МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»
Институт математики физики и информационных технологий
(наименование института полностью)

Кафедра Прикладная математика и информатика
(наименование)

01.04.02 Прикладная математика и информатика
(код и наименование направления подготовки)

Математическое моделирование
(направленность (профиль))

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

на тему «Синтез новых классов в системах искусственного интеллекта, решающих задачу классификации»

Студент М. Р. Мирзоева
(И.О. Фамилия) (личная подпись)

Научный руководитель канд. пед. наук., доцент, О.М. Гущина
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

**Оглавление**

# Введение

В последние несколько лет популярность разработок из области искусственного интеллекта достигла своего пика. На сегодняшний день искусственный интеллект используется практически во всех отраслях начиная от бизнеса до медицины. Интеллектуальные алгоритмы, в том числе алгоритмы машинного обучения, решают широкий спектр задач, в основном это задачи кластеризации, классификации и регрессионного анализа.

При решении задачи классификации, алгоритм определяет принадлежность объекта, поданного ему на вход, к одному классу из множества заранее определенных классов. Однако возможны ситуации, когда данные на входе не принадлежат ни к одному из тренировочных классов или принадлежат сразу нескольким классам одновременно с условно равными долями вероятностей. В таких ситуациях возникает возможность для синтеза нового, неизвестного ранее классификатору класса, который может быть создан на основе известных ему классов.

**Гипотеза исследования:** математическая модель алгоритма классификации KNN может быть дополнена правилами синтеза новых классов.

**Научная проблема** – возможность синтеза новых классов для интеллектуальных алгоритмов, решающих задачу классификации.

**Объект исследования** – математическая модель работы алгоритмов, решающих задачу классификации, **предмет исследования** – способы синтеза новых классов в условиях, задаваемых рассматриваемой математической моделью.

**Цель исследования** – разработка возможных правил синтеза новых классов для алгоритмов, решающих задачу классификации, используя имеющиеся данные в обучающей выборке.

Для достижения поставленной цели требуется решение следующих **задач:**

– Рассмотреть историю вопроса реализации творческих функций искусственным интеллектом.

– Обосновать современность и актуальность решаемой проблемы, ее значимость.

– Провести поиск и анализ возможных решений поставленной проблемы, предложенных в ранних исследованиях.

– Формализовать задачу синтеза классов.

– Разработать математическую модель предлагаемого метода решения проблемы**.**

– Реализовать предлагаемую математическую модель.

– Провести эмпирическое исследование разработанной математической модели**.**

– Оформить прикладные выводы по проделанной работе.

Задачу синтеза новых классов непосредственно можно считать одним из проявлений творческих функций интеллекта, что естественно для интеллекта человеческого, но на текущий момент недоступно интеллекту искусственному. В упрощенной формулировке, классификатор создаст и запомнит новую сущность (класс), опираясь лишь на свои знания, полученные в процессе обучения на выборке, где данная сущность (класс) не были представлены в явном виде.

Тема является актуальной, так как машинное обучение находит все большее применение в современном мире. Сейчас уже очевидно, что искусственный интеллект будет все больше применятся в решении прикладных задач. Решение поставленной задачи позволит приблизить существующие алгоритмы машинного обучения к генерации собственных знаний.

Работа изложена на 76 страницах, содержит 34 рисунков.

# Глава 1 Генерация новых знаний с использованием методов машинного обучения

## 1.1 Современное развитие искусственного интеллекта

Мышление – способность рассуждать, которая представляет из себя процесс отражения объективной действительности в представлениях, суждениях, понятиях. По сути мышление – это непрерывный процесс построения особых моделей, с помощью которых происходит познание.

Отсутствие у животных мышления говорит об их неспособности обучаться целесообразному поведению по средствам наблюдений [7]. Обучение мышлению не может происходить без познания того, что из себя представляет мышление.

Автор термина “Artifical Intelligence” («искусственный интеллект») Д. Маккарти отмечает [7] недостаточно высокий уровень научной изученности природы человеческого интеллекта. Исходя из этого, вопрос об адекватности модели, заложенной в задаче имитации или воссоздания умственной деятельности человека, остается открытым.

Под искусственным интеллектом понимаются системы средств, воспроизводящих «определенные функции человеческого мышления» [5]. Такая имитация мыслительной деятельности заставляет человека думать о том, что перед ним не вычислительное устройство, а разумное существо [8].

Понятие творчества тесно связано с понятием мышления, однако их нельзя отождествлять, так как мышление представляет собой один из видов познания, а творчество же возможно не только в сфере познания, но и в различных направлениях искусства: живопись, пение, литература и т.д. [8]. Любое творчество – это новый взгляд на проблему, разрушение стереотипа, который был создан мышлением для сокращения мыслительной деятельности человека. При стереотипном мышлении человек не уделяет внимание незначительным моментам, что позволяет ему без особых умственных усилий

обрабатывать операции. Искусственный интеллект, в отличие от человека, способен обрабатывать большие объемы данных, однако это не является творчеством. Создать что-то новое способен человек с особенностями его мышления, а искусственный интеллект только помогает человеку овладеть имеющейся информацией. Интеллектуальные системы способны выполнять функции, которые раньше считались творческими, но мышление человека остается областью подлинного творчества.

Моделирование человеческого мышления крайне сложная задача [11], она требует детальной оценки характеристик мозга и является практически невыполнимой. Создание искусственно интеллектуальной модели, которая бы основывалась на способности творить, стало бы одним из самых главных прорывов в этой области. Творчество является фундаментальной чертой человеческого мозга, это особенность человеческого интеллекта в целом, основанное на способностях и умениях таких как: запоминание, восприятие, аналитическое мышление. Помимо когнитивного измерения, которое генерирует новые понятия, творчество включает в себя и мотивацию, эмоции, личностный фактор.

Также в работе [11] сказано: «Творческая идея, прежде всего, это новый, удивительный и ценный образ представления информации. Способность производить новинки первого рода можно условно назвать «P-творчеством», последнего же – «H-творчеством». «P-творчество» является более фундаментальным понятием, а «H-творчество» представляет собой всего лишь частный случай, его производную»

Современные модели искусственного интеллекта ориентированы только на когнитивные измерения, это связано с тем, что специалисты пришли к выводу, что если удастся смоделировать эту способность правильным образом, то дедуктивное умозаключение воспроизведет образы второго рода.

Существует три основных типа творчества, предполагающих различные способы генерации новых идей. Первый тип, «комбинационный», предполагает сочетание хорошо знакомых концепций. Второй и третий типы

предполагают исследование структурированных концептуальных пространств (соответственно, «исследовательский») и трансформацию некоего измерения пространства («трансформационный») [3], [19]. Компьютерные модели творчества включают в себя примеры всех трех типов, но на данном этапе второй тип является наиболее успешным, так как первый и третий типы более сложны для воспроизведения. Приближение к богатству ассоциативной памяти человека, а также проблема в идентификации наших ценностей и выражения их в вычислимой форме являются основными трудностями данных типов творчества.

Изучение «комбинационного» творчества производится путем поиска ассоциаций и требует взаимосвязанной базы знаний в качестве основы. Задача поиска ассоциации достаточно проста, однако сама ассоциация не всегда может быть уместна в том или ином контексте. Характер и структура ассоциативной связи имеют большое значение, оценка оригинальности ассоциативных комбинаций оценивается системой искусственного интеллекта. Структурированные и концептуальные пространства для «исследовательского» и «трансформационного» типов творчества могут быть описаны вычислительными концепция, к примеру классификация на основе правил «если-то».

На сегодняшний день большинство современных моделей искусственного интеллекта не преобразовывают, а только исследуют пространства, это связано с тем, что результирующие структуры измененного пространства будут являться новаторскими, но не творческими, а значит не будут иметь никакой ценности. Творческая инициатива все еще принадлежит человеку, однако возможности усовершенствования искусственного интеллекта до уровня синтеза новых концептуальных идей и образов существуют.

## 1.2 Общая модель задачи классификации и возможные подходы к решению задачи синтеза новых классов

Анализ научных публикаций по тематикам, связанным с областью исследуемой проблемы, показал, что реализовывать творческую функцию мышления, то есть создавать совершенно новые объекты, на текущий момент способен только человеческий интеллект [2]. Системы искусственного интеллекта, создаваемые человеком, с такой задачей справиться в настоящее время еще не способны, это связано с тем, что они в большинстве случаев работают строго в рамках знаний, полученных ими на заданной обучающей выборке.

В задачах классификации обучающая выборка состоит из размеченного множества пар «элемент-класс», в которых элемент описывается определенными признаками, и для него проставляется метка принадлежности к конкретному классу.

Решив задачу обучения по прецедентам (1) можно получить классификатор в общем виде:

$$\langle X, Y, y^*, X^M \rangle \quad (1)$$

где X – множество классифицируемых объектов;

Y – множество состоящее из меток класса;

$y^*: X \to Y$ – целевая зависимость, значения которой известны только на элементах обучающей выборки $X^M = (x_i, y_i)_{i=1}^M$, в которых $x_i$ – элементы обучающей выборки, а $y_i = y^*(x_i)$ –метка класса.

Алгоритм $\alpha: X \to Y$, аппроксимирующий целевую зависимость на всем пространстве X, будет являться решением данной задачи [4].

При решении задачи классификации системы искусственного интеллекта определяют принадлежность объекта, поданного на вход классификатора, к одному классу из множества заранее определенных

классов. Однако возможны ситуации, когда данные на входе не принадлежат ни к одному из тренировочных классов [1]. Так же могут быть ситуации, когда объект с равной степенью вероятности принадлежит к нескольким классам одновременно, так как системы искусственного интеллекта не способны на творчество, в частности на создание нового класса. Из этого возникает проблема, которую можно сформулировать следующим образом: как можно выйти за пределы рамок, задаваемых содержимым обучающей выборки.

Рассмотрим пример. При простейшей классификации на «лошадь» и «не лошадь» классификатор, встретив зебру, вполне может решить, что это на 50% «лошадь», и 50% –«не лошадь». Человек, впервые встретив зебру, благодаря особенностям своего мышления стал бы исследовать этот вопрос, он бы строил гипотезы («специфичный окрас лошади» или «лошадь испачкалась»), и на основании единичного случая не стал бы изобретать новое понятие «зебра». С точки зрения искусственного интеллекта, единожды встретив зебру, можно этот факт проигнорировать, однако если зебр становится слишком много, стоит поднять вопрос о синтезе совершенно нового класса «зебра».

Рассмотрим также пример с классификацией цвета. Допустим в обучающей выборке есть классы цветов: красный и синий, а в процессе работы классификатор встречает фиолетовый – возникает вопрос, можем ли мы заставить классификатор понять, что фиолетовый это смесь красного и синего, т.е. «класс1» + «класс2».

Синтезировать новые классы мы можем либо, используя один базовый класс (лошадь – зебра), либо несколько базовых классов (красный + синий = фиолетовый).

Люди в силу своего мышления встречая подобные единичные случаи не «создают новый класс», так как память у человека «не резиновая», чего нельзя сказать о машинах, у которых проблем с этим меньше [30]. Возможно, машинам следует создавать новую сущность каждый раз при встрече объекта «непонятного класса», просто эта новая сущность будет обладать «меньшей

важностью», однако при частом появлении подобных объектов, «важность» новой сущности будет расти.

В любом случае, задача создания определенных правил синтеза новых классов представляется вполне реализуемой. Данные правила в своей основе будут опираться на определенный математический аппарат, а также должны быть созданы с учетом имеющихся особенностей работы алгоритмов, решающих задачи классификации [6]. Не исключено, что для различных алгоритмов потребуется переработка данных правил, то есть универсальное решение, включающее в себя конкретные правила синтеза сущностей (классов), может быть недостижимо. Однако универсальное решение может представлять собой обобщенные, строго формализованные математические формулировки и модели, которые будут являться основной для реализации конкретных методов синтеза новых классов под определенные алгоритмы классификации.

### 1.3 Потенциальная возможность синтеза новых классов в конкретных алгоритмах классификации

Существует множество алгоритмов, решающих задачу классификации, которые делятся на две группы:

– Алгоритмы, работающие без учителя, самостоятельно пытаются выполнить поставленную перед ними задачу, без вмешательства человека (алгоритмы кластеризации и тематического моделирования).

– Для алгоритмов, подразумевающих обучение с учителем, требуется провести обучение на обучающей выборке. К этой группе относят алгоритмы, основанные на деревьях решений, нейронные сети и другие, как правило решающие задачи регрессионного анализа и классификации.

К классическим алгоритмам, решающим задачи классификации (в том числе текстовых документов) относят:

– Деревья решений.

– Алгоритм K ближайших соседей (KNN).
– Метод опорных векторов (Support Vector Machine).
– Нейронные сети.
– Ансамблевые методы классификации.

Дерево принятия решений (также «дерево классификации», «дерево регрессии», «дерево решений») – это схематическое представление проблемы принятия решения в виде графа, позволяющее получить окончательное решение о классификации объекта [9], [20]. Основное назначение деревьев решений – решение задач регрессии и классификации. Деревья решений также используется в области статистики и анализа данных для прогнозных моделей.

Существует несколько алгоритмов построения деревьев решений, которые отличаются используемыми методами и математическим аппаратом. Среди самых распространенных алгоритмов выделяют следующие:

– Алгоритм ID3.
– Алгоритм C4.5.
– Алгоритм CART.

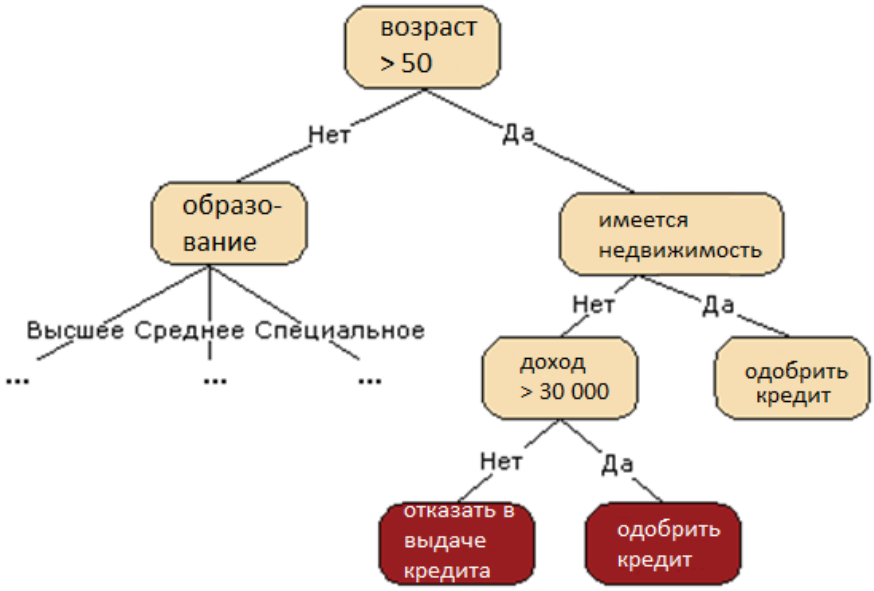


Рисунок 1 – Пример построения дерева решений для выдачи кредита заемщику

Алгоритм kNN (англ. “k-Nearest Neighbors” – «k ближайших соседей») – простейший алгоритм классификации и регрессии. Особенность алгоритма заключается в том, что в процессе обучения он просто сохраняет тренировочные данные. В этом и есть отличие от «активных» классификаторов, которые создают определенную классификационную модель, обучаясь на размеченных элементах обучающей выборке.

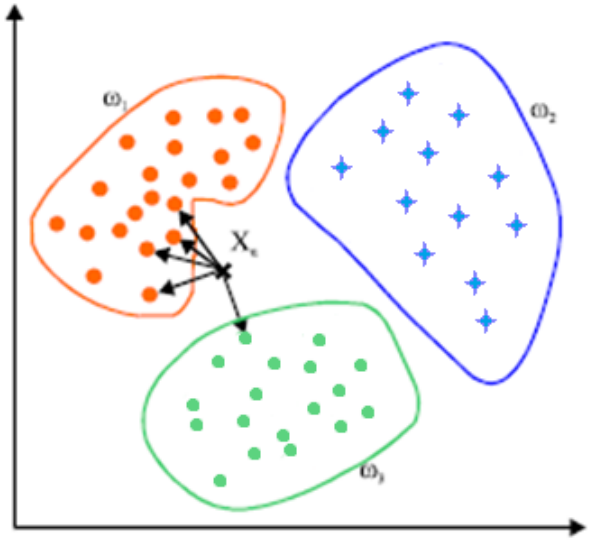


Рисунок 2 – Пример разделения объектов на множества алгоритмом kNN

Метод опорных векторов (Support Vector Machine, SVM) – линейный алгоритм, который используется в задачах регрессии и классификации и относится к бинарным классификаторам, хотя существуют способы его применения для задач мультиклассификации [29]. Слабой стороной метода опорных векторов является необходимость выбора ядра и плохая интерпретируемость модели.

Алгоритмы построения искусственных нейронных сетей нашли широкое применение в решении различных задач, в том числе при классификации данных.

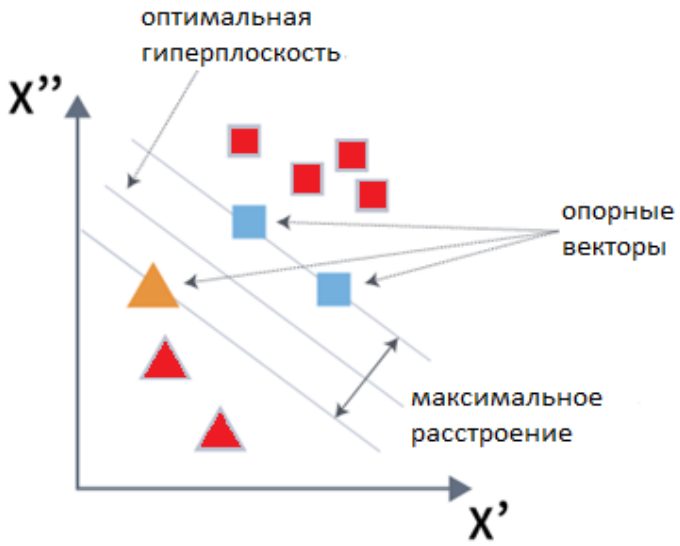


Рисунок 3 – Схематичный пример построения гиперплоскости в работе метода опорных векторов

Нейронная сеть представляет собой набор нейронов, объединенных в несколько слоев. В классической модели присутствует входной слой, выходной слой и как минимум один скрытый слой (рисунок 4). Входной сигнал возбуждает нейроны входного слоя, которые затем передают сигнал дальше по цепочке синапсов [28]. Нейроны выходного слоя формируют выходное значение – реакцию нейронной сети на поданный ей возбуждающий сигнал.

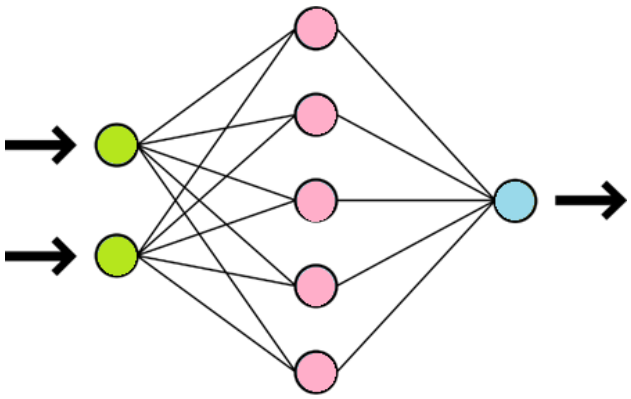

Рисунок 4 – Простейшая модель искусственной нейронной сети

Общая идея ансамблевых методов классификации заключается в том, что с помощью нескольких слабых классификаторов (точность классификации не на много лучше простого «угадывания») создается решающее правило, таким образом создается один сильный мета-классификатор.

Среди ансамблевых методов классификации бэггинг (bagging, bootstrap aggregating), бустинг (boosting) и стекинг (stacking) являются наиболее популярными.

Бэггинг удобно использовать при отсутствии большой обучающей выборки. Такой метод объединяет результаты предсказания различных классификаторов, обученных на случайных подмножествах

Бустинг – последовательного строит композиции алгоритмов машинного обучения, где каждый следующий алгоритм стремится компенсировать недостатки композиции всех предыдущих алгоритмов [12]. Его считается жадным алгоритмом построения композиции алгоритмов.

Стекинг – это еще один из способов объединения классификаторов, в отличие от бустинга и бэггинга стекинг объединяет классификаторы «разной природы».

Несмотря на существенные различия в работе алгоритмов классификации, все они основаны на одном и том же принципе, указанном в (1). Это значит, что при решении поставленной ранее проблемы потенциально каждый из перечисленных алгоритмов может быть дополнен определенными методами синтеза новых классов, что позволит расширить их функционал. Однако прежде всего требуется формализация самих возможных правил синтеза новых классов.

# Глава 2 Синтез новых классов для алгоритма KNN

## 2.1 Алгоритм KNN и его модификация для синтеза новых классов на основе ошибки классификации

Опишем математическую модель алгоритма K ближайших соседей с возможностью создания новых классов на основе имеющихся. Пусть имеется обучающая выборка пар (описание объекта, метка класса) (1) [4].

$$X^m = \{(x_1, y_1), \dots, (x_m, y_m)\}, \quad (1)$$

где $x_i$ – описание объекта обучающей выборки;

$y_i$ – метка класса для объекта $x_i$.

Метрика расстояния – это функция, определяющая расстояние между классифицируемым элементом и элементами обучающего множества. Существует порядка 11 метрик [18], [27] применимых для алгоритма KNN. Рассмотрим только три из них, так как они являются наиболее распространенными:

– метрика Евклида;
– расстояние Хемминга;
– расстояние городских кварталов (метрика Манхэттена).

Метрика Евклида (2) находит расстояние между двумя объектами и является частным случаем метрики Минковского, где p = 2.

$$d_{st}^2 = (\mathrm{x_s} - \mathrm{y_t})(\mathrm{x_s} - \mathrm{y_t})', \quad (2)$$

где (здесь и далее) $x_s$ и $y_t$ – векторы, для которых определяется расстояние.

Расстояние Хемминга также является метрикой различия объектов одинаковой размерности, определяет процент различных координат двух векторов и вычисляется по формуле (3).

$$d_{st} = \left(\frac{\#(x_{sj} \neq y_{tj})}{n}\right) \quad (3)$$

Метрика Манхэттена определяет расстояние между двумя объектами по сумме абсолютной разности декартовых координат и определяется по формуле (4) и является частным случаем метрики Минковского, где p=1.

$$d_{st} = \sum_{j=1}^{n} |x_{sj} - y_{tj}| \quad (4)$$

Для вышеприведенных метрик расстояния истинно следующее утверждение: чем меньше значение функции расстояния, тем более схожими являются рассматриваемые объекты [10] . В контексте алгоритма KNN, более схожими будут являться классифицируемый объект и объект из обучающей выборки.

Расположим объекты обучающей выборки $x_i$ в порядке возрастания расстояний до классифицируемого объекта u (5).

$$d(u, x_{1,u}) \leq d(u, x_{2,u}) \leq \cdots \leq d(u, x_{m,u}) \quad (5)$$

где $x_{i,u}$ является i-ым соседом для классифицируемого объекта u из обучающей выборки, аналогичное обозначение введем и для ответа на i-ом соседе: $y_{i,u}$. Таким образом, объект u порождает перенумерацию выборки.

Более общий вид алгоритма ближайших соседей можно описать как (6).

$$a(u) = \arg\max_{y \in Y} \sum_{i=1}^{k} [x_{i,u} = y] w(i, u) \quad (6)$$

Положим, что w(i, u) – весовая функция, неотрицательно определена и не возрастает по i, которая оценивает степень важности i-го соседа при классификации объекта u. Если по-разному задавать весовую функцию, то можно получать различные варианты метода ближайших соседей.

Простейший метод ближайшего соседа можно получить, если весовая функция принимает значение (7):

$$w(i,u) = [i = 1] \quad (7)$$

Метод k ближайших соседей будет получен при (8):

$$w(i,u) = [i \leq k] \quad (8)$$

А при (9) – метод k экспоненциально взвешенных соседей (предполагается, что $q < 1$):

$$w(i,u) = [i \leq k] * q^i \quad (9)$$

Далее будет рассматриваться весовая функцию (8) – метод k ближайших соседей.

После того, как алгоритм нашел k ближайших соседей для классифицируемого объекта, необходимо найти суммарную ошибку классификации E. Для этого для каждого класса k из множества всех классов обучающей выборки K ($k \in K$) суммируется расстояние от классифицируемого объекта u до каждого объекта x класса k ($x \in k$), после чего полученное значение делится на количество объектов в классе. Таким образом, имеем среднее расстояние от классифицируемого объекта до каждого класса, или, иными словами, ошибку классификации объекта u для k-ого класса $E_k$ (10). Чем больше значение $E_k$, тем «дальше» объект от этого класса.

– $$E_k = \frac{\sum_{i=1}^{len(k)}(d(u,x_i))}{len(k)}\ , \quad (10)$$

где $d(u, x_i)$ – используемая метрика расстояния;

len(k) – количество объектов класса k в обучающей выборке;

$x_i$ – элемент класса k из обучающей выборки;

u – классифицируемый элемент.

После подсчета всех $E_k$, простейший способ для определения класса нового элемента u – это выбор такого класса k, у которого $E_k$ будет минимальна (11):

$$min_{k \in K}(E_k) \tag{11}$$

Однако может возникнуть ситуация, при которой значение $E_k$ будет одинаковое для нескольких классов (примем их за $k_m$ и $k_t$) и при этом минимальным, либо будет отличаться незначительно, в пределах задаваемой погрешности [26]. В таком случае, при отнесении классифицируемого элемента к одному из классов $k_m$ или $k_t$, алгоритм может допустить ошибку классификации, так как объект может не относиться ни к одному из данных классов, либо к нескольким классам одновременно. Схожая проблема может возникнуть также из-за недостаточного уровня «доверия», т.е. в ситуации, когда нет достаточно малого значения $E_k$, удовлетворяющего (11).

В таком случае представляется разумным создать новый класс $k_{new}$, который будет представлять собой симбиоз из классов $k_m$ и $k_t$. Разницу между классифицируемым объектом и средним расстоянием до классов, на основе которых будет создан новый $k_{new}$ класс, следует задавать самостоятельно. Например, если разница между двумя классами (12) составляет $\xi$, алгоритм будет синтезировать новый класс.

$$|E_t - E_s| \leq \xi \tag{12}$$

Определить оптимальное значение «порога» создания нового класса возможно только экспериментальным путем.

## 2.2 Моделирование работы модифицированного алгоритма KNN на основе ошибки классификации

Для более наглядной демонстрации работы модифицированного алгоритма k-ближайших соседей, алгоритм будет решать задачу классификаций точек на плоскости.

Пусть $P = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ – множество точек на плоскости, где $n$ – число точек на плоскости, $C = \{c_1, c_2, \dots, c_m\}$ – множество классов, где $m$ – число классов. Тогда $z: P \to C$ будет являться отображением множества $P$, на множество $C$. В данной задаче нам известно значение отображения $z$. Точки, принадлежащие определенному классу, будут задаваться следующим образом. Определим $g_i \subseteq P$, где $i = 1 \dots m$ – индекс класса из $C$, а $g_i$ подмножество множества $P$ относящихся к классу $c_i$. Каждый набор точек $g_i$ будет задаваться случайным образом. Случайным образом на плоскости выбирается центр множества точек $center_i$ и относительно него в произвольном направлении генерируется точка $p_j$, где $j = 1 \dots |g_i|$, с распределением Гаусса и среднеквадратичным отклонением $\sigma$. Будем считать, что мы уже имеем идеально сбалансированное множество точек. В таком случае число общее число точек $n$ будет задаваться как $m * s$, $s \in \mathbb{N}^*$.

Приведем пример того, как моделируются эти точки на плоскости (рисунок 5). Одинаковым цветом заданы точки, принадлежащие одному классу. На рисунке изображены 4 класса точек по 10 в каждом классе, сгенерированные на плоскости и $\sigma = 0.3$.

Обучим классификатор для k = 4 и посмотрим на то, как разделилась плоскость. На рисунке 2 изображено разбиение множества точек плоскости на классы. Сетка рисуется с шагом h = 0.1 (рисунок 6).

Точки обучающего множества точек выделены более ярким цветом на фоне точек, получившихся в результате их классификации.

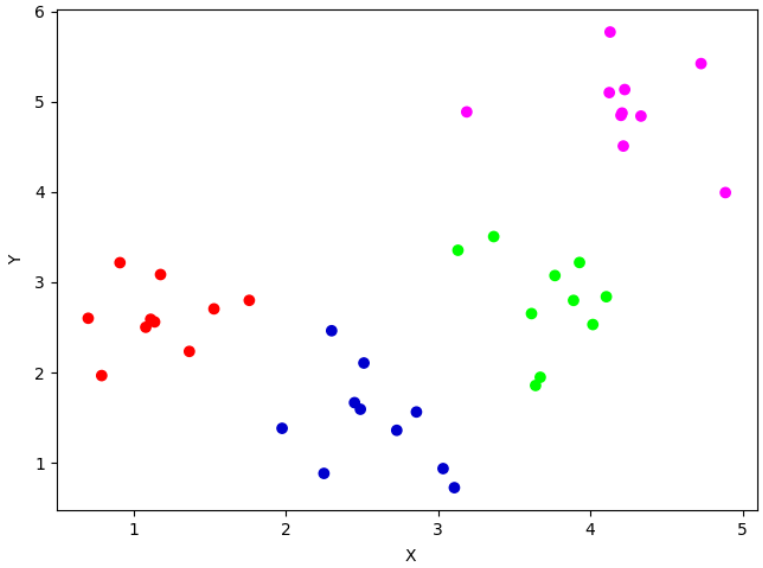


Рисунок 5 – Пример расположения точек, принадлежащих разным классам, на плоскости

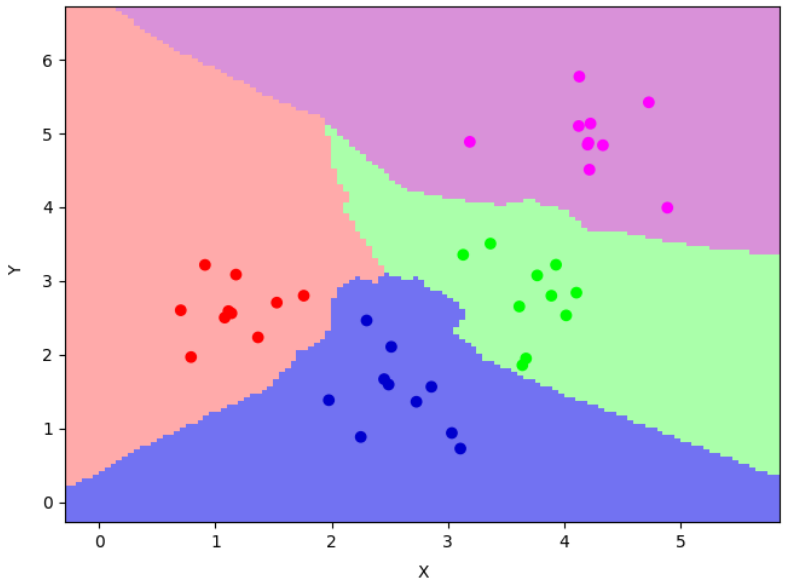


Рисунок 6 – Получившееся разбиение точек плоскости

Как можно заметить мы наблюдаем очень четкую границу между множествами принадлежащим разным классам. Выделим эту границу золотым цветом и зададим ей ширину равную $b = 0.1$ (рисунок 7).

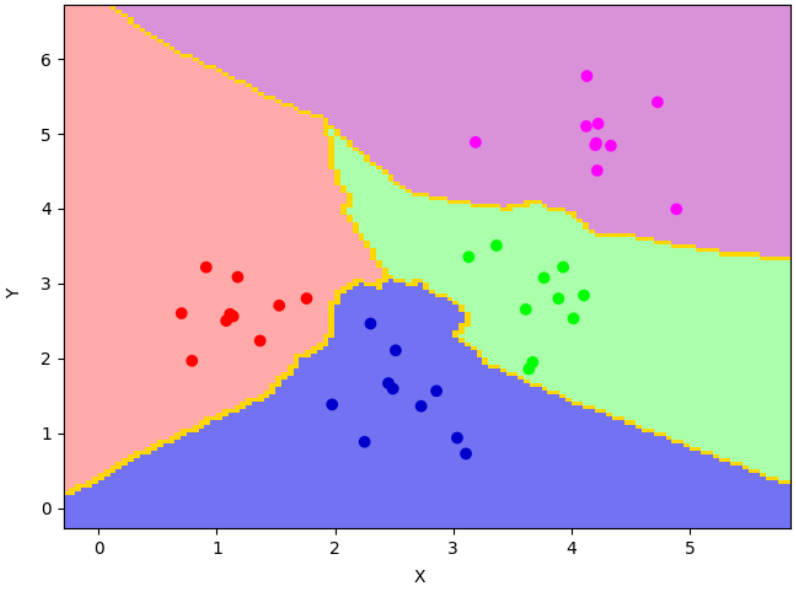


Рисунок 7 – Выделенная граница между множествами принадлежащих разным классам

Как можно заметить на границах множеств, принадлежащих разным классам, будет высокая вероятность ошибки и, помимо этого, на этих границах, могут присутствовать точки, которые не принадлежат ни одному из приведенных классов. Особенно это касается таких областей, на которых тренировочные данные вообще отсутствуют. На рисунке 3 это левая верхняя часть рисунка.

Задача состоит в том, чтобы научиться фиксировать элементы множества, которые могут принадлежать совершенно другому классу. Не представленному среди уже существующих.

Начнем поэтапное построение модифицированного алгоритма. Как уже говорилось ранее мы будем вводить некую величину $\xi$ которая будет определять порог создания нового класса. Решение о создании нового класса мы принимаем по формуле 12.

$E_t$ и $E_s$ в формуле 12 являются ошибками классификации для двух множеств, имеющих наименьшее их значение.

Графически такие области в двумерном пространстве будут выглядеть как некие границы между множествами [13], [25]. Следовательно, чем выше значение $\xi$ тем граница между областями будет шире и тем больше вероятность образования нового класса. Ниже на рисунке 4 приведена иллюстрация границ между классами при различных значениях $\xi = 0{,}05$, $\xi = 0{,}1$, $\xi = 0{,}2$ и $\xi = 0{,}5$ соответственно (рисунок 8). Желтым цветом выделены области, на которых разница между ошибками классификации двух множеств $E_t$ и $E_s$ ниже, чем порог $\xi$.

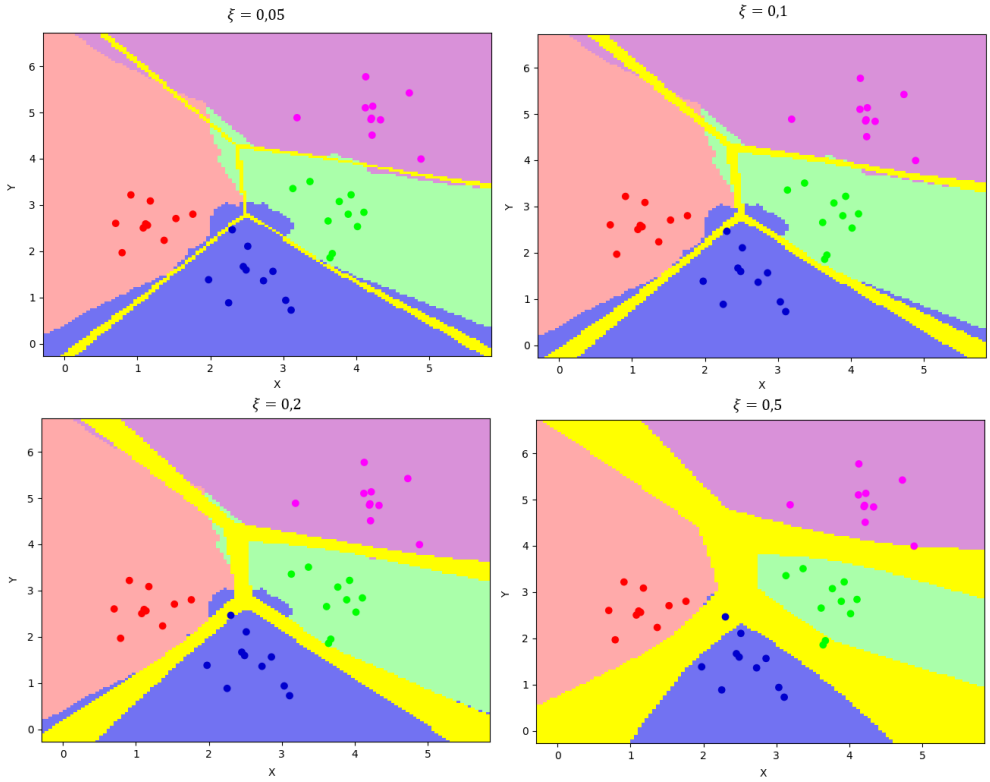


Рисунок 8 – Области, подходящие под формулу 1 при различных значениях $\xi$

Как можно заметить, чем дальше точки расположены от тренировочных данных, тем шире жёлтая область. Это оправданно тем, что значение, расположенное дальше от кластеров, будет являться аномальным, а, следовательно, велика вероятность что оно принадлежит новому классу.

Благодаря уже такой модификации мы можем фиксировать и находить объекты, принадлежащие другим классам. Далее мы рассмотрим ещё одну модификацию алгоритма, основанную на построении выпуклой оболочки множества.

### 2.3 Выпуклая оболочка множества

Выпуклой оболочкой называется множество $X$ называется наименьшее выпуклое множество, содержащее $X$. Под минимальным выпуклым множеством подразумевается такое множество, что оно содержится в любом другом выпуклом множестве также содержащие множество $X$.

Выпуклым множеством называется такое множество точек, что если взять любые две точки этого множества, то все точки, лежащие на полученном отрезке, тоже принадлежат этому множеству [17].

Приведем пример выпуклой оболочки для двумерного пространства. На рисунке 9 сплошной линией обозначена минимальная выпуклая оболочка, содержащая все точки.

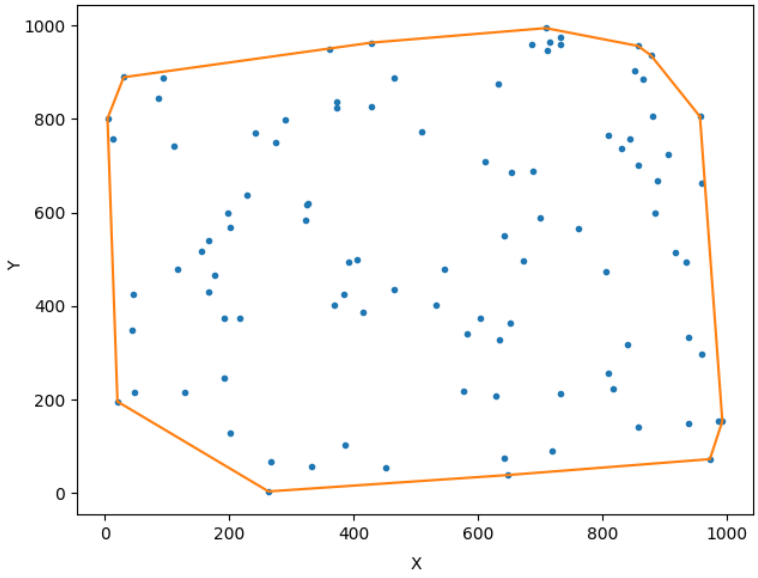


Рисунок 9 - Минимальная выпуклая оболочка

Как видно на рисунке 9 красным цветом выделена минимальная выпуклая оболочка для 100 точек, случайно расположенных на плоскости.

Таким образом мы можем использовать выпуклую оболочку для определения области, которую занимает каждый класс.

### 2.4 Алгоритм Грэхема

Алгоритм Грэхема был предложен в 1972 году как алгоритм поиска наименьшего выпуклого многоугольника, содержащий все заданные на плоскости точки. Алгоритм Грэхема позволяет найти минимальную выпуклую оболочку за время $O(N * \log(N))$, где $N$ – число точек на плоскости. Алгоритм использует только операции сложения, умножения и сравнения, а также доказано, что не существует алгоритма с лучшей асимптотикой чем у рассматриваемого алгоритма. Конечно, алгоритм имеет один существенный недостаток. Он не подходит для случаев параллельной обработки точек и для случая работы алгоритма в online. Для практического применения эти недостатки критичны, так как необходимо иметь возможность увеличивать быстродействие алгоритма за счет параллельных вычислений. В нашем случае мы будем искать минимальную выпуклую оболочку для каждого класса, а, следовательно, можно m классов распределить на p процессоров и считать независимо [24]. Проблема с невозможностью работать в online сильно влияет на производительность, но в дальнейшем планируется заменить алгоритм Грэхема на другой алгоритм.

Разберемся в том, как работает алгоритм Грэхема. Он состроит из трех этапов.

Первым этапом является поиск точки $st = (x_1, x_2)$, которая гарантированно входит в минимальную выпуклую оболочку. Для этого возьмем самую левую точку по координате $x_1$ то есть с самым наименьшим значением $x_1$.

Вторым этапом необходимо отсортировать все точки на плоскости по степени их левизны относительно стартовой точки $st$. Это можно сделать с помощью знаковой площади треугольника.

Пусть даны три точки $pt_1, pt_2, pt_3$. Эти точки образуют треугольник и знак площади этого треугольника будет говорить нам о типе поворота (правый, левый). Для этого воспользуемся определением псевдоскалярного произведения векторов. Оно будет равно удвоенной знаковой площади треугольника (формула 13).

$$a \wedge b = |a||b|\sin \angle(\mathrm{a}, \mathrm{b}) = 2\mathrm{S} \tag{13}$$

Псевдоскалярное произведение можно вычислить и по формуле 14 как величину определителя следующей матрицы.

$$2S = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \tag{14}$$

Если раскрыть определитель, то получится формула 15.

$$2S = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2) \tag{15}$$

Далее выполним группировку третьего слагаемого с первым и вторым. Получится формула 16.

$$2S = (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1) \tag{16}$$

Теперь перепишем формулу 5 в матричном виде и получим следующий определитель (формула 17).

$$2S = \begin{vmatrix} x_2 - x_1 & y_2 - y_1 \\ x_3 - x_1 & y_3 - y_1 \end{vmatrix} \quad (17)$$

По получившейся формуле мы сможем определять значение знаковой площади треугольника, а, следовательно, отсортировать вершины по величине поворота для точки $st$.

Третьем этапом будет соединение всех точек в отсортированном на предыдущем этапе порядке. Начальной точкой будет являться точка $st$. Далее мы будем использовать структуру данных стек, в которую будем поочередно добавлять вершины. На начальном этапе стек содержит вершину $st$ и вершину $s_1$, где $s$ вектор отсортированных вершин.

Теперь поочередно добавляя вершины в стек, мы рассматриваем знаковую площадь трех вершин на вершине стека. Если знаковая площадь положительная, то мы оставляем вершину иначе извлекаем из стека.

## 2.5 Модифицированный алгоритм KNN с использованием выпуклой оболочки

Алгоритм Грэхема будет применяться для построения минимальной выпуклой оболочки для элементов каждого класса. Более формально для каждого класса из множества $C = \{c_1, c_2, \dots, c_m\}$ будет построено множество $h_i$, где $i$ – это индекс класса из множества $C$. Результат такой модификации изображен на рисунке 10.

Если увеличить число элементов в классе до 100, то получим более плавные границы множеств. На рисунке 11 изображены полученные множества.

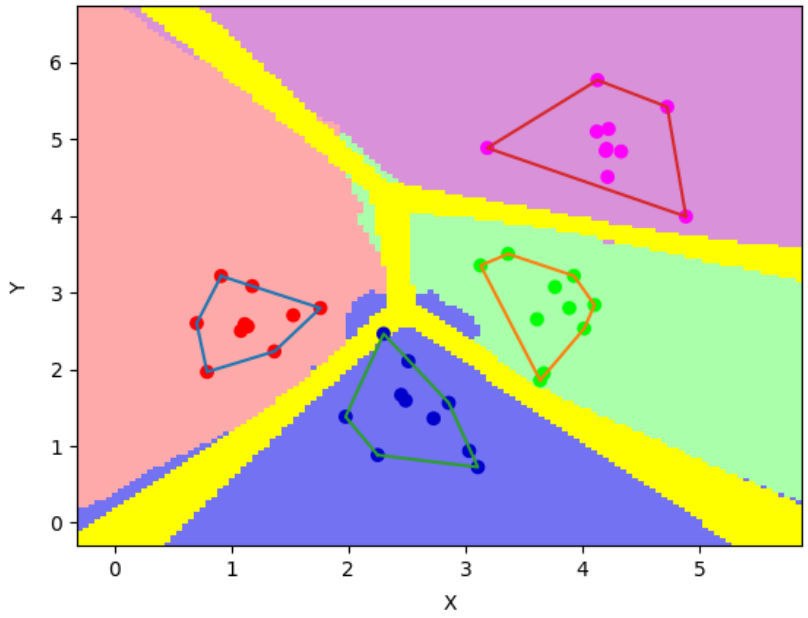


Рисунок 10 – Применение алгоритма Грэхема для элементов каждого класса

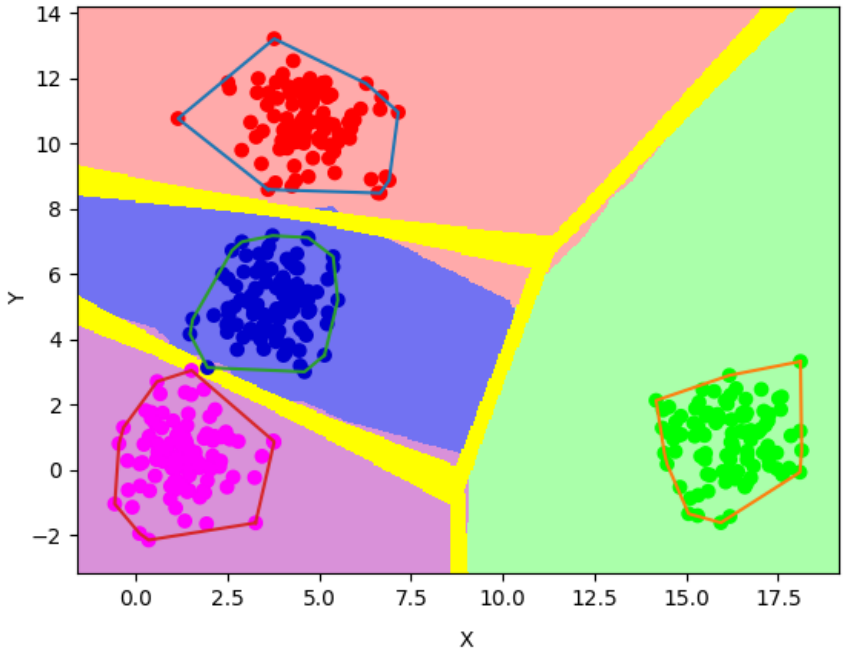


Рисунок 11 – Применение алгоритма Грэхема для 100 элементов каждого класса

Таким образом ограничили зоны каждых классов, что позволяет находить и синтезировать новые классы не только на границах классов, но в

множестве, которое построил классификатор k-ближайших соседей для одного класса. Далее будет рассмотрен ещё один вариант определения зоны принадлежности одному из классно классов на основе алгоритма MCD.

## 2.6 Алгоритм MCD

Детерминант минимального ковариата (MCD) является одним из первых аффинных равнозначных и высоконадежных оценщиков многомерного расположения и рассеяния (Rousseeuw, 1984, 1985). Устойчивость к внешним наблюдениям делает MCD очень полезным для обнаружения выбросов (отклонений). Хотя он уже был представлен в 1984 году, его основное применение началось только после создания вычислительно эффективного алгоритма FastMCD фирмы Rousseeuw & Van Driessen (1999 г.). С тех пор MCD применяется во многих областях, таких как медицина, финансы, анализ изображений и химия [22]. Более того, MCD также использовался для разработки многих робастных многомерных методов, которые включают робастный анализ основных компонентов, факторный анализ и множественную регрессию. Последние модификации MCD включают детерминированный алгоритм и регуляризованную версию для данных большой размерности.

В настройке многомерного местоположения и разброса данные хранятся в n × p матрице данных X = (x1,…, xn)′ с xi = (x i1,…, x ip) ′ i-м наблюдением, поэтому n обозначает количество объектов и p для количества переменных. Мы предполагаем, что наблюдения выбираются из эллиптически симметричного унимодального распределения с неизвестными параметрами μ и Σ, где μ - вектор с p-компонентами, а Σ - положительно определенная матрица размера p × p. Чтобы быть точным, многомерное распределение называется эллиптически симметричным и унимодальным, если существует строго убывающая вещественная функция g такая, что плотность может быть записана в виде формулы 18.

$$f(x) = \frac{1}{\sqrt{|\sum|}} g(d^2(x, \mu, \sum)) \tag{18}$$

В которой статистическое расстояние d(x, μ, Σ) определяется выражением описанным формулой 19.

$$d(x, \mu, \sum) = \sqrt{(x-\mu)'\sum^{-1}(x-\mu)} \tag{19}$$

Чтобы проиллюстрировать MCD, мы сначала рассмотрим набор данных по винам. Этот набор данных содержит количество 13 компонентов, содержащихся в трех типах итальянских вин. Мы рассматриваем первую группу, содержащую 59 вин, и концентрируемся на составляющих «яблочная кислота» и «пролин». Это дает бинарный набор данных, то есть p = 2. Диаграмма разброса данных показана на рисунке 12, на котором мы видим, что точки в нижней правой части диаграммы удалены по отношению к большинству данных.

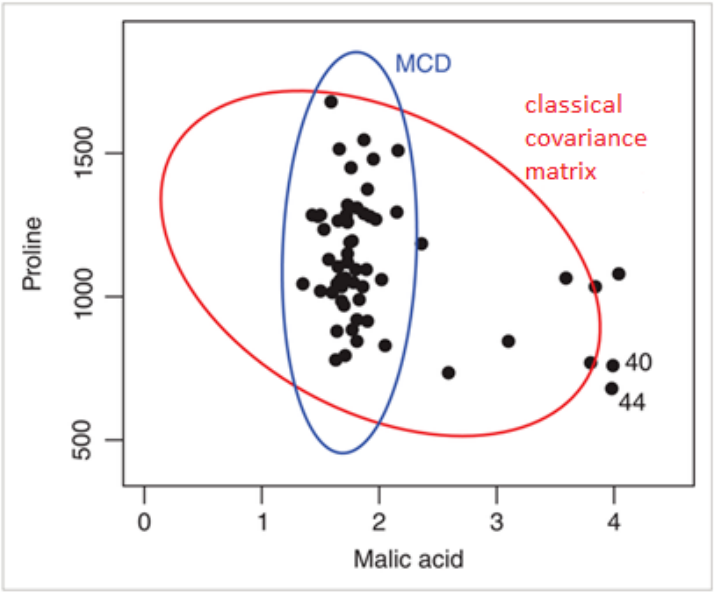


Рисунок 12 – Сравнение MCD с классической ковариационной матрицей

Двумерные данные о вине: эллипс допуска классической матрицы среднего и ковариационной (красный) и матрицы робастного местоположения и разброса (синий).

На рисунке мы видим два эллипса. Классический эллипс допуска определяется как набор p-мерных точек x, расстояние Махаланобиса которых определяется по формуле 20.

$$MD(x) = d(x, \bar{x})'Cov(X)) = \sqrt{(x, \bar{x})'Cov(X)^{-1}(x - \overline{x)}} \quad (20)$$

Здесь $\bar{x}$ - выборочное среднее, а Cov(X) –ковариационная матрица выборки. Расстояние Махаланобиса MD ($x_i$) должно сказать нам, как далеко $x_i$ находится от центра облака данных относительно его размера и формы. На рисунке 1 мы видим, что красный эллипс допуска пытается охватить все наблюдения. Следовательно, ни одно из расстояний Махаланобиса не является исключительно большим, как мы можем видеть на рисунке 13а. Основываясь только на рис. 13а, мы можем сказать, что в данных есть только три незначительных отклонения (мы игнорируем пограничные случаи).

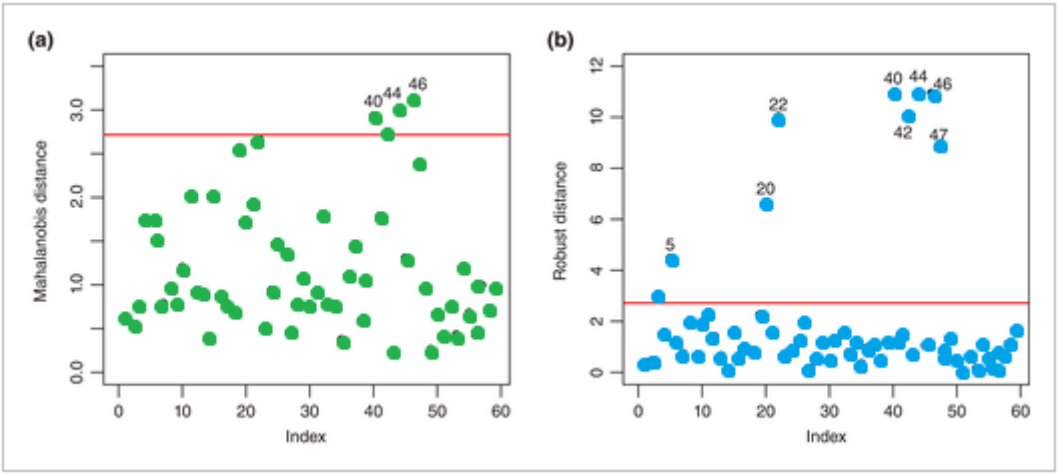


Рисунок 13 – (а) расстояния Махаланобиса и (б) робастные расстояния для двумерных данных вина

С другой стороны, эллипс устойчивых допусков основан на робастных расстояниях, определяемых формулой 21.

$$RD(x) = d(x, \hat{\mu}_{MCD}, \hat{\sum}_{MCD}) \tag{21}$$

где $\hat{\mu}_{MCD}$ - оценка местоположения MCD, а $\hat{\sum}_{MCD}$ - оценка ковариации MCD.

На рисунке 12 мы видим, что робастный эллипс (синий) намного меньше и охватывает только обычные точки данных. Робастные расстояния, показанные на рисунке 13б, теперь ясно показывают восемь отклонений.

Это иллюстрирует эффект маскировки: классические оценки могут быть настолько сильно подвержены влиянию загрязнения, что диагностические инструменты, такие как расстояния Махаланобиса, не могут обнаружить отклонения. Чтобы избежать маскировки, нам нужны надежные оценки, которые могут противостоять отклонениям, при их возникновении. MCD - является робастным оценщиком.

## 2.7 Модифицированный алгоритм KNN с использованием MCD метода

Как способ обнаружения новых классов является предположение, что регулярные данные генерируются с помощью определенного распределения вероятностей, и объявление точек с низкой плотностью вероятности. Для данных с эллиптическим распределением (например, по Гауссу) это можно сделать путем вычисления расстояния Махаланобиса от каждой точки до среднего и определив новые классы как точки с расстоянием выше некоторого порога [28]. Расстояние Махаланобиса требует параметров распределения (среднего и ковариационной матрицы). Поскольку они неизвестны, их необходимо оценивать на основе данных.

Будем применять алгоритм MCD для каждого класса. Таким образом мы получим k областей, за границами которых мы будем относить данные к

новым классам. На рисунке 14 изображены области, построенные MCD на имеющихся тренировочных данных.

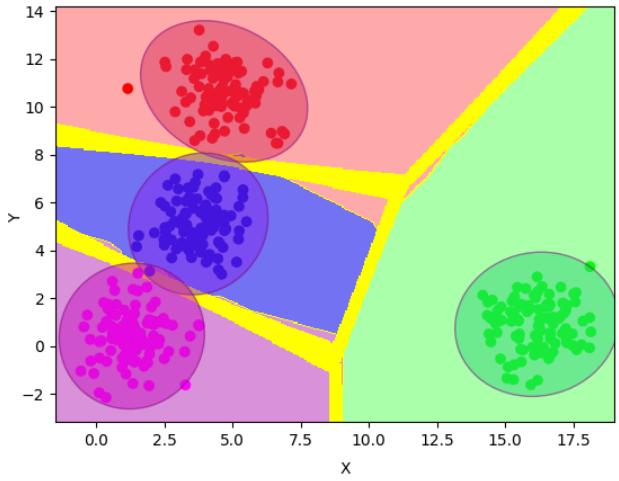


Рисунок 14 – Применение MCD к алгоритму k-ближайших соседей

Изменяя коэффициент $\alpha$ можно получать области различной размерности. На рисунке 15 изображены результаты с различным коэффициентом $\alpha$.

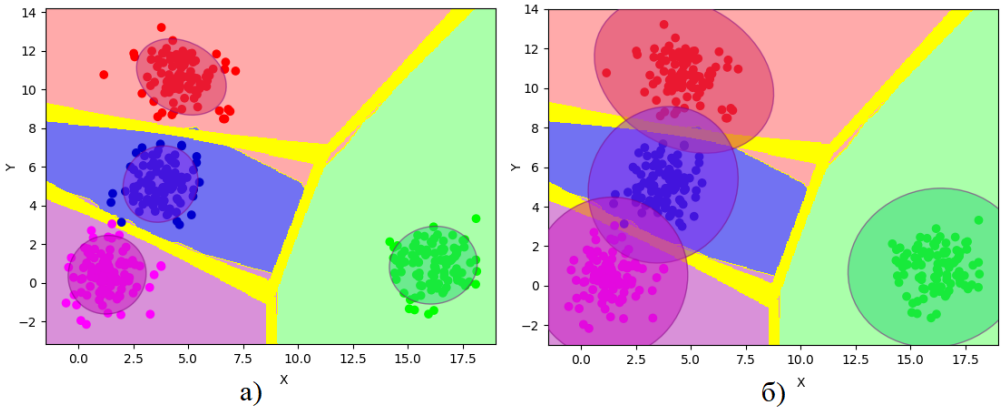


Рисунок 15 – (а) области классов при $\alpha = 2$ и (б) области классов при $\alpha = 4$

Таким образом можно подобрать наилучшее значение коэффициента $\alpha$, которое будет наиболее точно подходить под характер входных данных. Такая

настройка области $b_i$ не доступна для модификации, использующей выпуклую оболочку.

### 2.8 Генерация новых классов для модифицированного алгоритма k-ближайших соседей с использованием выпуклой оболочки

В случае появления новой точки $p_{new} = (x_1, x_2)$ будут выполнятся следующие шаги:

– Если точка $p_{new}$ принадлежит нескольким областям $h_i$, то алгоритм ведет себя аналогично алгоритму k-ближайших соседей

– Если точка $p_{new}$ принадлежит одной области $h_i$, то точка принадлежит классу $c_i$

– Если точка $p_{new}$ принадлежит области, в которой выполняются условия формулы 1, то будет сгенерирован новый класс $c_{new}$

– Во всех остальных случаях генерируется новый класс $c_{new}$

Рассмотрим, как ведет себя модифицированный алгоритм k-ближайших соседей с использованием выпуклой оболочки. Добавим несколько новых точек которые заведомо не принадлежат не одному из представленных классов.

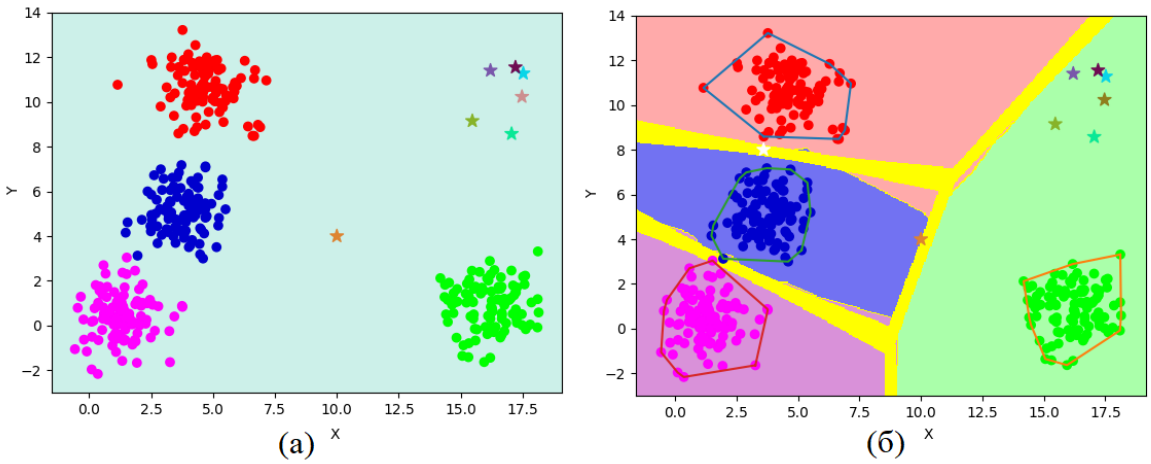


Рисунок 16 - Генерация новых классов

Группа точек в правом верхнем углу созданы по 4 пункту правил генерации новых классов. Оранжевая точка в желтой области создана на основе правила 3.

### 2.9 Генерация новых классов для модифицированного алгоритма k-ближайших соседей с использованием алгоритма MCD

В случае появления новой точки $p_{new} = (x_1, x_2)$ будут выполнятся следующие шаги:

– Если точка $p_{new}$ принадлежит нескольким областям $h_i$, то алгоритм ведет себя аналогично алгоритму k-ближайших соседей

– Если точка $p_{new}$ принадлежит одной области $h_i$ и принадлежит области, где выполняется условие формулы 1, то будет сгенерирован новый класс $c_{new}$

– Если точка $p_{new}$ принадлежит одной области $h_i$, то точка принадлежит классу $c_i$

– Если точка $p_{new}$ принадлежит области, в которой выполняются условия формулы 1, то будет сгенерирован новый класс $c_{new}$

– Во всех остальных случаях генерируется новый класс $c_{new}$

Рассмотрим, как ведет себя модифицированный алгоритм k-ближайших соседей с использованием выпуклой оболочки. На рисунке 17 звездочками обозначены точки, которые алгоритм отнес к новым классам.

Группа точек в правом верхнем углу созданы по 5 пункту правил генерации новых классов. Оранжевая точка в желтой области создана на основе правила 4. Белая точка на границе красного и зеленого класса создана на основе правила 2.

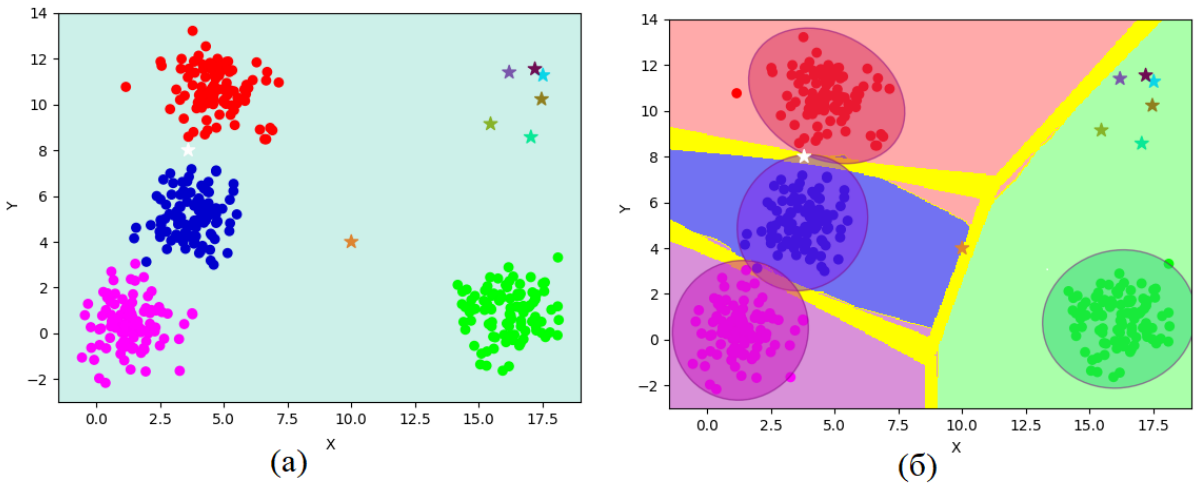


Рисунок 17 – Генерация новых классов

### 2.10 Причина применения правила 1

Причина, по которой было принято решение использовать стандартный алгоритм k-ближайших соседей для ситуации, когда новый объект принадлежит нескольким областям является следующей. Алгоритмы построения неких выпуклых множеств таких как алгоритм построения минимальной выпуклой оболочки и алгоритм MCD плохо работают на невыпуклых множествах. На рисунке 18 приведен пример таких множеств.

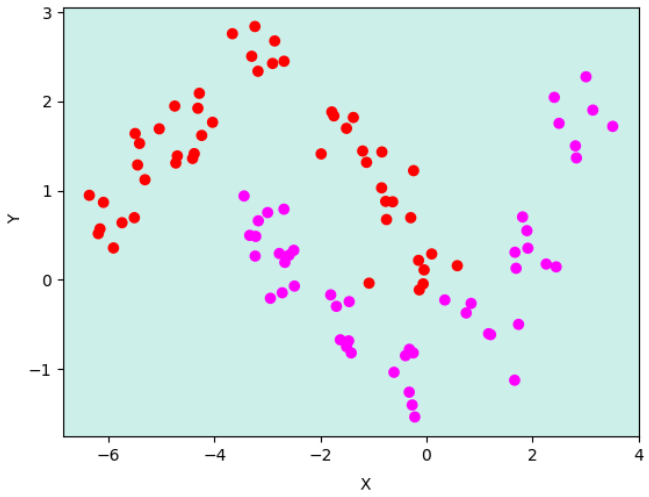


Рисунок 18 – Точки, образующие два невыпуклых множества

Попробуем применить к таким множествам алгоритм построения минимальной выпуклой оболочки (рисунок 19).

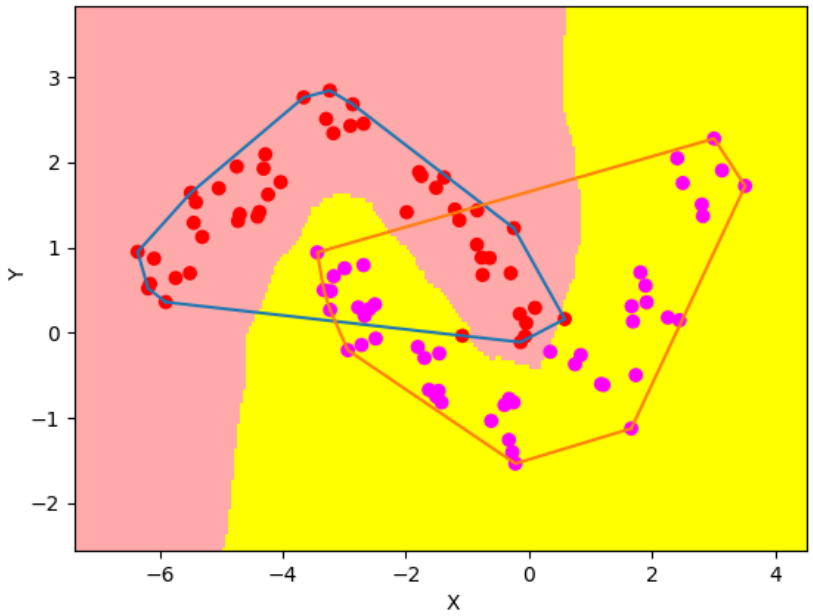


Рисунок 19 – Построение выпуклой оболочки на невыпуклых множествах

Как можно заметить при таком построении получается пересечение множеств, но при этом стандартный алгоритм k-ближайших соседей работает корректно [24]. Такой подход позволяет гарантировать, что модифицированный алгоритм k-ближайших соседей даст результат не хуже, чем стандартный алгоритм.

### 2.11 Метод объединения классов

Генерация новых классов дает свои преимущества в том, что помогает обратить внимание на странные данные и пометить их как что-то новое. Такие новые точки могут образовывать группы, которые имеют схожие параметры. Следовательно, важно не только фиксировать новые классы, но и уметь их объединять. Общий ход алгоритма будет одинаковый как для модифицированного алгоритма k-ближайших соседей с использованием

выпуклой оболочки, так и для модифицированного алгоритма k-ближайших соседей с использованием MCD.

Общий алгоритм объединения сгенерированных классов:

– Алгоритм принял решение о том, что точка не принадлежит не одному из представленных классов

– Алгоритм принимает решение о принимает решение об объединении нового класса с уже имеющимся сгенерированным классом

– Если алгоритм объединяет классы, то происходит перестроение области объединенного класса

– Иначе генерируется область нового класса

Этого алгоритма будет использоваться в реализации алгоритма, чтобы объединять сгенерированные классы и использовать их для обучения классификатора KNN.

Таким образом удалось сформировать модель алгоритма KNN способную генерировать новые классы и при этом иметь высокую точность и стабильность на разных наборах данных.

# Глава 3 Реализация модифицированного алгоритма KNN

## 3.1 Выбор алгоритма кластеризации

Кластеризация — это способ сгруппировать набор точек данных таким образом, чтобы похожие точки данных были сгруппированы вместе. Таким образом, алгоритмы кластеризации ищут сходства или различия между точками данных. Кластеризация является неконтролируемым методом обучения, поэтому нет никакой маркировки, связанной с точками данных. Алгоритм пытается найти основную структуру данных [15], [16].

Существуют различные подходы и алгоритмы для выполнения задач кластеризации, которые можно разделить на три подкатегории:

– Кластеризация на основе разделов: например, k-means, k-median.
– Иерархическая кластеризация: Agglomerative, Divisive.
– Кластеризация на основе плотности: DBSCAN

Задача состоит в выборе наиболее подходящего алгоритма объединения новых классов.

## 3.2 Кластеризация на основе разделения данных

Секционирование Кластеризация — это тип техники кластеризации, которая делит набор данных на определенное количество групп. (Например, значение $k$ в k-means и будет принято решение, прежде чем мы обучим модель). Его также можно назвать методом, основанным на центроиде. При таком подходе центр кластера (центроид) формируется таким образом, чтобы расстояние точек данных в этом кластере было минимальным при расчете с другими центроидами кластера. Наиболее популярным примером этого алгоритма является алгоритм k-means.

Кластеризация на основе центроидов организует данные в неиерархические кластеры, в отличие от иерархической кластеризации,

определенной ниже. k-means - это наиболее широко используемый алгоритм кластеризации на основе центроидов, также он достаточно эффективный, действенный и простой алгоритм кластеризации. В алгоритме k – количество кластеров, гиперпараметр алгоритма. Основная идея алгоритма состоит в том, чтобы найти k центроидов с последующим нахождением k наборов точек, которые сгруппированы на основе близости к центроиду таким образом, чтобы квадраты расстояний от точек в кластере до центроида были минимизированы. Эта оптимизация – очень сложная задача, поэтому для ее решения используется приближение.

В некоторых случаях трудно интерпретировать центроиды, например, в тех случаях, когда необходимо кластеризовать текстовые данные, центроиды не интерпретируемы. Для решения этой проблемы, можно использовать алгоритм K-medoids, где в качестве центра кластера будет выбран наиболее центрированный элемент данных, и он, как правило, более устойчив к выбросам, чем другие алгоритмы.

При кластеризации на основе центроидов кластеры представлены центральным вектором, который не обязательно может быть членом набора данных. Когда количество кластеров фиксировано на k, k- означает, что кластеризация дает формальное определение как задача оптимизации: найти k центров кластера и назначить объекты ближайшему центру кластера, так что квадраты расстояний от кластера минимизированы.

Большинство алгоритмов типа k-средних требуют, чтобы количество кластеров k – было указано заранее, что считается одним из самых больших недостатков этих алгоритмов. Кроме того, алгоритмы предпочитают кластеры примерно одинакового размера, так как они всегда присваивают объект ближайшему центроиду. Это часто приводит к неправильной обрезке границ кластеров (что неудивительно, поскольку алгоритм оптимизирует центры кластеров, а не границы кластеров).

Алгоритмы на основе центроидов эффективны, но чувствительны к начальным условиям и выбросам.

### 3.3 Иерархическая кластеризация

Алгоритмы иерархической кластеризации делятся на 2 категории: «сверху-вниз» или «снизу-вверх». Алгоритмы «снизу-вверх» обрабатывают каждую точку данных как единый кластер вначале, а затем последовательно объединяют (или агрегируют) пары кластеров, пока все кластеры не будут объединены в один кластер, содержащий все точки данных. Поэтому восходящая иерархическая кластеризация называется иерархической агломеративной кластеризацией или HAC. Эта иерархия кластеров представлена в виде дерева (или дендрограммы).

Корень дерева — это уникальный кластер, который собирает все образцы, а листья — это кластеры только с одним образцом.

– Мы начинаем с обработки каждой точки данных как единого кластера, т.е. если в нашем наборе данных есть $X$ точек данных, то у нас есть $X$ кластеров. Затем мы выбираем метрику расстояния, которая измеряет расстояние между двумя кластерами. В качестве примера мы будем использовать среднюю связь, которая определяет расстояние между двумя кластерами как среднее расстояние между точками данных в первом кластере и точками данных во втором кластере.

– На каждой итерации мы объединяем два кластера в один. Два кластера, которые необходимо объединить, выбираются как кластеры с наименьшей средней связью. Т.е. согласно выбранной нами метрике расстояния, эти два кластера имеют наименьшее расстояние между собой и, следовательно, наиболее похожи и должны быть объединены.

– Шаг 2 повторяется до тех пор, пока мы не достигнем корня дерева, т.е. у нас будет только один кластер, содержащий все точки данных. Таким образом, мы можем выбрать, сколько кластеров мы хотим в конечном итоге, просто выбрав, когда остановить объединение кластеров, т.е. когда мы перестанем строить дерево.

Иерархическая кластеризация не требует от нас указания количества кластеров, и мы даже можем выбрать, какое количество кластеров выглядит лучше всего, поскольку мы строим дерево. Кроме того, алгоритм нечувствителен к выбору метрики расстояния; все они, как правило, работают одинаково хорошо, тогда как с другими алгоритмами кластеризации выбор метрики расстояния имеет решающее значение. Особенно хороший вариант использования методов иерархической кластеризации – это когда базовые данные имеют иерархическую структуру, и вы хотите восстановить иерархию; другие алгоритмы кластеризации не могут этого сделать. Эти преимущества иерархической кластеризации достигаются за счет более низкой эффективности, поскольку она имеет временную сложность $O\ (n^3)$

### 3.4 Кластеризация на основе плотности

Методы иерархической кластеризации на основе разделов и иерархической кластеризации высокоэффективны с кластерами нормальной формы. Однако, когда дело доходит до кластеров произвольной формы или обнаружения выбросов, методы на основе плотности более эффективны.

Например, набор данных на рисунке 20 можно легко разделить на три кластера с помощью алгоритма k-means.

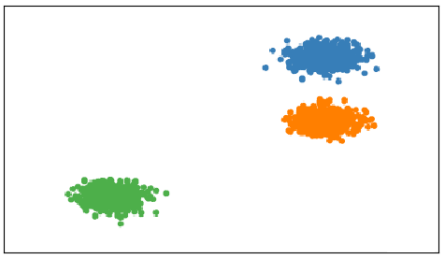

Рисунок 20 – Алгоритм кластеризации k-means

Для кластеризации таких данных алгоритм k-means подходит отлично, но встречаются и такие данные, которые образуют более сложные кластеры.

Примеры таких кластеров изображены на рисунках 2 и 3. На таких данных алгоритм k-means работает некорректно (рисунок 21, 22).

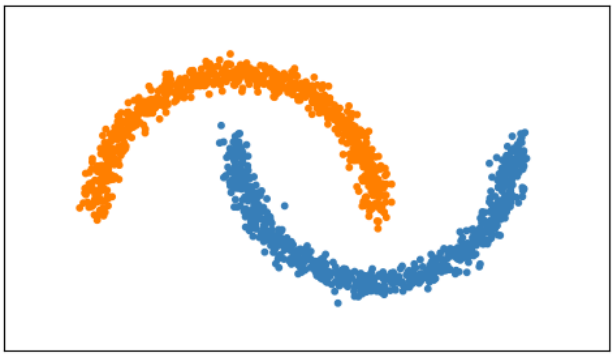

Рисунок 21 - Пример сложного кластера 1

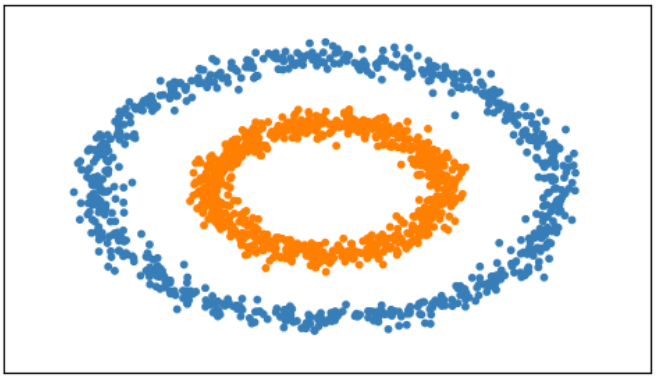

Рисунок 22 – Пример сложного кластера 2

Точки данных на этих рисунках сгруппированы в произвольные формы или включают выбросы. Алгоритмы кластеризации на основе плотности очень эффективны при обнаружении областей с высокой плотностью и выбросов. Очень важно обнаруживать выбросы в некоторых задачах, таких как обнаружение аномалий.

Кластеризация на основе плотности относится к методам обучения без учителя, которые идентифицируют отличительные группы/кластеры в данных, основываясь на идее, что кластер в пространстве данных представляет

собой непрерывную область с высокой плотностью точек, отделенную от других таких кластеров смежными областями с низкой плотностью точек.

Алгоритмы кластеризации на основе плотности могут изучать кластеры произвольной формы, а с помощью алгоритма Level Set Tree можно изучать кластеры в наборах данных, которые демонстрируют большие различия в плотности.

Однако, следует отметить, что эти алгоритмы несколько сложнее настраивать, в отличие от параметрических алгоритмов кластеризации, таких как k-means. Такие параметры, как эпсилон для DBSCAN или дерево Level Set Tree менее интуитивны по сравнению с параметрами кластеров для k-means, поэтому для этих алгоритмов сложнее выбрать хорошие начальные значения параметров.

### 3.5 Алгоритм кластеризации DBSCAN

Пространственная кластеризация приложений с шумом на основе плотности (DBSCAN) – это базовый алгоритм кластеризации на основе плотности. Он может обнаруживать кластеры разных форм и размеров на большом количестве данных, содержащих шум и выбросы.

Алгоритм DBSCAN использует два параметра:

– $minPts$ - минимальное количество точек (пороговое значение), сгруппированных вместе, чтобы область считалась плотной.

– $eps$ - мера расстояния, которая будет использоваться для определения местоположения точек в окрестности любой точки.

Эти параметры можно понять, если мы исследуем две концепции, называемые плотностью достижимости и плотностью связи.

Достижимость с точки зрения плотности определяет точку, доступную для другой, если она находится на определенном расстоянии $eps$ от нее.

Связность, с другой стороны, включает в себя цепочный подход, основанный на переходных процессах, для определения того, находятся ли

точки в определенном кластере. Например, точки p и q могут быть связаны, если $p \to r \to s \to t \to q$, где $a \to b$ означает, что b находится в окрестности $a$.

После завершения кластеризации DBSCAN есть три типа точек (рисунок 23):

– Центральная точка – это точка, которая имеет не менее $m$ точек на расстоянии $n$ от себя.

– Граничная точка – это точка, которая имеет хотя бы одну центральную точку на расстоянии $n$.

– Шум — это точка, которая не является ни центральной, ни граничной. И на расстоянии $n$ от себя у него меньше $m$ точек.

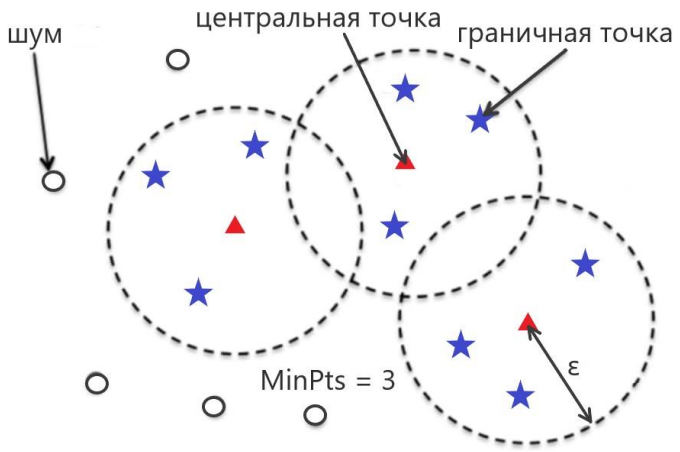


Рисунок 23 – Типы точек

Определив типы точек, опишем шаги работы алгоритма DBSCAN.

– Алгоритм действует путем произвольного выбора точки в наборе данных до тех пор, пока все точки не будут посещены.

– Если есть хотя бы $minPoint$ точек в радиусе $eps$ от точки, мы считаем все эти точки частью одного кластера.

– Затем кластеры расширяются путем рекурсивного повторения вычисления окрестности для каждой соседней точки.

Алгоритм DBSCAN требует настройки параметров. В следующем подразделе описываются базовые правила подбора параметров алгоритма.

### 3.5.1 Оценка параметров алгоритма DBSCAN

В каждой задачи интеллектуального анализа данных есть проблема настройки параметров. Каждый параметр определенным образом влияет на работу алгоритма. Для DBSCAN необходимы параметры $eps$ и $minPts$.

Как правило минимальный $minPts$ может быть получен из числа размеров $D$ в наборе данных, как $\mathrm{minPts} \geq \mathrm{D} + 1$. Низкое значение $\mathrm{minPts} = 1$ не имеет смысла, так как тогда каждая точка сама по себе будет уже кластером. При $\mathrm{minPts} \leq 2$ результат будет таким же, как и при иерархической кластеризации с метрикой одиночной связи с разрезом дендрограммы на высоте $eps$. Следовательно, $minPts$ необходимо выбрать не менее 3. Однако большие значения обычно лучше для наборов данных с шумом и будут давать более значимые кластеры. Как правило, можно использовать $\mathrm{minPts} = 2 * \mathrm{dim}$, но может потребоваться выбрать большие значения для очень больших данных, для зашумленных данных или для данных, содержащих много дубликатов.

Затем значение $eps$ можно выбрать с помощью графика k-расстояний, на котором расстояние до ближайшего соседа $k = \mathrm{minPts} - 1$ упорядочено от наибольшего к наименьшему значению. Хорошие значения $eps$ находятся там, где этот график показывает «изгиб». Если $eps$ выбрано слишком маленьким, большая часть данных не будет кластеризована, тогда как при слишком высоком значении $eps$ кластеры объединятся, и большинство объектов окажется в одном кластере. В общем, предпочтительны небольшие значения $eps$, и, как показывает опыт, только небольшая часть точек должна находиться на этом расстоянии друг от друга.

Выбор функции расстояния тесно связан с выбором $eps$ и имеет большое влияние на результаты. Как правило, необходимо сначала определить разумную меру подобия для набора данных, прежде чем можно будет выбрать

параметр $eps$. Оценка для этого параметра отсутствует, однако функции расстояния должны быть выбраны в соответствии с набором данных.

### 3.5.2 Оценка характеристик алгоритма DBSCAN

Для принятия решения об использовании алгоритма кластеризации необходимо определить возможность его применения и ограничения, которые он накладывает.

Плюсы алгоритма DBSCAN:

– Не требует заранее указывать количество кластеров.

– Хорошо работает с кластерами произвольной формы.

– DBSCAN устойчив к выбросам и способен обнаруживать выбросы.

Минусы алгоритма DBSCAN:

– В некоторых случаях определение подходящего расстояния соседства ($eps$) непросто и требует знания предметной области.

– Если кластеры сильно различаются по плотности внутри кластера, DBSCAN не подходит для определения кластеров. Характеристики кластеров определяются комбинацией параметров eps-minPts. Поскольку мы передаем в алгоритм одну комбинацию eps-minPts, он не может хорошо обобщаться на кластеры с сильно различающейся плотностью.

Алгоритмическая сложность алгоритма кластеризации DBSCAN:

– Лучший случай: если для хранения набора данных используется система индексации так, чтобы соседние запросы выполнялись в логарифмическое время, то мы получаем $O(nlogn)$ среднюю сложность выполнения.

– Худший случай: без использования индексации или на вырожденных данных (например, всех точек на расстоянии меньше $eps$) сложность выполнения в худшем случае остается $O(n^2)$.

– Средний случай: такой же, как лучшем/худшем случае, в зависимости от данных и реализации алгоритма.

Как видно алгоритм отлично подходит под решение текущей задачи. Далее поясним выбор алгоритма DBSCAN и его преимущество над более простым алгоритмом k-means.

### 3.6 Сравнение DBSCAN и k-means

Кластеризация K-средних может объединять слабо связанные наблюдения. Каждое наблюдение со временем становится частью некоторого кластера, даже если наблюдения разбросаны далеко в векторном пространстве. Поскольку кластеры зависят от среднего значения элементов кластера, каждая точка данных играет роль в формировании кластеров. Небольшое изменение точек данных может повлиять на результат кластеризации. Эта проблема значительно сокращается в DBSCAN из-за способа формирования кластеров. Обычно это не большая проблема, если только мы не сталкиваемся с какими-то данными странной формы.

Другая проблема k-means заключается в необходимости указывать количество кластеров, однако, большую часть времени мы не будем знать, какое значение $k$ является разумным.

Алгоритм DBSCAN хорош тем, что в нем не нужно указывать количество кластеров для его использования. Все, что для него нужно, это функция для расчета расстояния между значениями и некоторые правила относительно того, какое расстояние будет считаться «близким». DBSCAN также дает более разумные результаты, чем k-средних, для множества различных распределений. Рисунок 24 ниже иллюстрирует этот факт [21], [27].

Особенно бросается в глаза такая особенной алгоритма k-means то, что он требует указания конкретного количества кластеров и строго ей придерживается. Так на последнем рисунке видно, как алгоритм поделил квадрат на части хоть и очевидно, что он образует один кластер [23].

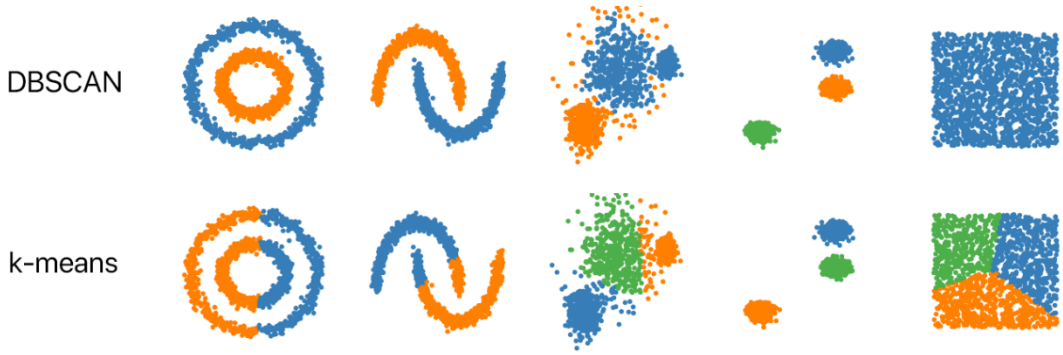


Рисунок 24 – Сравнение результатов кластеризации k-means и DBSCAN

Такое поведение недопустимо в поставленной задаче. Также мы не можем даже примерно сказать сколько классов сейчас присутствует в данных. Чтобы решить эту задачу придется дополнительно модифицировать алгоритм k-means и ввести дополнительную систему определения приблизительного количества кластеров. Следовательно, выбор однозначно падает на алгоритм DBSCAN у которого эти недостатки отсутствуют.

### 3.7 Программная реализация модифицированного алгоритма KNN

Определившись с алгоритмом кластеризации, реализуем модифицированный алгоритм KNN.

Реализовывать алгоритм мы будем с использованием языка программирования Python 3. Также потребуется следующий набор библиотек.

– NumPy – библиотека для языка программирования Python. Предоставляет реализации вычислительных алгоритмов (в виде функций и операторов), оптимизированные для работы с многомерными массивами. В результате любой алгоритм, который может быть выражен в виде последовательности операций над массивами (матрицами) и реализованный с использованием NumPy, работает так же быстро, как эквивалентный код, выполняемый в MATLAB.

– Matplotlib — это комплексная библиотека для создания статических, анимированных и интерактивных визуализаций в Python.

– Scilit-learn – это библиотека которая предоставляет десятки встроенных алгоритмов и моделей машинного обучения, называемых оценками. Каждый оценщик может быть приспособлен к некоторым данным, используя свой метод подбора.

– Python’s standard library – это библиотека содержит встроенные модули, написанные на C, которые обеспечивают доступ к функциям системы, таким как файловый ввод-вывод, которые в противном случае были бы недоступны для программистов Python, а также модули, написанные на Python, которые предоставляют стандартизированные решения для многих проблем, возникающих в повседневное программирование. Некоторые из этих модулей специально разработаны для поощрения и повышения переносимости программ Python за счет абстрагирования специфических особенностей платформы в платформенно-нейтральные API.

– SciPy — библиотека для языка программирования Python с открытым исходным кодом, предназначенная для выполнения научных и инженерных расчётов.

Выполним импорт необходимых библиотек (листинг 1).

Листинг 1 – Импорт необходимых для работы алгоритма библиотек

```
import random
import math
import pylab as pl
import numpy as np
from matplotlib.colors import ListedColormap
from matplotlib.colors import BoundaryNorm
import scipy as sp
import scipy.sparse
import scipy.spatial.qhull
```

```
from scipy import interpolate
from matplotlib.patches import Ellipse
import matplotlib.transforms as transforms
```

Далее необходимо получить исходные данные для тренировки классификатора.

### 3.7.1 Создание тестового и тренировочного набора данных

Фундаментальная цель машинного обучения - обобщение за пределами экземпляров данных, используемых для обучения моделей. Мы хотим оценить модель, чтобы оценить качество модели для данных, на которых модель не была обучена. Однако, поскольку будущие экземпляры имеют неизвестные целевые значения, и мы не можем сейчас проверить точность наших прогнозов для будущих экземпляров, нам необходимо использовать некоторые данные, ответ на которые мы уже знаем, в качестве прокси для будущих данных. Оценка модели с использованием тех же данных, которые использовались для обучения, бесполезна, потому что она поощряет модели, которые могут «запоминать» данные обучения, в отличие от обобщения на их основе.

Общая стратегия состоит в том, чтобы взять все доступные помеченные данные и разделить их на подмножества обучения и оценки, обычно с соотношением 70-80 процентов для обучения и 20-30 процентов для оценки. Система машинного обучения использует обучающие данные, чтобы обучать модели видеть закономерности, и использует оценочные данные для оценки качества прогнозирования обученной модели. Система машинного обучения оценивает эффективность прогнозирования путем сравнения прогнозов по набору данных оценки с истинными значениями (известными как наземная истина) с использованием различных показателей. Обычно вы используете «лучшую» модель для оценочного подмножества, чтобы делать прогнозы на будущие случаи, для которых вы не знаете целевой ответ. На рисунке 25 изображена схема разделения тестовых и тренировочных данных.

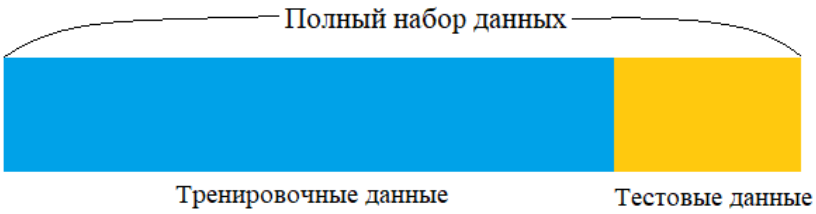


Рисунок 25 – Схема разделения тестовых и тренировочных данных

Разработанный алгоритм разделяет данные, отправленные для обучения модели, на 90 процентов для обучения и 10 процентов для оценки. По умолчанию алгоритм использует первые 90 процентов входных данных в том порядке, в котором они появляются в исходных данных для источника данных обучения, а оставшиеся 10 процентов данных - для источника данных оценки (листинг 2).

Листинг 2 - Генерация тестовых данных для классификации

```
def generateData (numberOfClassEl, numberOfClasses):
    data = []
    for classNum in range(numberOfClasses):
            for point in centers[classNum]:
                for rowNum in range(numberOfClassEl):
                    data.append([ [random.gauss(centerX,0.3 *
1), random.gauss(centerY,0.3 * 1)], classNum])
    return data
```

Разделение будет выполнено 1 к 10. В листинг 3 приведен код разделения данных.

Листинг 3 – Функция разделения тестовых данных

```
def splitTrainTest (data, testPercent):
    trainData = []
    testData  = []
```

```
    for row in data:
        if random.random() < testPercent:
            testData.append(row)
        else:
            trainData.append(row)
    return trainData, testData
```

Далее реализуем модифицированных алгоритм KNN основанный на ошибки классификации.

### 3.7.2 Реализация модифицированного алгоритма KNN основанного на ошибке классификации

Модифицированный алгоритм KNN отличается тем, что имеет некую граничную ошибку классификации $eps$, которая определяет области, где принадлежность к конкретному классу не определена. В листинге 4 приведен код модифицированного алгоритма KNN основанного на ошибке классификации.

Листинг 4 – Модифицированный алгоритм KNN основанный на ошибке классификации

```
def classifyKNN (trainData, testData, k, numberOfClasses,
itemsInClasses, eps):
    def dist (a, b):
        return math.sqrt((a[0] - b[0])**2 + (a[1] - b[1])**2)
    testLabels = []
    for testPoint in testData:
        testDist = [ [dist(testPoint, trainData[i][0]),
trainData[i][1]] for i in range(len(trainData))]
        stat = [0 for i in range(numberOfClasses)]
        for d in sorted(testDist)[0:k]:
            stat[d[1]] += 1
        error = [0 for i in range(numberOfClasses)]
```

```
        for d in testDist:
            error[d[1]] += d[0]
        for i in range(numberOfClasses):
            error[i] /= itemsInClasses[i]
        error.sort()
        if (error[1] - error[0] <= eps):
            testLabels.append(numberOfClasses)
        else:
            testLabels.append( sorted(zip(stat,
range(numberOfClasses)), reverse=True)[0][1] )
    return testLabels
```

После выполнения классификации необходимо отобразить полученные данные.

### 3.7.3 Реализация визуализации результатов работы модифицированного алгоритма KNN

Визуализация данных — это графическое представление информации и данных. Используя визуальные элементы, такие как диаграммы, графики и карты, инструменты визуализации данных предоставляют доступный способ видеть и понимать тенденции, выбросы и закономерности в данных.

В мире больших данных инструменты и технологии визуализации данных необходимы для анализа огромных объемов информации и принятия решений на основе данных.

Наш взгляд привлекают цвета и узоры. Мы можем быстро отличить красный от синего, квадрат от круга. Наша культура визуальна, включая все, от искусства и рекламы до телевидения и фильмов.

Визуализация данных — это еще одна форма визуального искусства, которая привлекает наш интерес и держит нас в курсе сообщения. Когда мы видим диаграмму, мы быстро видим тенденции и выбросы. Если мы что-то видим, мы быстро усваиваем это. Это повествование с определенной целью. Если вы когда-либо смотрели на огромную электронную таблицу данных и не

видели тенденции, вы знаете, насколько более эффективной может быть визуализация. По мере того, как «эпоха больших данных» набирает обороты, визуализация становится все более важным инструментом для понимания триллионов строк данных, генерируемых каждый день. Визуализация данных помогает рассказывать истории, преобразовывая данные в более понятную форму, выделяя тенденции и выбросы. Хорошая визуализация рассказывает историю, удаляя шум из данных и выделяя полезную информацию.

Однако это не так просто, как просто нарядить график, чтобы он выглядел лучше, или добавить «информационную» часть инфографики. Эффективная визуализация данных — это тонкий баланс между формой и функцией. Самая простая диаграмма может быть слишком скучной, чтобы ее можно было заметить, или она может дать важный аргумент; самая потрясающая визуализация может совершенно не передать правильное сообщение или может говорить о многом. Данные и визуальные эффекты должны работать вместе, и есть искусство сочетать отличный анализ с отличным повествованием. Для визуализации данным мы используем заранее настроенную конфигурацию библиотеки matplotlib. В листинге 5 приведены функции отображения тестовых данных, разделенных по цветам. Каждый цвет — это отдельный класс.

Листинг 5 - Функции отображения полученных данных в результате классификации

```
def showData (nClasses, nItemsInClass):
    trainData      = generateData (nItemsInClass, nClasses)
    classColormap  = ListedColormap(['#FF0000', '#00FF00',
'#FFFF1F', '#0000CD', '#FF00FF'])
    pl.scatter([trainData[i][0][0] for i in
range(len(trainData))],
               [trainData[i][0][1] for i in
range(len(trainData))],
```

```
                c=[trainData[i][1] for i in
range(len(trainData))],
                cmap=classColormap)
    pl.show()
def showData (trainData):
    classColormap  = ListedColormap(['#FF0000', '#00FF00',
'#FFFF1F', '#0000CD', '#FF00FF'])
    pl.scatter([trainData[i][0][0] for i in
range(len(trainData))],
                [trainData[i][0][1] for i in
range(len(trainData))],
                c=[trainData[i][1] for i in
range(len(trainData))],
                cmap=classColormap)
    pl.show()
```

Как результат применения этой функции мы можем наблюдать следующий результат (рисунок 26).

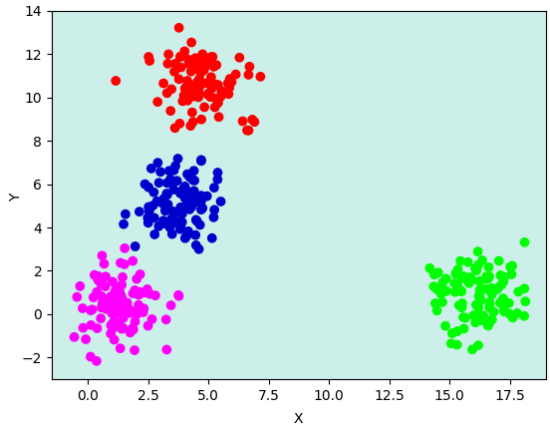


Рисунок 26 – Применение функции визуализации набора данных

Деления на рисунке 26 и на всех последующих являются отображение неких условных единиц и необходимы для наглядного понимания масштаба. Также необходимо отображать зоны, которые создаются в результате работы

алгоритма. Для этого мы накладываем на поле сетку с определенным шагом. Затем каждую ячейку сетки отправляем на вход модифицированному алгоритму KNN. Тот в свою очередь определяет тип переданной точки. Точки разного типа отображаются разными цветами. В листинге 5 приведен код, который выполняет описанные выше действия.

Листинг 5 – Код, визуализирующий зоны, классифицируемые модифицированным алгоритмом KNN.

```
def showDataOnMesh (trainData, nClasses, itemsInClasses, k,
eps):
    def generateTestMesh (trainData):
        x_min = min( [trainData[i][0][0] for i in
range(len(trainData))] ) - 1.0
        x_max = max( [trainData[i][0][0] for i in
range(len(trainData))] ) + 1.0
        y_min = min( [trainData[i][0][1] for i in
range(len(trainData))] ) - 1.0
        y_max = max( [trainData[i][0][1] for i in
range(len(trainData))] ) + 1.0
        h = 0.05
        testX, testY = np.meshgrid(np.arange(x_min, x_max, h),
                                   np.arange(y_min, y_max, h))
        return [testX, testY]
    testMesh       = generateTestMesh (trainData)
    testMeshLabels = classifyKNN (trainData,
zip(testMesh[0].ravel(), testMesh[1].ravel()), k, nClasses,
itemsInClasses, eps)
    classColormap  = ListedColormap(['#FF0000', '#00FF00',
'#FFFFFF', '#0000CD', '#FF00FF'])
    testColormap   = ListedColormap(['#FFAAAA', '#AAFFAA',
'#7272f2', '#d991d9', '#FFFF00'])#, '#4B0082' '#FFD700'
    def addCorner(markers):
        for i in range(len(markers)):
```

```
                for j in range(len(markers[i])):
                    cur = markers[i][j]
                    if (i < len(markers) - 1 and markers[i+1][j] !=
cur or j < len(markers[i]) - 1 and markers[i][j+1] != cur):
                        markers[i][j] = 4
        return 0
    markers =
np.asarray(testMeshLabels).reshape(testMesh[0].shape)
    norm = BoundaryNorm(np.unique(markers),
len(np.unique(markers)) - 1)
    addCorner(markers)
    pl.pcolormesh(testMesh[0],
                  testMesh[1],
                  markers,
                  cmap=testColormap,
                  shading="auto",
                  )

    hulls = []
```

На рисунке 27 приведен результат выполнения описанной выше функции.

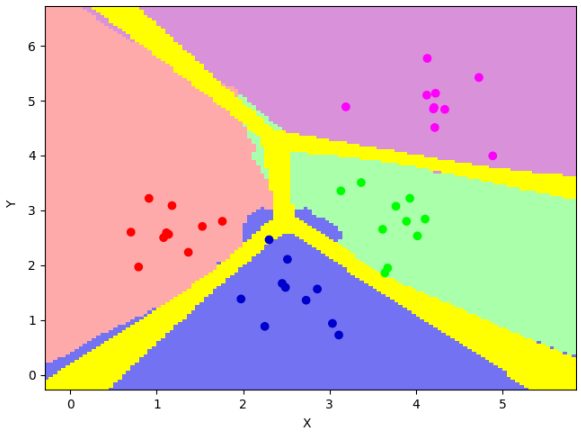


Рисунок 27 – Результат выполнения функции отображения зон, которые классифицированы модифицированным алгоритмом KNN

Как видно такой способ отображения данных помогает наглядно продемонстрировать логику работы алгоритма.

### 2.7.4 Реализация алгоритма построения выпуклой оболочки

Реализация алгоритма Грэхема выполнена совместно с функцией отображения данных. Функция принимает на вход результат работы алгоритма модифицированного алгоритма KNN основанного на ошибки классификации. Далее функция строит поверх имеющегося изображения линии выпуклой оболочки. В листинге 6 приведен фрагмент описанной выше функции.

Листинг 6 – Функция построения выпуклой оболочки

```
def showConvexHull():
    backgroundColor = ['#d63160', '#31d676', '#8428ed',
'#c912c0']
    for k in range(nClasses):
        X = []
        for i in trainData:
            if (i[1] == k):
                X.append(i[0])
        hull = sp.spatial.qhull.Delaunay(X).convex_hull
        g = sp.sparse.csr_matrix((np.ones(hull.shape[0]),
hull.T), shape=(hull.max() + 1,) * 2)
        sorted_hull = sp.sparse.csgraph.depth_first_order(g,
hull[0, 0], directed=False)[0]
        xHull = [X[i][0] for i in sorted_hull]
        yHull = [X[i][1] for i in sorted_hull]
        xHull = np.append(xHull, xHull[0])
        yHull = np.append(yHull, yHull[0])
        hulls.append([xHull,yHull])
        pl.plot(xHull, yHull)
        phi = np.linspace(0, 2. * np.pi, len(xHull))
```

```
        # xnew = np.linspace(xHull.min(), xHull.max(), 100)
        bspline = interpolate.make_interp_spline(phi,
np.c_[xHull, yHull])
        phi = np.linspace(0, 2. * np.pi, 50)
        x_new, y_new = bspline(phi).T
        pl.plot(x_new, y_new)
        ax = pl.axes()
        mu = 2, -3
        scale = 6, 5
        x = np.array([i[0] for i in X])
        y = np.array([i[1] for i in X])
        x, y = get_correlated_dataset(500, dependency_kwargs,
mu, scale)
        Plot the ellipse with zorder=0 in order to demonstrate
        its transparency (caused by the use of alpha).
        11.1 = 2, 11.2 = 4
        confidence_ellipse(x, y, ax,
                           alpha=0.5,
                           n_std=3,
                           facecolor=backgroundColor[k],
                           edgecolor='purple', zorder=4)
```

Результат выполнения функции изображен на рисунке 28.

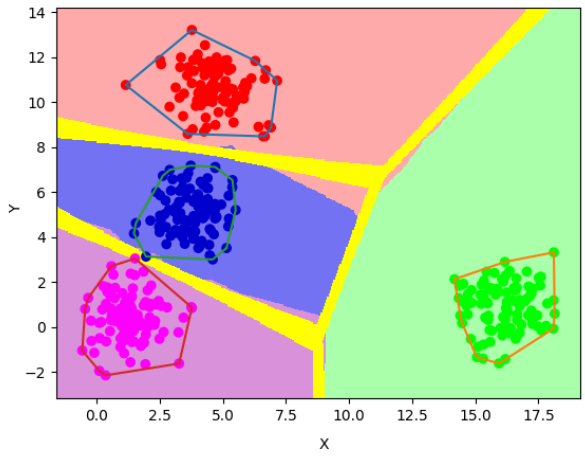


Рисунок 28 – Результат построения выпуклой оболочки

Как видно функция отлично строит выпуклую оболочку для переданных в неё данных и отображает результат.

### 2.7.5 Реализация алгоритма построения MCD области

Построение области по алгоритму MCD работает аналогично функции построения выпуклой оболочки. На вход алгоритму передается результат работы модифицированного алгоритма KNN основанного на ошибке классификации, а затем на полученных данных выполняется алгоритм MCD. В листинге 7 приведена описанная выше функция.

Листинг 7 – Функция построения области по алгоритму MCD

```
def confidence_ellipse(x, y, ax, n_std=3.0, **kwargs):
    cov = np.cov(x, y)
    pearson = cov[0, 1]/np.sqrt(cov[0, 0] * cov[1, 1])
    ell_radius_x = np.sqrt(1 + pearson)
    ell_radius_y = np.sqrt(1 - pearson)
    ellipse = Ellipse((0, 0), width=ell_radius_x * 2,
height=ell_radius_y * 2,
                      facecolor='none', **kwargs)
    scale_x = np.sqrt(cov[0, 0]) * n_std
    mean_x = np.mean(x)
    scale_y = np.sqrt(cov[1, 1]) * n_std
    mean_y = np.mean(y)
    transf = transforms.Affine2D() \
        .rotate_deg(45) \
        .scale(scale_x, scale_y) \
        .translate(mean_x, mean_y)
    ellipse.set_transform(transf + ax.transData)
    return ax.add_patch(ellipse)
```

На рисунке 29 приведен результат выполнения описанной выше функции.

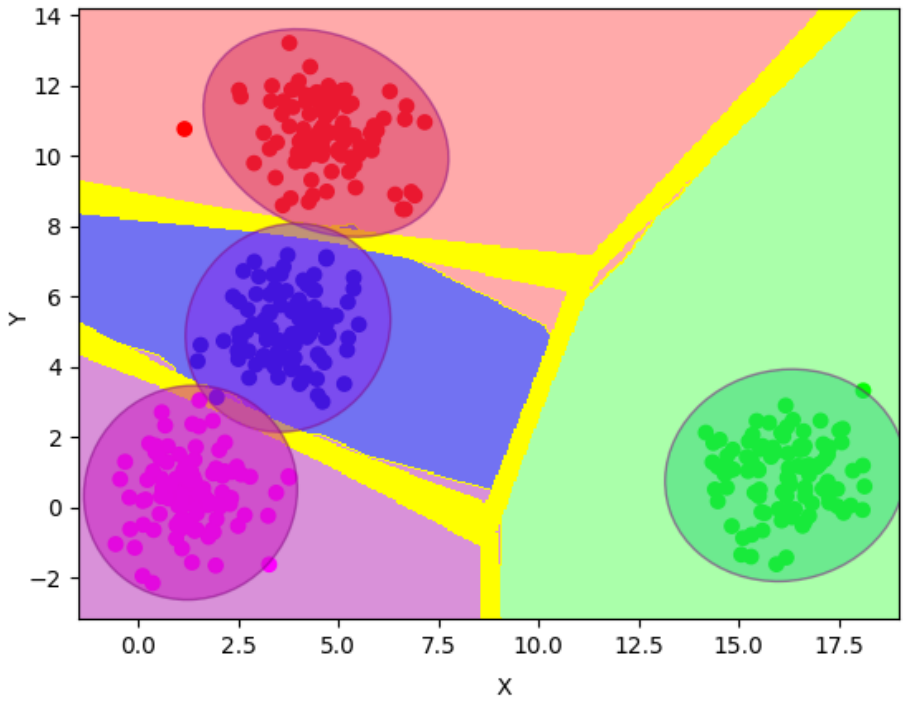


Рисунок 29 – Результат выполнения функции MCD

После получения областей принадлежности к какому-либо из классов можно переходить к следующему этапу реализации алгоритма, а именно к объединению новых классов.

### 2.7.6 Реализация алгоритма объединения новых классов

Объединение новых классов в один кластер является ключевой особенность разработанного алгоритма. Такое объединение позволяет дополнять число классов в стандартный алгоритм KNN и дает возможность ему корректно работать с неизвестными ранее данными. Опишем реализацию далее.

После выполнения классификации возникает следующая ситуация. На рисунке 30 изображены данные после обработки их модифицированным классификатором k-ближайших соседей.

Теперь среди этих данных необходимо оставить только новые классы. Выполним простую фильтрацию по типу точки. Если точка принадлежит

тренировочным данным или принадлежит к конкретному классу их исходных, то она отбрасывается. В результате мы получаем следующую картину (рисунок 31).

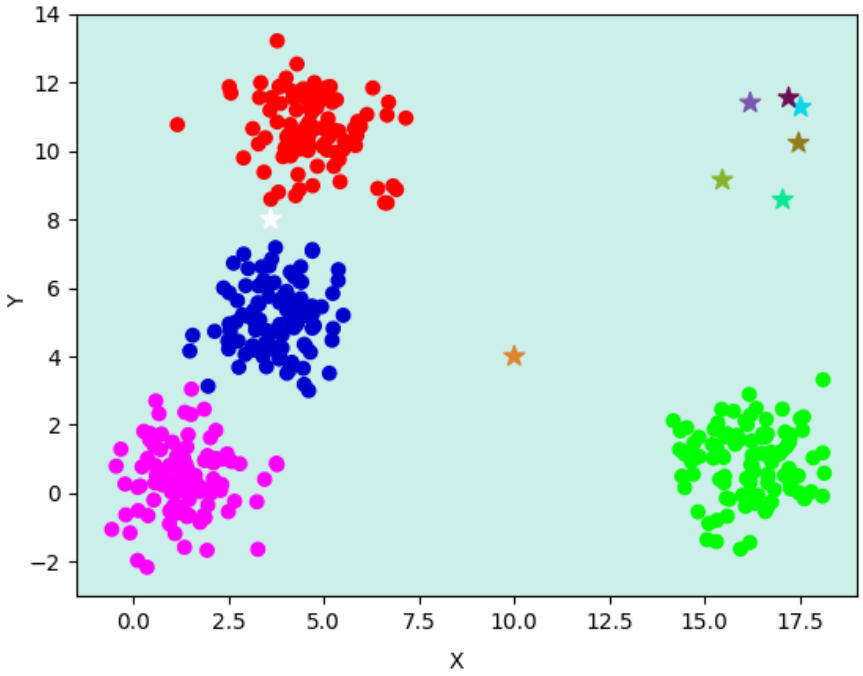


Рисунок 30 – Данные после обработки их модифицированным классификатором k-ближайших соседей

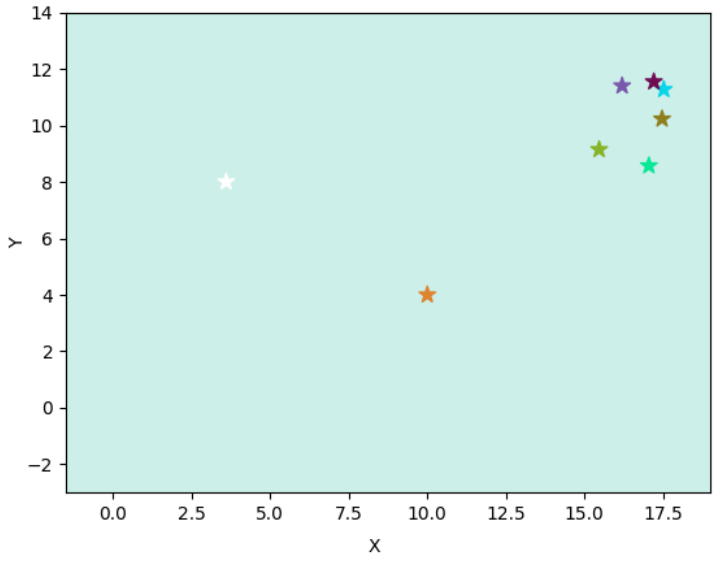


Рисунок 31 – Данные после фильтрации

В таком виде будут приходить данные на алгоритм DBSCAN. Его задача состоит в определении числа новых классов и объединении классов образующих один кластер. Как известно алгоритм DBSCAN умеет отсеивать шумы, но в нашем случае мы не желаем терять данные и хотим сохранять любые точки. Для этого мы задаем каждой точке задаем один из двух типов. Тип А – говорит о том, что алгоритм DBSCAN принял это тучку как точку принадлежащему одному из кластеров. Тип Б – говорит о том, что алгоритм DBSCAN принял точку как шум. Такая классификация точек потребуется в дальнейшей работе модифицированного алгоритма k-ближайших соседей. В листинге 8 представлен код применения алгоритма DBSCAN.

Листинг 8 - Код применения алгоритма DBSCAN

```
db = DBSCAN(eps=0.3, min_samples=10).fit(X)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)
unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]
    class_member_mask = (labels == k)
    xy = X[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o',
markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=14)
    xy = X[class_member_mask & ~core_samples_mask]
```

```
    plt.plot(xy[:, 0], xy[:, 1], 'o',
markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=6)
plt.title('Estimated number of clusters: %d' % n_clusters_)
plt.show()
```

В результате будет получен следующее изображение (рисунок 32).

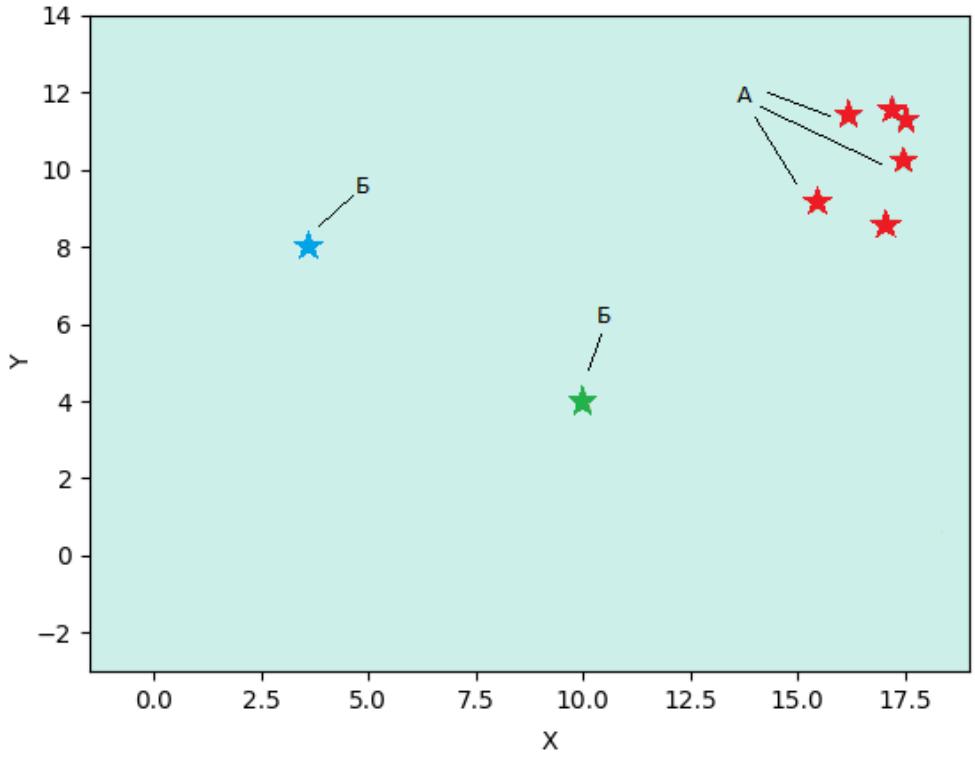


Рисунок 32 – Точки после выполнения алгоритма DBSCAN

Как можно заметить сгруппированные точки алгоритм принял за один кластер и присвоил им тип А. Одиночным же точкам алгоритм присвоим тип Б, то есть воспринял их как шум. В нашем же случае они по-прежнему являются новыми классами, но во время классификации они будут обрабатываться различным образом.

В результате выполнения алгоритма получается совершенно новый класс, который далее будет учитываться в вычислениях и к нему могут быть отнесены другие попадающиеся точки рисунок 33.

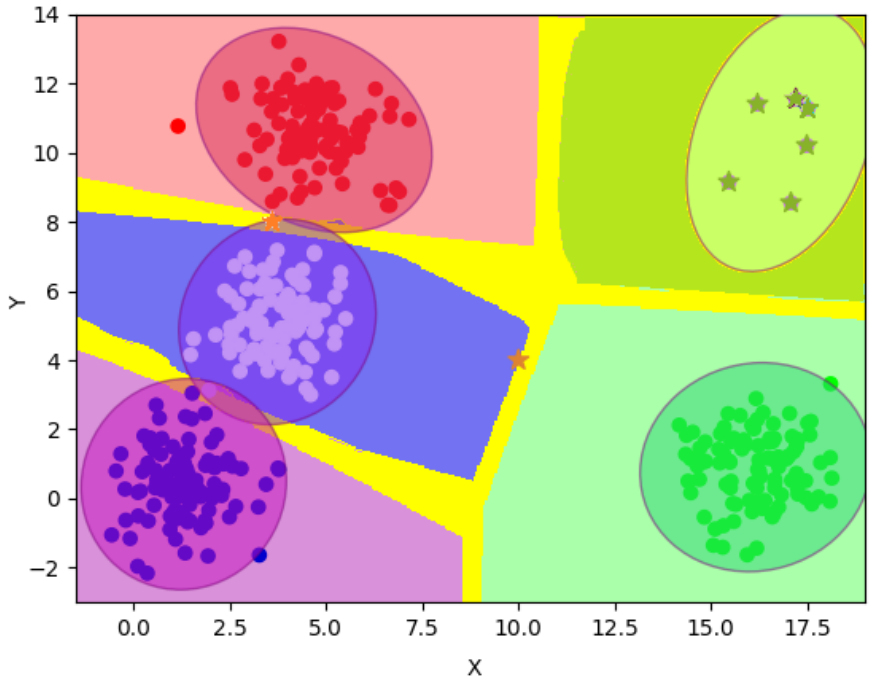


Рисунок 33 – Результат синтеза и объединения нового класса модифицированного алгоритма KNN

В результате удалось решить поставленные задачи в реализации модифицированного алгоритма KNN. Итоговая реализация алгоритма является достаточно простой и понятной, а также использует известные алгоритмы. Также удалось реализовать визуализацию работы алгоритма, которая может помочь в обнаружении аномалий и в корректировки параметров алгоритма.

# Глава 4 Анализ полученных результатов и дальнейшее развитие алгоритма

## 4.1 Сравнение модифицированного алгоритма KNN и оригинального алгоритма KNN

Основным отличием оригинального алгоритма KNN от модифицированного является то, что оригинальный алгоритм учится один раз и затем выполняет классификацию по заранее определенной разметке. Модифицированный же алгоритм не зависит от заранее определенного числа классов и способен доучиваться, то есть число классов в классификаторе может меняться. Наглядно это продемонстрировано на рисунке 34.

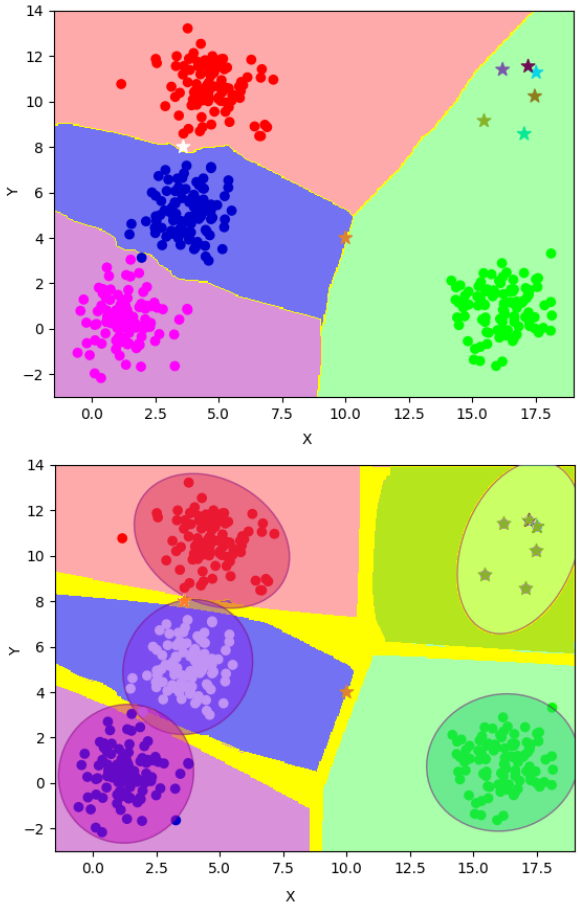


Рисунок 34 – Сравнение оригинального алгоритма KNN с модифицированным

На верхней части рисунка 34 изображен оригинальный алгоритм, а на нижней его модификация. Как видно число классов в оригинальном алгоритме не меняется на протяжении его работы хотя и очевидно, что новые данные не вписываются в текущее представление. Модифицированный алгоритм способен замечать такие аномалии и выделять их в отдельный класс.

## 4.2 Применение модифицированного алгоритма KNN

Одно из мест, где может применятся разработанный алгоритм так это обнаружение аномалий.

Обнаружение аномалий (также известный как анализ выбросов) — это этап интеллектуального анализа данных, на котором выявляются точки данных, события и / или наблюдения, которые отклоняются от нормального поведения набора данных. Аномальные данные могут указывать на критические инциденты, такие как технический сбой, или потенциальные возможности, например, изменение поведения потребителей. Машинное обучение все чаще используется для автоматизации обнаружения аномалий.

Теперь, когда доступны все аналитические программы и различное программное обеспечение для управления, компаниям стало проще, чем когда-либо, эффективно измерять каждый аспект деловой активности. Это включает в себя операционную производительность приложений и компонентов инфраструктуры, а также ключевые показатели эффективности (KPI), которые оценивают успех организации. Имея миллионы показателей, которые можно измерить, компании, как правило, получают довольно впечатляющий набор данных для изучения эффективности своего бизнеса. В этом наборе данных есть шаблоны данных, которые представляют обычный бизнес. Неожиданное изменение в этих шаблонах данных или событие, которое не соответствует ожидаемому шаблону данных, считается аномалией.

Другими словами, аномалия — это отклонение от привычного поведения, но что тогда мы подразумеваем под «обычным бизнесом», когда

речь идет о бизнес-показателях? Конечно, мы не имеем в виду «неизменный» или «постоянный»; нет ничего необычного в том, что веб-сайт электронной коммерции собирает большой доход за один день - конечно, если этот день наступает в Киберпонедельник. В этом нет ничего необычного, потому что высокий объем продаж в Киберпонедельник — это хорошо зарекомендовавший себя пик естественного цикла деловой активности любого бизнеса с интернет-магазином. Действительно, было бы аномалией, если бы такая компания не имела высоких объемов продаж в Киберпонедельник, особенно если бы объемы продаж в Киберпонедельник за предыдущие годы были очень высокими. Отсутствие изменений может быть аномалией, если оно нарушает шаблон, который является нормальным для данных из этой конкретной метрики. Аномалии категорически не являются хорошими или плохими, это просто отклонения от ожидаемого значения метрики в данный момент времени.

Успешное обнаружение аномалий зависит от способности точно анализировать данные временных рядов в реальном времени. Данные временных рядов состоят из последовательности значений во времени. Это означает, что каждая точка обычно представляет собой пару из двух элементов - метку времени, когда была измерена метрика, и значение, связанное с этой метрикой в это время.

Данные временных рядов сами по себе не являются прогнозом. Скорее, это запись, содержащая информацию, необходимую для обоснованных предположений о том, чего можно разумно ожидать в будущем. Системы обнаружения аномалий используют эти ожидания для выявления действенных сигналов в ваших данных, выявляя выбросы в ключевых KPI, чтобы предупредить вас о ключевых событиях в вашей организации.

В зависимости от бизнес-модели и варианта использования обнаружение аномалий данных временных рядов может использоваться для таких ценных показателей, как:

– Просмотры веб-страниц;

– Ежедневно активные пользователи;
– Установки мобильного приложения;
– Стоимость за клик;
– Затраты на привлечение клиентов;
– Показатель отказов;
– Скорость оттока;
– Доход от клика;
– Объем транзакций;
– Средняя стоимость заказа.

Обнаружение аномалий данных временных рядов должно сначала создать основу для нормального поведения основных ключевых показателей эффективности. При понимании этой базовой линии системы обнаружения аномалий данных временных рядов могут отслеживать сезонность - циклические модели поведения в ключевых наборах данных. Ручной подход может помочь определить сезонные данные на одном графике данных. Но когда вам нужно масштабироваться до тысяч или миллионов метрик, необходимо автоматизировать отслеживание данных временных рядов и обнаружение аномалий, чтобы получить ценную бизнес-информацию

Так разработанный алгоритм может классифицировать доходность от продаж по дням и в случае резкого всплеска продаж в какой-то день. Алгоритм сочтет это за аномалию, подаст сигнал и запомнит этот класс, но в случае, если в этот же день произойдет такой же всплеск. Алгоритм уже знает о нем и не будет выдавать предупреждение, а просто присвоит данные известному классу. Так тот же Киберпонедельник будет уже известным классом.

## 4.3 Дальнейшее развитие алгоритма

Алгоритм может быть модифицирован для повышения точности на областях с особой формы. Так для выпуклых областей можно применить

сглаживание областей, построенных с помощью выпуклой оболочки. Может быть применен метод построения кубического сплайна.

В математической области численного анализа сплайн-интерполяция — это форма интерполяции, в которой интерполянт представляет собой особый тип кусочно-полинома, называемый сплайном. То есть, вместо того, чтобы подгонять один полином высокой степени ко всем значениям одновременно, сплайн-интерполяция подгоняет полиномы низкой степени к небольшим подмножествам значений: например, подгонка девяти кубических многочленов между каждой из пар из десяти точек, вместо того, чтобы подгонять ко всем один многочлен десятой степени.

Сплайн-интерполяция часто предпочтительнее полиномиальной, потому что ошибка интерполяции может быть небольшой даже при использовании полиномов низкой степени для сплайна. Сплайн-интерполяция также позволяет избежать проблемы явления Рунге, при котором колебания могут возникать между точками при интерполяции с использованием многочленов высокой степени.

## Заключение

Главной отличительной чертой искусственного интеллекта, как и следует из названия термина, является его искусственность. Так как искусственный интеллект является продуктом интеллектуальной деятельности человека, то он априори не может превзойти своего создателя. Связано это прежде всего с тем, что сам человек, создавая искусственный интеллект, находится под воздействием определенных ограничений, задаваемых его пространством существования. На текущий момент указанная проблема не решена, что характеризуется определенным застоем в развитии этой области, поскольку достижение того же уровня качества, например, творческой деятельности искусственного интеллекта по сравнению с человеческим, не представляется легко разрешимой задачей.

«Мышлению» искусственного интеллекта не свойственна гибкость и творческая функция человеческого мышления. Как было показано в рассмотренных примерах, могут существовать ситуации, когда объект не принадлежит ни к одному из тренировочных классов, или принадлежит сразу нескольким классам. В случае, когда классифицируемый объект, по мнению алгоритма искусственного интеллекта на 50% принадлежит к классу 1, и на 50% к классу 2 алгоритм выдает равновероятностное распределение меток класса. Истоки такой проблемы кроятся в том, что обучение алгоритма классификации проводится человеком, и если «учителем» было заложено в математическую модель классификации не более двух классов, то и интеллектуальный алгоритм будет в своей работе опираться ровно на два класса, не имея способности нарушить аксиомы своего пространства.

Как представляется, именно в связи с указанной причиной, методы интеллектуального управления в настоящее время довольно успешно применяется в промышленной сфере, поскольку при решении подобных задач пространство, в котором существует используемый в задаче «искусственный интеллект», достаточно хорошо формализуется и не подвержено

значительным изменениям с течением времени. Таким образом, ключевым отличием искусственного интеллекта от естественного является неспособность адаптироваться к изменившимся условиям пространства, если человеком-создателем не были учтены изначально, в момент создания искусственного интеллекта, все требуемые характеристики пространства его существования.

Разработка методов синтезирования новых классов, на основе имеющихся в обучающей выборке – один из потенциальных способов решения описанной проблемы машинного обучения

Для дальнейшего исследования необходимо формализовать задачи синтеза классов основываясь на одном или нескольких базовых классов. Разработать математические модели каждого из предполагаемых методов решения проблемы, смоделировать процесс их выполнения. Выявить наиболее эффективную и достоверную из построенных моделей на основе эмпирического исследования.

## Список используемой литературы


1. Воронцов К.В. Лекции по методу опорных векторов // Вычислительный центр им. А.А. Дородницына Российской академии наук Федерального исследовательского центра «Информатика и управление» Российской академии наук, 2007.
2. Германова В. А., Маяцкая О. Б., Юнусова И. Р. Техника, технологии и системы с искусственным интеллектом как инструмент самопознания и творчества человека // Евразийский юридический журнал. - 2018. - №10 (125).
3. Гилимханов Р. Р., Минкин А.В. Возможности искусственного интеллекта в творчестве // Форум молодых ученых. - 2018. - №10 (26).
4. Дряева Х. Ш., Цораев Э.Ч. Применение и исследование искусственного интеллекта // Научно-техническая конференция обучающихся и молодых ученых СКГМИ (ГТУ) "НТК-2018". - Владикавказ: Северо-Кавказский горно-металлургический институт (Государственный технологический университет), 2018.
5. Карпенко И.Д. Искусственный интеллект// Новейший философский словарь. Мн.: Интерпресссервис; Книжный Дом, 2001. С.442.
6. Кудряшев А.Ф., Елхова О.И. Процесс творчества в системах с искусственным интеллектом // Вестник Башкирского университета. - 2016. - №4.
7. Куликов Д. К. Особенности мышления, или зеркало самосознания для искусственного интеллекта // Инженерный вестник Дона, ч.2. - 2014. - №4.
8. Пушкарев А. В. Творчество и искусственный интеллект: постановка проблемы // Гуманитарные, социально-экономические и общественные науки. - 2014. - №12-1.
9. Раков Н. С., Пальмов С.В. Машинное творчество // Форум молодых ученых. - 2018. - №4 (20).

10. Расторгуев С.П. О вложенности пространств (теория эгрегоров) / Сергей Расторгуев. — [б. м.] : Издательские решения, 2016. — 164 с.

11. Стрижаков Е. А., Чирикова М. В. История и перспективы развития творческих способностей искусственного интеллекта // Известия лаборатории древних технологий. - 2019. - №2 (31).

12. Струк П. В. Конфликт из-за искусственного интеллекта. Разные типы исследования // Теория и практика современной науки. - 2019. - №7 (49).

13. Уланова А. Е. Творчество естественного и искусственного интеллекта: границы и перспективы // Творчество как национальная стихия: медиа и социальная активность. - СПб.: Санкт-Петербургский государственный экономический университет, 2018.

14. Ambrozy. M Searching for a new criterion determining the machine's ability to think // SGEM 2016, BK 3: Anthropology, archelogy, history & philosophy conference proceedings, VOL II. - Albena, Bulgaria: STEF92 technology LTD, 1 Andrey Lyapchev BLVD, Sofia, 1797, Bulgaria, 2016.

15. Arthur Zimek, Erich Schubert. Outlier Detection // Encyclopedia of Database Systems. — Springer New York, 2017.

16. Arthur Zimek, Peter Filzmoser. There and back again: Outlier detection between statistical reasoning and data mining algorithms // Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. — 2018.

17. Bruder. J Infrastructural intelligence: Contemporary entanglements between neuroscience and AI // Vital models: The Making and use of models in the brain sciences. - Basel: Elsevier science BV, Sara Burgerhartstraat 25, PO BOX 211, 1000 AE Amsterdam, Netherlands, 2017.

18. Chang. JCH Black Box of Human Consciousness and "Two Cultures Dispute" Reading David Lodge's Thinks... // Chung WAI Literary Quarterly. - Taiwan: NATL Taiwan UNIV Press, No 1 Section 4, Roosevelt RD, Taipei, 106, Taiwan, 2017.

19. de Saint Laurent, In Defence of Machine Learning: Debunking the Myths of Artificial Intelligence // Europes journal of psychology. - 2018. - №4.

20. Dimensions of Creativity, MIT Press, Cambridge, MA, 1994.

21. Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, Xiaowei Xu. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN // ACM Trans. Database Syst. — 2017. — Июль (т. 42, вып. 3). — ISSN 0362-5915.

22. Gazoni R.M. Creative Thinking in Artificial Intelligence: a Peircean Account // 2016 International conference on computational science & computational intelligence (CSCI). - Las Vegas, NV: IEEE, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2016.

23. Hans-Peter Kriegel, Erich Schubert, Arthur Zimek. The (black) art of runtime evaluation: Are we comparing algorithms or implementations? // Knowledge and Information Systems. — 2016.

24. Havlik. V The naturalness of artificial intelligence from the evolutionary perspective // AI & Society. - 2019. - №4.

25. M. Boden, Creativity and artificial intelligence, Artificial Intelligence, 1998.

26. Park. Y Can Artworks by Artificial Intelligence be Artworks? // AM Journal of ART and media studies. - 2019. - №20.

27. Ricardo J. G. B. Campello, Davoud Moulavi, Arthur Zimek, Jörg Sander. Hierarchical Density Estimates for Data Clustering, Visualization, and Outlier Detection // ACM Transactions on Knowledge Discovery from Data. — 2015.

28. S. Schaffer, Making up discovery, in: M.A. Boden (Ed.),

29. Silapachote. P, Srisuphab. A Engineering Courses on Computational Thinking Through Solving Problems in Artificial Intelligence // International journal of engineering pedagogy. - 2017. - №3.

30. V. C. Muller and N. Bostrom, “Future progress in artificial intelligence: A survey of expert opinion,” in Fundamental issues of artificial intelligence. Springer, 2016.