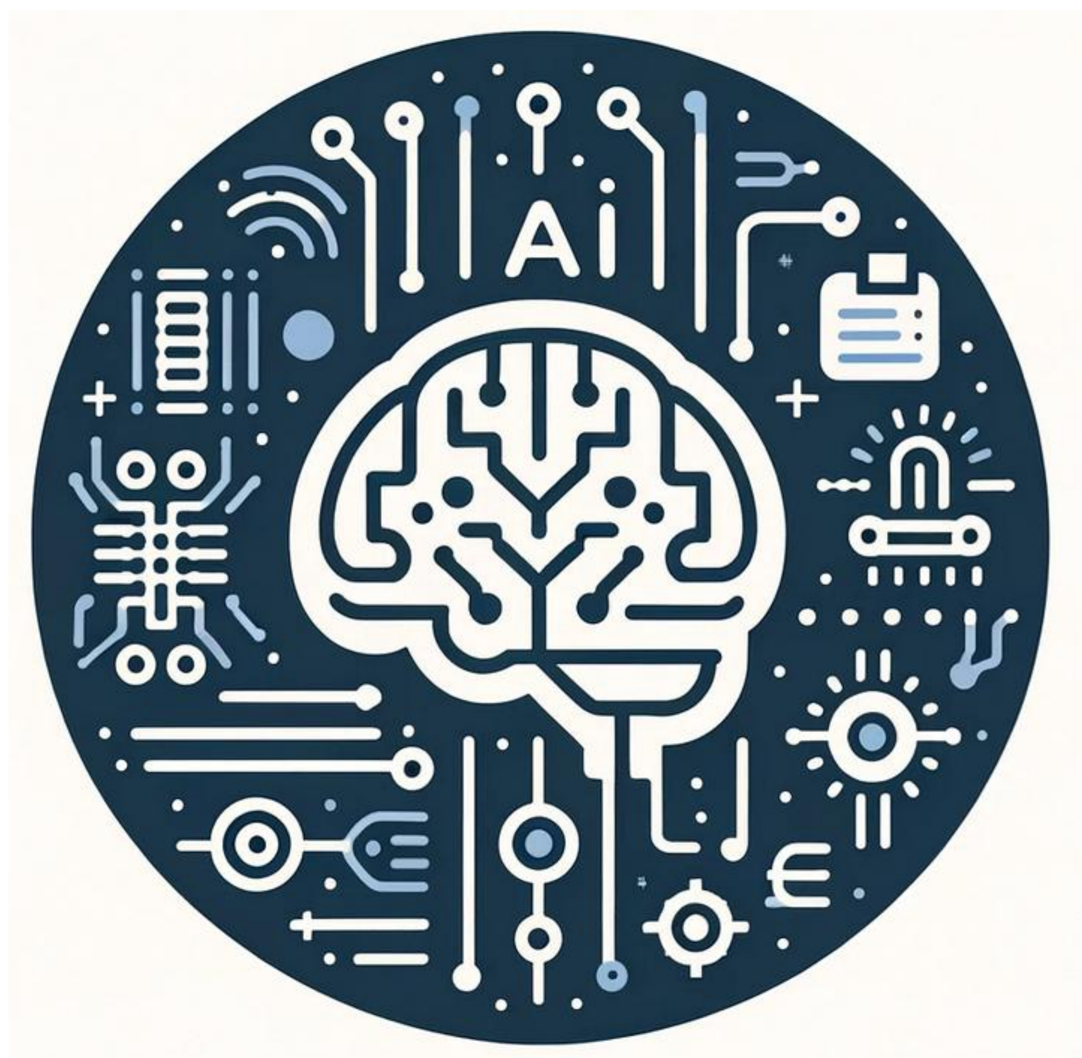
ИТМО

И.А. Бессмертный, А.Е. Авдюшина, А.В. Кугаевских,
Ю.А. Королева, Н.Г. Рущенко

Системы искусственного интеллекта



Санкт-Петербург
2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

**И.А. Бессмертный, А.Е. Авдюшина, А.В. Кугаевских,
Ю.А. Королева, Н.Г. Рущенко**

Системы искусственного интеллекта

МЕТОДИЧЕСКОЕ ПОСОБИЕ
РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлению подготовки 09.03.01, 09.03.04, 45.03.04.
в качестве Учебно-методического пособия для реализации основных
профессиональных образовательных программ высшего образования
бакалавриата

ИТМО

Санкт-Петербург
2024

И.А. Бессмертный, А.Е. Авдюшина, А.В. Кугаевских, Ю.А. Королева, Н.Г. Рущенко., Системы искусственного интеллекта – СПб: Университет ИТМО, 2024. – 81 с.

Рецензент(ы):
Поляков Владимир Иванович, кандидат технических наук, доцент, доцент (квалификационная категория "ординарный доцент") факультета программной инженерии и компьютерной техники, Университета ИТМО.

Методическое пособие содержит материалы для выполнения и защиты лабораторных работ по дисциплине «Системы искусственного интеллекта». Учебное пособие включает в себя логические методы искусственного интеллекта и методы, основанные на машинном обучении, уже ставшие классикой, что позволит студенту в дальнейшем самостоятельно выбирать оптимальные пути для решения поставленных прикладных задач. Практические задания позволят углубить навыки практического программирования и алгоритмического мышления студентов.

ИТМО (Санкт-Петербург) — национальный исследовательский университет, научно-образовательная корпорация. Альма-матер победителей международных соревнований по программированию. Приоритетные направления: IT и искусственный интеллект, фотоника, робототехника, квантовые коммуникации, трансляционная медицина, Life Sciences, Art&Science, Science Communication.
Лидер федеральной программы «Приоритет-2030», в рамках которой реализовывается программа «Университет открытого кода». С 2022 ИТМО работает в рамках новой модели развития — научно-образовательной корпорации. В ее основе академическая свобода, поддержка начинаний студентов и сотрудников, распределенная система управления, приверженность открытому коду, бизнес-подходы к организации работы. Образование в университете основано на выборе индивидуальной траектории для каждого студента.
ИТМО пять лет подряд — в сотне лучших в области Automation & Control (кибернетика) Шанхайского рейтинга. По версии SuperJob занимает первое место в Петербурге и второе в России по уровню зарплат выпускников в сфере IT. Университет в топе международных рейтингов среди российских вузов. Входит в топ-5 российских университетов по качеству приема на бюджетные места. Рекордсмен по поступлению олимпиадников в Петербурге. С 2019 года ИТМО самостоятельно присуждает ученые степени кандидата и доктора наук..

© Университет ИТМО, 2024
© И.А. Бессмертный, А.Е. Авдюшина, А.В. Кугаевских,
Ю.А. Королева, Н.Г. Рущенко, 2024

Содержание

БАЗЫ ЗНАНИЙ И ОНТОЛОГИИ 6
Основы программирования на языке Prolog 6
Prolog как декларативный язык 6
Понятие предиката 7
Как работает интерпретатор Prolog? 10
Факты и правила в Prolog 11
Рекурсии в языке Prolog 14
Рекурсии и итерации 17
Отсечения в Prolog 18
Красное и зеленое отсечения 21
Списки в Prolog 22
Пример: Решение логической задачи о волке, козе и капусте 23
Семантические сети 26
Историческая справка 26
Типы семантических сетей 28
Типы отношений в семантических сетях 30
Онтологии и правила наследования отношений 33
Проблемы построения семантических сетей 34
Факты и правила в семантической сети 35
Интеллектуальный агент семантической сети 38
Управление контекстом 38
Семантическая сеть и Семантическая паутина 39
Семантическая Паутина: принципы и текущее состояние 40
Контрольные вопросы 42
МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ 43
Линейная регрессия 43
Метод k-ближайших соседей 45

Деревья решений 47
Логистическая регрессия 50
Лабораторные работы 56
Модуль 1. БАЗЫ ЗНАНИЙ И ОНТОЛОГИИ 56
Лабораторная работа 1. Создание базы знаний и выполнение запросов в Prolog 56
Лабораторная работа 2. Создание онтологии в Protege 57
Лабораторная работа 3. Разработка системы поддержки принятия решения на основе базы знаний или онтологии 57
Модуль 2. МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ 59
Лабораторная работа 4. Линейная регрессия 59
Лабораторная работа 5. Метод k-ближайших соседей 60
Лабораторная работа 6. Деревья решений 60
Лабораторная работа 7. Логистическая регрессия 61
ПРИЛОЖЕНИЯ 62
Приложение 1. Пример реализации онтологии в Protege 62
Приложение 2. Структура отчёта по первому блоку лабораторных работ 73
Приложение 3. Структура отчёта по второму блоку лабораторных работ 74
СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ 76

Введение

В методическом пособии рассматривается цикл лабораторных работ по дисциплине «Системы искусственного интеллекта», которые необходимо выполнить для успешного освоения курса.

Задачей изучения дисциплины «Системы искусственного интеллекта» является формирование у студента необходимых компетенций, позволяющих выбирать методы искусственного интеллекта для решения практических задач. Студент познакомится:

- с базами знаний, онтологиями и классическими методами машинного обучения;
- с основными математическими методами, применяемыми для решения как типовых задач, так и сложных научно-технических задач;
- с причинами возникновения погрешностей и их учете при оценке результата вычислений;

В результате практических занятий студент должен уметь:

- выбрать метод, которым необходимо воспользоваться при решении конкретной задачи;
- написать программное решение, реализующее данный метод;
- адекватно оценить полученные результаты.

Курс состоит из двух блоков:

- базы знаний и онтологии;
- классические методы МО

Каждый блок включает в себя несколько лабораторных работ. Для защиты блока лабораторных работ необходимо сформировать отчёт о проведённых для лабораторных работ исследованиях и описать практическую часть, включающую разработку.

БАЗЫ ЗНАНИЙ И ОНТОЛОГИИ

Основы программирования на языке Prolog

Теория искусственного интеллекта включает в себя два подхода: ИИ на основе логики и ИИ на основе данных.

Естественный интеллект обычно ассоциируют со способностью мыслить логически, искусственный интеллект также изначально развивался как система моделирования рассуждений. Теоретическую основу для этого составила математическая логика, основы которой были заложены еще в XIX веке итальянским математиком Джузеппе Пеано. Таким образом, математическая логика явилась основой классического подхода к искусственному интеллекту. Практически идея построения интеллектуальных систем на основе логики была реализована в начале 70-х гг. XX века в виде языка программирования Prolog. Особенность этого языка состоит в том, что в нем не реализуются алгоритмы, а лишь задаются начальные условия, правила и цель, а поиск решения выполняется самостоятельно. В этой связи в 80-х годах во всем мире на Prolog возлагались большие надежды в части создания систем искусственного интеллекта. В частности, язык Prolog должен был стать основой ЭВМ пятого поколения. Несмотря на то, что эти ожидания оказались завышенными, язык является изящным средством решения многих логических задач и используется во многих сферах, где требуется перебирать множество вариантов: составление расписаний, транспортно-логистическая деятельность, оперативное планирование и ситуационное управление, раскрой материалов и др.

Изучение основ программирования на языке Prolog позволит понять принципы построения и границы применимости систем искусственного интеллекта на его основе, чтобы в дальнейшем принимать обоснованный выбор между ИИ на основе логики и ИИ на основе данных.

Prolog как декларативный язык

Разработка языка Prolog началась в 1970 г. Аланом Кулмероэ и Филиппом Русселом. Они хотели создать язык, который мог бы делать логические заключения на основе заданного текста. Название Prolog является сокращением от "PROgramming in LOGic". Этот язык был разработан в Марселе в 1972 г.

Prolog (Пролог) – язык программирования, который основан не на алгоритме, а на логике предикатов. Если программа на алгоритмическом (процедурном) языке является последовательностью инструкций, выполняющихся в заданном порядке, то программа на Прологе содержит только описание задачи, а Пролог-машина выполняет поиск решения, руководствуясь этим описанием. Например, существует логическая задача покрытия шахматной доски ходом коня. На любом алгоритмическом языке

решение этой задачи требует построения достаточно сложного алгоритма. На Прологе достаточно описать правила, по которым ходит конь, после чего Пролог самостоятельно отыщет решение. Обратной стороной такой простоты является ресурсоемкость программ. Например, в другой популярной задаче размещения на шахматной доске восьми ферзей, которые не бьют друг друга, полное дерево решений имеет 64^8 вершин. Очевидно, что нахождение решения в таком дереве займет неприемлемо много времени.

Современный подход к решению таких задач состоит в использовании технологий машинного обучения, которое позволяет гораздо быстрее находить решение на большом пространстве поиска. Пролог, который работает методом полного перебора, не в состоянии решить ни одну задачу реального мира без использования приемов, сокращающих размерность поиска. Изучение Пролога позволяет понять логику поиска, т.е. «перебросить мостик» от наивного поиска к методам редуцирования пространства поиска решений, в том числе с использованием методов машинного обучения.

Программирование на языке Пролог состоит из следующих этапов:

- объявления некоторых фактов об объектах и отношениях между ними,
- определения некоторых правил об объектах и отношениях между ними,
- формулировки вопросов об объектах и отношениях между ними.

Понятие предиката

В математической логике выделяют дескрипционную логику и логику первого порядка. В дескрипционной логике используются атомарные высказывания, неделимые в том смысле, что они могут быть истинными или ложными, но почему они истинные или ложные, нам неизвестно. В дескрипционной логике можно записывать факты, но построить на них сколь-нибудь полезные рассуждения невозможно ввиду их очевидности. Логика первого порядка оперирует с более сложными логическими единицами – предикатами.

Основным элементом программы на Прологе является предикат. С математической точки зрения предикат – это функция, которая возвращает бинарное значение (истина или ложь). В Прологе предикатом обозначается отношение между объектами, которое также может быть истинным.

Рассмотрим понятие предиката в Прологе на примере звездной семьи Пугачевой – Киркорова, правда, теперь уже бывшей. Вначале запишем отношения типа родитель – ребенок. В синтаксисе Пролога выражение «Борис является родителем Аллы» выглядит следующим образом:
parent(boris, alla).

Здесь *parent* – это имя предиката, а *boris* и *alla* – аргументы. Аргументы *boris* и *alla* являются константами, поэтому записаны строчными буквами. С прописной буквы в Прологе начинаются переменные. Точка означает конец предиката, так же, как и конец предложения на естественном языке. Запишем

также родительские отношения для других членов семьи:

parent(bedros, filipp).
parent(edmuntas, kristina).
parent(alla, kristina).
parent(kristina, nikita).
parent(vladimir, nikita).

Теперь дадим понятие «супруг» (spouse):

spouse(filipp, alla).
spouse(vladimir, kristina).

Полученный набор предикатов образует базу знаний о звездной семье. Сравним с тем, как те же данные будут представлены в реляционной базе.

Таблица Родители	
Parent	**Child**
Boris	*Alla*
Bedros	*Filipp*
Alla	*Kristina*
Edmuntas	*Kristina*
Kristina	*Nikita*

Таблица Супруги	
S1	**S2**
Alla	*Filipp*
Kristina	*Vladimir*

Как видно, на уровне представления данных сходство налицо. Но на этом оно и заканчивается, и на уровне извлечения данных имеет место большое различие. Реляционной базе данных для того, чтобы извлечь знания, требуется создать запрос на выборку данных, например, на языке SQL.

Пусть мы хотим узнать, кто родители Кристины. Мы должны написать запрос следующего вида:

SELECT *Parent* FROM *Родители* WHERE *Child = "Kristina"*

В Прологе запрос на извлечение знаний описывается такими же предикатами, какими эти знания представляются. Если мы подставим Прологу такой предикат (предикат цели):

parent(alla, kristina).

Тогда цель может быть прочитана следующим образом: Является ли Алла родителем Кристины? Сопоставив эту цель с содержимым базы знаний, Пролог установит, что данное утверждение истинно, и сообщит об этом.

Вышеприведенный SQL запрос (Кто является родителем Кристины?) в Прологе выглядит следующим образом:

parent(X, kristina).

Здесь *X* является переменной, которой должны быть присвоены искомые значения. Напомним, что все переменные в Прологе начинаются с прописной буквы. Переменная в Прологе является аналогом местоимения или вопросительного слова. Из приведенной выше базы знаний Пролог

извлечет два ответа:

X = alla

X = edmuntas

Мы можем сформулировать вопрос следующим образом: Есть ли у Кристины родители?

parent(_, kristina).

Пролог выдаст ответ: *Yes*.

Переменная, начинающаяся со знака подчеркивания, называется анонимной переменной и может принимать любые значения (аналог местоимения некто).

Для более сложных запросов в базах данных необходимо создавать представления (view) или создавать вложенные запросы в SQL. В Прологе все гораздо проще. Давайте найдем, чьей внучкой является Кристина. Запрос будет выглядеть следующим образом:

parent(X, kristina), parent(Y, X).

Данная запись означает: Найти *Y*, являющийся родителем *X*, который, в свою очередь, является родителем Кристины. Запятая в Прологе идентична союзу "И" или конъюнкции. В ответ на такой запрос Пролог выдаст следующий ответ:

X = alla

Y = boris

Можно выдать следующий запрос:

parent(alla, _).

Теперь анонимная переменная использована в качестве второго аргумента. Этот запрос можно прочитать следующим образом: Есть ли у Аллы дети? Пролог выдаст ответ: *Yes*.

Поскольку предикат – это бинарная функция, которая может возвращать истину либо ложь (высказывание, которое может также быть истинным или ложным), результат работы программы на Прологе – это определение, истинна или ложна цель. Присвоение значение переменным, вывод результатов и т.п. – это всего лишь побочные результаты.

Здесь следует внести небольшое уточнение. Действительно, мы спрашиваем у Пролога, есть ли у Аллы дети? Но Пролог выдает ответ на несколько другой вопрос: Есть ли в базе знаний сведения о наличии детей у Аллы? Если в рамках данной микро-базы знаний мы спросим о матери Аллы, то окажется, что она сирота, хотя это может быть ложью. В этом состоит главный недостаток бинарной логики: она не делает различия между отсутствием решения и его отрицанием. Это обстоятельство носит название отрицание как неудача (negation as failure) и может негативно влиять на качество моделирования рассуждения.

Таким образом, программа на Прологе состоит из предикатов. Программа на Прологе и база знаний - синонимы. Цель формулируется также в виде предикатов. Выполнение программы на Прологе – это резолюция цели.

Как работает интерпретатор Prolog?

Процесс нахождения решения в Прологе заключается в сопоставлении предиката цели с предикатами базы знаний. Этот процесс называется ***унификацией.*** Пусть Пролог-системе предъявлена цель из предыдущего подраздела (мы хотим найти, чьей внучкой является Кристина):

parent(X, kristina), parent(Y, X).

Пролог выделяет из нее первую подцель *parent(X, kristina)* и начинает сопоставлять ее с базой знаний (проводить унификацию). База знаний повторена ниже.

parent(boris, alla).
parent(bedros, filipp). parent(edmuntas, kristina).
parent(alla, kristina). parent(kristina, nikita).

Вначале сопоставляются первый предикат и подцель:

parent(boris, alla) и *parent(X, kristina)*

Первый аргумент *boris* сопоставляется с переменной *X*.

Следует иметь в виду, что в Прологе различаются состояния переменных *free* (свободные) и *bound* (связанные). **Если обе переменные связаны, то при унификации происходит их сравнение. Если одна из них свободна, то происходит присвоение. Переприсвоение значений переменным не допускается.** Это существенно отличает Пролог от прочих языков.

Таким образом, переменной *X*, которая пока является свободной, присваивается значение *boris*. После этого унифицируются вторые аргументы, *alla* и *kristina*. Поскольку это константы, и *alla* не равно *kristina*, то унификация предиката *parent(boris, alla)* и подцели *parent(X, kristina)* заканчивается неудачей (*fail*).

Поскольку в базе знаний несколько экземпляров предиката parent, такой предикат называется неоднозначным (*non-deterministic*). Если предикат один, то он называется однозначным (*deterministic*).

В случае неоднозначного предиката после неудачи выполняется откат – переход к следующему экземпляру предиката. При этом отменяется также присвоение значение переменным, если таковое имело место. Затем выполняется унификация предикатов:

parent(bedros, filipp) и *parent(X, kristina)*

Очевидно, что результат унификации будет тот же, неудача (*fail*). При откате на следующий предикат *parent(edmuntas, kristina)* картина будет иная: *X* присвоится значение *edmuntas*, а сопоставление вторых аргументов будет также успешным, так как *kristina = kristina*. Таким образом, первая подцель окажется выполненной. Пролог запоминает, какой экземпляр предиката сработал, и устанавливает на следующий предикат указатель отката:

parent(boris, alla).
parent(bedros, filipp).
parent(edmuntas, kristina).

> *parent(alla, kristina).*
parent(kristina, nikita).

После чего перейдет ко второй подцели
parent(Y, X).

X = edmuntas, т.е.Пролог ставит себе такую подцель:
parent(Y, edmuntas).

В поисках родителя Эдмунтаса Пролог снова начинает унифицировать этот предикат с начала базы знаний, начиная с *parent(boris, alla)*. Нетрудно видеть, что на этот раз перебор всех предикатов закончится неудачей, т.е. подцель *parent(Y, edmuntas)* не дала положительного решения. В этом случае Пролог откатывается к предыдущей подцели (продвигается назад по списку подцелей) и пытается найти альтернативное решение для *parent(X, kristina).* При этом *X* опять становится свободной переменной, а Пролог возвращается к точке отката:

parent(boris, alla).
parent(bedros, filipp).
parent(edmuntas, kristina).
> *parent(alla, kristina).*
parent(kristina, nikita).

то есть к предикату *parent(alla, kristina),* сопоставляя его с подцелью *parent(X,kristina).*

Теперь *X* присваивается значение *alla*, указатель отката устанавливается на предикат *parent(kristina, nikita)* и опять выполняется переход к следующей подцели *parent(Y, X)*, где *X = alla*. Пролог снова начинает унификацию подцели *parent(Y, alla)* с базой знаний, начиная с первого предиката. В первом предикате происходит унификация константы *boris* и свободной переменной *Y*. Происходит присвоение *Y=boris,* затем сопоставляются вторые аргументы. Так как *alla* = *alla*, сопоставление завершается успешно.

Таким образом, решение найдено: Кристина является внучкой Бориса. Заметим, что неудача, которая постигла нас в поисках деда Кристины по отцовской линии, связана только с неполнотой базы знаний.

Итак, интерпретатор Пролога автоматически выполняет поиск решения. Механизм поиска реализован с помощью отката после неудачи. Откат происходит на следующий экземпляр неоднозначного предиката. Выполнение программы на Прологе (резолюция цели) заключается в унификации цели с базой знаний.

Факты и правила в Prolog

Описанный выше запрос, устанавливающий отношение типа "прародитель-внук", может потребоваться в дальнейшем неоднократно. В этой связи его целесообразно запомнить для дальнейшего использования в других запросах. В базе знаний Пролога можно хранить не только факты, но и правила, т.е. условные отношения. Отношение типа "прародитель-внук"

может быть записано следующим образом:

grandparent(X,Y) if parent(X,Z), parent(Z,Y).

Читать это нужно следующим образом: *X* является прародителем *Y*, если *X* является родителем *Z* и *Z* является родителем *Y*. Предикат *grandparent(X,Y)* называется заголовком правила, а выражение справа от *if* – телом правила.

Примечание: Синонимом связки "if" в правиле являются символы ":-" – стилизованное отображение символа логического следования в математической логике.

Таким образом, как и в базах данных, в базе знаний Пролога в виде фактов мы храним первичные знания, а производные от них записываем в виде правил, к которым обращаемся так же, как и к фактам.

Факт – это то, что известно.

Правило – это способ порождения новых фактов на основе имеющихся.

Для родственных отношений мы можем установить множество правил, избавляясь от необходимости вводить дополнительные факты, например, кто кому приходится братом, племянником и т.д. Правило, определяющее отношение брат (сестра):

sibling(X,Y) :- parent(Z,X), parent(Z,Y), X<>Y.

Предикат сравнения *X<>Y* (*X\=Y*) нужен для разрешения коллизии типа "сын моего отца, но мне не брат". Правило, определяющее отношение типа дядя, выглядит следующим образом:

uncle(X,Y) :- parent(Z,Y), sibling(X,Z).

Когда в ходе резолюции цели Пролог встречает не факт, а правило, то вначале унифицирует заголовок правила, т.е. сравнивает связанные переменные и присваивает значения свободным переменным. В случае успешной унификации аргументов Пролог подставляет значения аргументов из заголовка в первый предикат в теле правила и ставит этот предикат себе в качестве подцели, которую начинает унифицировать с базой знаний. В случае успешной резолюции данной подцели Пролог переходит к следующему условию правила. Если унификация этого предиката условия приводит к неудаче, то Пролог выполняет откат к предыдущему условию правила. Этот откат происходит только в том случае, если этот предыдущий предикат является неоднозначным. Поясним это на примере. Зададимся целью найти, кто является прародителем Кристины:

grandparent(Who, kristina).

Получив такую цель, Пролог начинает унифицировать ее с правилом: *grandparent(X,Y)* **:-** *parent(X,Z), parent(Z,Y).* Переменная *Who* в предикате цели является свободной переменной, и ее унификация с переменной *X* в заголовке правила будет успешной всегда. Следует заметить, что в Прологе все переменные являются локальными, т.е. существует только внутри правила. Мы могли бы использовать *X* вместо *Who*, и это были бы разные переменные, которые бы унифицировались точно так же. При необходимости создания глобальных переменных используют динамические

факты, которые создаются предикатом *assert* и уничтожаются предикатом *retract* или *retractall.*

Далее унифицируются константа *kristina* с переменной *Y*. Поскольку переменные в заголовке правила всегда сначала являются свободными, выполняется присвоение: *Y = kristina.* Поскольку унификация заголовка правила прошла успешно, Пролог углубляется в тело правила и ставит себе в качестве подцели первый предикат тела правила, подставляя переменные, если они связанные:

parent(X, Z).

Переменные *X* и *Z* являются свободными, поэтому успешной будет унификация данной подцели с первым же предикатом *parent* из базы знаний:

X = boris, Z = alla

После этого Пролог переходит ко второму предикату в правиле, подставляя значение *X* и *Y*:

parent(alla, kristina).

Резолюция данной подцели дает истину, а значения переменных, присвоенные в ходе унификации, возвращаются Прологом:

Who = X = boris.

Таким образом, в ходе резолюции основной цели Пролог самостоятельно ставит себе подцели, руководствуясь правилами, находящимися в базе знаний. Рассмотрим другой пример:

grandparent(Who, nikita).

Аналогично предыдущему примеру, Пролог унифицирует заголовок правила *grandparent(X,Y)* и присваивает значение *Y = nikita.* Углубляясь в тело правила, Пролог формирует подцель *parent(X, Z).* Данная подцель возвращает, как и в предыдущем примере,

X = boris, Z = alla

Пролог переходит ко второму предикату правила, подставляя в *parent(Z,Y)* значения переменных:

parent(alla, nikita).

Пытаясь унифицировать данную подцель, Пролог сопоставляет переменные (*alla, nikita*) с первым экземпляром предиката *parent*, терпит неудачу, откатывается к следующему экземпляру, и так далее. Поскольку факта *parent(alla, nikita)* в базе знаний нет, резолюция данной подцели оказывается неудачной. Следовательно, унификация первого предиката правила значениями *X = boris, Z = alla* является неверной. Поэтому **Пролог выполняет откат к предыдущему условию правила и пытается найти другое решение для подцели** *parent(X, Z)*. При этом отменяется присвоение переменных (*X = boris*, *Z = alla*). Переменные *X* и *Z* вновь становятся свободными. Заметим, что откат здесь возможен только на неоднозначный предикат. Если в цепочке предикатов внутри правила встречаются как однозначные, так и неоднозначные предикаты, то откат после неудачи выполняется на ближайший неоднозначный предикат.

При первой унификации данного предиката Пролог установил

указатель отката на следующий экземпляр факта *parent*:

parent(boris, alla).
> *parent(bedros, filipp).*
parent(edmuntas, kristina).
parent(alla, kristina). parent(kristina, nikita).

При откате Пролог приступает к унификации данного факта и устанавливает указатель отката на третий экземпляр:

parent(boris, alla).
parent(bedros, filipp).
> *parent(edmuntas, kristina).*
parent(alla,kristina).
parent(kristina, nikita).

После унификации второго предиката parent с подцелью *parent(X,Z)* Пролог присвоит значения переменным:

X = bedros, Z = filipp

и снова переходит ко второму предикату *parent(Z,Y)*:

parent(bedros, nikita).

Очевидно, резолюция и этой подцели завершается неудачей. Пролог снова откатывается к предыдущему предикату правила и к третьему экземпляру факта *parent*. Успешой окажется только унификация четвертого факта: *parent(alla, kristina),* в результате чего мы получим

Who = X = alla.

Так работает интерпретатор Пролога в случае наличия правил в базе знаний.

Таким образом,

Факт – знания, основанные на константах (неизменяемые знания).
Правила – знания, которые выводятся на основании фактов.
Набор фактов и правил не содержит в себе алгоритма.
Правила и факты существуют независимо друг от друга.
Объединение правил для вывода результата происходит в ходе резолюции цели.
Переменные в заголовке правила существуют только внутри данного правила.
При откате внутри правила происходит переход к предыдущему неоднозначному предикату в правиле.

Рекурсии в языке Prolog

Если нас интересует, является ли X предком *Y*, то мы должны ставить цели последовательно:

parent(X,Y).
grandparent(X,Y).
grandgrandparent(X,Y).
grandgrandgrandparent(X,Y).

и так далее, где

grandgrandparent(X,Y) :- parent(X,Z), grandparent(Z,Y).
grandgrandgrandparent(X,Y) :- parent(X,Z), grandgrandparent(Z,Y).

Вместо этого Пролог позволяет записать данное правило следующим образом:

predecessor(X,Y) :- parent(X,Y).
predecessor(X,Y) :- parent(X,Z), predecessor(Z,Y).

Здесь имеет место рекурсивный вызов предиката *predecessor*. Рекурсия в Прологе является мощным средством, позволяющим строить очень компактные и эффективные программы.

Рассмотрим использование рекурсии на примере вычисления факториала. $n! = n\cdot(n-1)\cdot(n-2)\cdot\ldots\cdot1$

Рекурсивное определение факториала: $0! = 1$; $n! = n\cdot(n-1)!$ Программа на Прологе, реализующая вычисление факториала, будет выглядеть следующим образом:

f(0,1).
f(N,F) :- N1=N-1, f(N1,F1), F=F1·N.

Обратите внимание, что программа, в сущности, состоит не более, чем из рекурсивного математического описания функции факториала, приведенного выше. Запустим программу в режиме трассировки и зададим цель *f(3, X)*. Пролог начинает сопоставлять предикат цели с базой знаний (CALL означает вызов предиката, RETURN – завершение работы предиката, FAIL – неудачу, REDO – откат):

CALL	*f (3, X)*	цель сопоставляется с *f(0,1)*
FAIL		неудача, т.к. $3 \neq 0$
REDO	*f (3, X)*	Происходит откат на следующий экземпляр *f()*
	N= 3,	Входим в тело правила. Присваиваем *N=3*
	N1= 2	Находим *N1=2*
	f (2, X)	Пролог ставит себе подцель которая
CALL	*f (2, X)*	сопоставляется с *f(0,1)*
FAIL		неудача, т.к. $2 \neq 0$
REDO	*f (2, X),*	Происходит откат на следующий экземпляр *f()*
	N= 2,	Опять входим в тело правила. Присваиваем *N=2*
	N1=1 f (1, X)	Находим *N1=1*. Это уже другое *N1* Пролог ставит себе подцель *f (1, X)* , которая
CALL	*f (1, X)*	сопоставляется с *f(0,1)*
FAIL		*неудача,* т.к. $1 \neq 0$
REDO	*f (1, X)*	Происходит откат на следующий экземпляр *f()*
	N=1	Опять входим в тело правила. Присваиваем *N=1*
	N1= 0	Находим *N1=0.*
	f(0, X)	Пролог ставит себе подцель *f(0, X)*
CALL	*f(0, X)*	которая сопоставляется с *f(0,1)*

RETURN	*X=1*	Успешно. Возврат из нижнего уровня рекурсии
	F=1	Умножение *1* на *1* (факториал от *1*)
RETURN	*X=1*	Возврат из следующего уровня рекурсии
	F=2	Умножение *2* на *1* (факториал от *2*)
RETURN	*X=2*	Возврат из следующего уровня рекурсии
	F=6	Умножение *3* на *2* (факториал от *3*)
RETURN	*X=6*	Возврат из программы

Можно заметить, что логика работы программы вычисления факториала зависит от расположения в тексте предиката, определяющего выход из рекурсии. Унификация выполняется в порядке следования предикатов в тексте программы, и если предикат *f(0,1)* поставить в конце, выход из рекурсии будет невозможен. Таким образом, декларативность Пролога не является абсолютной для удобства его использования. Вариант программы, в котором предикаты могут располагаться в любом порядке, представлен ниже.

f(N,F) :- N>0, N1=N-1, f(N1,F1), F=F1·N.f(0,1).

Заметим также, что в данном рекурсивном предикате есть действия в теле правила после рекурсивного вызова. Это называют нарушением хвостовой рекурсии (tail recursion).

Нарушение хвостовой рекурсии, называемое также нехвостовой рекурсией, требует запоминания всего окружения рекурсивного вызова (а не только результата), поэтому приводит к большим затратам памяти. Существуют приемы устранения нехвостовых рекурсий. Ниже приведен пример вычисления факториала без нехвостовой рекурсии.

```
f(N1,N,F1,F) :-      % если N1! = F1, то N! = F
      N2=N1+1,
      F2=F1*N2,      %  (N1+1)! = F2
      f(N2,N,F2,F).  % если N2! = F2, то N! = F
      f(N,N,F,F).    % Условие выхода из рекурсии
```

Цель для вычисления 3! выглядит следующим образом: *f(0,3,1,F).*

Рекурсия в Прологе не всегда используется для выполнения многократно повторяющихся действий. Вспомним базу знаний «звездной» семьи, в которой есть предикат, описывающий супружеские отношения, в частности, *spouse(filipp,alla).*

Супружеские отношения, в отличие от родительских отношений, являются симметричными. Филипп является супругом Аллы, равно как и Алла является супругой Филиппа. При этом в предикате положение аргументов является фиксированным. Иными словами, если факт в базе знаний записан следующим образом:

spouse(filipp,alla).

а предикат цели таким:

spouse(alla,filipp).

то результат будет отрицательным. Для того, чтобы показать Прологу, что это предикат является симметричным в отношении аргументов, мы можем

применить правило:

spouse(X,Y) :- spouse(Y,X).

В качестве примера рассмотрим известную коллизию. В деревне жили две семьи: мать с дочерью и отец с сыном. Дадим им имена. Пусть мать и дочь зовут Мария и Даша, а отца и сына Олег и Сергей соответственно. Первые буквы имен подскажут нам, кто есть кто, иначе мы запутаемся. В базе знаний эти факты найдут свое отражение в следующем виде:

parent(oleg, sergei). parent(maria, dasha).

В силу превратностей судьбы Олег женился на Даше, и Мария вышла замуж за Сергея:

spouse(oleg,dasha). spouse(sergei, maria).

Для симметрии супружеских отношений введем правило:

spouse(X, Y) :- spouse(Y, X).

Нравы в деревне простые, поэтому жену отца положено называть мамой, а мужа матери – отцом. Правила для этого будут такими:

parent(X,Y) :- spouse(X,Z), parent(Z,Y).

Попробуем найти внука Сергею. Ставим цель

grandparent(sergei, Who).

Пролог находит правило *grandparent(X,Y) :- parent(X,Z), parent(Z,Y)* и ставит себе подцель из первого предиката в теле этого правила:

parent(sergei, Z).

У Сергея детей нет, поэтому Пролог обращается к правилу *parent(X,Y) :- spouse(X, Z), parent(Z,Y)* и ставит себе цель *spouse(sergei, Z).* Пролог дает результат *Z = maria*. Второй предикат правила

parentparent(maria,Y).

Возвращается значение *Y = dasha*. То есть Сергей в качестве мужа Марии числится отцом Даши. Теперь Пролог переходит ко второму предикату в правиле *grandfather*:

parent(dasha,Y).

Даша также не имеет детей, поэтому Пролог обращается к правилу *parent(X,Y) :- spouse(X,Z), parent(Z,Y)* и сначала пытается найти супруга Даше:

spouse(dasha, Z).

Результат: *Z = oleg*. Второй предикат этого правила *parent(oleg,Y)* даст *Y = sergei.* То есть Сергей приходится внуком самому себе! Созданная нами база знаний верно отражает данную коллизию.

Рекурсии и итерации

Напишем на Прологе простую программу, которая имитирует на компьютере пишущую машинку:

type :- readchar(X), write (X), type.

Стандартный предикат *readchar* выполняет чтение символа с клавиатуры. Это бесконечная рекурсия – из нее нет выхода. Модифицируем программу

таким образом, чтобы она обеспечивала ввод только одного предложения (до первой точки):

type :- readchar(X), write(X), X <> '.', type.type.

Первый предикат *type* читает символ, выводит его на экран, после чего сравнивает его с точкой. Если введенный символ не точка, то происходит рекурсивный вызов *type*, в противном случае – откат на следующий, пустой предикат *type*. Второй предикат *type* нужен лишь для того, чтобы вся программа завершилась успехом, а не неудачей.

Заметим, что рекурсия в этой программе совершенно не нужна, так как вызов предиката из самого себя не имеет никакого смысла для логики программы. По программистской привычке нам было бы предпочтительней зациклить данное правило. Но в Прологе GOTO нет в принципе. Вместо этого есть возможность вызывать откат. Напишем следующую конструкцию:

type :- readchar(X), write(X), X = '.'.

При первом выполнении данного правила (введенный символ не равен точке) предикат завершится неудачей. Отката же не будет по той причине, что все предикаты в теле правила являются однозначными. Кстати, ***все стандартные предикаты являются однозначными.***

Таким образом, для того, чтобы откатиться на начало правила, необходимо, чтобы там был неоднозначный предикат. Создадим такую конструкцию:

repeat.

repeat :- repeat

Предикат *repeat* ничего не делает кроме того, что является неоднозначным. Можно даже сказать, очень неоднозначным, поскольку перебрать все его варианты невозможно. Это бесконечная рекурсия.

Программа, имитирующая пишущую машинку, без рекурсивного вызова *type* будет выглядеть следующим образом:

repeat.

repeat :- repeat

type :- repeat, readchar(X), write(X), X = '.'.

Здесь рекурсия заменена итерацией, хотя рекурсивный вызов в виде предиката *repeat* все же присутствует.

Отсечения в Prolog

Из всего вышесказанного можно сделать справедливый вывод о том, что интерпретатор Пролога, выполняя резолюцию цели, всегда делает полный обход дерева решений. Спуск по дереву вниз соответствует углублению в тело правила, возврат наверх и переход на соседнюю ветвь – откату после неудачи. Рассмотрим концепцию отсечения на конкретном примере.

Пусть нас пригласили на дачу. При этом обычно дают подробную инструкцию, как проехать и найти конкретный дом. Например, полученная нами инструкция выглядит следующим образом:

1. Въехать в деревню Васино.
2. Повернуть направо.
3. Доехать до колодца.
4. Искомый дом из красного кирпича – первый от колодца по правой стороне.

Запишем правило поиска дачи предикатами Пролога:

```
dacha1(X) :-
        enter_village(X),f
        ind_a_house.
find_a_house :
        turn_right,
        meet_mine,
        see_a_red_brick_house.
enter_village(vino).
enter_village(vanino).
```

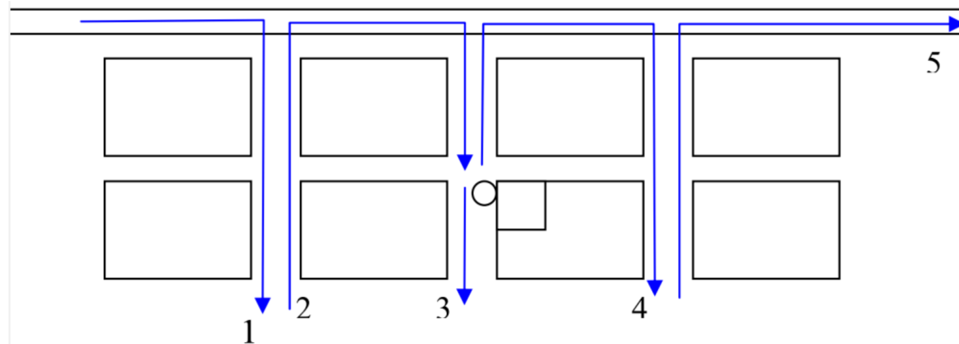


Рис.1 Иллюстрация отсечения в Прологе

Начинаем поиск (рисунок 1), въезжаем в деревню и поворачиваем направо (стрелка 1). Проезжаем до конца деревни и не находим колодца. Выполняем откат, то есть возвращаемся на шоссе и едем до следующего поворота направо (стрелка 2), доезжаем до колодца и видим, что ближайший к нему дом из белого кирпича. Едем до конца улицы и видим, что колодцев больше нет (стрелка 3). Откатываемся на шоссе и едем до следующего поворота (стрелка 4), проезжаем всю улицу и откатываемся, поскольку колодцев здесь нет (стрелка 5). Из этого делаем вывод, что название деревни мы прочитали неверно, и дачу следует искать в следующей деревне. В деревне Ванино повторяем все сначала.

Отсечение позволяет сократить количество пробегаемых ветвей дерева решений. Если хозяин дачи даст нам важную дополнительную информацию о том, что в каждой деревне есть только один колодец (а если в качестве ориентира заданы почта или милиция, то можем догадаться сами, что они могут быть в деревне только в одном экземпляре), мы не будем выполнять действия, обозначенные стрелками 3 и 4, а сразу поймем, что попали не в ту деревню.

Отсечение – это предикат, который обозначается восклицательным знаком и вставляется в правило. Срабатывает отсечение после того, как Пролог пройдет через него. Отсечение уничтожает указатели отката, установленные в ходе унификации данного правила, т. е. делает предыдущие предикаты данного правила однозначными.

Если мы хотим указать в программе поиска дачи, что колодец в деревне единственный, мы должны поставить отсечение после найденного колодца:

```
dacha2(X) :- enter_village(X), find_house.
find_a_house :-     turn_right,
                    meet_mine, !,
                     see_a_red_brick_house.
enter_village(vasino).
enter_village(vanino).
```

Как только Пролог достигнет отсечения, весь предикат *find_house* станет однозначным, следовательно, дальнейшая неудача в предикате *see_a_red_brick_house* приведет к тому, что предикат *find_house* также завершится неудачей. Если при этом предикат *enter_village* является неоднозначным (как показано в программе), то мы имеем шанс найти дачу в другой деревне, поскольку отсечение в правиле *find_house* не затрагивает свойства предиката *dacha* и установленные в нем указатели отката.

Если предикат *enter_village* является однозначным, то нам придется возвращаться домой. Дачи мы не нашли. Если дом возле колодца окажется из красного кирпича, то наличие отсечения никак не скажется на результате поиска. Таким образом, отсечение следует включать в программу на Прологе всегда, когда нам точно известно, что решение либо единственное, либо его нет вовсе.

Примечание. Если программа поиска дачи будет выглядеть следующим образом:

```
dacha3(X) :-
        enter_village(X),
        turn_right,
        meet_mine,
        !,
        see_a_red_brick_house.
enter_village(vasino).
enter_village(vanino).
```

то отсечение, вставленное в указанном месте, приведет к тому, что все предыдущие предикаты в правиле *dacha* станут однозначными, в том числе и *enter_village*. Иными словами, нам будет запрещено искать дачу в следующей деревне.

Красное и зеленое отсечения

Если отсечение не влияет на логику работы программы, а только сокращает возможные спуски по ложным ветвям дерева решений, то такое отсечение называется зеленым.

Если отсечение влияет на логику программы (без него программа работает иначе), такое отсечение называется красным.

В приведенном выше правиле *dacha2* отсечение зеленое, поскольку не влияет на логику программы, а только избавляет нас от бесполезного блуждания по деревне. В правиле *dacha3* отсечение меняет логику программы, так как поиск ограничивается первым найденным колодцем на всем пути следования к цели, а не в каждой деревне. Следует заметить, что и зеленое отсечение в правиле *dacha2* все же меняет логику программы. Правда, это отсечение отсекает лишь возможность поехать на две дачи сразу.

Использование отсечений позволяет существенно сокращать как время работы программы на Прологе, так и объем требуемой оперативной памяти. Кроме того, вдумчивая расстановка отсечений сокращает отладку программ, поскольку устраняет коллизии, подобные той, что приведена в следующем примере.

Пусть человек считается уважаемым в обществе, если он/она состоит в браке и имеет свой дом. Правило для этого выглядит следующим образом:

respectable(X) :- spouse (X,_),
has_home(X).

Пусть в базе знаний имеются следующие факты и правила:

spouse(alla, filipp).
spouse (X,Y) :- spouse (Y, X).
has_home(alla).

Зададим следующую цель: *respectable(filipp).* Для резолюции данной цели Пролог войдет в тело правила *respectable* и поставит себе подцель: *spouse (filipp,_).* Поскольку такого факта в базе нет, Пролог выполнит откат на правило

spouse (X,Y) :- spouse (Y, X).

из которого создаст себе подцель: *spouse (_, filipp).*

В ходе резолюции данной подцели будет получено успешное решение, поскольку такой факт в базе есть. Далее Пролог ставит себе следующую подцель из правила *respectable*: *has_home(filipp).*

Такого факта в базе нет, и, поскольку предыдущий предикат правила *respectable*, то есть *spouse*, является неоднозначным, то выполняется откат на следующий экземпляр этого предиката, а именно:

spouse (X,Y) :- spouse (Y, X).

Смысл отката заключается в поиске другой жены для Филиппа. Будет поставлена следующая подцель:

spouse (filipp, _).

Таким образом, будет происходить бесконечное углубление в рекурсию.

Совершенно очевидно, что, найдя одну жену Филиппа, мы должны сообщить Прологу, что это решение является единственным, и искать другие не нужно.

```
respectable(X) :- spouse (X,_), !,
has_home(X). spouse(alla, filipp) :- !.
spouse (X,Y) :- spouse (Y, X).
 has_home(alla).
```

Списки в Prolog

Все переменные, с которыми мы работали ранее, были скалярами. Теперь рассмотрим агрегаты данных в Прологе. Один из них – список. Список – это упорядоченная последовательность однотипных элементов. Элементы списка заключаются в квадратные скобки и разделяются запятыми:

```
[ 1, 2, 3, 4, 6, 7, 8]                    – integer
[mon, tue, wed, thu, fri, sat, sun]       – symbol
["Иванов", "Петров"]                      – string
[1.5, 2.22, 0.001, 0]                     – real
[]                                        – пустой список
```

Объявляются списки следующим образом:

```
domains
sym = symbol*       % список символьных значений
intlist = integer*  % список целых
realist = real*     % список действительных чисел
```

Единственная операция, которая допускается над списком – это отсечение головы от хвоста, причем «разместить» эту операцию можно только в аргументах предикатов.

Напишем некоторые полезные предикаты для работы со списками. В частности, как определить, является ли некоторая переменная элементом списка? Разобьем список на голову и хвост. Искомое значение находится либо в голове списка, либо в хвосте:

```
member (H, [H | _ ] ).
member (X, [H | T] ) :- member (X, T).
```

Вышеописанным приемом можно искать не только один элемент, но и больше (например пару последовательных значений в списке)

```
memb2(H1, H2, [H1, H2 | _ ] ).
memb2(H1, H2, [H | T] ) :- memb2(H1, H2, T)
```

Чтобы включить значение в список (проще всего в голову), можно составить такое правило: *incl (H, T, [H | T]).*

Исключить значение из списка несколько сложнее:

```
excl (H, [H | T], T ).        % исключаем из головы
excl (X, [H,T], [H | TT] ) :- excl (X, T, TT).  % исключаем из хвоста
```

Ниже приведены другие примеры предикатов работы со списками:

1. Программа вывода на экран элементов списка по одному:

```
print_list([]).                          % выход из рекурсии
print_list([ H | T ]) :- write(H, " "),  % вывод головы списка и пробела
print_list(T).                           % вывод хвоста
```

Если задать цель

printlist([“я”, “помню”,”чудное”,”мгновенье”]).

то результат будет следующим: *я помню чудное мгновенье*

2. Та же программа, но выводящая список в обратном порядке:

```
print_inverse([]).                % выход из рекурсии
print_inverse ([ H | T ]) :-
        print_inverse (T),        % вывод хвоста
        write (H), write(" "),    % вывод головы списка и пробела
```

Цель:

write_inverse ([“я”, “помню”,”чудное”,”мгновенье”]).

Результат: *мгновенье чудное помню я*

3. Программа, подсчитывающая сумму элементов списка:

```
sum([], 0).              % выход из рекурсии,
sum([ H | T ], S ) :-
        sum(T,S1),       % S1 – сумма элементов в хвосте списка
S = S1 + H.              % S – остается добавить к сумме только голову
```

Цель:

sum([1,2,3,4,5,6], S), write (“Сумма =”,S).

Результат: *Сумма=21*

4. Программа, подсчитывающая количество элементов списка:

```
count([], 0).          % выход из рекурсии,
count([ H | T ], S ) :-
        count(T,S1),   % S1 – счетчик элементов в хвосте списка
S = S1 + 1.            % S – остается добавить к сумме единицу
```

Цель:

count([1,2,3,4,5,6], S),
write (“Количество =”,S).

Результат: *Количество =6*

Пример: Решение логической задачи о волке, козе и капусте

Рассмотрим известную логическую задачу о волке, козе и капусте. Фермер должен переправить на другой берег реки волка, козу и капусту. Грузоподъемность лодки такова, что за один раз можно взять на борт что-нибудь одно: или волка, или козу, или капусту. В присутствии старика никто никого не ест. Если же он отлучится, то волк съест козу, а коза – капусту.

Для решения этой задачи организуем два списка. Один список будет отражать содержимое левого берега, второй – правого. Первоначально все

находятся на левом берегу. Список левого берега: [wolf, goat, cabbage], список правого берега пустой: [ ].

Определим предикаты, описывающие задачу.

stuff(wolf). % перечисление груза
stuff(goat).
stuff(cabbage).

/* условия возникновения конфликта */
conflict(X): - member(wolf, X), member(goat, X).
conflict(X); - member(goat, X), member(cabbage, X).

/* Предикаты, описывающие перемещения лодки:
С левого берега на правый */
go_right([ ], _). % условие завершения (список левого берега пуст)
go_right(L, R) :-
stuff(X), % выбираем груз,
member(X, L), % который есть на левом берегу
excl(X, L, LL), % исключаем из списка левого берега
not(conflict(LL)), % на левом берегу конфликта нет
incl(X,R,RR), % включаем в список правого берега write(LL,"--",X,”-->”,R),
% выводим сообщение
go_left(LL,RR). % гребем назад

/* Движение влево возможно в двух вариантах. Если на правом берегу конфликт не возникает, то фермер едет один */
go_left(L,R) :- not(conflict(R)),
write(L,"<-------”,R), % выводим сообщение
go_right(L, R). % вызываем предикат движение вправо

/* Если на правом берегу возникает конфликт, надо кого-нибудьувезти обратно. Это единственная подсказка, которую мы сообщаем программе. В остальном Пролог будет искать решение вполне самостоятельно */
go_left(L,R) :-
stuff(X), % выбираем груз,
member(X, R), % который есть на правом берегу
excl(X, R, RR), % исключаем из списка правого берега
not(conflict(RR)), % на правом берегу конфликта нет
incl(X,L,LL), % включаем в список левого берега
write(L,"<--",X,”--”,RR),% выводим сообщение
go_right(LL,RR). % гребем назад

Вызов цели должен быть следующим:
go_right([wolf, goat, cabbage], [ ]).

Если запустить эту программу, то быстро обнаружится, что первым рейсом старик отвезет на правый берег козу. Вторым рейсом – волка. Оставить волка с козой на правом берегу нельзя, поэтому он отвезет первый попавшийся груз, а им окажется волк, обратно. И так будет возить его бесконечно. Недостаток данной программы заключается в том, что фермер не помнит, кого он только что привез. Для укрепления его памяти добавим в предикаты

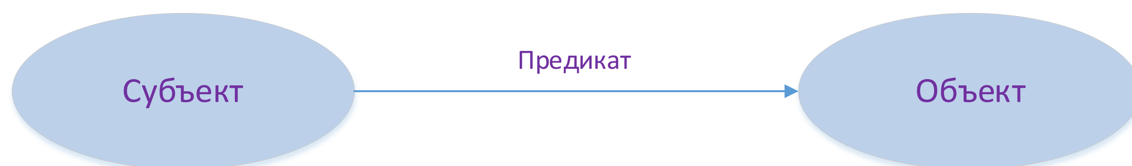
go_left и *go_right* название последнего перевезенного груза. Финальный вариант программы выглядит следующим образом (изменения выделены полужирным шрифтом):

```
/*      Задача о волке, козе и капусте      */
 domains    % раздел требуется для PDC-Пролога. В SWI-Прологе не нужен
        stuff = wolf; goat; cabbage; nil % создаем свой тип данных ( nil – пустое)
        list  = stuff* % тип данных список
        predicates  % раздел требуется для PDC-Пролога. В SWI-Прологе не
        нужен
        member(stuff,list)
        incl(stuff,list,list)
        excl(stuff,list,list)
        conflict(list)
        go_right(list, list, stuff)
        go_left(list, list, stuff)
        clauses
        stuff(wolf). stuff(goat). stuff(cabbage). member (H, [H | _ ] ).
        member (X, [H | T] ) :- member (X, T).incl (H, T, [H | T]).
        excl (H, [H | T], T ).
        excl (X, [H,T], [H | TT] ) :- excl (X, T, TT).
        conflict(X): - member(wolf, X), member(goat, X).
        conflict(X); - member(goat, X), member(cabbage, X).
        go_right( L, R,Last) :-
        stuff(X),       % выбираем груз,
        X \= Last,      % но не тот, что везли только что и
        member(X, L),       % который есть на левом берегу
        excl(X, L, LL),     % исключаем из списка левого берега
        not(conflict(LL)),  % на левом берегу конфликта нет
        incl(X,R,RR),       % включаем в список правого берега
        write(LL,"--",X,”-->”,R),       % выводим сообщение
        go_left(LL,RR,X).  % гребем назад
/*      Если на правом берегу конфликт не возникает, то едет один */
        go_left(L, R, Last) :- not(conflict(R)),
                                  write(L,"<-------”,R),  % выводим сообщение
        o_right(L, R, nil).  % вызываем предикат движение вправо
                               % nil – значит, не везли ничего
/*      Если на правом берегу возникает конфликт, надо кого-нибудьувезти
обратно, но не того, кого везли только что*/
        go_left(L,R,Last) :-
                        stuff(X),       % выбираем груз,
                        X <> Last,   % не тот, что везли только что и
                        member(X, R),       % который есть на правом берегу
                        excl(X, R, RR),      % исключаем из списка правого берега
```

```
                    not(conflict(RR)),  % на правом берегу конфликта нет
                    incl(X, L, LL),       % включаем в список левого берега
                    write(L,"<--",X,"--",RR),         % выводим сообщение
                    go_right(LL, RR, X).    % гребем назад и помним, что везли X
    goal
    go_right(([wolf, goat, cabbage], [ ], nil).
/* Конец программы      */
```

Семантические сети

Семантическая сеть - структура для представления знаний с помощью графа, в виде узлов, соединенных дугами. Узлы соответствуют понятиям, а дуги – отношениям между ними. Элементом семантической сети в простейшем случае является триплет следующего вида:



Одно и то же понятие может присутствовать в нескольких триплетах, что и обуславливает сетевую структуру. Основным свойством семантических отношений является **арность**, т.е. количество аргументов. Выше приведен пример **бинарного** отношения, т.е. отношения с арностью 2. Если все отношения в сети однотипные, то сеть называется **однородной**. Пример однородной сети – классификация биологических видов. В **неоднородной** сети количество типов отношений больше одного.

Основная цель применения семантических сетей для представления знаний – обеспечение независимости от языка, а также устранение неточностей и двусмысленностей, свойственных естественным языкам. Естественный язык – как живой организм, развивается и эволюционирует в сторону совершенствования основной своей цели – обеспечения понимания участников разговора. Однако, в силу природной лени носителей языка, эта оптимизация зачастую сводится к предельному упрощению конструкций, в результате чего смысл отдельной фразы выявить можно только из контекстного окружения. Хрестоматийный пример даже не двусмысленности, а «трехсмысленности» являет фраза «Он встретил ее на поляне с цветами». Совершенно непонятно, где цветы: у него, у нее или на поляне. От неоднозначности страдают все языки, включая английский. Так, вопрос “Tell me, what has four wheels and flies” заставляет нас вспомнить, какой летательный аппарат имеет четыре колеса. Речь при этом идет о “garbage truck”, и «flies» означает «мухи», а не «летает». Семантическая же сеть должна содержать знания в математически точной форме.

Историческая справка

Математика позволяет описать большинство явлений в окружающем мире в виде логических высказываний. Семантические сети возникли как попытка визуализации математических формул. Прародителями современных семантических сетей можно считать экзистенциальные графы, предложенные Чарльзом Сандерзом Пирсом в 1909 г. Они использовались в органической химии для представления логических высказываний в виде особых диаграмм. Пирс назвал этот способ «логикой будущего».

Важным начинанием в исследовании сетей стали работы немецкого психолога Отто Зельца 1913 г. и 1922 г. В них для организации структур понятий и ассоциаций, а также изучения методов наследования свойств он использовал графы и семантические отношения. Научные изыскания Зельца имели огромное влияние на изучение тактики в шахматах, которые в свою очередь повлияли на таких теоретиков, как Саймон и Ньюэлл. Исследователи Дж. Андерсон (1973), Д. Норман (1975) и другие использовали эти работы для моделирования человеческой памяти и интеллектуальных свойств.

Что касается лингвистики, то первым ученым, занимавшимся разработкой графических описаний, стал Теньер. Он использовал графическую запись для своей грамматики зависимостей. Теньер существенно повлиял на развитие лингвистики в Европе. Компьютерные семантические сети были детально разработаны Ричардом Риченсом в 1956 году в рамках проекта Кембриджского центра изучения языка по машинному переводу. Процесс машинного перевода подразделяется на 2 части: перевод исходного текста в промежуточную форму представления, а затем эта промежуточная форма транслируется на нужный язык. Такой промежуточной формой как раз и были семантические сети. Первая такая система, которую создала Мастерман, включала в себя 100 примитивных концептов, таких как, например, НАРОД, ВЕЩЬ, ДЕЛАТЬ, БЫТЬ. С помощью этих концептов она описала словарь объемом 15000 единиц, в котором также имелся механизм переноса характеристик с гипертипа на подтип. Некоторые системы машинного перевода базировались на корреляционных сетях Цеккато, которые представляли собой набор 56 различных отношений, некоторые из которых - падежные отношения, отношения подтипа, члена, части и целого. Он использовал сети, состоящие из концептов и отношений для руководства действиями программы текстового разбора и разрешения неоднозначностей. Эти исследования были продолжены Робертом Симмонсом (1966), Уилксом (1972) и другими учёными.

В системах искусственного интеллекта семантические сети используются для ответа на различные вопросы, изучение процессов обучения, запоминания и рассуждений. В конце 70-х сети получили широкое распространение. В 80-х годах границы между сетями, фреймовыми структурами и линейными формами записи постепенно стирались. Выразительная сила больше не является решающим аргументом в пользу

выбора сетей или линейных форм записи, поскольку идеи, записанные с помощью одной формы записи, могут быть легко переведены в другую. И наоборот, особо важную роль стали играть такие факторы, как читаемость, эффективность, неискусственность и теоретическая элегантность, также учитываются легкость введения в компьютер, редактирование и распечатка [5]. На понимание проблем представления знаний с помощью семантических сетей большое влияние оказало эссе Дрю МакДермота «Искусственный интеллект сталкивается с естественной глупостью» [6], которое цитируется уже в течение 40 лет.

Наиболее продвинутой отечественной разработкой в области применения аппарата семантических сетей в задачах анализа и поиска текстовой информации является набор программных продуктов под торговой маркой RCO (Russian Context Optimizer) компании «Гарант Парк Интернет» (www.rco.ru).

Наконец, следует упомянуть концепцию Семантической Паутины (Semantic Web), которая будет подробнее рассмотрена в разделах «Проблемы построения семантических сетей» и «Факты и правила в семантической сети». Семантическая Паутина – это дальнейшее развитие Всемирной Паутины, заключающееся в том, что документы, размещаемые на ее ресурсах, содержат семантическую разметку, позволяющую извлечь смысл информации, содержащейся в этих документах. Такие семантические документы создаются, в основном, в формате RDF (Resource Description Framework) – расширении языка XML, дополненном средствами представления триплетов субъект-предикат-объект.

Типы семантических сетей

Семантическая сеть, в которой все отношения бинарные, образует **реляционный граф** (рисунок 2).

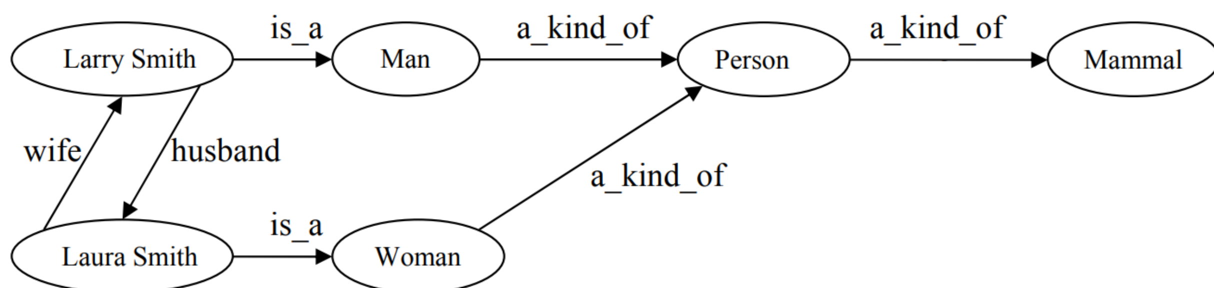


Рис.2. Пример реляционного графа

Несмотря на то, что понятия “Larry Smith” и “Man” обозначаются на графе одинаковыми эллипсами, их смысловое качество различно. “Larry Smith” – экземпляр класса мужчин, а “Man” – множество мужчин. В свою очередь, “Man” и “Woman” являются подмножествами более мощного множества людей (“Person”). Таким образом, как и в реляционных базах данных, здесь имеют место отношения «один к одному», «один ко многим»,

«многие к многим».

Если арность отлична от двух, изобразить семантическую сеть в виде графа уже сложнее. Но, как и в базах данных, можно провести нормализацию и привести все к бинарным отношениям. Например, унарное отношение типа «мотор – работает» можно привести к виду «мотор» -> «состояние» -> «работа». Отношение с арностью 3 типа «Самолет летит из Петербурга в Москву» оперирует с тремя понятиями: субъект – это самолет, а объекты – Москва и Петербург. Предикат – выполнение полета («летит»). Введя дополнительное понятие «полет», данное отношение можно представить следующим фрагментом семантической сети (рисунке 3):

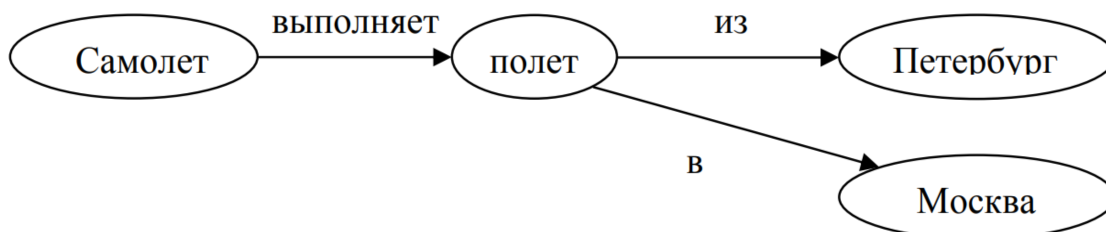


Рис. 3. К определению концептуального графа

Во многих случаях в качестве понятия в отношении участвует не простой объект или субъект, а другое отношение или целый фрагмент семантической сети. Например, мальчик видит в небе самолет и думает, что этот самолет летит из Петербурга в Москву.

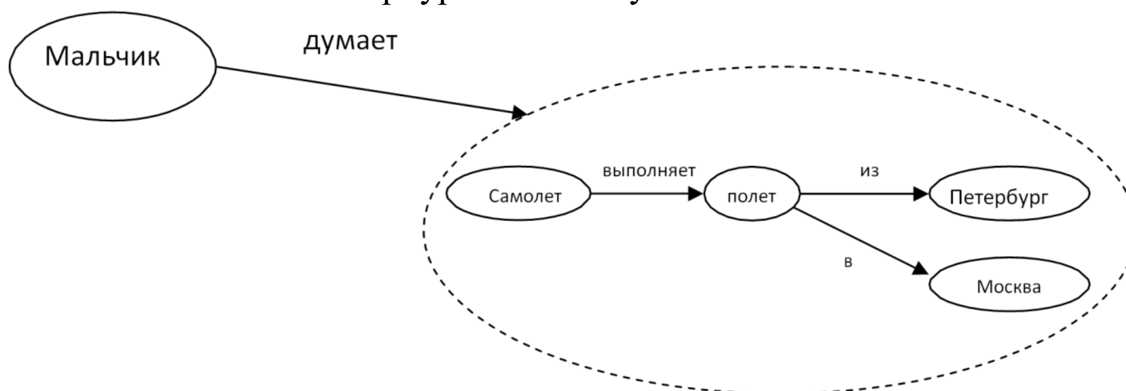


Рис.4. Концептуальный граф

Такие сети называются **пропозиционными сетями**, а граф такой сети – **концептуальным графом** (рисунок 4). Вложенность отношений в пропозиционных сетях может быть сколь угодно велика. Например, родители мальчика могут полагать, что он ошибается, если думает, что самолет летит из Петербурга в Москву, а их дедушка и бабушка, в свою очередь, могут считать, что родители недооценивают дедуктивные способности внука, и т.д. Для представления событий более подходит **граф с глаголом в центре** или граф Растье. Пример: Собака кусает почтальона (рисунок 5).

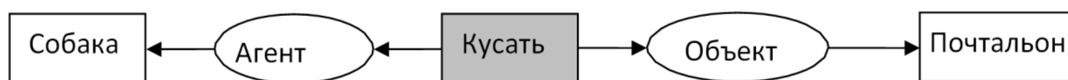


Рис.5. Граф Растье

В основе лежит не понятие, а действие или событие, в данном случае кусание. Субъектом или агентом кусания является собака, а объектом – почтальон. Граф с центром в глаголе позволяет обходиться без вложенности графов.

Мы можем достаточно просто расширить базу знаний о данном событии (рисунок 6).

Следствием того, что собака свирепо покусала почтальона, стало то, что почтальон поколотил во дворе хозяина собаки.

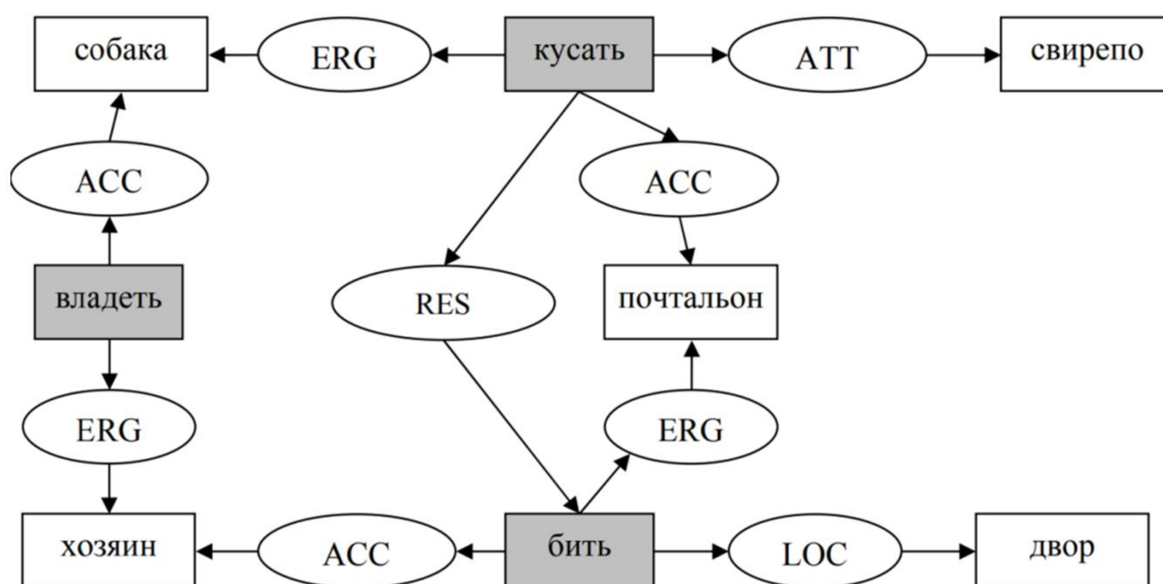


Рис.6. Семантическая сеть «Почтальон»

Типы отношений в семантических сетях

Самым распространенным типом отношений в семантических сетях является иерархический тип, описывающий отношения между элементами, множествами и частями объектов. К ним относятся:

отношение классификации ISA (от английского “is a”). Говорят, что множество (класс) классифицирует свои экземпляры (например, “Сократ есть человек”). Иногда это отношение именуют “member of”. По-русски это может называться «есть» (единственное число) или «суть» (множественное число). Обратное отношение – “example of” или «пример».

Отношение между множеством и подмножеством AKO (“a kind of”), например, «Магистры – подмножество студентов». Отличие от отношения ISA заключается в том, что классификация – отношение «один ко многим», а подмножество – «многие ко многим».

Отношение целого и части. Отношение меронимии – отношение целого к части (“**has part**”). Мероним – объект, являющийся частью другого объекта. Отношение холонимии – отношение части к целому (“is a part”). Рука – холоним для тела. Тело – мероним для руки. Рука – часть тела!

Применяя иерархические типы отношений, следует четко различать, какие объекты являются классами, а какие – экземплярами классов. При этом вовсе необязательно одно и то же понятие будет классом или экземпляром во всех предметных областях. Так, «человек» всегда будет классом в базах знаний типа «студенческая группа» или «трудовой коллектив», но может быть экземпляром класса млекопитающих в базе знаний по биологии.

Вершины семантического графа могут обозначать не только объекты, но и свойства или значения свойств. Отображение свойств на графе повышает его наглядность, но может сильно загромоздить его.

Кроме иерархических отношений в семантических сетях часто используются следующие типы отношений (во вторых скобках указаны типы вершин):

- функциональные связи («производит», «влияет», …) (объект – объект);
- количественные («больше», «меньше», «равно», …) (объект – объект илиобъект – свойство);
- пространственные («далеко от», «близко к», «за», «над», «под», «выше», …) (объект – объект);
- временные («раньше», «позже», «одновременно с», …) (объект – объект);
- атрибутивные («иметь свойство», «иметь значение»,…) (объект – свойство или свойство - значение);
- логические («и», «или», «не») (объект – объект или свойство – свойство);
- лингвистические.

Число типов отношений может быть очень большим. Основная проблема при этом заключается в возможности идентификации этих отношений в запросах к базе знаний. В этой связи предпочтительным является сокращение числа типов связей (и вершин) за счет увеличения числа вершин. Например, вместо отношения «семантсеть» - «предназначена» - «представление_данных» можно использовать «семантсеть» - «имеет» - «назначение»; «назначение» - «есть» - «представление»; «представление» - «чего» - «данных». Сети с глаголом в центре (сети Растье) оперируют со следующими типами связей [5], приведенными в таблице 1. Рисунок 7 содержит граф Растье, описывающий пример с почтальоном, где отношения приведены к стандартной форме.

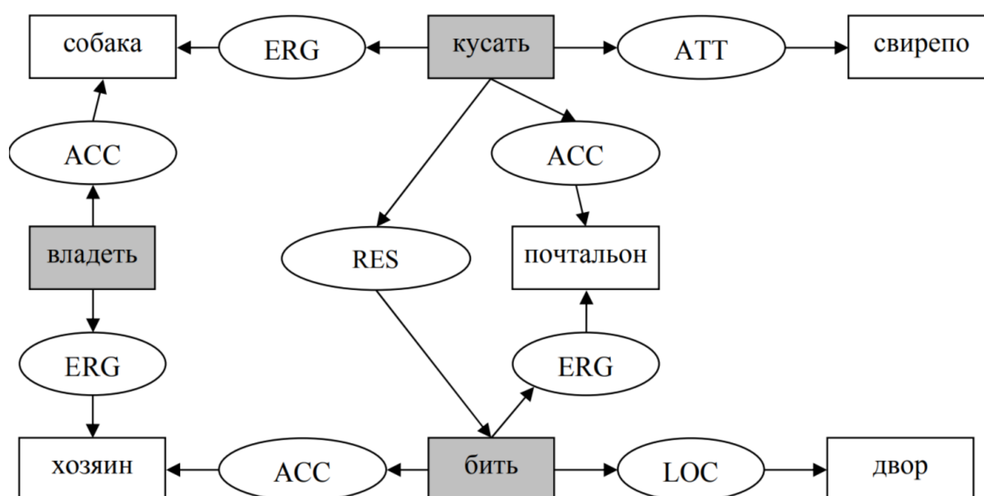


Рис.7. Граф Растье со стандартными отношениями

Таблица 1. Типы связей в графах Растье

Имя	Тип	Определение	Упрощенное имя
(ACC)	accusative	Объект воздействия	PATient
(ASS)	assumptive	Точка зрения	PERspective
(ATT)	attributive	Свойство, характеристика	CHARacteristic
(BEN)	benefactive	Сущность, выступающая вроли выгодоприобретателя	BENeficiary
(CLAS)	classitive	Экземпляр класса	CLASsitive
(COMP)	comparative	Элементы, объединяемые сравнением	COMParison
(DAT)	dative	Получатель	RECeiver
(ERG)	ergative	Эргатив, агент процесса или действия	AGEnt
(FIN)	final	Результат или ожидаемая цель	GOAL
(INST)	instrumental	Использованные средства	MEAns
(LOC S)	spatial locative	Положение (позиция) в пространстве	SPAce
(LOC T)	temporal locative	Положение (позиция) во времени	TIME
(MAL)	malefactive	Сторона, пострадавшая в результате действия	MALeficiary
(PART)	partitive	Часть целого	PARTitive
(RES)	resultative	Результат, эффект, следствие	EFFect (или CAUse)

Онтологии и правила наследования отношений

Представление знаний – весьма сложный и творческий процесс. Основная проблема здесь заключается в том, что создание базы знаний почти всегда начинается «с нуля»; при этом отсутствуют так называемые знания начального уровня, которыми человек обзаводится, начиная с раннего детства. Создание таких баз бытовых знаний ведется уже более 35 лет компанией Cycorp. (www.cyc.com), однако, по признанию ее основателя Дагласа Лената (Douglas Lenat), машина все еще нуждается в том, чтобы ей в явном виде сообщали, что родители старше своих детей, и что люди перестают выписывать газеты, когда умирают. Знания конкретной предметной области можно разделить на общие для всех экземпляров и индивидуальные для каждого. Очевидно, что если формализация знаний для класса объектов будет выполнена один раз, то затем она может использоваться другими авторами при описании отдельных экземпляров.

Таким образом, формализация знаний любой предметной области должна базироваться на знаниях более общего уровня, описывающих основные отношения между объектами и общие свойства объектов. Такие знания называются онтологиями. Например, онтология для описания класса «человек», может содержать отношения “has part” с объектами «рука», «голова» и свойства «имеет имя», «имеет дату рождения» и т.п.

Применение таких знаний к объектам нижнего уровня осуществляется с помощью правил наследования:

Если X AKO Y и Y AKO Z то X AKO Z – если X является подмножеством Y, а тот, в свою очередь, частью еще большего множества Z, то класс X является подмножеством Z.

Если X ISA Y и Y AKO Z то X ISA Z – если X является экземпляром класса Y, а тот, в свою очередь, частью множества Z, то X является экземпляром класса Z.

Если X has_part Y и Y has_part Z то X has_part Z – если X имеет в своем составе Y, а тот, в свою очередь, имеет составную часть Z, то X имеет в своем составе Z.

Если X ISA Y и Y has_part Z то X has_part Z – если X является экземпляром класса Y, а тот, в свою очередь, имеет составную часть Z, то экземпляр X имеет в своем составе Z.

Если X AKO Y и Y has_part Z то X has_part Z – если X является подмножеством Y, а тот, в свою очередь, имеет составную часть Z, то класс X имеет в своем составе Z.

Если X ISA Y и Y has_a Z то X has_a Z – если X является экземпляром класса Y, а тот, в свою очередь, имеет свойство Z, то экземпляр X имеет свойство Z.

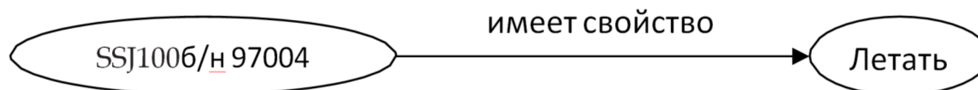
Если X AKO Y и Y has_a Z то X has_a Z – если X является подмножеством Y, а тот, в свою очередь, имеет свойство Z, то класс X имеет свойство Z.

Использование правил наследования позволяет описать все свойства каждого экземпляра, указать лишь принадлежность к классу, для которого все необходимые свойства уже описаны.

Проблемы построения семантических сетей

Семантическая сеть должна хранить знания в математически точной форме. В этой связи её построение требует аккуратности и хорошего понимания предметной области и всех связанных с ней понятий. Проблемы представления знаний были изложены в уже упомянутой работе Дрю Макдермота [6]. На первый взгляд, построение примеров графов, подобных приведенным в разделе «Типы семантических сетей», может проходить легко и непринужденно. Однако это далеко не всегда так, рассмотрим следующую конструкцию:

Иными словами, SSJ100 с бортовым номером 97004 является экземпляром класса самолетов, а самолеты обладают свойством летать. Поскольку отдельный представитель класса наследует признаки класса, мы делаем вывод, что SSJ100 с бортовым номером 97004 тоже может летать,



и будем правы. Но если теперь мы наложим на такую же схему другие факты (очень старый и широко известный парадокс):

Сократ – один из людей.
Людей много.
Следовательно, Сократов много.

А ведь в отношении самолета это работало! Мы попали в ловушку по следующей причине. Понятия «Самолеты» и «Люди» - это множества самолетов и людей соответственно. Свойство «летать» относится не к множеству, а к его экземплярам. Свойство же «Многочисленность» относится к классу в целом и неприменимо к отдельным экземплярам данного класса. Таким образом, всегда следует четко понимать, что соответствует тому или иному понятию, единичный экземпляр или подмножество, и правильно применять предикаты.

Другая проблема связана с присвоением понятиям имён. В приведенном выше примере с самолетом не зря конкретный самолет был идентифицирован бортовым номером. В противном случае было бы непонятно, какой именно экземпляр самолета имеется в виду.

Для выявления свойства «летать» точная идентификация экземпляра не критична, чего нельзя сказать, если речь идет о необходимости устранении

неисправности. Приведенная выше идентификация 97004 также не является исчерпывающей. Во-первых, данная комбинация цифр может относиться к чему угодно, например, номеру телефона. Во-вторых, самолет может попасть в гражданскую авиацию из авиации военной, где идентификация совершенно другая, и мы не сможем ничего узнать о прошлой жизни самолета, например, о предыдущих ремонтах, а это совершенно недопустимо.

Проблема идентификации понятий имеет место и в реальной жизни, но здесь мы мудро пользуемся «бритвой Оккама»: «Не умножай сущностей сверх необходимости». Если мы общаемся в небольшой компании, достаточно имен. В студенческой группе можно идентифицировать каждого фамилией. Однофамильцы при этом удостаиваются имени, а тезки – еще и отчества. Если создается семантическая сеть, охватывающая большое количество объектов, то неизбежно возникает проблема **синонимии**, когда одно имя указывает на различные понятия. И если в древности достаточно было сказать «Труффальдино из Бергамо», «Иисус из Назарета», чтобы идентифицировать человека, то сейчас даже используемая в паспортном учете триада «ФИО» - «дата рождения» - «место рождения» не гарантирует отсутствия повторяющихся идентификаторов. Кроме того, такой громоздкий ключ (выражаясь в терминах баз данных) не способствует наглядности и простоте восприятия. В этой связи для локальных семантических сетей могут использоваться принятые в практике номера зачетных книжек, табельные номера, ИНН и т.п. Другая проблема – **полисемия**, когда одно слово используется для обозначения различных понятий. Синонимия и полисемия могут катастрофически усложнить проблему построения больших сетей и, в частности, объединение фрагментов, написанных разными авторами.

Название вершины является всего лишь символическим именем, его осмысленность только увеличивает наглядность графа. Полностью идентифицируют вершину ее свойства, например, для человека – фамилия, имя, отчество, дата рождения и т.п.

Факты и правила в семантической сети

Рассмотренные выше отношения, записанные в виде субъект-предикат-объект, представляют собой неизменные знания, т.е. факты. Занесение всех известных фактов о каждом объекте может потребовать неоправданно много времени. В качестве примера уместно привести родственные отношения. Для любых двух родственников есть название отношения между ними: дядя-племянник, свекровь-зять и т.п. Таким образом, для семьи из n=10 человек число отношений будет равно n*(n-1) = 90. При этом часть отношений являются первичными (супруг-супруга и родитель-ребенок), остальные отношения вытекают из первичных. Если информацию о том, как вторичные отношения определяются на основе первичных, записать в виде правил, то для каждого объекта можно заносить в базу знаний только первичные факты. Для семейных отношений это означает сокращение не менее чем в n/3 раз,

если считать, что каждый член семьи является чьим-то ребенком и родителем, а также чьим-то супругом, и не более того.

Одним из стандартов языка представления правил является SWRL - Semantic Web Rule Language (http://www.w3.org/Submission/SWRL/). Данный язык является расширением XML, и конструкции на нем предназначены исключительно для машинной интерпретации. Редакторы правил обычно предоставляют вариант "human readable" правил в формате, подобном правилам на Прологе, для их создания и отладки. Ниже приведен фрагмент правила определения отношения «дядя» для чтения человеком и исходный текст данного правила на языке SWRL.

```
hasParent (?x1, ?x2)& hasBrother (?x2, ?x3) => hasUncle (?x1, ?x3)
<swrl:Imp rdf:ID = "Bros">
  <swrl:body>
    <swrl:AtomList>
      <rdf:first>
        <rdf:Description>
          <rdf:type rdf:resource = "&swrl;ClassAtom"/> <swrl:argument1>
            <rdf:Description rdf:about = "#x3"/> </swrl:argument1>
          <swrl:classPredicate rdf:resource = "#Person"/> </rdf:Description>
      </rdf:first>
      <rdf:rest>
        <swrl:AtomList>
          <rdf:first>
            <rdf:Description>
              <rdf:type rdf:resource = "&swrl;IndividualPropertyAtom"/>
              <swrl:argument2>
                <rdf:Description rdf:about = "#x3"/>
            <rdf:Description rdf:about = "#x1"/> </swrl:argument1>
          <swrl:propertyPredicate rdf:resource = "#hasBrother"/>
        </rdf:Description>
  </rdf:first>
  <rdf:rest rdf:resource =  &rdf;nil"/>
  </swrl:AtomList>
  </swrl:head>
</swrl:Imp>
```

Устанавливая правила для объектов семантической сети, мы можем столкнуться с проблемой открытого или закрытого мира (Open or Closed World Assumption). Допущение об открытом мире предполагает, что никто не располагает полной информацией об окружающем мире, следовательно, выводы должны делаться исключительно на основании того, что известно. Допущение о закрытом мире предполагает, что вся информация известна наблюдателю. В качестве примера можно привести родственное отношение

типа «мачеха». Пусть в базе знание имеются следующие факты:

Андрей является родителем Егора.

Юлия является супругой Андрея.

В соответствии с допущением закрытого мира, Юлия является мачехой Егора, поскольку в базе нет сведений о том, что она его мать. В открытом мире Юлию можно считать мачехой Егора только в том случае, если известно, что его матерью является не Юлия, а другая женщина.

Применение правил крайне полезно в тех случаях, когда в базе знаний содержится неполная информация. Пусть, например, в предыдущем примере указано, Андрей – это человек, но нет сведений, что Юлия – человек. Тогда все правила наподобие

*Если **X** человек, то **X** имеет фамилию*

не смогут быть применены к Юлии. Если же создать правило

*Если **X** человек И **X** супруг **Y** ТО **Y** человек*

то можно будет установить факт, что Юлия тоже человек. Помимо положительного эффекта от использования правил имеется и недостаток: комбинаторная сложность, которая по мере увеличения объема базы знаний довольно быстро вырастает до космических масштабов. Так, например, если в достаточно маленькой базе знаний имеется 100 фактов и 10 правил по три факта в каждом, то общее количество попыток применить правила к фактам может достигать значения $10*100*100*100 = 10^7$, поскольку в каждое правило последовательно будут подставляться все возможные факты. Очевидно, что такая «наивная реализация» поиска в семантической сети нежизнеспособна. Как в любой задаче поиска, здесь нужно решать проблему сокращения комбинаторной сложности. В качестве примера ускорения обработки правил можно привести алгоритм **Rete** (Рити) (http://en.wikipedia.org/wiki/Rete_algorithm), основной смысл которого заключается в том, что строится дерево, каждый узел которого соответствует части условий правил и хранит список фактов, удовлетворяющих этим условиям. Поскольку в ходе применения правил постоянно возникают новые факты, они прогоняются по сети, и списки фактов при вершинах обновляются. Узким местом алгоритма **Rete** является большой требуемый объем памяти, поскольку одни и те же факты многократно дублируются в списках при вершинах графа.

В качестве одного из альтернативных путей решения можно запускать все возможные правила для каждого документа один раз и сохранять результаты в виде фактов. Для базы родственных связей это будет означать, что в документ вначале заносятся только первичные связи (родители-дети и супруги), затем из них вычисляются все опосредованные отношения (внучатые племянники и т.д.), которые на равных правах затем пополняют базу знаний. После этого все факты будут извлекаться одинаково быстро. Такой подход называется выводом на основе прецедентов (Case based reasoning), и его можно считать аналогом навыков в человеческом интеллекте. На самом деле, мы почти всегда действуем по аналогиям,

например, в устной речи. Если бы мы при построении каждой фразы применяли правила языка, то скорость речи не превышала бы нескольких предложений в час. Это особенно заметно, когда мы переводим русский текст на иностранный язык. Если языковая конструкция нам знакома (многократно использовалась ранее), то перевод идет в быстром темпе. Если мы создаем предложение в первый раз, то процесс перевода замедляется в десятки и сотни раз.

Интеллектуальный агент семантической сети

Построить семантическую сеть – задача непростая. Но после того, как мы с ней справимся, возникнет вопрос, а как извлекать знания из семантической сети? Очевидно, для этой цели должна быть разработана специальная программа, которая на основе запроса пользователя проведет поиск требуемых знаний и выдаст результат.

В настоящее время существует несколько языков запросов к базам знаний в виде семантических сетей в формате RDF, в частности, DQL, R-DEVICE, RDFQ, RDQ, RDQL, SeRQL. Наиболее стандартизованным является язык SPARQL, прошедший стандартизацию в группе Data Access Working Group (DAWG) консорциума World Wide Web (W3C). Существуют несколько реализаций языка SPARQL для различных программных платформ. Запросы на языке SPARQL обрабатывают только факты (триплеты субъект-предикат-объект), но не понимают правил. Тем самым вся работа по созданию онтологий становится бессмысленной.

Для устранения этого недостатка разработан упрощенный язык представления семантических документов и программа, поддерживающая визуализацию знаний и выполнение несложных запросов. Программа SEMANTIC, предлагаемая в рамках данной дисциплины в качестве оболочки для создания и исследования семантических сетей, содержит зачатки свойств такого интеллектуального агента. В частности, программа применяет ко всем фактам, записанным в базу знаний, правила наследования, а также позволяет пользователю создавать собственные правила.

Управление контекстом

Необходимость однозначно идентифицировать все объекты семантической сети приводит не только к усложнению процедуры добавления фактов, но и к тому, что извлечение знаний становится очень громоздким. Упростить понимание этой проблемы можно на простом примере. Пусть мы хотим на денек попросить у соседа конспект лекций по искусственному интеллекту, который он, в свою очередь, одолжил у своей подружки. Тогда диалог будет приблизительно следующим:

«Гражданин Российской Федерации Сидоров Владимир Иванович, родившийся в 2002 году в г. Саратове, имеющий паспорт № 60 04 123456,

выданный 20.05.2016 51-м ОМ г. Санкт-Петербурга, дай мне, гражданину Российской Федерации Петрову Ивану Викторовичу, родившемуся в 22.04.2003 г. в г. Пскове, имеющему паспорт № 6606 654321, на 24 часа 00 минут 00 секунд конспект лекций по дисциплине «Искусственный интеллект», который читает д.т.н., профессор факультета программной инженерии и компьютерной техники Бессмертный Игорь Александрович, ...».

Фраза, немыслимая в повседневной ситуации, но совершенно нормальная в милицейском протоколе.

Очевидно, что при создании семантической сети один раз можно постараться и идентифицировать все объекты однозначно, хотя это существенно усложнит работу. Но для доступа к знаниям необходимо дать возможность вести упрощенный диалог, подобный имеющему место в реальной жизни. Такая функция может быть возложена на интеллектуальный агент, осуществляющий доступ к знаниям.

База контекста должна состоять из двух компонентов: постоянного и временного. Постоянный контекст – это знания, не изменяющиеся в процессе диалога. Например, мы хотим узнать, который час. На этот вопрос, который ни у кого не вызывает затруднений, не может быть получен ответ без информации о местоположении субъекта. Следовательно, в базе контекста должна быть информация о том, где находится субъект, а также часовой пояс данного места. Иными словами, в базу должно быть загружено контекстное окружение.

Временный контекст – это факты, которые устанавливаются или уничтожаются (забываются) в процессе диалога, а также временные ассоциации, устанавливаемые для упрощения диалога. Временные факты – это, например, ответы на вопросы, которые были заданы ранее, т.е. знания, принесенные извне и не требующие сохранения в базе знаний. Примером таких фактов могут быть ответы пациента на вопросы врача, который пытается поставить диагноз. Отсутствие такой памяти сделает диалог похожим на многочисленные анекдоты про деменцию. Временные ассоциации дают возможность присвоить объектам или фактам короткие имена для использования только в данном диалоге. Временные ассоциации широко используются как в повседневной жизни, так и в документах. Например, в текстах договоров обычно используется оборот типа «ООО РОГА И КОПЫТА в лице директора Фунта А.А., действующего на основании устава, именуемое в дальнейшем ПОКУПАТЕЛЬ…».

Таким образом, база контекста позволит создать для пользователя упрощенное представление (модель) семантической сети, которое позволит вести диалог в привычном виде.

Семантическая сеть и Семантическая паутина

Семантическая сеть имеет давнюю историю, и до последнего времени это понятие не вызывало никаких двусмысленностей. Однако, после того, как

изобретатель Всемирной паутины (WWW) Тим Бернерс Ли в 2001 году провозгласил концепцию "Semantic Web", в русскоязычной литературе эти два понятия стали смешиваться. Для устранения недоразумений предлагается переводить понятие "Semantic Web" как «Семантическая Паутина» или, в данном тексте – просто «Паутина». Суть идеи Тима Бернерса Ли заключается в том, чтобы снабдить ресурсы Интернета специальными метаданными, доступными для компьютерной обработки и однозначно характеризующими свойства и содержание ресурсов Всемирной паутины, вместо используемого в настоящее время текстового анализа документов. В частности, предполагается снабдить все ресурсы Интернета тегами, позволяющими установить автора каждого документа. Следует отметить, что это решение сделано «вдогонку» к концепции WWW после того, как Интернет из хранилища знаний превратился в информационную «помойку» из-за допускаемой анонимности ресурсов. Однако данное свойство семантической паутины нас интересует в последнюю очередь. В центре нашего интереса – свойства Семантической Паутины доставлять пользователю информацию, которую он хочет получить.

Основное отличие Семантической Паутины от WWW заключается в том, что связи между ресурсами WWW отражают размещение документов, а связи Паутины – содержание. Правда, как в WWW, так и в Паутине ссылки содержат адреса ресурсов, но в WWW ссылки обезличены; их смысл должен понимать пользователь. В Паутине же ссылки имеют явно указанный смысл, который доступен для машинной обработки. Таким образом, Семантическая Паутина подразумевает роботизированный поиск информации вместо имеющего место сейчас Web сёрфинга, когда пользователь Интернета переходит по ссылкам от документа к документу.

Семантическая Паутина: принципы и текущее состояние

В основе Семантической Паутины лежит тот же триплет субъект-предикат-объект. Отличие заключается в представлении каждого из элементов триплета. В Паутине субъект, объект и предикат представлены универсальными идентификаторами ресурсов или URI (Uniform Resource Identifier). Так, граф визитной карточки основателя Википедии выглядит следующим образом (рисунок позаимствован из Википедии), рисунок 8.

URI только выглядит как адрес ресурса URL. На самом деле по любому из приведенных на рисунке адресов, таких как http://www.w3.org/2000/10/swap/pim/contact#mailbox, вы не найдете ничего похожего на почтовый ящик, или хотя бы его описание, как это может показаться на первый взгляд. Такая система именования нужна только для обеспечения уникальности имен.

Наиболее популярным языком представления знаний в Паутине является язык RDF. Фрагмент RDF-документа, описывающие отношения Сергей имеет ребенка Никиту, а Никита имеет родителя Сергея, приведен ниже.

```
<Person rdf:ID = "Nikita">
    <HasParent rdf:resource = "#Sergei"/>
</Person>
<owl:Class rdf:ID = "Person"/>
<Person rdf:ID = "Sergei">
    <HasChild rdf:resource = "#Nikita"/>
   </Person>
```

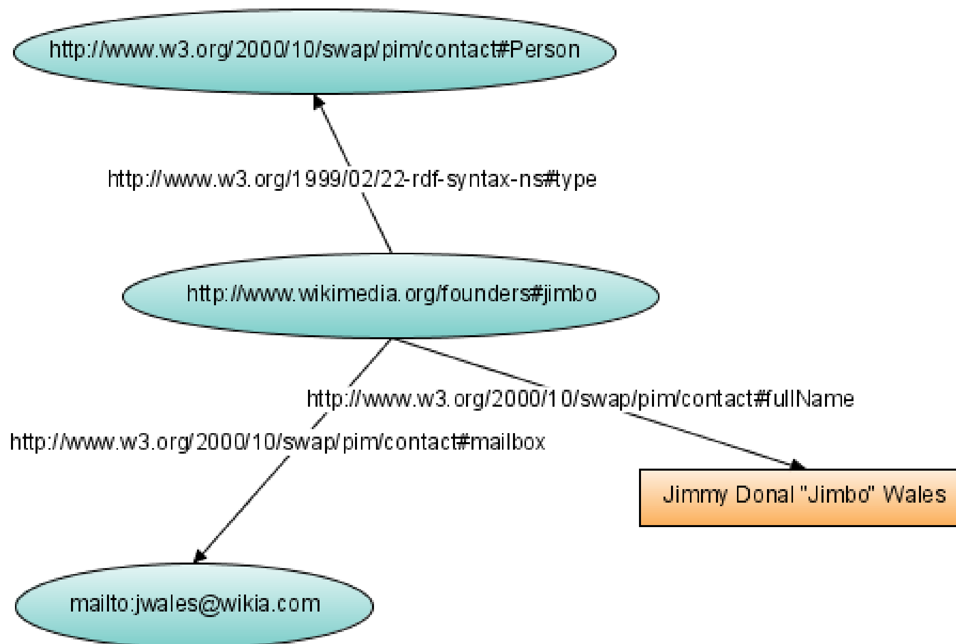


Рис.8. Граф визитной карточки основателя Википедии

RDF-формат документа не предназначен для чтения человеком. Редакторы семантических документов, например, наиболее распространенный редактор онтологий Protégé (http://protege.stanford.edu/), дают возможность вводить и редактировать факты в экранных формах.

В настоящее время создание RDF/OWL документов ведется отдельными энтузиастами. Количество документов в Интернете невелико, и их можно найти с помощью специального поискового сервера, например, SWOOGLE (http://swoogle.umbc.edu/).

Развитие проекта сдерживается по многим причинам. В основном это отсутствие быстрой и прямой выгоды от создания семантических ресурсов, сложность формализации знаний и отсутствие универсальных агентов для извлечения знаний.

Представленные в Интернете реализации интеллектуальных агентов ограничиваются студенческими разработками вроде путеводителя по пивным города Саутхемптон (http://www.twine.com/item/11by40gxk-p1/southampton-pub-guide-in-rdf) и винного агента, советующего, какое вино лучше употребить с каждым из блюд (http://ksl.stanford.edu/people/dlm/webont/wineAgent/).

Контрольные вопросы

1. Напишите правило для отношения двоюродный брат или сестра. Используйте правила, приведенные в подразд.1.4.
2. Имеется программа, состоящая из двух экземпляров правила

```
a: a :- b, c, !, d, fail.
a :- e.
```

Какие предикаты будут вызваны при выполнении правила a, если каждый из предикатов b, c, d, e заканчивается успехом, а предикат fail — стандартный предикат для искусственного вызова неудачи? Что изменится, если убрать предикат отсечения?
3. Имеется предикат

```
dosomething([]).
dosomething([H|T]) :- member(H,T), dosomething(T).
dosomething([H|T]) :- write(H), dosomething(T).
```

Какой результат даст вызов dosomething([1,0, 0,1, 2,0,10])?
4. Какое отсечение, красное или зеленое, стоит в следующем правиле. Обоснуйте.

```
sister(X,Y) :- sibling(X,Y), !, female(X).
```

5. Правомерно ли использование отсечения в следующем правиле? Почему?

```
grandfather(X,Y) :- father(X,Z), !, parent(Z,Y).
```

6. Что такое семантическая сеть?
7. Для решения, каких задач использую метод линейной регрессии?
8. Для решения, каких задач использую метод k-ближайших соседей?
9. Для решения, каких задач использую деревья решений?
10. Для решения, каких задач использую метод логистической регрессии ?

МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ

Линейная регрессия

Линейная регрессия хорошо описана в курсе статистики, здесь ограничимся кратким описанием, которое поможет нам перейти к логистической регрессии, которая для машинного обучения уже более интересна.

Итак, пусть есть некоторые данные, по которым должна быть построена линейная зависимость, рисунок 9. Соответственно, необходимо эту зависимость восстановить в виде параметров функции. К слову, линейная регрессия очень чувствительна к мультиколлинеарности.

Пусть дана обучающая выборка $X^l = (x_i, y_i)_{i=1}^l$, $x_i \in \mathbb{R}^n$, $y_i \in \mathbb{R}$ и тестовая выборка $X^k = (\widetilde{x}_i, \widetilde{y}_i)_{i=1}^k$.

Модель линейной регрессии ($w \in \mathbb{R}^n$) представляет собой параметрическую функцию: $a(x, w) = \sum_{j=1}^n w_j f_j(x)$

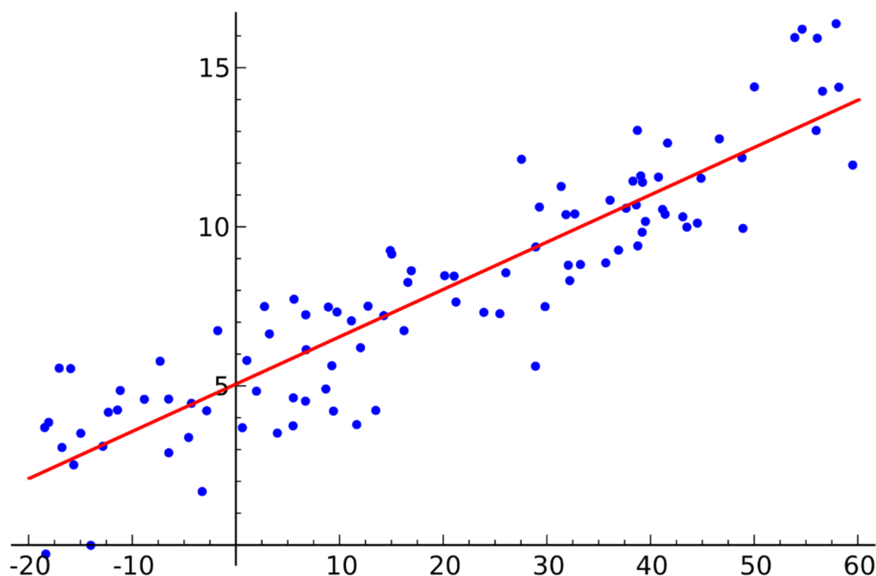


Рис.9. Пример линейной регрессии

Чтобы модель аппроксимировала данные из обучающей выборки наилучшим образом, необходимо подобрать вектор весов W. Вручную это делать достаточно неприятно. Задача подбора значений вектора весов при наличии функционала, определяющего стоимость (он так и называется, функция стоимости или функция потерь), является задачей безусловной оптимизации.

Линейная регрессия – это простая модель, поэтому можно применить метод наименьших квадратов. В качестве функции потерь используется среднеквадратическая ошибка (MSE) $L(a, y) = (a - y)^2$, которую следует минимизировать, что по сути своей является уменьшением ошибки на обучающей выборке:

$$Q(w) = \sum_{i=1}^{l} (a(x_i, w) - y_i)^2 \to \min_w$$

Тогда проверка качества модели на тестовой выборке будет представлять собой сравнение прогноза модели и соответствующего реального значения из тестовой выборки:

$$\bar{Q}(w) = \frac{1}{k} \sum_{i=1}^{k} (a(\tilde{x}_i, w) - \tilde{y}_i)^2.$$

Перейдем к матричным обозначениям:

$$A = \begin{pmatrix} a_1(x_1) & \cdots & a_n(x_1) \\ \vdots & \ddots & \vdots \\ a_1(x_l) & \cdots & a_n(x_l) \end{pmatrix} Y = \begin{pmatrix} y_1 \\ \vdots \\ y_l \end{pmatrix} W = \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}$$

Соответственно, обучение запишется в виде:

$Q(w) = \|AW - Y\|^2 \to \min_w$

Необходимым условием минимума будет равенство градиента по вектору W нулю в точке минимума

$$\frac{\partial Q}{\partial w}(w) = 2A^T(AW - Y) = 0.$$

Получаем систему

$A^T AW = A^T Y.$

Выражаем искомый вектор весов

$W^* = (A^T A)^{-1} A^T Y = A^+ Y$

$Q(W^*) = \|P_A Y - Y\|^2,$

где $P_A = AA^+ = A(A^T A)^{-1} A^T$ – проекционная матрица.

В таком виде можно заметить сходство матричного уравнения поиска вектора весов с сингулярным разложением матрицы (SVD).

Произвольная $l \times n$ матрица A представима в виде **сингулярного разложения**, т.е. произведения трех матриц:

$A = VDU^T.$

Основные свойства:

– $l \times n$ матрица $V = (v_1, \dots, v_n)$ ортогональна, $V^T V = I_n$, v_j - собственные векторы матрицы AA^T;

– $n \times n$ матрица $U = (u_1, \dots, u_n)$ ортогональна, $U^T U = I_n$, u_j - собственные векторы матрицы $A^T A$;

– $n \times n$ матрица $D = diag(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_n})$ диагональна, λ_j - собственные значения матриц $A^T A$ и AA^T.

Применяя сингулярное разложение, получаем решение:

$$A^+ = (UDV^T VDU^T)^{-1} UDV^T = UD^{-1} V^T = \sum_{j=1}^{n} \frac{1}{\sqrt{\lambda_j}} u_j v_j^T$$

$$W^* = A^+ Y = UD^{-1} V^T Y = \sum_{j=1}^{n} \frac{1}{\sqrt{\lambda_j}} u_j \left(v_j^T y\right)$$

$$AW^* = P_A Y = (VDU^T) UD^{-1} V^T Y = VV^T Y = \sum_{j=1}^{n} v_j \left(v_j^T y\right)$$

$$\|W^*\|^2 = \|UD^{-1} V^T Y\|^2 = \|D^{-1} V^T Y\|^2 = \sum_{j=1}^{n} \frac{1}{\lambda_j} \left(v_j^T y\right)^2$$

Метод k-ближайших соседей

Метод k-ближайших соседей (k-nearest neighbors algorithm, kNN) — это метод машинного обучения, который используется для классификации и регрессии. Его основное назначение - предсказание значения целевой переменной на основе значений ближайших к ней соседей в пространстве признаков.

Принцип работы метода заключается в том, что для нового объекта предсказывается значение целевой переменной на основе значений этой переменной y k-ближайших к ней соседей. Для классификации выбирается наиболее часто встречающийся класс среди соседей, а для регрессии – вычисляется среднее или медианное значение целевой переменной среди соседей.

Метод -ближайших соседей представляет собой основной метрический алгоритм классификации, который классифицирует объекты на основе их схожести. Объект отнесён к классу, к которому принадлежат его наиболее близкие соседи в наборе данных для обучения.

Метод -ближайших соседей увеличивает точность классификации за счёт принадлежности объекта к классу, к которому относится большинство из k его ближайших соседей в обучающем наборе данных x_i . В случаях, когда имеется два класса, количество соседей выбирают нечётным числом, чтобы избежать неопределённости, когда равное количество соседей относится к различным классам.

В контексте классификации, когда предъявляется запрос (новый объект, который нужно классифицировать), алгоритм kNN идентифицирует k ближайших обучающих примеров к этому запросу в пространстве признаков. Затем, на основе меток этих соседей, алгоритм присваивает метку новому объекту. Присвоение метки может осуществляться посредством голосования большинством; объекту присваивается метка, которая чаще всего встречается среди его k ближайших соседей.

Выбор параметра k

Выбор числа k является критическим для производительности алгоритма. Слишком маленькое значение k может привести к высокой чувствительности к шуму в данных, в то время как слишком большое значение k может сгладить границы между классами, увеличивая риск неправильной классификации. Оптимальное значение k обычно определяется с помощью кросс-валидации.

Важность метрик расстояния

Метод kNN зависит от метрики расстояния для определения "близости" между объектами. Евклидово расстояние является наиболее распространенной метрикой, но в зависимости от природы данных и задачи могут быть более подходящими другие метрики, такие как манхэттенское, Чебышева или косинусное расстояние.

Преимущества и недостатки

kNN является ленивым алгоритмом, что означает, что он не строит явную модель данных во время обучения, а вместо этого "запоминает" обучающий набор данных. Это делает алгоритм относительно простым в реализации и хорошо подходящим для задач, где взаимосвязь между признаками и целевой переменной сложна или неизвестна. Однако это также означает, что kNN может быть вычислительно затратным, особенно для больших наборов данных с большим количеством признаков, поскольку необходимо вычислить расстояние между запросом и каждым обучающим примером. К тому же алгоритм чувствителен к масштабированию признаков и наличию шумовых признаков.

Добавление этого раздела перед описанием метрик расстояния обеспечит более полное понимание метода k ближайших соседей и его применения в машинном обучении.

Расстояние Минковского

Расстояние Минковского порядка p (где p - целое число) между двумя точками $x = \{x_1, x_2, \dots, x_n\}$ и $y = \{y_1, y_2, \dots, y_n\}$определяется как $d(x, y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}}$. При изменении порядка p в расстоянии Минковского можно получить различные метрики расстояния.

Манхэттенское расстояние

Когда $p = 1$, мы получаем манхэттенское расстояние как $d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$

Евклидово расстояние

Когда $p = 2$, мы имеем евклидово расстояние как $d(x, y) = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}$ Евклидово расстояние является наиболее часто используемой метрикой расстояния и по умолчанию используется в kNN библиотеки sklearn.

Расстояние Чебышева

В предельном случае, когда p стремится к бесконечности, мы получаем расстояние Чебышева: $d(x, y) = \max_i |x_i - y_i|$

Косинусное расстояние

Косинусное сходство измеряет сходство между двумя векторами внутреннего произведения. Оно измеряется косинусом угла между двумя векторами и определяет, указывают ли два вектора примерно в одном направлении.

$$\cos(x, y) = \frac{x^T y}{\|x\| \|y\|}$$

Результирующее сходство варьируется от -1 (точно противоположные) до 1 (точно такие же), с 0, указывающим на ортогональность или декорреляцию. Следовательно, косинусное расстояние определяется как $d(x, y) = 1 - \cos(x, y)$. Когда p и y нормализованы до единичной длины, $\|x\| = \|y\| = 1$, связь между евклидовым расстоянием и косинусным сходством выражается как $d_{euc}(x, y) = \sqrt{2 - 2\cos{(x, y)}}$.

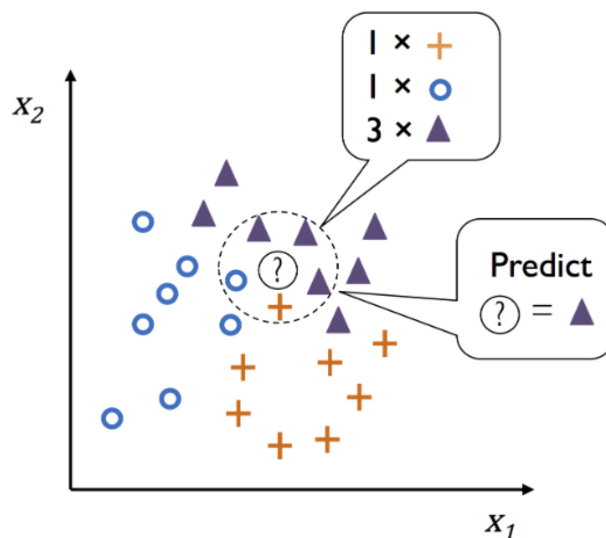


Рис.10. kNN для задачи с тремя классами при k = 5

Деревья решений

На рисунке 11 изображена задача «ирисы Фишера», предполагающая классификацию на три класса, линейными классификаторами ее решить невозможно, нелинейные будут избыточны из-за малого количества признаков. Но с этой задачей справятся деревья решений (decision trees), которые последовательно применяют решающие правила (предикаты).

Дерево решений – способ представления правил в иерархической, последовательной структуре, где каждому объекту соответствует единственный узел, дающий решение. В результате формируется покрывающий набор конъюнкций.

Каждый узел дерева содержит признак, ребра – значения признака, листья – метки классов, пример такого дерева показан на рисунке 12. Классы должны быть дискретными. Каждый пример должен однозначно относиться к одному из классов.

С4.5 [45] – алгоритм построения дерева решений, количество потомков у узла не ограничено. Решает только задачи классификации. В нем есть важное требование: количество классов должно быть значительно меньше количества записей в исследуемом наборе данных.

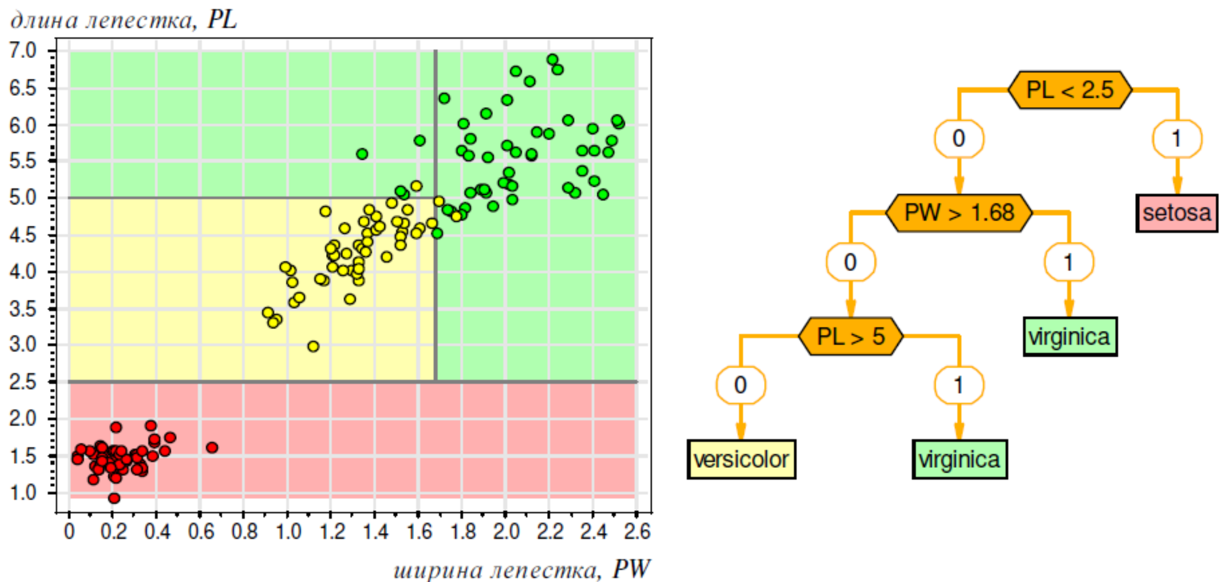


Рис. 11. Задача «ирисы Фишера»

Пусть T – множество примеров, где каждый элемент описывается m атрибутами, C_j - метка класса. Процесс построения дерева будет происходить итеративно сверху вниз.

На первом шаге есть пустое дерево (имеется только корень) и исходное множество T (ассоциированное с корнем). Требуется разбить исходное множество на подмножества. Делается через выбор одного из атрибутов в качестве проверки.

Тогда в результате разбиения получаются n (по числу значений атрибута) подмножеств и, соответственно, создаются n потомков корня, каждому из которых поставлено в соответствие свое подмножество, полученное при разбиении множества T.

В процессе построения любого дерева решений необходимо выбрать критерий разбиения (в случае C4.5 это прирост информации) и правило остановки (при небольшой выборке дерево строят до ее исчерпания, при большой – применяют отсечение).

Отсечение ветвей (pruning) – эвристический метод. Идём от листьев к корню, помечая по некоторому критерию, например, качество классификации, узлы на удаление. Вместо узлов ставится лист с меткой класса с наибольшим количеством исходов в этом поддереве.

Критерий разбиения

Пусть мы имеем проверку X (в качестве проверки может быть выбран любой атрибут), которая принимает n значений $A_1, A_2, \dots, A_n$. Тогда разбиение T по проверке X даст нам подмножества $T_1, T_2, \dots, T_n$, при X равном соответственно $A_1, A_2, \dots, A_n$. $freq(C_j, T)$ – количество примеров из множества T, относящихся к классу C_j.

Оценка среднего количества информации, необходимого для определения класса примера из множества T (энтропия):

$$Info(T) = -\sum_{j=1}^{k} \frac{freq(C_j, T)}{|T|} * \log_2\left(\frac{freq(C_j, T)}{|T|}\right)$$

Оценка среднего количества информации, необходимого для определения класса примера из множества T после разбиения множества T по X (условная энтропия):

$$Info_X(T) = \sum_{i=1}^{n} \left(\frac{|T_i|}{|T|} * Info(T_i)\right)$$

Оценка потенциальной информации, получаемой при разбиении множества T на n подмножеств, необходим для учета атрибутов с уникальными значениями.

$$split_{info(X)} = -\sum_{i=1}^{n} \left(\frac{|T_i|}{|T|} * \log_2\left(\frac{|T_i|}{|T|}\right)\right)$$

Нормированный прирост информации

$$Gain_ratio(X) = \frac{Info(T) - Info_X(T)}{split_{info(X)}}$$

Критерий $Gain_ratio$ считается для всех атрибутов. Выбирается атрибут с максимальным $Gain_ratio$. Этот атрибут будет являться проверкой в текущем узле дерева, а затем по этому атрибуту производится дальнейшее построение дерева.

Такие же рассуждения можно применить к полученным подмножествам $T_1, T_2, \dots, T_n$ и продолжить рекурсивно процесс построения дерева до тех пор, пока в узле не окажутся примеры из одного класса.

Для примера возьмем таблицу 2 из [4].

Таблица 2. Пример данных для построения дерева решений

№	Признаки				Класс
	Outlook	Temperature	Humidity	Windy	
1	Sunny	Hot	High	False	N
2	Sunny	Hot	High	True	N
3	Overcast	Hot	High	False	P
4	Rain	Mild	High	False	P
5	Rain	Cool	Normal	False	P
6	Rain	Cool	Normal	True	N
7	Overcast	Cool	Normal	True	P
8	Sunny	Mild	High	False	N
9	Sunny	Cool	Normal	False	P
10	Rain	Mild	Normal	False	P
11	Sunny	Mild	Normal	True	P
12	Overcast	Mild	High	True	P
13	Overcast	Hot	Normal	False	P
14	Rain	Mild	High	True	N

Посчитаем критерий разбиения для всех признаков, чтобы определить какой признак будет помещен в корень дерева. В начале, у нас полный набор данных, поэтому энтропия:

$$Info(T) = -\left(\frac{5}{14} * \log_2\left(\frac{5}{14}\right) + \frac{9}{14} * \log_2\left(\frac{9}{14}\right)\right)$$

Для признака Outlook имеем 3 значения, каждое из которых дает свое подмножество T_i:

$$Info_X(T) = \left(\frac{5}{14} * -\left(\frac{3}{5} * \log_2\left(\frac{3}{5}\right) + \frac{2}{5} * \log_2\left(\frac{2}{5}\right)\right)\right) + \left(\frac{4}{14} * -\left(\frac{4}{4} * \log_2\left(\frac{4}{4}\right) + \frac{0}{4} * \log_2\left(\frac{0}{4}\right)\right)\right) + \left(\frac{5}{14} * -\left(\frac{3}{5} * \log_2\left(\frac{3}{5}\right) + \frac{2}{5} * \log_2\left(\frac{2}{5}\right)\right)\right)$$

Точно также считается для оставшихся признаков. Затем выбираем признак с максимальным нормированным приростом информации и помещаем его в корень. Следующим шагом заполняем узлы по значениям признака в корне, но в этот раз множество T будет состоять не из 14 строк, а из такого количества строк, где признак в корне принял то или иное значение.

Логистическая регрессия

В модели линейной регрессии $X\beta + \varepsilon$ существует два типа переменных – объясняющие переменные $X_1, X_2, \dots, X_k$ и переменные исследования y. Эти переменные могут быть измерены как на непрерывной шкале, так и в виде индикаторных переменных. Когда объясняющие переменные являются качественными, их значения выражаются в виде индикаторных переменных, и затем используются модели с фиктивными переменными.

Когда переменная исследования является качественной, её значения могут быть выражены с помощью индикаторной переменной, принимающей только два возможных значения 0 или 1. В таком случае используется логистическая регрессия. Например, переменная может обозначать значения типа успех или неудача, да или нет, нравится или не нравится, которые могут быть представлены двумя значениями 0 и 1.

Рассмотрим модель

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_k x_{ik} + \varepsilon_i = x_i' \beta + \varepsilon_i, i = 1,2, \dots, n$$

где $x_i' = [1, x_{i1}, x_{i2}, \dots, x_{ik}], \beta' = [\beta_0, \beta_1, \beta_2, \dots, \beta_k]$.

Переменная исследования принимает два значения: $y_i = 0$ или $y_i = 1$. Предположим, что y_i следует распределению Бернулли с параметром π_i , поэтому её вероятностное распределение записывается как:

$$y_i = \begin{cases} 1 & \text{with } P(y_i = 1) = \pi_i \\ 0 & \text{with } P(y_i = 0) = 1 - \pi_i. \end{cases}$$

Полагая $E(\varepsilon_i) = 0$,

$$E(y_i) = 1.\pi_i + 0.(1-\pi_i) = \pi_i.$$

Из модели $y_i = x_i'\beta + \varepsilon_i$, получим

$$E(y_i) = x_i'\beta$$

$$\Rightarrow E(y_i) = x_i'\beta = \pi_i$$

$$\Rightarrow E(y_i) = P(y_i = 1).$$

Таким образом, функция отклика $E(y_i)$ просто представляет собой вероятность того, что $y_i = 1$. Заметим, что $\varepsilon_i = y_i - x_i'\beta$, таким образом,

- когда $y_i = 1$, тогда $\varepsilon_i = 1 - x_i'\beta$
- $y_i = 0$, тогда $\varepsilon_i = -x_i'\beta$.

Вспомним, что ранее предполагалось, что ε_i следует нормальному распределению, когда y не является индикаторной переменной. Когда y является индикаторной переменной, то ε_i принимает только два значения, поэтому нельзя предполагать, что она следует нормальному распределению.

В обычной модели регрессии ошибки гомоскедастичны, то есть $Var(\varepsilon_i) = \sigma^2$, и таким образом $Var(y_i) = \sigma^2$. Когда y является индикаторной переменной, то есть:

$$\begin{aligned} Var(y_i) &= E\left[y_i - E(y_i)\right]^2 \\ &= (1-\pi_i)^2\pi_i + (0-\pi_i)^2(1-\pi_i) \\ &= \pi_i(1-\pi_i)\left[1-\pi_i+\pi_i\right] \\ &= \pi_i(1-\pi_i) \\ &= E(y_i)\left[1-E(y_i)\right] \\ &= \sigma_{y_i}^2 \ . \end{aligned}$$

Таким образом, $Var(y_i)$ зависит от y_i и является функцией среднего значения y_i. Более того, $E(y_i) = \pi_i$, а π_i является вероятностью, то есть $0 \leq \pi_i \leq 1$, и, следовательно, существует ограничение на $E(y_i)$, которое состоит в том, $0 \leq E(y_i) \leq 1$. Это накладывает значительное ограничение на выбор линейной функции отклика. Нельзя подобрать модель, в которой прогнозируемые значения выходят за пределы интервала от 0 до 1.

Когда y является дихотомической переменной, эмпирические доказательства показывают, что функция $E(y)$, определенная на всей вещественной прямой, которая может быть отображена в интервал [0,1], имеет форму сигмоиды. Это нелинейная S-образная форма, показана на рисунке 12. Естественным выбором для $E(y)$ была бы кумулятивная функция распределения случайной величины. В частности, логистическое распределение, кумулятивная функция распределения которого является упрощенной логистической функцией, определяется следующим образом:

$$E(y) = \frac{\exp(y)}{1+\exp(y)} = \frac{\exp(x'\beta)}{1+\exp(x'\beta)} = \frac{1}{1+\exp(-x'\beta)} .$$

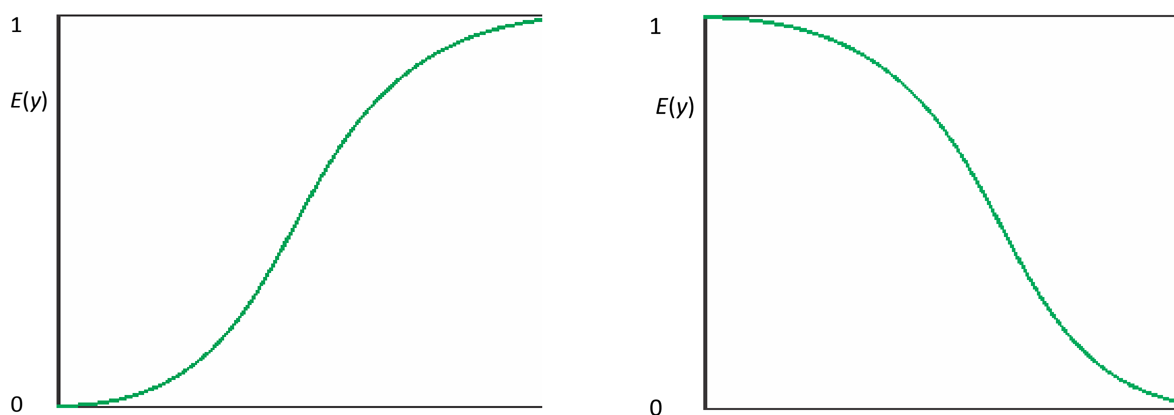


Рис. 12.Сигмоида

Линейный предиктор и функции связи

Систематическая компонента в $E(y)$ – это линейный предиктор и обозначается как

$$\eta_i = \sum_j \beta_j x_{ij} = x_i^{'}\beta,\ i = 1, 2, ..., n,\ j = 0, 1, 2, ..., k.$$

Функция связи в обобщенной линейной модели устанавливает связь между линейным предиктором η_i и средним откликом μ_i.

Таким образом,

$$g(\mu_i) = \eta_i \text{ или } \mu_i = g^{-1}(\eta_i).$$

В обычных линейных моделях, основанных на нормально распределенной исследуемой переменной, используется связь $g(\mu_i) = \mu_i$, которая называется идентичной связью. Функция связи отображает область значений μ_i на всю вещественную прямую, обеспечивает хорошее эмпирическое приближение и имеет осмысленную интерпретацию в реальных приложениях.

В случае логистической регрессии функция связи определяется как

$$\eta = \ln\frac{\pi}{1-\pi}.$$

Это преобразование называется **logit-преобразованием** вероятности π, а $\frac{\pi}{1-\pi}$ называется **шансами** (odds). Функция связи η также называется **логарифмом шансов** (log-odds). Эта функция связи получается следующим образом:

$$\pi = \frac{1}{1+\exp(-\eta)}$$

или $\pi[1+\exp(-\eta)] = 1$

или $e^{-\eta} = \dfrac{1-\pi}{\pi}$

или $\mu = \ln\dfrac{\pi}{1-\pi}$

Примечание: подобно logit-функции, существуют и другие функции, которые имеют такую же форму, как логистическая функция. Эти функции также могут быть преобразованы через π . Существует две такие популярные функции – пробит-преобразование и преобразование комплементарного логарифма-логарифма. Пробит-преобразование основано на преобразовании π с использованием кумулятивной функции распределения нормального распределения, и на основе этого строится **пробит-регрессионная модель**.

Комплементарное логарифмическое преобразование π представляет собой следующую функцию: $\ln[-\ln(1-\pi)]$.

Оценка максимального правдоподобия параметров

Рассмотрим общую форму модели логистической регрессии

$$y_i = E(y_i) + \varepsilon_i$$

где y_i – независимые случайные переменные Бернулли с параметром π_i с

$$E(y_i) = \pi_i = \frac{exp\,(x_i'\beta)}{1+exp\,(x_i^i\beta)}.$$

Функция плотности вероятности y_i равна

$$f_i(y_i) = \pi_i^{y_i}(1-\pi_i)^{1-y_i}, i = 1, 2, ..., n, \; y_i = 0 \text{ or } 1.$$

Функция правдоподобия (likelyhood function) записывается как

$$L(y_1, y_2, ..., y_n, \beta_1, \beta_2, ..., \beta_k) = L = \prod_{i=1}^{n} f_i(y_i)$$

$$= \prod_{i=1}^{n} f_i(y_i)(1-\pi_i)^{1-y_i}$$

$$\ln L = \sum_{i=1}^{n} \left[\ln \pi_i^{y_i} + \ln(1-\pi_i)^{1-y_i} \right]$$

$$= \sum_{i=1}^{n} \left[y_i \ln \pi_i + (1-y_i)\ln(1-\pi_i) \right]$$

$$= \sum_{i=1}^{n} \left[y_i \ln\left(\frac{\pi_i}{1-\pi_i} \right) \right] + \sum_{i=1}^{n} \left[\ln(1-\pi_i) \right].$$

Начиная с

$$\pi_i = \frac{\exp(x_i'\beta)}{1+\exp(x_i'\beta)},$$

$$1-\pi_i = \frac{1}{1+\exp(x_i'\beta)},$$

$$\frac{\pi_i}{1-\pi_i} = \exp(x_i'\beta),$$

$$\ln \frac{\pi_i}{1-\pi_i} = \exp x_i'\beta \quad ,$$

Так что

$$\ln L = \sum_{i=1}^{n} y_i x_i'\beta - \sum_{i=1}^{n} \ln\left[1+\exp(x_i'\beta) \right]$$

Предположим, что доступны повторные наблюдения на каждом уровне

переменных x. Пусть y_i – это количество единиц, наблюдаемых для i-го наблюдения, и пусть n_i – это количество испытаний для каждого наблюдения. Тогда

$$\ln L = \sum_{i=1}^{n} y_i \pi_i + \sum_{i=1}^{n} n_i \ln(1-\pi_i) - \sum_{i=1}^{n} y_i \ln(1-\pi_i)$$

Оценка максимального правдоподобия β для вектора параметров β получается численной максимизацией.

Если $V(\varepsilon) = \Omega$, тогда асимптотически

$$E(\hat{\beta}) = \beta$$

$$V(\hat{\beta}) = (X'\Omega^{-1}X)^{-1} .$$

После получения β линейный предиктор оценивается с помощью

$$\hat{\eta}_i = x_i'\beta .$$

Значение, подобранное моделью

$$\hat{y}_i = \hat{\pi}_i = \frac{exp\ (\hat{\eta}_i)}{1+exp\ (\hat{\eta}_i)} = \frac{1}{1+exp\ (-\hat{\eta}_i)} = \frac{1}{1+exp\ (-x_i'\hat{\beta}\ )}$$

Интерпретация параметров

Для понимания интерпретации связанных параметров β' в модели логистической регрессии сначала рассмотрим простой случай с только одной переменной

$$\eta(x) = \beta_0 + \beta_1 x.$$

После подгонки модели получены оценки $\hat{\beta}_0$ и $\hat{\beta}_1$для параметров β_0 и β_1 соответственно. Тогда подогнанный линейный предиктор при $x = x_i$ равен:

$$\hat{\eta}(x_i) = \hat{\beta}_0 + \hat{\beta}_1 x_i$$

Это логарифм шансов при $x = x_i$ Значение, подогнанное моделью, при $x = 1 + x_i$ равно:

$$\hat{\eta}(x_i + 1) = \hat{\beta}_0 + \hat{\beta}_1 (x_i + 1)$$

Это логарифм шансов при $x = 1 + x_i$

Таким образом,

$$\hat{\beta}_1 = \hat{\eta}(x_i + 1) - \hat{\eta}(x_i) = ln\,[odds(x_i + 1)] - ln\,[odds(x_i)$$

$$= ln\left[\frac{odds(x_i + 1)}{odds(x_i)}\right] => \frac{odds(x_i + 1)}{odds(x_i)} = exp\ (\hat{\beta}_1)$$

Это называется **отношением шансов**, которое представляет собой оценочное увеличение вероятности успеха при изменении значения объясняющей переменной на одну единицу.

Когда в модели есть более одной объясняющей переменной, интерпретация параметров β_j аналогична случаю одной объясняющей переменной. Отношение шансов – $\exp(\beta_j)$ связано с объясняющей переменной x_j, при этом другие объясняющие переменные остаются постоянными. Это аналогично интерпретации параметра β_j в модели множественной линейной регрессии.

Если в объясняющей переменной происходит изменение на m единиц,

то оценочное увеличение в отношении шансов составляет exp $(m\beta_j)$.

Тест гипотезы

Тест гипотезы для параметров в модели логистической регрессии основан на асимптотической теории. Это тест большой выборки, основанный на отношении правдоподобия и использующий статистику, называемую **отклонением** (deviance).

Модель с точно p параметрами, которая идеально подходит для выборки данных, называется **насыщенной моделью** (saturated model).

Статистика, сравнивающая логарифмические правдоподобия подогнанных и насыщенных моделей, называется **отклонением модели** (model deviance). Она определяется как

$$\lambda(\beta) = 2lnL(\text{saturated model}) - 2lnL(\hat{\beta}),$$

где $\ln L(\cdot)$ – это логарифмическое правдоподобие, а $\hat{\beta}$ – оценка максимального правдоподобия для β.

В случае модели логистической регрессии y_i может принимать значения 0 или 1, и π_i полностью не ограничены. Поэтому правдоподобие будет максимальным при $\pi_i = y_i$, и максимальное значение L для насыщенной модели составляет:

$$Maximum\ L(\text{saturated model}) = 1$$

$$\Rightarrow \ln Maximum\, L(\text{saturated model}) = 0.$$

Пусть $\hat{\beta}$ – оценка максимального правдоподобия для β. Тогда логарифмическое правдоподобие максимально при $\beta = \hat{\beta}$, и

$$lnL(\hat{\beta}) = \sum_{i-1}^{n} y_i x_i \hat{\beta}_i - \sum_{i=1}^{n} ln[1 + exp\ (x_i'\beta)] \geq lnL(saturated\ model).$$

Предполагая, что функция логистической регрессии верна, большая выборочная дисперсия статистики теста отношения правдоподобия $\lambda(\beta)$ распределена приблизительно как $\chi^2(n-p)$, когда n большое.

Большое значение $\lambda(\beta)$ указывает на неправильность модели. Малое значение $\lambda(\beta)$ означает, что модель хорошо подходит и примерно равна насыщенной модели. Следует отметить, что обычно у подогнанной модели будет меньше параметров, чем у насыщенной модели, основанной на всех параметрах.

Лабораторные работы

Модуль 1. БАЗЫ ЗНАНИЙ И ОНТОЛОГИИ

В модуле три лабораторных работы. По каждой лабораторной необходимо сделать краткий отчёт с результатами выполнения лабораторной работы. По завершению практического модуля курса – расширенный отчёт.

Лабораторная работа 1. Создание базы знаний и выполнение запросов в Prolog

Цель работы: изучить базы знаний, развить навыки работы с фактами, предикатами и правилами в логическом программировании.

Задание

Создать базу знаний в языке программирования Prolog и реализовать набор запросов, используя эту базу знаний.

Создание базы знаний:

База знаний должна включать в себя **не менее 20 фактов с одним аргументом, 10-15 фактов с двумя аргументам, которые дополняют и показывают связь с другими фактами, и 5-7 правил.** Факты могут описывать объекты, их свойства и отношения между ними. Факты с двумя и более аргументами могут описывать различные атрибуты объектов, а правила - логические законы и выводы, которые можно сделать на основе созданных фактов.

Выполнение запросов:

Напишите несколько запросов для БЗ. Запросы **должны быть разной сложности** и включать в себя:

Простые запросы к базе знаний для поиска фактов.

Запросы, использующие логические операторы (**и, или, не**) для формулирования сложных условий (или использовать логические операторы в правилах).

Запросы, использующие переменные для поиска объектов с определенными характеристиками.

Запросы, которые требуют выполнения правил для получения результата.

Документация:

В коде должны быть комментарии описания фактов, предикатов и правил.

Критерии оценки:

Корректность базы знаний и выполненных запросов.

Сложность и разнообразие запросов.

Качество документации и комментариев к коду.

Тематика для базы знаний:

Любая, связанная с играми. Например:
видеоигры,
правила настольных игр,
профили игроков,
игровые персонажи,
история игры,
игровые механики.

Лабораторная работа 2. Создание онтологии в Protege

Цель работы: изучить онтологии, познакомиться со средой разработки онтологий Protege и преобразованием базы знаний, созданной в первой лабораторной работе, в онтологическую форму в Protege.

Задание
Преобразовать факты и отношения из Prolog в концепты и свойства онтологии. Описать классы и свойства в онтологии, которые соответствуют объектам и отношениям из базы знаний. Например, если были классы "Человек" и "Машина" и свойство "возраст", создайте аналогичные классы и свойства в онтологии в Protege.
Критерии оценки:
Корректное создание онтологии в Protege на основе базы знаний в Prolog.
Качество перевода фактов, предикатов и отношений из Prolog в онтологию.
Определение классов, свойств и иерархии классов в Protege.
Корректность работы правил в SWRL
Тестирование онтологии и демонстрация ее функциональности (визуализация и проверка запросов).

Лабораторная работа 3. Разработка системы поддержки принятия решения на основе базы знаний или онтологии

Цель работы: разработка программы (рекомендательной системы), которая будет использовать базу знаний или онтологию для предоставления данных приложению. Приложение должно давать рекомендации на основе введенных пользователем данных. (Knowledge-based support system)

Задание

Создать программу, которая позволяет пользователю ввести запрос через командную строку или интерфейс пользователя и получить рекомендацию.

Например, пользователь вводит информацию о себе, своих интересах и предпочтениях в контексте выбора видеоигры.

Использовать введенные пользователем данные, описать логический запрос и обратиться к фактам из базы знаний, созданной в первой

лабораторной работе или онтологии из второй.

На основе полученных результатов запросов система должна предоставить рекомендации или советы, связанные с введёнными пользователем данными и существующими актами в БЗ или онтологии.
Приложение должно возвращать рекомендации не сразу, а после краткого диалога с пользователем.

Пример

Входная строка:
Мне 18 лет, мне нравятся: RPG и инди-игры

Реализация

Проанализировать введённую строку, разбить на факты, построить запрос, используя полученные предикаты. (Формат входной строки фиксированный, искать частичное соответствие подстроки не нужно)

Критерии оценки:

Корректность и эффективность реализации системы поддержки принятия решения.

Способность программы адекватно использовать базу знаний или онтологию для выдачи рекомендаций.

Качество тестирования и обработки строки, введённой пользователем.
Качество документации и описание работы системы.

Модуль 2. МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ

В модуле четыре лабораторные работы. Реализовывать можно на любом языке программирования. По каждой лабораторной необходимо сделать краткий отчёт с результатами выполнения лабораторной работы. По завершению практического модуля курса – расширенный отчёт.

Лабораторная работа 4. Линейная регрессия

Задание

Выбор набора данных (dataset):

Студенты с **четным** порядковым номером в группе используют набор данных о жилье в Калифорнии Скачать можно тут

Студенты с **нечетным** порядковым номером в группе используют данные про обучение студентов

Получите и визуализируйте (графически) статистику по набору данных, включая количество, среднее значение, стандартное отклонение, минимум, максимум и различные квантили.

Проведите предварительную обработку данных, включая обработку отсутствующих значений, кодирование категориальных признаков и нормализацию данных.

Разделите данные на обучающий и тестовый наборы данных.

Реализуйте линейную регрессию с использованием метода наименьших квадратов без использования сторонних библиотек, кроме NumPy и Pandas (для расчёта коэффициентов использовать библиотеки тоже нельзя). Использовать минимизацию суммы квадратов разностей между фактическими и предсказанными значениями для нахождения оптимальных коэффициентов.

Постройте **три модели** с различными наборами признаков.

Для каждой модели проведите оценку производительности, используя метрику коэффициент детерминации, чтобы измерить, насколько хорошо модель соответствует данным.

Сравните результаты трех моделей и сделайте выводы о том, какие признаки работают лучше всего для каждой модели.

Дополнительное задание

Ввести синтетический признак при построении модели.

Лабораторная работа 5. Метод k-ближайших соседей

Задание

Выбор набора данных (dataset):

Студенты с **чётным** порядковым номером в группе используют набор данных о напитках .

Студенты с **нечётным** порядковым номером в группе используют данные про диабет

Проведите предварительную обработку данных, включая обработку отсутствующих значений, кодирование категориальных признаков и масштабирование.

Получите и визуализируйте (графически) статистику по набору данных, включая количество, среднее значение, стандартное отклонение, минимум, максимум и квантили, постройте 3d-визуализацию признаков.

Реализуйте метод k-ближайших соседей без использования сторонних библиотек, кроме NumPy и Pandas.

Постройте две модели kNN с различными наборами признаков:

Модель 1: Признаки выбираются случайным образом.

Модель 2: Фиксированный набор признаков, который выбирается студентом заранее.

Для каждой модели проведите оценку на тестовом наборе данных при разных значениях k. Выберите несколько различных значений k, например, k=3, k=5, k=10, и т. д. Постройте матрицу ошибок.

Лабораторная работа 6. Деревья решений

Задание

Выбор набора данных (dataset):

Для студентов с чётным порядковым номером в группе – датасет с классификацией грибов, а нечётным – датасет с данными про оценки студентов инженерного и педагогического факультетов (для данного датасета нужно ввести метрику: студент успешный/неуспешный на основании оценок).

Отобрать **случайным** образом sqrt(n) признаков

Реализовать без использования сторонних библиотек построение дерева решений (дерево не бинарное, numpy и pandas использовать можно, использовать список списков для реализации дерева - нельзя) для решения задачи бинарной классификации

Провести оценку реализованного алгоритма с использованием Accuracy, precision и recall

Построить кривые AUC-ROC и AUC-PR (в пунктах 4 и 5 использовать библиотеки нельзя).

Лабораторная работа 7. Логистическая регрессия

Выбор набора данных (dataset):
Набор данных о пассажирах Титаника: Titanic Dataset
Набор данных о диабете: Diabetes Dataset

Загрузите выбранный набор данных и выполните предварительную обработку данных.

Получите и визуализируйте (графически) статистику по датасету (включая количество, среднее значение, стандартное отклонение, минимум, максимум и различные квантили).

Разделите данные на обучающий и тестовый наборы в соотношении, которое вы считаете подходящим.

Реализуйте логистическую регрессию "с нуля" без использования сторонних библиотек, кроме NumPy и Pandas. Ваша реализация логистической регрессии должна включать в себя:

Функцию для вычисления гипотезы (sigmoid function).

Функцию для вычисления функции потерь (log loss).

Метод обучения, который включает в себя градиентный спуск.

Возможность варьировать гиперпараметры, такие как коэффициент обучения (learning rate) и количество итераций.

Исследование гиперпараметров. Проведите исследование влияния гиперпараметров на производительность модели. Варьируйте следующие гиперпараметры:

Коэффициент обучения (learning rate).

Количество итераций обучения.

Метод оптимизации (например, градиентный спуск или оптимизация Ньютона).

Оценка модели. Для каждой комбинации гиперпараметров оцените производительность модели на тестовом наборе данных, используя метрики, такие как accuracy, precision, recall и F1-Score.

Сделайте выводы о том, какие значения гиперпараметров наилучшим образом работают для исследуемого набора данных и задачи классификации. Обратите внимание на изменение производительности модели при варьировании гиперпараметров.

ПРИЛОЖЕНИЯ

Приложение 1. Пример реализации онтологии в Protege

Отобразим факты базы знаний на предикаты онтологии (ObjectProperties/DataProperties):

Annotation properties | Datatypes | Individuals
Classes | Object properties | Data properties

Object property hierarchy: beDesc

Asserted

- owl:topObjectProperty
  - beAncestorOf
    - parent
  - beDescendantOf
    - childOf
  - spouse

beDescendantOf — http://www.semanticweb.org/indianmax/ontologies/2

Annotations | Usage

Annotations: beDescendantOf

Annotations

Characteristics: beDescendant

- [ ] Functional
- [ ] Inverse functional
- [x] Transitive
- [ ] Symmetric
- [ ] Asymmetric
- [ ] Reflexive
- [ ] Irreflexive

Description: beDescendantOf

Equivalent To

SubProperty Of

Inverse Of
beAncestorOf

Domains (intersection)
owl:Thing

Ranges (intersection)
owl:Thing

Disjoint With

SuperProperty Of (Chain)

Annotation properties | Datatypes | Individuals
Classes | Object properties | Data properties

Object property hierarchy: childOf

Asserted

- owl:topObjectProperty
  - beAncestorOf
    - parent
  - beDescendantOf
    - childOf
  - spouse

childOf — http://www.semanticweb.org/indianmax/ontologies/202

Annotations | Usage

Annotations: childOf

Annotations

Characteristics: childOf

- [ ] Functional
- [ ] Inverse functional
- [ ] Transitive
- [ ] Symmetric
- [x] Asymmetric
- [ ] Reflexive
- [x] Irreflexive

Description: childOf

Equivalent To

SubProperty Of
beDescendantOf

Inverse Of
parent

Domains (intersection)
owl:Thing

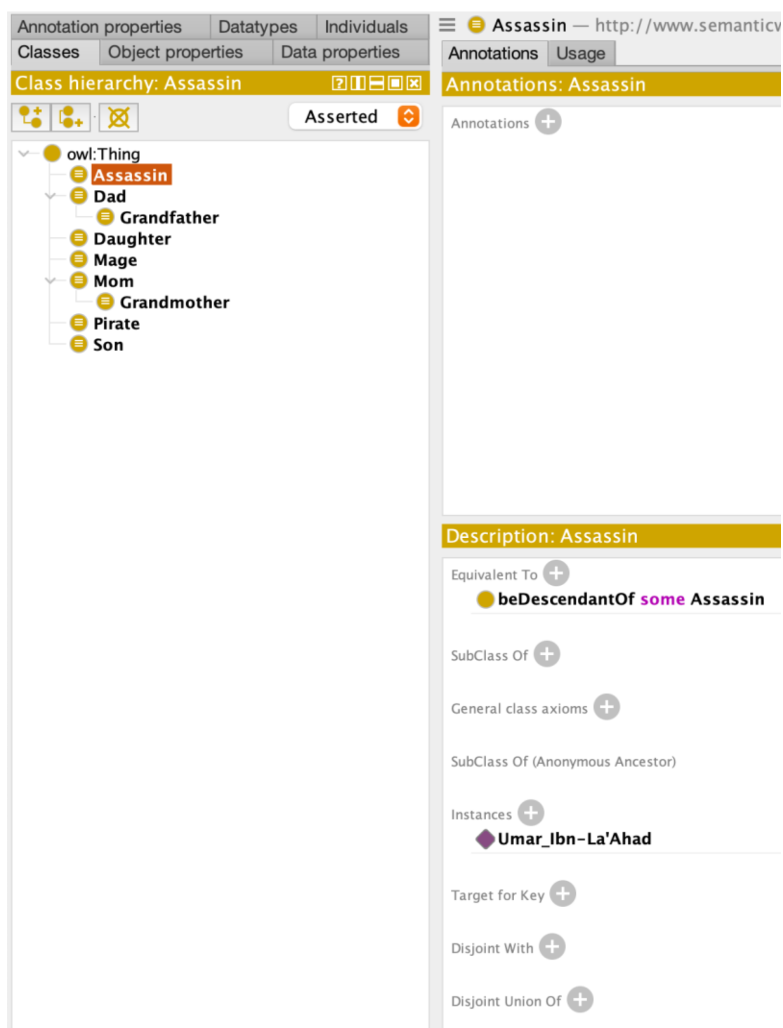
Ranges (intersection)
owl:Thing

Disjoint With

SuperProperty Of (Chain)

На основании предикатов определим Individuals:

В связи с особенностями строения онтологии определим правила базы знаний как классы, используя определенные ранее предикаты. Помимо этого, для отображения рекурсии предков, определим причастие (ярлык) также через класс:



Annotation properties | Datatypes | Individuals
Classes | Object properties | Data properties

Class hierarchy: Dad

Asserted

- owl:Thing
  - Assassin
  - Dad
    - Grandfather
  - Daughter
  - Mage
  - Mom
    - Grandmother
  - Pirate
  - Son

Dad — http://www.semant

Annotations | Usage

Annotations: Dad

Annotations

Description: Dad

Equivalent To
(parent some owl:Thing)
and (man value true)

SubClass Of

General class axioms

SubClass Of (Anonymous Ancestor)

Instances

Target for Key

Disjoint With
Mom

Disjoint Union Of

Annotation properties | Datatypes | Individuals
Classes | Object properties | Data properties

Class hierarchy: Grandfather

Asserted

- owl:Thing
  - Assassin
  - Dad
    - Grandfather
  - Daughter
  - Mage
  - Mom
    - Grandmother
  - Pirate
  - Son

Grandfather — http://www.semanticweb

Annotations | Usage

Annotations: Grandfather

Annotations

Description: Grandfather

Equivalent To
(parent some (parent some owl:Thing))
and (man value true)

SubClass Of
Dad

General class axioms

SubClass Of (Anonymous Ancestor)
(parent some owl:Thing)
and (man value true)

Instances

Target for Key

Disjoint With
Grandmother

Disjoint Union Of

Annotation properties | Datatypes | Individuals
Classes | Object properties | Data properties

Class hierarchy: Daughter

Asserted

- owl:Thing
  - Assassin
  - Dad
    - Grandfather
  - Daughter
  - Mage
  - Mom
    - Grandmother
  - Pirate
  - Son

Daughter — http://www.se

Annotations | Usage

Annotations: Daughter

Annotations

Description: Daughter

Equivalent To

(childOf some owl:Thing) and (woman value true)

SubClass Of

General class axioms

SubClass Of (Anonymous Ancestor)

Instances

Target for Key

Disjoint With

Son

Disjoint Union Of

Annotation properties | Datatypes | Individuals
Classes | Object properties | Data properties

Class hierarchy: Mage

Asserted

- owl:Thing
  - Assassin
  - Dad
    - Grandfather
  - Daughter
  - Mage
  - Mom
    - Grandmother
  - Pirate
  - Son

Mage — http://www.semanticv

Annotations | Usage

Annotations: Mage

Annotations

Description: Mage

Equivalent To

beDescendantOf some Mage

SubClass Of

General class axioms

SubClass Of (Anonymous Ancestor)

Instances

Dominico_Auditore

Target for Key

Disjoint With

Disjoint Union Of

Annotation properties | Datatypes | Individuals
Classes | Object properties | Data properties

Class hierarchy: Mom

Asserted

- owl:Thing
  - Assassin
  - Dad
    - Grandfather
  - Daughter
  - Mage
  - Mom
    - Grandmother
  - Pirate
  - Son

Mom — http://www.sema

Annotations | Usage

Annotations: Mom

Annotations

Description: Mom

Equivalent To
(parent some owl:Thing) and (woman value true)

SubClass Of

General class axioms

SubClass Of (Anonymous Ancestor)

Instances

Target for Key

Disjoint With
Dad

Disjoint Union Of

Annotation properties | Datatypes | Individuals
Classes | Object properties | Data properties

Class hierarchy: Grandmother

Asserted

- owl:Thing
  - Assassin
  - Dad
    - Grandfather
  - Daughter
  - Mage
  - Mom
    - Grandmother
  - Pirate
  - Son

Grandmother — http://www.semanticwe

Annotations | Usage

Annotations: Grandmother

Annotations

Description: Grandmother

Equivalent To
(parent some (parent some owl:Thing)) and (woman value true)

SubClass Of
Mom

General class axioms

SubClass Of (Anonymous Ancestor)
(parent some owl:Thing) and (woman value true)

Instances

Target for Key

Disjoint With
Grandfather

Disjoint Union Of

Annotation properties | Datatypes | Individuals
Classes | Object properties | Data properties

Class hierarchy: Pirate

Asserted

- owl:Thing
  - Assassin
  - Dad
    - Grandfather
  - Daughter
  - Mage
  - Mom
    - Grandmother
  - Pirate
  - Son

Pirate — http://www.semantic

Annotations | Usage

Annotations: Pirate

Annotations

Description: Pirate

Equivalent To
beDescendantOf some Pirate

SubClass Of

General class axioms

SubClass Of (Anonymous Ancestor)

Instances
Bernard_Kenway

Target for Key

Disjoint With

Disjoint Union Of

Annotation properties | Datatypes | Individuals
Classes | Object properties | Data properties

Class hierarchy: Son

Asserted

- owl:Thing
  - Assassin
  - Dad
    - Grandfather
  - Daughter
  - Mage
  - Mom
    - Grandmother
  - Pirate
  - Son

Son — http://www.semanti

Annotations | Usage

Annotations: Son

Annotations

Description: Son

Equivalent To
(childOf some owl:Thing)
and (man value true)

SubClass Of

General class axioms

SubClass Of (Anonymous Ancestor)

Instances

Target for Key

Disjoint With
Daughter

Disjoint Union Of

Запустим Reasoner и выполним запросы:

Выведем всех мам

Query (class expression)

Mom

Execute   Add to ontology

Query results

Instances (12 of 12)

- 'Sef's_Daughter_(1)_Ibn-La'Ahad'
- 'Sef's_wife_(Ibn-La'Ahad)'
- Flavia_Auditore
- Kanieht:io
- Linette_Hopkins
- Maria_Thorpe
- Maria_de'Mozzi_da_Firenze
- Maud
- Mother_of_Desmond_Miles
- Sofia_Sartor
- Tessa_Stephenson-Oakley
- William's_Mother_Miles

Выведем всех всех матерей, являющихся Ассассинами

Query (class expression)

Mom and Assassin

Execute   Add to ontology

Query results

Instances (2 of 2)

- 'Sef's_Daughter_(1)_Ibn-La'Ahad'
- Mother_of_Desmond_Miles

Выведем сущности, являющиеся пиратом, Магом и Ассассином одновременно

Query (class expression)

Pirate and Mage and Assassin

Execute Add to ontology

Query results

Instances (1 of 1)

Subject_17_Desmond_Miles

Проверим , является ли Maud женщиной

Query (class expression)

{Maud} and woman value true

Execute Add to ontology

Query results

Instances (1 of 1)

Maud

Проверим, Является ли William_Miles мамой

Query (class expression)

{William_Miles} and Mom

Execute Add to ontology

Query results

Instances (0 of 0)

Приложение 2. Структура отчёта по первому блоку лабораторных работ

Введение:

Описание целей проекта и его значимости.

Анализ требований:

Определение основных требований к системе поддержки принятия решений.

Выявление требований к базе знаний и онтологии для представления знаний.

Изучение основных концепций и инструментов:

Обзор основных концепций баз знаний и онтологий.

Изучение Prolog и его возможностей для разработки систем искусственного интеллекта.

Ознакомление с инструментами и библиотеками, подходящими для работы с базами знаний и онтологиями на Prolog.

Реализация системы искусственного интеллекта (системы поддержки принятия решений):

Создание правил и логики вывода для принятия решений на основе базы знаний и онтологии.

Тестирование и отладка системы, обеспечение ее функциональности и эффективности.

Оценка и интерпретация результатов:

Примеры запросов для БЗ и онтологии, сравнение разницы реализации.

Оценка соответствия системы поставленным требованиям и достижению целей проекта.

Интерпретация результатов и описание дальнейших возможностей развития и улучшения системы.

Заключение:

Описание преимуществ и потенциальных применений разработанной системы искусственного интеллекта на базе Prolog, баз знаний и онтологий.

Приложение 3. Структура отчёта по второму блоку лабораторных работ

Лабораторная работа 1. Метод линейной регрессии

Введение

Описание задачи и целей.

Описание метода

Краткое описание метода линейной регрессии, его назначения и принципа работы.

Псевдокод метода

Псевдокод алгоритма метода линейной регрессии.

Результаты выполнения

Опишите полученные результаты после применения метода линейной регрессии. Можете включить графики, численные значения и примеры.

Примеры использования метода

Приведите примеры ситуаций, когда метод линейной регрессии может быть полезен. Объясните, почему именно этот метод выбран.

Лабораторная работа 2. Метод k-ближайших соседей (kNN)

Введение

Описание задачи и целей.

Описание метода

Краткое описание метода k-ближайших соседей, его назначения и принципа работы.

Псевдокод метода

Псевдокод алгоритма метода k-ближайших соседей.

Результаты выполнения

Опишите полученные результаты после применения метода k-ближайших соседей. Можете включить графики, численные значения и примеры.

Примеры использования метода

Приведите примеры ситуаций, когда метод k-ближайших соседей может быть полезен. Объясните, почему именно этот метод выбран.

Лабораторная работа 3. Деревья решений

Введение

Введение в тему лабораторной работы, описание задачи и целей.

Описание метода

Краткое описание метода деревьев решений, его назначения и принципа работы.

Псевдокод метода

Псевдокод алгоритма метода деревьев решений.

Результаты выполнения

Опишите полученные результаты после применения метода деревьев решений. Можете включить графики, численные значения и примеры.

Примеры использования метода

Приведите примеры ситуаций, когда метод деревьев решений может быть полезен. Объясните, почему именно этот метод выбран.

Лабораторная работа 4. Логистическая регрессия

Введение

Описание задачи и целей.

Описание метода

Краткое описание метода логистической регрессии, его назначения и принципа работы.

Псевдокод метода

Здесь представлен псевдокод алгоритма метода логистической регрессии.

Результаты выполнения

Опишите полученные результаты после применения метода логистической регрессии. Можете включить графики, численные значения и примеры.

Примеры использования метода

Приведите примеры ситуаций, когда метод логистической регрессии может быть полезен. Объясните, почему именно этот метод выбран.

Сравнение методов

Сравнительный анализ методов. Произведите сравнительный анализ результатов и производительности каждого метода. Опишите их преимущества и ограничения.

Примеры лучшего использования каждого метода

Укажите, в каких ситуациях каждый из методов наиболее эффективен и почему.

Заключение

Окончательные выводы и обобщение результатов.

Приложения

Код

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Бессмертный И.А., Искусственный интеллект. – Санкт-Петербург: Университет ИТМО. 2010. – 133 с. Текст: элек-тронный — URL: https://books.ifmo.ru/file/pdf/658.pdf
2. Кугаевских А.В., Муромцев Д.И., Кирсанова О.В., Классические методы машинного обучения. – Санкт-Петербург: Университет ИТМО. 2022. – 52 с. Текст: элек-тронный — URL: https://books.ifmo.ru/file/pdf/3075.pdf
3. Хорриддж М., Практический справочник по построению онтологий OWL в Protege 4, глава 4. – Манчестрский университет. 2012. – 41 с.
4. Quinlan J.R. C4.5: programs for machine learning. San Mateo, Calif: Morgan Kaufmann Publishers, 1993. 302 p.
5. McDermott, Drew. «Artificial Intelligence Meets Natural Stupidity», SIGART Newsletter, No.57 (April, 1976), pp. 4-9.
6. Зыков, С. В. Программирование. Функциональный подход : учебник и практикум для академического бакалавриата / С. В. Зыков. — М.: Издательство Юрайт, 2019.
7. Загорулько, Ю. А. Искусственный интеллект. Инженерия знаний : учеб. пособие для вузов / Ю. А. Загорулько, Г. Б. Загорулько. — М.: Издательство Юрайт, 2019.
8. Рассел, С. Искусственный интеллект: современный подход / С. Рассел, П. Норвиг. — 2-е изд. — М. : Вильямс, 2015.
9. Осипов, Г. С. Методы искусственного интеллекта / Г. С. Осипов. — М.: ФИЗМАТЛИТ, 2011.
10. Рубашкин, В. Ш. Онтологическая семантика. Знания. Онтологии. Онтологически ориентированные методы информационного анализа текстов / В. Ш. Рубашкин. — М. : ФИЗМАТЛИТ, 2012.

Бессмертный Игорь Александрович
Авдюшина Анна Евгеньевна
Кугаевских Александр Владимирович
Королева Юлия Александровна
Рущенко Нина Геннадиевна

Системы искусственного интеллекта

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49, литер А