers = [1, 1, 1, 1, 0, 0, 0, 0]
    return data, answers
```

Листинг 3.4 – Функция расчёта все уникальне критерии и их количества

```
def calculation_of_unique_values(x, y):
    # все уникальне критерии и их количества
    unique_data = []
    for j in range(len(x[0])):
        mass_to_unique = []
        for i in range(len(x)):
            if x[i][j] not in mass_to_unique:
                mass_to_unique.append(x[i][j])
        mass_to_unique.sort()
        unique_data.append(mass_to_unique)
    print('Уникальные значения у критериев:', unique_data)
    print()
    count_data = []
    for i in range(len(unique_data)):
        mass_to_unique = []
        for j in range(len(unique_data[i])):
            count = 0
            for k in range(len(x)):
                if x[k][i] == unique_data[i][j]:
                    count += 1
            mass_to_unique.append(count)
        count_data.append(mass_to_unique)
    print('Количество значений у критериев:', count_data)
    print()
    # все уникальные ответы и их количества
    unique_answer = []
    for i in y:
        if i not in unique_answer:
            unique_answer.append(i)
    unique_answer.sort()
    print('Уникальные классы:', unique_answer)
    print()
    count_answer = []
    for i in unique_answer:
        count = 0
        for j in y:
            if j == i:
                count += 1
        count_answer.append(count)
    print('Количество для уникальных классов:', count_answer)
    print()
    return count_data, unique_data, count_answer, unique_answer
```

Листинг 3.4 – Функция расчёта начальной энтропии

```
def calculation_initial_entropy(count_answer, y):
    # рачсёт начальной энтропии
    start_entropy = 0
    for i in count_answer:
        start_entropy -= (i / len(y)) * math.log2(i / len(y))
    print('Начальная энтропия =', start_entropy)
    print()
    return start_entropy
```

Листинг 3.5 – Функция расчёта энтропии и выбора 1 критерия

```
def calculation_first_variable(count_data, unique_data, count_answer,
unique_answer, massive_entropy, massive_tree):
    search_entropy = count_elements_calculate_entropy(unique_data,
unique_answer)
    entropy_calculation_for_all_initial_outcomes(search_entropy,
massive_entropy, massive_tree)
```

Листинг 3.6 – Функция расчёта количество элементов для подсчёта энтропии

```
def count_elements_calculate_entropy(unique_data, unique_answer):
    # Количество элементов для подсчёта энтропии
    search_entropy = []
    for criterion in range(len(unique_data)):
        count_criterion = []
        for unique_criterion in range(len(unique_data[criterion])):
            mass_answer_to_elem = [0] * len(unique_data)
            for h in range(len(unique_answer)):
                for i in range(len(x)):
                    if x[i][criterion] ==
unique_data[criterion][unique_criterion] and y[i] == unique_answer[h]:
                        mass_answer_to_elem[h] += 1
            count_criterion.append(mass_answer_to_elem)
        search_entropy.append(count_criterion)
    print('Количество элементов для подсчёта энтропии:', search_entropy)
    print()
    return search_entropy
```

Листинг 3.7 – Функция расчёта энтропии

```
def entropy_calculation_for_all_initial_outcomes(search_entropy,
massive_entropy, massive_tree):
    # расчёт энтропии для всех начальных исходов
    mass_entropy = []
    for i in range(len(search_entropy)):
        entropy_critery = []
        for j in range(len(search_entropy[i])):
            entropy = 0
            for k in range(len(search_entropy[i][j])):
                if search_entropy[i][j][k] / count_data[i][j] > 0:
                    entropy += -(search_entropy[i][j][k] / count_data[i][j]) * \
                               math.log2(search_entropy[i][j][k] /
count_data[i][j])
                else:
                    entropy += 0
            entropy_critery.append(entropy)
        mass_entropy.append(entropy_critery)
    print('mass_entropy', mass_entropy)
    IG = [0] * len(unique_data)
    # print('IG', IG)
    for i in range(len(IG)):
        IG[i] += massive_entropy[-1]
        ig = 0
        for j in range(len(count_data[i])):
            ig += (count_data[i][j] / (len(x))) * mass_entropy[i][j]
        IG[i] -= ig
    print('IG', IG)
    maximum = [0, 0]
```

Продолжение – Листинг 3.7

```
    for i, ii in enumerate(IG):
        if ii > maximum[1]:
            maximum = [i, ii]
    print(maximum)
    print()
    massive_tree.append([maximum[0]])
    massive_entropy.append(mass_entropy[maximum[0]])
```

Листинг 3.8 – Функция расчёта этропии и критериев

```
def calculate_next_entropy(massive_tree, massive_entropy, count_data,
unique_data, count_answer, unique_answer, x, y):
    new_entropy = []
    for i in unique_answer:
        mass_1 = []
        for j in unique_data[0]:
            mass_2 = []
            for k in unique_data[1]:
                count = 0
                for d, dd in enumerate(x):
                    if x[d][0] == j and x[d][1] == k and y[d] == i:
                        count += 1
                mass_2.append(count)
            mass_1.append(mass_2)
        new_entropy.append(mass_1)
    print(new_entropy)
    summ_ent = calculation_new_entropy(new_entropy)
    mass_entropy = []
    for i, ii in enumerate(new_entropy):
        # расчёт энтропии для всех начальных исходов
        entropy_critery = []
        for j in range(len(ii)):
            entropy = 0
            for k in range(len(ii[j])):
                if ii[j][k] / summ_ent[i][j] > 0:
                    entropy += -(ii[j][k] / summ_ent[i][j]) * \
                               math.log2(ii[j][k] / summ_ent[i][j])
                else:
                    entropy += 0
            entropy_critery.append(entropy)
        mass_entropy.append(entropy_critery)
    print('mass_entropy', mass_entropy)
    IG = [0] * len(unique_data)
    # print('IG', IG)
    for i in range(len(IG)):
        IG[i] += massive_entropy[-1][0]
        ig = 0
        for j in range(len(count_data[i])):
            ig += (count_data[i][j] / (len(x))) * mass_entropy[i][j]
        IG[i] -= ig
    print('IG', IG)
    massive_entropy.append(mass_entropy)
    massive_tree.append([1, 1])
    print()
    return new_entropy
```