ИНТЕЛЛЕКТУАЛЬНЫЕ ТЕХНОЛОГИИ

Практикум

М. И. Озерова, И. Е. Жигалов, Д. В. Шевченко

Авторская редакция 2021 г.

г. Владимир, 2021

Введение

«Искусственный интеллект» (от англ. Artificial intelligence, AI) - раздел компьютерной лингвистики и информатики, занимающийся формализацией проблем и задач, которые напоминают задачи, выполняемые человеком. При этом в большинстве случаев алгоритм решения задачи неизвестен заранее. Точного определения этой науки нет, поскольку в философии не решен вопрос о природе и статусе человеческого интеллекта. Нет и точного критерия достижения компьютером «разумности», хотя перед искусственным интеллектом был предложен ряд гипотез, например, Тест Тьюринга или гипотеза Ньюэлла-Саймона. Сейчас существует много подходов, как к пониманию задач искусственного интеллекта, так и к созданию интеллектуальных систем.

Эта наука связана с психологией, нейрофизиологией, транс гуманизм и другими. Как и все компьютерные науки, она использует математический аппарат. Особое значение для нее имеют философия и робототехника.

Термин интеллект (лат. intellectus) означает ум, рассудок, способность мышления и рационального познания. Обычно, под этим подразумевается способность приобретать, запоминать, применять и преобразовывать знания для решения каких-то задач. Благодаря этим качествам человеческий мозг способен решать разнообразные задачи. В том числе те, для которых нет заранее известных методов решения.

Термин искусственный интеллект возник сравнительно недавно, однако уже сейчас практически невозможно представить себе мир без него. Ее исторический путь напоминает синусоиду, каждый «взлет» которой инициировался некоторой новой идеей. Чаще всего, люди не замечают его присутствия, но если бы, вдруг, его не стало, то это коренным образом отразилось бы на нашей жизни. Сферы, в которых

используются технологии искусственного интеллекта, постоянно пополняются: когда-то это были программы для игры в шахматы, потом – роботы пылесосы, сейчас алгоритмы способны сами проводить торги на биржах.

Данное направление образовалось на базе утверждения, что интеллект человека может быть детально описан и, впоследствии, успешно имитироваться машиной. Искусственный интеллект являлся причиной огромного оптимизма, однако вскоре показал ошеломляющую сложность реализации.

Основные направления развития искусственного интеллекта включают рассуждения, знания, планирование, обучение, языковую коммуникацию, восприятие и способность двигаться и манипулировать объектами. Универсальный искусственный интеллект (или «сильный ИИ») всё ещё в планах на будущее. В настоящее время популярные подходы включают статистические методы, вычислительного интеллекта и традиционной символической ИИ. Существует огромное количество инструментов, использующих искусственный интеллект: разные версии поисковых алгоритмов, алгоритмы математической оптимизации, логики, методы, основанные на вероятности и многие другие.

1. Системы искусственного интеллекта.

В данный момент ИИ используется в самых разных областях человеческой деятельности, начиная от универсальных задач типа обучения и восприятия, и заканчивая очень конкретными задачами типа игры в шахматы, доказательства математических теорем, написания поэзии, а также диагностики заболеваний. Что же такое ИИ? Можно встретить различные определения. За 50 лет развития этой области давались различные определения основного понятия, зачастую явно противоречащие друг другу. Эта путаница возникала в силу того, что под основное понятие можно было подвести совершенно различные виды систем. Легче всего выделить специальные четыре класса, для которых применимо понятие ИИ:

· Системы, которые рассуждают, как люди.

· Системы, которые рассуждают рационально.

· Системы, которые действуют, как люди.

· Системы, которые действуют рационально.

Первые два типа систем концентрируются на вопросах мышления и рассуждения, тогда как остальные два типа систем касаются поведения. На протяжении истории данной области ученые применяли все четыре подхода к определению ИИ. Особо стоит отметить слово “рационально” - оно подразумевает, что существует какая-то идеальная концепция разумности, которую мы и считаем рациональной. Рассмотрим по порядку все подходы с точки зрения истории:

1.1 Системы, действующие как человек - тест Тюринга

В 1950 году Алан Тюринг в своей знаменитой работе предложил специальный тест для проверки того, можно ли считать ту или иную систему разумной. Вместо того, чтобы перечислять и определять качества, присущие разумной системе, он предложил тест на проверку того, можно ли отличить поведение такой системы от поведения человека. Для того, чтобы вести диалог согласно этому тесту, системе придется обладать следующими способностями:

· Обработка текстов на естественном языке для успешного ведения диалога. · Представление знаний для хранения текста, а также дополнительных сведений о контексте диалога.

· Автоматизированный механизм рассуждений, который на основе имеющейся информации позволяет делать умозаключения, а также задавать наводящие вопросы.

· Машинное обучение, для адаптации к новым обстоятельствам, а также обнаружению и экстраполяции шаблонов.

Оригинальный тест Тюринга специально исключил физическое взаимодействие жюри и машины, поскольку физическая симуляция не является необходимым признаком разумности. Однако существует также полный тест Тюринга, включающий в себя видеосигналы и способность взаимодействия с системой физически.

Для прохождения этого теста системе также понадобятся следующие способности:

· Компьютерное зрение, для восприятия объектов.

· Робототехника, для манипулирования объектами.

Эти 6 областей, по большей части и являются вопросами ИИ, и надо отдать должное Тюрингу за создание теста, покрывающего все эти вопросы, причем в самом начале славного пути ИИ. Однако этот тест на протяжении всей своей истории сталкивался с весьма серьезной критикой. Главное возражение заключается в том, что история науки и техники свидетельствует о неэффективности попыток имитировать живую природу при попытке построить техническую систему. Веками люди пытались сконструировать систему крыльев, которая позволила бы им летать, подобно птицам. Все эти попытки потерпели неудачу. Летать человек стал после того, как отказался от попыток подражать природе, и пошел своим путем. Сначала через постройку воздушных шаров и дирижаблей, а затем путем постройки летательных аппаратов тяжелее воздуха и искусственными двигателями. Ни в одном тексте, посвященном самолетостроению и вопросам аэродинамики, вы не встретите задачи типа “Сделать самолет, который бы летал как голуби, и делал это так искусно, что ввел бы в заблуждение других голубей”. Сама постановка задачи в таком стиле привела бы авиаконструкторов в удивление.

1.2 Системы, мыслящие как человек - когнитивные науки

Чтобы реализовать такого рода системы, надо разобраться с тем, как мыслит сам человек. Этого можно достичь либо интроспекцией, либо с помощью психологических экспериментов. Как только появится достаточно точная теория мыслительных процессов, ее можно будет формализовать и выразить в виде программы.

Например, Саймон и Ньюэлл, создавшие универсальный решатель задач, не были удовлетворены тем, что он просто решал задачи. Их цель заключалась в сравнении процесса рассуждений этой программы с процессом рассуждения человека при решении задачи. Их универсальный решатель задач (GPS - General problem solver) был ценен не столько своими практическими результатами в плане

доказательства теорем, сколько тем подходом, который они использовали, и который оказался весьма плодотворным. На заре существовании ИИ часто смешивались два подхода - модели ИИ для компьютеров и экспериментальные подходы психологии. Автор в статье тех времен мог заявлять, что алгоритм, который хорошо себя показывает на определенной задаче, является поэтому адекватной моделью для рассуждения человека над этой задачей, и наоборот. В настоящий момент принято разделять эти две области, что позволило им развиваться куда быстрее, чем во времена смешения. Правда, в области компьютерного зрения и естественных языков эти две области по-прежнему серьезно взаимодействуют. Здесь основная проблема та же, что и для теста Тюринга. Мы пытаемся построить систему, мыслящую так же, как и человек. Опыт науки и техники подсказывает, что с куда большей вероятностью удастся добиться успеха при построении разумных систем, мыслящих не так, как человек.

1.3 Системы, рассуждающие рационально - законы логики

Еще Аристотель предпринял попытку зафиксировать правила “правильных рассуждений”. Его силлогизмы представляли шаблоны для рассуждений, которые всегда приводили к истиным заключениям, если их посылки были верны. Эти законы мышления и послужили основанием для такой науки как логика. Один из известнейших силлогизмов такого рода звучит так:

```
Большая посылка. Все люди смертны.
Малая посылка. Сократ - человек.
Заключение. Сократ - смертен.
```

В 19-м веке логики умудрились выработать нотации для обозначения практически всех вещей и взаимоотношений между ними. К 1965 году уже были разработаны программы, которые, в теории, могли решить любую разрешимую задачу, описанную в логической нотации. Однако на этом пути встретилось два перпятствия. Во-первых, не всегда легко из неформального описания задачи получить формальное математическое представление (возьмите поваренные

рецепты), особенно если знание не является 100% достоверным. Во-вторых, существует на практике огромная пропасть между “может решить в принципе” и “решит на практике за приемлемые время и цену” (Кук в своем классическом труде, заложившем основные понятия NP-полных задач, заострил этот вопрос до предела). И хотя эти препятствия имеют место в любой системе ИИ, наиболее драматически и ярко они проявились в системах на основе логических правил.

1.4 Системы, ведущие себя рационально - интеллектуальные агенты

Агент - это что-то, что действует (от латинского agere - делать что-то). Но агент имеет ряд существенных отличий от обычных программ:

· Способен действовать автономно.
· Способен воспринимать внешнее окружение.
· Способен адаптироваться к изменениям.
· Способен следовать к поставленной цели.

В предыдущем подходе упор делался на корректных рассуждениях (корректных с точки зрения логики). Иногда такие рассуждения являются частью агента. С другой стороны, логические рассуждения - это еще не все, чтобы было возможно действовать рационально. Есть рациональные действия, которые вообще не используют логических рассуждений (например, рефлексы). Зачастую такой подход оказывается эффективней с точки зрения рациональности поведения. Это легко показать на примере из биологии человека. Предположим, что Вы дотронулись до раскаленного чайника рукой. Измерения показали, что если бы сигнал от кожи руки прошел по нервной системе до мозга, мозг проанализировал эту информацию и отдал приказ отдернуть руку, то итоговое время, требуемое при таком подходе, равно примерно 1 секунде - вполне достаточно, чтобы получить серьезный ожог. Однако

большинство людей успевают отдернуть руку от чайника где-то за 0.3 секунды, используя выработанные в ходе жизни рефлексы.

2 Основания искусственного интеллекта

2.1 Философия

Философия пытается ответить на следующие вопросы:

· Можно ли использовать формальные правила для того, чтобы прийти к корректным заключениям?

· Откуда берется разум в физическом мозгу?

· Каков источник наших знаний?

· Как знание приводит к действию?

Как упоминалось выше, Аристотель был первым, кто описал законы для рационального поведения. Он создал неформальную систему силлогизмов. Позднее Раймон Лулль высказал идею, что полезные для нас рассуждения могут возникать чисто механически. Он построил специальную машину, для доказательства Бытия Божьего. Автоматизация вычислений была тоже на полном ходу в средние века - уже Леонардо да Винчи предложил конструкцию механического калькулятора. Хотя он его не построил, современные реконструкторы подтвердили верность его чертежей, построив на его основе действующий калькулятор.

Рене Декарт был сторонником дуализма. Он считал, что часть нашего разума (или души или духа) была вне природы и не зависела от чисто физических особенностей мозга. Он считал, что у животных этой части нет. Альтернативой дуализму является материализм (отсюда споры о свободе воли). Что касается источника знания, то здесь определяющим для науки является движение эмпиризма, основоположником которого являлся Фрэнсис Бэкон.

Дэвид Юм предложил принцип индукции - что общие правила могут быть выведеы из наблюдения над повторяющимися связями между элементами. В начале XX века возникло движение логического позитивизма - все наши знания могут быть охарактеризованы логическими теориями, связанными с наблюдениями и фактами, которые мы воспринимаем с помощью сенсоров.

Наконец, уже Аристотель задумывался над вопросами, как знание приводит к действию, различая концепцию целей и средств (цель - подцели - действия). Эта концепция была с успехом воплощена в одной из ранних систем ИИ - General Purpose Solver.

2.2 Математика

Применительно к ИИ математика пытается ответить на следующие вопросы:

· Каковы формальные правила для вывода корректных умозаключений?

· Что можно вычислить?

· Что делать с недостоверной информацией?

Мат. логика связана с работами Буля. В 1879 Г. Фридж расширил его логику, включив в нее объекты и отношения, и тем самым создав логику первого порядка, которая и лежит в основе представлений знаний большинства современных систем.

Развитие компьютерной отрасли привело к появлению огромного количества алгоритмов. Однако ряд алгоритмов возникли задолго до появления компьютеров. Самый древний из известных - это алгоритм Евклида, позволяющий найти наибольший общий делитель для двух чисел. В начале XX века знаменитый математик Дэвид Гильберт поставил вопрос о том, является ли система аксиом арифметики логически непротиворечивой. Рассел и Уайтхед предприняли попытку формализации и аксиоматизации всего знания математики, имевшегося на тот момент. Если бы удалось выявить такой набор аксиом, то, используя стандартные правила, можно было бы получать новые знания чисто механическим способом. Эти попытки перечеркнул Курт Гедель своей знаменитой теоремой о неполноте. Она известна в нескольких формулировках. В контексте данного предмета представляют интерес следующие формулировки его теорем (их на самом деле было две):

1. Если формальная логика непротиворечива, то в ней существует невыводимая и неопровержимая формула.
2. Если формальная логика непротиворечива, то в ней невыводима некоторая формула, содержательно утверждающая непротиворечивость этой арифметики.

3. Всякая достаточно сильная рекурсивно аксиоматизируемая непротиворечивая теория первого порядка неполна (обобщенная формулировка).

Применительно к данной теме, Гедель показал, что в логике первого порядка (которая чаще всего используется в интеллектуальных системах) невозможно описать принцип индукции - один из наиболее распространенных способов доказательства утверждений в математики. Что еще хуже, существуют функции над целыми числами, которые могут быть однозначно описаны (т.е. для каждого входного значения они выдают вполне конкретное выходное значение), но которые при этом невозможно описать в виде алгоритма.

Стивен Кук в своей пионерской работе в 1971 году описал класс NP-полных задач, и показал, что существуют задачи, решение которых по времени либо по памяти будет выражаться экспоненциальной зависимостью от размера задачи. Он же поставил вопрос о том, можно ли любую задачу подобного рода свести к задаче, решаемой за полиномиальное время (знаменитая формула $P \stackrel{?}{=} NP$). На этот вопрос ответа пока что найти не удалось, однако большинство специалистов данной области склоняются к тому, что $P \neq NP$, т.е. всегда будут существовать задачи, решение которых быстро найти не удастся (в связи с комбинаторным взрывом). В данной статье[1] один из ведущих специалистов в данной области приводит ряд серьезных аргументов, почему практически наверняка $P \neq NP$. Это показывает ограниченность как человеческого интеллекта, так и всех искусственных реализаций. Всегда будут существовать сложные задачи, решение которых будет представлять проблему даже для самого совершенного интеллекта.

[1] https://www.scottaaronson.com/blog/?p=122

2.3 Экономика

Применительно к ИИ экономика пытается ответить на следующие вопросы:

· Как принимать решения, чтобы максимизировать прибыль?

· Как это делать, когда другие мешают?

· Что делать, если прибыль может быть лишь в отдаленном будущем?

Уже Адам Смит, заложивший основы современной политэкономии, рассматривал понятие “предпочтительного результата” и полезности действий экономических агентов. Нейман и Моргенштерн в труде “Теория игр и экономическое поведение” рассматривали интеллектуальных агентов с точки зрения нахождения такой стратегии поведения, которая позволит получить максимальный выигрыш или добиться минимального ущерба, если выигрыш невозможен. Чаще всего в реальном мире агенты действуют при неточной информации, и им нужно выбрать на очередном шаге не то действие, которое приносит максимальный локальный выигрыш (на такой основе действуют т.н. жадные алгоритмы). Агент должен подобрать последовательность операций, которые приведут в конечном итоге к максимальному выигрышу, с учетом того, что каждый отдельный шаг может вполне вести к проигрышу. Особый интерес в рамках таких исследований представляют марковские процессы.

2.4 Нейробиология

Применительно к ИИ нейробиология пытается ответить на следующие вопросы:

· Как мозг обрабатывает информацию?

На вопрос о роли мозга в рассуждениях и выработки решений значительно влияли споры философов двух направлений - дуалистов и материалистов. Дуалисты считали, что есть в живом человеке некоторая часть (возможно, душа) сознания и подсознания, которая существует независимо от материи и не исчезает после смерти. Материалисты отвергали это предположения, считая, что все

когнитивные процессы должны иметь какое-то материальное воплощение.

Что точно было известно насчет мозга, это его влияние на характер. Это было ясно из того факта, что у людей, имевших травмы головы в той или иной области, зачастую кардинально менялся характер поведения, а также изменялся уровень интеллекта. Окончательно роль мозга стала ясна, когда Брок открыл в мозгу специальную область, которая была ответственна за речевой аппарат (т.н. зона Брока). Это положило старт бурным исследованиям мозговой деятельности. Бергер в 1929 году разработал новый метод исследования - ЭЭГ мозга. Работы Огавы привели в 1990 году к появлению нового метода исследования мозга - МРТ. В результате этих исследований стало ясно, что большое количество клеток в мозгу, объединенных сложными взаимосвязями, позволяет создавать весьма сложные когнитивные процессы. Эта точка зрения подкрепляется такой областью ИИ, как нейронные сети. Интересно сравнить развитие элементной базы у компьютеров и людей на данный момент:

	Computer	Human Brain
Comp. units	1 CPU, 10^8 gates	10^{11} neurons
Stor. units	10^{10} bits RAM	10^{11} neurons
	10^{11} bits disk	10^{14} synapses
Cycle time	10^{-9} sec	10^{-3} sec
Bandwidth	10^{10} bits/sec	10^{14} bits/sec
Memory updates/sec	10^9	10^{14}

Таблица 1: сравнение людей и компьютеров.

Следует учитывать, что, согласно закону Мура, каждые несколько лет производительность вычислительных систем растет. По количество элементарных ячеек компьютеры вскоре превзойдут человеческий мозг (уже почти превзошли по ряду показателей). Однако у мозга есть одно неоспоримое преимущество в том плане, что множество нейронов могут менять свое состояние одновременно, передавая сигналы изменения практически всем элементам сети мозга. У компьютеров с таким параллелизмом пока что серьезные проблемы (чаще всего в каждый момент времени компьютер может изменить состояние одной или нескольких ячеек памяти, а отнюдь не все сразу).

2.5 Психология

Применительно к ИИ психология пытается ответить на следующие вопросы:

· Как люди и животные мыслят и действуют?

· Всегда ли мыслительный процесс предшествует действию?

Начало экспериментальной психологии (на крысах и птицах) положил Гельмгольц со своей лабораторией в Лейпциге. К середине XX века в психологии стало доминировать такое направление, как бихевиоризм, которое рассматривало человека как сложную машину. Это направление, по сути, отвергало мыслительные конструкции, рассматривая человека как систему “стимул-ответ”. Этот подход оказался весьма плодотворным при описании обычных млекопитающих (например, крыс), однако применительно к человеку и другим нашим собратьям из семейства африканских человекообразных обезьян он оказался малоприменим, в силу наличия способности к мыслительным процессам.

2.6 Компьютерная инженерия

Существование ИИ хотя и возможно без компьютеров в теории, однако на практике крайне редко встречается.

Применительно к ИИ компьютерная инженерия пытается ответить на следующие вопросы:

· Как построить эффективный компьютер?

· Какой язык программирования будет наиболее эффективен при построении искусственного интеллекта?

В ИИ есть множество концепций, изначально появившихся в других областях компьютерной инженерии.

Перечислим кратко некоторые из них:

· Концепция разделения времени, позволяющая множеству пользователь одновременно общаться с одним приложением.

· Интерактивные интепретаторы - LISP задал тон в этом направлении на несколько десятилетий вперед.

· Списочные типы - опять же LISP.

· Управление памяти в автоматическом режиме (сборщики мусора) - опять же LISP.

· Символьное программирование.

· Функциональное программирование.

· Объектно-ориентированное программирование.

2.7 Теория контроля и кибернетика

Применительно к ИИ теория контроля пытается ответить на следующие вопросы:

· Как могут технические устройства сами себя контролировать и менять свое поведение?

Тесибий Александрийский создал водяные часы с регулятором. Это позволяло часам держать поток воды на стабильном уровне, необходимом для их исправного функционирования. Часы могли понижать или повышать скорость потока. Фактически это первое известное устройство, которое могло само контролировать свое поведение и изменять его произвольно, без контроля со стороны человека. Джеймс Уатт создал паровой двигатель с системами саморегуляции. Дреббел создал термостат. В XIX в. бурно развивается теория стабильных систем с обратной связью. Это область оказалась достаточно важной для последующего развития ИИ. Из этой теории благодаря трудам Норберта Винера выросла теория кибернетики.

Следует заметить, что между теорией контроля и ИИ есть много схожего, однако гораздо больше различий. Поскольку теория контроля позволяет создавать устройства со сложным поведением, однако это поведение заложено конструктивно. Устройство не может изменить свое поведение самостоятельно, если возникла некоторая непредвиденная ситуация. Оно неспособно в большинстве случаев объяснить свое поведение. Наконец, в такого рода системах отсутствуют какие-либо мыслительные процессы (или их имитация).

2.8 Лингвистика

Применительно к ИИ лингвистика пытается ответить на следующие вопросы:

· Как связаны язык и мышление?

Уже основатель бихевиоризма Скиннер заложил основы вербального бихевиоризма. Интересны в этом плане не столько сами исследования Скиннера, сколько рецензия на эти исследования за авторством Ноама Хомского - “Синтактические структуры”. Хомский показал, как ребенок может понимать и создавать предложения, которые он никогда раньше не слышал. Эта теория была достаточно формальна для того, чтобы ее можно было записать в виде программы. **Ключевой проблемой лингвистики в контексте ИИ является гипотеза, что любой интеллект должен интерпретировать данные в символьной форме.** Вопроса на данный ответ пока что нет.

3 История развития ИИ

3.1 Зачатки области ИИ Период - 1943-1955.

Первой работой, которая ныне является общепризнанным трудом именно в области ИИ, считают работу Мак-Каллога и Питтса (1943). Они опирались в своей работе на три основных источника:

· Базовые сведения о физиологии и функционировании нейронов в человеческом мозгу.

· Формальный анализ пропозициональной логики, проведенный Расселом и Уайтхедом.

· Теория вычислений, предложенная Тюрингом.

В своей работе Мак-Каллог и Питтс предложили модель искусственного нейрона, где каждый нейрон мог быть в состояниях “on” и “off”. Переключение в режим “on” происходило в результате воздействия достаточного количества стимулов от соседних нейронов. Авторы работы также показали, что любая вычислимая функция могла быть представлена с помощью некоторой сети взаимосвязанных нейронов, и что все логические операции (И, ИЛИ, НЕ и т.д.) могут быть представлены в виде стандартных нейронных структур. Авторы также высказали предположение, что достаточно сложная и правильно сконструированная нейронная сеть способна к обучению. Дональд Хебб в своей работе (1949) предложил простое правило для обновления весов связей между нейронами. Правило Хебба по-прежнему остается в ходу даже сегодня.

Два выпускника Принстона, Минский и Эдмондс, сумели построить первую нейронную сеть в 1951 году. Эта сеть использовала 3000 вакуумных ламп и специальный механизм автопилота бомбардировщика B-24 для моделирования сети, состоящей из 40 нейронов. Однако аттестационная комиссия, которой Минский представил свою магистерскую диссертацию, проявила определенный скептицизм в плане того, что эту работу вообще следует рассматривать как исследование в области математики. Положение спас фон Нейман, который заявил “Если сейчас она не является таковой, то станет потом”. Впоследствии Минский доказал ряд важных теорем, которые свидетельствовали об определенном наборе ограничений, присущих нейронным сетям.

Самым важным прорывом в теории в этот период следует считать статью Тюринга (1950), в которой он предложил знаменитый тест Тюринга. Помимо этого, в этой работе были предложены такие концепции, как машинное обучение, генетические алгоритмы и обучение с подкреплением.

3.2 Рождение области ИИ

Год рождения - 1956. Не отрицая важности предшествующего периода, описанного выше, именно 1956 год следует считать водоразделом, выделившим ИИ в отдельную область. В этом году Мак-Карти (одна из наиболее влиятельных фигур в этой области и автор языка LISP) убедил Минского, Шеннона (создавшего первую программу для игры в шахматы) и Рочестера собрать наиболее авторитетных исследователей США в данной области вместе. Был организован двухмесячный семинар в Дартмуте летом 1956.

Именно на этом семинаре два исследователя из Карнеги, Ньюэлл и Саймон, представили работоспособную программу, способную доказывать математические теоремы. Вскоре после семинара, эта программа сумела найти доказательства большинства теорем, упомянутых во второй главе “Принципов математики” Рассела и Уайтхеда. Особое впечатление на самого Рассела произвела демонстрация доказательства одной из теорем, найденного программой, которое оказалось короче доказательства, найденного людьми. Однако редакторы одного из самых авторитетных американских журналов “Journal of symbolic logic” были не столь

впечатлены. Они отказали в публикации статье, где в качестве авторов были указаны Ньюэлл, Саймон и их программа (программа была указана в качестве основного автора). Хотя сам по себе семинар в Дартмуте не привел к каким-то прорывам в научном плане, однако он был чрезвычайно важен в плане организации научной деятельности.

В последующие 20 лет прогресс в области ИИ определялся в основном работами участников семинара и их научных школ. Именно на этом семинаре для этой научной области было принято предложенное Мак-Карти определение - искусственный интеллект.

3.3 Время больших ожиданий Период - 1956-1969.

Успех Ньюэлла и Саймона получил свое развитие в программе, получившей название General Problem Solver (GPS). В отличие от программы, представленной в Дартмуте, эта программа с самого начала пыталась имитировать способы рассуждения, которые использует человек. Для задач из определенных областей подобный подход оказался крайне плодотворным. Успех GPS и других программ того периода привели к тому, что Ньюэлл и Саймон сформулировали знаменитую символьную гипотезу, которая гласила, что “физическая система представления символов является необходимой и достаточной для разумных действий в общем случае”. Говоря иначе, они имели в виду, что любая система (машина или живое существо), чтобы действовать разумно, должна манипулировать последовательностями символов.

Тем временем в IBM был создан ряд важных программ в области ИИ. Гелернтер (1959) создал программу Geometry Theorem Prover, которая сумела доказать ряд теорем, представляющих сложность для большинства студентов специальностей физ.-мат. факультетов. Начиная с 1952, Артур Сэмьюэл написал ряд программ для игры в шашки, постепенно доведя их до уровня профессионалов. В этой разработке он сумел опровергнуть устойчивое убеждение, что программы могут делать только то, что заложит в них создатель, поскольку его программы достаточно быстро научились играть сильнее, чем он сам.

Что касается MIT, то Мак-Карти перебрался из Дартмута в этот институт и сделал три важных вклада в область ИИ в 1958 году:

· Он разработал высокоуровневый язык LISP, который стал доминирующим в ИИ на несколько десятилетий. Это был второй высокоуровневый язык (первым был FORTRAN).

· Столкнувшись с жесткой нехваткой вычислительных ресурсов в институте, Мак-Карти и его коллеги, страдавшие от той же проблемы, придумали концепцию разделения времени, которая оказалась исключительно важной для компьютерной индустрии в целом.

· Наконец, в том же году он опубликовал статью, в которой была описана программа Advice Taker, которая считается первой полноценной программой в области ИИ. Подобно предшественникам, эта программа должна была находить решения определенных задач. Однако ключевым отличием данной программы является то, что она должна была содержать лишь самые общие знания о мире. К примеру, он продемонстрировал, как путем использования нескольких простейших аксиом программа могла составить маршрут поездки из дома до аэропорта. Программа также поддерживала добавление новых аксиом для решения требуемой задачи, и поэтому могла быть использована для новых задач без переписывания исходного текста программ.

Марвин Минский также перебрался в MIT. Между ним и Мак-Карти возникли серьезные разногласия. Если Мак-Карти делал постоянно упор на символьных структурах и формальной логике, то Минского скорее интересовало, как создать работоспособную программу, способную решить действительно важную задачу. Поэтому Минский со своими учениками сосредоточился на создании программ, способных проявлять рациональность для задач из узких областей. Эти области были названы микромирами, и в решении задач из таких областей удалось добиться впечатляющих успехов. Программа Saint (1963) могла решать задачи интегрирования, которые студенты решают на первом курсе. Программа Analogy (1968) решала различные геометрические задачи из тестов на IQ. Программа Student (1967) за авторством Дэниела Боброва решала алгебраические задачи, записанные в форме:

```
If the number of customers Tom gets is twice the square of 20 percent of the number
of advertisements he runs, and the number of advertisements he runs is 45,
what is the number of customers Tom gets?
```

Здесь специально приводится оригинал задачи на английском, поскольку русский вариант представляет дополнительные сложности в силу специфики нашего языка (наличие окончаний в глаголах и т.д.).

Не стоит забывать и про нейронные сети. Работа Винограда и Коуэна (1963) продемонстрировала, что большое количество базовых блоков сети могут представлять отдельные концепции. Правило Хебба было усовершенствованов Видроу (1960). Розенблатт представил концепцию персептронов (1962). Также Розенблатт доказал теорему о сходимости персептронов о том, что алгоритм обучения (предложенный им в этой же работе) может настроить веса связей между нейронами для любых входных данных, если подходящий набор весов для этого набора вообще в принципе существует.

3.4 Отрезвление и возврат к реальности Период 1966-1973.

Предыдущий период был временем, когда исследователи не стеснялись делать очень смелые прогнозы в области ИИ. Например, Саймон в 1957 году предсказал, что в течение 10 лет компьютеры смогут выиграть чемпионат мира по шахматам, а также доказать серьезные математические теоремы, недоказанные людьми. Вместо 10 лет на это ушло примерно 40 лет.

Первые тревожные признаки возникли при попытке построить программы, способные переводить текст. Изначально работы в этой области спонсировали военные, которые хотели ускорить перевод русских научных работ после запуска СССР спутника в 1957 году. Предполагалось, что простые синтаксические преобразования между русской и английской грамматиками, а также замена слов с помощью элементарных словарей в памяти будут вполне достаточными инструментами для полноценного перевода предложений с сохранением исходного смысла. Однако на поверку оказалось, что для такого перевода система должна располагать хотя бы минимальными знаниями о той конкретной профессиональной области, к которой относится данная статья, для разрешения неоднозначностей и обработки контекста статьи. Знаменитый перевод машинами библейской фразы на английском “the spirit is willing but the flesh is weak” в русский вариант “Водка хорошая, а вот закуска протухла” наглядно демонстрировал всю сложность данной задачи. В 1966 году

специальный отчет военного ведомства США констатировал, что нет ни одной системы, способной переводить научные статьи в целом, и нет даже ни одного перспективного прототипа. Практически все разработки в данном направлении были немедленно свернуты министерством обороны США.

Вторая проблема, с которой столкнулись ранние программы ИИ - это комбинаторный взрыв. Большинство программ тех времен работали путем перебора различных комбинаций, в надежде найти таким путем решение. Это работало для микромиров из-за достаточно ограниченного пространства возможных комбинаций. Однако развитие теории вычислительной сложности

показало, что существуют задачи, где нельзя найти решение путем перебора, даже если наращивать память и скорость оборудования (NP-полные задачи), поскольку время для решения растет экспоненциально с ростом размера задачи.

Третья проблема, с которой столкнулись ранние программы ИИ, заключалась в фундаментальных ограничениях, присущих тем структурам, которыми пытались оперировать данные программы. Например, Минский и Пейперт показали (1969), что, хотя персептроны (простая форма нейронных сетей) могут изучить все, что можно представить с их помощью, представить с их помощью можно очень немногое. Более конкретно, персептрон с двумя входами нельзя обучить распознавать ситуацию, когда значения входов не совпадают. Хотя это не распространялось на более сложные многослойные сети, однако прогресс в данном направлении остановился до открытия алгоритмов обучения сетей с помощью обратной связи.

4 Основные понятия интеллектуальных систем

Можно выделить следующие наиболее важные направления развития интеллектуальных систем (т.е. систем, решающих задачи, традиционно относимые к интеллектуальной сфере), в которых широко используются методы распознавания образов:

• распознавание символов (печатного и рукописного текстов, банковских чеков и денежных купюр и т.д.);

• распознавание изображений, полученных в различных частотных диапазонах (оптическом, инфракрасном, радиочастотном, звуковом) и анализ сцен;

• распознавание речи;
• медицинская диагностика;
• системы безопасности;
• классификация, кластеризация и поиск в базах данных и знаний (в том числе и в Интернет-ресурсах).

Для системы обработки информации **образ -** это совокупность данных об объекте или явлении, включающая параметры и связи

Образ - это описание объекта или процесса, позволяющее выделять его из окружающей среды и группировать с другими объектами или процессами для принятия необходимых решений.

Целью процедуры распознавания (классификации) является ответ на вопрос: относится ли объект, описанный заданными характеристиками, к интересующим нас категориям (классам) и если относится, то к какой именно.

Классы – это категории объектов, которые должны быть выделины или на которые необходимо разделить все множество образов в процессе распознавания

При распознавании изображений существенное влияние на точность оказывают:

•Масштаб. Изображения имеют разный масштаб. Предметы, которые мы воспринимаем как одинаковые, на самом деле занимают разную площадь на разных изображениях.

•Место. Интересующий нас объект может находиться в разных местах изображения.

•Фон и помехи. Предмет, который мы воспринимаем как что-то отдельное, на изображении никак не выделен, и находится на фоне других предметов. Кроме того, изображение не идеально и может быть подвержено всякого рода искажениям и помехам.

•Проекция, вращение и угол обзора. Изображение является лишь двумерной проекцией нашего трехмерного мира. Поэтому поворот объекта и изменение угла обзора кардинальным образом влияют на его двумерную проекцию — изображение. Один и тот же объект может давать совершенно разную картинку, в зависимости от поворота или расстояния до него.

•Анализ образа - отсутствует качественное продвижение в решении таких задач, как *анализ* 3-мерных сцен и перевод с одного языка на другой.

При сравнении зрительного образа с эталоном в зависимости от эталона методы можно разделить на:

1) Растровые – в виде матрицы точек, каждая точка обладает яркостью разной силы (0,1). Важно чтобы входное изображение и эталон были приведены к одинаковым размерам и т.д.

Недостатки:

- Необходимо выделять объекты из общего изображения
- Чувствительны к размерам

2) Признаковый. На изображении выделяются признаки: размер, цвет, объем и т.д. Выделяется множество признаков, которое можно представить, как вектор в пространстве

Недостатки:

- Необходимо выделять объекты из образа
- Выделять признаки априори, что чревато потерей информации из-за неполного (не адекватного) представления объекта.
- Кластеризация - разбиение множества объектов на прикластеры по признакам.

3) Структурный метод. На образе выделяются структурные элементы отрезки, дуги, точки. Структурные элементы и распознаются.

Недостатки

- Необходимость сегментации
- Априорное выделение признаков
- Реальное восприятие (человеческое мышление)
- Использует целостное восприятие среды (картины).

В мозгу строится целостная модель образа, работают и признаковое и структурное восприятие образа. Сведения об объекте поступают по конечному числу сенсоров (анализируемых каналов) в мозг, и каждому сенсору можно сопоставить соответствующую характеристику объекта.

Человеку необходимо мало времени, что б распознать лицо в толпе, человек лего воспринимает огромное количество информации, которые содержать в том числе и изображения и точно определяет на них объекты.

Помимо признаков, соответствующих нашим измерениям объекта, существует так же выделенный признак, либо группа признаков, которые мы называем классифицирующими признаками, и в выяснении их значений при заданном векторе X и состоит задача, которую выполняют естественные и искусственные распознающие системы.

Сложная задача распознавания образов требует разработки гибкой распознающей системы, которая могла бы классифицировать любой образ, имеющийся на изображении.

В целом проблема распознавания образов состоит из двух частей:

- обучение (система должна приобрести способность реагировать одинаковыми реакциями на все объекты одного образа)
- распознавание (распознавание новых объектов)

Основные принципы разработки распознающих систем:

1) заложить в компьютер как можно больше известных образов-шаблонов и сравнивать их с поступающими для распознавания неизвестными образами;
2) на первой стадии обязательно обрабатывают изображение и выделяют характерные признаки;
3) процесс обучения.

Любой алгоритм распознавания [1] в общем виде можно представить, как абстрактную функциональную систему R, состоящую из трех компонент:

R={A,S,P}, где

A={Ak }, k=1,...,K – алфавит классов – множество категорий, по которым должны распределить образы,

S={Sj}, j=1,...,n - словарь признаков - множество характеристик, из которых составляется описание образа,

P={Pl}, l=1,...,L - множество правил принятия решения.

Схема общего алгоритма распознавания образов представлена на рисунке 1.

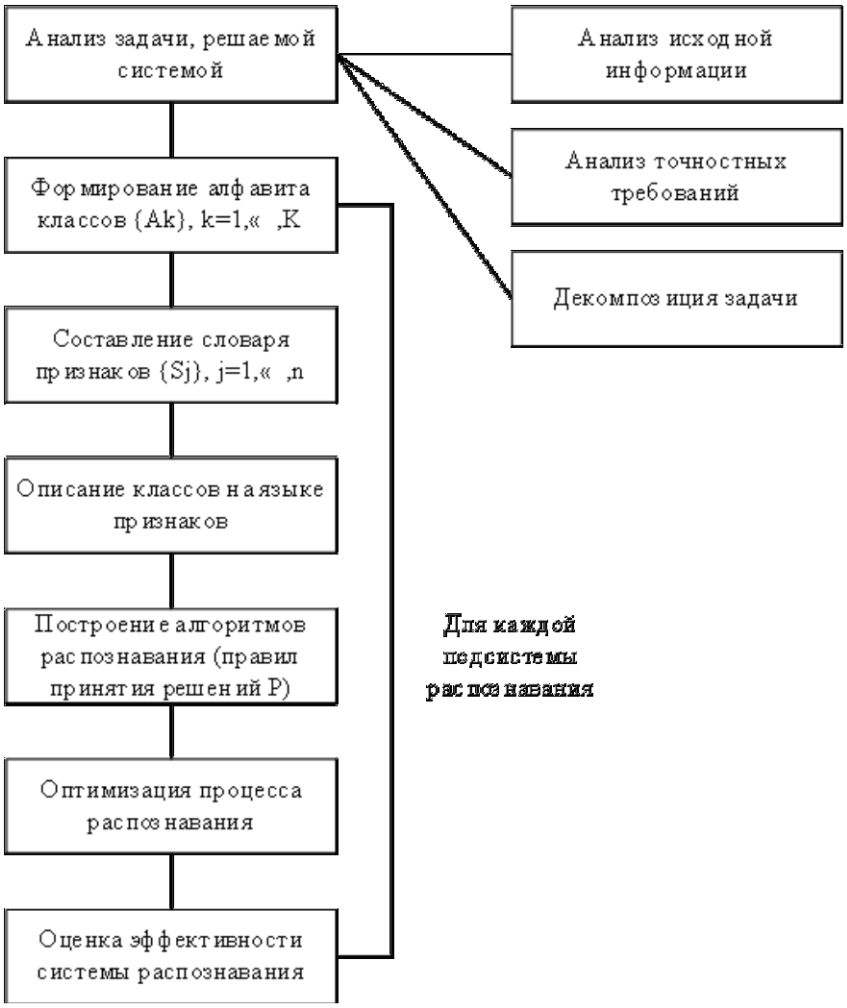


Рисунок *1* – Схема общего алгоритма системы распознавания

На вход подается образ – некоторая конфигурация из элементов множества S, к ней применяется определенная последовательность правил из P, в результате конфигурации присваивается индекс, соответствующий одному из элементов множества A.

Качество функционирования системы определяется тем, насколько часто присвоенный образу индекс совпадает с ожидаемым нами результатом.

Компоненты A, S представляют собой информационную часть системы, а P – методологическую.

Система распознавания включает как процесс **синтеза образов**, то есть формирования описаний объектов распознавания и их классов, так и **анализа образов**, то есть сам процесс принятия решений.

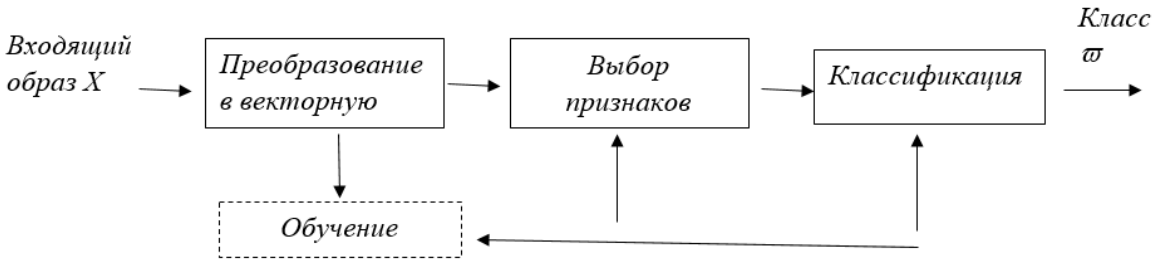


Рисунок 2- Общая схема построения системы распознавания

Математическая постановка задачи

Пусть U – множество образов в данной задаче распознавания. Отдельный образ из этого множества будем обозначать символом x. Каждый образ x U∈ может характеризоваться бесконечным (и даже несчетным) числом признаков. На этапе формирования алфавита признаков мы должны выбрать некоторое подмножество признаков (как правило, конечное), которое называют пространством признаков. Это множество будем обозначать через X . Предположим, что во множестве образов U в данной задаче распознавания нас интересуют некоторые подмножества – классы. Множество классов $\Omega = \{\omega_{1,\dots}, \omega_m\}$ является конечным, и классы образуют полную группу подмножеств из U.

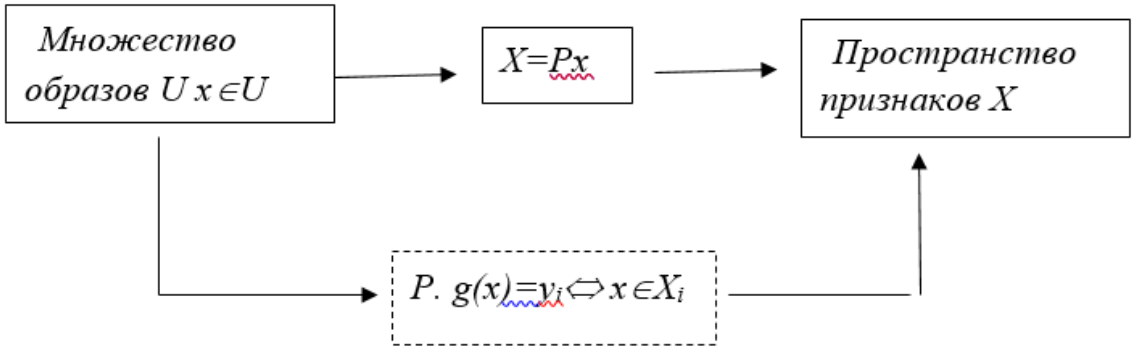


Рисунок 3-Постановка задачи

Будем считать, что входной образ-символ X может принадлежать некоторому классу из множества всех классов $\Omega = \{\omega_{1,\dots}, \omega_m\}$ – каждый класс соответствует некоторому символу (букве, цифре и т.д.). Предполагается, что классы являются непересекающимися. Классифицировать образ x $U\in$ по классам $\omega_{1,\dots}\ \omega_m$ – это значит найти функцию $g(x_i) = y$, если $x_i \in \varpi$.

Такая функция $U \rightarrow Y$, $Y=\{y_{1,..}\ y_m\}$, ставит в соответствие образу $x \in U$ метку $y_i \in Y$ того класса ϖ_i , которому он принадлежит, т.е. $g(x_i) = y$, если $x_i \in \varpi$. Задача распознавания образов состоит в соотнесении исходного образа x_i одному из классов ϖ_i.

Правила соотнесения образа одному из классов называются классификатором и реализуются в блоке классификации. Если образам соответствуют векторы – элементы метрического пространства, то соотнесение образа классу можно осуществить, например, с помощью вычисления расстояния между вектором и классом. На выходе классификатора должны получить тот класс (номер класса), которому принадлежит входной образ с указанием степени достоверности классификации или получить информацию о том, что входной образ не принадлежит ни одному из классов.

В некоторых задачах (например, при кластеризации данных) множество меток Y неизвестно. В этом случае система распознавания сама должна разбить обучающую выборку на классы, исходя из некоторых критериев оптимальности.

1. **Выбор наиболее информативных признаков, описывающих данный образ.**

Это одна из основных и важных задач в теории распознавания образов – найти минимальное количество признаков, наиболее информативно описывающих образы в данной системе (или задаче) распознавания. Полный набор выбранных для распознавания признаков называют алфавитом признаков. Минимальный же набор признаков, достаточный для решения данного класса задач распознавания, называют словарем признаков.

3. **Описание классов распознаваемых образов**.

Эта задача сводится к определению границ классов. Границы классов могут быть заданы явно на этапе разработки системы распознавания или система сама должна их найти в процессе своей работы.

4. **Нахождение оптимальных решающих процедур** (методов классификации), т.е. методов соотнесения вектора признаков образа некоторому классу.

5. **Оценка достоверности классификации образов**. Это необходимо для оценки величины потерь, связанных с неправильной классификацией

5 Представление изображений в векторной форме

Существует большое число различных форм представления изображений в распознающих устройствах или программах. Одной из наиболее простых и понятных является форма, использующая представление изображений в виде точек в некотором n-мерном пространстве. Каждая ось такого пространства естественным образом соотносится с одним из n входов или с одним из n рецепторов распознающей системы. Каждый из рецепторов может находиться в одном из m состояний, если они дискретны, или иметь бесконечно большое число состояний, если рецепторы непрерывны. В зависимости от вида используемых рецепторов может порождаться непрерывное, дискретное или непрерывно-дискретное n-мерное пространство.

Как правило, в пространстве изображений вводится метрика - функция, которая каждой упорядоченной паре точек x и y пространства ставит в соответствие действительное число d(x, y). При этом функция d(x, y) обладает следующими свойствами:

1. d(x, y) > 0, d(x, y) = 0 тогда и только тогда, когда x = y;
2. d(x, y) = d(y, x);
3. d(x, y) < d(x, z) + d(z, y).

Введение метрики $d(x, y)$ в пространстве изображений позволяет говорить о близости или удаленности точек в этом пространстве или о

мере сходства или различия анализируемых изображений. Понятие меры сходства изображений широко используется в теории распознавания образов. Однако формализация этого понятия при решении конкретных задач распознавания, как правило, не является тривиальной задачей. Более того, эта задача является одной из основных задач теории распознавания образов. Рассмотрим общие требования к мере сходства изображений.

Пусть задано некоторое конечное множество $S=\{S_1, S_2, \dots S_n\}$ входных изображений, каждое из которых является точкой в *n*-мерном пространстве изображений. Меру сходства изображений можно ввести как функцию двух аргументов $L(S_k, S_i), где S_k, S_i » S$. При этом функция $L(S_k, S_i)$ обладает следующими свойствами:

свойством симметрии, т.е. $L(S_k, S_i)=L(S_i, S_k)$;

- областью значений функции является множество неотрицательных чисел, т.е. $L(S_k, S_i)\geq 0, k, i=1,2\dots, n$;
- мера сходства изображения с самим собой принимает экстремальное значение по сравнению с любым другим изображением, т.е. в зависимости от способа введения меры сходства выполняется одно из двух соотношений:

$L(S_k, S_k)=max(L(S_k, S_i))$,

$L(S_k, S_k)=min(L(S_k, S_i))$;

- в случае компактных образов функция $L(S_k, S_i)$ является монотонной функцией удаления точек $S_k u S_i$ друг от друга в *n*-мерном пространстве.

Анализ свойств метрики и меры сходства изображений показывает, что

требования к функции $L(S_k, S_i)$ нетрудно выполнить в метрических пространствах. В частности, если в метрическом пространстве введено расстояние, то оно может быть использовано в виде меры сходства изображений.

Имеется 8 различных способов расчёта расстояний между изображениями:

$$L(S_i, X_j) = \sqrt{\sum_{k=1}^{n} |S_{ik} - X_{jk}|^2}, (1)$$

$$L(S_i, X_j) = \sqrt[\lambda]{\sum_{k=1}^{n} |S_{ik} - X_{jk}|^{\lambda}}, (2)$$

$$L(S_i, X_j) = \sum_{k=1}^{n} |S_{ik} - X_{jk}|, (3)$$

$$L(S_i, X_j) = \sqrt{\sum_{k=1}^{n} \eta_k |S_{ik} - X_{jk}|^2}, (4)$$

$$L(S_i, X_j) = \sqrt[\lambda]{\sum_{k=1}^{n} \eta_k |S_{ik} - X_{jk}|^{\lambda}}, (5)$$

$$L(S_i, X_j) = \sum_{k=1}^{n} \eta_k |S_{ik} - X_{jk}|, (6)$$

$$L(S_i, X_j) = \sum_{k=1}^{n} \left|\frac{S_{ik} - X_{jk}}{S_{ik} + X_{jk}}\right|, (7)$$

$$L(S_i, X_j) = 1 - \frac{2}{n(n-1)} \sum_{q=1}^{n-1} \sum_{\substack{k=2 \\ k>q}}^{n} \Delta_{qk}^{i} \Delta_{qk}^{j}, (8)$$

Однако большинство из них копируют друг друга с незначительными изменениями. Расстояния по Камберру(7) и по Кендалу(8) уникальны. А вот формулы 1 – 6 можно свести к одной, что облегчит программирование. Итоговая формула, которая будет использоваться для описания расстояний 1 – 6 выглядит $(S_i, X_j) = \sqrt[\lambda]{\sum_{k=1}^{n} \eta_k |S_{ik} - X_{jk}|^{\lambda}}$, (9). Формулы 1 – 6 получаются из 9 путём изменения параметров η_k и λ.

6 Использование метода главных компонент в анализе данных. Реализация на Python

В задачах машинного обучения, анализа данных приходится сталкиваться с огромными наборами информации, обладающими сотнями или тысячами классов и переменных, что затрудняет процесс их исследования. Для решения этой проблемы удобно сокращать число переменных таким образом, чтобы при меньшем их числе можно было бы охватить большую часть информации, необходимой для анализа.

Простым способом уменьшения размерности пространства переменных является применение методов матричной факторизации, в частности, одним из них является метод главных компонент.

Метод главных компонент — это статистическая операция, использующая ортогональное преобразование для трансформации набора коррелированных переменных в линейно некоррелированные, называемые главными компонентами. При этом первая главная компонента имеет наибольшую возможную дисперсию, и каждая последующая, в свою очередь, тоже, но возможную при условии, что она ортогональна предыдущим компонентам. Результирующие векторы образуют некоррелированный ортогональный базис.

В качестве набора данных для реализации метода главных компонент использовались ирисы Фишера. Они содержат информацию о 150 экземплярах ириса, по 50 экземпляров из трёх видов — ирис щетинистый, Ирис виргинский и ирис разноцветный. Каждый экземпляр характеризуется четырьмя характеристиками: длина наружной доли околоцветника; ширина наружной доли околоцветника; длина внутренней доли околоцветника; ширина внутренней доли околоцветника.

Для начала необходимо загрузить набор ирисов (листинг 1). К слову, для работы с данными в экосистеме Python существует большое количество библиотек и инструментов, в данном случае используется Pandas.

Листинг 1 – Загрузка набора данных

```
import pandas as pd
df = pd.read_csv(
    filepath_or_buffer='https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data',
    header=None,
    sep=',')
df.columns=['sepal_len', 'sepal_wid', 'petal_len', 'petal_wid', 'class']
df.dropna(how="all", inplace=True)
```

Теперь набор данных хранится в виде матрицы, где столбцы являются характеристиками, а каждая строка представляет собой класс цветка.

Поскольку метод главных компонент дает подпространство признаков, которое максимизирует дисперсию вдоль осей, имеет смысл стандартизировать данные. Хотя все объекты в наборе ирисов были измерены в сантиметрах, выполним преобразование данных в единичный масштаб (это является требованием для оптимальной производительности многих алгоритмов машинного обучения), как показано в листинге 2.

Листинг 2 – Масштабирование набора данных

```
x = df.loc[:, features].values
x_std = StandardScaler().fit_transform(x)
```

Следующим шагом необходимо вычислить собственные вектора и значения корреляционной матрицы. Они являются "ядром" метода главных компонент: собственные векторы или главные компоненты определяют направления нового пространства признаков, а собственные значения определяют их величину. Классическим примером для данного метода является выполнение разложения по собственным значениям для матрицы ковариации $\sum$, являющаяся матрицей $d \times d$. Ковариация между двумя объектами вычисляется следующим образом:

$$\sigma_{ijk} = \frac{1}{n-1}\sum_{i=1}^{N}(x_{ij} - \acute{x})(x_{ik} - \acute{x}_k)$$

Мы можем суммировать расчет ковариационной матрицы с помощью следующего матричного уравнения:

$$\Sigma = \frac{1}{n-1}$$

С помощью Python (листинг 3).

Листинг 3 – Расчёт ковариационной матрицы

```
mean_vec = np.mean(X_std, axis=0)
cov_mat = (X_std - mean_vec).T.dot((X_std - mean_vec)) / (X_std.shape[0]-1)
```

Затем необходимо выполнить разложение по собственным значениям для получившейся ковариационной матрицы (листинг 4).

Листинг 4 – Получение собственных векторов и значений

```
cov_mat = np.cov(X_std.T)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
```

Типичная цель метода главных компонент – уменьшить размерность исходного пространства характеристик, проецируя его на меньшее подпространство, где собственные векторы образуют оси. Однако собственные векторы определяют только направления новой оси.

Чтобы решить, какой собственный вектор(ы) может быть отброшен, не теряя слишком много информации для построения подпространства более низкой размерности, нужно проверить соответствующие собственные значения: собственные векторы с наименьшими собственными значениями несут наименьшую информацию о распределении данных; те, могут быть отброшены. Общий подход состоит в том, чтобы ранжировать собственные значения от самого высокого к самому низкому, чтобы выбрать вершину k собственных векторов, как реализовано в листинге 5.

Листинг 5 – Сортировка значений собственных векторов

```
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]
eig_pairs.sort(key=lambda x: x[0], reverse=True)
print('Отсортированные значения:')
for i in eig_pairs:
    print(i[0])
```

```
Отсортированные значения:
2.9303537755893174
0.9274036215173421
0.14834222648163944
0.020746013995595943
```

В результате получается, что большая часть дисперсии (~72,77%) может быть объяснена только первой главной составляющей. Второй компонент в свою очередь содержит примерно 23,03%, в то время как третий и четвертый компоненты могут быть безопасно удалены.

Следующим шагом будет построение проекционной матрицы, которая будет использоваться для преобразования данных ирисов в новое подпространство, уменьшая 4-мерное пространство признаков до 2-мерного подпространства, выбирая "верхние 2" собственных вектора с самыми высокими собственными значениями, чтобы построить $d \times k$-размерную матрицу собственных векторов W (листинг 6).

Листинг 6 – Построение проекционной матрицы

```
matrix_w = np.hstack((eig_pairs[0][1].reshape(4,1), eig_pairs[1][1].reshape(4,1)))
```

Далее используется 4×2-размерная проекционная матрица W для преобразования данных в новое подпространство через уравнение $Y = X \times W$, где Y – матрица ранее преобразованных ирисов (листинг 7). На рисунке 1 продемонстрирован результирующий график, который отображает каждый класс данных.

Листинг 7 – Преобразование данных в новое подпространство

```
Y = X_std.dot(matrix_w)

with plt.style.context('seaborn-whitegrid'):
    plt.figure(figsize=(6, 4))
    for lab, col in zip(('Iris-setosa', 'Iris-versicolor', 'Iris-virginica'),
                        ('blue', 'red', 'green')):
        plt.scatter(Y[y==lab, 0],
                    Y[y==lab, 1],
                    label=lab,
                    c=col)
    plt.legend(loc='lower center')
    plt.tight_layout()
    plt.show()
```

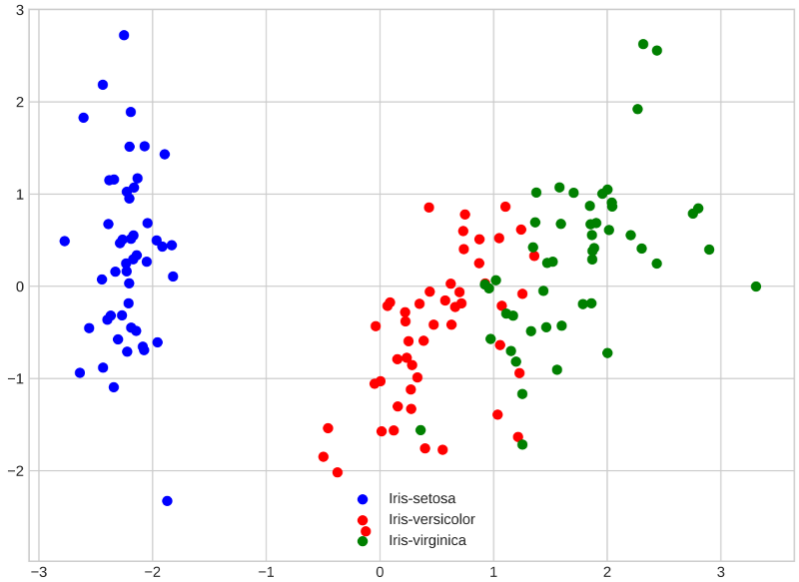


Рисунок 4- График классифицированных классов данных

Таким образом, метод главных компонент позволяет идентифицировать многомерные структуру данных с уменьшением их размерности. И в данной статье были описаны основные этапы и концепции анализа данных с использованием данного метода, реализованного на языке программирования Python.

7 Обзор существующих алгоритмов выделения границ на изображениях.

Роль сегментации – это выделение на изображении областей с определенными свойствами. Сегментация [1] является сложным процессом в обработке и анализе изображений, т.к. зачастую на изображениях присутствуют шумы, искажения, текстурные области, схожие с областями, принадлежащими исследуемому объекту. Все это затрудняет процесс распознавания объектов, поэтому при сегментировании используют алгоритмы обнаружения границ.

Рассмотрим наиболее популярные алгоритмы выделения границ на изображениях.

1. *Детектор границ Кэнни*

Алгоритм Канни [2] был разработан в 1986 году, когда компьютерное зрение только формировалось, как отдельное направление компьютерной графики. Однако, он и в наши дни остается одним из лучших детекторов границ. Алгоритм базируется на трех основных критериях:

- хорошее обнаружение, т.е. отношения сигнал/шум повышается;
- хорошая локализация;
- единственный отклик на одну границу.

Далее по полученным результатам находилась целевая функция стоимости ошибок, минимизацией которой находится «оптимальный» линейный оператор для свёртки с изображением.

Для многих методов обработки изображений, в том числе и для данного алгоритма, для уменьшения вычислительных затрат изображение преобразовывается в оттенки серого [3].

В начале алгоритма применяется первая производная Гауссиана для уменьшения чувствительности алгоритма к шуму. После сглаживания на контуре границ изображения остаются только точки максимума градиента. Детектор Кэнни использует четыре фильтра для выявления горизонтальных, вертикальных и диагональных границ для того, чтобы удалять точки именно рядом с границей и не разрывать саму границу вблизи локальных максимумов градиента. Далее с помощью двух порогов происходит удаление слабых границ. В итоге алгоритма при определении границ происходит подавление краев, которые не связаны с сильными границами.

2. *Оператор Собеля*

Данный оператор [3] базируется на приближении значений градиента яркости изображения. Он вычисляет градиент яркости в каждой точке на изображении, тем самым находит величину изменения яркости и ее направление. Результат показывает изменения яркости изображения в каждой точке, т.е. вероятность нахождения точки на границы, а также ориентацию.

Можно сказать, что результат оператора в точке, которая находится в постоянной яркости – это нулевой вектор. А точки, находящиеся в границе с различной яркостью – это вектор, который пересекает границу в направлении возрастания яркости.

Оператор Собеля использует фильтрацию изображения на основе свертки по горизонтали и вертикали, поэтому он легко вычисляется. Оператор использует матрицу 3x3, благодаря которым свертывают исходное изображение для дальнейшего вычисления приближенных производных по горизонтальным и вертикальным направлениям.

3. Векторный оператор Лапласа

Алгоритм Лапласа [4] основан на поиске нулей. Он использует производные второго порядка. В отличие от алгоритмов, использующих градиентный подход для выделения границ. Лапласиан является скалярной функцией. Он находит применение для выделения границ следующем образом, лапласиан принимает максимальное значение на участках «перегибов» функции яркости.

Из минусов данного алгоритма следует указать раздваивание границ, если она недостаточно резкая, т.е. имеются участки постоянного наклона. В этом случае поможет специальная обработка для устранения раздваивания линий.

4. Оператор Прюитт

Данный алгоритм [5] работает по принципу определения максимального отклика на множестве используемой матрицы для обнаружения локальной ориентации границ в каждом пикселе.

Для этого используются различные матрицы. Из одной матрицы можно получить восемь, переставляя ее коэффициенты.

Максимальное значение каждого пикселя – это пиксель в полученном изображении. Его значения могут быть в границах от 1 до 8, в зависимости от того, какая матрица дала наибольший результат.

Этот метод еще называют подстановкой шаблонов границ. Изображению сопоставляется набор шаблонов, каждый из которых показывает расположение границ. Тогда расположение границы в пикселе формируется шаблоном, который больше всех соответствует близлежащему в окрестности пикселю.

5. Перекрестный оператор Робертса

Является одним из первых алгоритмов выделения границ. В данном методе используется суммарный вектор из двух диагональных векторов [6]. Данный вектор показывает наибольшую разницу градиентов между охваченными точками. А направление данного вектора будет указывать наибольшую величину перепада между точками.

Оператор Робертса чаще всего используют благодаря его быстрым вычислениям, но по качеству он проигрывает альтернативным алгоритмам из-за чувствительности к шуму.

Сравнение алгоритмов

Из выполненного обзора существующих алгоритмов сегментации, можно сделать вывод, что они работают по-разному для различных задач. Требуется определить критерии сравнения работы данных алгоритмов. В задачах распознавания изображений важными критериями являются: точность, скорость работы алгоритма, наличие готовых решений и простота использования.

Далее по выбранным критериям был произведен сравнительный анализ рассмотренных ранее алгоритмов. Оценка производилась с использованием шкалы от 1 до 5. Сравнение представлено см. Табл. I.

Таблица 1 - Сравнение алгоритмов выделения границ на изображениях

Критерии	**Оператор Кэнни**	**Оператор Собеля**	**Оператор Лапласа**	**Оператор Прюитт**	**Оператор Робертса**
Скорость	5	2	3	2	5
Точность	3	4	3	3	2
Шумоподавление	5	5	4	5	1
Простота применения	4	3	1	3	2

Доступность (количество готовых решений)	4	4	4	1	3
Итого	21	18	15	14	13

8 Обзор интеллектуальных методов машинного перевода

Перевод текстов с одного естественного языка на другой рассматривается как процесс создания на другом языке некоторого текста, эквивалентного по содержанию и способам языкового выражения исходному. При этом человек-переводчик, осуществляя перевод, истолковывая и стилистически преобразуя текст, опирается на свое видение мира, проецирует текст сквозь призму своей личности, что приводит к проявлению индивидуальности в переводе, к неизбежному отклонению от текста оригинала. Использование технологии машинного перевода позволяет эффективно решить проблему субъективных трактовок, так как машина, содержащая в базе данных множество возможных вариантов, не истолковывает, а лишь передает обнаруженные текстовые соответствия [1, 10]. Машинный перевод, будучи одной из наиболее важных областей компьютерной лингвистики, включает в себя все проблемы обработки речи на всех языковых уровнях. Среди преимуществ машинного перевода отмечают возможность обработки большого объема данных и высокой скорости перевода при общей “нейтральности” выходных текстов [2, 117-118]. Необходимо различать системы автоматизированного перевода(computer aided software) и машинный (автоматический) перевод. Автоматизированный перевод, к классу которого относится программное обеспечение класса translation memory, -- это те программные средства, которые используются человеком-переводчиком в процессе перевода для повышения производительности труда. К машинному переводу (machine translation) относят технологии, позволяющие осуществлять перевод с одного языка на другой с помощью компьютерной программы без участия человека. При использовании программ автоматизированного перевода сокращается время, затрачиваемое на перевод, и

увеличивается его качество, однако основная часть работы лежит на человеке-переводчике. Средства автоматизации перевода обеспечивают более высокое качество перевода за счет единообразия терминологии и стиля, позволяют сохранить оригинальное форматирование, создавать память перевода на основе уже переведенных текстов и их оригиналов. Машинный перевод, осуществляя перевод без участия человека, требует при этом редактирования переведенного текста, так как неизбежно возникают ошибки и неточности, связанные с самой природой естественных языков: многозначностью, контекстуальностью, синонимичностью [3]. История развития машинного перевода берет начало в 1954 году, во время первой публичной демонстрации перевода с помощью вычислительной техники. Ранние технологии не обладали возможностями решения проблем многозначности, не проводили лингвистического анализа, а сам перевод был приближен к пословному. Второе поколение (вплоть до 1970-х годов) характеризовалось усилением роли языковых правил. В процессе перевода осуществлялось построение синтаксической структуры для каждого предложения, основанное на правилах грамматики входного языка. Затем происходило преобразование в синтаксическую структуру выходного языка, подстановка слов из словаря и синтез предложения на выходном языке. Третье поколение (до 1980-х) ознаменовалось появлением систем семантического типа, к которым относят системы машинного перевода с теорией “Смысл ^ Текст” в основе. Суть данной теории заключалась в использовании мета-языка, который позволил бы наиболее точно передавать не только форму, но и содержание языковых знаков. Использование таких уровней, как морфологический, фонологический, синтаксический и семантический, как предполагалось, должно было существенно повысить точность и качество перевода. Однако исследователи столкнулись с определенными трудностями при осуществлении перевода, основываясь на данной теории. В частности, не удалось разрешить проблему нахождения универсального для всех естественных языков смыслового представления. Немногим позже возникли интерактивные системы машинного перевода с привлечением участия человека на разных этапах перевода: пред- и постредактирование, частично автоматизированный перевод, смешанные системы. Доминирующей

технологией для конца 20-го столетия стало обучение машины посредством предоставления достаточно большого количества параллельных текстов на разных языках. Такой подход в дальнейшем получил название статистического. [4] В настоящее время различают следующие технологии машинного перевода: аналитический, статистический и нейронный машинный перевод. Аналитический машинный перевод или машинный перевод, основанный на правилах (rule-based machine translation) стал исторически первой технологией машинного перевода. В качестве основы основывается на создании связей текста на исходном языке с текстом на требуемом, сохраняя при этом оригинальное значение. Различают три типа систем машинного перевода, основанного на правилах:

● Прямые системы(Dictionary Based Machine Translation -- Машинный перевод, основанный на словаре) -- в их основе лежит словарный пословный перевод, т.е. слова представлены так же, как и в словаре, содержащимся в базе данных системы,

● Трансферные системы машинного перевода, основанного на правилах (Transfer Based Machine Translation) -- это один из наиболее широко используемых методов машинного перевода. В отличие от прямой модели машинного перевода, трансферный метод предполагает выполнение трех этапов в переводе: анализ исходного языка с целью определения грамматической структуры, перенос (трансфер) результирующей структуры на структуру целевого языка, генерация текста.

● Интерлингвальные системы (Interlingual RBMT Systems) -- при использовании данного метода текст исходного языка представляется в виде “нейтральной” структуры, находящейся вне зависимости от от каких-либо естественных языков. Текст на целевом языке формируется на основе этого нейтрального варианта. Одним из преимуществ данного метода является то, что возрастает значение языка-посредника, что и позволяет увеличить количество языков перевода. [5] Среди этапов процесса перевода выделяют морфологический анализ, объединение отдельных слов в группы, синтаксический анализ и определение посредством алгоритма каждого слова как члена предложения, синтез предложений. Рассмотрим перевод предложения “A girl eats an apple” с исходного английского языка на немецкий. На

первом этапе необходимо получить информацию о каждом из исходных слов: a - неопределенный артикль, girl - существительное, eats - глагол, an - неопределенный артикль, apple - существительное. Далее необходимо получить синтаксическую информацию о глаголе “to eat”: NP-eat-NP; eat - простое настоящее время, третье лицо. единственное число, активный залог. На третьем этапе происходит парсинг исходного предложения: (NP an apple) = прямое дополнение, зависимость от глагола “to eat”. Следует отметить, что возможны варианты, когда достаточно и частичного парсинга структуры предложения для создания карты структуры выходного текста. На четвертом этапе непосредственно происходит перевод английских слов на немецкий язык: a (категория: неопределенный артикль) = ein (категория: неопределенный артикль) girl (категория: существительное) = Mädchen (категория: существительное) eat (категория: глагол)= essen (категория: глагол) an (категория: неопределенный артикль)= ein (категория: неопределенный артикль) apple (категория: существительное)= Apfel (категория: существительное) На пятом этапе происходит генерация предложения на требуемом языке: изменяются словарные формы слов (применяются другое склонение, спряжение, число). Итогом становится перевод предложения на немецкий язык: A girl eats an apple => Ein Mädchen isst einen Apfel. Несмотря на такие положительные моменты, как синтаксическая и морфологическая точность, а также возможность настройки на предметную область, аналитический перевод требует постоянного поддержания и актуализации баз данных, длителен в разработке и игнорирует контекст [6]. Однако наиболее существенным преимуществом данного метода является отсутствие необходимости использования двуязычных текстов. Это позволяет создать систему перевода для таких языков, у которых нет общих текстов или какой-либо оцифрованной информации. Кроме того, созданную единожды систему машинного перевода, основанного на правилах, можно впоследствии использовать для перечислены все известные системе слова и фразы, варианты их перевода и вероятность этих переводов. Знания системы о языке представлены в виде вероятностной модели языка. Ее использование обусловлено необходимостью выбора тех или иных вариантов и связей в зависимости от контекста. Задача декодера заключается в подборе вариантов перевода для исходного текста,

сочетая между собой фразы из модели перевода и сортируя их по убыванию вероятности. Далее происходит оценивание получившихся вариантов с помощью модели языка[6]. В случае использования технологии статистического перевод возможно включение дополнительной лингвистической информации для языков с богатым словоизменением. К положительным сторона данного метода относят быструю настройку, экономию вычислительных ресурсов за счет исключения глубокого анализа текста, а также легкость, с которой системы справляются с переводом сложных и редких слов и терминов. Тем не менее, у этого подхода есть и существенные недостатки, вызванные, прежде всего, особенностями естественных языков. Низкое качество перевода отмечается у языков, принадлежащим к разным языковым семьям -- в таком случае необходимо использование сложных моделей типа tree-to-tree/tree-to-string (как в случае с английским и японским языками). Также на точности выбора лексических единиц сказывается и дефицит параллельных корпусов текстов. Помимо указанного, при использовании метода статистического машинного перевода возникают также следующие проблемы:

● статистические аномалии -- часто при внесении информации о реальном мире используются имена собственные, которые в дальнейшем могут быть ошибочно использованы при переводе, к примеру, предложение “I took the train to Paris” может быть ошибочно переведено как “Я сел на поезд в Берлин” из-за обилия вариантов “train to Berlin” в тренировочном наборе.

● Порядок слов в разных языках может существенно отличаться. Несомненно, можно синтаксически классифицировать слова в предложениях и получить обобщенную модель типа SVO (подлежащее-сказуемое-дополнение), но при этом сложно учитывать положение служебных частей речи и изменение порядка слов при смене типа предложения (например, на вопросительное). ● Слова, не вошедшие в словарь, возникают в виду недостатка данных при обучении, различиях в морфологии. Системы статистического перевода обычно хранят такие слова как отдельные символы без создания связи с другими словоформами или фразами [8]. Две указанные выше технологии развиваются и в настоящее время и сохраняют свою актуальность. Их переплетение и слияние породило

отдельный метод машинного перевода, получивший название гибридный. Утверждается, что гибридный метод машинного перевода способен использовать особенности как машинного перевода, основанного на правилах, так и статистического машинного перевода. Перечислим некоторые подходы:

● Правила, доработанные статистикой -- в данном случае словарный перевод улучшается и корректируется за счет статистических правил,

● Статистика, управляемая правилами -- правила используются для предварительной обработки информации, а также на этапе статистической коррекции результирующего перевода.

Вместе с этим на пике популярности находятся методы, использующие искусственные нейронные сети (neural machine translation). перевода обучаются совместно от начала до конца, чтобы максимизировать эффективность перевода [10]. Рассматривая проблемы машинного перевода, уместно отметить и явление культурологической непереводимости отдельных единиц перевода. Предполагается, что в случае нахождения возможности идентификации таких единиц в тексте перевода, станет возможен их анализ и внесение в базы данных программ перевода. В связи с этим Дж.К.Катфорд предлагает разработать алгоритмы, базирующиеся на правилах, позволяющих обращаться к контекстуальному значению. Для целей машинного перевода эти правила могут иметь вид операционных команд для текстуального поиска элементов, маркированных в машинном словаре специальными диакритиками с предписанием вывести в каждом конкретном случае обусловленный эквивалент. Точное выполнение таких алгоритмов, по мнению исследователя, может существенно повысить качество, корректность перевода. [11]

9 Лабораторная работа Линейная регрессия

1 Цель работы

Познакомится со средой выполнения MatLab и организацией линейной регрессии.

2 Ход работы

2.1 Постановка задачи

Дана обучающая выборка $(x^{(1)}, y^{(1)}, \ldots)$. Необходимо найти оптимальную функцию вида:

$$h(x) = a * x + b \qquad (1)$$

которая даёт наименьшую ошибку на обучающей выборке.

2.2 Считывание данных

Для начала следует запустить MatLab (или Octave) и в нём выполнить следующую команду:

```
cd('/home/user/ai/01');
```

Естественно, путь следует указать собственный, а именно каталог в котором будет располагаться программные файлы, а также файлы с входными данными. Создайте там файл input.txt который будет содержать 20 строк следующего вида:

```
1, 1.5
2, 2
3, 2.5
…
20, 11
```

Иными словами, речь здесь идёт о зависимости вида:

$$h(x) = 1 + 0.5 * x \qquad (2)$$

Наша ключевая задача – считать эти данные из файла. Для этого в командной строке нужно выполнить следующие действия:

```
data = load('input.txt');
x = data( : , 1);
y = data( : , 2);
m = length(y);
```

Первая команда считывает данные из файла и создаёт на их основе матрицу размера 20×2. Вторая команда создаёт переменную X

и записывает в неё первый столбец матрицы. Третья команда записывает в y второй столбец матрицы. Последняя строка вычитает размер вектора y.

По итогам выполнения в окне переменных должны отобразится четыре новых переменные.

2.3 Приведение к матричному виду

Чтобы упростить порядок вычислений, полезно прибегнуть к следующему приёму, а именно ввести параметр x_0, который для всех строк будет равен единице.

```
X = [ones(m,1), data( : , 1)];
```

Теперь у нас получаются следующие строки в таблице:

№	x_0	x_1	y
1	1	1	1.5
2	1	2	2
3	1	3	2.5
20	1	20	11

Таблица 1: строки обучающей выборки после введения базового элемента

2.4 Формализация задачи

Теперь поставим задачу следующим образом. Пусть у нас имеется вектор θ из двух чисел. Тогда для любой строки $x^{(i)}$ (которая теперь содержит два элемента) $\theta * x^{(i)}$ даёт число (речь идёт об операции произведения матрицы на вектор в том смысле, в котором эта операция определяется в линейной алгебре). Задача состоит в том, чтобы минимизировать функцию стоимости:

$$J(\theta) = \frac{1}{2 * m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \qquad (3)$$

где

$$h_\theta(x) = \theta^x * x = \theta_0 + \theta_1 * x_1 \qquad (4)$$

Напоминаю, что для всех строк:

$$x_0 = 1 \qquad (5)$$

Добиться этого можно с помощью постепенного спуска по итерациям, используя следующую формулу:

$$\theta_j = \theta_j - \frac{\alpha * 1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) * x_j^{(i)} \qquad (6)$$

2.5 Реализация функции стоимости

Создайте в каталоге файл compute Cost.m следующего содержания:

```
function J = compute Cost(X, y, theta)
m = length(y); % number of traning examples
J = 0;
for i = 1:m
        delta = y(i)-X(i,:)*theta;
        J = J + delta*delta;
end
J = J/(2*m)
end
```

Данный код реализует напрямую цикл выражение 3. Стоит обратить внимание, что здесь используется матричное произведение.

2.6 Реализация функции градиентного спуска

Создайте файл в каталоге gradientDescent.m следующего содержания:

```
function [theta, J_history] = gradientDescent(X, y, theta, alpha, num_iters)
        m = length(y); % number of training examples
J_history = zeros(num_iters, 1);
n = length(theta);
for iter = 1:num_iters
```

```
theta = theta - (alpha/m)*X'*(X*theta-y);
J_history(iter) = computeCost(X, y, theta);
end
end
```

2.7 Проверка градиентного спуска

Теперь в командной строке выполните следующие инструкции:

```
theta = zeros(2, 1);
theta = gradientDescent(X, y, theta, 0.01, 500);
```

Данный код будет выполнять градиентный спуск с шагом 0.01 и за 500 итераций. Проверьте, каков будет итоговый theta в результате выполнения данного кода.

2.8 Введение дополнительных столбцом

Предположим, что мы хотим провести линейную регрессию от одной переменной X, однако предполагаем, что на самом деле зависимость выражается полиномом более высокого порядка, например:

$$f(x) = x^2 + 2 * x + 1(7)$$

Для того, чтобы разрешить такую проблему, достаточно ввести новый столбец x_2, который будет содержать квадраты x_1 (то же самое можно проделать для кубов и более высоких степеней). Конкретно для нашей задачи это можно сделать так:

```
X = [ones(m,1), data(:,1), data(:,1).^2];
```

Выражение:

x.^2

в MatLab означает не то, что матрица умножается сама на себя, а то, что каждый элемент матрицы возводятся в кважрат.

3 Вариант заданий

В каждом варианте заданий необходимо выполнить, что описано в предыдущем разделе, а также реализовывать линейную регрессию дополнительного выражения:

1. $z = x^2 + y^2$
2. $z = x^2 + x \cdot y + 1.$
3. $z = x^2 + y^2 + 2 \cdot x \cdot y.$
4. $z = x^3 + y^2.$
5. $z = x + y/x$.
6. $z = x + \log(y^2)$.
7. $z = sin(y) + x.$
8. $z = cos(x) + sin(y).$
9. $z = cos(x)^2 + sin(y).$
10. $z = x^2 + 2 \cdot x + y.$

Для каждого варианта необходимо создать файл из 100 строк, содержащий значения входных переменных и выходной переменной. Следует провести регрессию как с введением дополнительных значений, так и без них.

4 Содержание работы

- Цель работы.
- Вариант задания.
- Исходный текст программы.
- Результаты выполнения.
- Выводы.

5 Контрольные вопросы

1. Как именно влияет параметр α на характеристики алгоритма линейной регрессии?
2. Что будет если сделать этот параметр слишком большим?

3. Что будет если сделать этот параметр слишком маленьким?

4. Почему мы стремимся свести количество циклов к минимуму в MatLab, и вместо этого использовать матричные произведения и обработку векторов.

Лабораторная работа 2. Кластеризация

1 Цель работы

Познакомиться с методами обучения без учителя.

2 Ход работы

2.1Постановка задачи

В предыдущих работах мы рассматривали обучающие выборки, где каждому входному значению сопоставлялось выходное значение. На основе этой выборки мы пытались построить систему, пытающуюся предсказать выходные значения для комбинаций, которые не встречались в обучающей выборке.

В данной работе мы рассмотрим ситуацию, когда выходных значений нет. Т.е. нам просто дан набор строк размера n, которые можно рассматривать как точки в n-мерном пространстве, и нам интересно, не существует ли каких-то скрытых закономерностей среди этих точек. Наиболее очевидная закономерность в данном случае – не принадлежат ли эти точки отдельным группам, которые четко отличаются друг от друга. Эти группы назовем *кластерами*, а сам процесс определения того, каким группам принадлежат точки, будем называть *кластеризацией*.

В методе К-средних алгоритм действует следующим образом. Ему дан набор из m точек в n-мерном пространстве, а также число k – количество кластеров, на которое надо разбить эти точки. Алгоритм должен найти k точек в пространстве, которые мы обозначим как μ_i – это будут центры кластеров. Кроме того, алгоритм должен сопоставить каждой точке из исходных индекс от 1 до k, т.е. определить, какому кластеру принадлежит эта точка. Обозначим индекс точки x_i как $c(x_i)$. Задача – найти центры кластеров и назначить точки таким образом,

чтобы их сумма квадратов расстояний от точек до их центров была минимальной:

$$\sum_{i=1}^{m} \vee x_i - \mu_{c(x_i)} \vee^2 \quad (1)$$

Зачем вообще может понадобиться кластеризация? Это может понадобиться в ряде задач. Например, для сегментации рынка, для выявления различных групп пользователей в социальных сетях. Это также может использоваться для сжатия изображений. Например, у нас есть изображение, где каждый цвет описывается с помощью представления RGB. Мы можем сжать его, скажем, до 256 цветов, попробовав найти 256 оптимальных кластеров, и их центры использовать как цвета для представления исходного изображения. Это особенно хорошо работает для случаев, когда в исходном изображении есть регионы примерно похожего цвета (например, тел или предметов), и в силу особенностей нервной системы и оптического восприятия человек даже не заметит разницы после сжатия.

2.2Подготовка выборки

Для примера создадим набор точек, где точки будут принадлежать трем группам:

- Кругу с радиусом 1 и центром в точке {0,0}.
- Кругу с радиусом 1 и центром в точке {4,4}.
- Кругу с радиусом 1 и центром в точке {2,2}.

Сделаем это с помощью следующего кода:

```
        X=
ze-
ros(150,2
); for i =
1:50
        X(i,1) =
rand()-0.5;
X(i,2)          =
rand()-0.5;
end

        for i = 51:100
        X(i,1) =
rand()+3.5;
X(i,2)          =
rand()+3.5;
end

        for i = 101:150
        X(i,1) = rand()+1.5;
        X(i,2) = rand()+1.5;
        end


        plot(X(:,1), X(:,2), 'ko', 'MarkerFaceColor', 'y',
'MarkerSize', 7);
```

Результат будет выглядеть так, как показано на рисунке ниже. Для человека очевидно, что эти точки разделяются на три различные группы. Вопросы в том, как заставить программу это определить.

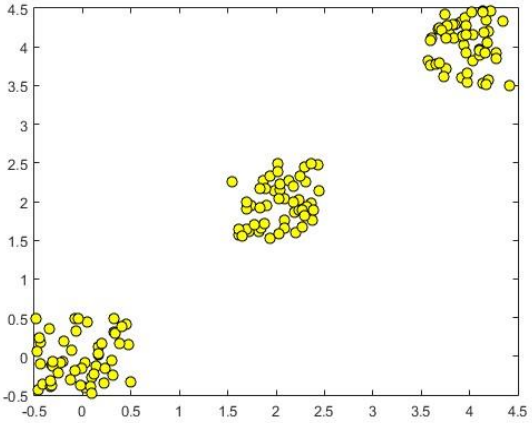


Рисунок 5-Исходная выборка

2.3Реализация алгоритма

Необходимо реализовать три основные функции. Первая, располагая точками x и уже имеющимся центрами кластеров, определяет, к какому центру ближе всего расположена эта точка:

```
function idx = findClosestCentroids(X, centroids)

K = size(centroids, 1);
idx = zeros(size(X,1), 1);


for i=1:size(X,1)
idx(i) = 1;
value = sum((X(i,:)-centroids(1,:)).^2); for
j = 2:K
value1 = sum((X(i,:)-centroids(j,:)).^2);
if value1 < value idx(i)
= j; value = value1;
end
end
end
end
```

В принципе здесь можно было бы избежать части циклов, но в данном случае код написан в таком стиле, чтобы было более понятно. Далее, вторая функция высчитывает новые центры на основе выборки x:

```
function centroids = computeCentroids(X, idx, K)
[m n] = size(X);

centroids = zeros(K, n);

for i = 1:K idx1
= (idx==i); s =
idx1'*X;

centroids(i,:) = s(:)/sum(idx1);

end
end
```

.

Здесь используется формула:

$$\mu_k = \frac{1}{C_k \vee \sum_{i \in C_k} x_i} \quad (2)$$

где C_k – индексы точек, которые принадлежат k-му кластеру. Иными словами, новый центр вычисляется как геометрическое среднее между точками, близкими к старому центру. Ну и основной алгоритм здесь выглядит так:

```
function [centroids, idx] = runkMeans(X, initial_centroids, ...
max_iters)

% Initialize values

[m n] = size(X);

K = size(initial_centroids,
1); centroids = initial_centroids;
idx = zeros(m, 1);

% Run K-
Means for
i=1:max_iters

% For each example in X, assign it to the closest centroid
idx = findClosestCentroids(X, centroids);

% Given the memberships, compute new centroids
centroids = computeCentroids(X, idx, K);

end
end
```

Как получить начальные центры? Их можно задать вручную, но можно сделать это с помощью следующей функции, выбравшей случайным образом k точек из уже имеющихся:

```
function centroids = kMeansInitCentroids(X, K)
randidx = randperm(size(X,1));
centroids = X(randidx(1:K), :);
```

Теперь попробуем вычислить центры кластеров, вызвав основной метод:

```
centroids = runkMeans(X, kMeansInitCentroids(X, 3), 400);
```

Ответ варьируется в зависимости от того, как были выбраны центры, но должно получиться что-то вроде этого:

```
2.0319;1.9701
3.9521;4.0336
-0.0258;-0.0175
```

2.4Визуализация кластеров

Теперь напишем функцию, которая, зная точки x и индексы кластеров, покажет это визуально:

```
function plotDataPoints(X, idx, K)
% Create palette
palette = hsv(K +
1);
colors = pal-
ette(idx, :);
% Plot the data
scatter(X(:,1), X(:,2), 15, col-
ors);
end
```

И вызовем функцию следующим образом:

```
plotDataPoints(X, findClosestCentroids(X, centroids), 3)
```

Получается результат, как на рисунке:

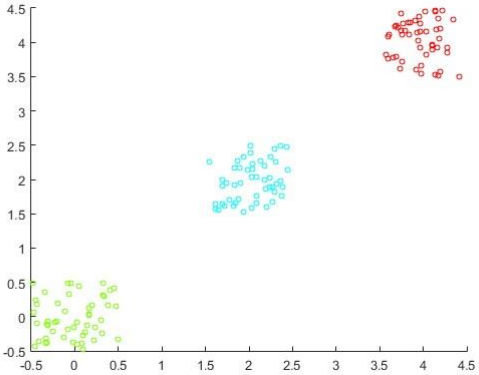


Рисунок 6-Результат разбиения на кластеры

Как видно, алгоритм действительно правильно разбил на группы точки.

3 Варианты задания

В каждом задании необходимо выполнить пример и убедиться, что кластеризация действительно работает. Также необходимо выполнить собственное задание:

1. Точки должны быть в трехмерном пространстве и располагаться внутри двух кубов с ребрами, равными единице, но в различных центрах.
2. Точки должны быть в трехмерном пространстве и располагаться внутри четырех кубов с ребрами, равными единице, но в различных центрах.
3. Точки должны быть в трехмерном пространстве и располагаться внутри трех шаров с радиусами, равными единице, но в различных центрах.
4. Точки должны быть в трехмерном пространстве и располагаться внутри пяти шаров с радиусами, равными единице, но в различных центрах.
5. Точки должны быть в трехмерном пространстве и располагаться внутри семи шаров с радиусами, равными единице, но в различных центрах.
6. Точки должны быть в двухмерном пространстве и располагаться внутри двух квадратов с ребрами, равными единице, но в различных центрах.

7. Точки должны быть в двухмерном пространстве и располагаться внутри четырех квадратов с ребрами, равными единице, но в различных центрах.

8. Точки должны быть в двухмерном пространстве и располагаться внутри трех кругов с радиусами, равными единице, но в различных центрах.

9. Точки должны быть в двухмерном пространстве и располагаться внутри пяти кругов с радиусами, равными единице, но в различных центрах.

10. Точки должны быть в двухмерном пространстве и располагаться внутри семи кругов с радиусами, равными единице, но в различных центрах.

Необходимо проверить для различного количества кластеров, какие результаты будут показаны.

4 Содержание работы

- Цель работы.
- Вариант задания
- Исходный текст программы.
- Результаты выполнения.
- Выводы.

5 Контрольные вопросы

1. Какова сигнатура функции scatter. Что означает 15 при ее вызове?
2. Как правильно определить какое именно количество кластеров имеется в выборке?
3. Как получить начальные центры кластеров?
4. Объясните как работает алгоритм К-средних.

Лабораторная работа 11

КЛАССИФИКАЦИЯ

Цель работы

Познакомиться с методами классификации (логистической регрессии).

Ход работы

2.1Постановка задачи

В предыдущей работе мы рассматривали обучающие выборки, где нам давались выходные значения *y*, и нам было интересно предсказать, каково будет значение *y* для вариантов, не встречающихся в выборке. Задача логистической регрессии возникает в том случае, когда *y* имеет фиксированный набор значений, и не может принимать каких-то значений не из этого набора. В предельном случае, *y* может принимать только два значения - {0,1}. Т.е. перед нами стоит задача что-то отнести к одному из двух классов. Рассмотрим различные примеры:

· У нас есть набор рисунков, на части из которых изображены котята, а на остальных нет. Мы можем привести все рисунки к одинаковой размерности (например, 256×256) и монохромному изображению (т.е. каждая точка определяется яркостью по шкале от 0 до 255). Тогда каждый рисунок будет примером, содержащим 256^2 столбцов, где каждый столбец будет принимать значения от 0 до 255. Каждому из рисунков мы сопоставим выходное значение 1, если на нем изображены котята, и 0 в противном случае. Задача состоит в том, чтобы обучить систему распознавать наличие котят на рисунках не из выборки.

У нас имеется набор писем, каждое из которых либо спам, либо нет. Каждое письмо можно преобразовать в набор элементов. Например, мы можем взять словарь и подсчитать, сколько раз встречается каждое слово в письме. Тогда каждое письмо можно представить как вектор, где каждый элемент будет соответствовать

определенному слову. Задача обучить систему классифицировать новые письма, являются ли они спамом или нет.

Для начала создадим обучающую выборку. В качестве таковой мы будем использовать точки на двумерной плоскости (координаты x_1 и x_2), а выходное значение определим следующим образом:

$$y = \begin{cases} 1, x_2 \geq x_1 \\ 0, x_2 < x_1 \end{cases} \quad (1)$$

Несложно видеть, что фактически плоскость разделена на две полуплоскости прямой, проходящей через центр координат. Для начала создадим обучающую выборку на 100 строк:

```
X = zeros(100, 2);
Y = zeros(100, 1);
```

Далее заполним эту выборку:

```
for i = 1:100
X(i,1) = rand();
X(i,2) = rand();
Y(i) = X(i,2)>=X(i,1);
end
```

Теперь попробуем нарисовать график:

```
hold on;
pos = find(Y==1);
neg = find(Y==0);
plot(X(pos,1), X(pos,2), 'k+', 'LineWidth', 2, 'MarkerSize', 7);
```

```
plot(X(neg,1), X(neg,2), 'ko', 'MarkerFaceColor', 'y', 'MarkerSize', 7);


hold off;
```

Обратите внимание на несколько интересных особенностей. Мы находим индексы *pos* и *neg* с помощью специальной команды. Далее мы указываем функции *plot*, какие именно строки обучающей выборки нарисовать разными символами. Если все проделано корректно, то будет показан рисунок ниже (он может несколько отличаться из-за случайности исходного набора):

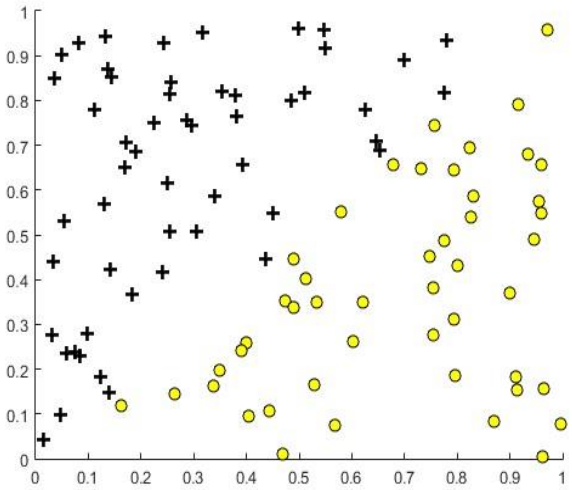


Рисунок 7-Пример обучающей выборки.

10 Варианты заданий

3.1Регрессия сигмоидной функции

Итак, что же делать с этой обучающей выборкой? Мы рассмотрим здесь новую функцию:

$$h_\theta(x) = g(\theta^T \cdot x) \quad (2)$$

где g - это сигмоид:

$$g(z) = \frac{1}{1+e^{-z}} \quad (3)$$

Схематично сигмоид представлен на рисунке ниже:

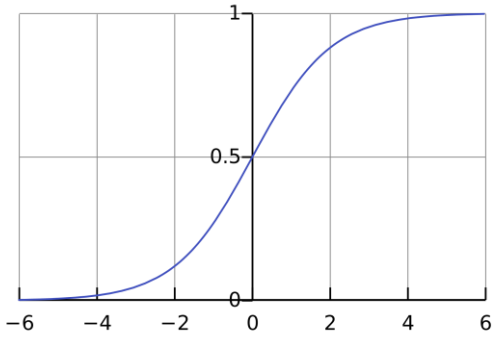


Рис. 2: Сигмоидная функция.

Несложно видеть, что для $x = 0$ эта функция равна 0.5, для положительных значений x она стремится к 1, а для отрицательных значений она стремится к 0. Эта функция на выходе всегда дает значения в интервале [0;1].

Фактически значение $g(x)$ можно трактовать как вероятность того, что y для данного x будет равен 1. Мы введем пороговое значение, и будем считать, что если $h_\theta(x) \geq 0.5$, то система классифицирует данный пример как положительный ($y = 1$).

Тогда функция стоимости может быть вычислена следующим образом:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \tag{4}$$

Несложно видеть, что эта сумма обладает несколькими характерными особенностями. По сути она состоит из двух частей. Первая, это где $y^i = 1$. Эта часть равна:

$$\frac{-1}{m}\sum_{i=0}^{m} \tag{5}$$

Из особенностей логарифма понятно, что чем ближе $h_\theta(x)$ к единице, тем ближе к нулю логарифм от этого значения, и тем меньше вклад в ошибку. Идеальным является случай, когда для всех положительных примеров обучающей выборки $h_\theta(x)$ будет давать значение, равное одному.

Аналогично рассуждая для отрицательных примеров, мы приходим к выводам, что $J(\theta)$ будет равна нулю только в том случае, когда $h_\theta(x)$ будет выдавать единицу для положительных примеров и 0 для отрицательных примеров. Т.е. попытка минимизировать эту

функцию приведет нас к более менее стабильному классификатору, т.е. к такому набору θ, который можно будет использовать на практике.

Частные градиенты для этой функции вычисляются следующим образом:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left[\left(h_\theta\left(x^i\right) - y^i\right) \times x_j^i\right] \quad (6)$$

В этом несложно убедиться, если применить стандартные правила дифференцирования, что и позволяет нам произвести градиентный спуск. Для этого мы напишем специальную функцию, которая возвращает как стоимость для текущей точки, так и значения градиентов.

```
function [J, grad] = costFunction(theta, X, Y)

% Initialize some useful values

m = length(Y); % number of train-
ing examples

% Calculate the sigmoids

sigmtable = sig-
moid(X*theta);

% Calculate the cost for all examples

J = -(Y'*log(sigmtable)+(1-Y)'*log(1-sigmtable))/m;

% Calculate the gradients for all columns at the
same time

grad = X'*(sigmtable-Y)/m;

end
```

Эта функция использует технику векторизации, т.е. она избегает циклов. Функцию *sigmoid* следует реализовать самостоятельно согласно формуле 3.

Теперь можно найти оптимальное значение θ с помощью следующего кода:

```
options = optimset('GradObj', 'on', 'MaxIter', 400);

[theta, cost] = fminunc(@(t)(costFunction(t,X,Y)), rand(3,1), options);
```

Данный код использует встроенную функцию *minunc*, которая позволяет найти локальный минимум. Но как удостовериться, что это действительно хороший классификатор? Для этого напишем функцию, которая будет классифицировать входные значения с помощью θ:

```
function p = predict(theta, X)

m = size(X, 1); % Number of training examples

p = zeros(m, 1);

prediction = sigmoid(X*theta);

positive = find(prediction>=0.5);

negative = find(prediction<0.5); p(positive,1)=1; p(negative,1)=0;

end
```

Имея этот код, можно вычислить количество совпадений между выходными значениями обучающей выборки, и классификатором:

```
sum(predict(theta, X) == Y)
```

Если все сделано правильно, то классификатор выдаст значение 100.

3.2Регуляризация

Предыдущий раздел не накладывал никаких ограничений на θ. Это может привести к тому, что классификатор будет либо недообучен (т.е. не для всех примеров обучающей выборки он выдает правильный ответ), либо переобучен (т.е. он дает правильный ответ для всех примеров обучающей выборки, но не более того). Значение 100 указывает на то, что недообученности не наблюдается. Однако давайте попробуем использовать классификатор на значениях не из обучающей выборки, и сгенерируем 100 строк случайно:

```
        Xcheck = zeros(100,
3);

        Ycheck = zeros(100,
1);

        for i = 1:100

        X1 = rand()*100;

        X2 = rand()*100;

        Xcheck(i,1) = 1;

        Xcheck(i,2) = X1;

        Xcheck(i,3) = X2;

        Ycheck(i)       =
X2>=X1;

        end
```

Если теперь запустить код:

```
        sum(predict(theta, Xcheck) == Ycheck)
```

то можно увидеть число меньше 100, что является следствием переобученности. Чтобы этого избежать, введем регуляризацию и специальный параметр λ. Тогда стоимость ошибки будет вычисляться так:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \quad (7)$$

Обратите внимание, что здесь не учитывается θ_0, и что чем больше значение всех остальных элементов θ, тем больше и ошибка, т.е. теперь мы пытаемся уменьшить значение θ по возможности. Градиент для x_0 не изменился, а вот для всех остальных он вычисляется как:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left[\left(h_\theta\left(x^i\right) - y^i\right) \times x_j^i\right] + \frac{\lambda}{m} \times \theta_j \quad (8)$$

Реализуем эту функцию следующим образом:

```
function [J, grad] = costFunctionReg(theta, X, y, lambda)

% Initialize some useful values m
= length(y); % number of training exam-
ples

% You need to return the following variables
correctly

J = 0;

grad = ze-
ros(size(theta));

n = size(theta);

sigmtable = sig-
moid(X*theta);

J = -(y'*log(sigmtable)+(1-y)'*log(1-sigmtable))/m +
(lambda/(2*m))*sum(theta(2:n).^2);

thetared = theta;

thetared(1,1) = 0;

grad = X'*(sigmtable-y)/m +
(lambda/m)*thetared;

end
```

Чтобы вычислить теперь значение θ, можно запустить следующий код:

```
[theta, cost] = fminunc(@(t)(costFunctionReg(t,X,Y,0.01)), rand(3,1), options);
```

Как вообще влияет λ на результат? Для столь простого примера это не будет особо заметно. Однако для случая, когда классификатором выступает отнюдь не прямая, а полином более высокого порядка, различие показано на рисунках ниже:

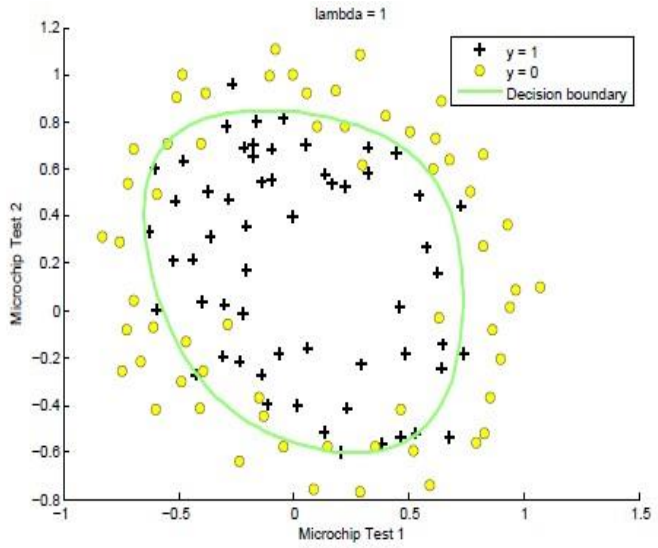


Рисунок 8- Нормальный классификатор ($\lambda = 1$).

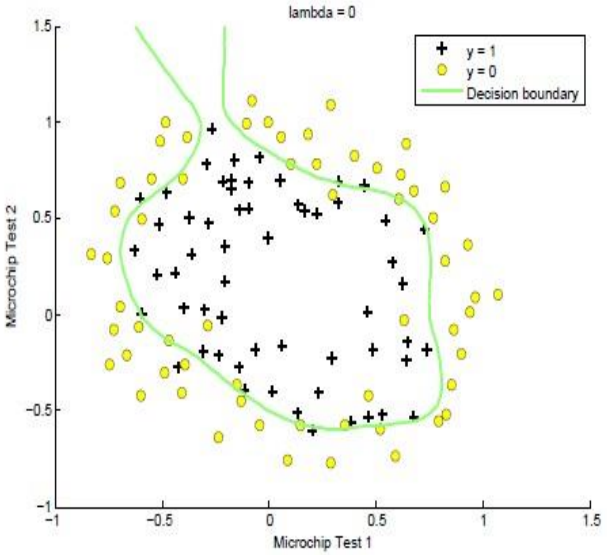


Рисунок 9- Переобученный классификатор (λ = 0).

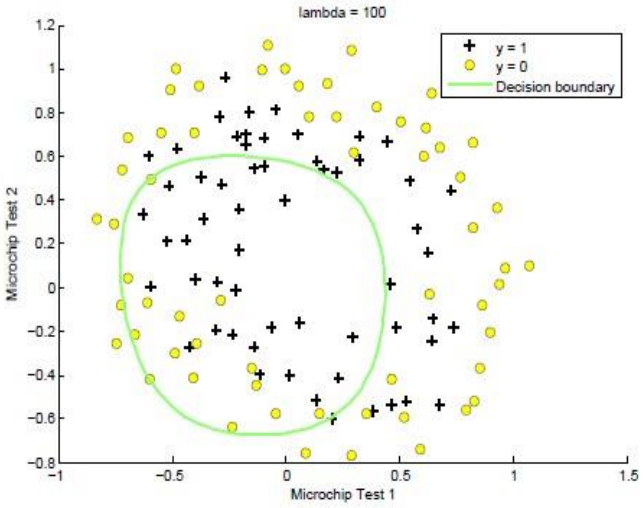


Рисунок 10-Недообученный классификатор (λ = 100).

Варианты задания

В каждом варианте необходимо выполнить все примеры, приведенные выше. Кроме того, надо построить классификатор для собственного варианта:

1. $x^2_1 + x^2_2 - 4 \geq 0.$
2. $x^3_1 + x_2 + 1 \geq 0.$
3. $x^3_1 + x^2_2 + 1 \geq 0.$
4. $x^2_1 + x_2 + x_1 \geq 0.$
5. $sin(x_1) + cos(x_2) \geq 0.$
6. $sin^2(x_1) + cos^2(x_2) - 1 \geq 0.$
7. $x^4_1 + x^2_2 + x_1 \geq 0.$

8. $x^4_1 + x^2_2 - 3 \geq 0$.

9. $\log(x^2_1 + x^2_2) \geq 0$.

10. $\log(x^2_1 + x^4_2) \geq 0$.

Для полученного классификатора надо проверить, насколько точно он распознает обучающую выборку, и насколько хорошо он ведет себя на примерах не из обучающей выборки. Сделать это следует с разными параметрами регуляризации. Также нужно показать график для обучающей выборки, как именно он выглядит.

Содержание работы

· Цель работы.

· Вариант задания.

· Исходный текст программы.

· Результаты выполнения.

· Выводы.

Контрольные вопросы

1. Объясните смысл функции *fminunc*. Что именно мы ей передаем, что означает @(*t*) в примерах выше, и все остальные аргументы?
2. Объясните смысл функции *find*. Что именно она принимает на вход и что возвращает?
3. Какие вы знаете параметры регуляризации.
4. При каких условиях применяется задача логистической регрессии?

Лабораторная работа 2

Регрессия для множества классов

Цель работы

Познакомиться с методами многокритериальной классификации

Ход работы

2.1 Постановка задачи

В одной из предыдущих лабораторных мы рассматривали задачи классификации, где постановка была сознательно упрощена для ознакомления с предметом. Вот два ключевых недостатка логистической регрессии из предыдущей лабораторной работы:

- Регрессия способна ответить лишь «ДА/НЕТ», чаще же всего интересуют случаи, когда классов для выходных переменных может быть несколько (например, пять различных видов изображений).
- Еще хуже то, что классификатор является линейным и не может классифицировать сложные нелинейные случаи. Можно попробовать добавить дополнительные столбцы в матрицу за счет комбинаций переменных, но это становится крайне дорогим для обучения.

В данной работе будет рассмотрена проблема классификации для множества возможных классов на примере такой задачи, как распознавание текста.

2.2 Входные данные

В данной работе используется дополнительный файл digits.mat, который содержит часть данных из MNIST. Он содержит 500 примеров для обучения распознавания цифрам (которые написаны с различными наклонами). Чтобы его загрузить, нужно выполнить в каталоге, где расположен этот файл, следующую команду:

```
> load('digits.mat');
```

В результате мы увидим две переменные X и Y в окне среды, где X имеет размерность 500x400, а Y имеет размерность 500x1. X является представлением рисунка размером 20x20 пикселей в

черно-белом представлении, где у каждого пикселя задана яркость изображения. Что касается Y, то цифра 0 представлена значением 10 для упрощения работы MatLab (Octave), а цифры от 1 до 9 представлены напрямую.

2.3 Отображение входных данных

Интересный вопрос заключается в том, как заставить MatLab отображать эти наборы в виде реальных рисунков. Для этого напишем соответствующую функцию:

```
function [retval] = displayRow (row)
% Compute the size of the matrix
n = floor(sqrt(size(row, 2)));
% Declare the matrix
matr = zeros(n,n);
% Fill the matrix from the linear row
for i = 1:n
    for j = 1:n
        matr(i,j) = row((i-1)*n+j);
    endfor
endfor

% Gray Image
colormap(gray);
% Draw the image itself
imagesc(matr);
% Do not show axis
axis image off

% Return just zero
retval = 0;
endfunction
```

Несложно видеть, что функция состоит из двух частей. Первая часть преобразует строку из 400 элементов в матрицу размера 20x20.

Вторая часть использует встроенную функцию imagesc, которая отображает набор пикселей, в данном случае в монохромном формате (где каждый пиксель обозначается его яркостью). Для примера попробуем отобразить одну из строк:

```
> displayRow(X(1,:));
```

Результат должен получиться такой, как показано на рисунке ниже:

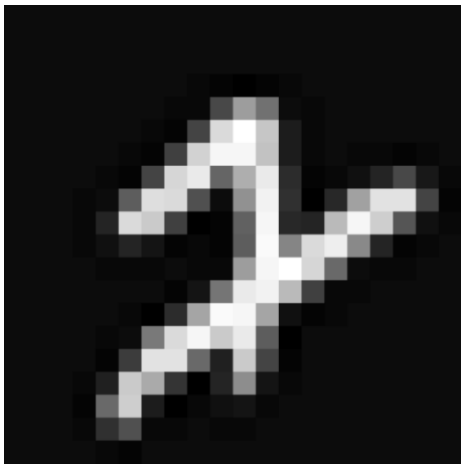


Рисунок *11*-отображение цифры.

2.4 Обучающий классификатор

Теперь напишем основную функцию, которая позволит нам получить сразу множество классификаторов для всех возможных классов значений Y:

```
function [all_theta] = oneVsAll(X, y, num_labels, lambda)

% Some useful variables
m = size(X, 1);
n = size(X, 2);
% You need to return the following variables correctly
all_theta = zeros(num_labels, n + 1);
% Add ones to the X data matrix
X = [ones(m, 1) X];
options = optimset('GradObj', 'on', 'MaxIter', 50);
initial_theta = zeros(n+1,1);
for i = 1:num_labels
```

```
        all_theta(i,:) = fmincg (@(t)(costFunctionReg(t, X, (y == i), lambda)), ...
                        initial_theta, options);
    endfor
    end
```

Логика очевидна - мы указываем количество возможных классов (от 1 до 10), и вычисляем для каждого класса его значение θ. Чтобы это запустить, нужно выполнить следующую команду:

> all_theta = oneVsAll(X,Y,10, 0.001);

2.5 Предсказание с помощью обучающего классификатора

Теперь напишем функцию, которая для набора строк для каждой строки вернет наиболее подходящий класс:

```
function p = predictOneVsAll(all_theta, X)
m = size(X, 1);
num_labels = size(all_theta, 1);

% Add ones to the X data matrix
X = [ones(m, 1) X];
result = X*all_theta';
[M,I] = max(result, [], 2);
p = I;
end
```

Чтобы определить точность, нужно подсчитать количество совпадений между результатом этой функции и исходным Y. Это в качестве самостоятельного задания.

3 Варианты задания

Для каждой задачи нужно составить скрипт, который подготовит тестовый набор данных и отобразит его в 2-х или 3-х мерной плоскости.

1. Имеются две входные переменные x и y. Тестовый набор выглядит следующим образом:

$$class(x,y)=\begin{cases}0, x\le 0 \vee x\ge 2\\ 1, y\le 0 \wedge x\in[0;2]\\ 2, otherwize\end{cases} \quad (1)$$

2. Имеются две входные переменные x и y. Тестовый набор выглядит следующим образом:

$$class(x,y)=\begin{cases}0, x\le 0 \vee x\ge 2\\ 1, y^2\le 100 \wedge x\in[0;2]\\ 2, otherwize\end{cases} \quad (2)$$

3. Имеются две входные переменные x и y .Тестовый набор выглядит следующим образом:

$$class(x,y)=\begin{cases}0, x^2+y^2\le 1\\ 1, x^2+y^2\in[0;2]\\ 2, otherwize\end{cases} \quad (3)$$

4. Имеются две входные переменные x и y. Тестовый набор выглядит следующим образом:

$$class(x,y)=\begin{cases}0, x\in[0;2]\\ 1, y\le 0 \wedge x\notin[0;2]\\ 2, otherwize\end{cases} \quad (4)$$

5. Имеются две входные переменные x и y. Тестовый набор выглядит следующим образом

$$class(x,y)=\begin{cases}0, x\in[0;2]\\ 1, y^2\le 100 \wedge x\notin[0;2]\\ 2, otherwize\end{cases} \quad (5)$$

6. Имеются две входные переменные x и y. Тестовый набор выглядит следующим образом:

$$class(x,y)=\begin{cases}0, x^2+y^2\le 1\\ 1, x^2+y^2\in[1;3]\\ 2, otherwize\end{cases} \quad (6)$$

7. Имеются две входные переменные x и y. Тестовый набор выглядит следующим образом:

$$class(x,y)=\begin{cases}0, sin(x)\le cos(y)\\ 1, sin^2(x)+cos^2(y)\le 1\\ 2, otherwize\end{cases} \quad (7)$$

8. Имеются две входные переменные x и y. Тестовый набор выглядит следующим образом:

$$class(x,y) = \begin{cases} 0, sin^2(x) + cos^2(y) \in [0; 0.25] \\ 1, sin^2(x) + cos^2(y) \in [0.25; 0.75] \\ 2, otherwize \end{cases} \quad (8)$$

9. Имеются две входные переменные x и y. Тестовый набор выглядит следующим образом:

$$class(x,y) = \begin{cases} 0, x \leq lny \\ 1, x \leq 2 * lny \\ 2, otherwize \end{cases} \quad (9)$$

10. Имеются две входные переменные x и y. Тестовый набор выглядит следующим образом:

$$class(x,y) = \begin{cases} 0, ln(x * y) \leq 5 \\ 1, ln(x) + ln(y) \leq 5 \\ 2, otherwize \end{cases} \quad (10)$$

4 Контрольные вопросы

1. Что возвращает функция *max* от трех аргументов в *predictOneV s All?*

2. Дайте определение многокритериальной классификации.

3. Как определить точность распознавания?

4. Что такое функция стоимости и для чего она нужна?

11 Лабораторная работа Эволюционные алгоритмы

1 Цель работы

Познакомится с эволюционными алгоритмами.

Генетические алгоритмы в той или иной вариации были известны, как минимум с 50-хгг. прошлого века, однако по большей части эти алгоритмы опирались на операцию одной лишь мутации, что не могло привести эти алгоритмы к успеху в общем случае. В 1975 году Холланд показал, как одновременное использование операции кроссовера и мутации в ходе эволюции может привести к нахождению успешного решения множества задач. В своей работе Холланд представил математическое обоснование того, почему эволюция успешна именно в том случае, когда использует именно мутации и скрещивание особей.

Перечислим основные этапы решения задачи с помощью генетического алгоритма:

- На первом шаге выбирается схема представления особи. Схема должна однозначно позволять восстанавливать из генетического описания особи то решение, которое мы ищем. Например, пусть мы хотим представить в виде набора генов формулу $f(x)=a\,x^2+bx+c$. Одним из способов представления будет такое- пусть особь состоит из 192 бит. Первые 64 бита соответствуют представлению числа a в двоичном коде, следующие 64 бита соответствуют b, и последнии 64 бита соответствуют c. Выбор формата в данном случае является произвольным и опирается на стандартную архитектуру Intel 64 – разрядных платформ, и может отличатся от реальной задачи. Важно то, что по представлению особи можно однозначно восстановить функцию f(x), которой она соответствует с конкретными коэффициентами. Кроме того, можно избежать чрезвычайно больших и чрезвычайно малых значений коэффициентов, ограниченных каким-то диапазоном. Пусть заранее известно, что a принадлежит диапазону от 0 до 100. Тогда значение a (которое представлено числом a' от 0 до 2^{64}-1) можно вычислить как $100(a'/2^{64})$.
- На втором шаге генерируется начальная популяция определенного размера N. Обычно N выбирают в диапазоне от 500 до нескольких тысяч. Начальная популяция генерируется с помощью генератора случайных чисел. После начинается процесс эволюции. Какие генераторы следует использовать зависит от конкретной задачи. Например, в качестве генератора можно использовать равномерное распределение в диапазоне от 0 до 100, если можно предположить заранее, что для решения все коэффициенты будут укладываться в этот диапазон.
- Для каждой популяции в особи вычисляется ее фитнесс-функция, или, другими словами, численное представление приспособленности конкретной функции.
- На основе текущего поколения создается новое. Чем выше приспособленность особи, тем больше ее шансы попасть в следующие.
- При операции простого клонирования особь переходит в новое поколение без изменений. Если ограничиться только этой операцией, то в ходе эволюции популяция придет к результату, когда

все особи являются копией, наиболее приспособленной из тех, которыми мы располагали изначально.

- При операции мутации особь переходит в новое поколение с незначительными изменениями - т.е. небольшое количество битов в представляемой особи поменяет знак на противоположный – с нуля на единицу или наоборот. В этом случае эволюция может находить оптимальные решения, но это будет происходить медленно и поэтому нужно использовать третью операцию.
- При операции кроссовера из поколения выбираются две особи. Особи не должны быть одинаковы, т.к. в этом случае кроссовер не даст эффекта. Затем выбирается точка разрыва. Начало первой особи склеивается с концом второй особи или наоборот. Полученные потомки переходят в новое поколение.

Виды фитнесс – функций.

Стандартная: $f_s(x)=max- f_r(x)$

Смещенная: $f_a(x)=1/1+f_s(x)$

Нормализованная: $f_n(x)=f_a(x)/\sum f_a(x_i)\ \ i=1....N$

2 Ход работы

Для начала создадим изображение земли черного цвета:

```
earth = zeros(100, 3);
imshow(earth);
```

Результат представлен на рисунке ниже:

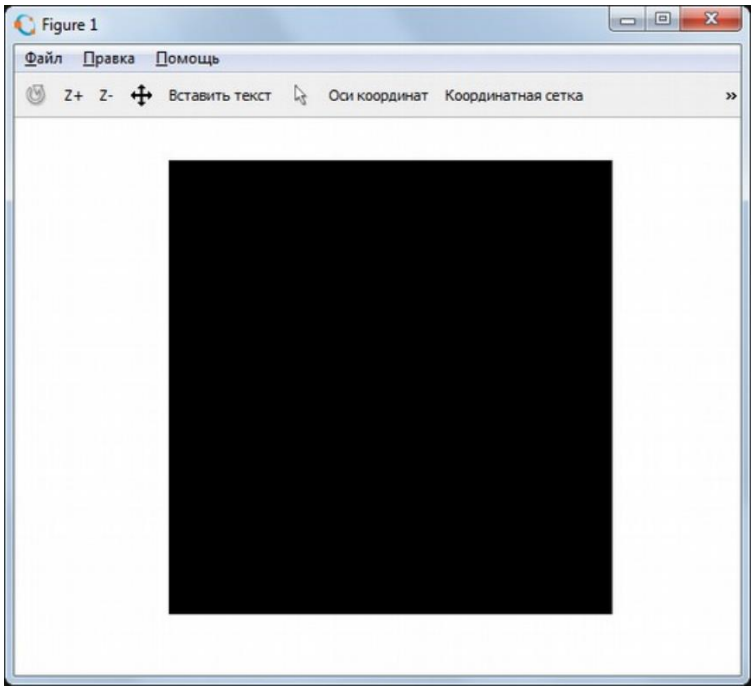


Рисунок 12-Земля черного цвета

Давайте сделаем землю более зеленой, причем с вариациями. Определим следующую функцию:

```
function variated = variate(matr, delta)
  m = size(matr,1);
  n = size(matr,2);
  variated = matr + rand(m,n)*2.0*delta-delta;
  variated = min(variated, ones(m,n));
```

Данная функция вносит небольшую вариацию в имеющуюся матрицу цветов. Применим её к земле:

```
earth(:, 2) = 1;
earth = variate(earth, 0.1);
imshow(earth);
```

Результат представлен на рисунке ниже:

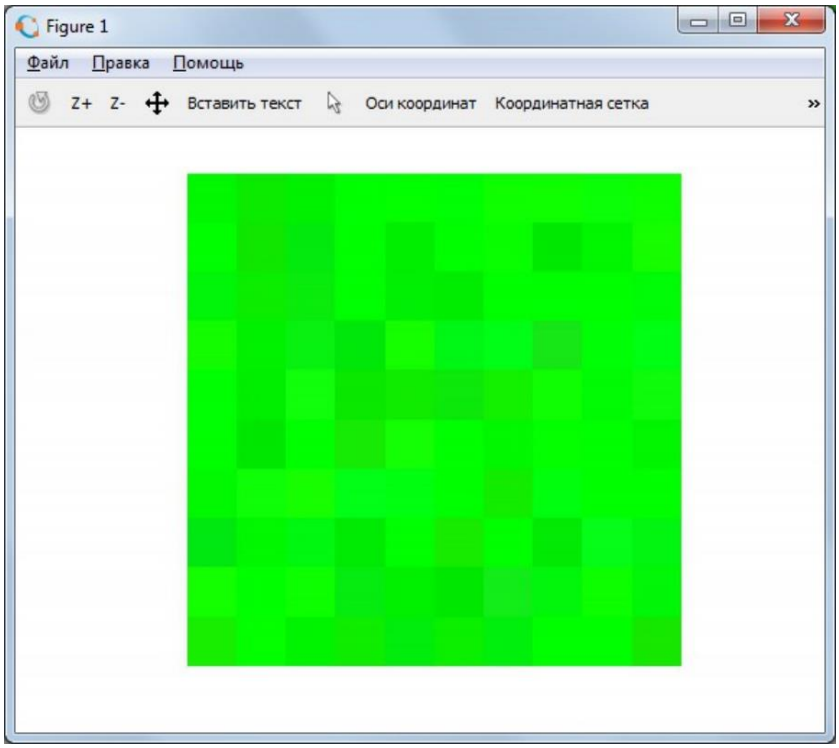


Рисунок 13-зелёная лужайка.

Для интереса прогоним цикл и посмотрим, что будет, если применять эту мутацию цвета дальше:

```
earth_random = earth;
for i = 1:10000
  earth_random = variate(earth_random, 0.01);
end
imshow(earth_random);
```

Результат представлен на рисунке ниже:

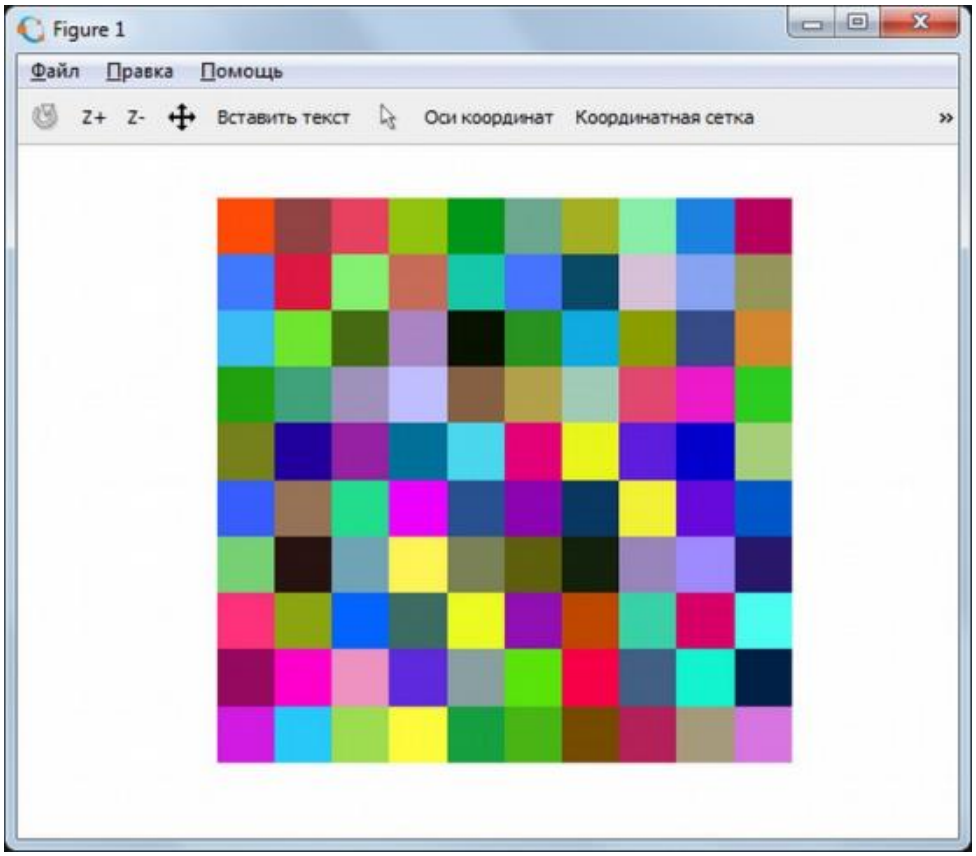


Рисунок *14* - зелёная лужайка и хаос

Несложно видеть, что сама по себе мутация приводит изначально однородную структуру к полному дисбалансу. Поэтому поступим иначе, создадим изначально разнородный набор животных и посмотрим, что с ним произойдет, если к нему применять мутацию в связке с естественным отбором:

```
animals = rand(400, 3);
imshow(animals);
```

Результат представлен на рисунке ниже:

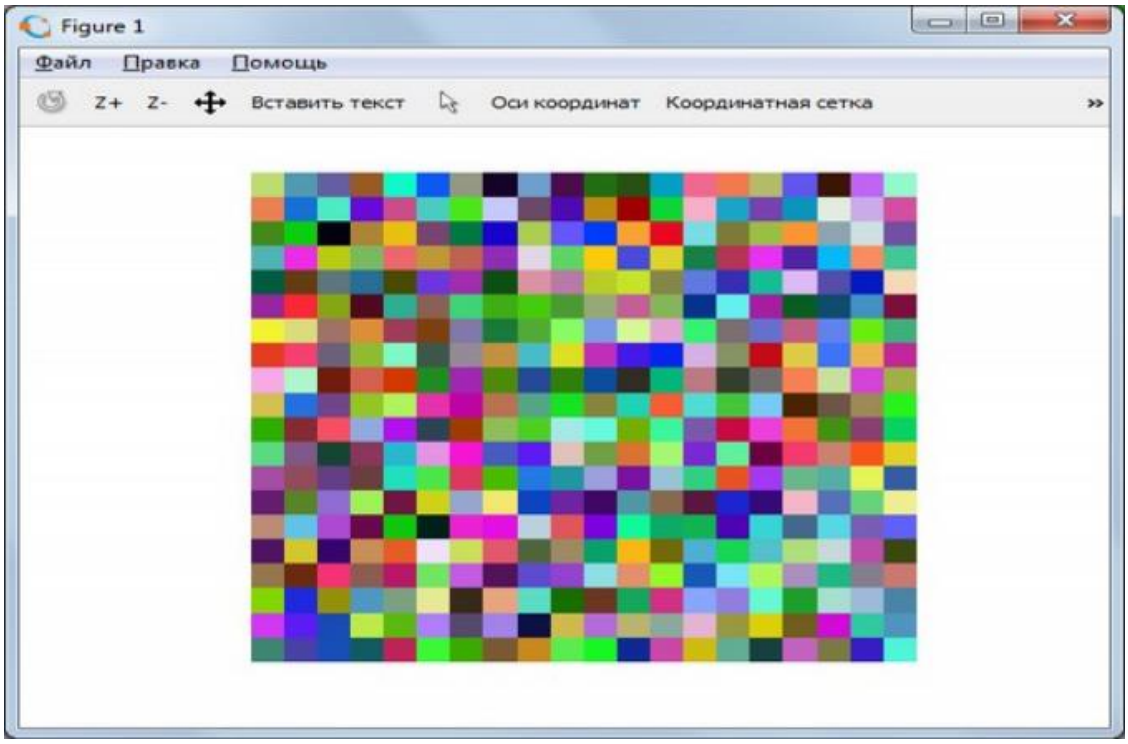


Рисунок 15-Изначальная популяция животных

Теперь, будем оценивать приспособленность каждого животного как разницу между его цветом и зеленым цветом лужайки, с помощью следующей функции:

```
function f = fitness(animals, colour)
  delta = animals - colour;
  delta = delta.^2;
  f = sum(delta')';
```

Чем меньше разница, тем более животное приспособлено. А теперь включим естественный отбор. На каждом раунде эволюции специальный отряд хищников будет убивать 10 животных ($\frac{1}{40}$от всей популяции), причем выбирая именно наиболее выделяющихся цветом:

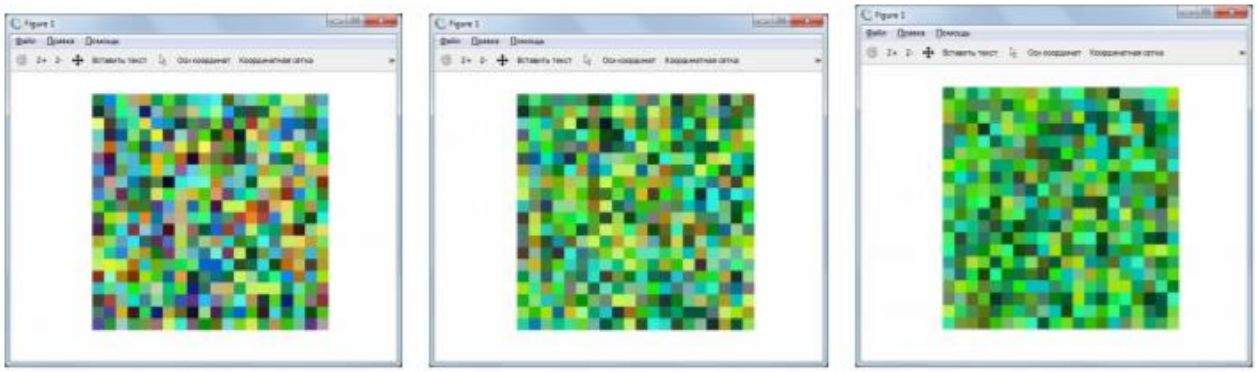
```
function evolved = evolve(animals, colour, kill)
   m = size(animals, 1);
   f = fitness(animals, colour);
   threshold = sort(f, 'descend')(kill+1);
   survivors = animals(find(f<=threshold), :);
   p = randperm(m-kill);
```

```
evolved = [survivors;survivors(p(1:kill),:)];
```

А теперь устроим несколько итераций эволюции:

```
for i = 1:20
  animals = evolve(animals, [0,1,0], 10);
  animals = variate(animals, 0.00001);
end
imshow(animals);
```

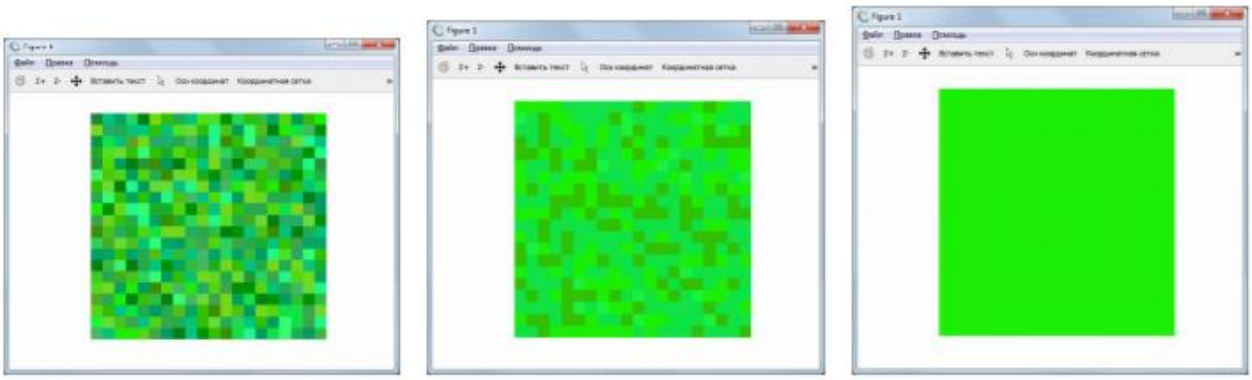
Результаты представлены ниже:



(a) 20 поколений (b) 40 поколений (c) 60 поколений

Рисунок *16*-Результат эволюции.

Как видно, цвета становятся все менее пестрыми с ходом эволюции, поскольку ярая окраска делает животных более заметными для хищников. Посмотрим, что будет по итогам нескольких сотен поколений:



(a) 100 поколений (b) 200 поколений (c) 400 поколений

Рисунок 18- Результат эволюции

Что и требовалось доказать - естественный отбор действительно работает, и его можно использовать для решения более практических задач.

3 Варианты задания

Независимо от варианта, необходимо доработать общую часть следующим образом. Для каждого поколения необходимо получить статистические показатели для фитнеса:

- Наилучший показатель
- Наихудший показатель.
- Средний показатель.
- Среднеквадратичное отклонение.

Изменения всех показателей, кроме отклонения, надо показать на графике. Помимо этого, с помощью эволюции необходимо решить задачу предсказания, т.е. животное будет тем приспособленнее, чем меньше его суммарная ошибка на обучающей выборке:

1. $z = x^2 + y^2$
2. $z = x^2 + x * y + 1$
3. $z = x^2 + y^2 + 2 * x * y$
4. $z = x^3 + y^3$

5. $z = x + \frac{y}{x}$
6. $z = x + log(y|\quad|2)$
7. $z = sin(y) + x$
8. $z = cos(x) + sin(y)$
9. $z = cos(x)^2 + sin(y)$
10. $z = x^2 + 2 * x + y$

4 Содержание работы

- Цель работы.
- Вариант задания
- Исходный текст программы.
- Результаты выполнения.
- Выводы.

Контрольные вопросы

1. Определение генетических алгоритмов.
2. Основные операции генетических алгоритмов.
3. Количественные характеристики успешности работы генетического алгоритма.
4. С помощью какой функции можно вносит вариацию в имеющуюся матрицу цветов.

12 Лабораторная работа. Нейросетевое распознавание печатных символов в среде MATLAB.

Цель работы.

1.Построение нейронных сетей в среде MATLAB.

2. Исследование возможностей распознавания печатных символов с помощью нейронных сетей.

Ход работы

Работа включает три этапа:

1. Подготовка эталонных (обучающих) образов печатных символов в виде набора графических файлов.

2. Создание и обучение нейронной сети (НС) в среде MATLAB.

3. Распознавание печатных символов с помощью обученной НС.

Искусственные нейронные сети представляют собой математическую модель функционирования биологических нейронных сетей – сетей нервных клеток живого организма. Как и в биологической нейронной сети, основным элементом искусственной нейронной сети является нейрон. Соединенные между собой нейроны, образуют слои, количество которых может варьироваться в зависимости от сложности нейронной сети и решаемых ею задач. Теоретические основы программирования таких нейронных сетей, описываются во многих работах [7, 10, 11].

Одной из актуальных задач является распознавание визуальных образов. Машины, способные распознавать росписи на бумагах, символы на банковских карточках или чеках, существенно облегчают человеческий труд и ускоряют рабочий процесс, при этом снижают риск ошибки за счет отсутствия человеческого фактора [6].

Цель разрабатываемого шаблона, создать нейронную сеть, которая сможет распознавать визуальные образы букв русского алфавита. Программный код, который решает подобную задачу, присутствует в системе Matlab, как демонстрационная программа с названием appcr1. Подробно этот код разобран и пояснен в книге «Нейронные сети» В. С. Медведева, В. Г. Потемкина [6], а так же описан в работе И. С. Миронова, С. В. Скурлаева [7]

В системе Matlab также присутствует инструмент NNtool, имеющий графический интерфейс пользователя, который существенно облегчает задачу и может быть легко использован даже неопытным пользователем. Подробно этот инструмент описан в работах В. Иванников, А. Ланнэ [5] и П. А. Сахнюка [8] и др. [12]. В работе А. И. Шеремет, В. В. Перепелицы, А. М. Денисовой показан пример

разработки нейронной сети для распознавания визуальных образов символов латинского алфавита с помощью NNtool [9]. Зарубежные ученые также применяю искусственные нейронные сети в своих исследованиях [13, 14].

1. Подготовка эталонных образов.

Примером такого набора является последовательность из десяти цифр от 0 до 9. В этом примере число образов M=10. В случае, когда каждый класс образов характеризуется лишь своим эталоном, имеем число классов, также равное M. Каждый образ формируется в виде графического файла в битовом формате. Тип файла (расширение) определяется используемыми в среде MATLAB типами графических файлов. Рекомендуется использовать расширение tif.

Для создания графических файлов образов удобно использовать среду “Adobe Photoshop”. В этом случае при создании каждого файла необходимо проделать следующую последовательность операций:

1) создать новый файл, задав его параметры: - имя : XXXX; - ширина: N1 пикселей; - высота: N2 пикселей; - цветовой режим: битовый формат. Значения N1, N2=8…20 задаются преподавателем.

2) используя инструменты типа «Кисть», «Ластик» и др. создать требуемый образ символа.

3) с помощью команды «Сохранить как» сохранить созданный образ в виде файла типа tif .

На рис. 1 приведены примеры графических символов цифр при N1=10, N2=12 пикс.

Рисунок *19*-Примеры графических символов цифр.

2. Создание и обучение НС с среде MATLAB.

На данном этапе выполнение работы в среде MATLAB производится с помощью программы sr_newff , которая реализует следующие функции:

- формирование числовых массивов эталонных образов, используемых в качестве обучающих;

- подготовка данных, необходимых для создания НС;

- создание НС, задание параметров обучения НС и обучение НС.

Эталонный образ каждого символа представлен в виде вектора-столбца [N,1], число элементов N которого равно числу признаков (иначе говоря, N – размерность пространства признаков). Такой вектор-столбец формируется из двумерного массиваизображения [N1,N2], который, в свою очередь, формируется при считывании графического файла образа с помощью команд:

imread (FILENAME) - процедура чтения графического файла;

X = reshape (A,[N,1]) - процедура преобразования двумерного массива A[N1,N2] в одномерный вектор-столбец X[N,1], где N=N1*N2.

Процедура умножения массива на 1 приводит к смене типа элементов массива с logical (для элементов битового формата) на double.

Для удовлетворительной работы НС недостаточно формирования лишь одного обучающего образа для каждого класса (типа символа) образов. Это связано с тем, что распознаваемые образы (на этапе работы НС в режиме распознавания) всегда отличаются от обучающих по ряду причин:

- различие шрифтов и стилей печатных символов;

- погрешности сканирования и неточности совмещения символа и окна сканирования; - низкое качество печати, дефекты бумаги и т.д.

В силу указанных причин для надежного распознавания образов НС следует обучать на достаточно представительном множестве образов, входящих в один и тот же класс. В программе sr_newff формирование дополнительных обучающих образов производится путем незначительного искажения эталонных образов, считываемых из графических файлов. Искажение образа-эталона каждого класса реализуется путем добавления к нему равномерного (по площади изображения) шума типа «Соль и перец», представляющего собой случайное искажение отдельных пикселей изображения. Степень искажения характеризуется числом p=[0;1], определяющим долю искаженных пикселей. Такой подход при формировании образов позволяет, во-первых, быстро получать большое число обучающих образов, и, во-вторых, регулировать (путем изменения значения p) степень разброса множества образов в пределах одного класса.

Подготовка данных, необходимых для создания НС, включает в себя:

1) формирование двумерного массива обучающих образов XR[N,K], каждый столбец которого представляет собой набор N признаков одного образа, а число столбцов K равно числу обучающих образов;

2) формирование двумерного массива желаемых откликов YR[NY,K], где NY – число выходов НС (т.е., число нейронов выходного слоя); K – число обучающих образов. Отклик YR[:,k] (в общем случае – вектор-столбец) соответствует k-му обучающему образу – вектору XR[:,k];

3) формирование двумерного массива R[N,2], определяющего минимальное R(n,1) и максимальное R(n,2) значение n-го признака, n=1,…,N. Создание НС. В общем случае НС net создается с помощью команды: net = nnnnn (P1,P2,…PL), где nnnnn – тип НС; P1,…,PL – параметры НС.

Рассмотрим встроенную функцию Matlab prprob, которая представляет собой матрицу, содержащую набор признаков букв латинского языка. Каждая буква имеет размерность 7 на 5 пикселей.

Создадим подобную матрицу с буквами русского алфавита. Для этого создадим в графическом редакторе шаблон каждого символа такой же размерностью (рис. 19, 20).

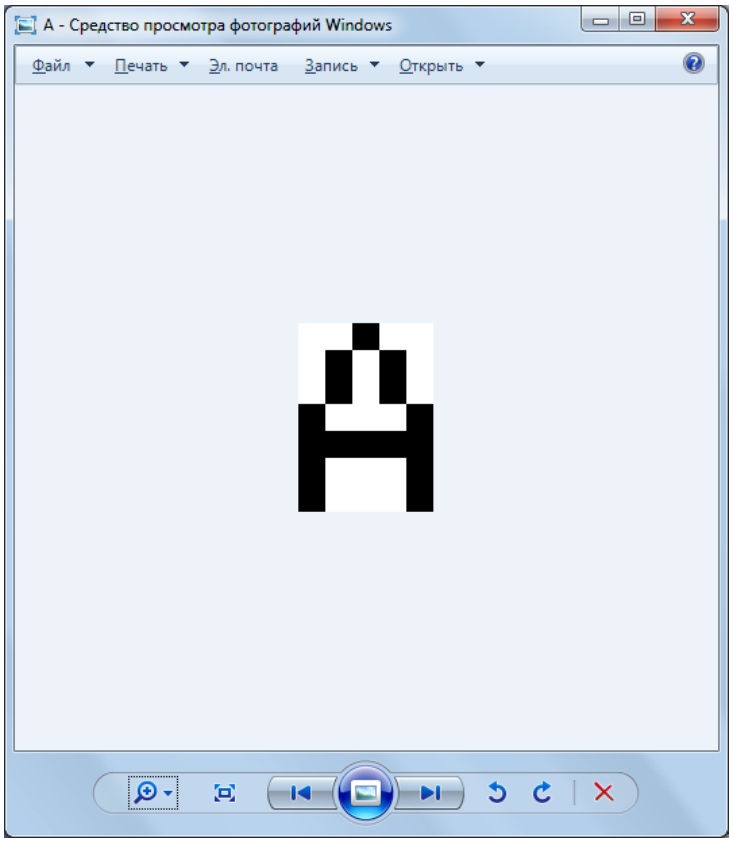


Рисунок 20- Шаблон буквы А созданный в графическом редакторе

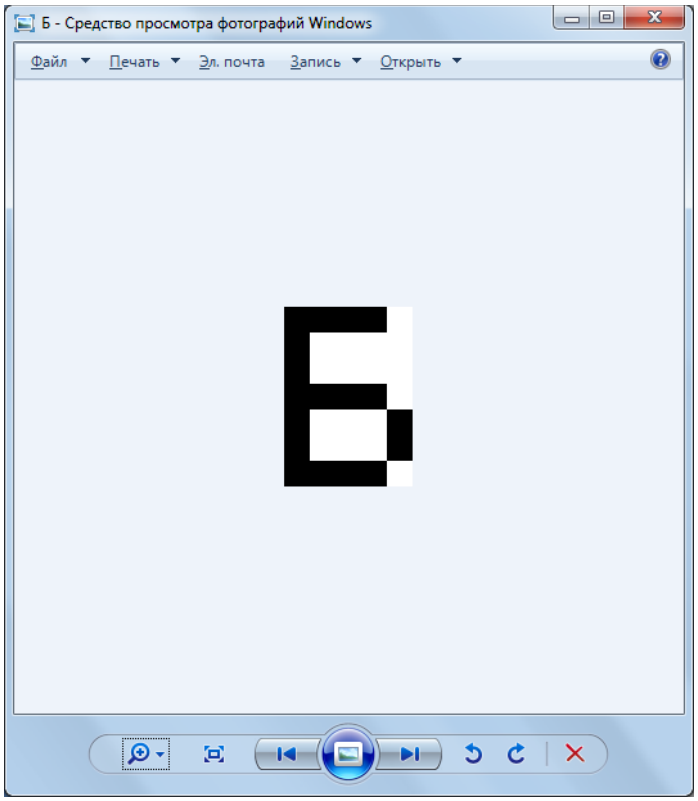


Рисунок 21-Шаблон буквы Б созданный в графическом редакторе

После того, как созданы шаблоны для каждой буквы, необходимо написать функцию, которая будет считывать необходимые признаки символов с графического файла в нужном нам формате.

Для этого выберем в командном меню: FileàNewàFunction M-file. Откроется графический редактор, в который необходимо вставить приведенный ниже код.

Код функции ImgRead:

```
function y = Imgread(x)
%UNTITLED Summary of this function goes here
% Detailed explanation goes here
img=imread(x);
img1=img(:,:,1)';
img2=reshape (img1,[35, 1]);
for i=1:35
if (img2(i,1)==0)
img2(i,1)=1;
end
end
for i=1:35
if (img2(i,1)==255)
img2(i,1)=0;
end
end
y=img2;
end
```

Использую данную функцию, создадим матрицу признаков русского алфавита, введя в окно команд следующий код:

```
images1=Imgread('C:\alphabet\A.png');
…
images33=Imgread('C:\alphabet\Я.png');
RA=[images1,images2,images3,images4,images5,images6,im-
ages7,images8,...
images9,images10,images11,images12,images13,images14,im-
ages15,images16,...
images17,images18,images19,images20,images21,images22,im-
ages23,...
```

```
images24,images25,images26,images27,images28,images29,im-
ages30,images31,...
images32,images33];
```

Заметим, что для того, чтобы код работал, шаблоны должны располагаться в каталоге C:\alphabet\, в качестве имени сам символ, с разрешением png, или внести соответствующие изменения в код.

Теперь есть матрица RA, которая содержит в себе набор признаков русского алфавита, и будет использоваться в роли входных данных при создании нейронной сети.

В качестве матрицы целей создадим единичную матрицу размерностью 33 на 33.

Для этого введем код. Так же объявим переменные, содержащие в себе количество строк и столбцов.

%Создаем переменные для создания нейронной сети в NNtool

```
P=double(RA);
T=eye(33);
[R,Q] = size(P);
[S2,Q] = size(T);
```

Для обучения сети нам понадобятся данные с шумом. Создадим эти данные, введя следующий код в окно команд:

```
%Создаем переменные для обучения на зашумленных данных нейронной сети в
%NNtool
P1=P;
T1=T;
for i=1:100
P1=[P1,P+rand(R,Q)*0.1,P+rand(R,Q)*0.2];
T1=[T1,T,T];
end
```

Для симуляции сети нам понадобится переменная, содержащая в себе набор признаков одной буквы, например, буквы И.

Введем соответствующую команду, заодно выведем на экран получившийся зашумленный образ (рис.21).

```
noisy10 = P(:,10) + randn(35,1)*0.2;
plotchar(noisy10); % Зашумленный символ И
```

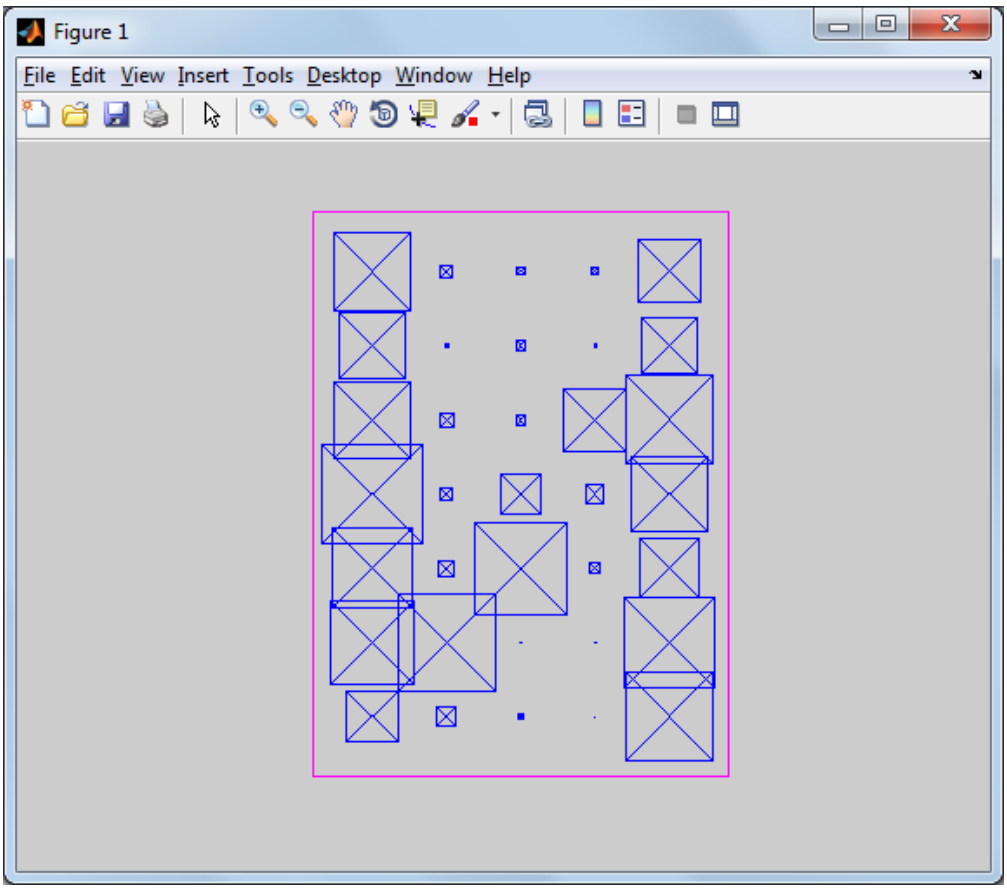


Рисунок 22-Зашумленный образ буквы И

Теперь вызовем инструмент NNtool, где с помощью графического интерфейса создадим нейронную сеть. Сделать это можно с помощью соответствующей команды nntool (рис.22).

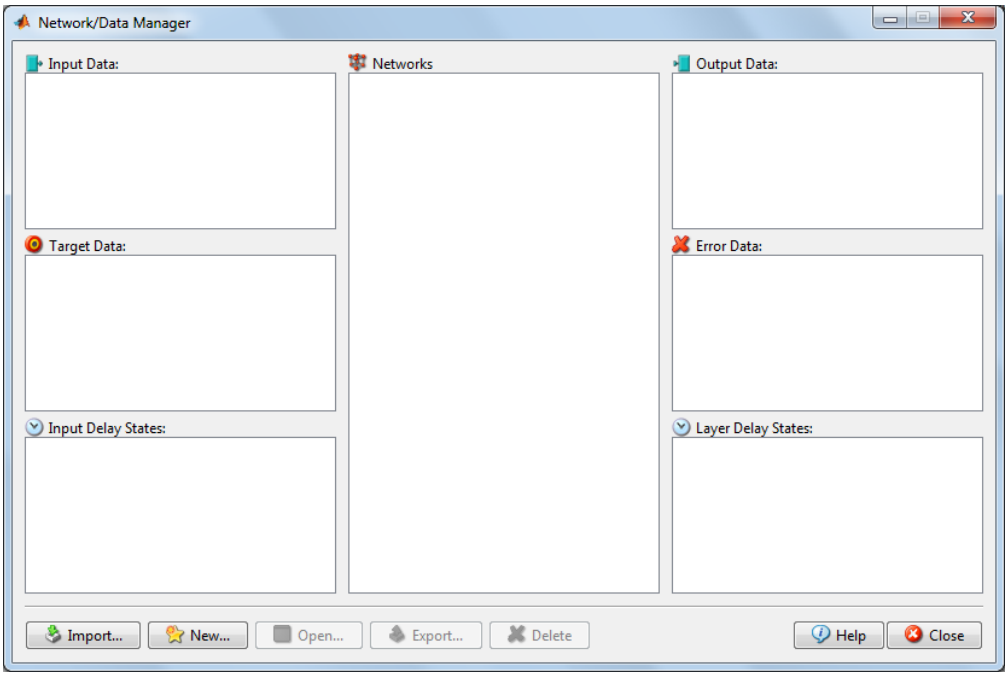


Рисунок 23-Окно менеджера данных нейронной сети

С помощью кнопки Import добавляем необходимые нам переменные (рис.23).

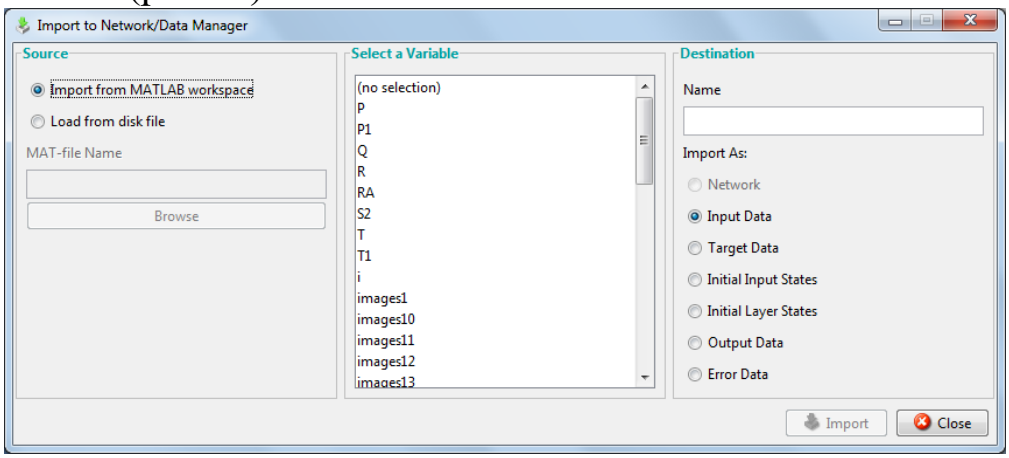


Рисунок 24-Окно импорта данных

Переменные P,P1 и noisy10 добавить как input data.

Переменные T,T1 добавить как Target data.

После импорта всех переменных окно должно выглядеть как на рис.24.

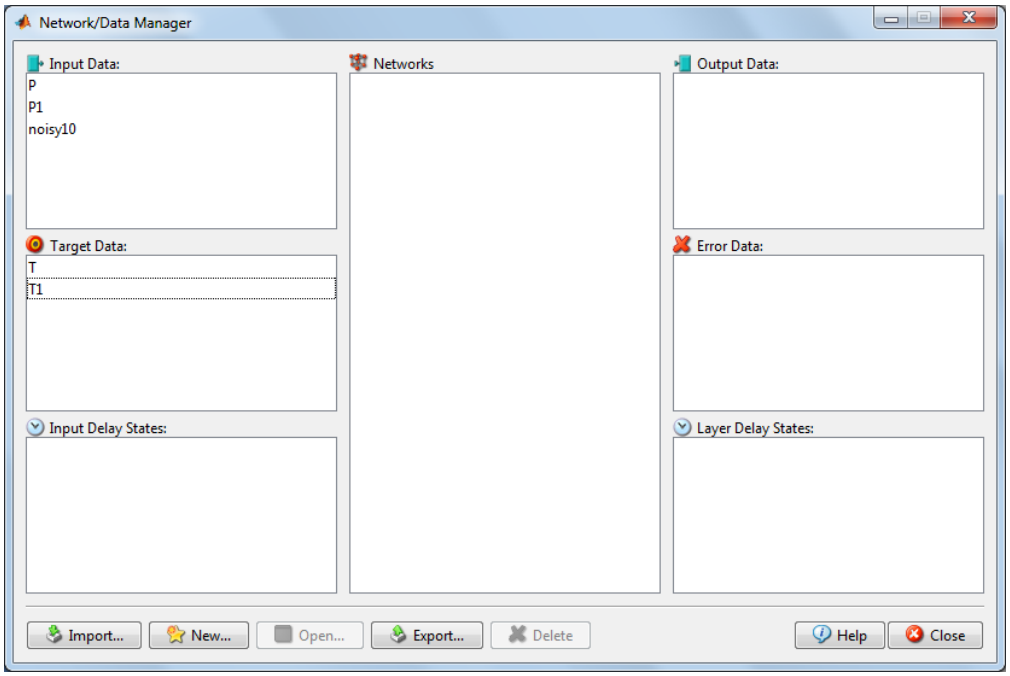


Рисунок 25-Окно менеджера данных нейронной сети после импорта переменных

Нажав кнопку New, приступим к созданию нейронной сети. В окне параметров нейронной сети введем настройки (рис.25).

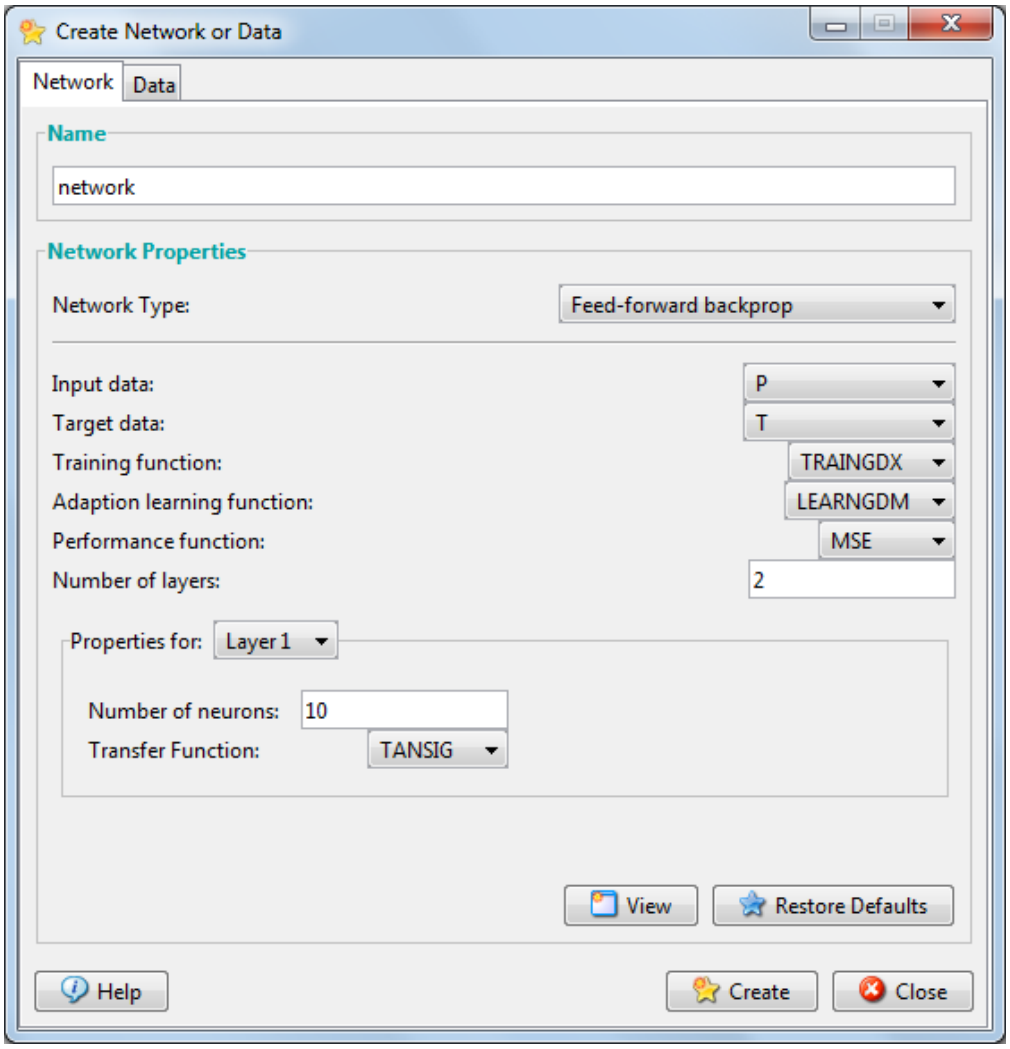


Рисунок 26-Окно создания нейронной сети

После выбранных параметров создаем сеть, нажав кнопку Create. После этого сеть должна появиться в окне менеджера данных нейронной сети в разделе networks (рис.26).

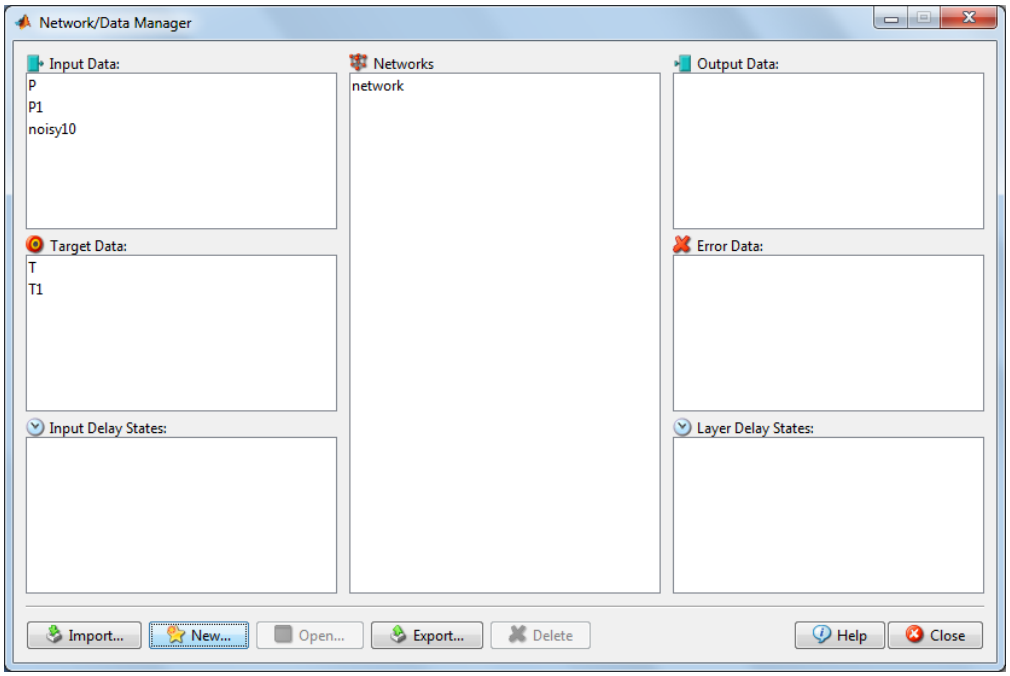


Рисунок 27-Окно менеджера данных нейронной сети после создания нейронной сети

Нажав дважды на созданную сеть, откроем нейронную сеть. Появится окно с вкладками (рис.27), в которых можно посмотреть структуру нейронной сети, обучить ее, провести симуляцию, изменять веса входных данных.

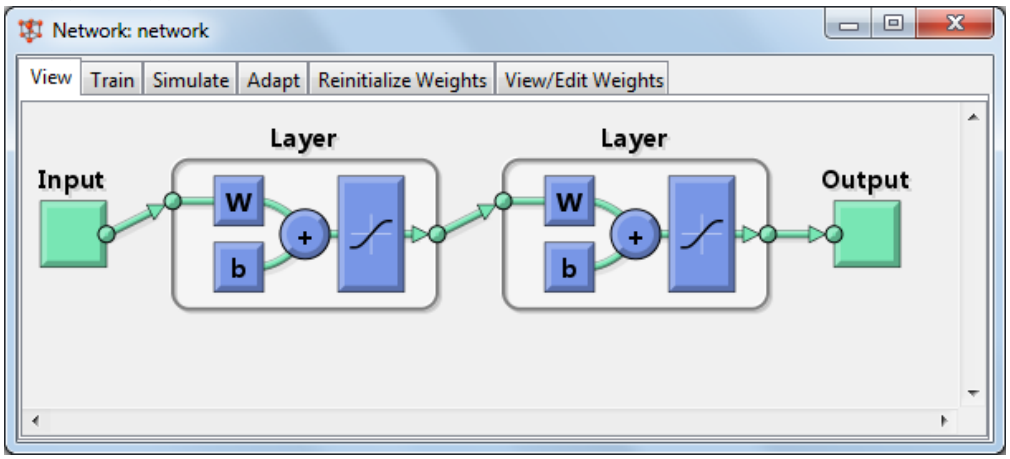


Рисунок 28-Структура нейронной сети

Рассмотрим вкладку Train, где будет проводиться обучение нейронной сети. В начале проведем обучение на идеальных данных. Введем соответствующие параметры (рис.28).

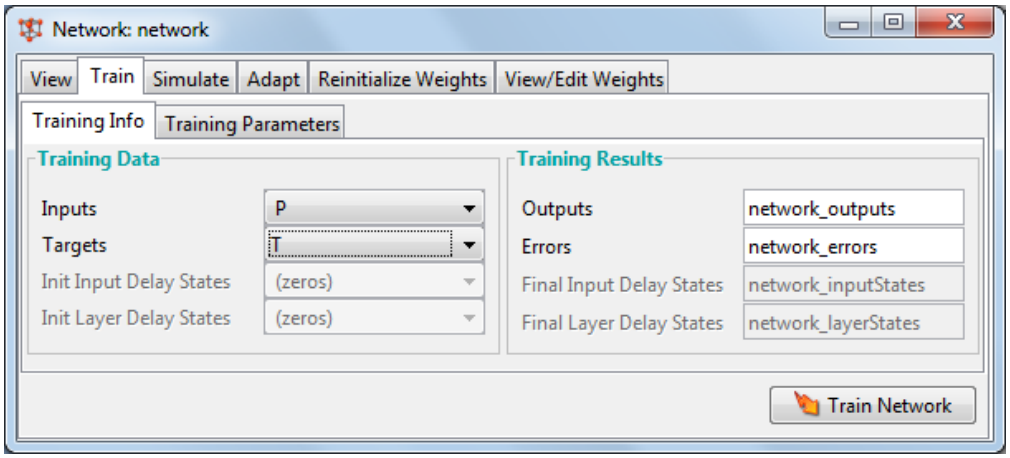


Рисунок 29-Настройка данных, с помощью которых будет происходить обучение

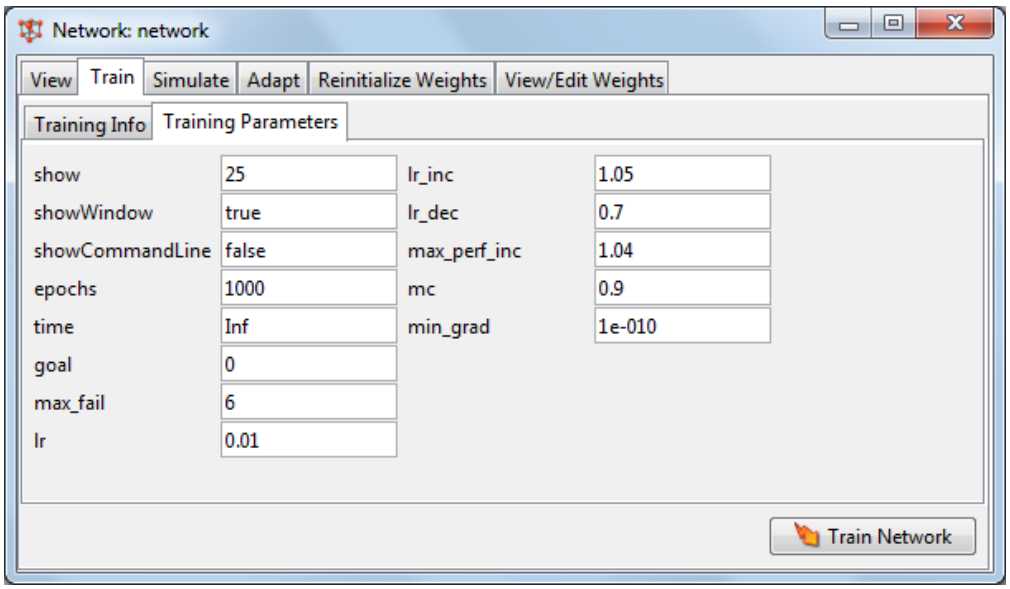


Рисунок 30- Параметры обучения нейронной сети

После установки всех параметров запускаем обучение, нажав кнопку Train Network (рис.29). В появившемся окне (рис.30) можем наблюдать процесс обучения нейронной сети.

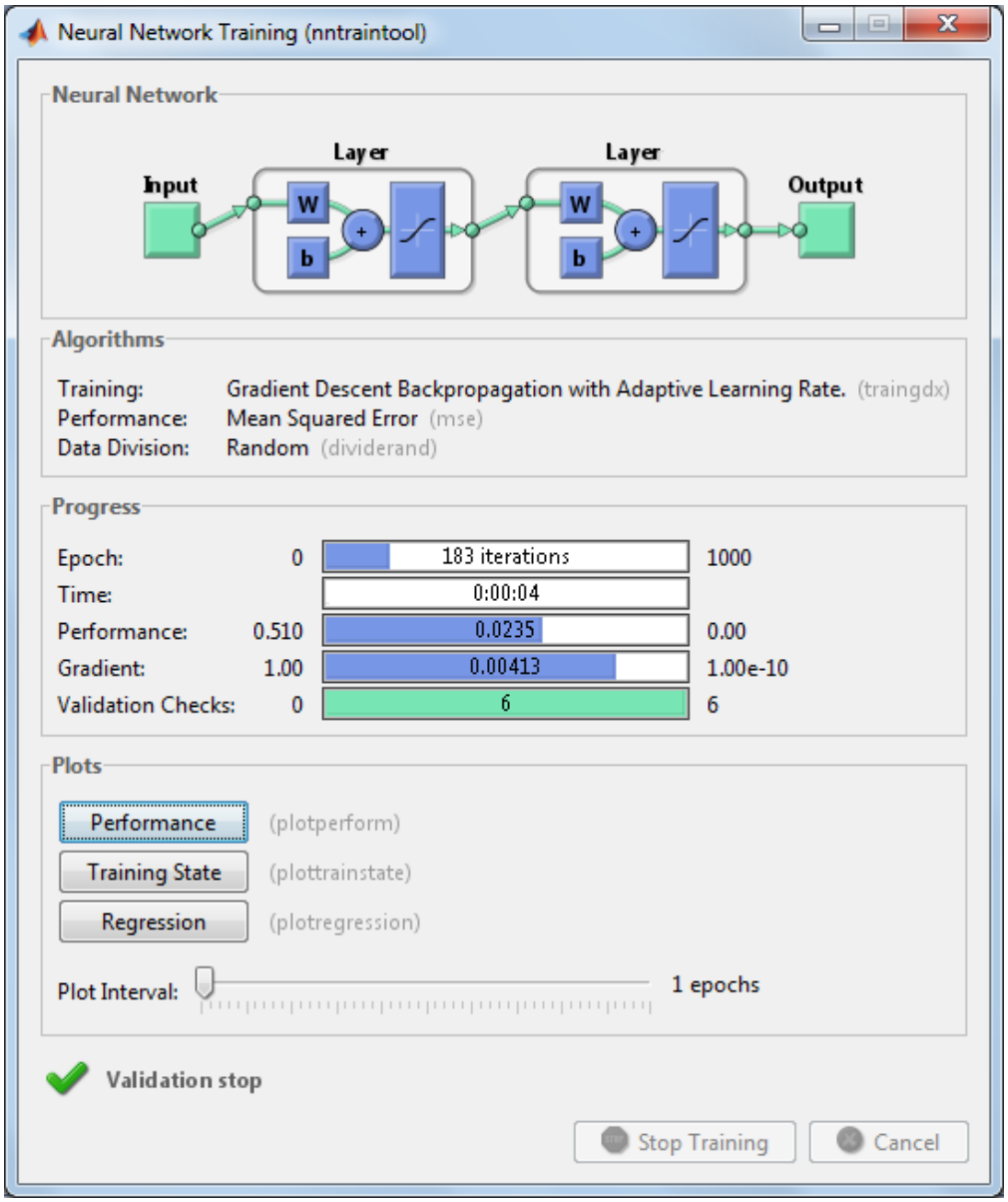


Рисунок 31-Процесс обучения

С помощью кнопки Performance можно просмотреть процесс обучения с помощью графика (рис.31).

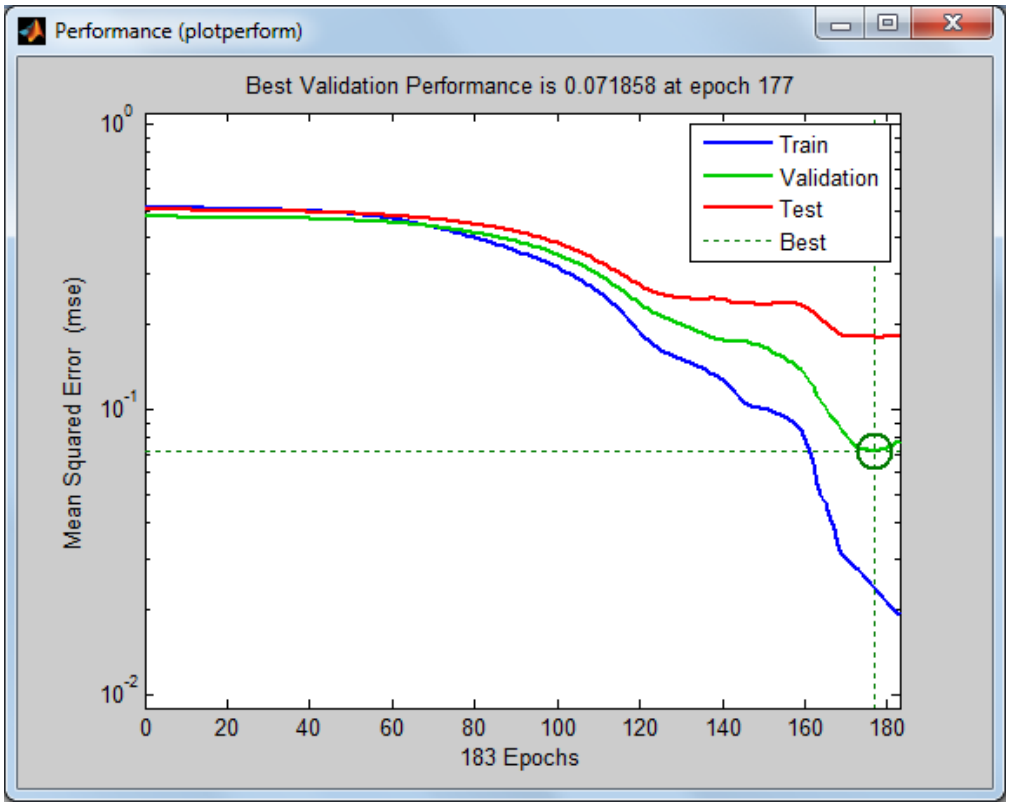


Рисунок 32-График обучения нейронной сети

Теперь необходимо провести обучение на данных с шумом. Для этого изменим параметры во вкладке train (рис.32, 35).

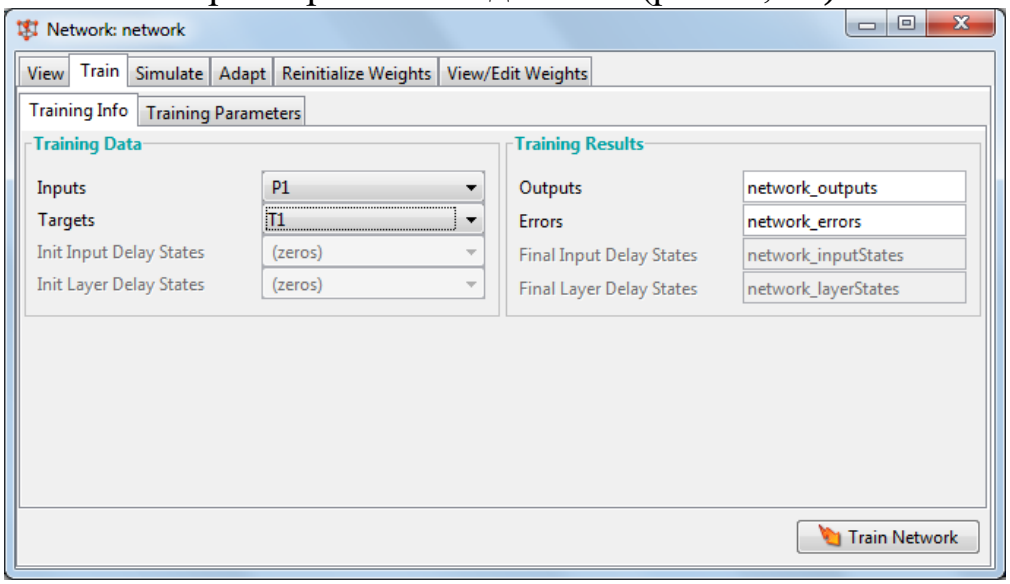


Рисунок 33-Параметры обучения на данных с шумом

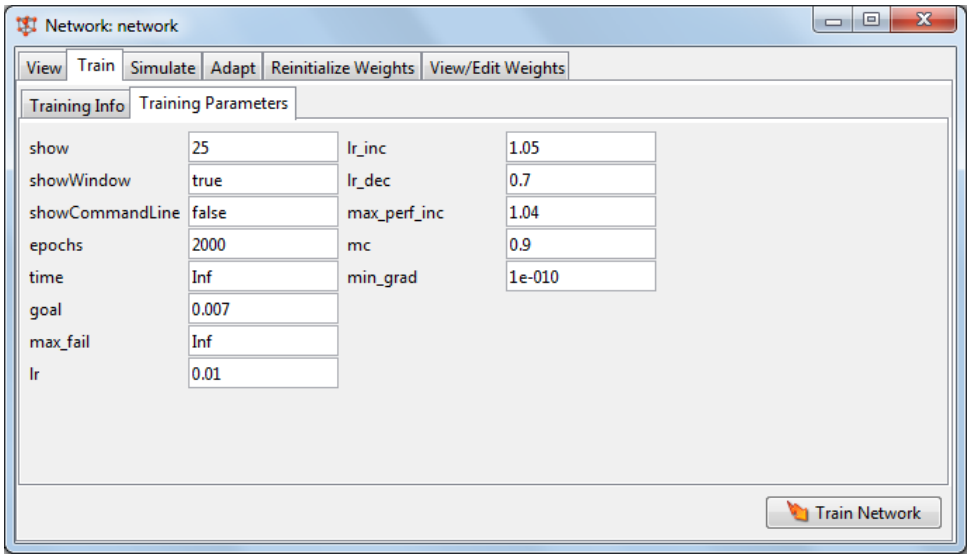


Рисунок 34-Параметра обучения на данных с шумом

Окно процесса обучения будет выглядеть как рис.34.

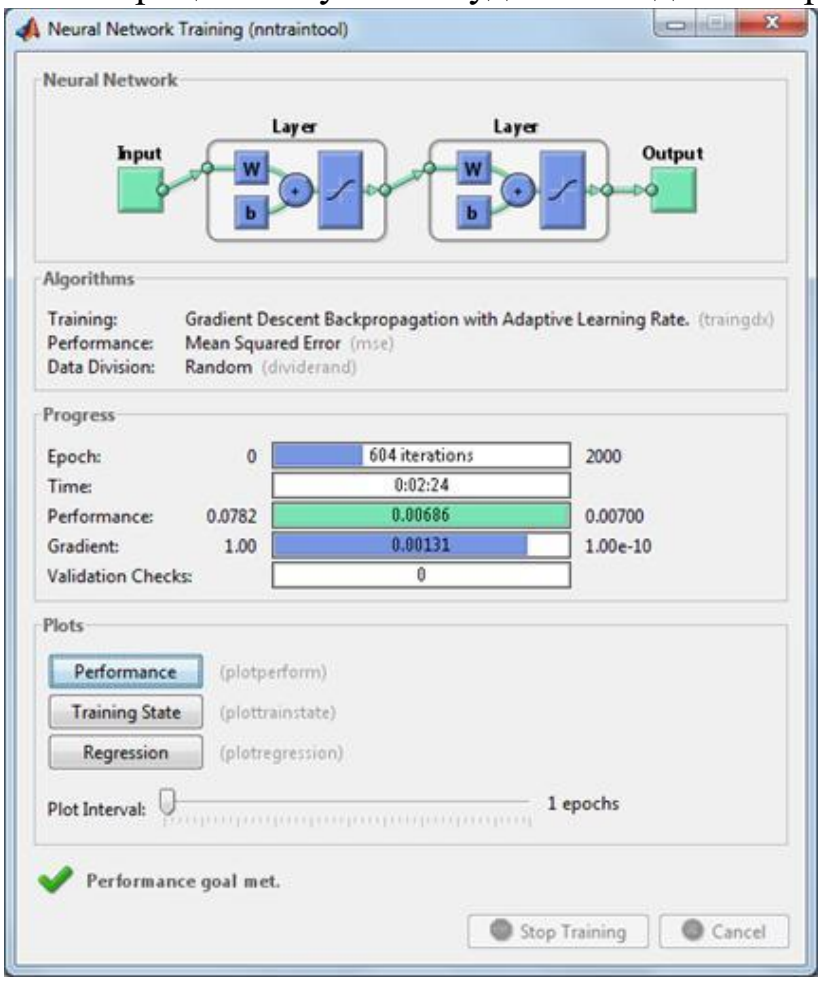


Рисунок 35- Процесс обучения сети на данных с шумом

График обучения представлен на рис. 35.

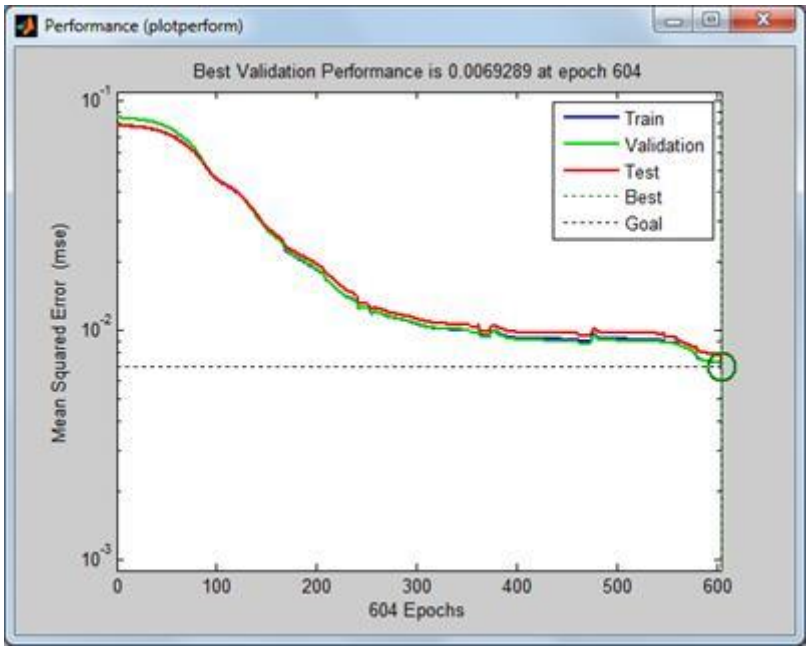


Рисунок 36- График обучения нейронной сети на данных с шумом

В результате сеть обучена. Теперь необходимо проверить нейронную сеть. Для этого создана переменная noisy10, которая содержит в себе символ «И» с шумом.

Перейдем на вкладку Simulate, в окне созданной сети (рис.36).

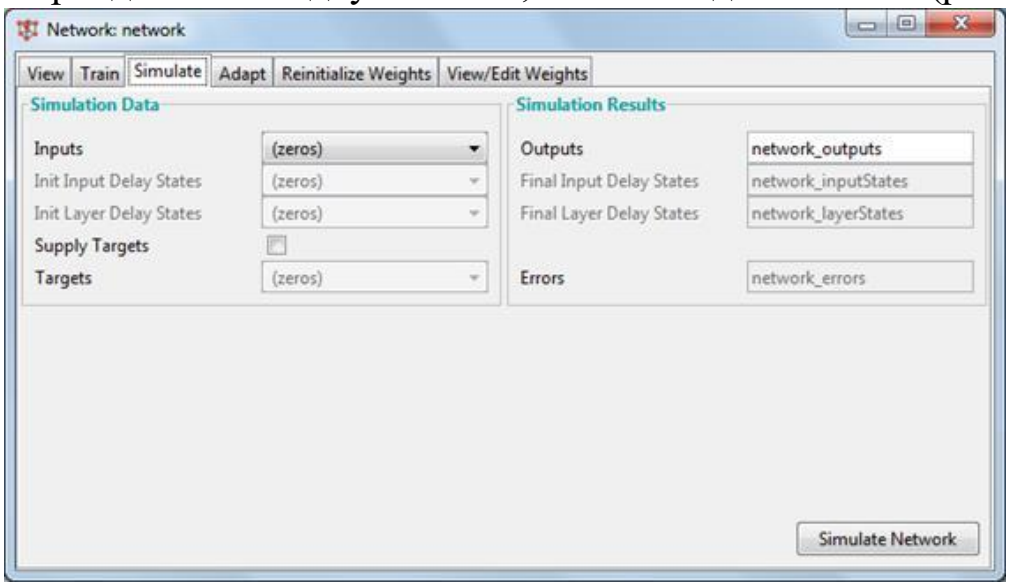


Рисунок 37 -Вкладка Simulate

Выберем в качестве входных данных переменную noisy10, а в качестве выходных данных напишем переменную Ans (рис.37).

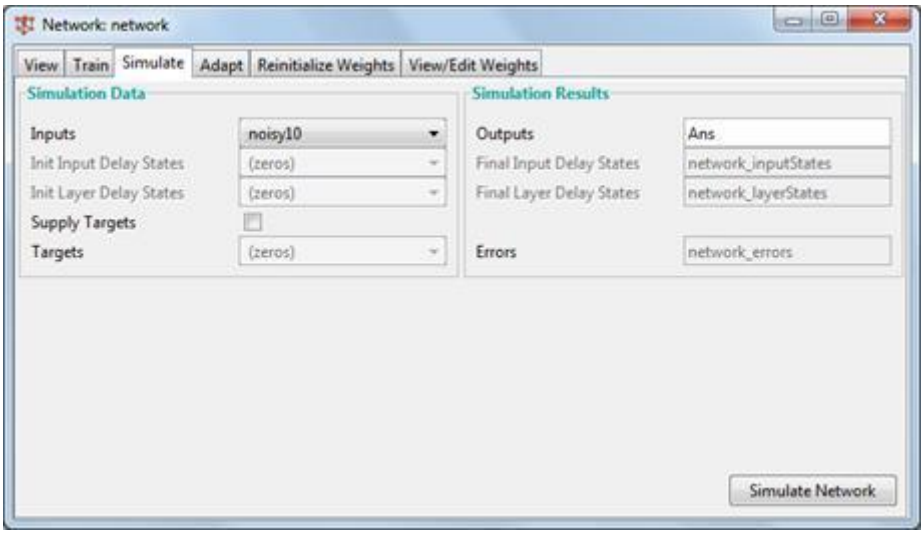


Рисунок 38-Вкладка Simulate с выбранными параметрами

Для начала процесса симуляции необходимо нажать кнопку Simulate Network.

После этого в окне менеджера данных нейронной сети появится переменная Ans (рис.38).

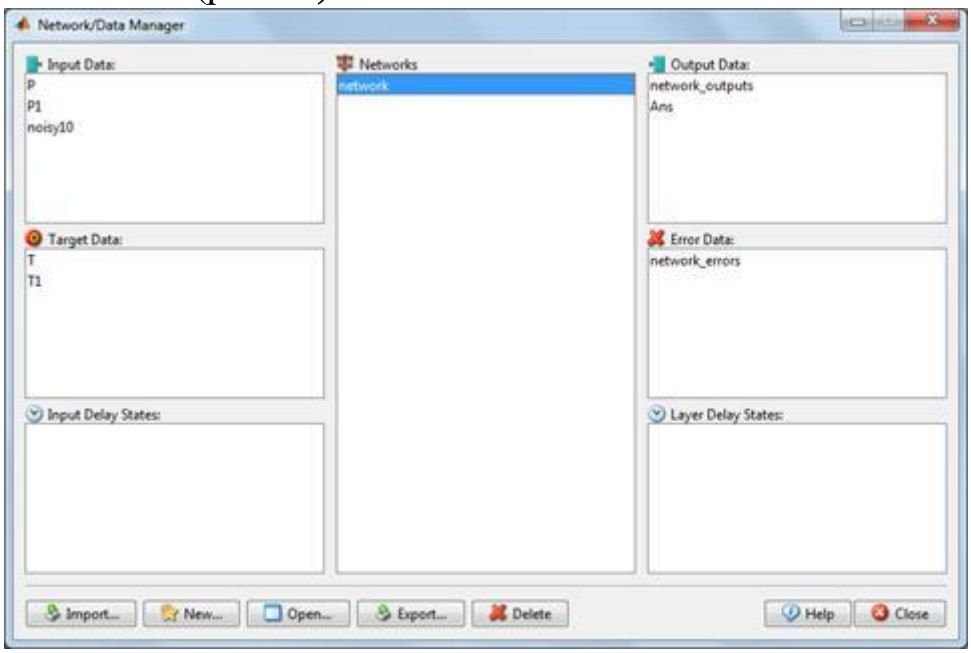


Рисунок 39-Окно менеджера данных нейронной сети после проведенной симуляции

Экспортируем нейронную сеть и переменную Ans. Для этого необходимо нажать кнопку Export и в появившемся окне выбрать необходимые нам переменные и еще раз нажать кнопку Export (рис.39).

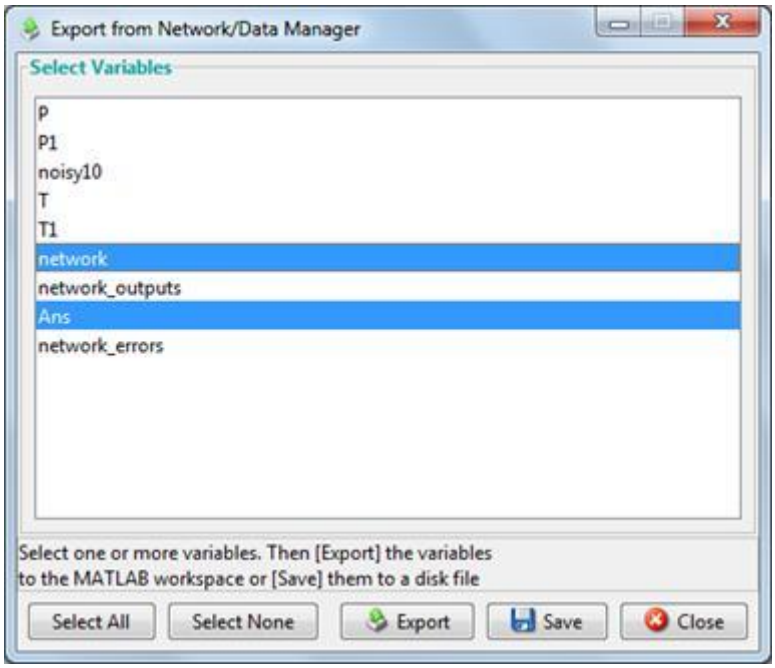


Рисунок 40- Экспорт переменных

После экспорта в окно команд Matlab вводим следующий фрагмент кода:

```
%Проверяем результат распознания
Ans = compet(Ans);
answer = find(compet(Ans) == 1)
plotchar(P(:,answer)); % Распознанный символ И
```

Перед нами появится окно (рис.40), отображающее распознанный символ.

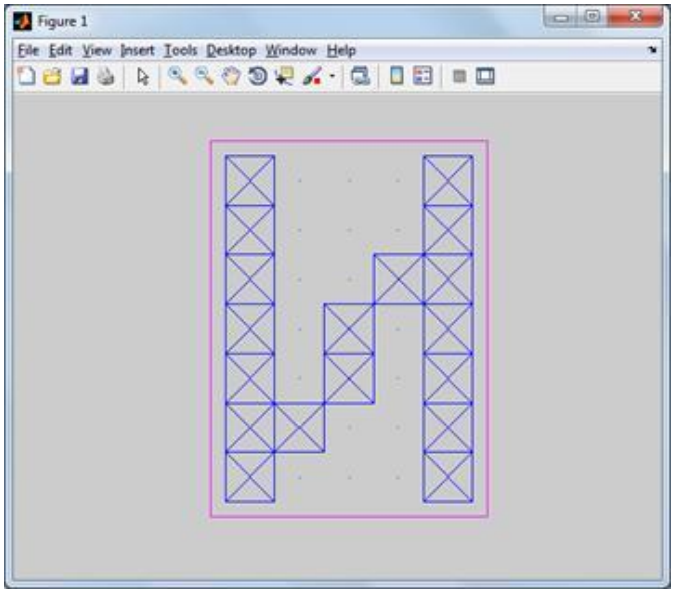


Рисунок 41-Распознанный символ И

А в окне команд появится строчка:

```
answer =
10
```

Означающая, что поступивший символ – это символ под номером 10 в нашем алфавите.

Таким образом, созданная нейронная сеть выполняет поставленную задачу.

Задание.

1. Подготовить графические файлы эталонных образов для символов, заданных преподавателем.

2. В среде MATLAB создать и обучить НС, предназначенную для распознавания печатных символов.

3. Исследовать зависимость качества работы НС от:

- степени искажения символов (параметр p);
- числа нейронов в скрытом слое.

Качество работы НС характеризуется вероятностями правильной классификации P (i) пр образа i-го класса, i=1,…,M. Оценка вероятностей P (i) пр производится по формуле:

$$\hat{P}_{np}(i) = \frac{N_{np}}{N_0},$$

где Nпр - число правильных распознаваний образа i-го класса; N0 - общее число распознаваний образов i-го класса. Число Nпр определяется экспериментально при запуске программы sr_work при значениях N0 =10…100.

4 Содержание работы

- Цель работы.
- Вариант задания
- Исходный текст программы.
- Результаты выполнения.
- Выводы.

Контрольные вопросы

1. Что включает подготовка данных, необходимых для создания НС?
2. Чем характеризуется качество работы НС?
3. За что отвечает переменная Ans?
4. С помощью какой программы происходит обучение НС?

Приложение к лабораторной работе. Нейросетевое распознавание печатных символов в среде MATLAB.

Листинг код

```
clear, [alphabet, targets] = prprob;
%Определим шаблон для символа A, который является первым элементом алфавита:
i = 6; ti = alphabet(:, i); % и — порядковый номер символа в алфавите
letter{i} = reshape(ti, 5, 7)'; letter{i} % число столбцов, число строк, не меняется
%Вызовем M-файл prprob, который формирует массив векторов входа alphabet размера %35, 26 с шаблонами символов алфавита и массив целевых векторов targets:

[alphabet,targets] = prprob;
[R,Q] = size(alphabet); [S2,Q] = size(targets);
%Двухслойная нейронная сеть создается с помощью команды newff:
%S1 -число нейронов  в скрытом слое

S1 = 10;
net=newff(minmax(alphabet),[S1 S2],...
{'logsig' 'logsig'}, 'traingdx');
net.LW{2,1} = net.LW{2,1}*0.01;
net.b{2} = net.b{2}*0.01;
gensim(net)
% конец создания нейросети
%Обучение
```

```
%Обучение в отсутствие шума
%Сеть первоначально обучается в отсутствие шума с максимальным числом циклов обучения 5000 либо до достижения допустимой средней квадратичной погрешности, равной 0.1:

P = alphabet; T = targets;
net.performFcn = 'sse'; net.trainParam.goal = 0.1;
net.trainParam.show = 20; net.trainParam.epochs = 500;net.train-Param.mc = 0.95;
[net,tr] = train(net,P,T);

%Обучение в присутствии шума
%При обучении с шумом максимальное число циклов обучения сократим до 300, а %допустимую погрешность увеличим до 0.6 (рис. 4.6):

netn = net; netn.trainParam.goal = 0.6;
netn.trainParam.epochs = 300;
T = [targets targets targets targets];for pass = 1:10
P = [alphabet, alphabet, ...
(alphabet + randn(R,Q)*0.1), ...
(alphabet + randn(R,Q)*0.2)];
[netn,tr] = train(netn,P,T);
end
% конец обучения с шумом
%Повторное обучение в отсутствие шума

netn.trainParam.goal = 0.1;% Предельная среднеквадратичная погрешность
netn.trainParam.epochs = 500;% Максимальное количество циклов обучения
net.trainParam.show = 5;% Частота вывода результатов на экран
[netn, tr] = train(netn,P,T);
% конец обучения без шума
%Эффективность функционирования системы

tic, noise_range = 0:.05:.5; max_test = 20;
```

```
network1 = []; network2 = []; T = targets;
% Выполнить тест

for noiselevel = noise_range
errors1=0; errors2=0;
for i=1:max_test
P = alphabet + randn(35,26)*noiselevel;
A = sim(net,P);   AA = compet(A);
errors1 = errors1 + sum(sum(abs(AA - T)))/2;
An = sim(netn,P);  AAn = compet(An);
errors2 = errors2 + sum(sum(abs(AAn - T)))/2; echo off
end
% Средние значения ошибок (100 последовательностей из 26 векторов)

network1 = [network1 errors1/26/100];
network2 = [network2 errors2/26/100];
end
toc
figure(1),clf
%Отображаем файл
%t = Tiff('lab1.tif', 'r');
imageData = imread('lab1.jpg','jpg');
% figure
image(imageData);
title('Образец')
grid on

    figure(2),clf
%Соответствующий график погрешности сети от уровня входного шума показан на
    plot(noise_range,network1*100,'--',noise_range,network2*100);
    grid on
%Проверим работу нейронной сети для распознавания символов. Сформируем зашумленный вектор входа для символа
```

```
%noisy = alphabet(:,6) + randn(35,1)*0.2;   % распознаем из массива
noisy=imageData;%распознаем из изображения
plotchar(noisy); % Зашумленный символ
A2 = sim(net,noisy); A2 = compet(A2);
answer = find(compet(A2) == 1)
plotchar(alphabet(:,answer)); % Распознанный символ
```

13 Лабораторная работа. фреймовая модель представления знаний

Начиная с 1960-х годов, использовалось понятие фрейма знаний или просто фрейма. Каждый фрейм имеет своё собственное имя и набор атрибутов, или слотов которые содержат значения; например фрейм дом мог бы содержать слоты цвет, количество этажей и так далее.

Использование фреймов в экспертных системах является примером объектно-ориентированного программирования с наследованием свойства, которое описывается связью «is-a» («является»). Однако в использовании связи «is-a» существовало немало противоречий: Рональд Брахман написал работу, озаглавленную «Чем является и не является IS-A», в которой были найдены 29 различных семантик связи «is-a» в проектах, чьи схемы представления знаний включали связь «is-a». Другие связи включают, например, «has-part» («имеет своей частью»).

Фреймовые структуры хорошо подходят для представления знаний, представленных в виде схем и стереотипных когнитивных паттернов. Элементы подобных паттернов обладают разными весами, причём большие веса назначаются тем элементам, которые соответствуют текущей когнитивной схеме (schema). Паттерн активизируется при определённых условиях: если человек видит большую птицу, при условии, что сейчас активна его «морская схема», а «земная схема» — нет, он классифицирует её скорее, как морского орлана, а не сухопутного беркута.

Фреймовые представления объектно-центрированы в том же смысле, что и семантическая сеть: все факты и свойства, связанные с одной концепцией, размещаются в одном месте, поэтому не требуется тратить ресурсы на поиск по базе данных.

Цель: изучение фреймовой модели, процесса разработки фреймов в виде взаимосвязанных таблиц и сложной иерархической структуры.

Фреймы – это структуры данных, в которой в определенном порядке представлены сведения о свойствах объекта.

Когда человек оказывается в новой ситуации, он извлекает из памяти ранее накопленные блоки знаний, имеющие отношение к текущей ситуации, и пытается применить их. Эти блоки знаний и представляют собой фреймы. Вероятно, знания человека организованы в виде сети фреймов, отражающих его прошлый опыт. Например: типовой номер в гостинице. Он имеет кровать, ванную комнату, шкаф для одежды, телефон и т.д. Детали каждого конкретного номера могут отличаться от приведенного описания. Но они легко уточняются, когда человек оказывается в конкретном номере: цвет обоев, положение выключателей.

Таким образом, любое представление о предмете, объекте, стереотипной ситуации у человека всегда обрамлено (отсюда frame – «рамка») характеристиками и свойствами объекта или ситуации.

Основной *структурной единицей* фрейма является **слот** – вложенная во фрейм структура данных, который представляется в виде:

‹имя слота›:{(A_i, v^i)},{r_i}

где A_i - имя признака, v^i- его значение, r_i- связь с другими слотами.

Слоты – это некоторые незаполненные подструктуры фрейма, после заполнения которых конкретными данными, фрейм будет представлять ту или иную ситуацию, явление или объект предметной

области. При конкретизации фрейма ему и егослотам присваиваются конкретные имена и происходит заполнение слотов.

В качестве значений слотов могут выступать имена других фреймов, что обеспечивает построение сети фреймов.

В общем виде фрейм выглядит следующим образом:

‹Имя фрейма›:
[‹роль 1›] (‹имя слота 1› : ‹значение слота 1›);
[‹роль 2›] (‹имя слота 2› : ‹значение слота 2›);
…….
[‹роль n›] (‹имя слота n› : ‹значение слота n›).

В общем случае структура данных фрейма может содержать более широкий набор информации, в который входят следующие **атрибуты**.

Имя фрейма. Оно служит для идентификации фрейма в системе и должно быть уникальным. Фрейм представляет собой совокупность слотов, число которых может быть произвольным. Число слотов в каждом фрейме устанавливается проектировщиком системы, при этом часть слотов определяется самой системой для выполнения специфических функций, примерами которых являются: слот-указатель родителя данного фрейма, слот-указатель дочерних фреймов, слот для ввода имени пользователя, слот для ввода даты определения фрейма, слот для ввода даты изме- нения фрейма и т.д.

Имя слота. Оно должно быть уникальным в пределах фрейма.

Значение слота. Оно должно соответствовать указанному типу данных и условию наследования. Значением слота могут быть числа или математические соотношения, тексты на естественном языке или программы, правила вывода или ссылки на другие слоты данного фрейма или других фреймов.

Пример.

Разработать фреймы в виде взаимосвязанных таблиц и сложной иерархической структуры.

Ход работы:

Для построения фреймовой модели представления знаний необходимо выполнить следующие шаги:

1) Определить абстрактные объекты и понятия предметной области, необходимые для решения поставленной задачи. Оформить их в виде фреймов-прототипов (фреймов-объектов, фреймов-ролей).

2) Задать конкретные объекты предметной области. Оформить их в виде фреймов-экземпляров (фреймов-объектов, фреймов-ролей).

3) Определить набор возможных ситуаций. Оформить их в виде фреймов-ситуаций (прототипы). Если существуют прецеденты по ситуациям в предметной области, добавить фреймы-экземпляры (фреймы-ситуации).

4) Описать динамику развития ситуаций (переход от одних к другим) через набор сцен. Оформить их в виде фреймов-сценариев.

5) Добавить фреймы-объекты сценариев и сцен, которые отражают данные конкретной задачи.

Решение.

1) Ключевые понятия данной предметной области – ресторан, тот, кто посещает ресторан (клиент) и те, кто его обслуживают (повара, метрдотели, официанты, для простоты ограничимся только официантами). У обслуживающего персонала и клиентов есть общие характеристики, поэтому целесообразно выделить общее абстрактное понятие – человек. Тогда фреймы «Ресторан» и «Человек» являются прототипами-образцами, а фреймы «Официант» и «Клиент» - прототипами-ролями. Также нужно определить основные слоты фреймов – характеристики, имеющие значения для решаемой задачи.

ЧЕЛОВЕК			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
пол	Мужской или	из внешних источников	
возраст	От 0 до 120 лет	из внешних источников	

РЕСТОРАН			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Название		из внешних источников	
Адрес		из внешних источников	
Часы работы		из внешних источников	
Специализация		из внешних источников	

Класс	Средний или высший	из внешних источников	

Фреймы-наследники содержат все слоты своих родителей, они явно прописываются только в случае изменения какого-либо параметра.

ОФИЦИАНТ (АКО ЧЕЛОВЕК)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
возраст	От 18 до 55 лет	из внешних источников	
стаж работы		из внешних источников	
зарплата		из внешних источников	
график работы		из внешних источников	
место работы	Фрейм-объект	из внешних источников	

КЛИЕНТ (АКО ЧЕЛОВЕК)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Вид оплаты	Наличные или карточка	По умолчанию (наличные)	
Статус	Обычный или Vip	По умолчанию (обычный)	
Форма заказа	Заказ есть или нет	По умолчанию (заказа нет)	
Чаевые		Из внешних источников	

2) Фреймы-образцы описывают конкретную ситуацию: какие рестораны имеются в городе, как именно организовывается посещение, кто является посетителем, кто работает в выбранном ресторане и т.д. Поэтому определим следующие фреймы-образцы, являющиеся наследниками фреймов-прототипов:

КАФЕ-РЕСТОРАН "ВКУСНЯТИНА" (АКО РЕСТОРАН)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Название	Вкуснятина	из внешних источников	
Адрес	г. Ульяновск, улица Минаева, 15	из внешних источников	
Часы работы	9:00-00:00	из внешних источников	
Специализация	Пиццерия	из внешних источников	
Класс	Средний или высший	из внешних источников	

КАФЕ "ВКУСНАЯ ЕДА" (АКО РЕСТОРАН)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Название	Вкусная еда	из внешних источников	
Адрес	г. Ульяновск, улица Карла Маркса, 5	из внешних источников	
Часы работы	9:00-00:00	из внешних источников	
Специализация	Паб	из внешних источников	
Класс	Средний	из внешних источников	

СЕРГЕЙ (АКО ОФИЦИАНТ)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
возраст	27	из внешних источников	
пол	мужской	из внешних источников	
стаж работы	5	из внешних источников	
зарплата	7 000	из внешних источников	
график работы	Через день с 18:00 до 00:00	из внешних источников	
место работы	КАФЕ "ВКУСНАЯ	из внешних источников	

МАРИНА (АКО ОФИЦИАНТ)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
возраст	24	из внешних источников	
Пол	женский	из внешних источников	
стаж работы	2	из внешних источников	
зарплата	8 200	из внешних источников	
график работы	Каждый день с 9:00 до 14:00	из внешних источников	
место работы	**КАФЕ-РЕСТОРАН "ВКУСНЯТИНА"**	из внешних источников	

ПЁТР (АКО КЛИЕНТ)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
пол	мужской	из внешних источников	
возраст	19	из внешних источников	
Вид оплаты	Наличные	По умолчанию (наличные)	
Статус	Обычный	По умолчанию (обычный)	
Форма заказа	Заказа нет	По умолчанию (заказа нет)	
Чаевые	7 % от суммы заказа	Из внешних источников	

3) Фреймы-ситуации описывают возможные ситуации. В ресторане клиент попадает в несколько типичные ситуаций: заказ и оплата. Возможны и другие не типичные ситуации: клиент подавился, у клиента нет наличности для оплаты счета и т.д. Рассмотрим типичные ситуации (их может быть больше):

ЗАКАЗ			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Перечень блюд		из внешних источников	IF-ADDED (изменяет слот «Перечень цен»)
Перечень цен		Присоединенная процедура	IF-ADDED (изменяет слот «Сумма заказчик»)
Сумма заказа		Присоединенная	
Принял заказ	Фрейм-образец	из внешнего источника	
Сделал заказ	Фрейм-образец	из внешнего источника	

ОПЛАТА			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Вид платежа		из внешних источников	IF-ADDED (изменяет слот «Чаевые»)
Чаевые		Присоединенная	
Оплатил	Фрейм-образец	Присоединенная процедура	
Заказ	Фрейм-образец	из внешних источников	IF-ADDED (изменяет слот «Оплатил»)

4) Ситуации возникают после наступления каких-то событий, выполнения условий и могут следовать одна за другой. Динамику предметной области можно отобразить в фреймах-сценариях. Их может быть множество, опишем наиболее общий и типичный сценарий посещения ресторана:

ПОСЕЩЕНИЕ РЕСТОРАНА			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Посетитель	Фрейм-объект	из внешних источников	
Ресторан	Фрейм-объект	из внешних источников	IF-ADDED, IF-REMOVED (изменяют слот «Официант»)
Официант	Фрейм-объект	присоединенная процедура (определяет по выбранному ресторану)	
Сцена 1	Вход, выбор	из внешних источников	
Сцена 2	Заказ	из внешних источников	
Сцена 3	Еда	из внешних источников	
Сцена 4	Оплата	из внешних источников	
Сцена 5	Выход	из внешних источников	

5) Пусть в рамках нашей задачи Пётр посетил ресторан «Вкусная еда». Тогда фреймы будут заполнены следующим образом:

ПОСЕЩЕНИЕ «Вкусной еды» (АКО ПОСЕЩЕНИЕ РЕСТОРАНА)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Посетитель	**ПЁТР**	из внешних источников	
Ресторан	**КАФЕ "ВКУСНАЯ ЕДА"**	из внешних источников	IF-ADDED, IF-REMOVED (изменяют слот «Официант»)
Официант	**СЕРГЕЙ**	присоединенная процедура (определяет по выбранному ресторану)	
Сцена 1	Вход, выбор	из внешних источников	
Сцена 2	**ЗАКАЗ ПЕТРА**	из внешних источников	
Сцена 3	Еда	из внешних источников	
Сцена 4	**ОПЛАТА ПЕТРА**	из внешних источников	
Сцена 5	Выход	из внешних источников	

ЗАКАЗ ПЕТРА (АКО ЗАКАЗ)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Перечень блюд	Отбивная, темное пиво	из внешних источников	IF-ADDED (изменяет слот «Перечень цен»)
Перечень цен	250, 75	Присоединенная процедура	IF-ADDED (изменяет слот «Сумма заказю»)
Сумма заказа	325	Присоединенная	
Принял заказ	**СЕРГЕЙ**	из внешнего источника	
Сделал заказ	**ПЕТР**	из внешнего источника	

ОПЛАТА ПЕТРА (АКО ОПЛАТА)			
Имя слота	**Значение слота**	**Способ получения значения**	**Демон**
Вид платежа	Наличные	из внешних источников	IF-ADDED (изменяет слот «Чаевые»)
Чаевые	30	Присоединенная процедура	
Оплатил	**ПЕТР**	из внешних источников	
Заказ	**ЗАКАЗ ПЕТРА**	из внешних источников	IF-ADDED (изменяет слот «Оплатил»)

Взаимосвязь различных видов фреймов отображается графически в виде графа (рис. 41).

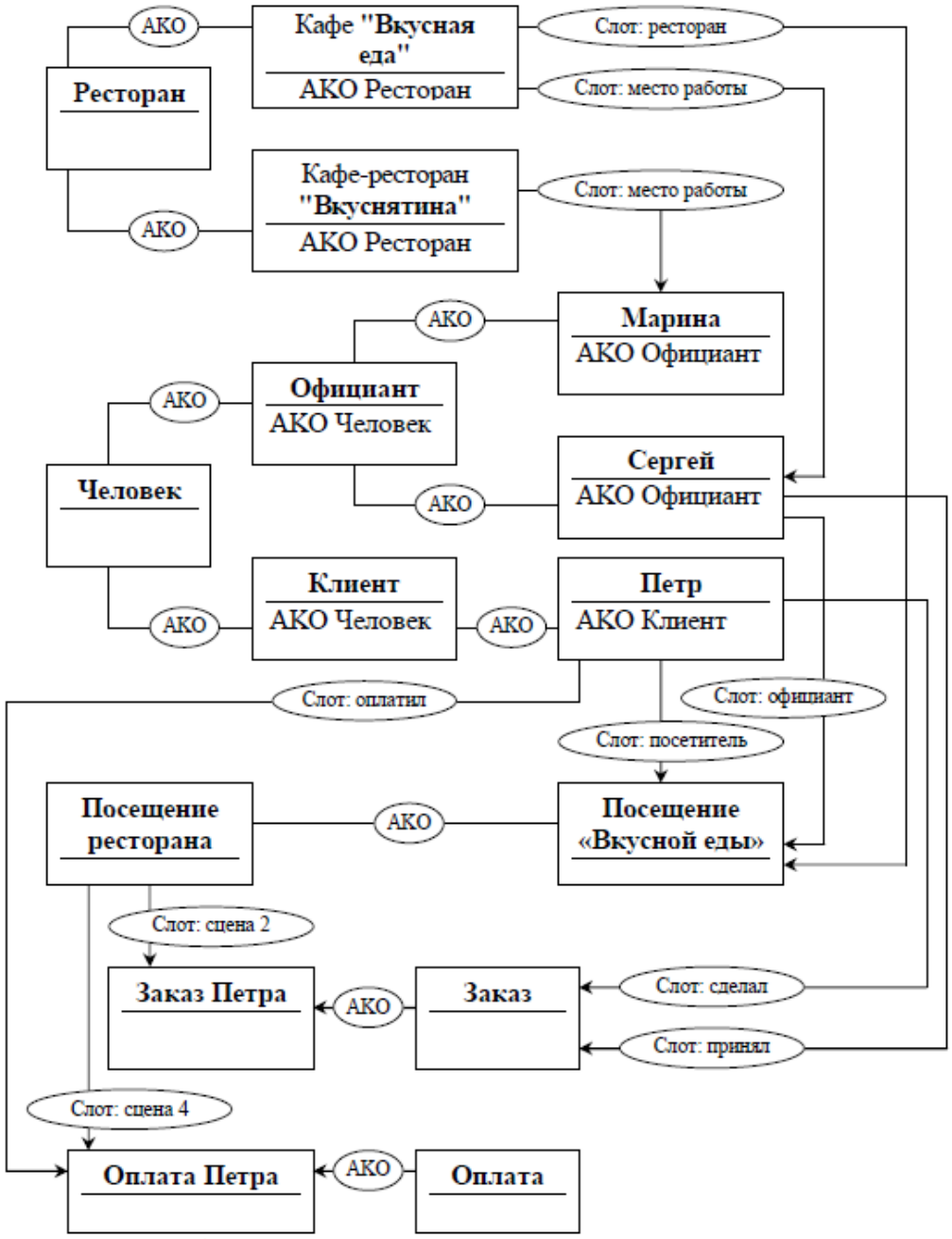


Рисунок 42- Схема фреймов для предметной области «Ресторан»

Задание.

Для выбранной самостоятельно предметной области разработать фреймы в виде взаимосвязанных таблиц и сложной иерархической структуры.

4 Содержание работы

- Цель работы.

- Вариант задания
- Исходный текст программы.
- Результаты выполнения.
- Выводы.

14 Лабораторная работа. Продукционная модель представления знаний

Системы продукций — это набор правил, используемый как база знаний, поэтому его еще называют базой правил. В Стэндфордской теории фактор уверенности CF (certainty factor) принимает значения от +1 (максимум доверия к гипотезе) до -1 (минимум доверия).

А.Ньюэлл и Г.Саймон отмечали в GPS, что продукции соответствуют навыкам решения задач человеком в долгосрочной памяти человека. Подобно навыкам в долгосрочной памяти эти продукции не изменяются при работе системы. Они вызываются по "образцу" для решения данной специфической проблемы. Рабочая память продукционной системы соответствует краткосрочной памяти, или текущей области внимания человека. Содержание рабочей области после решения задачи не сохраняется.

Работа продукционной системы инициируется начальным описанием (состоянием) задачи. Из продукционного множества правил выбираются правила, пригодные для применения на очередном шаге. Эти правила создают так называемое конфликтное множество. Для выбора правил из конфликтного множества существуют стратегии разрешения конфликтов, которые могут быть и достаточно простыми, например, выбор первого правила, а могут быть и сложными эвристическими правилами. Продукционная модель в чистом виде не имеет механизма выхода из тупиковых состояний в процессе поиска. Она продолжает работать, пока не будут исчерпаны все допустимые продукции. Практические реализации продукционных систем содержат механизмы возврата в предыдущее состояние для управления алгоритмом поиска.

Цель: приобретение студентами умений и навыков реализации пополняемой динамической базы знаний, не включаемой непосредственно в текст программы.

Необходимые для достижения поставленной цели задачи состоят в следующем:

• научиться оценивать возможности применения современных языков высокого уровня

для реализации баз знаний экспертных систем (ЭС);

• изучение технических аспектов реализации продукционной модели представления знаний.

Задание:

Реализовать продукционную систему со следующей структурой:

• база правил: область памяти, которая содержит базу знаний – совокупность знаний,

представленных в форме правил вида «ЕСЛИ-ТО»;

• глобальная база данных – область памяти, содержащая факты, которые описывают вводимые данные и состояния системы;

• интерпретатор правил (механизм логического вывода) – компонент системы, который формирует заключения, используя базу правил и базу данных.

Для синтаксического представления продукций в модели использовать язык исчисления предикатов первого порядка, то есть основными формализмами представления продукций должны быть:

а) терм, устанавливающий соответствие знаковых символов описываемому объекту;

б) предикат для описания отношения сущностей в виде реляционной формулы, содержащей в себе термы.

Термы должны быть двух видов: терм-переменная и терм-константа.

В записи условной части должна быть предусмотрена возможность наличия логических связок «И» и «ИЛИ».

Требования к системе, реализующей продукционную модель представления знаний:

а) наличие механизма заполнения базы правил и глобальной базы данных, а также

отображение результата логического вывода (интерфейс с пользователем);

б) механизм логического вывода

− нечетные номера – прямой вывод;

− четные номера – обратный вывод;

в) представление системы продукций графом «И/ИЛИ»;

г) предусмотреть механизм разрешения конфликта на этапе вывода;

д) обнаружение ошибочных правил в случае, когда либо доказательство заключения

закончилось неуспехом, либо получено неверное заключение;

е) протоколирование поиска на графе: описать процесс вывода в системе, используя

частичный граф; в случае неуспешного вывода указать невыполнимое условие.

Ход работы:

Для построения продукционной модели представления знаний необходимо выполнить следующие шаги:

1) Определить целевые действия задачи (являющиеся решениями).

2) Определить промежуточные действия или цепочку действий, между начальным состоянием и конечным (между тем, что имеется, и целевым действием).

3) Опередить условия для каждого действия, при котором его целесообразно и возможно выполнить. Определить порядок выполнения действий.

4) Добавить конкретики при необходимости, исходя из поставленной задачи.

5) Преобразовать полученный порядок действий и соответствующие им условия в продукции.

6) Для проверки правильности построения продукций записать цепочки продукций, явно проследив связи между ними.

Этот набор шагов предполагает движение при построении продукционной модели от результата к начальному состоянию, но возможно и движение от начального состояния к результату (шаги 1 и 2).

Решение.

1) Обязательное действие, выполняемое в ресторанах – поглощение пищи и ее оплата. Значит, есть уже два целевых действия «съесть пищу» и «оплатить», которые взаимосвязаны и следуют друг за другом.

2) Прежде чем что-либо съесть в ресторане, туда нужно придти, дождаться официанта и сделать заказ. Кроме того, нужно выбрать, в какой именно ресторан пойти. Значит, цепочка промежуточных действий: «выбор ресторана и путь туда», «сделать заказ официанту».

3) Прежде чем идти в ресторан, необходимо убедиться, что есть необходимая сумма денег. Выбор ресторана может обуславливаться многими причинами, выберем территориальный признак – к какому ближе в тот и идем. В разных ресторанах работают разные люди, поэтому в зависимости от выбора ресторана, официанты будут разные. Кроме того, разные рестораны специализируются на разных кухнях, поэтому заказанные блюда будут в разных ресторанах отличаться. Значит вначале идут действия, позволяющие выбрать ресторан, затем характеризующие рестораны, а уже после заказ, еда, и оплата заказа.

4) Пусть в задаче будут рассматриваться два ресторана: «Вкусная еда» и «Вкуснятина». Первый –паб и заказы приносят быстрее, чем во

втором, второй –пиццерия. В первом работает официант Сергей, а во втором официантка Марина. Петр –это клиент.

5) Выше описанное можно преобразовать в следующие предложения типа «Если, то»:

· *Если субъект хочет есть и у субъекта есть достаточная сумма денег, то субъект может пойти в ресторан.*

· *Если субъект ближе к ресторану «Вкусная еда», чем к ресторану «Вкуснятина» и субъект может пойти в ресторан, то субъект идет в ресторан «Вкусная еда».*

· *Если субъект ближе к ресторану «Вкуснятина», чем к ресторану «Вкусная еда» и субъект может пойти в ресторан, то субъект идет в ресторан «Вкуснятина».*

· *Если субъект идет в ресторан «Вкуснятина» и в ресторане «Вкуснятина» работает официант Марина, то у субъекта принимает заказ Марина.*

· *Если субъект идет в ресторан «Вкусная еда» и в ресторане «Вкусная еда» работает официант Сергей, то у субъекта принимает заказ Сергей.*

· *Если субъект выбрал блюда и у субъекта принимает заказ Марина, то заказ принесут через 20 мин.*

· *Если субъект выбрал блюда и у субъекта принимает заказ Сергей, то заказ принесут через 10 мин.*

· *Если заказ принесут через 20 мин. или заказ принесут через 10 мин., то субъект может есть.*

· *Если субъект может есть, то после еды субъект должен оплатить заказ.*

Введем обозначения для фактов (Ф), действий (Д) и продукций (П), тогда:

Субъект = *Петр*;

Ф1= *субъект хочет есть*;

Ф2= *у субъекта есть достаточная сумма денег*;

Ф3= *субъект ближе к ресторану «Вкусная еда», чем к «Вкуснятина»*;

Ф4=*в ресторане «Вкуснятина» работает официант Марина*;

Ф5=*в ресторане «Вкусная еда» работает официант Сергей*;

Ф6= *субъект выбрал блюда*;

Д1= *субъект может пойти в ресторан*;

Д2=*субъект идет в ресторан «Вкусная еда»;*

Д3=*субъект идет в ресторан «Вкуснятина»;*

Д4= *у субъекта принимает заказ Марина;*

Д5=*у субъекта принимает заказ Сергей;*

Д6=*заказ принесут через 20 мин.*

Д7=*заказ принесут через 10 мин.*

Д8=*после еды субъект должен оплатить заказ.*

Для продукций установим приоритет (в скобках перед запятой, чем выше приоритет, чем раньше проверяется правило).

П1(4 , Ф1 и Ф2)= Д1;

П2(5 , Ф3 и Д1)= Д2;

П3(4 , не Ф3 и Д1)= Д3;

П4(3 , Д3 и Ф4)= Д4;

П5(3 , Д2 и Ф5)= Д5;

П6(2 , Д4)= Д6;

П7(2 , Д5)= Д7;

П8(1 , Д6 или Д7)= Д8;

6) Для отображения взаимосвязи продукций построим граф (рис. 42).

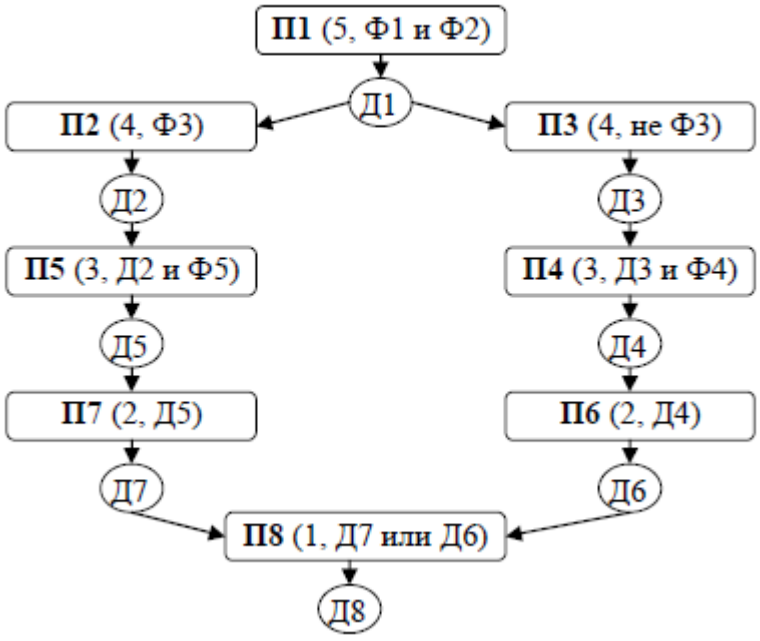


Рисунок 43- Схема продукций предметной области «Ресторан».

Задание

Реализовать продукционную систему со следующей структурой:

• база правил: область памяти, которая содержит базу знаний – совокупность знаний, представленных в форме правил вида «ЕСЛИ-ТО»;

• глобальная база данных – область памяти, содержащая факты, которые описывают вводимые данные и состояния системы;

• интерпретатор правил (механизм логического вывода) – компонент системы, который формирует заключения, используя базу правил и базу данных.

Варианты предметных областей для разработки экспертной системы

Таблица 1- Варианты предметных областей для разработки экспертной системы.

№ Вар.	Предметная область
1	Профориентация школьника
2	Выбор оптимального языка программирования в зависимости от поставленной задачи
3	Подбор учебных курсов, которые необходимо изучить для получения определенных навыков (веб-программист, windows-программист, unix-программист, тестировщик, архитектор ПО и т. д.)
4	Выбор оптимального способа вложения денег в соответствии с потребностями инвестора
5	Выбор Вуза для обучения по направлению Информационные системы и технологии
6	Ошибки системы Windows

4 Содержание работы

- Цель работы.
- Вариант задания
- Исходный текст программы.
- Результаты выполнения.
- Выводы.

Контрольные вопросы

1. Понятие фрейма, его структура, классификация фреймов.

2. Виды присоединенных процедур и принципы их функционирования.

3. Принципы организации фреймовых систем. Семантические сети, их классификация и принципы построения.

4. Понятие продукции. Структура продукции. Продукционные правила, их типы и основные структуры.

5. Продукционные системы, их структура, основные принципы организации и функционирования

Заключение

Развитие искусственного интеллекта как науки и технологии создания машин началось чуть более чем века назад. И те достижения, которых удалось достичь на текущий момент – ошеломительные. Они окружают человека практически везде. У технологий искусственного интеллекта есть особенность: человек считает их чем-то интеллектуальным только первое время, затем он привыкает и они кажутся ему естественными.

Важно помнить, что наука об искусственном интеллекте находится в тесной связи с математикой, комбинаторикой, статистикой и другими науками. Но не только они оказывают на него влияние, но и развитие искусственного интеллекта позволяет по-другому взглянуть на то, что уже было создано, как это было с программой Logic Theorist.

Важную роль в развитии технологий искусственного интеллекта играет развитие компьютеров. Едва ли можно представить серьёзную программу интеллектуального анализа данных, которой бы хватило 100 килобайт оперативной памяти. Компьютеры позволяли технологиям развиваться экстенсивно, в то время как теоретические исследования служили предпосылками для интенсивного развития. Можно сказать, что развитие науки об искусственном интеллекте являлось следствием развития компьютеров.

История развития искусственного интеллекта не закончена, она пишется прямо сейчас. Постоянно совершенствуются технологии, создаются новые алгоритмы, открываются новые области применения. Время постоянно открывает перед исследователями новые возможности и новые вопросы.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Л.Н.Чабан. Теория и алгоритмы распознавания образов. Учебное пособие. М.: МИИГАиК. 2004. – 70с.
2. ЛепскийА.Е., БроневичА.Г. Математическиеметодыраспознаванияобра-зов: Курслекций. – Таганрог: Изд-воТТИЮФУ, 2009. – 155 с.
3. Linda G. Shapiro and George C. Stockman: «Computer Vision». New Jersey, Prentice-Hall, 2001, pp 279—325.
4. Canny J.F. Finding edges and lines in images. MIT, USA, 1983, pp. 50-67
5. Детектор границ Канни.URL: https://habr.com/ru/post/114589/ (дата обращения: 01.01.2020).
6. K. Engel Real-time volume graphics, 2006, , с. 112-114
7. Р. Гонсалес, Р. Вудс. Цифровая обработка изображений. М.: Техносфера. 2012.
8. Е. Петров, Е. Медведева, Н. Харина, Е. Курбатова. Методы и алгоритмы обработки цифровых изображений // Инфокоммуникацинные технологии. 2011.
9. Анисимов Б.В. Распознавание и цифровая обработка изображений – М.: Высш. школа, 1983 – 295с
10. Марчук Ю.Н. Модели перевода. Модели перевода : учеб. пособие для студ. учреждений высш. проф. образования / Ю. Н. Марчук. — М.: Издательский центр «Академия», 2010. — 176 с.
11. Карасев И.В., Артюшина Е.А. Системы машинного перевода // Успехи современного естествознания. 2011, №7, С. 117-118
12. Морозкина Е.А., Шакирова Н.Р. Использование информационных технологий для оптимизации процесса перевода // Вестник Башкирского университета, 2012.
13. Дроздова К. А. Машинный перевод: истори, классификация, методы // Вестник омского государственного педагогического университета. Гуманитарные исследования, 2015.
14. Koehn, Philipp (2010). Statistical Machine Translation. Cambridge: Cambridge University Press. p. 15.

15. Колганов Д.С., Данилов Е.А. Обзор аналитической, статистической и нейронной технологий машинного перевода // Международный студенческий научный вестник. – 2018. – № 3-2.;

16. Lagarda, A.-L.; Alabau, V.; Casacuberta, F.; Silva, R.; Díaz-de-Liaño, E. (2009). "Statistical Post-Editing of a Rule-Based Machine Translation System" (PDF). Proceedings of NAACL HLT 2009: Short Papers, pages 217–220, Boulder, Colorado. Association for Computational Linguistics.

17. Wołk K.; Marasek K. (2013). Polish-English Speech Statistical Machine Translation Systems for the IWSLT 2013. Proceedings of the 10th International Workshop on Spoken Language Translation, Heidelberg, Germany. pp. 113–119.

18. Хорошилов А.А. Теоретические основы и методы построения систем фразеологического машинного перевода. М, 2016

19. Kyunghyun Cho; Bart van Merrienboer; Dzmitry Bahdanau & Yoshua Bengio (3 September 2014), "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches

20. Основные проблемы машинного перевода, Дубровина Е.В., Городищева А.Н. // Актуальные проблемы авиации и космонавтики, 2014.

21. Баженов Р.И. Интеллектуальные информационные технологии. Биробиджан: ПГУ им. Шолом-Алейхема, 2011. 176 с.

22. Баженов Р.И. Информационная безопасность и защита информации: практикум. Биробиджан: Изд-во ГОУВПО «ДВГСГА», 2011. 140 с.

23. Баженов Р.И. Проектирование методики обучения дисциплины «Информационные технологии в менеджменте» // Современная педагогика. 2014. № 8 (21). С. 24-31.

24. Баженов Р.И. Об организации научно-исследовательской практики магистрантов направления «Информационные системы и технологии» // Современные научные исследования и инновации. 2014. № 9-2 (41). С. 62-69.

25. Иванников В., Ланнэ А. Matlab для DSP. Нейронные сети: графический интерфейс пользователя [Электронный ресурс]. URL: http://www.chipinfo.ru/literature/chipnews/200108/1.html#lanne8

26. Медведев В. С., Потемкин В. Г. Нейронные сети. MATLAB 6. М.: ДИАЛОГ-МИФИ, 2001. 630 с.

27. Миронов И.С., Скурлаев С.В. Распознавание образов при помощи нейронной сети [Электронный ресурс]. URL: http://confonline.susu.ru/index.php?option=com_content&view=article&id=57:2011-05-06-04-36-21&catid=16:-2----&Itemid=18

28. Сахнюк П.А. Интеллектуальные информационные системы. Лабораторные работы [Электронный ресурс]. URL: http://www.stgau.ru/company/personal/user/7684/files/lib/202.pdf

29. Шеремет А.И., Перепелица В.В., Денисова А.М. Проектирование нейронной сети для распознавания символов в программной среде MATLAB [Электронный ресурс]. URL: http://nauka.zinet.info/13/sheremet.php

30. Principe J.C., Euliano N.R., Lefebvre W.C. Neural and Adaptive Systems. Fundamentals Through Simulations. New York. John Wiley & Sons Inc. 2000.

31. Luo F-L., Unbehauen R. Applied Neural Networks for Signal Processing. Cambridge University Press. 1998.

32. Demuth H., Beale M. Neural Network Toolbox. For Use with MATLAB. The MathWorks Inc. 1992-2000.

33. Awadalla M. H. A., Ismaeil I. I., Sadek M. A. Spiking neural network-based control chart pattern recognition //Journal of Engineering and Technology Research. 2011. Т. 3. №. 1. С. 5-15.

34. Dede G., Sazlı M. H. Speech recognition with artificial neural networks //Digital Signal Processing. 2010. Т. 20. №. 3. С. 763-768.

35. Айвазян С. А., Бухштабер В. М., Енюков И. С., Мешалкин Л. Д. Прикладная статистика. Классификация и снижение размерности.— М.: Финансы и статистика, 1989.— 607 с.

36. Gorban A. N., Kegl B., Wunsch D., Zinovyev A. Y. (Eds.), Principal Manifolds for Data Visualisation and Dimension Reduction, Series: Lecture Notes in Computational Science and Engineering 58, Springer, Berlin — Heidelberg — New York, 2007, XXIV, 340 p. 82 illus. ISBN 978-3-540-73749-0

37. Pearson K., On lines and planes of closest fit to systems of points in space, Philosophical Magazine, (1901) 2, 559—572; а также на сайте PCA.

38. Sylvester J.J., On the reduction of a bilinear quantic of the nth order to the form of a sum of n products by a double orthogonal substitution, Messenger of Mathematics, 19 (1889), 42—46

39. Онлайн руководство «Метод Главных Компонент (PCA)» http://www.chemometrics.ru/materials/textbooks/pca.htm