МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Ижевский государственный технический университет
имени М.Т. Калашникова»

К защите

Руководитель направления
Лялин В.Е.
«____»_____20___г.

Филиппова Татьяна Павловна

Разработка системы кластеризации на основе алгоритма COBWEB

09.04.04 «Программная инженерия»

09.04.04-2 «Системы мультимедиа и компьютерная графика»

Диссертация на соискание академической степени магистра

Магистрант
Филиппова Т.П.

Научный руководитель
к.т.н., доцент Коробейников А.В.

Научный руководитель программы
д.т.н., профессор Мурынов А.И.

Ижевск 2015

РЕФЕРАТ

Диссертация содержит введение, 3 главы, заключение и 2 приложения, изложенные на 60 с. машинописного текста. В работу включены 19 рисунков, 6 таблиц, список литературы из 26 наименований.

В данной работе рассматриваются алгоритмы кластеризации. На основе существующего алгоритма кластеризации COBWEB предлагается создание системы кластеризации, включающей в себя и алгоритм классификации данных. Предполагается, что адаптированный алгоритм будет обладать меньшей вычислительной сложностью, обрабатывать данные, представленные числовыми значениями; кроме того, в разработанной системе увеличится быстродействие обработки данных по сравнению с реализацией такой же системы с использованием исходного алгоритма. На базе разработанного алгоритма реализовано программное обеспечение для проверки высказанных предположений, проведен ряд тестов и испытаний. Результаты работы алгоритма и программы отражены в главе 3, а так же в заключении.

Ключевые слова: кластеризация, классификация, алгоритм COBWEB, нечеткая функция принадлежности, массив кластеров, разбиение, анализ данных.

ОГЛАВЛЕНИЕ

Список сокращений

БД	–	база данных
ГА	–	генетический алгоритм
ПО	–	программное обеспечение
СУБД	–	система управления базами данных
CU	–	категория полезности

ВВЕДЕНИЕ

Актуальность темы. Человечество узнает об окружающем мире все больше. По оценкам журнала Economist, объем данных каждый год растет на 60%. По прогнозам Digital Universe, к 2020 году на каждого жителя Земли, включая стариков и детей, будет приходиться 5200 ГБ данных, а полный объем данных, которыми будет обладать человечество, составит 35 ЗБ. С 2013 года большие данные (англ. big data) в информационных технологиях как академический предмет изучаются в появившихся вузовских программах по науке о данных и вычислительным наукам и программной инженерии. Компания IDC оценивает, что к 2020 году, 33% всех данных будут содержать информацию, которая может быть ценной, если ее анализировать. Необходимо разработать методы обнаружения знаний, скрытых в больших объёмах исходных «сырых» данных. В текущих условиях глобальной конкуренции именно найденные закономерности (знания) могут быть источником дополнительного конкурентного преимущества. Здесь на первый план встают методы Data Mining (интеллектуального анализа данных) — обнаружения в данных ранее неизвестных, нетривиальных, практически полезных и доступных для интерпретации знаний, необходимых для принятия решений в различных сферах человеческой деятельности.

Методами Data Mining занимаются такие ученые, как Григорий Пятецкий-Шапиро, Шабан Джерабе, Лиз Гетур, Роберт Гроссман, Ронен Фелдман.

Однако достигнутые в настоящее время результаты не решают в полной мере проблему анализа данных.

Таким образом, актуальной является задача **кластеризации** - автоматического разбиения элементов некоторого множества на группы в зависимости от их схожести[1]. Кластеризация, во-первых, применятся для анализа данных (упрощение работы с информацией, визуализация данных). Во-вторых, для группировки и распознавания объектов. В-третьих, для извлечения и поиска информации, построения удобных классификаторов.

Целью работы является разработка алгоритма кластеризации числовых данных.

Для достижения поставленной цели решаются следующие **задачи:**

1) Разработка алгоритма кластеризации и классификации образцов, представленных числовыми значениями.
2) Оптимизация работы алгоритма на этапе нахождения суммы при вычисления полезности категории кластера.
3) Разработка технологии и создание программного обеспечения обработки данных, реализующих разработанные средства и методы кластеризации и классификации.

Проведение экспериментальных исследований разработанных средств и методов для оценки их эффективности и возможностей использования при решении различных прикладных задач, связанных с обработкой данных.

Объектом исследования является алгоритм кластеризации и классификации.

Предметом исследования являются математические методы, используемые на различных этапах алгоритмов кластеризации и классификации.

Методы исследования. В работе применялись теоретические и экспериментальные методы исследования. К теоретическим методам исследования можно отнести сравнительный анализ различных алгоритмов кластеризации, применимых к числовым значениям, разработка алгоритма кластеризации и классификации, снижение вычислительной сложности алгоритма, а также программная реализация системы на основе разработанного алгоритма.

Достоверность изложенных положений работы подтверждается результатами практического применения разработанных методов, алгоритмов, программных средств и технологии обработки данных, научной публикацией. Достоверность и обоснованность полученных в работе результатов и выводов подтверждается при их сравнительном анализе с известными результатами современных исследований и разработок.

Теоретические положения, установленные в работе, обосновываются

адекватным выбором исходных посылок и последовательным применением математического аппарата при получении из них выводов, а также верификацией этих выводов данными систематического исследования полученных аналитических результатов.

Достоверность экспериментальных результатов подтверждается их согласованностью с теоретическими выводами, обоснованным выбором корректных критериев при построении алгоритмов обработки информации, воспроизводимостью результатов на больших объемах экспериментального материала при выполнении серий вычислительных экспериментов с большим количеством изменяемых значений влияющих параметров, наглядностью интерпретации полученных практических результатов обработки информации.

На защиту выносятся результаты разработки и исследования методов и алгоритмов кластеризации, а также результаты практической реализации этих методов и алгоритмов – технология и программные средства обработки данных для решения различных прикладных задач, связанных с обработкой информации, в том числе:

- алгоритм кластеризации и классификации, полученный на основе алгоритма COBWEB;
- программное средство, реализующее данный алгоритм;
- разработанная структура нейросети с использованием алгоритма в качестве кластеризатора;
- результаты экспериментов, проведенные на больших объемах данных (кардиоциклах сигнала ЭКГ).

Научная новизна полученных результатов определяется впервые проведенными исследованиями, направленными на разработку алгоритмов и программных средств для кластеризации и классификации, в ходе которых:

- реализован алгоритм кластеризации и классификации на основе алгоритма COBWEB, позволяющий самостоятельно производить разбиение данных на разумное количество кластеров;
- улучшено быстродействие путем применения плотности вероятно-

сти совместно с алгоритмом COBWEB.

Практическая ценность работы заключается в применении новых методов анализа данных и разработке программного обеспечения, реализующего алгоритм кластеризации.

Апробация работы. Созданный алгоритм был применен для кластеризации кардиоциклов сигналов ЭКГ.

Публикации. По теме данной работы была представлена статья в соавторстве с А.В. Коробейниковым «Разработка методов кластеризации и классификации на основе алгоритма *Cobweb*» - Информационные технологии в науке, промышленности и образовании: Сборник трудов региональной научно-технической конференции. – Ижевск: Изд-во ИжГТУ, 2013. – С. 110-115.

Готовится к печати статья «Система кластеризации на основе алгоритма COBWEB».

Объем и структура диссертационной работы. Диссертация содержит введение, 3 главы, заключение и 2 приложения, изложенные на 60 с. машинописного текста. В работу включены 19 рисунков, 6 таблиц, список литературы из 26 наименований.

В первой главе представлен краткий обзор методов кластеризации, инструментов для решения поставленных задач.

Во второй главе описывается математическая модель разработанного алгоритма, рассматривается структура системы, механизм взаимодействия подсистем.

В третьей главе подробно описываются алгоритмы подсистем, реализация алгоритма кластеризации и классификации. Кроме того, приведены примеры и результаты работы алгоритмов с помощью разработанного программного обеспечения.

В заключении диссертационной работы сформулированы основные выводы и результаты выполненных исследований и намечены возможные перспективные направления их развития.

В приложениях помещены экспериментальные данные и результаты.

1. АНАЛИТИЧЕСКИЙ ОБЗОР

В данной главе рассмотрим обзор методов кластеризации, а так же инструментов для решения поставленных задач.

1.1.Классификация алгоритмов кластеризации

Число кластерных алгоритмов довольно велико, но все их можно разделить на иерархические и неиерархические. Иерархические в свою очередь делятся на 2 основные группы[19]:

на *агломеративные*, характеризуемые последовательным объединением исходных элементов и соответствующим уменьшением числа кластеров (построение кластеров снизу вверх);

на *дивизимные (делимые)*, в которых число кластеров возрастает начиная с одного, в результате чего образуется последовательность расщепляющих групп (построение кластеров сверху вниз).

Неиерархические алгоритмы в свою очередь могут подразделятся[12]:

статистические алгоритмы – основаны на том, что кластеры неплохо описываются некоторым семейством вероятностных распределений. В этом случае задача кластеризации сводится к разделению смеси распределений по конечной выборке.

графовые алгоритмы – обширный класс алгоритмов кластеризации основан на представлении выборки в виде графа. В данном типе алгоритмов вершинам графа сопоставляются объекты выборки, а ребрам попарные расстояния между вершинами.

Также отдельно можно выделить применение *нейронных сетей* и *генетических алгоритмов* для разбиения исходного множества объектов на кластеры. [21,22,23]

1.2.Обзор существующих алгоритмов кластеризации

1.2.1. k-Means алгоритм

Наиболее простой, но в то же время достаточно неточный метод кластеризации в классической реализации – алгоритм k-Means[2]. Он разбивает множество элементов векторного пространства на заранее известное число кластеров k. Действие алгоритма таково, что он стремится минимизировать среднеквадратичное отклонение на точках каждого кластера. Основная идея заключается в том, что на каждой итерации заново вычисляется центр масс для каждого кластера, полученного на предыдущем шаге, затем векторы разбиваются на кластеры вновь, в соответствии с тем, какой из новых центров оказался ближе по выбранной метрике.

Данный алгоритм состоит из следующих шагов:

1) случайно выбрать k точек, являющихся начальными координатами «центрами масс» кластеров (любые k из n объектов, или вообще k случайных точек),
2) отнести каждый объект к кластеру с ближайшим «центром масс»,
3) пересчитать «центры масс» кластеров согласно текущему членству,
4) если критерий остановки не удовлетворен, вернуться к шагу 2.

В качестве критерия остановки обычно выбирают один из двух: отсутствие перехода объектов из кластера в кластер на шаге 2 или минимальное изменение среднеквадратической ошибки.

Недостатки алгоритма:

1) не гарантируется достижение глобального минимума суммарного квадратичного отклонения, а только одного из локальных минимумов,
2) результат зависит от выбора исходных центров кластеров, их оптимальный выбор неизвестен,
3) число кластеров надо знать заранее.

1.2.2. Метод нейронных сетей SOM

Самоорганизующаяся карта Кохонена (англ. Self-organizing map — SOM) — соревновательная нейронная сеть с обучением без учителя, выполняющая задачу визуализации и кластеризации[3]. Идея сети предложена финским учёным Т. Кохоненом. Является методом проецирования многомерного пространства в пространство с более низкой размерностью (чаще всего, двумерное), применяется также для решения задач моделирования, прогнозирования и др. Является одной из версий нейронных сетей Кохонена.

Самоорганизующаяся карта состоит из компонент, называемых узлами или нейронами. Их количество задаётся аналитиком. Каждый из узлов описывается двумя векторами. Первый — т. н. вектор веса m, имеющий такую же размерность, что и входные данные. Второй — вектор r, представляющий собой координаты узла на карте. Обычно узлы располагают в вершинах регулярной решётки с квадратными или шестиугольными ячейками.

Изначально известна размерность входных данных, по ней некоторым образом строится первоначальный вариант карты. В процессе обучения векторы веса узлов приближаются к входным данным. Для каждого наблюдения (семпла) выбирается наиболее похожий по вектору веса узел, и значение его вектора веса приближается к наблюдению. Также к наблюдению приближаются векторы веса нескольких узлов, расположенных рядом, таким образом если в множестве входных данных два наблюдения были схожи, на карте им будут соответствовать близкие узлы. Циклический процесс обучения, перебирающий входные данные, заканчивается по достижении картой допустимой (заранее заданной аналитиком) погрешности, или по совершении заданного количества итераций.

Сеть может быть использована для кластерного анализа только в том случае, если заранее известно число кластеров.

Сущность алгоритма SOM, предложенного Кохоненом, состоит в простом геометрическом вычислении свойств Хеббоподобного правила обучения и ла-

теральных взаимодействий. Существенными характеристиками этого алгоритма являются следующие:

- Непрерывное входное пространство образов активации, которые генерируются в соответствии с некоторым распределением вероятности.
- Топология сети в форме решетки, состоящей из нейронов. Она определяет дискретное выходное пространство.
- Зависящая от времени функция окрестности $h_{j,i(x)}(n)$ которая определена в окрестности нейрона-победителя $i(\mathbf{x})$.
- Параметр скорости обучения $\eta(n)$, для которого задается начальное значение и который постепенно убывает во времени n, но никогда не достигает нуля.

Для функции окрестности и параметра скорости обучения на этапе упорядочивания (т.е. приблизительно для первой тысячи итераций) можно использовать формулы (1.1) и (1.2) соответственно.

$$h_{j,i(\mathbf{x})}(n) = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2(n)}\right), \quad n = 0, 1, 2 \ldots, \tag{1.1}$$

$$\eta(n) = \eta_0 \exp\left(-\frac{n}{\tau_2}\right), \quad n = 0, 1, 2, \ldots, \tag{1.2}$$

Для хорошей статистической точности на этапе сходимости параметр $\eta(n)$ должен быть установлен в очень малое значение (0,01 или меньше).

Что же касается функции окрестности, то в начале этапа сходимости она должна содержать только ближайших соседей нейрона-победителя (при этом может включать только его самого).

Не считая инициализации, алгоритм проходит три основных шага: *подвыборка* (sampling), *поиск максимального соответствия* (similarity matching) и *корректировка* (updating).

Кратко весь алгоритм можно описать следующим образом:

1) Инициализация. Для исходных векторов синаптических весов $\mathbf{W}_j(0)$ выбираются случайные значения. Единственным требованием

здесь является различие векторов для разных значений j = 1,2,...,*l*, где *l* - общее количество нейронов в решетке. При этом рекомендуется сохранять малой амплитуду значений. Еще одним способом инициализации алгоритма является случайный выбор множества весов $\{\mathbf{w}_j(0)\}_{j=1}^{l}$ из доступного множества входных векторов $\{\mathbf{x}_i\}_{i=1}^{N}$.

2) Подвыборка. Выбираем вектор **x** из входного пространства с определенной вероятностью. Этот вектор представляет собой возбуждение, которое применяется к решетке нейронов. Размерность вектора **x** равна *m*.
3) Поиск максимального подобия. Находим наиболее подходящий (победивший) нейрон *i(x)* на шаге *n*, используя критерий минимума Евклидова расстояния:

$$i(\mathbf{x}) = \arg\min_j \|\mathbf{x} - \mathbf{w}_j\|, \quad j = 1, 2, \ldots, l. \tag{1.3}$$

4) Коррекция. Корректируем векторы синаптических весов всех нейронов, используя следующую формулу:

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n) h_{j,i(x)}(n)(\mathbf{x} - \mathbf{w}_j(n)), \tag{1.4}$$

где $\eta(n)$ – параметр скорости обучения; $h_{j,i(x)}(n)$ – функция окрестности с центром в победившем нейроне *i(x)*.

Оба этих параметра динамически изменяются во время обучения с целью получения лучшего результата.

5) Продолжение. Возвращаемся к шагу 2 и продолжаем вычисления до тех пор, пока в карте признаков не перестанут происходить заметные изменения.

По завершении процесса сходимости алгоритма SOM вычисленная им карта признаков (feature tap) отображает важные статистические характеристики входного пространства.

Для начала обозначим пространстенно-непрерывное входное пространство (данных) (spatially continous (data) space) символом **X**. Его

топология определяется метрическими связями векторов $\mathbf{x} \in \mathbf{X}$. Теперь обозначим символом **A** пространственно дискретное выходное пространство (spatially discrete output space). Его топология определяется упорядоченным множеством нейронов вычислительных узлов решетки. Обозначим символом **Ф** нелинейное преобразование (называемое картой признаков), которое отображает входное пространство **X** в выходное пространство **A**:

$$\mathbf{\Phi} : \mathbf{X} \rightarrow \mathbf{A}. \quad (1.5)$$

Выражение (1.5) можно рассматривать как соотношение (определяющее положение нейрона $i(\boldsymbol{x})$, победившего при подаче на вход сети вектора **x**) в абстрактной форме. Например, в контексте нейробиологии входное пространство **X** может представлять множество координат соматосенсорных рецепторов, распределенных по всей поверхности тела. Соответственно выходное пространство A представляет множество нейронов, расположенных в тех слоях коры головного мозга, которые отвечают за соматосенсорные рецепторы.

Для данного входного вектора **x** алгоритм SOM определяет наиболее подходящий (т.е. победивший) нейрон $i(\boldsymbol{x})$ в выходном пространстве **A**, руководствуясь картой признаков **Ф**. Вектор синаптических весов W_i нейрона $i(\boldsymbol{x})$ после этого можно рассматривать как указатель на этот нейрон из входного пространства **X**.

Это значит, что синаптические элементы вектора W_i можно рассматривать как координаты образа нейрона i, проектируемые во входное пространство. [26]

1.2.3. Нечеткая кластеризация

Алгоритм нечеткой кластеризации называют FCM-алгоритмом (Fuzzy Classifier Means, Fuzzy C-Means)[4][5]. Целью FCM-алгоритма кластеризации является автоматическая классификация множества объектов, которые задаются векторами признаков в пространстве признаков. Другими словами, такой ал-

горитм определяет кластеры и соответственно классифицирует объекты. Кластеры представляются нечеткими множествами, и, кроме того, границы между кластерами также являются нечеткими.

Вместо однозначного ответа на вопрос, к какому кластеру относится объект, он определяет вероятность того, что объект принадлежит к тому или иному кластеру.

Недостатки данного алгоритма:

1) вычислительная сложность,
2) задание количества кластеров,
3) неопределенность с объектами, которые удалены от центров всех кластеров,
4) выделение кластеров сферической формы.

1.2.4. Генетический алгоритм

Это алгоритм, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, напоминающих биологическую эволюцию[6]. Является разновидностью эволюционных вычислений. Отличительной особенностью генетического алгоритма является акцент на использование оператора «скрещивания», который производит операцию рекомбинации решений-кандидатов, роль которой аналогична роли скрещивания в живой природе.

Задача формализуется таким образом, чтобы её решение могло быть закодировано в виде вектора («генотипа») генов, где каждый ген может быть битом, числом или неким другим объектом. В классических реализациях ГА предполагается, что генотип имеет фиксированную длину. Однако существуют вариации ГА, свободные от этого ограничения.

Общая схема данного подхода:

1) выбрать начальную случайную популяцию множества решений и получить оценку качества для каждого решения,

2) создать и оценить следующую популяцию решений, используя эволюционные операторы: оператор выбора – с большой вероятностью предпочитает хорошие решения; оператор рекомбинации (обычно это «кроссовер») – создает новое решение на основе рекомбинации из существующих; оператор мутации – создает новое решение на основе случайного незначительного изменения одного из существующих решений;
3) повторять шаг 2 до получения оптимального разбиения.

Главным достоинством генетических алгоритмов в данном применении является то, что они ищут глобальное оптимальное решение. Большинство популярных алгоритмов оптимизации выбирают начальное решение, которое затем изменяется в ту или иную сторону. Таким образом получается хорошее разбиение, но не всегда самое оптимальное. Операторы рекомбинации и мутации позволяют получить решения, сильно не похожие на исходное – таким образом осуществляется глобальный поиск.

Недостатки алгоритма:

1) не гарантирует обнаружения глобального решения за приемлемое время,
2) не гарантирует того, что найденное решение будет оптимальным решением,
3) вычислительная сложность.

1.2.5. Алгоритм COBWEB

В алгоритме COBWEB реализовано вероятностное представление категорий[7]. Принадлежность категории определяется не набором значений каждого свойства объекта, а вероятностью появления значения. Например, $P(A_j = \upsilon_{ij} \mid C_k)$ - это условная вероятность, с которой свойство A_j, принимает значение υ_{ij}, если объект относится к категории C_k. Для каждой категории в иерархии определены вероятности вхождения всех значений каждого свойства. При предъявлении

нового экземпляра система COBWEB оценивает качество отнесения этого примера к существующей категории и модификации иерархии категорий в соответствии с новым представителем. Критерием оценки качества классификации является полезность категории (category utility). Критерий полезности категории был определен при исследовании человеческой категоризации.

Критерий полезности категории максимизирует вероятность того, что два объекта, отнесенные к одной категории, имеют одинаковые значения свойств и значения свойств для объектов из различных категорий отличаются. Полезность категории определяется формулой (1.6):

$$CU = \sum_{k=1}^{n} \sum_i \sum_j P(A_j = v_{ij} | C_k) P(C_k | A_j = v_{ij}) P(A_j = v_{ij}), \qquad (1.6)$$

где $P(A_j = \upsilon_{ij} \mid C_k$ –предсказуемость;

$P(C_k \mid A = \upsilon_{ij})$ – предиктивность;

$P(A = \upsilon_{ij})$ – весовой коэффициент.

Значения суммируются по всем категориям C_k, всем свойствам A_j и всем значениям свойств υ_{ij}. Предсказуемость – это вероятность того, что объект, для которого свойство A_j- принимает значение υ_{ij}, относится к категории C_k. Чем выше это значение, тем вероятнее, что свойства двух объектов, отнесенных к одной категории, имеют одинаковые значения. Предиктивность – это вероятность того, что для объектов из категории C_k свойство A_j принимает значение υ_{ij}. Чем больше эта величина, тем менее вероятно, что для объектов, не относящихся к данной категории, это свойство будет принимать указанное значение. Весовой коэффициент усиливает влияние наиболее распространенных свойств. Благодаря совместному учету этих значений высокая полезность категории означает высокую вероятность того, что объекты из одной категории обладают одинаковыми свойствами, и низкую вероятность наличия этих свойств у объектов из других категорий.

Описание алгоритма представлено на рис. 1.1.

Этот алгоритм достаточно эффективен и выполняет кластеризацию на разумное число классов. Поскольку в нем используется вероятностное представление принадлежности, получаемые категории являются гибкими и робастны-

ми. Кроме того, в нем проявляется эффект категорий базового уровня, поддерживается прототипирование и учитывается степень принадлежности. Он основан не на классической логике, а, подобно методам теории нечетких множеств, учитывает "неопределенность" категоризации как необходимый компонент обучения и рассуждений в гибкой и интеллектуальной манере.

Недостатки алгоритма:

1) вероятностное представление кластеров делает очень сложным их обновление, особенно в том случае, когда атрибуты имеют большое число возможных значений,
2) возможность работы только с качественными данными – данными, представленными в форме описательного материала.

Наиболее интересен алгоритм кластеризации COBWEB. Этот алгоритм будет использован в дальнейшем в работе: он относительно прост в реализации, не требует действий со стороны конечного пользователя, гибок. Однако необходимо устранить недостатки этого алгоритма: будет произведена модификация алгоритма для обработки числовых значений, а так же адаптация алгоритма для классификации, с целью упрощения добавления нового объекта.

Алгоритм Cobweb

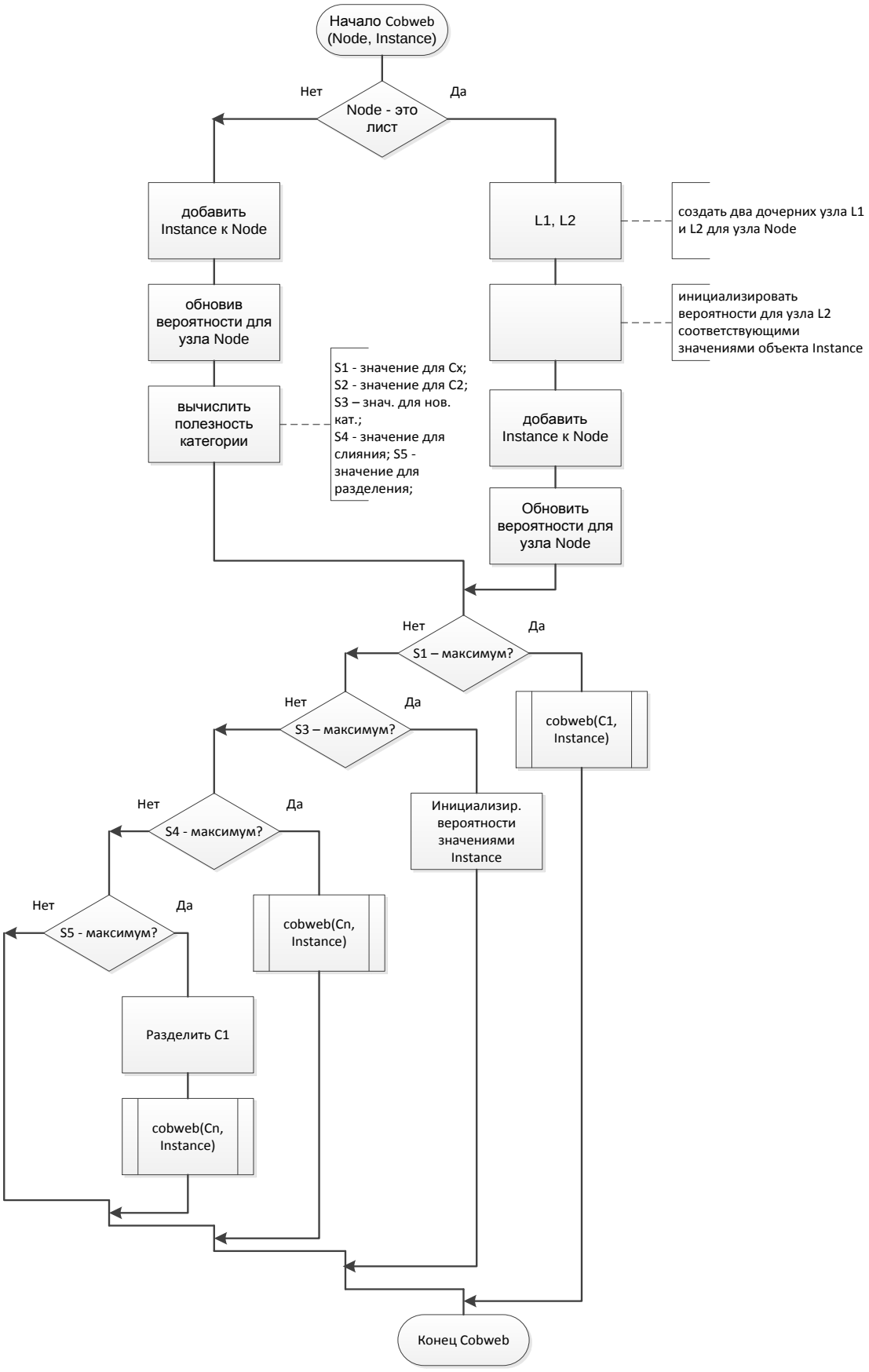

Рис. 1.1

1.3.Инструментальные средства

1.3.1. Среда программирования .NET

.NET Framework — программная платформа, выпущенная компанией Microsoft в 2002 году. Основой платформы является общеязыковая среда исполнения Common Language Runtime (CLR), которая подходит для разных языков программирования. Функциональные возможности CLR доступны в любых языках программирования, использующих эту среду.

Основной идеей при разработке .NET Framework являлось обеспечение свободы разработчика за счёт предоставления ему возможности создавать приложения различных типов, способные выполняться на различных типах устройств и в различных средах.

Программа для .NET Framework, написанная на любом поддерживаемом языке программирования, сначала переводится компилятором в единый для .NET промежуточный байт-код Common Intermediate Language (CIL) (ранее назывался Microsoft Intermediate Language, MSIL). В терминах .NET получается сборка. Затем код либо исполняется виртуальной машиной Common Language Runtime (CLR), либо транслируется утилитой NGen.exe в исполняемый код для конкретного целевого процессора. Использование виртуальной машины предпочтительно, так как избавляет разработчиков от необходимости заботиться об особенностях аппаратной части. В случае использования виртуальной машины CLR встроенный в неё JIT-компилятор «на лету» (just in time) преобразует промежуточный байт-код в машинные коды нужного процессора. Современная технология динамической компиляции позволяет достигнуть высокого уровня быстродействия. Виртуальная машина CLR также сама заботится о базовой безопасности, управлении памятью и системе исключений, избавляя разработчика от части работы.

1.3.2. Язык программирования Java

Программы на языке Java транслируются в байт-код, выполняемый виртуальной машиной Java (JVM) — программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор.

Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание.

Часто к недостаткам концепции виртуальной машины относят снижение производительности. Ряд усовершенствований несколько увеличил скорость выполнения программ на Java:

- применение технологии трансляции байт-кода в машинный код непосредственно во время работы программы (JIT-технология) с возможностью сохранения версий класса в машинном коде,
- широкое использование платформенно-ориентированного кода (native-код) в стандартных библиотеках,
- аппаратные средства, обеспечивающие ускоренную обработку байт-кода (например, технология Jazelle, поддерживаемая некоторыми процессорами фирмы ARM).

Примечательно исследование, проведённое компанией Google, согласно которому отмечается существенно более низкая производительность и большее потребление памяти в тестовых примерах на Java в сравнении с аналогичными программами на C++.

Идеи, заложенные в концепцию и различные реализации среды виртуаль-

ной машины Java, вдохновили множество энтузиастов на расширение перечня языков, которые могли бы быть использованы для создания программ, исполняемых на виртуальной машине. Эти идеи нашли также выражение в спецификации общеязыковой инфраструктуры CLI, заложенной в основу платформы .NET компанией Microsoft.

1.3.3. Программа анализа данных WEKA

Weka (Waikato Environment for Knowledge Analysis) — свободное программное обеспечение для анализа данных, написанное на языке Java в университете Уайкато (Новая Зеландия), распространяющееся по лицензии GNU GPL.

Weka представляет собой набор средств визуализации и алгоритмов для интеллектуального анализа данных и решения задач прогнозирования, вместе с графической пользовательской оболочкой для доступа к ним.

Weka позволяет выполнять такие задачи анализа данных, как подготовку данных (preprocessing), отбор признаков (feature selection), кластеризацию, классификацию, регрессионный анализ и визуализацию результатов.

Однако у системы есть и свои недостатки. В первую очередь к ним стоит отнести наличие собственного сложного формата файла входных данных. Во-вторых, многие алгоритмы в программе взаимодействуют между собой, и изменение какой-либо части программы приведет к необходимости разрешения получившихся межмодульных конфликтов.

1.4.Выводы и постановка целей и задач исследования

Существующие алгоритмы кластеризации в полной мере выполняют все свои функции, однако не лишены некоторых недостатков: необходимость задавать число кластеров, высокая вычислительная сложность, жесткая привязка к начальным условиям кластеризации. В данной работе ставится цель разработки более эффективного алгоритма кластеризации на основе уже существующего за счет применения нечеткой функции принадлежности, которая будет работать с

целочисленными значениями. Так же ставится цель увеличить число настраиваемых параметров, по которым рассчитываются вероятности принадлежности к тому или иному кластеру.

В круг решаемых задач данной работы входит:

- модификация алгоритма кластеризации,
- адаптация алгоритма для классификации,
- определение настраиваемых параметров, влияющих на разбиение.

В качестве инструмента разработки будет использоваться язык программирования C#, так как данный язык позволяет довольно быстро разрабатывать различное ПО любой сложности в кратчайшие сроки.

2. РАЗРАБОТКА АЛГОРИТМА

В данной главе описывается математическая модель разработанного алгоритма, рассматривается структура системы, механизм взаимодействия подсистем.

2.1.Математическая модель алгоритма

Для кластеризации используем алгоритм COBWEB, адаптированный для числовых значений: предлагается использовать нечеткую функцию принадлежности на основе формулы нормального распределения (2.1). [10].

$$f(m,i,j) = \exp\left(-\frac{(a_{mj}-v_{ij})^2}{2\sigma_j^2}\right), \qquad (2.1)$$

где a_{mj}– m-ное значение параметра A_j (случайной величины);

v_{ij} – i-е значение центра отсчета параметра A_j (мат. ожидание);

σ_j^2 – дисперсия A_j.

Для замены вероятностей предлагается использовать формулы (2.2) – (2.4):

$$\bar{P}\left(A_j = v_{ij}\middle|C_k\right) = \frac{1}{l_k}\sum_{m=1}^{l_k} f(m,i,j), \qquad (2.2)$$

$$\bar{P}\left(C_k\middle|A_j = v_{ij}\right) = \frac{\sum_{m=1}^{l_k} f(m,i,j)}{\sum_{m=1}^{l} f(m,i,j)}, \qquad (2.3)$$

$$\bar{P}\left(A_j = v_{ij}\right) = \frac{1}{l}\sum_{m=1}^{l} f(m,i,j), \qquad (2.4)$$

где l_k – количество элементов в кластере C_k;

l – общее количество элементов.

Эвристика полезности разбиения с учетом изменений:

$$\overline{CU} = \sum_{k=1}^{n}\sum_{j=1}^{q}\sum_{i=1}^{d} \bar{P}(A_j = v_{ij})\bar{P}\left(C_k\middle|A_j = v_{ij}\right)\bar{P}\left(A_j = v_{ij}\middle|C_k\right), \quad (2.5)$$

где n – количество элементов в кластере;

q – количество свойств;

d– количество отсчетов параметра A_j.

Значения v_{ij} представляют собой узлы некоторой сетки размером d:

$$v_{ij} = (i - 0.5)\frac{\max(A_j) - \min(A_j)}{d}; i = 1..d, \qquad (2.6)$$

Для уменьшения вычислительной сложности алгоритма, а так же для увеличения быстродействия предлагается перейти от суммы при вычислении полезности категории к плотности распределения. Для этого необходимо использовать следующие формулы:

$$\overline{P}(A_j = v_{il} \mid C_k)_{new} = \frac{\overline{P}(A_j = v_{ij} \mid C_k)_{old} \times l_k + f(m,i,j)_z}{l_k + 1}, \qquad (2.7)$$

$$\overline{P}(C_k \mid A_j = v_{ij})_{new} = \frac{\overline{P}(A_j = v_{ij})_{old} \times l + f(m,i,j)_z}{l + 1}, \qquad (2.8)$$

$$\overline{P}(A_j = v_{ij})_{new} = \frac{\overline{P}(A_j = v_{ij} \mid C_k)_{new} \times (l_k + 1)}{\overline{P}(A_j = v_{ij})_{new} \times (l + 1)}, \qquad (2.9)$$

С целью повышения удобства представления результатов кластеризации, предлагается уйти от построения дерева кластеров, заменив его на массив кластеров. Так как элементы одного кластера являются листьями только одного узла дерева, в котором хранится значение категории полезности, предлагается использовать структуру, каждый экземпляр которой будет представлять один кластер; внутри структуры выделить поле для хранения категории полезности этого кластера, а так же список объектов кластера.

При слиянии двух кластеров K_1 и K_2 использовать формулы:

$$\bar{P}\left(A_j = v_{ij} \middle| C_k\right)_{K_1+K_2} = \frac{\bar{P}(A_j = v_{ij}|C_k)_{K_1} \times l_{kK_1} + \bar{P}(A_j = v_{ij}|C_k)_{K_2} \times l_{kK_2}}{l_{kK_1} + l_{kK_2}}, \qquad (2.10)$$

$$\bar{P}\left(A_j = v_{ij}\right)_{K_1+K_2} = \frac{(\bar{P}(A_j = v_{ij})_{K_1} + \bar{P}(A_j = v_{ij})_{K_2}) \times l}{2 \times l}, \qquad (2.11)$$

$$\bar{P}\left(A_j = v_{ij}\right)_{K_1+K_2} = \frac{\bar{P}(A_j = v_{ij}|C_k)_{K_1+K_2} \times (l_{kK_1} + l_{kK_2})}{2 \times \bar{P}(A_j = v_{ij})_{K_1+K_2} \times l}, \qquad (2.12)$$

где l_k – количество элементов в кластере,

l – общее количество элементов.

2.2.Структура системы

Опишем подсистемы, из которых состоит разработанная система. Система состоит из 3-х основных подсистем:

1) подсистема сбора метаданных образцов;
2) подсистема кластеризации образцов;
3) подсистема классификации образцов.

Структурная схема, описывающая систему и подсистемы, изображена на рисунке 2.1.

Подсистема сбора метаданных образцов представляет собой подсистему предобработки. В результате работы данной подсистемы создаются первоначальные данные об объеме обрабатываемых данных, считываются или рассчитываются все необходимые данные для настройки алгоритма.

Рис. 2.1. – Структурная схема

Подсистема кластеризации образцов производит кластеризацию на основе параметров полученных из результатов работы подсистемы сбора метаданных.

Подсистема классификации образцов производит отнесение нового образца к одному из существующих кластеров на основе массива кластеров, полученного в результате работы подсистемы кластеризации.

Опишем потоки данных в системе на основе схемы модели потоков данных в нотации Гейна-Сарсона, изображенной на рисунке 2.2.

На рисунке изображена схема взаимодействия системы с внешними сущ-

ностями «Пользователь или система сбора данных» и «Пользователь». На вход системы из созданного пользователем файла поступают данные для обработки. На выходе система выдает файл с данными после кластеризации и/или классификации, который и переходит к конечному пользователю системы.

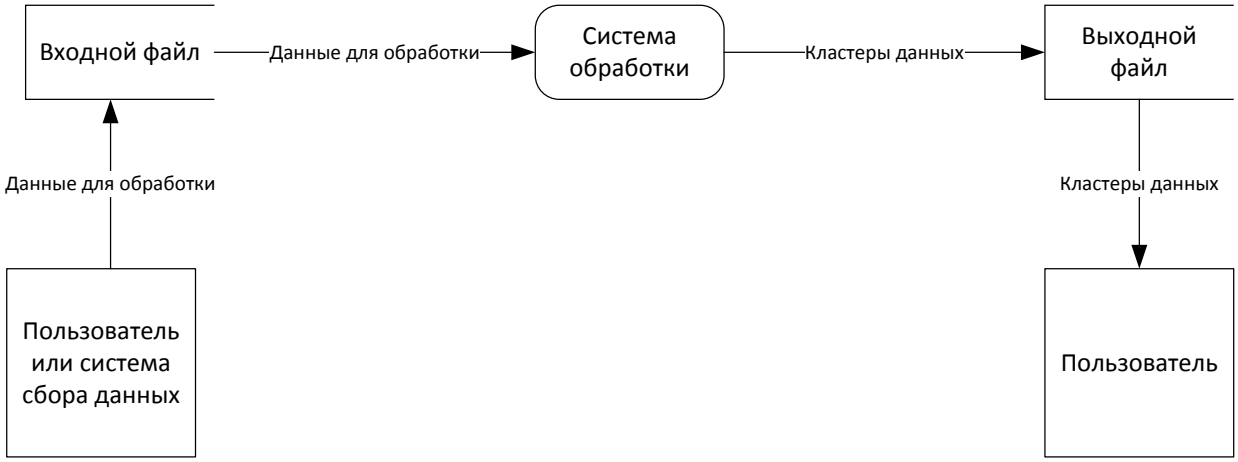

Рис. 2.2. – Контекстная диаграмма потока данных системы

2.3.Описание подсистем

Работа системы происходит в 3 этапа: на первом этапе происходит сбор метаданных, на втором – кластеризация, а на третьем – классификация.

На вход подсистемы сбора данных из созданного пользователем файла поступают данные для обработки. Подсистема сбора метаданных формирует метаданные (границы и размерность сетки, дисперсию) и направляет их вместе с исходными данными в подсистему кластеризации. Подсистема кластеризации обрабатывает данные с помощью модифицированного алгоритма и создает файл с данными, разбитыми на кластеры, а так же с настройками кластеризатора, которые применялись при обработке. Этот файл вместе с пользовательскими данными поступает в систему классификации. После прохождения классификации, пользователю предоставляется файл с результатами.

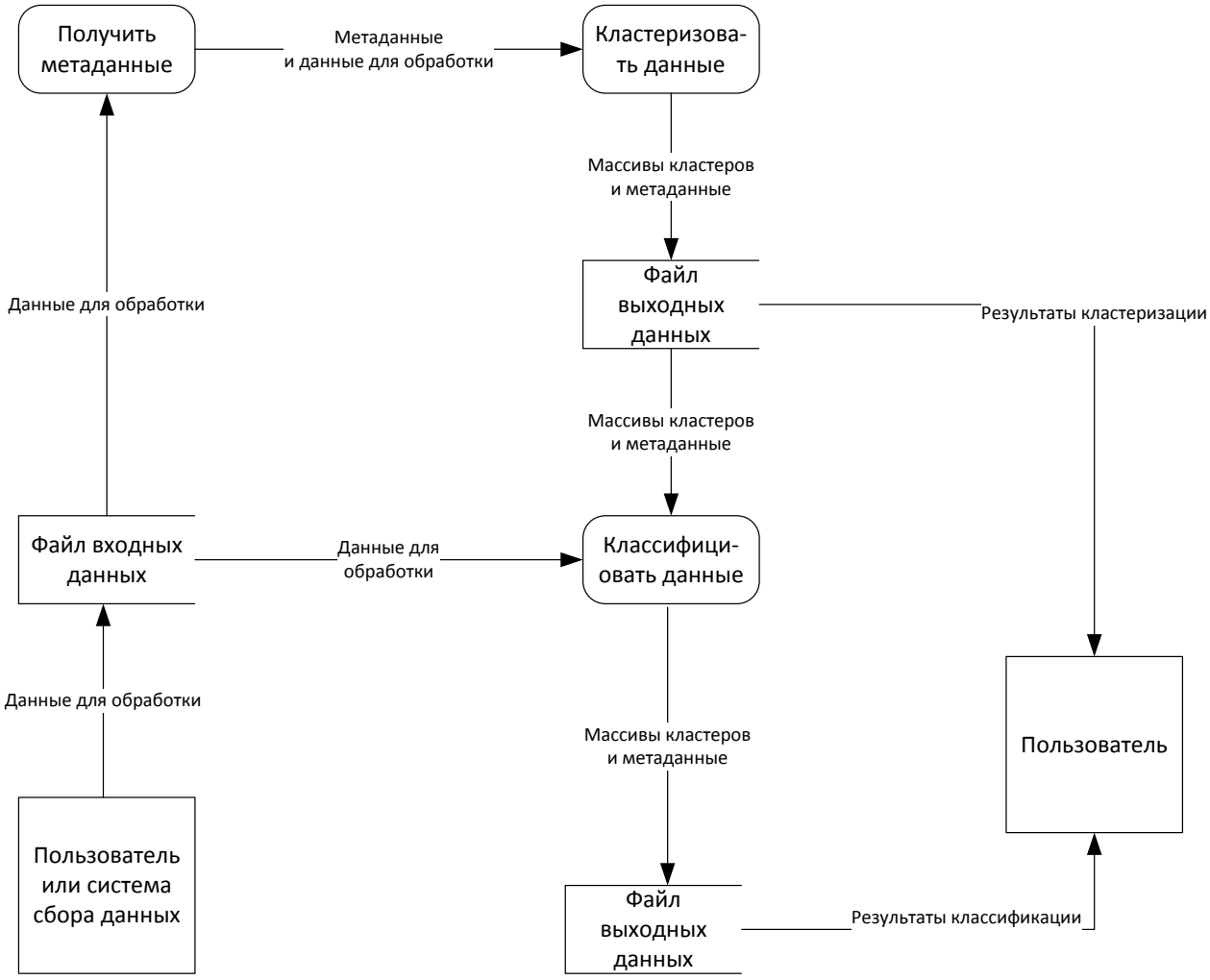

Рис. 2.3. Контекстная диаграмма потока данных подсистем

3. РЕАЛИЗАЦИЯ АЛГОРИТМА

В этой главе подробно описываются алгоритмы подсистем, реализация алгоритма кластеризации и классификации. Кроме того, приведем примеры и результаты работы алгоритмов с помощью разработанного программного обеспечения.

3.1.Описание постановки задачи

3.1.1. Характеристика задачи

Необходимо получить данные для кластеризации из файла. Числовые образцы кластеризовать с помощью алгоритма COBWEB, полученные в результате массивы вывести в текстовый файл в формате, пригодном для дальнейшей обработки алгоритмом классификации. Выполнить классификацию в соответствии с полученными данными.

3.1.2. Входная информация

На вход подаются данные из текстового файла с расширением .txt. Данные могут иметь различное количество параметров структуры, описанной в табл.3.1.

Таблица 3.1

Структура файла входных данных

Название	Формат	Описание
Parameter 1	int	Параметр 1
Parameter 2	int	Параметр 2
…		
Parameter n	int	Параметр n

Таблица 3.2

Пример файла входных данных

Название	Значение
Parameter 1	-5
Parameter 2	10

Данные с графического интерфейса программы описаны в таблице 3.3.

Таблица 3.3

Структура файла выходных данных

Название	Формат	Описание
Capasity	Int	Размерность сетки
Hat	double	«Шляпа» - дисперсия распределения
MaxClustering	Int	Максимальное количество элементов для кластеризации

3.1.3. Выходная информация

Выходной информацией является текстовый файл с расширением .txt, структуры, описанной в табл.3.4.

Таблица 3.4

Структура файла выходных данных

Название	Формат	Описание
Capasity	Int	Размерность сетки
Hat	double	«Шляпа» - дисперсия распределения
Min	int	Минимальное значение сетки
Max	Int	Максимальное значение

		сетки
Num	Int	Номер экземпляра
X	Int	Параметр 1
Y	Int	Параметр 2
numClaster	Int	Номер кластера
CU	Double	Категория полезности

Таблица 3.5

Пример выходных данных

Название	Значение
Capasity	9
Hat	4
Min	-10
Max	10
Num	1
X	-6
Y	7
numClaster	1
CU	0.78432

3.1.4. Математическая постановка задачи

3.1.4.1. Предварительная обработка

Предварительная обработка данных включает в себя занесение данных в память из текстового файла и с интерфейса программы.

3.1.4.2. Кластеризация

Для кластеризации используем алгоритм COBWEB, адаптированный для числовых значений.

3.1.4.3. Классификация

При классификации используем данные, полученные при кластеризации – массив с кластерами, в который мы будем добавлять новые точки; а так же метаданные, полученные в результате предварительной обработки.

3.2.Предварительная обработка

3.2.1. Характеристика задачи

Необходимо выбрать файл с первоначальными данными и ввести параметры кластеризатора в графическом интерфейсе программы. Рассчитать оптимальную размерность сетки, выяснить границы кластеризуемой области.

3.2.2. Входная информация

Входной информацией является файл с расширением txt, содержащий образцы и данные с формы (см. п. 3.2.2.).

3.2.3. Выходная информация

В выходной информации (табл. 3.6) представлены исходные данные о точках, а так же метаданные – размерность и границы сетки, дисперсия.

Таблица 3.6

Структура файла выходных данных

Название	Формат	Описание
Capasity	Int	Размерность сетки
Hat	double	«Шляпа» - дисперсия распределения
Min	int	Минимальное значение сетки
Max	Int	Максимальное значение

		сетки
X	Int	Параметр 1
Y	Int	Параметр 2

3.3.Кластеризация

3.3.1. Характеристика задачи

Необходимо разбить входные данные на множества в соответствии с модифицированным алгоритмом.

3.3.2. Входная информация

Входной информацией являются данные, полученные на этапе предварительной обработки (см. п. 2.3.2)

3.3.3. Выходная информация

Результатом решения алгоритма является файл с расширением txt, содержащий номера кластеров и данные, отнесенные к ним, а также информацию из интерфейса программы (см. п. 3.1.3.).

3.3.4. Алгоритм решения

Логика алгоритма представлена схематически в виде блок-схемы, представленной на рис. 3.1.

Блок-схема алгоритма кластеризации

Рис. 3.1

3.4.Классификация

3.4.1. Характеристика задачи

Задача классификации – это отнесение элемента к одной из существующих категорий, полученных с помощью кластеризации.

3.4.2. Входная информация

Входной информацией являются файлы с расширением txt, содержащие образцы (см. п. 3.2.2.) и категории разбиения (см. п. 3.2.3.).

3.4.3. Выходная информация

Результатом решения алгоритма является файл с расширением txt, содержащий номера кластеров и данные, отнесенные к ним, а также информацию из интерфейса программы (см. п. 3.2.3.).

3.4.4. Алгоритм решения

Логика алгоритма представлена схематически в виде блок-схемы, представленной на рис. 3.2.

Блок-схема алгоритма классификации

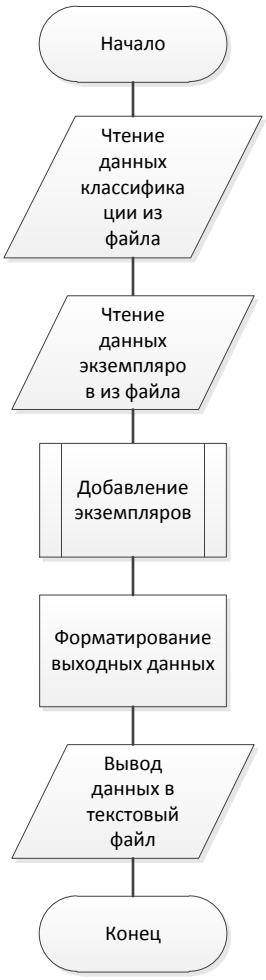

Рис. 3.2.

3.5.Описание контрольного примера

3.5.1. Назначение

Данный контрольный пример служит для проверки корректности работы модифицированных алгоритмов кластеризации и классификации COBWEB.

3.5.2. Исходные данные

Пусть в файле data.txt хранится информация о 29 координатах точек на плоскости.

Запустим программу. В результате на экран должна быть выведена основная форма программы (рис 3.3). При нажатии на кнопку «Загрузка данных» должно появиться окно для ввода пути к файлу.

После открытия файла, при выборе элемента «Кластеризация» должны быть доступны поля для добавления параметров кластеризации, чекбокс для запуска классификации и кнопка «Запуск» - при нажатии на нее, система начинает кластеризацию, результаты которой выводятся в виде диаграммы внизу.

При выборе элемента «Классификация» и нажатии на кнопку «+» должно появиться окно для добавления текстового файла с массивом кластеров и метаданными для кластеризации.

3.5.3. Результаты работы программы на контрольном примере

После запуска приложения, мы видим главное окно программы, представленное на рис.3.3. Выбираем файл данных и вводим параметры, запускаем алгоритм кнопкой «Запуск». Результат работы видим внизу окна (рис.3.4), а так же в текстовом файле (рис.3.5)

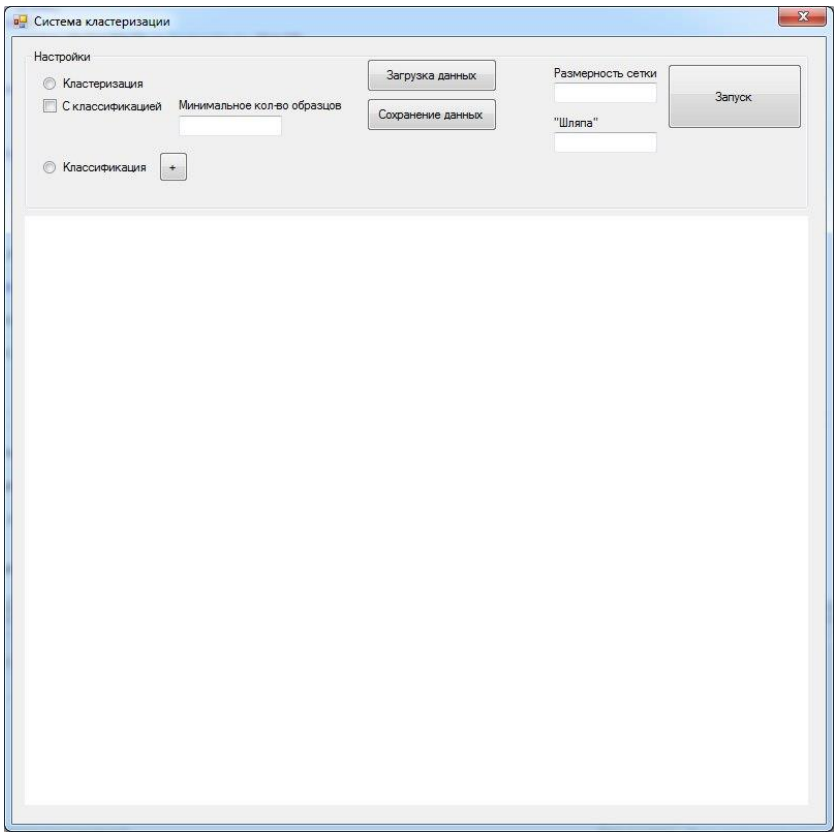

Рис 3.3.Начальное окно программы.

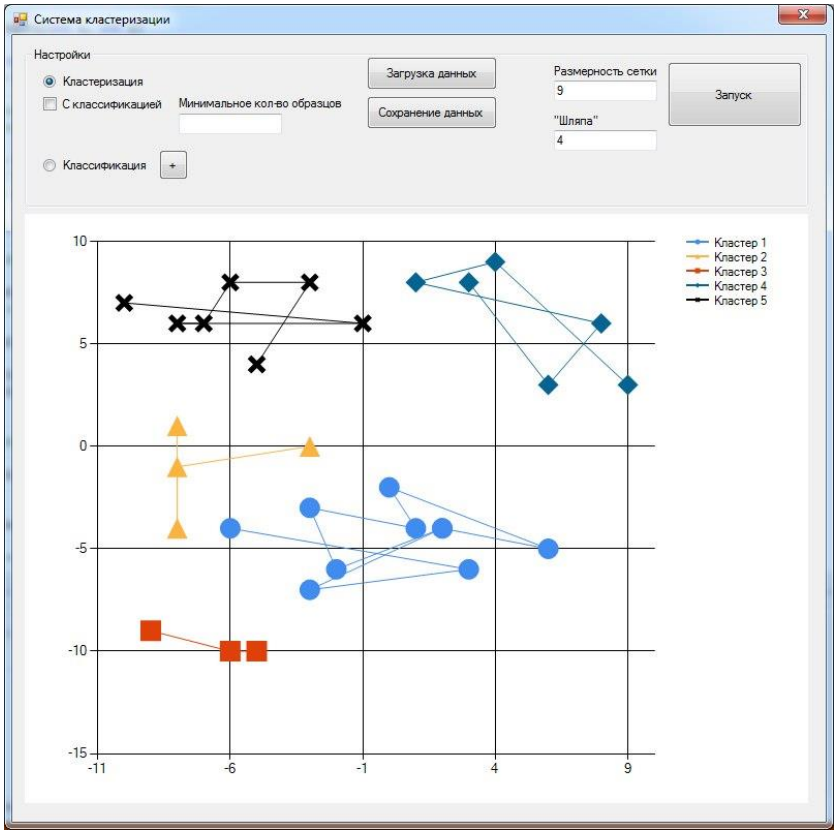

Рис 3.4. Результат работы программы.

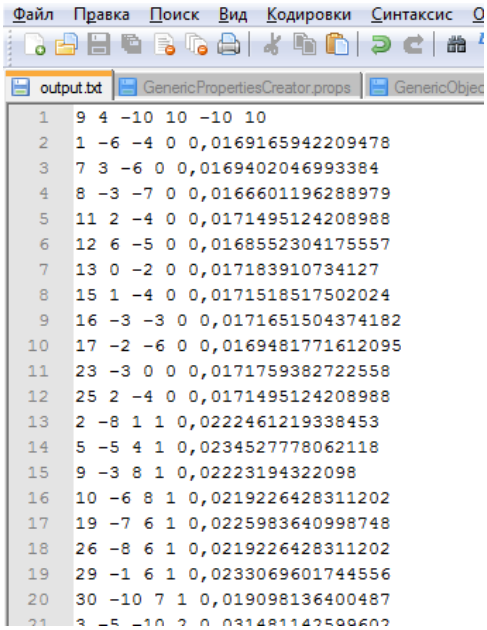

Рис 3.5. Текстовый файл с результатом работы программы

Теперь уменьшим количество подаваемых на вход точек до 20 (рис 3.6.). Мы можем наблюдать, что количество создаваемых кластеров уменьшилось с пяти до четырех.

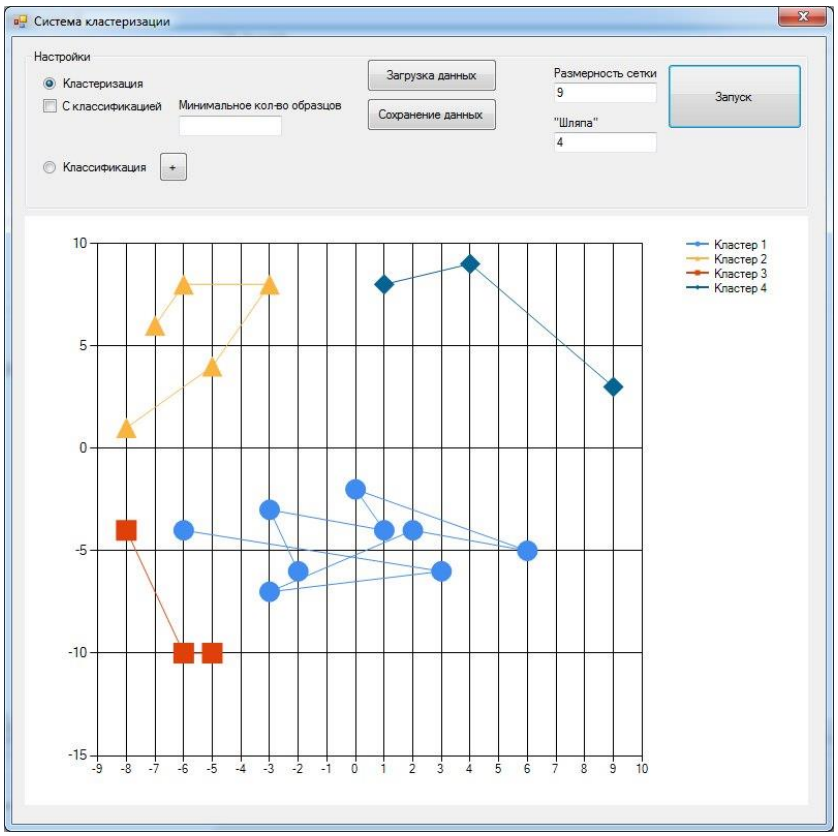

Рис 3.6. Кластеризация 20 точек

Классифицируем оставшиеся 9 точек: они добавятся в уже существующие кластеры (рис 3.7.)

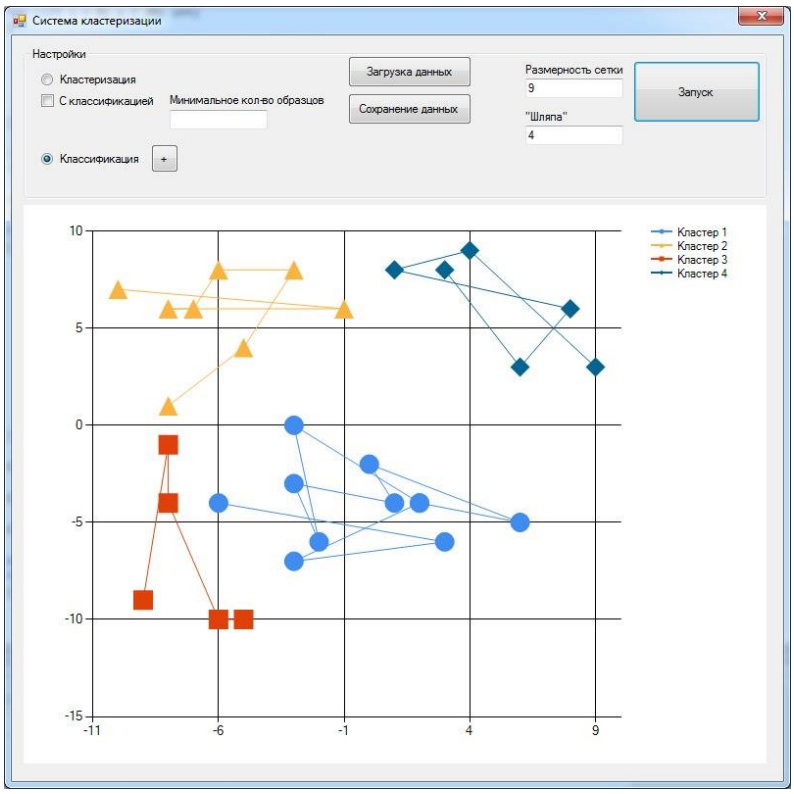

Рис 3.7. Результат работы программы при классификации 9 точек на построенные кластеры из 20 точек

Сделаем это же в одно действие: выберем режим «кластеризация» и отметим галочкой «С классификацией». Зададим 20 образцов для кластеризации:

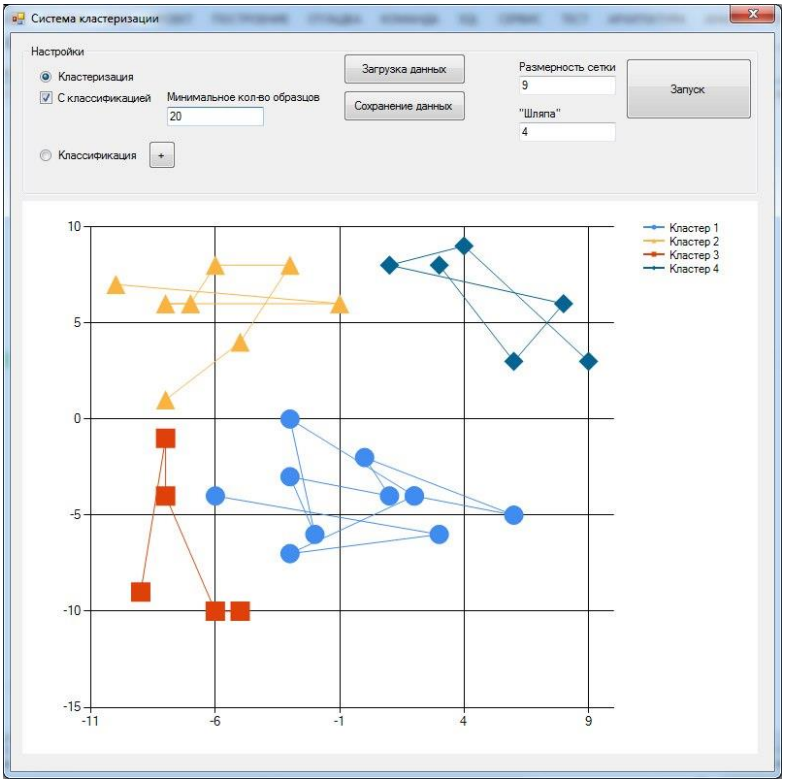

Рис 3.8. Режим классификации и кластеризации

Генерируем большое количество точек и попробуем разбить их (рис 3.9):

Рис 3.9. Работа с большим количеством точек

Отсортируем входные данные по возрастанию абсциссы и ординаты и запустим процесс кластеризации. К сожалению, мы можем наблюдать, что алгоритм очень чувствителен к порядку подачи данных. (рис 3.10)

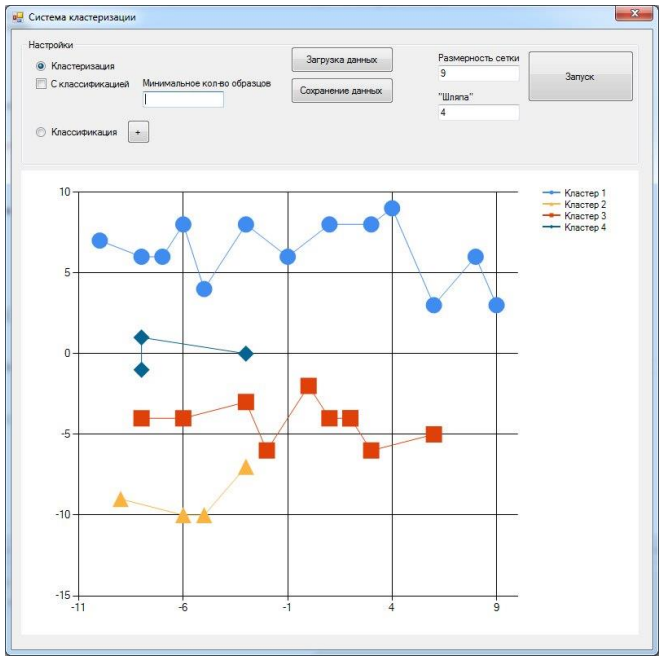

Рис 3.10. Кластеризация отсортированных данных.

ЗАКЛЮЧЕНИЕ

В данной работе на тему: «Система кластеризации и классификации на основе алгоритма COBWEB» была разработана модификация алгоритма COBWEB и его адаптация под задачу классификации, разработано и реализовано ПО для проверки работоспособности алгоритма.

По результатам тестирования можно сделать вывод о пригодности алгоритма для кластеризации и классификации, т.к. он выполняет разбиение исходного множества элементов на разумное количество кластеров; однако, необходимо отметить высокую чувствительность алгоритма к порядку подаваемых данных.

Система в дальнейших разработках может выступать как основа для разработки системы кластеризации данных в режиме реального времени, так как именно в таком режиме минимизируется отрицательный эффект алгоритма от порядка подачи данных и будет максимально извлечена польза из увеличенного быстродействия.

СПИСОК ЛИТЕРАТУРЫ

1) Мандель И.Д. Кластерный анализ. – М.: Финансы и Статистика, 1988

2) Ткаченко А. Н., Грийо Тукало О. Ф.; Дзись О. В.; Лаховец С. М. Метод кластеризации на основе последовательного запуска k-средних с усовершенствованным выбором кандидата на новую позицию вставки. – Информационные технологии и компьютерная техника – Изд. Винницкого НТУ, 2012 – №2. – С.10-20.

3) Зиновьев А. Ю. Визуализация многомерных данных. — Красноярск: Изд. Красноярского государственного технического университета, 2000. — 180 с.

4) Литинский Л.Б., Романов Д.Е. Кластеризация эмпирических данных с использованием нейросетевого подхода. – Искусственный интеллект – Донецк: Наука і освіта, 2006 – №3. – С. 302-308.

5) Кукса П.П. Анализ алгоритма нечеткой кластеризации – Информатика и системы управления в XXI веке. Сборник трудов. — М.: Изд-во МГТУ им. Н.Э. Баумана, 2003. С. 249 - 253.

6) Гладков Л. А., Курейчик В. В., Курейчик В. М. Генетические алгоритмы: Учебное пособие. — 2-е изд. – М: Физматлит, 2006. — С. 320.

7) Люгер, Джордж, Ф. Искусственный интеллект: стратегии и методы решения сложных проблем, 4-е издание. : Пер. с англ. – М.: Вильямс, 2003. – С. 421-425.

8) Герберт Шилдт. Java. Полное руководство. Java SE 7 = Java 7: The Complete Reference. — 8-е изд. – М.: Вильямс, 2012. — 1104 с.

9) Weka 3: Data Mining Software in Java – URL: http://www.cs.waikato.ac.nz/~ml/weka/ Дата обращения: 15.04.2013

10) Коробейников А.В., Исламгалиев И.И. Модификация алгоритма концептуальной кластеризации COBWEB для количественных данных с ис-

пользованием нечеткой функции принадлежности – Приволжский научный вестник. – Ижевск: Самохвалов А.В., 2013. – С. 7-14.

11) ГОСТ 19.105-78 ЕСПД. Общие требования к программным документам. – М: ИПК издательство стандартов, 2001.

12) ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. – М: ИПК издательство стандартов, 2001.

13) ГОСТ 19.505-79. ЕСПД. Руководство оператора. Требования к содержанию и оформлению. – М: ИПК издательство стандартов, 2001.

14) Коробейников А.В. Алгоритмы и комплексы программ мониторно-компьютерных систем для анализа морфологии и ритма электрокардиограмм : Диссертация на соискание ученой степени кандидата технических наук. Иж., 2004.

15) Горбань А.Н., Россиев Д.А. Нейронные сети на персональном компьютере. – Новосибирск, Наука, 1996. – 275 с.

16) Андерсон Т.В. Введение в многомерный статистический анализ. – М: Физматгиз, 1963.

17) Змитрович А.И. Интеллектуальные информационные системы // Киев, 1997. – 368 с.

18) Классификация и кластеризация. Под ред. Дж. Вэн Райзина. М.: Мир, 1980.

19) Дюран Б., Оделл П. Кластерный анализ. – М.: Статистика, 1977.

20) Олдендерфер М.С., Блэшфилд Р.К. Кластерный анализ / Факторный, дискриминантный и кластерный анализ: пер. с англ.; Под. ред. И.С. Енюкова. - М.: Финансы и статистика, 1989.

21) Воронцов К.В. Алгоритмы кластеризации и многомерного шкалирования. Курс лекций. МГУ, 2007.

22) Загоруйко Н. Г. Прикладные методы анализа данных и знаний. - Новосибирск: ИМ СО РАН, 1999.

23) Загоруйко Н. Г., Ёлкина В. Н., Лбов Г. С. Алгоритмы обнаружения эмпирических закономерностей. – Новосибирск: Наука, 1985.

24) Гурман В.Е. Теория вероятностей и математическая статистика: Учебное пособие для вузов. – 9-е изд. – М.: Высш. шк., 2003. – 479с.

25) Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. – 4-е изд. — СПб.: Питер, 2013. — 896 с.

26) Хайкин Саймон. Нейронные сети: полный курс, 2-е издание. : Пер. с англ. - М. Издательский дом "Вильямс", 2006. С. 586-590.

Исходные коды модулей программы

Массив кластеров

```
public class TClaster
{
    public List<TValues> arrValues = new List<TValues>();
    public double A { set; get; }
    public double B { set; get; }
    public double C { set; get; }
    public double CU { set; get; }
}
```

Экземпляр кластера

```
[Serializable]
public class TValues
{
    public int X { set; get; }
    public int Y { set; get; }
    public double A { set; get; }
    public double B { set; get; }
    public double C { set; get; }
    public double CU { set; get; }
}

public class Clasterer
{
    double maxX, maxY, minX, minY;
    public List<TClaster> arrClaster = new List<TClaster>();
    public int Count = 0;
    public int Capasity { get; set; }
    public int Hat { get; set; }

    public Clasterer()
    {

    }

    public void AddClaster(int x, int y)
    {
        int len;

        len = arrClaster.Count;
        arrClaster.Add(new TClaster());
        arrClaster[len].AddValue(x, y);
    }

    public void InsertClaster(int x, int y, int number)
    {
        arrClaster[number].AddValue(x, y);
    }

    public int insertValue(int x, int y)
    {
        int idClasterAdd = 0;
        this.Count++;
```

```
            //UpdateMaxAndMin(x, y);

            if (arrClaster.Count == 0)
            {
                AddClaster(x, y);
            }
            else
            {
                idClasterAdd = CalcCategory(x, y);
                if (arrClaster.Count == idClasterAdd)
                {
                    AddClaster(x, y);
                }
                else
                {
                    InsertClaster(x, y, idClasterAdd);
                }
            }
            Console.WriteLine("Точка ({0}; {1}) добавлена в кластер {2}", x, y, idClaste-
rAdd);

            return idClasterAdd;
        }

        public void SetBorderGrid(int minX, int maxX, int minY, int maxY)
        {
            this.minX = minX;
            this.maxX = maxX;
            this.minY = minY;
            this.maxY = maxY;
        }

        public void UpdateMaxAndMin(int x, int y)
        {
            if (arrClaster.Count == 0)
            {
                this.maxX = x;
                this.minX = x;
                this.maxY = y;
                this.minY = y;
            }
            else
            {
                if (x > this.maxX)
                {
                    this.maxX = x;
                }

                if (x < this.minX)
                {
                    this.minX = x;
                }

                if (y > this.maxY)
                {
                    this.maxY = y;
                }

                if (y < this.minY)
                {
                    this.minY = y;
                }
            }
        }
```

```
        public int CalcCategory(int x, int y)
        {
            double[] CU = new double[arrClaster.Count + 1];
            int indMaxCU = 0;

            //Console.WriteLine();
            //Console.WriteLine(" Точка: ({0}; {1})", x, y);

            int Culength = CU.Length;
            if (Culength > 4)
                Culength = Culength - 1;

            for (int i = 0; i < Culength; i++)
            {
                //Console.WriteLine("Кластер {0}. ", i);

                CU[i] = GetSumProbabilityByValue(x, y, i);
                if (CU[i] > CU[indMaxCU])
                {
                    indMaxCU = i;
                }
                Console.WriteLine("CU = {0}", CU[i]);
                Console.WriteLine("========================================");
            }

            //Console.WriteLine("{0}, {1} : {2}", x, y, CU[indMaxCU]);

            return indMaxCU;
        }

        public double GetSumProbabilityByValue(int x, int y, int numberCluster)
        {
            double result = 0;
            double[] arrMeans = new double[this.Capasity * 2];
            int clasterCount;

            for (int j = 0; j < this.Capasity; j++)
            {
                arrMeans[j] = this.minX + (this.maxX - this.minX) * j / (this.Capasity - 1);
                arrMeans[j + this.Capasity] = this.minY + (this.maxY - this.minY) * j /
(this.Capasity - 1);
                //Console.WriteLine("{0} {1}", arrMeans[j], arrMeans[j+4]);
            }

            if (arrClaster.Count == numberCluster)
            {
                clasterCount = arrClaster.Count + 1;
            }
            else
            {
                clasterCount = arrClaster.Count;
            }

            for (int i = 0; i < clasterCount; i++)
            {
                if (i == numberCluster)
                {
                    result += CalcSumPartitionPoint(x, y, arrMeans, numberCluster);
                }
                else
                {
                    result += CalcSumPartition(x, y, arrMeans, i);
                }
```

```
                //Console.WriteLine();
            }

            return result;
        }

        public double CalcSumPartition(int x, int y, double[] arrMeans, int numberCluster)
        {
            double result = 0, a, b, c, devX, devY, dev;
            int value, type;

            devX = 4;// Math.Abs(arrMeans[1] - arrMeans[0]) * this.Hat - 0.0000000000001;
            devY = 4;// Math.Abs(arrMeans[this.Capasity + 1] - arrMeans[this.Capasity]) *
this.Hat - 0.0000000000001;
            // Console.WriteLine("{0} {1}", devX, devY);

            //Console.WriteLine("   Vij     a       b       c     a*b*c");

            for (int i = 0; i < this.Capasity * 2; i++)
            {
                if (i >= this.Capasity)
                {
                    dev = devY;
                    value = y;
                    type = 1;
                }
                else
                {
                    dev = devX;
                    value = x;
                    type = 0;
                }

                a = GetSumByClaster(arrMeans[i], dev, numberCluster, type);
                c = GetSum(value, arrMeans[i], dev, type);
                b = (a * arrClaster[numberCluster].arrValues.Count) / (c * this.Count);

                /*if (i == this.Capasity)
                    Console.WriteLine("-----------------------------------------");
                Console.WriteLine("{0, 7} {1, 7} {2, 7} {3, 7} {4, 7}",
Math.Round(arrMeans[i], 4), Math.Round(a, 4), Math.Round(b, 4), Math.Round(c, 4),
Math.Round(a * b * c, 4));
                 */
                result += a * b * c;
            }

            return result;
        }

        public double CalcSumPartitionPoint(int x, int y, double[] arrMeans, int numberClus-
ter)
        {
            double result = 0, a, b, c, devX, devY, dev;
            int value, type;

            devX = 4;// Math.Abs(arrMeans[1] - arrMeans[0]) * this.Hat;
            devY = 4;// Math.Abs(arrMeans[this.Capasity + 1] - arrMeans[this.Capasity]) *
this.Hat;
            // Console.WriteLine("{0} {1}", devX, devY);

            //Console.WriteLine("   Vij     a       b       c     a*b*c");

            for (int i = 0; i < this.Capasity * 2; i++)
            {
```

```
                if (i >= this.Capasity)
                {
                    dev = devY;
                    value = y;
                    type = 1;
                }
                else
                {
                    dev = devX;
                    value = x;
                    type = 0;
                }

                if (arrClaster.Count == numberCluster)
                {
                    a = GetSumByNewClaster(value, arrMeans[i], dev, numberCluster, type);
                    c = GetSum(value, arrMeans[i], dev, type);
                    b = a / (c * this.Count);
                }
                else
                {
                    a = GetSumByClaster(value, arrMeans[i], dev, numberCluster, type);
                    c = GetSum(value, arrMeans[i], dev, type);
                    b = (a * (arrClaster[numberCluster].arrValues.Count + 1)) / (c *
this.Count);
                }
                /*
                if (i == this.Capasity)
                    Console.WriteLine("-----------------------------------------");
                Console.WriteLine("{0, 7} {1, 7} {2, 7} {3, 7} {4, 7}",
Math.Round(arrMeans[i], 4), Math.Round(a, 4), Math.Round(b, 4), Math.Round(c, 4),
Math.Round(a * b * c, 4));
                 */
                result += a * b * c;
            }

            return result;
        }

        public double GetSumByClaster(double mean, double dev, int numberCluster, int type)
        {
            double result = 0;
            int i;

            for (i = 0; i < arrClaster[numberCluster].arrValues.Count; i++)
            {
                if (type == 0)
                {
                    result += CalcExp(arrClaster[numberCluster].arrValues[i].X, mean, dev);
                }
                else
                {
                    result += CalcExp(arrClaster[numberCluster].arrValues[i].Y, mean, dev);
                }
            }

            arrClaster[numberCluster].arrValues[arrClaster[numberCluster].arrValues.Count -
1].A = result;

            return result / arrClaster[numberCluster].arrValues.Count;
        }

        public double GetSumByClaster(int value, double mean, double dev, int numberCluster,
int type)
```

```
        {
            double result = 0;
            int i;

            for (i = 0; i < arrClaster[numberCluster].arrValues.Count; i++)
            {
                if (type == 0)
                {
                    result += CalcExp(arrClaster[numberCluster].arrValues[i].X, mean, dev);
                }
                else
                {
                    result += CalcExp(arrClaster[numberCluster].arrValues[i].Y, mean, dev);
                }
            }
            result += CalcExp(value, mean, dev);

            return result / (arrClaster[numberCluster].arrValues.Count + 1);
        }

        public double GetSumByNewClaster(int value, double mean, double dev, int numberClus-
ter, int type)
        {
            double result = 0;

            result += CalcExp(value, mean, dev);

            return result;
        }

        public double GetSum(int value, double mean, double dev, int type)
        {
            double result = 0;
            int i, j;

            for (j = 0; j < arrClaster.Count; j++)
            {
                for (i = 0; i < arrClaster[j].arrValues.Count; i++)
                {
                    if (type == 0)
                    {
                        result += CalcExp(arrClaster[j].arrValues[i].X, mean, dev);
                    }
                    else
                    {
                        result += CalcExp(arrClaster[j].arrValues[i].Y, mean, dev);
                    }
                }

                arrClaster[j].arrValues[arrClaster[j].arrValues.Count - 1].C = result;
            }
            result += CalcExp(value, mean, dev);

            return result / (this.Count);
        }

        public double CalcExp(int value, double mean, double dev)
        {
            return Math.Exp(-Math.Pow(mean - value, 2) / (2 * dev * dev));
        }

        public static byte bDimensionCount = 2;
        public static byte bExemplarCountLimit = 100;
        public double[,] d2Exemplar = new double[bExemplarCountLimit, bDimensionCount];
```

```
        public byte bExemplarCount = 0;

        public static byte bSpanCountLimit = 10;
        public double[] d1DimensionMin = new double[bDimensionCount];
        public double[] d1DimensionMax = new double[bDimensionCount];
        public double[] d1DimensionSpan = new double[bDimensionCount];
        public double[] d1DimensionSigma = new double[bDimensionCount];

        public byte[] b1ExemplarCluster = new byte[bExemplarCountLimit];

        public static byte bClusterCountLimit = bExemplarCountLimit;
        public byte bClusterCount = 0;
        public double[, ,] d3ExemplarSpanMembership = new double[bExemplarCountLimit, bDi-
mensionCount, bSpanCountLimit + 1];

        public double[, ,] d3ClusterSpanMembershipA = new double[bClusterCountLimit, bDimen-
sionCount, bSpanCountLimit + 1];
        public double[, ,] d3ClusterSpanMembershipB = new double[bClusterCountLimit, bDimen-
sionCount, bSpanCountLimit + 1];
        public double[, ,] d3ClusterSpanMembershipC = new double[bClusterCountLimit, bDimen-
sionCount, bSpanCountLimit + 1];

        public byte[] b1ClusterExemplarCount = new byte[bClusterCountLimit];

        public double[] dClusterUtility = new double[bExemplarCountLimit];
        public double[, ,] dClusterUtility_cu = new double[bExemplarCountLimit, bDimen-
sionCount, bSpanCountLimit + 1];

        public double dCalcMembership(double _dValue, double _dCenter, double _dSigma)
        {
            return Math.Exp(-Math.Pow(_dValue - _dCenter, 2) / (2 * _dSigma * _dSigma));
        }

        public void vCalcExemplarSpanMembership()
        {
            for (byte e = 1; e <= bExemplarCount; e++)
            {
                for (byte d = 1; d <= bDimensionCount; d++)
                {
                    for (byte i = 1; i <= d1DimensionSpan[d - 1] + 1; i++)
                    {
                        d3ExemplarSpanMembership[e - 1, d - 1, i - 1] = 0;
                    }
                }
            }

            for (byte e = 1; e <= bExemplarCount; e++)
            {
                for (byte d = 1; d <= bDimensionCount; d++)
                {
                    for (byte i = 1; i <= d1DimensionSpan[d - 1] + 1; i++)
                    {
                        d3ExemplarSpanMembership[e - 1, d - 1, i - 1] +=
                                dCalcMembership(d2Exemplar[e - 1, d - 1],
                                        d1DimensionMin[d - 1] + (i - 1) * (d1DimensionMax[d
- 1] - d1DimensionMin[d - 1]) / d1DimensionSpan[d - 1],
                                        d1DimensionSigma[d - 1]);
                    }
                }
            }
        }

        public void vCalcClusterSpanMembership()
        {
```

```
            for (byte c = 1; c <= bClusterCount; c++)
            {
                b1ClusterExemplarCount[c - 1] = 0;
                for (byte d = 1; d <= bDimensionCount; d++)
                {
                    for (byte i = 1; i <= d1DimensionSpan[d - 1] + 1; i++)
                    {
                        d3ClusterSpanMembershipA[c - 1, d - 1, i - 1] = 0;
                        d3ClusterSpanMembershipB[c - 1, d - 1, i - 1] = 0;
                        d3ClusterSpanMembershipC[c - 1, d - 1, i - 1] = 0;
                    }
                }
            }

            for (byte e = 1; e <= bExemplarCount; e++)
            {
                b1ClusterExemplarCount[b1ExemplarCluster[e - 1]]++;
                for (byte d = 1; d <= bDimensionCount; d++)
                {
                    for (byte i = 1; i <= d1DimensionSpan[d - 1] + 1; i++)
                    {
                        d3ClusterSpanMembershipA[e - 1, d - 1, i - 1] +=
                                d3ExemplarSpanMembership[e - 1, d - 1, i - 1];
                        for (byte c = 1; c <= bExemplarCount; c++)
                        {
                            if (b1ExemplarCluster[e - 1] == b1ExemplarCluster[c - 1])
                            {
                                d3ClusterSpanMembershipC[e - 1, d - 1, i - 1] +=
                                    d3ExemplarSpanMembership[c - 1, d - 1, i - 1];
                            }
                        }
                    }
                }
            }

            for (byte e = 1; e <= bExemplarCount; e++)
            {
                for (byte d = 1; d <= bDimensionCount; d++)
                {
                    for (byte i = 1; i <= d1DimensionSpan[d - 1] + 1; i++)
                    {
                        d3ClusterSpanMembershipB[e - 1, d - 1, i - 1] =
                            d3ClusterSpanMembershipA[e - 1, d - 1, i - 1] /
                            d3ClusterSpanMembershipC[e - 1, d - 1, i - 1];
                    }
                }
            }

            for (byte e = 1; e <= bExemplarCount; e++)
            {
                for (byte d = 1; d <= bDimensionCount; d++)
                {
                    for (byte i = 1; i <= d1DimensionSpan[d - 1] + 1; i++)
                    {
                        d3ClusterSpanMembershipA[e - 1, d - 1, i - 1] /=
b1ClusterExemplarCount[b1ExemplarCluster[e - 1]];
                        d3ClusterSpanMembershipC[e - 1, d - 1, i - 1] /= bExemplarCount;
                    }
                }
            }
        }

        public double[] dCalcClusterUtility()
        {
```

```
        //dClusterUtility = 0;
        for (byte e = 1; e <= bExemplarCount; e++)
        {
            for (byte d = 1; d <= bDimensionCount; d++)
            {
                for (byte i = 1; i <= d1DimensionSpan[d - 1] + 1; i++)
                {
                    dClusterUtility[e - 1] +=
                            d3ClusterSpanMembershipA[e - 1, d - 1, i - 1] *
                            d3ClusterSpanMembershipB[e - 1, d - 1, i - 1] *
                            d3ClusterSpanMembershipC[e - 1, d - 1, i - 1];

                    dClusterUtility_cu[e-1, d-1, i-1] =
                        d3ClusterSpanMembershipA[e - 1, d - 1, i - 1] *
                            d3ClusterSpanMembershipB[e - 1, d - 1, i - 1] *
                            d3ClusterSpanMembershipC[e - 1, d - 1, i - 1];
                }
            }
        }
        return dClusterUtility;
    }

    public int iAddExemplar(double[] _d1Exemplar)
    {
        if (bExemplarCount != bExemplarCountLimit)
        {
            bExemplarCount++;
            for (byte d = 1; d <= bDimensionCount; d++)
                d2Exemplar[bExemplarCount - 1, d - 1] = _d1Exemplar[d - 1];
            return bExemplarCount;
        }
        else return 0;
    }

    public void vSetDimensions(byte _bDimensionIndex,
            double _dDimensionMin, double _dDimensionMax,
            double _dDimensionSpan, double _dDimensionSigma)
    {
        if (_bDimensionIndex == 0)
        {
            for (byte d = 1; d <= bDimensionCount; d++)
            {
                d1DimensionMin[d - 1] = _dDimensionMin;
                d1DimensionMax[d - 1] = _dDimensionMax;
                d1DimensionSpan[d - 1] = _dDimensionSpan;
                if (d1DimensionSpan[d - 1] > bSpanCountLimit)
                    d1DimensionSpan[d - 1] = bSpanCountLimit;
                d1DimensionSigma[d - 1] = _dDimensionSigma;
            }
        }
        else
        {
            d1DimensionMin[_bDimensionIndex - 1] = _dDimensionMin;
            d1DimensionMax[_bDimensionIndex - 1] = _dDimensionMax;
            d1DimensionSpan[_bDimensionIndex - 1] = _dDimensionSpan;
            if (d1DimensionSpan[_bDimensionIndex - 1] > bSpanCountLimit)
                d1DimensionSpan[_bDimensionIndex - 1] = bSpanCountLimit;
            d1DimensionSigma[_bDimensionIndex - 1] = _dDimensionSigma;
        }
    }
}
```

```
public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            int value = 10;
            double[] d1Exemplar = new double[2];
            Clasterer Claster = new Clasterer();
            Claster.SetBorderGrid(-value, value, -value, value);
            Claster.Capasity = value - 1;
            Claster.vSetDimensions((byte)0, -value, value, Claster.Capasity - 1, 4);
            Claster.Hat = 4;

            //StreamWriter sw = new StreamWriter("D:\\TestFile.txt");
            StreamReader sr = new StreamReader("D:\\TestFile.txt");
            Random rand1 = new Random();
            TValues[] arrayIn = new TValues[30];
            string[] str;

            for (int i = 0; i < 30; i++)
            {
                str = sr.ReadLine().Split(new Char[] {','});

                arrayIn[i] = new TValues();
                arrayIn[i].X = Convert.ToInt32(str[0]);
                arrayIn[i].Y = Convert.ToInt32(str[1]);
            }

            //arrayIn = arrayIn.OrderBy(x => rand1.Next()).ToArray();
            //arrayIn = arrayIn.OrderBy(y => y.Y).ToArray();

            Claster.bClusterCount = 4;
            int numberCluster;
            for (int i = 0; i < 30; i++)
            {
                //x = rand1.Next(-value, value);
                //y = rand1.Next(-value, value);

                numberCluster = Claster.insertValue(arrayIn[i].X, arrayIn[i].Y);

                d1Exemplar[0] = arrayIn[i].X;
                d1Exemplar[1] = arrayIn[i].Y;
                Claster.iAddExemplar(d1Exemplar);

                Claster.b1ExemplarCluster[i] = (byte)numberCluster;
                //sw.WriteLine("{0}, {1}", x, y);
            }

            Claster.vCalcExemplarSpanMembership();

            Claster.vCalcClusterSpanMembership();

            Claster.dCalcClusterUtility();

            double CU = 0;
            for (byte e = 1; e <= 30; e++)
            {
                for (byte d = 1; d <= 2; d++)
                {
                    for (byte i = 1; i <= Claster.d1DimensionSpan[d - 1] + 1; i++)
                    {
                        CU += Claster.d3ExemplarSpanMembership[e - 1, d - 1, i - 1];
                    }
```

```
                }
                if (Claster.b1ExemplarCluster[e - 1] == 0)
                    textBox4.Text += e.ToString() +" "+ Claster.d2Exemplar[e-1,
0].ToString() + " " + Claster.d2Exemplar[e - 1, 1].ToString() + " " +
Claster.b1ExemplarCluster[e-1].ToString() + " " + Claster.dClusterUtility[e - 1].ToString()
+ "\n";
            }

            sr.Close();
            //sw.Close();

            for (int i = 0; i < Claster.arrClaster.Count; i++)
            {
                Series series = new Series();
                series.ChartType = SeriesChartType.Line;
                series.Name = "Кластер " + (i + 1).ToString();
                switch (i)
                {
                    case 0:
                        series.MarkerStyle = MarkerStyle.Circle;
                        break;
                    case 1:
                        series.MarkerStyle = MarkerStyle.Triangle;
                        break;
                    case 2:
                        series.MarkerStyle = MarkerStyle.Square;
                        break;
                    case 3:
                        series.MarkerStyle = MarkerStyle.Diamond;
                        break;
                    case 4:
                        series.MarkerStyle = MarkerStyle.Cross;
                        series.Color = Color.Black;
                        break;
                    case 5:
                        series.MarkerStyle = MarkerStyle.Star10;
                        series.Color = Color.Black;
                        break;
                    case 6:
                        series.MarkerStyle = MarkerStyle.Star4;
                        series.Color = Color.Black;
                        break;
                    default:
                        series.MarkerStyle = MarkerStyle.Star5;
                        series.Color = Color.Black;
                        break;
                }

                series.MarkerSize = 20;

                for (int j = 0; j < Claster.arrClaster[i].arrValues.Count; j++)
                        {
                    series.Points.AddXY(Claster.arrClaster[i].arrValues[j].X,
Claster.arrClaster[i].arrValues[j].Y);
                        }
                chart1.Series.Add(series);
            }


        }

        public Chart Graphic { get; set; }
    }
```

ПРИЛОЖЕНИЕ 2

Руководство пользователя

П. 2.1. Назначение программы

Программа предназначена для разбиения образцов на группы

Программа предназначена для выполнения следующих функций:

- кластеризация;
- классификация;
- сохранение результатов работы.

П.2.2. Условия выполнения программы

Требования к техническому обеспечению рабочего места пользователя системы:

ПК со следующими минимальными характеристиками:

а) процессор с частотой 1 ГГц и выше;

б) объем ОЗУ 512 МБ;

в) свободное пространство на жестком диске не менее 100 МБ;

г) видеокарта обеспечивающая разрешение не хуже 800x600x16;

д) монитор;

е) клавиатура;

ж) мышь.

П.2.3. Пуск программы

Запуск программы происходит при запуске файла COBWEB.EXE.

П.2.4. Команды оператора

На рисунке П.2.1. изображена главная форма программы, которая реализовывает разработанные алгоритмы.

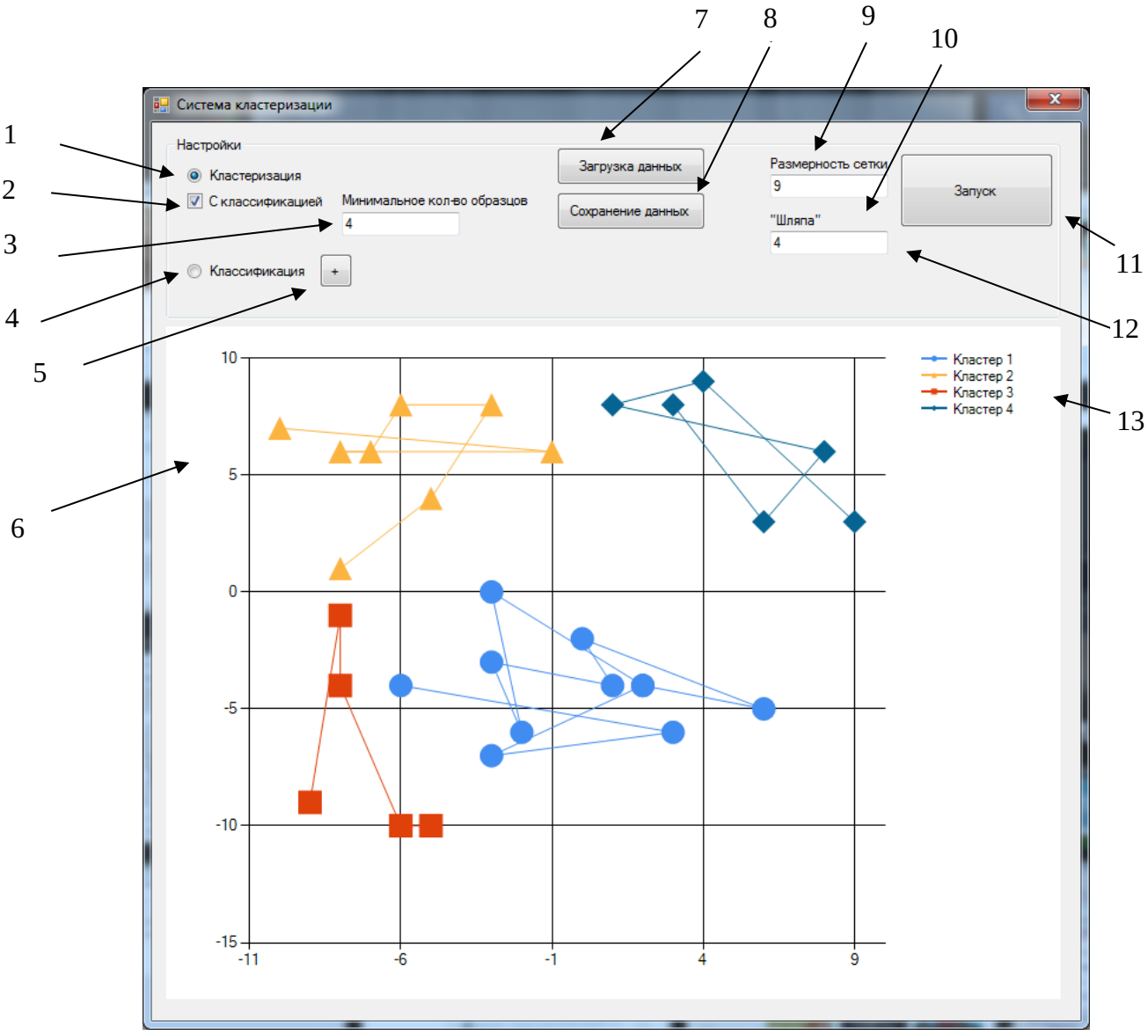

Рисунок П.2.1. – Главная форма приложения

Для начала работы программы необходимо загрузить данные кнопкой «7», задать размерность сетки и «Шляпу» - поля «9» и «10» соответственно. Нажать кнопку запуск «11».

После запуска можно сохранить полученные результаты с помощью кнопки «8», чтобы использовать полученную классификацию в последующих расчетах.

Если отметить галочку «2» и ввести минимальное количество образцов в

поле «3», то после обработки минимального количества образцов запустится классификация.

Результаты работы можно увидеть на графике «6» изображенном на рисунке П.2.2.

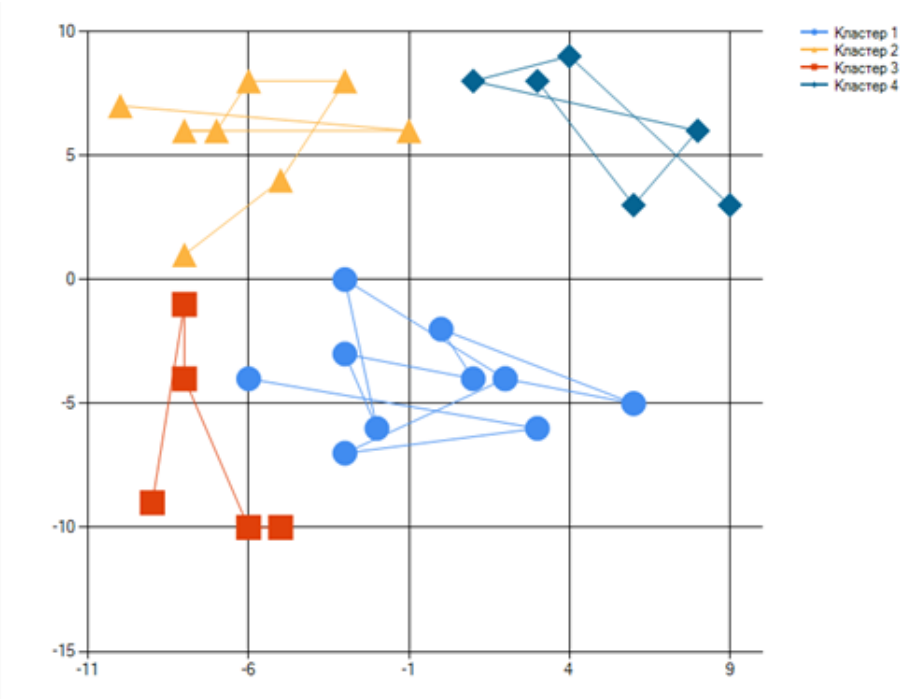

Рисунок П.2.2. – График точек

С помощью кнопки «5» можно добавить дополнительные экземпляры точек для классификации.

В процессе работы программы могут быть получены следующие сообщения об ошибке:

1) неверный формат данных (рис. П.2.3). Необходимо выбрать файл с верной структурой.

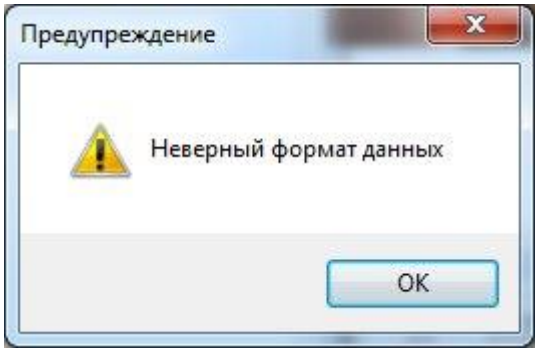

Рисунок П.2.3.

2) не выбран файл с входными данными, для запуска кластеризации необходимо указать входной файл (рис. П.2.4)

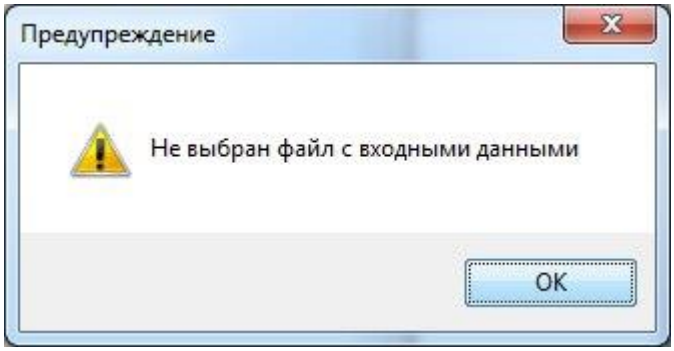

Рисунок П.2.4.

3) Необходимо указать входные параметры (рис. П.2.5).

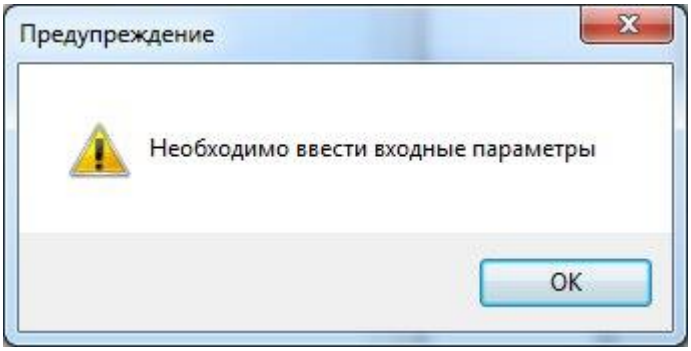

Рисунок П.2.5.