Н. Н. Самылкина, А. А. Салахова,
Московский педагогический государственный университет

ОСНОВЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА В ШКОЛЬНОМ КУРСЕ ИНФОРМАТИКИ: ИСТОРИЯ ВОПРОСА И НАПРАВЛЕНИЯ РАЗВИТИЯ

Аннотация
В статье рассмотрена история включения вопросов, связанных с изучением искусственного интеллекта, в школьные учебники информатики. Предложена современная концепция рассмотрения этой темы на уроках в школе, основанная на практической реализации задач с использованием возможностей языка Python.

Ключевые слова: искусственный интеллект, экспертная система, интеллектуальный самообучающийся масштабируемый алгоритм APriori, Python.
DOI: 10.32517/2221-1993-2019-18-7-32-39

Несмотря на свою популярность в настоящее время вопросы искусственного интеллекта (ИИ) в содержании курса информатики освещаются довольно слабо и в основном исключительно с теоретической точки зрения. Отсутствие в учебниках практических работ по этой теме не позволяет сформировать у обучаемых необходимые компетенции для применения современных инструментов на основе интеллектуальных систем.

После 2017 года произошел резкий скачок в развитии технологий, связанных с искусственным интеллектом и Data Science, увеличилось количество открытых проектов и исследований, что только усилило разрыв между теорией, изложенной в учебниках, и практикой применения ИИ. Развитие высокотехнологичных проектов подтолкнуло к росту числа открытых фреймворков, библиотек и других инструментов, которые могут быть применены не только программистами, но и непрофессионалами. Появление подобных инструментов с простыми командами, зачастую понятными без чтения документации, позволяет использовать их при рассмотрении сложных тем в школе как на углубленном, так и на базовом уровне изучения информатики.

Джон Маккарти, считающийся автором термина «искусственный интеллект», в 1956 году на конференции в Дартмутском университете привел следующее определение: «**Искусственный интеллект** (*англ.* artificial intelligence, AI) — наука и технология создания интеллектуальных машин, особенно интеллектуальных компьютерных программ. Искусственный интеллект связан со сходной задачей использования компьютеров для понимания человеческого интеллекта, но не обязательно ограничивается биологически правдоподобными методами» [13].

В мире есть разные школы искусственного интеллекта, различающиеся в основном в подходах к пониманию проблем ИИ, и это, как следствие, влияет на направления развития ИИ, а также определяет понимание основного определения данной науки. В России традиционно существует единая школа искусственного интеллекта Д. А. Поспелова, возникшая еще в СССР в 1960-х годах на механико-математическом факультете МГУ имени М. В. Ломоносова [6].

В 1974 году был создан Научный совет при Президиуме АН СССР по проблеме «Искусственный интеллект», возглавляемый Д. А. Поспеловым и его заместителем Л. И. Микуличем. Деятельность совета была распределена на несколько проектов по разным направлениям ИИ [2]:

- «Диалог» — понимание естественного языка (под руководством А. П. Ершова, А. С. Нариньяни);

Контактная информация

Самылкина Надежда Николаевна, канд. пед. наук, доцент, профессор кафедры теории и методики обучения математике и информатике, Московский педагогический государственный университет; *адрес:* 119991, г. Москва, ул. Малая Пироговская, д. 1, стр. 1; *e-mail:* nsamylkina@yandex.ru

Салахова Алёна Антоновна, аспирант кафедры теории и методики обучения математике и информатике, Московский педагогический государственный университет; *адрес:* 119991, г. Москва, ул. Малая Пироговская, д. 1, стр. 1; *e-mail:* aa.salakhova@yandex.ru

N. N. Samylkina, A. A. Salakhova,
Moscow Pedagogical State University

THE BASICS OF ARTIFICIAL INTELLIGENCE AT SCHOOL INFORMATICS COURSE: BACKGROUND AND DIRECTIONS OF DEVELOPMENT

Abstract
The article describes the history of the inclusion of issues related to the study of artificial intelligence in school informatics textbooks. A modern concept is proposed for considering this theme at school lessons, based on the practical implementation of tasks using the capabilities of the Python language.

Keywords: artificial intelligence, expert system, intelligent self-learning scalable algorithm APriori, Python.

- «Банк» — банки данных (под руководством Л. Т. Кузина);
- «Ситуация» — ситуационное управление (под руководством Д. А. Поспелова);
- «Интеллект робота» — робототехника и экспертные системы (под управлением Д. Е. Охомицкого);
- «Конструктор» — поисковое конструирование (под руководством А. И. Половинкина).

Что касается изучения ИИ в школе, то основы искусственного интеллекта представлены в нескольких УМК для старшеклассников, где в основном рассматриваются экспертные системы (ЭС). Исторически экспертные системы — одно из самых первых направлений в ИИ, и в СССР оно развивалось активнее, чем нейронные сети, особенно в контексте теории управления. Знакомство школьников с ИИ через ЭС позволяет перейти к изучению искусственного интеллекта после изучения информационных систем, где упор делается на базы данных и управление ими.

Экспертная система — это специальный программный комплекс, который автоматизирует процессы поиска и принятия решений, обычно в конкретной предметной области [4]. Она состоит из фактов, базы знаний и машины логического вывода.

Школьники знакомятся с экспертными системами, например, в учебнике Н. Д. Угриновича для IX класса в параграфе «Экспертные системы распознавания химических веществ» [10], затем углубленно — в учебнике для X—XI классов (для естественно-математического профиля) [9], причем в этом учебнике даже название темы остается прежним. В указанном параграфе автор рассматривает экспертные системы и самообучающиеся алгоритмы как фактически тождественные понятия. В учебнике вводится понятие дерева решений, но практическая реализация популярного алгоритма в УМК Н. Д. Угриновича не предусмотрена.

Это же понятие «дерево решений» широко представлено (но уже как основа самообучающихся алгоритмов) в курсе И. А. Калинина, Н. Н. Самылкиной [4] при рассмотрении теоретической основы алгоритма CART, пересекающегося с темой бинарных деревьев поиска, рассматриваемой в содержательной линии «Алгоритмизация и программирование». На усмотрение учителя возможна практическая реализация алгоритма на изучаемом языке программирования. Авторы УМК также рассматривают экспертные системы, связывая их в том числе с известными проектами семантических сетей, например, с Semantic Web.

В остальных УМК по информатике, по которым ведется обучение в школе, экспертные системы (как представители ИИ) в основном рассматриваются при изучении баз данных.

Так, в учебнике К. Ю. Полякова, Е. А. Еремина для XI класса (углубленный уровень) [5] предлагается построить дерево решений для задачи, придуманной обучающимися (но решение не приводится).

В учебнике А. Г. Гейна, А. Б. Ливчака, А. И. Сенокосова, Н. А. Юнерман для X класса (базовый уровень) [3] экспертные системы описываются в параграфе «Базы знаний и экспертные системы», причем там рассматривается и программирование экспертной системы на языке Prolog.

Для курса информатики, который ведется по УМК И. Г. Семакина и соавторов, для изучения искусственного интеллекта предлагается курс по выбору [8], который основан на учебном пособии Л. Н. Ясницкого «Искусственный интеллект. Элективный курс» [11]. Вторая глава этого учебного пособия полностью посвящена экспертным системам, однако в основном внимание уделено моделям представления знаний и переходу к нейронным сетям.

В систематическом курсе С. А. Бешенкова, Н. В. Кузьминой, Е. А. Ракитиной для XI класса [1] в параграфе «Системы искусственного интеллекта» приводятся свойства экспертных систем:

- способность рассуждать при неполных или противоречивых данных;
- четкое разделение фактов и механизмов вывода;
- четкий (однозначный) совет на выходе;
- система должна быть экономически выгодной;
- цепочка рассуждений должна быть представлена понятным для пользователя образом.

Затем вводится еще одно свойство — обязательная самообучаемость экспертной системы. Данный тезис напрямую связан с задачей оптимизации ресурсов. Однако практическая реализация в рамках данной темы авторами не предусмотрена.

В настоящее время, когда усилилось внимание к изучению программирования на языке Python и строить графы начинают в основной школе, можно предложить обучающимся и **практическую реализацию темы «Экспертные системы».**

Для демонстрации работы с экспертными системами мы используем **библиотеку PyKnow языка Python**. Идейным основанием для нее стала другая широко распространенная библиотека (оболочка) CLIPS, работающая с базами знаний. Однако, в отличие от предшественницы, PyKnow представлена в более удобном виде для интеграции в программы и для комбинирования ее инструментов со средствами других библиотек.

При составлении условия задачи для демонстрации работы с экспертными системами за основу взят стандартный пример Python zebra.py, в котором рассматривается решение задачи про определение владельца зебры среди пяти жильцов разных домов. Но если решение оригинальной задачи — таблица из пяти строк и пяти столбцов, то в учебном примере решение формируется в формате 3×4.

Сама задача сводится к «игре в имитацию», как дань памяти об Алане Тьюринге и его статье*.

Задача.

Всего в доме есть четыре комнаты: в одной (гостиной) сидят игроки (жюри), пьющие чай, в трех других — загадочные личности. Игроки могут рассуждать вслух, решая, кто находится за каждой из оставшихся дверей. Здесь мы отметим, что двери сделаны из матового «запыленного» стекла, поэтому все мы видим силуэты женщин и их спокойные движения. Ответы на вопросы звучат женским голосом. Подобное не удивляет, поскольку один из дже-

* «Игра в имитацию» — игра, описанная в знаменитой статье А. Тьюринга «Может ли машина мыслить?»

миноидов* работает моделью в модном бутике в Токио, другой — администратором в отеле, а третий и вовсе снялся в фильме. Вместе с игроками в гостиной находится эксперт-компьютер. У каждой девушки есть история или легенда, которую жюри записывает в виде анкет с указанием имени девушки, места ее рождения и профессии. Участницы приехали из разных городов: Афин, Оксфорда, Лос-Анджелеса, но все говорят на английском языке.

Однако девушки не всегда рассказывают про себя. Иногда они добавляют фразы про своих соседок. Жюри узнало, что...

Девушка из первой комнаты назвалась Алисой. Она сказала, что никогда не видела Голливуд — известнейший район в Лос-Анджелесе, о котором рассказала живущая там актриса. В одной из комнат находится также известная писательница. Говорливая Барбара заверила, что никогда не была в Греции или Англии. Кассандра находится через одну дверь от Алисы. До интервью ей сказали, что в отборе участвует физик из Оксфорда — она была бы рада пообщаться, чтобы заимствовать образ для книги.

Мы знаем, что жительницы Оксфорда не существует — это джеминоид, принадлежащий актрисе, забравшей робота из лаборатории в этом городе. Кто в какой комнате находится? Кто из девушек соврал и о чем?

Итоговое решение задачи представлено в таблице 1.

Таблица 1

Решение задачи, поставленной перед экспертной системой

Номер комнаты	Имя	Город	Профессия
1	Алиса	Оксфорд	Физик
2	Барбара	Лос-Анджелес	Актер
3	Кассандра	Афины	Писатель

Подключим к проекту необходимые библиотеки. Мы будем использовать в том числе библиотеку перечислений Enum, поскольку стандартные средства языка не поддерживают проверку типа данных. Данная библиотека позволяет реализовать ограничения множества допустимых значений для некоторого типа данных. Библиотека TextWrap потребуется для работы с форматированным текстом, а модуль IterTools — для оптимизированной и выгодной работы с собственными итераторами и бесконечными итерациями. Конкретный блок chain позволяет итерировать списки, а не только отдельные объекты в них.

```
from enum import Enum
from textwrap import wrap, dedent
from itertools import chain
```

Загрузим основную библиотеку PyKnow:

```
from pyknow import *
```

Создадим необходимые переменные с ограниченным множеством значений и оформим их как классы с соответствующими именами:

```
class Name(Enum):
  Alice = 0
  Barbara = 1
  Cassandra = 2

class City(Enum):
  Oxford = 0
  LA = 1
  Athens = 2

class Profession(Enum):
  physicist = 0
  actress = 1
  writer = 2
```

Объявим два класса для будущих фактов. В библиотеке PyKnow факты являются основным ресурсом, технически представленным словарем, а также допускается смешение разных типов данных. Если данные используются без ключа, то он формируется автоматически как индекс. Мы будем использовать два типа фактов с разными функциями внутри: значения и решения:

```
class Value(Fact):
  pass

class Solution(Fact):
  pass
```

Следуя критериям к экспертным системам, отдельно используем факты и механизмы вывода — KnowledgeEngine. Внутри будет срабатывать бесконечный итератор. Выведем как факт по умолчанию текст задачи:

```
class Geminoid(KnowledgeEngine):
```

Для декорирования всех правил библиотекой предусмотрен инструмент @DefFacts, в который можно внести задачу полным текстом.

```
@DefFacts()
def startup(self):
  print(dedent("""
    \t Всего в доме есть четыре комнаты: в одной
(гостиной) сидят игроки (жюри), пьющие чай, в трех
других - загадочные личности. Игроки могут
рассуждать вслух, решая, кто находится за каждой
из оставшихся дверей. Здесь мы отметим, что двери
из матового "запыленного" стекла, поэтому все мы
видим силуэты женщин и их спокойные движения.
Ответы на вопросы звучат женским голосом. Подобное
не удивляет, поскольку один из джеминоидов
работает моделью в модном бутике в Токио, другой -
администратором в отеле, а третий и вовсе снялся
в фильме. Вместе с игроками в гостиной находится
эксперт-компьютер. У каждой девушки есть история
или легенда, которую жюри записывает в виде анкет
с указанием имени девушки, места ее рождения
и профессии. Участницы приехали из разных городов:
Афин, Оксфорда, Лос-Анджелеса, но все говорят на
английском языке.
     \t Однако девушки не всегда рассказывают про
себя. Иногда они добавляют фразы про своих
соседок. Жюри узнало, что...
     \n
     \t Девушка из первой комнаты назвалась Алисой.
Она сказала, что никогда не видела Голливуд -
известнейший район в Лос-Анджелесе, о котором
рассказала живущая там актриса. В одной из комнат
находится также известная писательница. Говорливая
Барбара заверила, что никогда не была в Греции или
Англии. Кассандра находится через одну дверь от
Алисы. До интервью ей сказали, что в отборе
участвует физик из Оксфорда - она была бы рада
пообщаться, чтобы заимствовать образ для книги.
```

* Джеминоид (*англ.* geminoid) — робот-андроид, созданный профессором Университета Осаки (Япония) Хироси Исигуро.

```
    \n
    \t Мы знаем, что жительницы Оксфорда не
существует - это джеминоид, принадлежащий
актрисе, забравшей робота из лаборатории в этом
городе. Кто в какой комнате находится? Кто из
девушек соврал и о чем?
    """))
```

Поскольку в анкете каждой девушки рассматривается сразу несколько свойств, для обработки фактов нужно рассматривать эти свойства взаимосвязанно. Здесь поможет инструмент из пакета IterTools для итерации комбинаций сразу нескольких объектов — рядов:

```
for x in chain(Name, City, Profession):
  yield Value(x)

  @Rule(AS.f << Value(MATCH.v))
  def generate_combinations(self, f, v):
    self.retract(f)
    self.declare(*[Fact(v, x) for x in range(1, 4)])
```

Прежде чем заниматься заполнением правил для экспертной системы, ***проверим разрешимость задачи***, перенеся ее на граф, поскольку база знаний представляет собой семантическую сеть (один из видов онтологий, изучаемых наукой искусственный интеллект при подходе представления знаний) [4].

Для того чтобы выявить, что задача может быть решена, определим достаточное и необходимое условия. Предположим, что в качестве достаточного условия выступает полный граф. Наличие связей между всеми вершинами графа является избыточным, поэтому необходимое условие — это построение любого связного графа на обозначенных вершинах, причем граф не обязательно должен быть деревом. Однако мы можем еще уточнить условия: должны быть связаны $(n-1)$ свойства одной категории. На естественном языке это понимается как факт, что последнее оставшееся свойство мы можем точно определить методом исключения.

Каждая вершина — это одно сведение, свойство или же (в более сложных случаях) искомая особенность (feature). Ребра графа — это существующие связи между свойствами, т. е. факты, учитываемые экспертной системой.

Стоит отметить, что мы реализуем простую экспертную систему с четкой логикой, поэтому для нас важно наличие связи или ее отсутствие (отрицание дает вес 0, наличие свойства — вес 1, логическую единицу). В случае систем с нечеткой логикой вес ребра может изменяться: обычно в отрезке [0; 1] или же [−1;1], где исключается 0 как отсутствие ребра как такового. То есть $[-1; 0) \cup (0; 1]$.

Построим подобный граф по условиям нашей задачи (рис. 1). Непрерывными линиями отмечены связи с весом 1, прерывистыми — связи с весом 0.

При решении мы по умолчанию учитываем, что свойства не повторяются: нет двух тезок, все девушки из разных городов и имеют разные профессии. Необходимое условие выполнено.

Составим правила. Все правила в PyKnow синтаксически разделены на две части: слева описываются условия, при которых правило будет выполнено, справа — набор действий, которые нужно исполнить при выполнении условий. Чтобы факт был истинным, все условия должны выполняться без исключения.

```
@Rule(
```

Пусть у нас есть:

- множество комнат — {a1, a2, a3};
- множество имен — {b1, b2, b3};
- множество городов — {c1, c2, c3};
- множество профессий {d1, d2, d3}.

Причем эти множества строго упорядочены. Ассоциации с элементами будут возникать по мере получения информации из текста. При решении задачи система при проверке связей переставит содержание элементов в нужном порядке.

Внесем в ЭС факт существования имени Алиса и информацию о том, что девушка с этим именем находится в первой комнате («Девушка из первой комнаты назвалась Алисой»). То есть в первой строке таблицы:

```
Fact(Name.Alice, MATCH.a1 & L(1)),
```

Добавим город Лос-Анджелес и внесем отрицание связи с Алисой («Она сказала, что никогда не видела Голливуд — известнейший район в Лос-Анджелесе»). Затем потребуется внести профессию «актриса», присвоить ей место во множестве и связать с городом («...известнейший район в Лос-Анджелесе, о котором рассказала живущая там актриса»).

```
Fact(City.LA, MATCH.b1 & ~MATCH.a1),
Fact(Profession.actress, MATCH.c1 & MATCH.b1),
```

Добавим профессию «писательница» («В одной из комнат находится также известная писательница»), однако следует указать, что при решении не может быть повторов (элементы внутри каждого множества не совпадают).

```
Fact(Profession.writer, MATCH.c2 & ~MATCH.a1 &
  ~MATCH.c1)
```

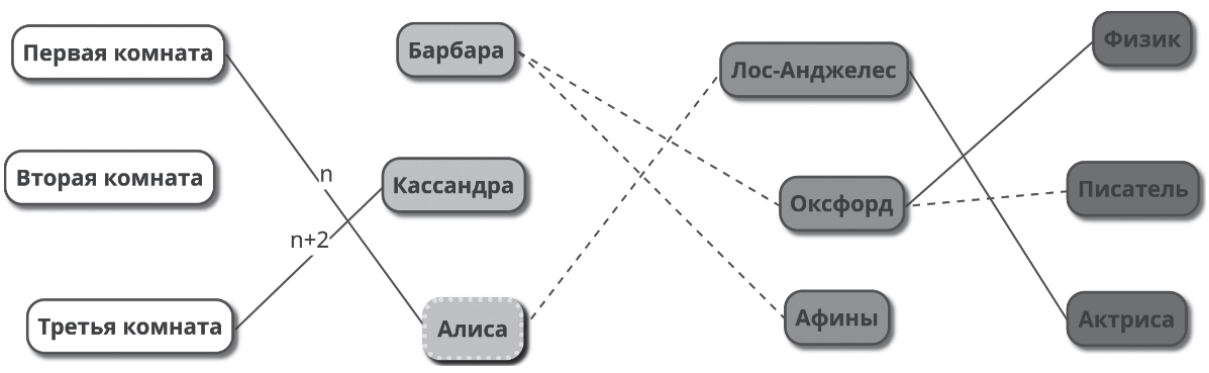

Рис. 1. Граф задачи

Внесем информацию о Барбаре и двух городах, в которых никогда она раньше не бывала («Говорливая Барбара заверила, что никогда не была в Греции или Англии»).

```
Fact(City.Oxford, MATCH.b2 & ~MATCH.b1),
Fact(City.Athens, MATCH.b3 & ~MATCH.b1 &
  ~MATCH.b2),
Fact(Name.Barbara, MATCH.a2 & ~MATCH.b2 &
  ~MATCH.b3 & ~MATCH.a1),
```

Для того чтобы указать информацию о Кассандре, понадобится использование отношения переменных внутри одного множества. Сначала выделим неявный факт: она пишет книги («она была бы рада пообщаться, чтобы заимствовать образ для книги»), следовательно, Кассандра — писатель. Также она рассуждает о человеке из Оксфорда («До интервью ей сказали, что в отборе участвует физик из Оксфорда»), следовательно, сама в нем не живет. Она находится через дверь от Алисы («Кассандра находится через одну дверь от Алисы»), значит, номер ее комнаты на два больше, чем номер комнаты Алисы. Наконец, укажем информацию про физика («физик из Оксфорда»).

```
Fact(Name.Cassandra, MATCH.a3 & MATCH.c2 &
  ~MATCH.b2),
TEST(lambda a3, a1: a3 == (a1+2)),
Fact(Profession.physicist, MATCH.c3 & MATCH.b2)
```

Позаботимся о поиске решения. Функция будет обращаться к самой себе в нескольких итерациях, пока не выведет устойчивое соответствие.

```
def find_solution(self, **match):
```

Прежде всего нужно объявить соответствие переменных из множеств и сущностей, связь между которыми анализируется:

```
self.declare(*[
  Solution(Name, 'Алиса', match["a1"]),
  Solution(City, 'Лос-Анджелес', match["b1"]),
  Solution(Profession, 'актриса', match["c1"]),
  Solution(Name, 'Барбара', match["a2"]),
  Solution(City, 'Оксфорд', match["b2"]),
  Solution(Profession, 'писатель', match["c2"]),
  Solution(Name, 'Кассандра', match["a3"]),
  Solution(City, 'Афины', match["b3"]),
  Solution(Profession, 'физик', match["c3"])])
```

Затем вводятся правила, по которым происходит решение. Количество строк зависит от свободных переменных, нумерация строк не учитывается. То есть остается таблица 3×3, что соответствует $3 \cdot 3 = 9$ решениям. Если бы у девушек еще указывалось хобби, необходимо было бы описать $4 \cdot 3 = 12$ решений.

Решения имеют форму факта, как и предыдущие соответствия, и составляются по правилам, которым мы присваиваем имена.

```
@Rule(
  AS.f1  << Solution(Name, MATCH.a1, 1),
  AS.f2  << Solution(City, MATCH.b1, 1),
  AS.f3  << Solution(Profession, MATCH.c1, 1),
  AS.f4  << Solution(Name, MATCH.a2, 2),
  AS.f5  << Solution(City, MATCH.b2, 2),
  AS.f6  << Solution(Profession, MATCH.c2 , 2),
  AS.f7  << Solution(Name, MATCH.a3, 3),
  AS.f8  << Solution(City, MATCH.b3, 3),
  AS.f9  << Solution(Profession, MATCH.c3, 3)
)
```

Еще одна важная функция внутри поиска решений — их вывод на экран. Поскольку название города чуть длиннее, чем остальные пункты, оставим больше символов в ширине ряда таблицы:

```
def print_solution(self,
     a1, a2, a3,
     b1, b2, b3,
     c1, c2, c3,
     **fs):
   for f in fs.values():
     self.retract(f)
   print("Дверь | %-10s | %-15s | %-10s " %
     ("Имя", "Город", "Профессия"))
   print("-----------------------")
   print("  1   | %-10s | %-15s | %-10s  " %
     (a1, b1, c1))
   print("  2   | %-10s | %-15s | %-10s  " %
     (a2, b2, c2))
   print("  3   | %-10s | %-15s | %-10s " %
     (a3, b3, c3))
   print("\n")
```

Осталось запустить машину вывода, приведя ее в исходное состояние, не содержащее лишних данных.

```
g = Geminoid()
g.reset()
g.run()
```

Сначала на экране будет воспроизведен текст условия, затем появится таблица с решением. Убедитесь, что экспертная система работает.

Экспертные системы и машины вывода в современном состоянии чаще всего рассматриваются в сочетании с **самообучающимися интеллектуальными алгоритмами**. Поскольку наиболее прикладной областью в науке о данных является **глубинный анализ данных (Data Mining)**, его изучение важно для профессионального самоопределения выпускников. Кроме того, результаты работы таких алгоритмов довольно демонстративны. Глубинный анализ появился как расширение бизнес-анализа (*англ.* Business Intelligence), он способен работать с большими данными, выявляя новые связи между признаками объектов и доступные интерпретации знаний, которые необходимы для принятия решений в различных областях деятельности человека.

Интеллектуальные алгоритмы можно реализовывать:

- на малых выборках — без применения средств автоматизации;
- в полуавтоматизированном режиме в табличном процессоре с применением сложных формул;
- с помощью языков программирования на двух уровнях: реализация с применением готовых инструментов и библиотек и самостоятельная реализация [7].

Кроме того, возможен промежуточный вариант — использование языков программирования и специальных надстроек для табличного процессора, однако большинство таких аддонов проприетарны (например, DataNitro). Естественно, в учебных примерах результаты будут отличны от случаев, когда работа проводится на настоящих больших данных, однако этого достаточно для ознакомления учащихся с принципом работы алгоритма.

В школьных УМК по информатике интеллектуальные алгоритмы рассматриваются в двух курсах — в элективном курсе по выбору Л. Н. Ясницкого [11] и в УМК

углубленного уровня для XI класса И. А. Калинина, Н. Н. Самылкиной [4]. В первом делается особый упор на математическое обоснование алгоритмов и применяемые формулы, во втором — на области применения и компьютерную реализацию алгоритмов. В УМК также дается обзор самых популярных алгоритмов Data Mining, включая алгоритм ассоциативных правил APriori.

Интеллектуальный самообучающийся масштабируемый **алгоритм APriori** (анализ потребительских корзин) был придуман в 1993 году сотрудником IBM Ракешем Агравалом [12] для поиска шаблонов покупок, которые совершают клиенты супермаркетов, с целью последующего выявления шаблонов поведения для составления акций, размещения продуктов на определенных полках и т. д. Довольно быстро алгоритм стали применять для поиска ассоциативных правил в различных областях: для прогноза заболеваемости, для выявления групп риска среди граждан, для поиска решений при управлении движением робота и даже для составления меню в ресторане. Сегодня APriori — самый популярный алгоритм поиска ассоциативных правил.

Ассоциативные правила (association rule mining) — это правила отношения к чему-либо, используемые для группировки данных, часто заменяемые конструкцией «if... then», аналогично правилам в нашей экспертной системе.

В основе алгоритма APriori лежит следующий факт: если набор из трех покупок встречается пять раз, то набор их двух любых покупок этого набора встречается не менее пяти раз. Именно данное правило позволяет реализовать масштабирование, поскольку данные о наборах, которые не являются часто встречающимися на текущем шаге, исключаются при дальнейшем анализе. Для ускорения работы алгоритма используют и манипуляции с «сырыми» данными, разбивая таблицы по разделам, например, выделяя в базе данных таблицы разных отделов магазина и исследуя транзакции только по выделенным категориям товаров [12]. У APriori существует множество вариантов реализации, образовавших целое семейство алгоритмов, самые известные из которых Apriori TID и APriori Hybrid.

Для оценки эффективности найденного правила вводят понятие «достоверность» (confidence), которая определяется как процент случаев, когда посылка (if) приводит к заключению (then) из общего числа возникновения такой посылки. Уровень достоверности, в свою очередь, влияет на дальнейшее применение правила: факт достоверен, если его уровень выше 60 %, а ниша 5–10 % показывает, что существуют единичные случаи, для которых правило работает всегда, — это ниша акций.

При работе с алгоритмом потребуется следующий **словарь понятий:**

- *множество товаров, корзина (item set)* — все различные наименования, которые могут встретиться при анализе, например, ассортимент магазина или перечень исследуемых полезных *свойств (features);*
- *товар (item)* — конкретный вид товара, наименование или исследуемое свойство;
- *транзакция (transaction)* — корзина покупок конкретного покупателя, его чек; в технической документации под транзакцией понимают входную запись, явно или неявно требующую обработки;
- *множество транзакций (transaction set)* — данные обо всех купленных товарах в магазине и их количестве, сводные данные всех чеков;
- *поддержка (support)* — встречаемость конкретного наименования или сочетания нескольких наименований среди всех транзакций;
- *минимальная поддержка (min support)* — минимальный порог встречаемости, задаваемый исследователем;
- *ассоциативное правило (association rule)* — правило отношения к чему-либо, используемое для группировки данных, которые проще всего записать конструкцией «если..., то ...» («if...then»), аналогично правилу, используемому в экспертной системе ранее;
- *надежность (confidence)* — вероятность соблюдения конкретного ассоциативного правила, не должна достигать пороговых значений (0 % и 100 %);
- *кандидат (candidate)* — товар или сочетания из k товаров для k-го шага, которые необходимо проверить на встречаемость;
- *множество кандидатов (candidate set)* — это все кандидаты, имеющиеся на данном k-м шаге;
- *часто встречаемый элемент (frequency)* — кандидат, поддержка которого выше минимальной;
- *набор часто встречаемых элементов (frequency set)* — множество всех часто встречаемых элементов, которые удалось найти.

Данные в исходной таблице с транзакциями приводят к нормализованному виду, заменяя имеющиеся продукты на единицу, а отсутствующие отмечая нулем. На каждом k-м шаге рассматриваются k-элементные множества кандидатов, формирующиеся из часто встречающихся наборов $(k - 1)$-го шага.

На первом шаге кандидатами являются все одноэлементные множества, сформированные из всех продуктов. Порядок внутри множеств не важен. Поддержка продуктов проверяется на каждом шаге как сумма встречающихся в транзакциях наборов (когда все элементы множества для конкретной транзакции имеют значение 1). Алгоритм продолжает работу до тех пор, пока есть множества с поддержкой, выше или равной минимальной, заданной исследователем.

Рассмотрим пример работы с кандидатами (табл. 2). Часто встречающиеся множества отмечены серым цве-

Таблица 2

Кандидаты и часто встречающиеся множества

Шаг (k)	Кандидаты (поддержка), мин. поддержка = 3				
1	Чай (5)	Кофе (5)	Сливки (3)	Конфеты (2)	Сахар (3)
2	Чай, кофе (1)	Чай, сливки (2)	Чай, сахар (3)	Кофе, сливки (3)	Кофе, сахар (3)
3	Кофе, сливки, сахар (2)				

том. На третьем шаге получилось составить только одно множество, так как в сочетании {чай, кофе, сливки} множество {чай, кофе} не было часто встречающимся.

Для каждого часто встречающегося множества можно составить как минимум два правила. Для нахождения достоверности необходимо разделить значение поддержки всего множества на значение поддержки элемента посылки. Например, для множества {чай, сахар} можно вывести:

- те, кто пьют чай, добавляют сахар:
  $(confidence = \frac{support\ \{\text{чай, сахар}\}}{support\ \{\text{чай}\}} = \frac{3}{5} = 60\,\%)$;
- те, кто любит сахар, добавляют его в чай:
  $(confidence = \frac{support\ \{\text{чай, сахар}\}}{support\ \{\text{сахар}\}} = \frac{3}{3} = 100\,\%)$.

Значение 100 % при работе с большими данными обычно сигнализирует об ошибке, однако часто встречается на учебных выборках.

Подробнее с теоретическими основами алгоритма можно ознакомиться в УМК И. А. Калинина, Н. Н. Самылкиной в главе 4 учебника для XI класса [4].

Перейдем к реализации алгоритма на языке Python с применением специализированной библиотеки Apryori.

Подключим необходимые библиотеки. Здесь потребуются две библиотеки для работы с представлением данных — это Nympy и Pandas.

```
import numpy as np
import pandas as pd
```

Из библиотеки Apryori нам понадобится только один модуль (обычная версия алгоритма):

```
from apyori import apriori
```

Далее импортируем данные из таблицы без заголовка. Сам файл должен находиться в той же папке, иначе надо прописать полный путь.

```
trans_set = pd.read_csv('prod.csv', header=None)
```

Библиотека Apryori требует использования списка списков. Данный пункт можно будет усовершенствовать при желании, используя инструменты библиотеки IterTools. Создаем таблицу с множеством транзакций:

```
list_trans_set = []
```

Теперь ее необходимо заполнить. Возможно, существует другой способ подсчитать колонки, но len() позволяет легко подсчитать размер любой таблицы в ширину. В документации к Pandas специфическую функцию, возвращающую не названия, а число колонок, авторам статьи найти не удалось.

```
for i in range(0, len(trans_set)):
  list_trans_set.append([str
  (trans_set.values[i,j]) for j in range
  (0, int(trans_set.size/len(trans_set)))])
```

Применим алгоритм APriori к данным, сразу преобразовав результат в список, указывая минимальную поддержку, достоверность и длину связей:

```
apriori_results = list(apriori(list_trans_set,
  min_support=0.3, min_confidence=0.05,
  min_length=3))
```

Перейдем к выводу результатов на экран. Мы будем выводить правила из трех элементов, причем правила будут пронумерованы, следовательно, потребуется создать счетчик этих правил.

```
print("=======Н=А=Ч=А=Л=О===Р=А=Б=О=Т=Ы======")
number=1
for item in range(0, len(apriori_results)):
```

Сразу отформатируем результат, выводимый одним блоком, в красивый локализованный вывод. Чтобы посмотреть, почему мы используем такой необычный сепаратор, взгляните на стандартный результат функции на примере print(apriori_results[1]) :

```
RelationRecord(items=frozenset({'булочка'}),
  support=0.4444444444444444, ordered_statistics
  =[OrderedStatistic(items_base=frozenset(),
  items_add=frozenset({'булочка'}),
  confidence=0.4444444444444444, lift=1.0)])
```

Читать подобный вывод достаточно неудобно. Требуется разбить информацию на блоки, границами которых станут знаки равенства.

```
current=str(apriori_results[item]).split('=')
```

При обработке образуются пропуски, которые при преобразовании строки стали текстовым значением «nan». В аргументах указана длина текстовых блоков. Пропустим наборы, где они участвуют:

```
if not "nan" in current[1]:
  print(f"========Набор № {number}========"[:50])
  print("|Набор:"+(current[1])[11:-11])
```

Поддержка в данной реализации алгоритма представлена как десятичная дробь. Вернем ей привычный вид целого числа:

```
print("|Поддержка:"+str(int(float((current[2])
  [:-20])*10)))
```

Мы оформляем частный случай, когда множества часто встречающихся вместе товаров не превосходят по мощности 3. Алгоритм для множества с мощностью 1:

```
if (current[1].count("'"))==2:
```

Оформим показание достоверности (confidence) в виде процентов, также округлим значение до двух знаков после запятой:

```
print("|Достоверность:"+str(round(float
  ((current[6])[:-6])*100,2))+"%")
```

Аналогично оформим вывод для множества с мощностью 2:

```
if(current[1].count("'"))==4:
  print("|Достоверность прямого правила:"+str
  (round(float((current[6])[:-6])*100,2))+"%")
  print("|Достоверность обратного правила:"+str
  (round(float((current[10])[:-6])*100,2))+"%")
```

Остается оформить вывод для множества с мощностью 3:

```
if (current[1].count("'"))==6:
  print("|Достоверность для двух последних
  к первому:"+str(round(float((current[6])
  [:-6])*100,2))+"%")
  print("|Достоверность первых двух к последнему:
  "+str(round(float((current[10])
  [:-6])*100,2))+"%")
  print("|Достоверность двух крайних к среднему:
  "+str(round(float((current[14])
  [:-6])*100,2))+"%")
```

```
  number=number+1
  print("====================================")
print("=======К=О=Н=Е=Ц====Р=А=Б=О=Т=Ы=======")
```

Внутри кода содержалась ссылка на таблицу в формате csv. Именно в эту таблицу заносятся данные, с которыми оперирует алгоритм. Первая строка заполняется всеми возможными наименованиями, в последующие строки вносятся сведения о покупках — наименования, учтенные в транзакциях. Алгоритм исследует файл без заголовка, однако именно перечисление всех наименований позволяет определить максимальную ширину таблицы.

Пример содержания такого файла:

```
хлеб, колбаса, масло, молоко, котлета, сметана,
  булочка, перец
хлеб, колбаса, масло, икра
булочка, масло, сметана, икра
хлеб, колбаса, котлета
перец, молоко, котлета
масло, молоко, булочка
хлеб, колбаса, масло, молоко, котлета
котлета, перец, икра
булочка, масло
икра, масло, хлеб
```

Содержание по смыслу не обязательно должно напоминать о прилавке магазина. Наиболее интересны для обучающихся задания, выполненные в формате кейсов с описаниями ситуаций, где они сами должны выявить важные наименования и их наличие. Подобные задания помогают сформировать у учащихся ассоциацию важности работы специалиста Data Science, который выделяет нужные для потребителя сущности (наименования). Тексты ситуаций в кейсах могут быть оформлены в форме шутливого рассказа или серьезной бизнес-корреспонденции — обучающиеся в любом случае вовлекаются в игру. Чем необычней или, наоборот, привычней ситуация, тем выше уровень мотивации. Например, кейс про кошачий приют содержит следующую запись: «Программист сказал, что идеальный кот должен писать код, но, если такие в приюте не водятся, то помещаться на рабочем столе, чтобы кота было удобно чесать». Из данного события-примера, представленного на естественном языке, исследователи выявляют черты, которые понравились конкретному человеку в животном: маленький, спокойный. Второе из приведенных свойств — неявное, его обнаружение полностью зависит от исследователя. Для специалиста Data Science важно понимать, что недостаточные данные не дают максимального результата при исследовании, но и избыточные данные могут негативно сказаться на анализе. Дополнительно подобный подход позволяет искоренить в сознании обучающихся миф, что искусственный интеллект и компьютеры полностью заменят человека.

Интеллектуальные алгоритмы максимально эффективны, когда их применяет квалифицированный специалист. Именно от него зависит, как работать и как поступить с данными. При решении кейсов у каждой группы или участника получаются индивидуальные результаты.

Подробнее примеры кейсов, практикумы для реализации без средств автоматизации (в формате лабораторной работы), практикум с решением с помощью табличного процессора представлены на странице авторского сайта: http://www.techlyc.ru/Apriori. Материалы этого сайта доступны для скачивания.

При желании для усложнения темы может быть использована платформа Microsoft Azure Machine Learning с образовательной лицензией в рамках программы Microsoft Imagine. Студия машинного обучения позволяет не только создать на облаке нейронную сеть с помощью графических блоков, но и в качестве блоков использовать модули нескольких интеллектуальных алгоритмов. Кроме того, недостающие или самостоятельно измененные (написанные) алгоритмы могут использоваться как блоки с кодом на языках Python или R с подключением необходимых библиотек.

Приведенные алгоритмы также могут быть применены как инструмент в проектной деятельности обучающихся X—XI классов или при подготовке исследований для других предметов (обществознание, биология, литература и т. д.).

Список использованных источников

1. *Бешенков С. А., Кузьмина Н. В., Ракитина Е. А.* Информатика. Систематический курс: 11 кл.: учебник для классов гуманитарного профиля. М.: БИНОМ. Лаборатория знаний, 2002. 206 с.

2. *Гаврилова Т. А., Хорошевский В. Ф.* Базы знаний интеллектуальных систем. СПб.: Питер, 2000. 384 с.

3. *Гейн А. Г., Ливчак А. Б., Сенокосов А. И., Юнерман Н. А.* Информатика и ИКТ: учебник для 10 класса общеобразовательных учреждений. М.: Просвещение, 2014. 272 с.

4. *Калинин И. А., Самылкина Н. Н.* Информатика. 11 класс. Углубленный уровень: учебник. М.: БИНОМ. Лаборатория знаний, 2013. 216 с.

5. *Поляков К. Ю., Еремин Е. А.* Информатика. 11 класс. Углубленный уровень: учебник в 2 ч. Ч. 1. М.: БИНОМ. Лаборатория знаний, 2013. 241 с.

6. *Поспелов Д. А.* Из истории искусственного интеллекта: история искусственного интеллекта до середины 80-х годов // Новости искусственного интеллекта. 1994. № 4. С. 70–90.

7. *Салахова А. А.* Изучение интеллектуальных алгоритмов на примере реализации Apriori в углубленном курсе информатики // Информатика. 2016. № 5/6. С. 3–17.

8. *Семакин И. Г., Ясницкий Л. Н.* О возможностях преподавания «искусственного интеллекта» в общеобразовательной школе. http://metodist.lbz.ru/lections/12/files/about.pdf

9. *Угринович Н. Д.* Информатика и информационные технологии: учебник для 10–11 классов. М.: БИНОМ. Лаборатория знаний, 2016. 512 с.

10. *Угринович Н. Д.* Информатика. 9 класс: учебник. ФГОС. М.: БИНОМ. Лаборатория знаний, 2015. 152 с.

11. *Ясницкий Л. Н.* Искусственный интеллект. Элективный курс: учебное пособие. М.: БИНОМ. Лаборатория знаний, 2012. 197 с.

12. *Agrawal R., Srikant R.* Fast algorithms for mining association rules in large databases // Proceedings of the 20th International Conference on Very Large Data Bases (VLDB). Santiago, Chile, 1994. https://vldb.org/conf/1994/P487.PDF.

13. *McCarthy J.* What is artificial intelligence? // Computer Science Department, Stanford University. November 12, 2007. http://www-formal.stanford.edu/jmc/whatisai.pdf

14. *Wasilewska A.* APRIORI Algorithm. http://www3.cs.stonybrook.edu/~cse634/lecture_notes/07apriori.pdf