СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 2](#_Toc135901096)

[1. AССОЦИАТИВНЫЕ ПРАВИЛА 3](#_Toc135901097)

[1.1 Постановка задачи 3](#_Toc135901098)

[1.2 Описание алгоритма 3](#_Toc135901099)

[1.3 Описание предметной области 5](#_Toc135901100)

[1.4 Ассоциотивные правила в программе deductor 6](#_Toc135901101)

[1.5 Расчёт поддержки 7](#_Toc135901102)

[1.6 Расчёт достоверности 8](#_Toc135901103)

[1.7 Расчёт лифта 8](#_Toc135901104)

[1.8 Программная реализация 8](#_Toc135901105)

[2. К-MEANS 10](#_Toc135901106)

[2.1 Постановка задачи 10](#_Toc135901107)

[2.2 Описание алгоритма 10](#_Toc135901108)

[2.3 Ручной расчёт 11](#_Toc135901109)

[2.4 Программная реализация 13](#_Toc135901110)

[3. ЛИНЕЙНАЯ РЕГРЕСИЯ 15](#_Toc135901111)

[3.1 Постановка задачи 15](#_Toc135901112)

[3.2 Описание предметной области 15](#_Toc135901113)

[3.3 Описание алгоритма 15](#_Toc135901114)

[3.4 Ручной расчёт 17](#_Toc135901115)

[3.5 Программная реализация 20](#_Toc135901116)

[4. БАЕСОВСКИЙ КЛАСИФИКАТОР 24](#_Toc135901117)

[4.1 Постановка задачи 24](#_Toc135901118)

[4.2 Описание предметной области 24](#_Toc135901119)

[4.3 Описание алгоритма 24](#_Toc135901120)

[4.4 Ручной расчёт 27](#_Toc135901121)

[4.5 Программная реализация 28](#_Toc135901122)

[ВЫВОД 31](#_Toc135901123)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 33](#_Toc135901124)

[ПРИЛОЖЕНИЯ 34](#_Toc135901125)

ВВЕДЕНИЕ

Вводим введение

1. AССОЦИАТИВНЫЕ 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 каr условие, так и следствие.

Например, для ассоциации 𝐴→𝐵 можно записать:

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

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:

**1.3 Описание предметной области**

Исходные данные из магазина автозапчастей представленные (Рисунок 1.1).

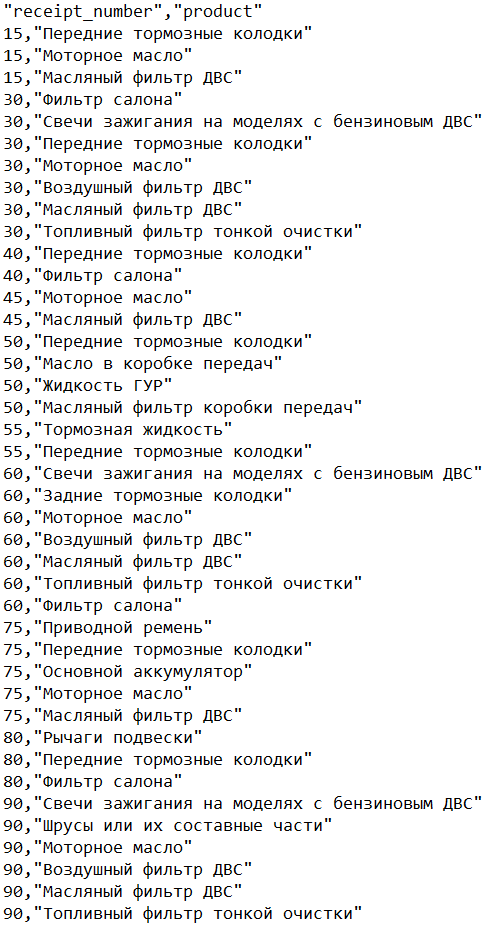


Рисунок 1.1 – Исходные данные

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).

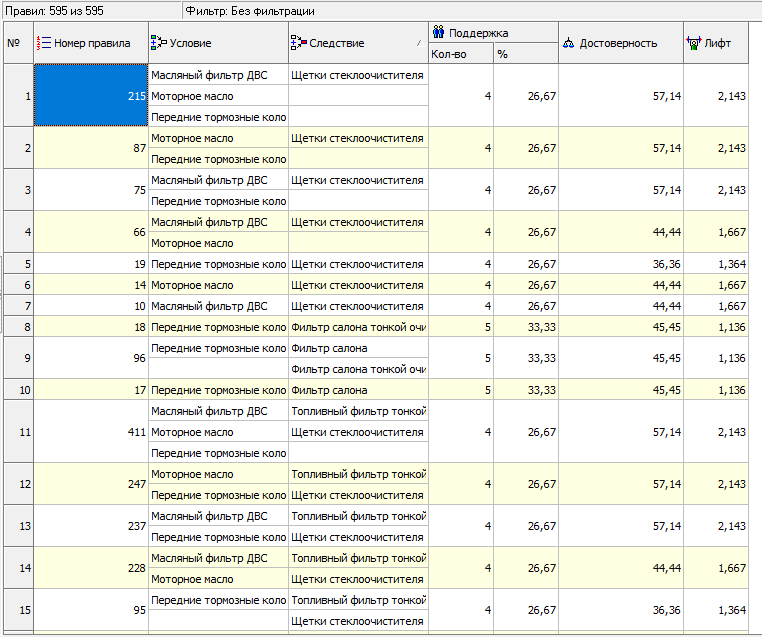


Рисунок 1.2 – Анализ данных в программе Deductor

* 1. 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:

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:

1.7 Расчёт лифта

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

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).

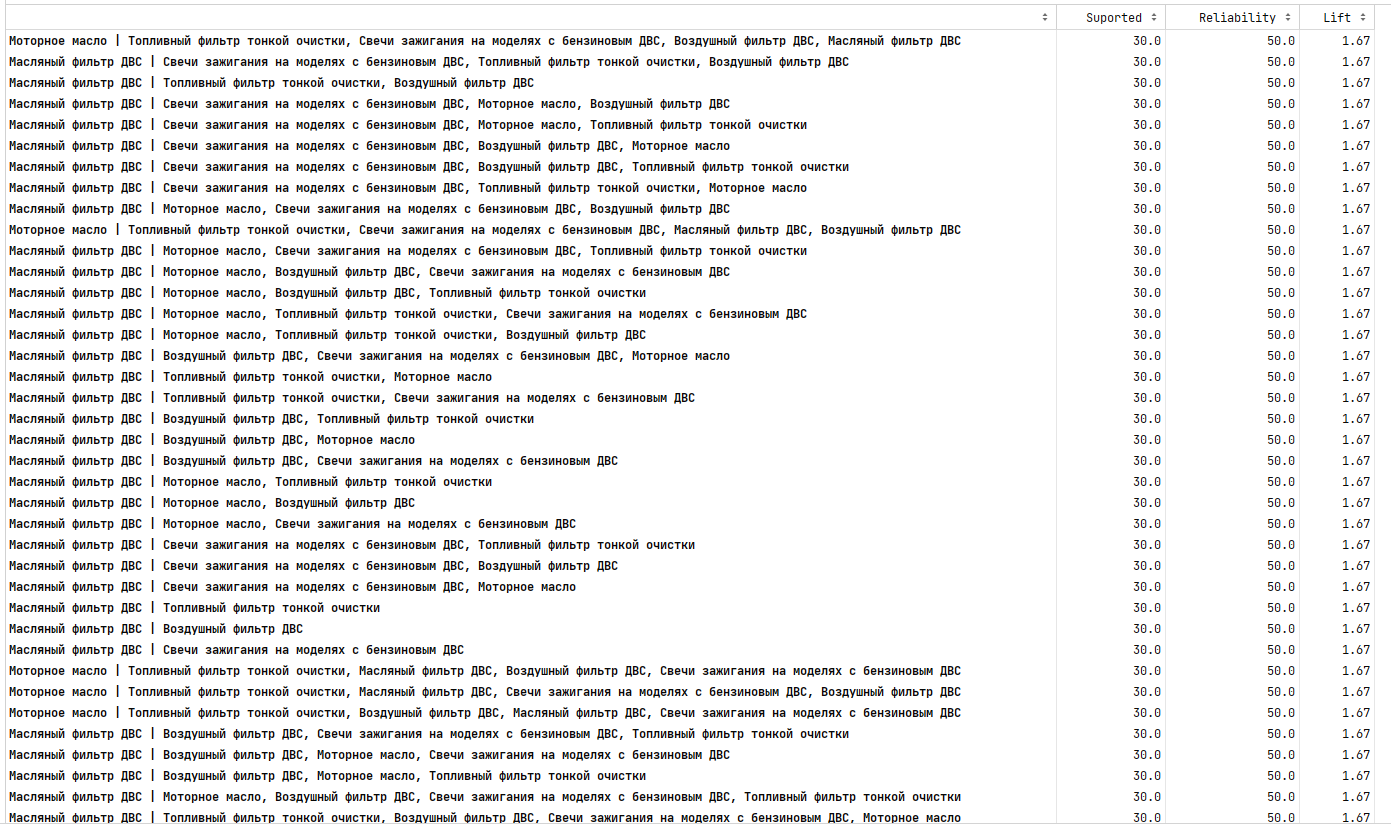


Рисунок 1.3 – Результат работы программы

2. К-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:

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

5) старый центр кластера смещается в его центроид.

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

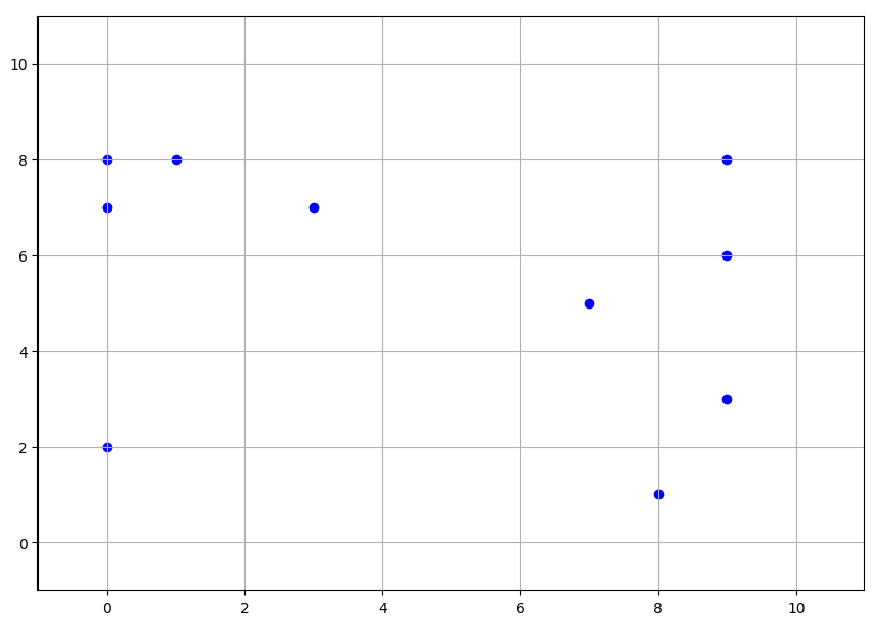


Рисунок 2.1 – Изначальные данные

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.

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 | Расстояние от | 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.

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).

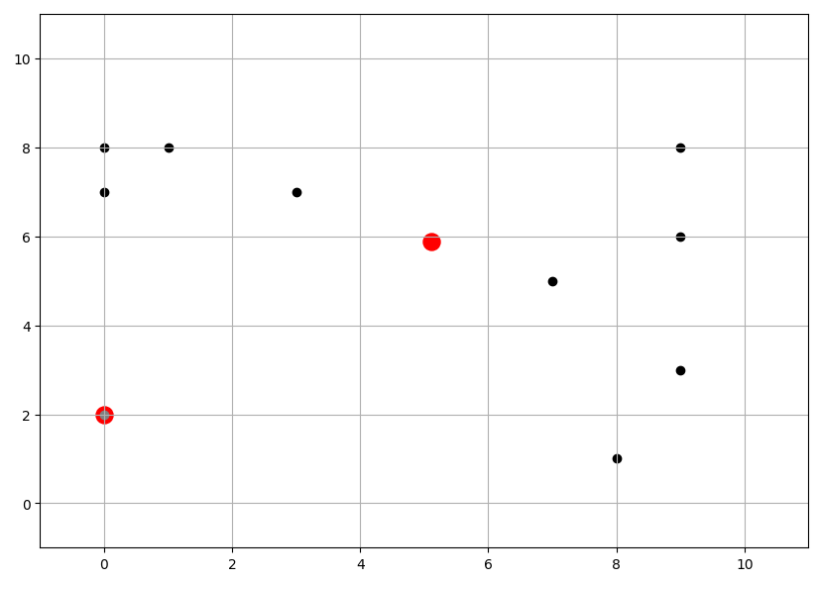


Рисунок 2.2 – Итоги первой итерации

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).

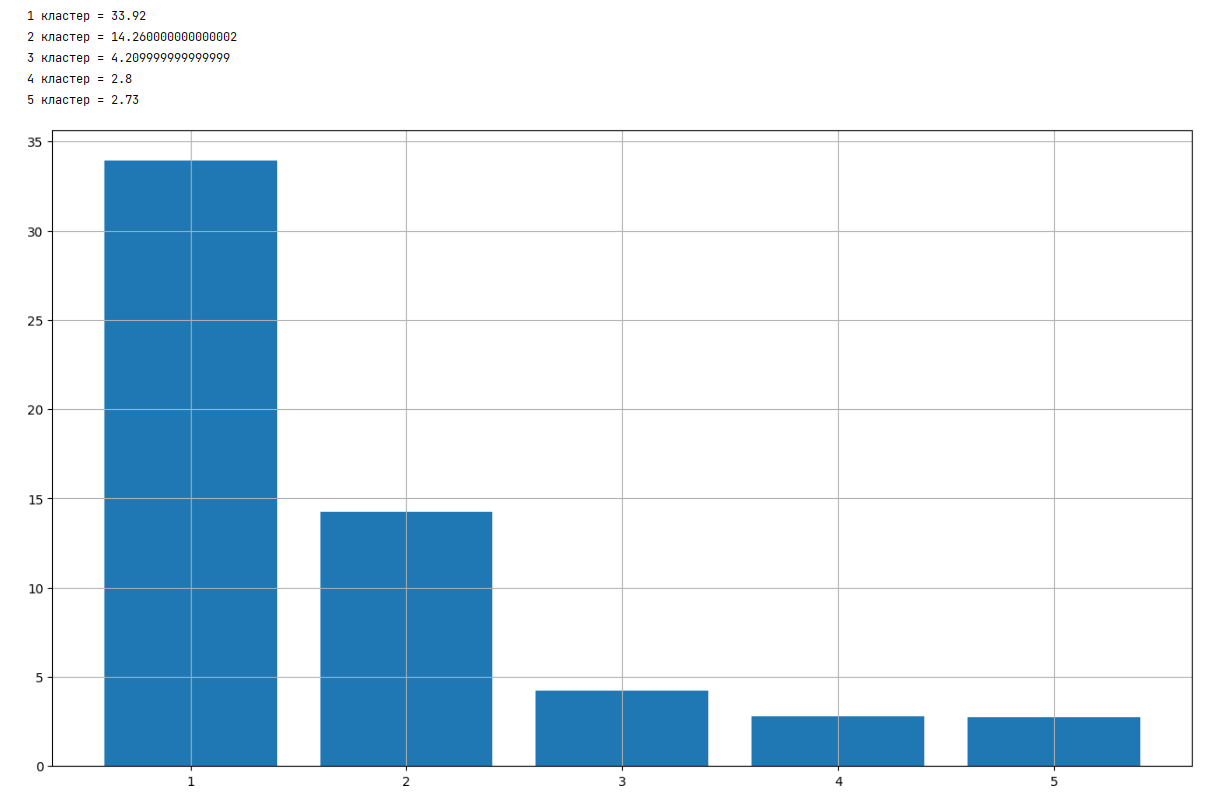


Рисунок 2.3 – Правило локтя для данных

Исходя из правила локтя количество кластеров должно быть 3.

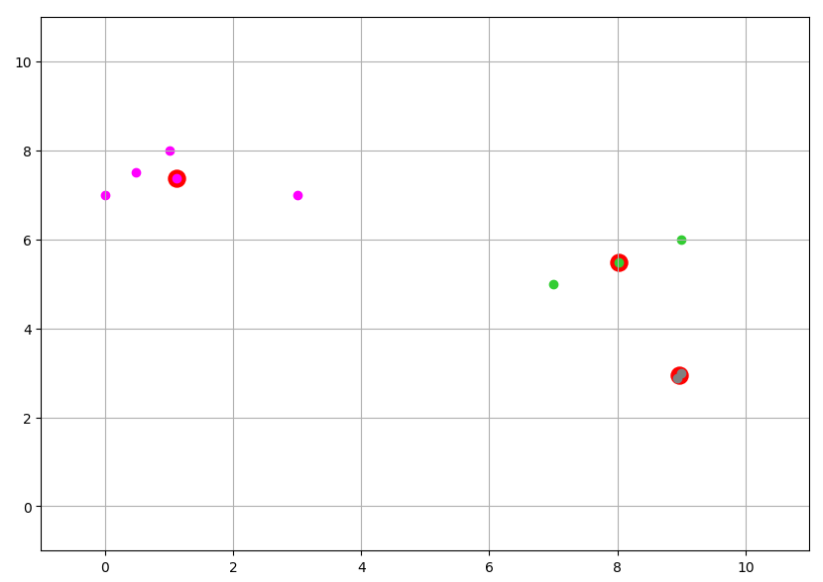


Рисунок 2.4 – Результат работы алгоритма k-means

1. 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, | Количество проданного картофеля, , 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 (), выбираемая таким образом, чтобы сумма квадратов расстояний от точек () до этой прямой, измеренных вертикально (то есть вдоль оси ), была минимальна, Уравнение, описывающее линию регрессии, называется уравнением регрессии :

где – оценка значения выходной переменной;

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

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

e-ошибка.

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

Можно найти значения и , которые минимизируют , путем дифференцирования уравнения (3.1) по и . Частные производные для уравнения (3.2) по 𝑏0 и 𝑏1 соответственно будут:

3.4 Ручной расчёт

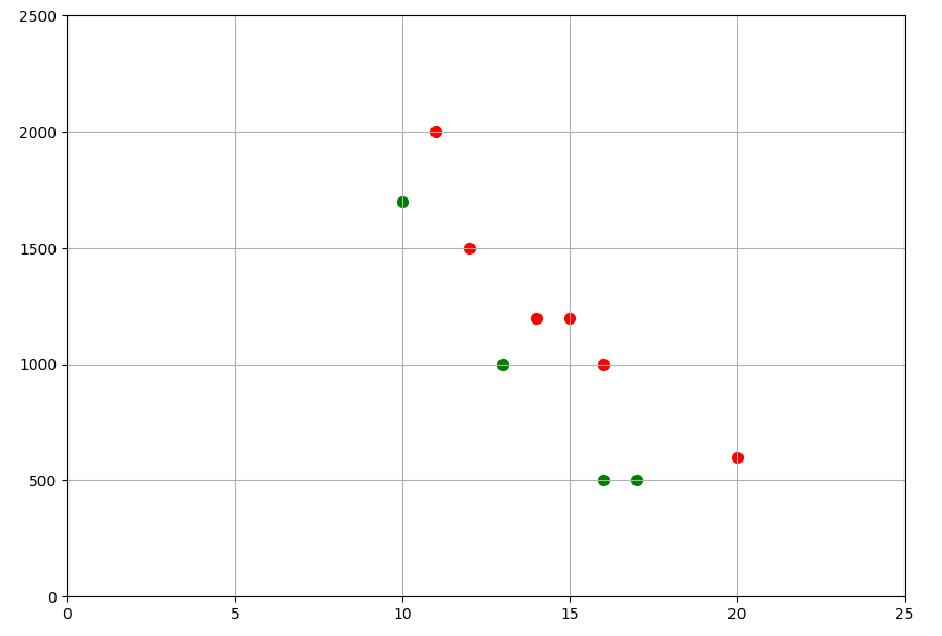


Рисунок 3.1 – Изначальные данные

Система нормальных уравнений.

Для наших данных система уравнений имеет вид

Решим систему уравнений выразим из формулы (3.4):

(1.5)

Подставим в формулу (3.6) выражение

Получим:

161280 – 2073,6 + 2156 + 149300

-2073,6 + 2156 = 149300 – 161280

82,4 = -11980

Подставим коэффициент в формулу (3.6) чтобы получить коэффициент

Получили коэффициенты регрессии:

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

Оценку значения получаем из выражения (1.6).

А также коэффициенты b0 и b1 можно найти другим способом. Они минимизируются , путем дифференцирования уравнения (3.1) по и . Можно использовать метод градиентного спуска или метод Ньютона.

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

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

можно рассматривать как меру изменчивости выходной переменной, объясняемую регрессией.

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

Найдем из формулы () значения:

= + + + + + + + + + = 593 989,28

= 74248,66

Найдем стандартную ошибку из формулы (3.7)

= = 272,4860730

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

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

– 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:

)

Найдем коэффициент корреляции

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

где и стандартные отклонения соответствующих переменных. Значение коэффициента корреляции всеrда расположено в диапазоне от до

Вычисление коэффициента корреляции по формуле (3.9)

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 возрастают, то и значения выходной переменной 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).

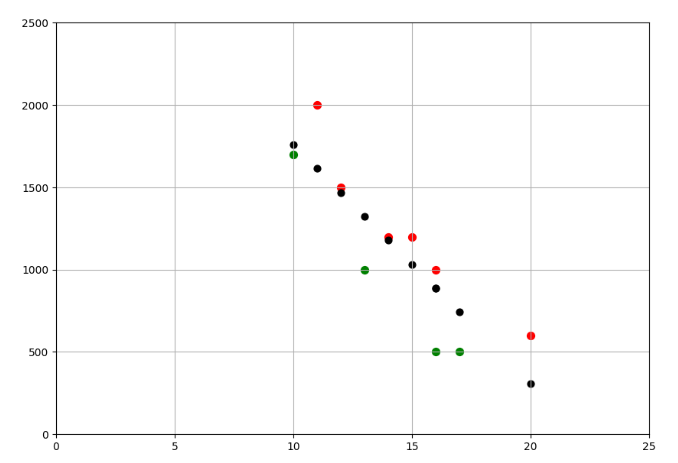


Рисунок 3.2 – График результат работы алгоритма линейной регрессии

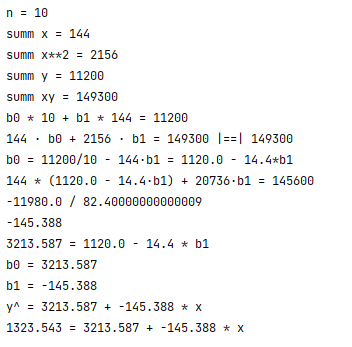


Рисунок 3.3 – Текстовый результат работы алгоритма линейной регрессии

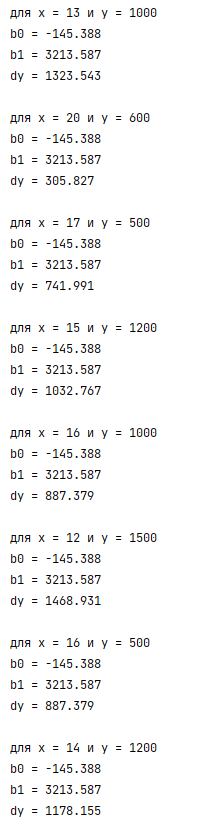


Рисунок 3.4 – Текстовый результат работы алгоритма линейной регрессии

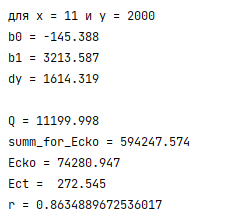


Рисунок 3.5 – Текстовый результат работы алгоритма линейной регрессии

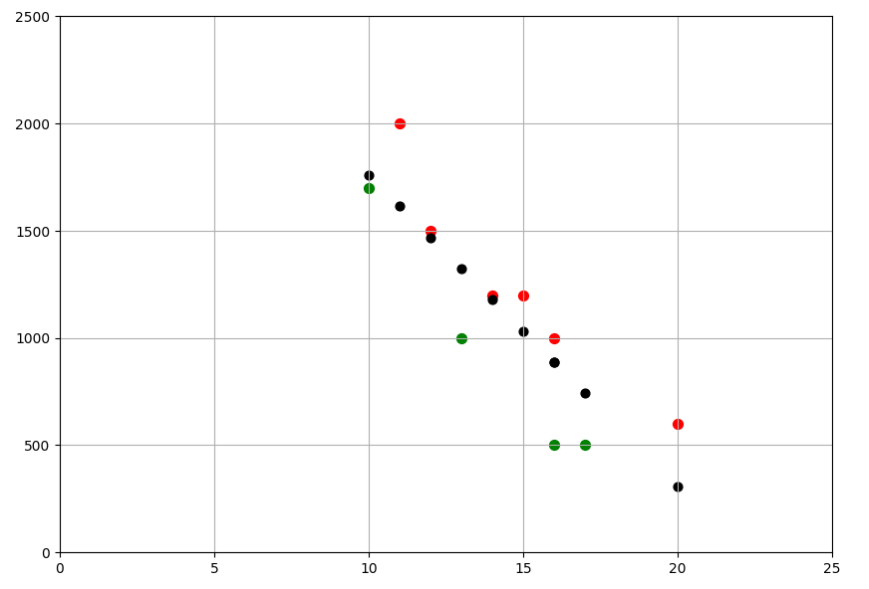


Рисунок 3.6 – Граф результата работы алгоритма линейной регрессии

4. БАЕСОВСКИЙ КЛАСИФИКАТОР

4.1 Постановка задачи

Придумать свою предметную область и реализовать алгоритм байесовской классификации.

4.2 Описание предметной области

Результатом собранных наблюдений явилась зависимость длины и ширины, от того гусеница это или божья коровка (Таблица. 3.1).

Таблица 3.1. 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:

Алгоритм работы простого байесовского классификатора содержит следующие шаги.

1. Пусть исходное множество данных 𝑆 содержит атрибуты 𝐴1, 𝐴2, ..., 𝐴𝑛 Тогда каждый объект или наблюдение 𝑋∈𝑆 будет представлено своим набором значений этих атрибутов где значение, которое принимает атрибут 𝐴𝑖 в данном наблюдении.

2. Предположим, что задано m классов ,,… 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 тогда и только тогда, когда выполняется условие для любых :.т.к, 𝑘≠𝑗.

По формуле Байеса:

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()=𝑃()=...=𝑃(), и, следовательно, мы должны выбрать максимальную вероятность 𝑃(|𝑋).

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

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

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) =, где – общее число наблюдений класса , в которых принимает значение , а – общее число наблюдении, относящихся к классу .

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

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

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

4.4 Ручной расчёт

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

Для критерия ширины:

Если класс гусеница, то:

Если класс божья коровка, то:

Для критерия длины:

Если класс гусеница, то:

Если класс божья коровка, то:

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 линейной регрессии (Рисунок 4.1 – Рисунок 4.3).

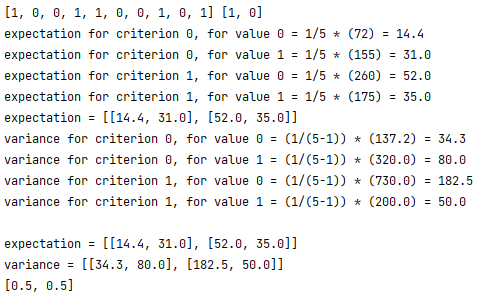


Рисунок 4.1 –результат работы байесовского классификатора



Рисунок 4.2 – Новые данные для обучения классификатора

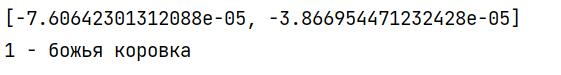


Рисунок 4.3 – Результат классификации

ВЫВОД

Информационная система, разработанная для web приложения по поиску авиабилетов, представляет собой комплекс программных средств, который обеспечивает автоматизированный процесс покупки билетов. Система должна соответствовать требованиям безопасности, обеспечивать быстрое и удобное использование, а также поддерживать управление данными о клиентах и транзакциях.

Для создания данной системы необходимо провести предварительный анализ требований к проектируемой системе. В процессе работы над проектом необходимо разработать диаграмму прецедентов для одного из классов или прецедентов проектируемой информационной системы. Важным шагом является уточнение требований, созданных на предыдущих этапах, на основании макета информационной системы.

Для проектирования функциональной модели информационной системы необходимо выбрать цель создания ИС, способ и средства создания, а также использовать нотацию IDEF0. На следующем этапе необходимо продолжить проектирование функциональной модели и провести моделирование двух уровней декомпозиции в нотации IDEF0, составив текстовое описание проектируемых модулей на уровнях декомпозиции.

Для выбранного функционального блока нижнего уровня декомпозиции необходимо выполнить декомпозицию в нотации DFD. Для более полного понимания и формализации заданной предметной области необходимо создать модель «сущность – связь» в нотации ERD.

Важным аспектом разработки информационной системы является создание диаграммы состояний проектируемой информационной системы с использованием соответствующей нотации языка UML. Наконец, необходимо закрепить имеющиеся знания о параметрах ИС и изучить методологию расчета требуемых параметров проектируемой информационной системы, а также приобрести навыки анализа и формализованного описания заданной предметной области.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сорокин А. Б. Лекции и методические материалы — РТУ МИРЭА, 2023.
2. Python documentation — URL: https://www.python.org/ (Дата обращения: 20.12.2022)
3. Matplotlib — Visualization with Python — URL: https://matplotlib.org/ (Дата обращения: 21.12.2022)
4. Информационные технологии, № 7, 2012. Теоретический и прикладной научно-технический журнал. — М.: Новые технологии, 2012. — 80 с.
5. Настройка весовых матриц ЗСУР регулятора с помощью биоинспирированных алгоритмов оптимизации. Вестник РГРТУ. 2016. № 55. — Рязань: РГРТУ. — С. 131–139.

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 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')