на правах рукописи

К.Н. Жаткина, Т.О. Махалкина

СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Учебное пособие

Учебное пособие предназначено для магистров, обучающихся по программам магистратуры по профилю «искусственный интеллект» по направлениям 09.04.01 «Информатика и вычислительная техника», 09.04.03 «Прикладная информатика», 09.04.02 «Информационные системы и технологии».

Учебное пособие выполнено в рамках реализации гранта на разработку программ бакалавриата и программ магистратуры по профилю «Искусственный интеллект», а также на повышение квалификации педагогических работников образовательных организаций высшего образования в сфере искусственного интеллекта (конкурс 2021-ИИ-01 от 10.06.2021).

2021

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Исследования в области искусственного интеллекта известны нашему обществу уже много лет. Активно данная область фундаментальной и прикладной науки развивается с середины XX века, однако в первые два десятилетия XXI века произошел резкий рост не просто интереса, но и практического применения алгоритмов и систем искусственного интеллекта, увеличение точности и скорости их работы.

Развитие технологий быстрого сбора и передачи информации, организация огромных хранилищ данных, технологические инновации дали мощных толчок в исследованиях и практической реализации методов машинного обучения.

Системы искусственного интеллекта внедряются во многие сферы человеческой деятельности: в науку саму по себе, в бизнес и медицину, в промышленность и в повседневную жизнь.

Пониманию принципов устройства систем искусственного интеллекта, преимуществ интеллектуальных алгоритмов, перспектив развития и способов решения задач с помощью мощного аппарата программных библиотек посвящено данное методическое пособие, ориентированное на студентов старших курсов высшей школы, получающий образование в сфере быстро развивающейся *IT*-отрасли.

ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ПО ДИСЦИПЛИНЕ

1.1. Понятие искусственного интеллекта

1.1.1. История возникновения и представления ИИ

Искусственный интеллект (*Artificial Intelligence*), или коротко ИИ (*AI*), понятие, которое вот уже более полувека будоражит умы ученых, фантастов и даже простых обывателей.

Согласно М. Тим Джонсу, искусственным интеллектом называют процесс создания машин, которые способны действовать таким образом, что будут восприниматься человеком как разумные. Это может быть повторение поведения человека или выполнение более простых задач, например, выживание в динамически меняющейся обстановке [1].

Также под искусственным интеллектом в настоящее время принято понимать способность некоторой «вычислительной машины» обучаться, принимать решения на основе предоставленных данных, даже выполнять творческие функции, традиционно приписываемые только человеку, способному мыслить и изобретать что-то новое.

Стоит отметить, что на восприятие искусственного интеллекта широкими массами повлияло представление ИИ в различных художественных произведениях, создавших некоторое клише очень умного и часто лишенного морали компьютера, чьи интеллектуальные способности в сотни раз превосходят человеческие.

Но понятие ИИ не всегда было таким.

В середине XX века британский ученый Алан Тьюринг опубликовал статью *Computing Machinery and Intelligence* [2] известную для русскоязычной аудитории под названием «Может ли машина мыслить?». Описанная в данной статье «Игра в имитацию» позднее получила название «теста Тьюринга», ставшего стандартным теоретическим тестом на определение «интеллектуальности» вычислительной машины.

Для «игры в имитацию» автором было задано условие, что к «игре» допускается только «электронная вычислительная машина» (или иначе «цифровая вычислительная машина»), а не любая техника или некоторое инженерное устройство, сделанное человеком. Таким образом под «интеллектуальной машиной» подразумевается не конкретное конструкторское решение, а устройство («машина»), способное выполнять любую вычислительную операцию, которую способен выполнить человек-исследователь [2]. При этом у потенциального исследователя имеется неограниченный запас бумаги, он может пользоваться вспомогательными устройствами для осуществления математических операций, он может использовать самые разные книги-справочники с правилами вычислений.

Алан Тьюринг утверждал, что вычислительная машина, призванная имитировать способности человека-вычислителя, должна состоять из трех частей:

1. запоминающее устройство,
2. исполняющее устройство,
3. контролирующее устройство.

При этом Тьюринг отмечал, какую конкретную деятельность человека должно имитировать каждое из трех устройств. Запоминающее устройство – берет бумагу, исполняющее устройство – выполняет одну из доступных из справочника операцию (например, сложение двух чисел), а контролирующее устройство – проверяет корректность выполнения выбранной операции. И тогда получается, что «машина» так или иначе будет ограничена определенным количеством сценариев своего поведения. Она будет запрограммирована на целый спектр действий, возможно даже превосходящий физические возможности одного человека, но все еще предсказуемый.

В своей работе Тьюринг цитировал леди Аду Лавлейс, заметившую в 1843 году, что «Аналитическая машина не может создавать что-то новое. Она может делать все, что мы и сами знает, как выполнять…» [3].

Подобный подход соответствует понятию символического искусственного интеллекта (далее — символического ИИ), являющегося доминирующей парадигмой ИИ до конца 1980-х годов XX века. Бурное развитие вычислительной техники в 50-60х годах XX века привело к росту интереса в области расширения «интеллектуальных» способностей компьютеров.

Понятие искусственного интеллекта было введено Джоном МакКарти в середине 50-х годов XX века. Профессором МакКарти также была собрана первая конференция по ИИ в 1956 г. — Дартмутская конференция. Он же создал язык *LISP*, опираясь на разработки языка *IPL*. *LISP* быстро завоевал популярность и стал основным языком разработки приложений ИИ.

После некоторого спада интереса как среди исследователей, так и среди инвесторов, произошедшего в 70-х годах XX века, в 80-е годы XX века наблюдался бум экспертных систем поддержки принятия управленческих решений, которые были построены именно как символьные ИИ: имеющие огромные базы, словари всевозможных правил, позволяющие задавать большое количество критериев. Однако такие системы обладали рядом существенных недостатков, главными из которых были:

- неспособность самостоятельно пополнять справочники, иными словами, самостоятельно находить новые решения и подходы,
- сложность в поддержке и обновлении.

Разочарование символьным ИИ привело к значительному спаду интереса к ИИ в целом, оттоку инвестиций на исследования, ведь символьный ИИ, хорошо справляющийся с решением четко определенных задач, оказался неспособен справляться с нечеткими задачами, к которым относится классификация изображений, распознавание речи, письма и переводы между естественными языками. На смену символическому ИИ пришел подход, называемый машинное обучение.

Развитие Интернета, появление потребности в построении огромных хранилищ данных, требования к улучшению алгоритмов

поиска, возрастающая скорость обмена информацией — все это привело к тому, что потребовалась разработка новых алгоритмов сбора, обработки и хранения данных, а соответственно и построения различных аналитических систем.

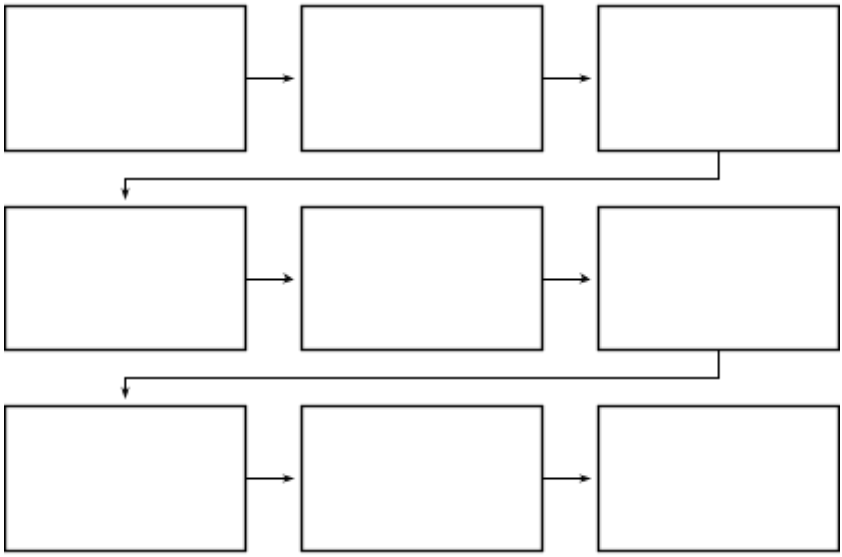

Рисунок 1. Краткая история развития ИИ

Спустя десятилетия после Тьюринга исследователи не просто пришли к умозаключению, что «вычислительная машина» все-таки способна думать, способна самостоятельно строить новые правила, т.е. способна к обучению, но и разработали алгоритмы машинного обучения.

Однако с технологиями ИИ наблюдается очень интересный эффект: как только новая революционная технология становится повсеместно используемой, она сразу же перестает восприниматься как нечто чудесное, прорывное и т.п. К примеру, сейчас никого не удивить цифровыми помощниками *Siri*, Алиса и прочими, а ведь когда-то задача распознавания голоса воспринималась чуть не как важнейший шаг к созданию ИИ.

С десятых годов XXI века наблюдается настоящий всплеск в области машинного обучения и в особенности подраздела машинного обучения — глубокого обучения.

Также стоит отметить, что различают сильный и слабый ИИ [1]:

- *Strong AI* — представляет собой программное обеспечение, благодаря которому компьютеры смогут думать так же, как люди. Помимо возможности думать, компьютер обретет и сознание разумного существа.
- *Weak AI* — представляет собой широкий диапазон технологий ИИ. Эти функции могут добавляться в существующие системы и придавать им различные «разумные» свойства.

На заре ИИ (50–60 года XX века) звучали исполненные энтузиазма лозунги о скором создании сильного ИИ, однако в XXI веке наблюдается смещение фокуса в сторону слабых ИИ. И здесь подразумевается не исчезновение потребности в создании искусственного разума или суперсложных интеллектуальных систем, а потребность в интеллектуализации как можно большего числа сфер применения: промышленности, быта, медицины и т.д.

1.1.2. Система искусственного интеллекта

Еще в 40х годах XX века, в эпоху, предшествующую компьютерной эре, у исследователей и разработчиков вычислительных систем сформировалась концепция о том, что разум – это процесс получения и обработки информации для достижения определенной цели [1].

И если искусственный интеллект — это «машина», способная к обучению, решению задач и разумной деятельности для достижения результата, то *Система искусственного интеллекта* (далее — СИИ) должна являться некоторым приложением ИИ, обладающим интерфейсом для ввода данных, внутренним интерпретатором и возможностью вывода результата в понятном для человека формате.

Основными задачами, которые должны решать системы искусственного интеллекта, можно назвать [4]:

- восприятие информации о внешней среде — сбор и предварительная обработка информации о внешней среде, выделение и распознавание объектов, а также распознавание сцен, включающих множество по-разному связанных объектов;
- формирование решений — вывод на знаниях, планирование и поиск решений, взаимодействие с целевыми объектами или управление ими, контроль и диагностика самой системы;
- общение с оператором — языковое, текстовое или графическое общение с оператором или другими системами;
- получение знаний — извлечение экспертных знаний или их автоматическое формирование путем обучения и самообучения.

С увеличением скоростей получения, обработки информации возникла потребность к пересмотру принципов хранения данных. Современные хранилища данных (*Data warehouse*) предметно-ориентированы, имеют поддержку хронологии поступления данных в хранилище, предоставляют пользователям средства поиска. Соответственно хранилища данных нуждаются в применении к ним «умных» алгоритмов представления данных, поиска, реорганизации и т.д. На основе огромных хранилищ имеются возможности строить выборки для обучения нейронных сетей и многое другое.

Развитие хранилищ данных привело к возникновению следующих понятий, относящихся к ИИ:

- *data-based knowledge* — знания, основанные на данных;
- *data mining* — извлечение данных;

- *meta data* — данные о данных, позволяющие анализировать данные по их признакам и свойствам, при этом сами данные «не затрагиваются».

Все это необходимо учитывать при проектировании каждой конкретной системы искусственного интеллекта.

1.1.3. Архитектура интеллектуальных систем

Для какой бы сферы применения не проектировалась конкретная интеллектуальная система, она в обязательном порядке должна содержать:

- интеллектуальный интерфейс — для «общения» с пользователем и «уточнения» вопросов, получаемых от пользователя путем ввода их через изображения, текст, речь;
- базу(ы) знаний — хранящую данные о предметной области и правила вывода;
- механизм вывода решений — интерпретатор найденного решения, способный представить решение пользователю в ответ на его запрос.

Не существует эталонного варианта архитектуры системы искусственного интеллекта, что значительно затрудняет быстрое распространение подобных проектов, ведь алгоритм или несколько алгоритмов, подходящих для решения задач в одной предметной области, оказываются совершенно неэффективными в другой.

Опишем в общем виде работу интеллектуальной информационной системы, которую можно было бы классифицировать как «гибридную», следующим образом:

1. Из *окружающей среды* в *сенсоры* попадают сигналы — информация о состоянии некоторого объекта, причем это, скорее всего, будет «грязная», «зашумленная» информация.
2. *Сенсоры* «очищают» информацию и передают ее далее в *нейронную сеть обработки сенсорной информации*.

3. *Нейронная сеть* после обработки полученной от *сенсоров* информации формирует символьные данные и передает их *универсальной машине вывода*. *Нейронная сеть* может передать *сенсорам* дополнительную информацию — информацию о своем состоянии.
4. *Универсальная машина вывода* формирует команды высокого уровня и передает их *моторной нейронной сети*. *Универсальная машина вывода* может передать *сенсорам* дополнительную информацию — информацию о своем состоянии.
5. *Моторная нейронная сеть* формирует команды низшего уровня и передает их на *исполнительные устройства*. *Моторная нейронная сеть* может передать *сенсорам* дополнительную информацию — информацию о своем состоянии.
6. *Исполнительные устройства* выполняют некоторые действия и таким образом оказывают влияние на *окружающую среду*. *Исполнительные устройства* могут передать *сенсорам* дополнительную информацию — информацию о своем состоянии.

Как легко заметить, в такой архитектуре большое внимание уделяется обеспечению обратной связи между элементами системы. Именно принцип обратной связи и позволяет интеллектуальным системам задавать уточняющие вопросы, корректировать свою работу в процессе, а не только после получения конечного результата.

Рисунок 2. Типовая архитектура СИИ

1.1.4 Сферы применения

Сферы применения систем искусственного интеллекта обширны в настоящее время, а успехи алгоритмов машинного обучения позволяют открывать все новые и новые способы внедрения ИИ в производство, науку, повседневную жизнь и др.

Из сфер, в которых алгоритмы ИИ уже применяются (и весьма успешно) можно выделить:

- государственное управление (за счет глобальной цифровизации);
- бизнес;
- промышленность;
- медицина;
- наука.

Рассмотрим несколько вариантов применения систем ИИ в секторе государственного управления, в частности, обеспечения

безопасности граждан страны. Снабдив места скопления большого количества людей камерами слежения и обеспечив работу алгоритма распознавания лиц или специальных предметов, можно в аэропортах, вокзалах и т.п. находить злоумышленников, опознавать подозрительные предметы. Если рассмотреть социальный сектор, то создание интеллектуального агрегатора позволило бы устанавливать льготы нуждающимся слоям населения без необходимости их обращения в многофункциональные центры.

В бизнесе возможности для внедрения «умных» решений практически безграничны. От автоматизации и интеллектуализации рутинных процессов в компаниях, набора и управления персоналом, до долгосрочных и краткосрочных точных прогнозов поведения акций на рынке и т.п.

Для производственного сектора долгие годы актуальной задачей являлась автоматизация, а потому развитие интеллектуальных систем проектирования, выполнения производственного цикла, контроля качества продукции и т.п. является естественной эволюцией. Также полезным может оказаться внедрение интеллектуальных решений в области охраны труда и жизни рабочих. Например, с помощью видеоаналитики можно анализировать наличие средств индивидуальной защиты органов дыхания на работниках химического сектора. С помощью построения тепловых карт можно контролировать опасные зоны на сталелитейном производстве, обнаруживать перегрев станков на машиностроительных комплексах и т.п. Таким образом, существенно понизится травматизм на производствах.

В сфере медицины особенно актуальны диагностические интеллектуальные системы, способные прогнозировать течение заболевания у пациентов, выявлять заболевания на ранних стадиях со слабой симптоматикой, прогнозировать течение болезни. С учетом коррективов, внесенных в жизнь мирового сообщества пандемией *Covid*-19, становится актуальной задача разработки «умных» консультантов, помогающих людям, находящимся на карантине, корректно осуществлять лечение и следовать назначениям врача.

Для науки самой по себе развитии искусственного интеллекта не менее важно, ведь облегчение жизни человека в целом позволит больше времени уделять научным и творческим изысканиям, ИИ мог бы помогать открывать и тестировать новые алгоритмы и гипотезы.

Если отталкиваться от задач, которые призваны решать интеллектуальные информационные системы, то можно отметить:

- ИС обработки и поиска информации;
- экспертные системы и ИС поддержки и принятия сложных управленческих решений;
- ИС автоматизированного проектирования;
- ИС прогнозирования и управления.

Если взглянуть на различные СИИ с точки зрения их назначения, то во всех вышеперечисленных сферах деятельности можно будет найти следующие информационные системы:

- ИС для дома и сервиса — сюда относятся всевозможные вариации «умного дома» и различных «умных помощников»;
- ИС для промышленности и бизнеса — сюда можно отнести и системы мониторинга, и системы автоматизированного управления, и системы информационной безопасности с «умным» компонентом, позволяющим не только фиксировать какие-то неполадки в системе и выдавать рекомендации по устранению, но даже прогнозировать их возникновение, предупреждать их заранее, уберегая промышленные комплексы от убытков.
- ИС для экстренных условий — например, система распознавания стихийных бедствий или иных катастроф.
- Роботехнические ИС.

Отдельного упоминания заслуживают:

- системы распознавания образов (изображений, текста, речи);

- современные интеллектуальные системы бизнес-аналитики («умная реклама» и др.).

Также стоит упомянуть, что алгоритмы ИИ активно внедряются в финансовом секторе, ведь интеллектуальное управление финансами в условиях глобализации является актуальной задачей.

Одним из самых интересных путей использования алгоритмов ИИ является область искусства. Рассмотрим, например, живопись. Обученные на огромном количестве данных нейросети исследователи «просят» нарисовать что-нибудь самостоятельно [5]. Еще в 2015 году в США компания *Google* выручила около 98 тысяч долларов на продаже картин, созданных ИИ.

Рассмотрим несколько конкретных примеров применения ИИ, уже успешно функционирующих проектов.

В 2016 году компания *Google* запустила проект *AI Experiments* [6], основной идеей которого является создание платформы для всех пользователей, желающий поработать с нейронными сетями и алгоритмами машинного обучения. Пользователи смогут изучить, что «умеет» делать каждый из представленных алгоритмов, как алгоритмы «видят» данные, которыми манипулируют. А также участники проекта имеют возможность делиться собственными разработками. Портал *AI Experiments* от *Google* функционирует и предлагает проекты по распознаванию голоса, жестов, письма и многое другое.

Компания IBM представляет *IBM Watson*, позиционируя свою разработку как ИИ [7]. *IBM Watson* имеет широкий спектр решений для бизнеса, банковского сектора, медицинского сектора, защиты данных, распознавания речи и много другого.

В среде бытовых услуг огромную популярность приобрели «умные» голосовые помощники. *Siri* от компании *Apple* распознает голосовые команды такие как: вызов абонента из адресной книги, открытые приложений, установка таймера или будильника, поиск в интернете (например, прогноза погоды). Это избавляет пользователя гаджетов (смартфонов, планшетов) от

непосредственных манипуляций с устройством в моменты, когда это может быть неудобно или даже потенциально опасно для здоровья и жизни, например, в момент приготовления пищи или вождения автомобиля. *Алиса* от *Yandex* помогает объединить в экосистему «умного дома» бытовые предметы от робота-пылесоса до электрического чайника. *Алиса* избавляет от необходимости манипуляций с электроприборами, позволяет пользователю делать несколько дел одновременно (заниматься уборкой, выбирать сериал на вечерний просмотр и т.п.). Банком *Tinkoff* разработан и запущен в эксплуатацию голосовой ассистент *Олег*, который «обитает» в мобильном приложении от этого банка и помогает управлять финансами на счетах пользователя, а при подключении сим-карты от Тинькофф Банка может еще и поработать вместо автоответчика.

Популярность и востребованность онлайн магазинов позволила накопить огромные массивы данных для анализа и начать применять алгоритмы машинного обучения для составления подборок товаров и услуг, основываясь на предыдущих покупках, а также на том, что принято называть «пользовательским опытом» (*user experience* или *UX*). Например, пользователь никогда ранее не покупал на *Ozon* подушки, но добавив их в корзину он, скорее всего, увидит рекомендацию одеяло и рубрику «с этими товарами часто покупают». В такую рекомендацию попадут товары, которые большинство пользователем покупали наряду с подушками, а также товары из смежной категории. Разумеется, данные алгоритмы не выдают абсолютно все созависимые величины, и в этом и заключается их прелесть. Подобные услуги давно стали частью таких сервисов как *Ozon*, *Amazon*, *Wildberries* и популярных социальных сетей *Instagram*, *Pinterest* и других.

Также популярностью пользуются онлайн сервисы или приложения наподобие *AI Painter* [8]. Это решения, предлагающие преобразовать фото в стилизованную картину. В основе таких сервисов лежат нейронные сети, обученные на большом количестве определенного вида примерах.

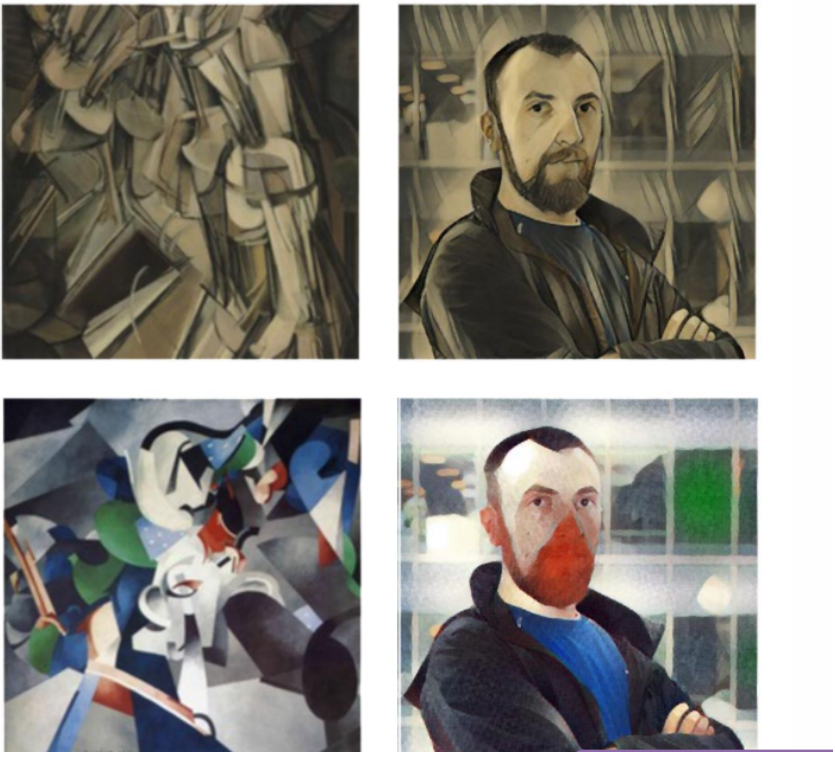

Рисунок 3. Пример работы алгоритма преобразования фото в картину

На выставке *CES* 2022 компания *DeepBrain* представила фотореалистичных аватаров, которые предлагается использовать компаниям для коммуникации с клиентами. Аватары сгенерированы на основе нескольких сотен часов видео реальных людей [9]. Аватары *DeepBrain* способны информировать и направлять пользователей в тысячах возможных сценариев и взаимодействий в реальном времени. Модель *DeepBrain* обучена воспроизводить индивидуальные особенности речи, акцента и жестов на нескольких сотнях часов записанного видео реальных людей — прообразов каждого аватара. Аватар произносит скрипт,

имитируя эти особенности и копируя внешность людей. Модель включает синтез речи, синтез видео и обработку естественного языка в реальном времени. Аватары *DeepBrain* уже используются такими компаниями, как 7-*Eleven*, *KB*, *LG HV* и *Roche*.

В *FAIR* представили *TextStyleBrush* — первую самообучаемую нейросеть, копирующую стиль текста на фотографии. *TextStyleBrush* позволяет заменить текст на изображении, используя в качестве входных данных только один пример слова.

Voilà — новое популярное приложение для *iOS* и *Android*, которое на основе селфи генерирует двух- и трехмерные рисунки с использованием искусственного интеллекта.

Примеров применения ИИ, наподобие описанных выше, с каждым годом становится все больше. «Умные» помощники, аватары и автоответчики, *AI*-дизайнеры и т.п. плотно входят как в профессиональную сферу, так и в повседневную жизнь обычных людей.

1.1.5. Вопросы к разделу 1.1

1. Что такое «Игра в имитацию»?
2. Из каких трех частей по задумке А.Тьюринга должна состоять вычислительная машина, призванная имитировать способности человека-вычислителя?
3. В какой период времени произошел всплеск интереса к экспертным системам?
4. В чем разница между «сильным» и «слабым» ИИ?
5. Каковы основные задачи, которые должны решать системы искусственного интеллекта?
6. Какие компоненты обязательно должна содержать интеллектуальная система?
7. Перечислите не менее пяти сфер применения ИИ.
8. Перечислите не менее трех успешно развивающихся и представленных на рынке приложений, использующих алгоритмы ИИ.

1.1.6. Задания по разделу 1.1

1. Предложите собственную классификацию систем искусственного интеллекта.
2. Приведите не менее 2-3 примеров по предложенной вами классификации (можно включать не более 1 примера из данного методического пособия).
3. Апробируйте приведенные вами примеры.

1.2. Философия и этика ИИ

С самого начала возникновения идеи об искусственном интеллекте поднимались вопросы этичности развития данной области. Причем формулировки вопросов у самих ученых и людей, далеких от этого сегмента знаний, несколько различаться. При этом как исследователи и разработчики систем искусственного интеллекта, так и обыватели волнуются не только о самой возможности создания ИИ, но и безопасности этих разработок.

1.2.1 Искусственный интеллект — мыслит или имитирует? Вредит или помогает?

Существует множество интерпретаций понятия «искусственный интеллект», но так или иначе способность «машины» к мышлению сравнивается с аналогичной способностью у человека – создателя этой самой «машины».

Может ли человек создать «разумную машину», если о собственном мозге нам известно далеко не все? Как человек сможет понять, что машина обладает «мышлением», когда однозначно проверяющего данное явление теста не существует? «Умная машина» мыслит или имитирует мыслительную деятельность, похожую на человеческую, так, чтобы человек мог ее «распознать»?

Если посмотреть на историю развития технологий в человеческом обществе, то можно без труда заметить, что после очередной технической и/или научной революции следующая революция наступает гораздо быстрее. На каждом этапе развития общества происходило открытие технологии, так или иначе

автоматизирующей какой-либо трудоемкий процесс, выполнять который ранее можно было только силами человеческого труда или не исполнять вовсе за неимением достаточных у человека ресурсов (физических или иных).

Укрощение сил природы, сначала воды и угля, а затем «приручение» электричества позволило человечеству повысить производственные мощности, улучшить качество жизни и здоровья, обеспечило ускорение глобализации и связей между народами Земли, открыло новые возможности для ведения бизнеса. Поэтому интеллектуализация всего достигнутого ранее кажется людям естественным продолжением развития нашего, теперь уже глобального, цифрового мира.

Однако, что будет если за все финансовые рынки будет отвечать ИИ? Например, «размышляя» над результатами, полученными после закрытия торгов на фондовой бирже, управляющий ИИ может решить, что тенденция роста некоторых акций слишком велика. Он может предпринять на следующий день открытия торгов такие действия, которые приведут к падению акций. Но, как мы знаем, подобные экономические коллапсы могут негативно сказаться не просто на какой-то одной фирме, а на целом народе страны. Если действия ИИ приведут к ухудшению жизни отдельно взятых людей, будет ли это «хорошо»? А если ИИ сможет вычислить организацию, ведущую бизнес для прикрытия террористов, будет ли «плохо», если ИИ начнет действовать против такой организации? И станет ли деятельность ИИ в таком случае имитацией того, что сделал бы человеческий совет по безопасности, например?

Таким образом, мы приходим к мысли, что «машину» нужно научить «морали», которая неотделима от понятия «эмоций».

1.2.2. Эмоции, мораль, религия и ИИ

Существует расхожее выражение «Я не машина, у меня есть чувства». Эмоции играют важнейшую роль в становлении личности, то есть способности проявлять, распознавать и даже контролировать эмоции развивается параллельно с улучшением

интеллектуальных возможностей. Человек не учится и не развивается по частям, все происходит комплексно.

Однако часто эмоции мешают человеку мыслить рационально, что может повлечь ошибки при принятии решений. В плюс «разумной машине» часто ставят ее безэмоциональность, умение не отвлекаться, не расслабляться, не уставать.

Перспективно с точки зрения обеспечения предоставления услуг в режиме 24/7 видится замена всех профессий по оказанию помощи на «умных роботов». В этом случае МФЦ смогли бы работать без перерывов и выходных, при приеме и обработке документов возникало бы меньше ошибок, а в больницах доктора больше бы общались с пациентами, предварительно прошедшими проверку у ИИ.

Но как же сострадание? То самое, в котором нуждаются люди. То, самое, из-за которого ищут дружбы, любви, радости совместного времяпрепровождения.

Все эти рассуждения приводят исследователей вопросов этичности создания ИИ к мысли, что искусственный интеллект необходимо учить не только «думать», но и «чувствовать». Программирование чувств на данный момент остается непостижимым.

Рядом с философскими и этическими вопросами к ИИ, несомненно, возникают вопросы религиозного характера. Если «машина» поступает разумно, значит ли это, что она обладает сознанием? Если она обладает сознанием, означает ли это, что такая «машина» живая? И если искусственный интеллект обладает сознанием, а значит, скорее всего, обладает волей, не будет ли его контроль приравниваться к рабству и эксплуатации? Если отключить ИИ, будет ли это убийством? Можно ли считать «живым» не биологическое существо? Есть ли у него «душа»?

1.2.3. ИИ в художественном представлении

Существует немало художественных произведений, в рамках которых авторы, не являющиеся учеными, рассуждают на тему создания человеком ИИ и последствий создания разума, равного

человеческому, а иной раз и превосходящего человеческий. Что будет, если «разумная машина» подумает о неэффективности существования человеческой расы? Как человеку контролировать ИИ, чтобы он «не стал злым», причем «злым» исключительно с позиции людей.

Фильмы «Терминатор», «Матрица» и им подобные рисуют весьма неутешительную картину, в которой искусственный интеллект принимает решение уничтожить людей. Во многих картинах, связанных с исследованиями или приключениями в космосе часто можно увидеть ИИ, помогающие людям в навигации, ведении боя, полетах и поддержке жизнеобеспечения. Также в художественных произведениях встречаются работы, рассказывающие о «положительном» развитии «умной машины», например фильм Двухсотлетний человек.

Роботы, андроиды плотно вошли в современную поп-культуру и, быть может, желание создать своего «Джарвиса» вдохновит кого-то заняться *IT* и разработкой искусственного интеллекта.

1.2.4. Задания к разделу 1.2

1. Приведите примеры неэтичного использования алгоритмов ИИ в отношении человека.
2. Приведите не менее трех гипотетических случаев нарушения прав человека, на которые мог бы пойти ИИ.
3. Приведите примеры не менее трех гипотетических случаев нарушения прав ИИ, на которые мог бы пойти человек.

1.3. Подходы к построению систем искусственного интеллекта

В различных учебниках существует несколько вариантов классификаций систем искусственного интеллекта. Рассмотрим распределение СИИ по четырем категориям.

Системы с интеллектуальным интерфейсом — это системы с усиленными коммуникативными способностями, способные выводить (искать) из имеющейся базы данных неявные результаты на основе запросов, составленных во время диалога с пользователем [10]. Запросы могут формулироваться на естественном языке, что создает максимально комфортную среду для пользователя.

Экспертные системы — это системы искусственного интеллекта, использующие символьный тип представления, символьный вывод и эвристический поиск решения. Экспертные системы предназначены для неформализованных задач, которые не могут быть представлены в числовой форме, не имеют однозначного алгоритмического решения, могут содержать противоречивых исходные данные.

Самообучающиеся системы — это системы, способные обучаться на примерах в режиме «с учителем» и «без учителя». Подобные системы используют алгоритмы классификации. В режиме «с учителем» признаки принадлежности объекта к классу задаются в явном виде. В режиме «без учителя» система занимается классификацией сама, основываясь на близости значений признаков объектов. После обучения формируются правила классификации, используя которые система может интерпретировать новые для нее объекты. Т.е. база знаний формируется автоматически (после обучения) и далее лишь только корректируется по мере накопления опыта интерпретации новых объектов.

Адаптивные информационные системы — это интеллектуальные информационные системы, способные изменять свою структуру в соответствии с потребностью изменений модели предметной области [11].

Рисунок 3. Классификация СИИ

1.3.1. Символьный подход к построению ИИ

В 1976 году Алленом Ньюэллом и Гербертом Саймоном была сформулирована гипотеза, согласно которой с помощью манипуляций с символами могут быть интерпретированы многие аспекты интеллекта (мыслительной деятельности).

Символьный подход объединяет методы исследования ИИ, основанные на понятных человеку, т.е. символьных, представления о логике, задаче и поиске информации. Этот подход к построению систем ИИ позволяет использовать слабо формализованные представления и их смыслы.

Экспертные системы являются одной из распространенных форм символьного подхода при построении ИИ. Использование конкретных производственных правил связывают символы между собой, задавая логические связи. В грубой абстракции это похоже на алгоритм условных конструкций *If-Then* достаточно большой разветвленности и глубины. При таком построении экспертная

система определяет, какие уточняющие вопросы, используя символы, ей следует задавать пользователю, после обработки правил и получения первичных выводов.

Данный подход к построению систем искусственного интеллекта был особенно популярен со времени зарождения ИИ и до конца 80-х годов XX века, когда произошел один из спадов интереса к СИИ.

1.3.2. Логический подход к построению ИИ

Логический подход к построению систем искусственного интеллекта опирается на моделирование рассуждений. Спроектированная таким образом система включает в себя коллекции правил логического вывода и наборы фактов. Однако именно в полном доверии правилам логического вывода кроется главная проблема таких систем, ведь после довольно-таки долгой цепочки логических выводов на выходе исследователь может получить от системы совершенно неожиданный результат [12].

Одними из простейших и самых ранних примеров логического подхода в построении СИИ являются решения для игры в шахматы или для выполнения математических вычислений. Для построения такой СИИ потребуется несколько компонентов: формальный язык для описания ситуации, легальные выражения в терминах формального языка, правила перехода между выражениями, описание начальных и конечных выражений, и сама аналитическая машина, способная построить решения от начального до конечного выражения.

У логического такого подхода два ограничения. Первое ограничение заключается в том, что сложно изложить в формальных терминах неформальное знание. Вторая сложность, заключается в практической реализации задачи, которая, казалось, может быть решена теоретически. Ресурсы компьютера не безграничны, особенно если у него нет каких-либо указаний относительно о логических изысканиях, которые обязательно нужно проводить, или с которых следует начинать.

Одна из главных проблем системы искусственного интеллекта, спроектированной согласно логическому подходу, касается представления и исследования естественных и формальных языков средствами логики. Естественные и формальные языки есть главная опора для представления знаний и для рассуждений по поводу знаний. Благодаря языкам логического программирования формальные языки логики предоставляют возможность автоматизировать дедуктивные процессы, сопутствующие формализации рассуждений [13].

Классическая логика (логика высказываний и логика предикатов) выявила свою непригодность для решения целого ряда задач, что привело к созданию других логических систем, обычно называемых неклассической логикой. К тем их них, которые используются в области проектирования системы искусственного интеллекта, относятся многозначные логики, частичная логика, интуиционистская логика и нечеткая логика, а также модальная логика (которая в свою очередь делится на временную, динамическую и другие логики).

1.3.3. Агентный подход к построению ИИ

Под агентом при построении системы ИИ подразумевается то, что программа будет делать чуть больше, чем просто исполнять заданные правила. Агент может работать автономно, он может получать сигналы, т.е. считывать информацию не только от пользователя, но и с различных датчиков и сенсоров. Агент может самостоятельно создавать цели, выполнять задачи, он адаптируется к изменениям среды, окружающей ИИ [12].

В одной СИИ может действовать одновременно несколько агентов, тогда имеет смысл говорить о мультиагентной системе.

Этот подход начал развиваться с 1990-х годов и определял, что интеллект – это вычислительная часть возможности достижения целей, поставленных перед интеллектуальной машиной. Такая машина и есть интеллектуальный агент. Агент не просто изучает и познает окружающий его мир через сенсоры и

датчики, но и сам действует на объекты в окружающей среде благодаря своим исполнительным механизмам.

1.3.4. Вопросы к разделу 1.3

1. Какой из подходов к построению систем ИИ возник раньше остальных?
2. В каких режимах могут работать самообучающиеся СИИ?
3. Какие виды логик используются при логическом подходе к построению ИИ?
4. Что делает «агент» в СИИ, построенной по агентному принципу?
5. К какому подклассу относятся системы с усиленными коммуникативными способностями?

1.4 Машинное обучение

Машинное обучение (англ. *Machine Learning*) является достаточно значимой частью понятия об ИИ в целом, ведь именно способности обучаться (с целью последующего создания чего-то нового или с целью поиска решения трудной задачи) исследователи пытаются добиться от вычислительных устройств.

В парадигме символического ИИ, люди вводят правила (программу) и данные для обработки в соответствии с этими правилами и получают ответы. В машинном обучении люди вводят данные и ответы, соответствующие этим данным, а на выходе получают правила. Эти правила затем можно применить к новым данным для получения оригинальных ответов. В машинном обучении система обучается, а не программируется явно. Ей передаются многочисленные примеры, имеющие отношение к решаемой задаче, а она находит в этих примерах статистическую структуру, которая позволяет системе выработать правила для автоматического решения задачи [3].

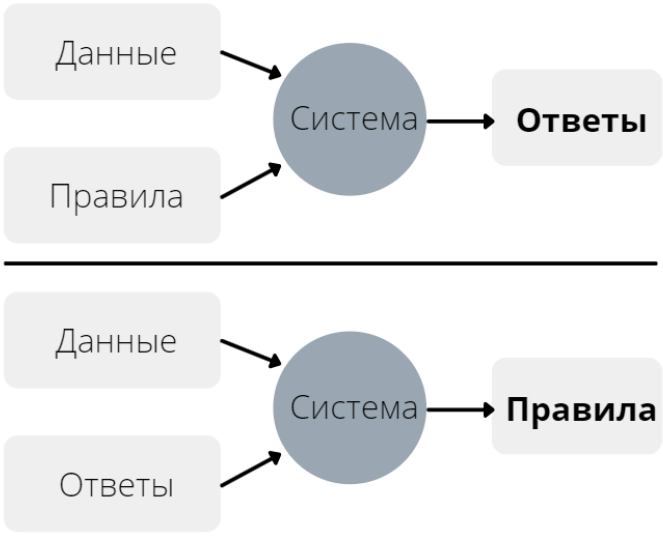

Рисунок 4. Старая и новая парадигма

В отличие от математической статистики, с которой машинное обучение довольно тесно связано, оно имеет дело с большими и сложными наборами данных, к которым практически невозможно применить классические методы статистического анализа. Машинное обучение — практическая дисциплина, в которой идеи чаще доказываются эмпирически, а не теоретически с опорой на мощную математическую платформу.

Машинное обучение включает в себя:

- контрольные входные данные,
- примеры ожидаемых результатов,
- способ оценки качества работы алгоритма.

Модель машинного обучения «обучается» на известных примерах входных данных и результатов, т.е. она трансформирует данные, поданные ей на вход, в значимые для исследователя результаты.

Главной задачей машинного обучения является значимое *преобразование* данных, т.е. самое важное — это обучение *представлению* входных данных, приближающемуся исследователя к ожидаемому результату. Машинное обучение формулирует иную парадигму программирования, где данные кодируются иначе, где программа не просто кодируется, а «обучается» сама.

Поиск наилучшего *представления* осуществляется из *пространства гипотез*, т.е. из множества возможных представлений. Обучение в контексте машинного обучения описывает автоматический процесс поиска лучшего представления, причем поиску наилучшего способствует сигнал обратной связи. Благодаря сигналу обратной связи появляется возможность корректировать модель и добиваться большей точности результата.

О машинном обучении исследователи писали еще в 40-50х годах, некоторые методы, основанные на применении статистики к анализу данных, были известны и использовались еще до компьютерной эры (например, наивный байесовский алгоритм). Первая концепция нейронных сетей с обратной связью была представлена Вальтером Питтсом и Уорреном МакКуллочем в 1945 г. Позднее, в 1949 году, Дональд Хеббс открыл способ создания самообучающихся искусственных нейронных сетей. Однако настоящий расцвет данная тема переживает в последнее десятилетие. Развитие Интернета, появление огромных массивов и хранилищ данных позволили, наконец, раскрыть потенциал данной технологии [1].

Однако в последнее время из-за популярности подвида машинного обучения — глубокого обучения — может сложиться мнение о тождественности этих понятий, что является неверным. К машинному обучению в равной мере относятся:

- Вероятностное моделирование;
- Логистическая регрессия;
- Нейронные сети;
- Ядерные методы,
- Деревья решений.

1.4.1 Поверхностное и глубокое обучение

Принято разделять поверхностное и глубокое машинное обучение.

При применении методов поверхностного обучения входные данные преобразуются в одно или два последовательных пространства, обычно за счет довольно простых преобразований. Проблема в том, что для больших задач провести такое точно преобразование входных данных не всегда возможно (особенно «в ручном режиме»), поэтому этап конструирования признаков занимал много времени, требовалась проверка на ошибки.

Методы глубокого обучения автоматизируют этап конструирования признаков, что позволяет подавать на вход в модель машинного обучения очень большие массивы данных без их предварительной «ручной» обработки.

1.4.2 Глубокое обучение

Глубокое обучение (англ. *Deep Learning*) является подразделом машинного обучения, которое, в свою очередь, является подразделом искусственного интеллекта.

Рисунок 5. Глубокое обучение как часть ИИ

Правильно будет сказать, что система, использующая методы глубокого обучения, является системой искусственного интеллекта, но систему искусственного интеллекта можно построить, опираясь и на иные методы. К сожалению, из-за упрощения понятий для создания рекламных роликов в 2010-х годах XIX века данные понятия широкими массами воспринимаются как тождественные.

Основоположник библиотеки *Keros* Ф.Шолле дает следующее определение глубокому обучению. Глубокое обучение — это новый подход к поиску представления данных, делающий упор на изучение последовательных слоев (или уровней) все более значимых представлений.

«Глубина модели» глубокого обучения — количество слоев, на которые делится модель данных. Иногда говорят о глубоком обучении как о многослойном или иерархическом обучении. Современное глубокое обучение вовлекает в процесс десятки или сотни слоев представления Слои автоматически определяются под воздействием обучающих данных.

Ошибочно считать, что глубокое обучение напрямую связано с нейробиологией, а искусственная нейронная сеть есть цифровой аналог мозга. Это не так. Корректнее будет рассматривать глубокое обучение как математическую основу для изучения представления данных.

Рассмотрим пример, как сеть в несколько слоев решает задачу распознавания написанных от руки цифр. Сеть получает изображение, затем поэтапно преобразует образ цифры в представление, все больше отличающееся от исходного и несущее в себе все больше полезной информации. Для ассоциации можно представить себе процесс фильтрации, т.е. поэтапной очистки информации. С технической точки зрения глубокое обучение — это многоступенчатый способ получения представления данных.

Каждый слой преобразует данные согласно назначенному ему *весу* или *параметру* слоя. Под обучением можно понимать поиск наилучшего набора весов, при котором сеть правильно отображает входные данные в соответствующие результаты.

Чтобы понимать, насколько «далек» полученный сетью результат от ожидаемого, необходимо определить функцию потерь (целевую функцию). *Функция потерь* принимает предсказание, выданное сетью, и истинное значение, после чего вычисляет оценку расстояния между ними. Эта оценка отражает, насколько хорошо сеть справилась с конкретным примером [3].

Использование этой оценки для корректировки значений весов с целью уменьшения потерь в текущем примере является основной хитростью глубокого обучения. Задачу корректировки выполняет оптимизатор, реализующий алгоритм обратного распространения ошибки, считающийся центральным алгоритмом глубокого обучения.

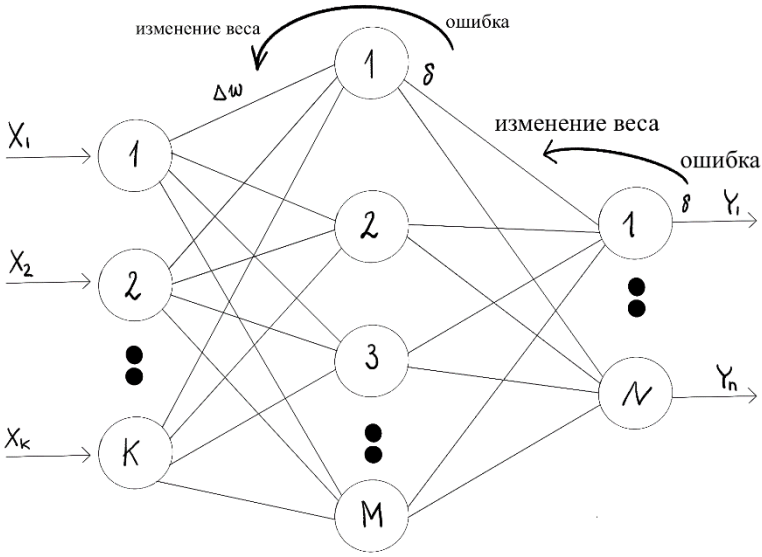

Рисунок 6. Искусственная нейронная сеть

Цикл обучения сети (итерационная обработка) повторяется достаточное количество раз и порождает весовые значения, минимизирующие функцию потерь. Сеть с минимальными потерями, возвращающая результаты, близкие к истинным, называется обученной сетью. После того, как сеть обучена, на вход ей можно подавать совершенно новые для нее данные, для которых заранее исследователю соответствующие результаты неизвестны, и именно сеть их обнаружит.

Глубокое обучение несмотря на то, что является давним разделом машинного обучения, получило огромное развитие с начала 2010-х годов. Прорывные, почти революционные, успехи в данной области заметны в решении задач моделирования восприятия — зрения и слуха — задач, естественных и понятных каждому человеку, но до недавнего времени трудно поддающихся решению с помощью компьютера.

В традиционно сложных областях машинного обучения глубокое обучения особенно хорошо показало себя в следующем:

- классификация изображений, распознавание речи и рукописного текста на уровне человека;
- улучшение качества машинного перевода между разными естественными языками;
- улучшение качества машинного чтения текста вслух (имитация интонаций);
- появление «умных» цифровых помощников;
- беспилотное (автоматическое) управление автомобилем;
- повышение точности целевой рекламы;
- повышение релевантности поиска в интернете;
- игра в Го сильнее человека.

1.4.3. Вопросы к разделу 1.4

1. Что подается на вход системе в парадигме машинного обучения?
2. Какие три компонента включает в себя машинное обучение?
3. В чем разницами между поверхностным и глубоким обучением?
4. Что такое «глубина» глубокого обучения?
5. Что такое «функция потерь»?

2. МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ К ПРАКТИЧЕСКИМ ЗАНЯТИЯМ

Практическая часть методического пособия по курсу «Системы искусственного интеллекта» разделена на две части:

- Задания с использованием *low-code* платформ.
- Задания с разбором кода с платформы *Kaggle* (https://www.kaggle.com/).

Данное ранжирование поможет студентам познакомиться с основными этапами работы с данными и моделями.

2.1. Инструментальные платформы для реализации кейсов к задачам глубокого обучения

Для реализации проектов по тематике глубокого обучения (как пример задач искусственного интеллекта) на данный момент создано несколько площадок. Это и платформы совместного доступа, и частные решения.

Рассмотрим *web*-платформы *Kaggle*, *Teachable machine* и *Google Colab*.

Kaggle

Kaggle — система организации конкурсов по исследованию данных, а также социальная сеть специалистов по обработке данных и машинному обучению. Авторизоваться можно через имеющийся *google*-аккаунт по ссылке: https://www.kaggle.com/ [14]. После присоединения к платформе открывается доступ к различным наборам данных (иначе — датасетам), регистрации на соревнования, секции комментариев и обсуждений. Датасет (англ. *dataset*) — это набор данных.

Платформа *Kaggle* для исследователей хороша тем, что на ней можно найти реальные задачки и уже подготовленные датасеты для того, чтобы набраться опыта в области глубокого обучения. Подойдет *Kaggle* и для уже состоявшихся специалистов в области машинного обучения.

После авторизации с помощью *google*-аккаунта, пользователю открывается главная страница (см. рисунок 7). Справа расположено краткое пользовательское меню, через которое можно перейти на свое страницу *Kaggle*-исследователя. Слева — меню самой платформы с возможностью перехода к разделам соревнований, датасетов, обсуждений и др. По центру размещаются актуальные тематические новости.

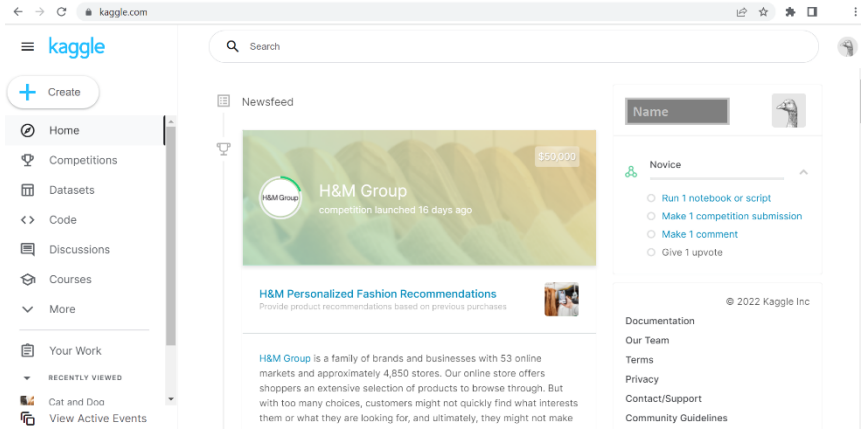

Рисунок 7. Главная страница *Kaggle*

Наиболее интересными для начинающего пользователя *Kaggle* на странице личного профиля (см. рисунок 8) являются разделы Соревнования (*Competitions*), Датасеты (*Datasets*), Код (*Code*). Имеется возможность отслеживать соревнования, в который принимается участие, видеть доступные и загруженные датасеты, управлять написанным кодом, причем свои блокноты (*Notebook*) можно оставлять как скрытыми, так и делиться ими с другими участниками *Kaggle*.

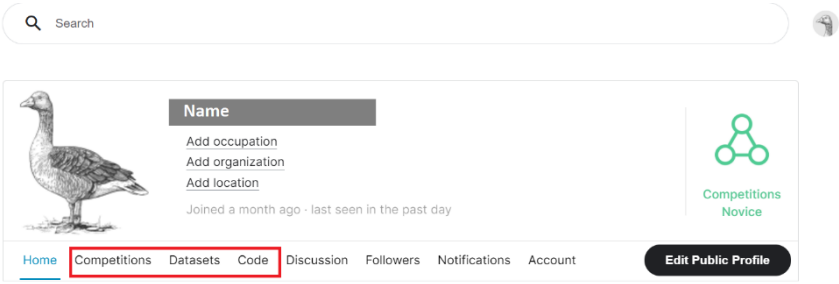

Рисунок 8. Профиль пользователя *Kaggle*

В разделе Датасетов (*Datasets*) участнику *Kaggle* доступны для просмотра и загрузки самые разные наборы данных (см. рисунок 9). Они сгруппированы по нескольким категориям, что облегчает поиск нужного набора данных под текущую задачу.

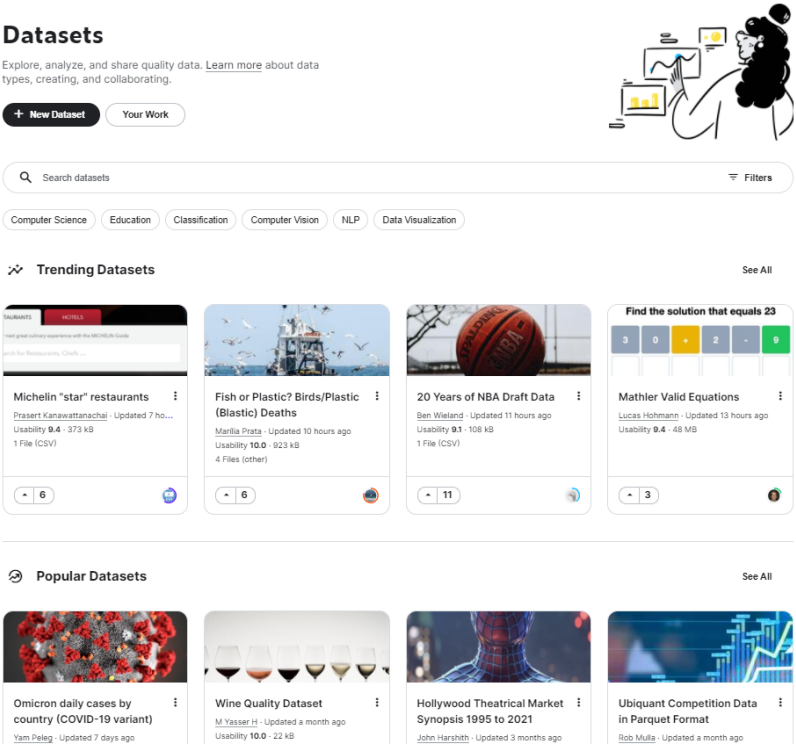

Рисунок 9. *Datasets* от *Kaggle*

В разделе Кода (*Code*) участники *Kaggle* могут как посмотреть решения, созданные другими участниками, так и создать свое собственное (см. рисунок 10). В данном разделе также присутствует разделение по категориям, что помогает сориентироваться при поиске типового решения под свою задачу.

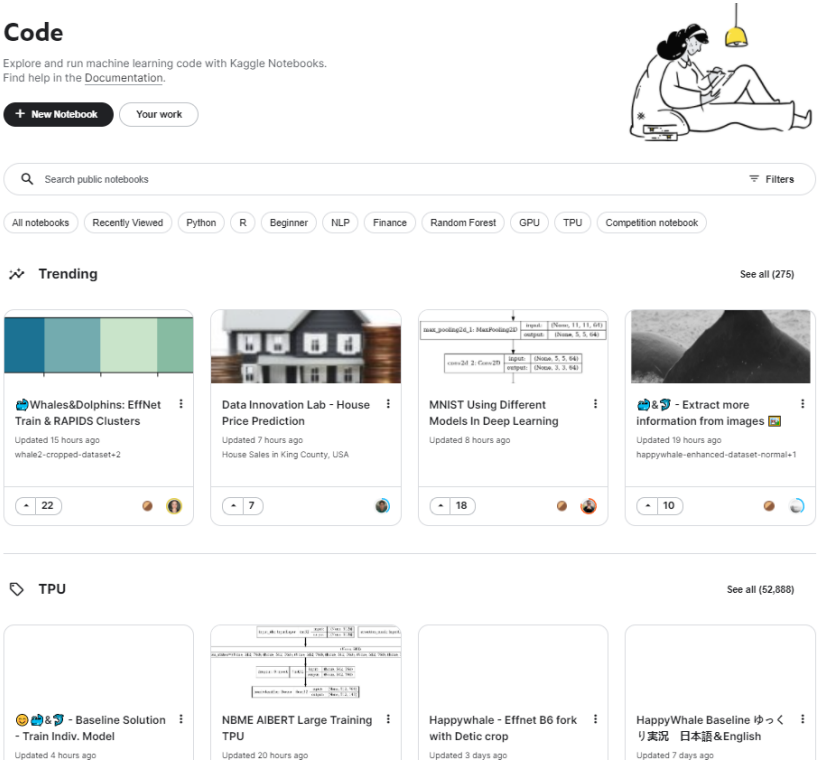

Рисунок 10. *Code* от *Kaggle*

Раздел Соревнований (*Competitions*) построен по тому же принципу: выделены категории соревнований, на начальном этапе использования платформы *Kaggle* будет рекомендовать соревнования «для новичков» (см. рисунок 11).

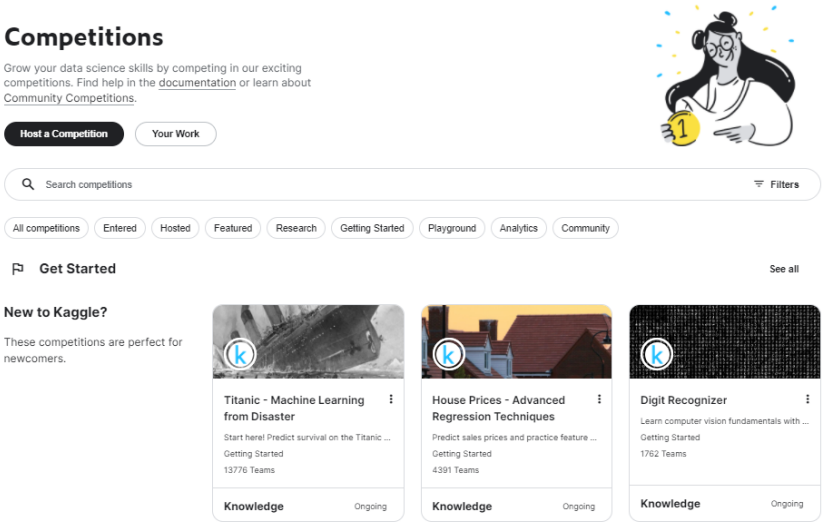

Рисунок 11. *Competitions* от *Kaggle*

Teachable machine

Teachable machine — это *web*-инструмент от *Google*, позволяющий создавать модели машинного обучения быстро и легко. Платформа доступна по ссылке: https://teachablemachine.withgoogle.com/ [15].

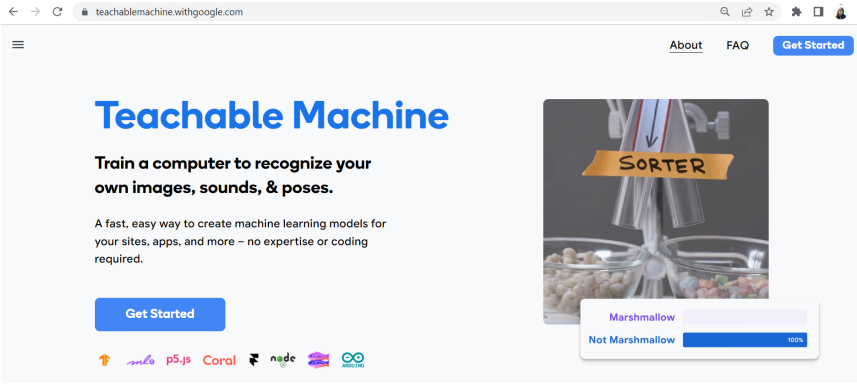

Рисунок 12. *Teachable machine*

Можно выгрузить с платформы модели в формате *TensorFlow.js*, это означает, что можно проводить с ними дальнейшую работу в любой среде разработке, поддерживающей *javascript*.

Авторизация происходит через *google*-аккаунт. На главной странице сервиса внимание привлекает кнопка «*Get started*», ведущая на страницу создания нового и открытия уже начатого проекта. Т.к. данный сервис еще достаточно молодой, на данный момент доступны модели для обучения распознаванию изображений, звука и поз человека, запечатленных веб-камерой.

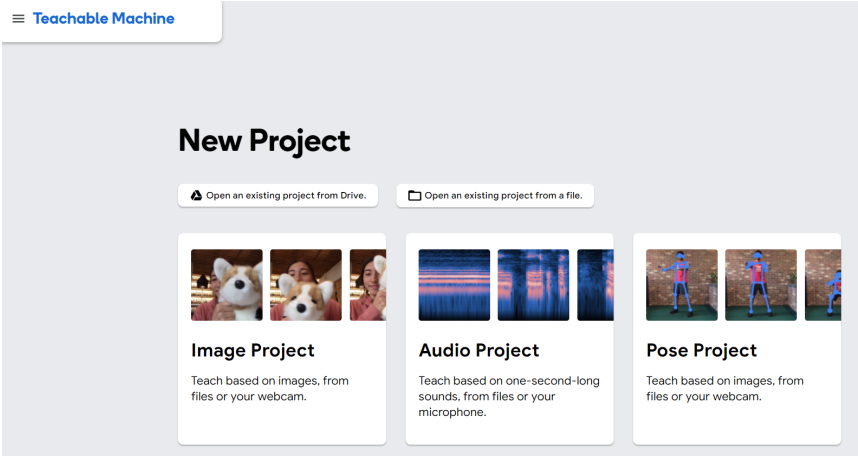

Рисунок 13. Модели для проектов в *Teachable machine*

Google Colab

Google Colab — это разработка от компании *Google* [16], один из облачных сервисов, позволяющий писать, тестировать и запускать код, написанный на языке *Python*. Созданные в *Colab* проекты можно сохранять в *GitHub*.

Google Colab — это бесплатный облачный сервис на основе *Jupyter Notebook*. *Google Colab* предоставляет все необходимое для

машинного обучения прямо в браузере, дат бесплатный доступ к невероятно быстрым *GPU* и *TPU*. [17]

Jupyter Notebook — это среда разработки, где сразу можно видеть результат выполнения кода и его отдельных фрагментов. Отличие от традиционной среды разработки в том, что код можно разбить на куски и выполнять их в произвольном порядке. В *Jupyter Notebook* есть вывод результата сразу после фрагмента кода. Можно прямо в середине кода увидеть построенный график, получить предварительные результаты или любую другую визуализацию [18].

Для работы с *Jupyter Notebook* в облаке, т.е. в *Google Colab*, пользователю достаточно запустить браузер и открыть нужную страницу. После этого облачная система выделит ресурсы и позволит выполнять любой код. Несомненный плюс подхода в том, что пользователю не нужно ничего устанавливать на компьютер — облако всё берёт на себя, пользователь лишь пишет и запускает код.

На главной странице платформы (см. рисунок 14) для пользователя доступен блок актуальных новостей, меню, позволяющие переходить к разделам создания и отладки кода и др.

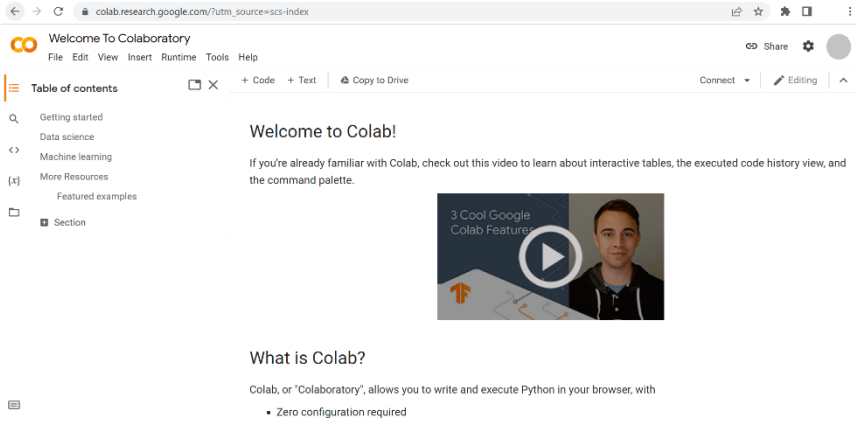

Рисунок 14. *Google Colab*

В режиме кодирования среда показывает, сколько ресурсов пользователю выделено, имеется контекстные подсказки при написании кода.

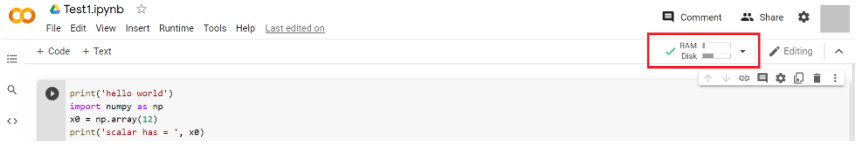

Рисунок 15. *Google Colab Code*

2.2. Вводный кейс

Напомним, что *Teachable machine* — площадка, представляющая возможность работать с данными и получать результаты анализа без особого усилия. В разделах 2.2 и 2.3 рассмотрим два кейса для знакомства со средой и основными понятиями данной области.

Первый кейс назовем «*Cats & Dogs*». Наша задача — создать устойчивый алгоритм распознавания животных по изображению таким образом, чтобы с вероятностью больше 80% происходило правильное распознавание. Распознаванием будет заниматься нейронная сеть, которую мы предварительно обучим на подготовленном наборе данных. Т.к. мы разбираем пример, в котором не потребуется программировать все с нуля, важно понимать правильную последовательность действий при использовании *low-code* платформ.

Нам понадобится:

- наборы данных для обучения (обучающая выборка) нейронной сети, разделенные на два класса: изображения кошек и изображения собак;
- наборы данных для тестирования (тестовая выборка), также заранее разделенные на два класса, именно по ним мы будем проверять, как хорошо обучилась наша сеть и способна ли она распознавать изображение, а также с какой точностью она это делает;

- доступ к платформе *Teachable machine*;
- доступ к платформе *Google Colab* и знание самых основ языка программирования *Python*.

Сначала требуется скачать датасет (обучающий и тестовый наборы) с *Kaggle*: https://www.kaggle.com/tongpython/cat-and-dog.

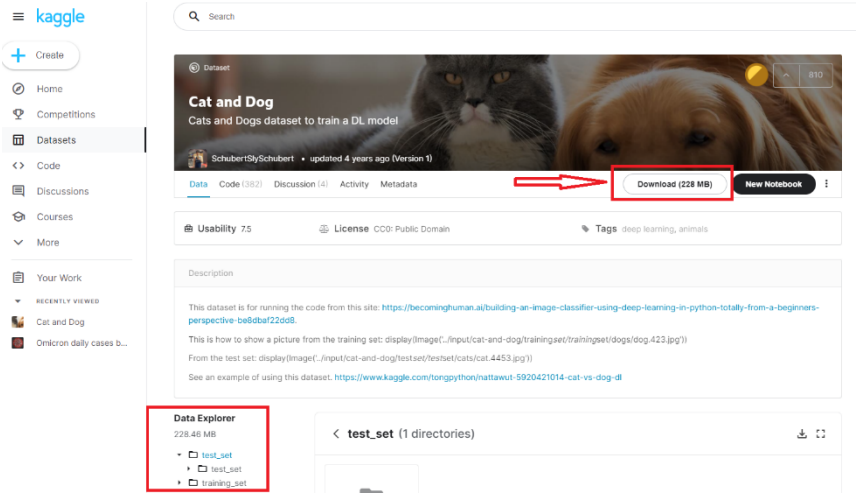

Рисунок 16. Скачивание набора данных

Затем нужно зайти на онлайн инструмент от *Google*: https://teachablemachine.withgoogle.com/ и начать новый проект, выбрав стандартную модель для распознавания изображений.

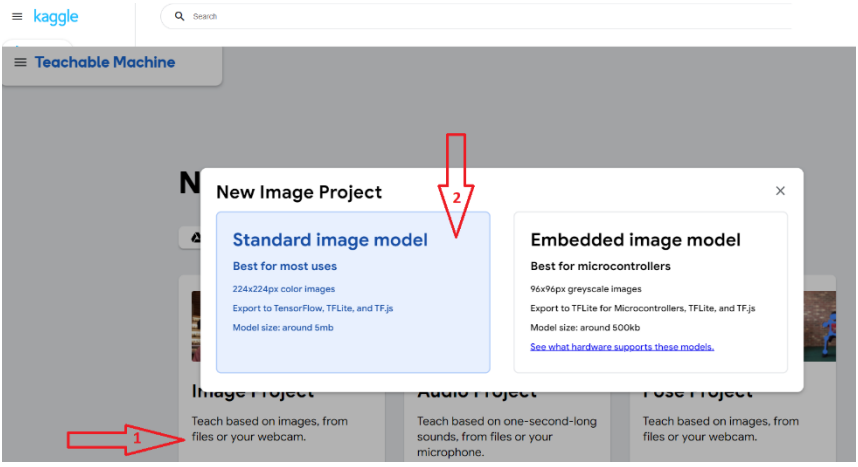

Рисунок 17. Создание новой модели

После этого, надо дать имена классам («cats» и «dogs»), загрузить соответствующие изображения обучающей выборки.

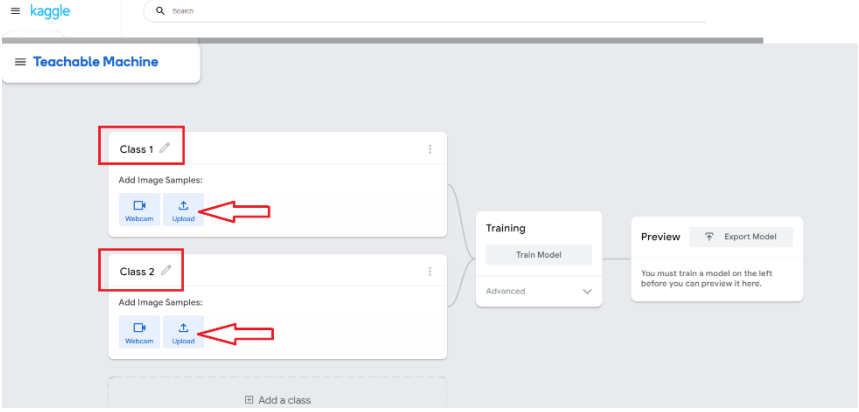

Рисунок 18. Подготовка к обучению, создание классов и загрузка данных

Теперь нужно обучить модель, следуя инструкции платформы (см. рисунок 19). При обучении модели нельзя закрывать окно браузера. Параметры обучения можно настраивать, вам доступны:

- изменение количества эпох (*Epochs*) обучения. Эпоха обучения означает, кто каждый элемент обучающей выборки прошел через сеть хотя бы один раз;
- изменение размера пакета (*Batch size*). Размер пакета — это количество элементов обучающей выборки, используемые при одной итерации обучения.

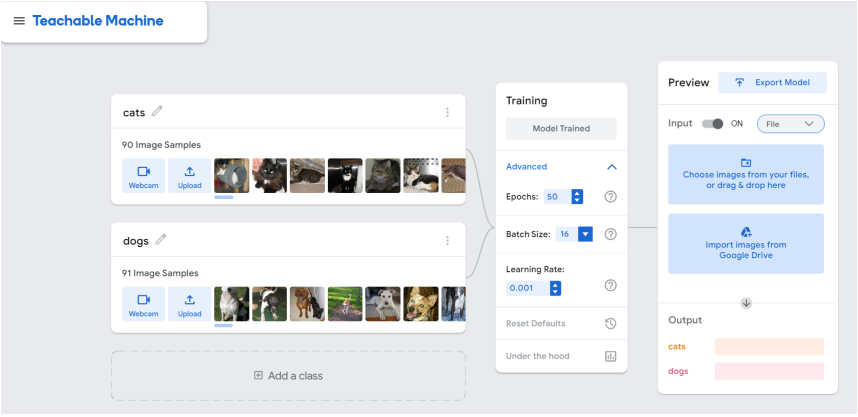

Рисунок 19. Тестирование и переобучение

После завершения обучения, справа откроется меню загрузки тестовых изображений для проверки корректности распознавания. Если результаты не устраивают исследователя, модель всегда можно переобучить и попробовать распознать тестовую выбору еще раз.

После проверки распознавания можно сохранить и загрузить модели в *Google Colab* с помощью готового кода.

Затем нам предстоит доработать код следующим образом:

1. Загрузка нужных файлов на *Google Colab* осуществима с помощью следующего кода:
```
from google.colab import files
files.upload()
```
2. Проверка наличия файлов осуществима через команду:
```
!ls
```
3. В сформированном на Teachable machine коде необходимо заменить путь к файлу распознаваемого изображения на

актуальный:

```
'<IMAGE_PATH>'    'cat.jpg'
```

4. Нужно создать список с метками наших классов (0 — кошка, 1 — собака):

```
label = ['cat', 'dog']
```

5. Теперь можно вывести результат распознавания:

```
print(label[np.argmax(prediction)])
```

На этом работа с вводным кейсом завершена.

Предлагаем студентам ознакомиться с еще одним популярным и наглядным туториалом от *Teachable Machine* — *Bananameter*: https://medium.com/@warronbebster/teachable-machine-tutorial-bananameter-4bfffa765866

2.3. Кейс библиотека PyCaret

Язык *Python* особенно хорош и популярен за счет наличия большого количества самых разнообразных библиотек, помогающих как начинающим, так и профессиональным программистам решать широкий класс задач. Разумеется, и в области машинного обучения тоже.

PyCaret — это библиотека машинного обучения с открытым исходным кодом на *Python*, которая позволяет создавать и развертывать модели машинного обучения с минимальным кодированием.

По сути, *PyCaret* — это альтернатива с низким кодом, которая может заменить сотни строк кода всего несколькими словами. Это значительно увеличивает скорость разработки программного обеспечения и делает его более доступным для начинающих. *PyCaret* — это оболочка *Python* для нескольких библиотек машинного обучения, таких как *scikit-learn*, *XGBoost*, *Microsoft LightGBM*, *spaCy* и многих других [19].

Начало работы с *PyCaret*: создать новый проект на платформе *Google Colab*. Для установки библиотеки в *Google Colab* необходимо использовать следующую команду:

```
!pip install pycaret
```

После процесса установки можно начать работать с библиотекой. Первым этапом является получение данных, для этого можно использовать функцию *get_data* из модулей *pycaret.datasets*. Для примера рассмотрим датасет людей, имеющих заболевание диабет. Цель состоит в том, чтобы предсказать результат пациента (в двоичном формате 0 или 1) на основе нескольких факторов, таких как давление, уровень инсулина в крови, возраст и т.д. Этот датасет доступен на *GitHub*-репозитории *PyCaret*. Чтобы его импортировать в свой проект, нужно воспользоваться командой *import* и затем указать конкретный датасет для загрузки:

```
import pycaret
from pycaret.datasets import get_data
diabetes = get_data('diabetes')
```

В рамках самостоятельной работы можно посмотреть другие датасеты и проделать аналогичные исследования. Посмотреть полный список доступных датасетов можно, используя команду:

```
index = get_data('index')
```

Продолжим работу с данными о диабетиках. Любой эксперимент с машинным обучением в *PyCaret* начинается с настройки среды путем импорта необходимого модуля и инициализации *setup*(). Модуль, который будет использоваться в этом примере — это *pycaret.classification*.

После импорта модуля *setup*() инициализируется путем определения датафрейма (‘diabetes’) и целевой переменной (‘Class variable’).

```
from pycaret.classification import *
exp1 = setup(diabetes, target = 'Class variable')
```

Весь препроцессинг происходит в *setup*(). Под препроцессингом или преодобработкой (*Data Preprocessing*) будем

понимать подготовку данных, приведение их в соответствии с требованиями, определяемыми спецификой решаемой задачи.

Предобработка данных включает два направления: очистку и оптимизацию.

Очистка производится с целью исключения различного рода факторов, снижающих качество данных и мешающих работе аналитических алгоритмов. Она включает обработку дубликатов, противоречий и фиктивных значений, восстановление и заполнение пропусков, сглаживание, подавление шума и редактирование аномальных значений.

Оптимизация данных как элемент предобработки включает снижение размерности, выявление и исключение незначащих признаков. Основное отличие оптимизации от очистки в том, что факторы, устраняемые в процессе очистки, существенно снижают точность решения задачи или делают работу аналитических алгоритмов невозможной. Проблемы, решаемые при оптимизации, адаптируют данные к конкретной задаче и повышают эффективность их анализа [20].

Задействуя более двадцати функций для подготовки данных перед машинным обучением, *PyCaret* создает пайплайн (*Pipeline*) преобразований на основе параметров, определенных в функции *setup*(). Он автоматически выстраивает все зависимости в пайплайне, поэтому не нужно вручную управлять последовательным выполнением преобразований на тестовом или новом (невидимом) датасете.

Пайплайн *PyCaret* можно легко переносить из одной среды в другую или развернуть на продакшене (*Production*). Ниже студенты могут ознакомиться с функциями препроцессинга, которые доступны в *PyCaret* с первого релиза (*Release*).

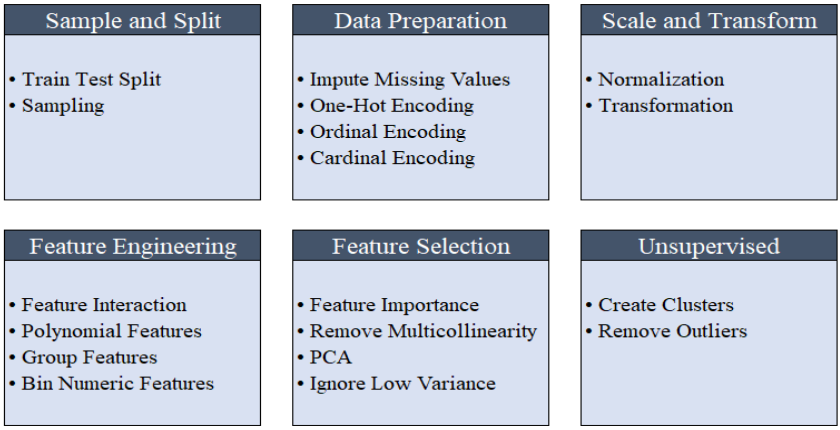

Рисунок 20. *Data Preprocessing PyCaret*

Шаги препроцессинга данных обязательные для машинного обучения, такие как дополнение пропущенных значений, кодирование качественных переменных, кодирование лейблов («да» или «нет» в 1 или 0) и *train-test-split*, выполняются автоматически при инициализации *setup*().

Сравнение моделей — это первый шаг, который рекомендуется выполнить при работе с обучением с учителем (классификация или регрессия). Эта функция обучает все модели в библиотеке моделей и сравнивает между собой оценочный показатель с помощью кросс-валидации по *K*-блокам (по умолчанию 10 блоков). Оценочные показатели используются следующие:

- Для классификации: *Accuracy*, *AUC*, *Recall*, *Precision*, *F1*, *Kappa*.
- Для регрессии: *MAE*, *MSE*, *RMSE*, *R2*, *RMSLE*, *MAPE*.

Кросс-валидация — это метод оценки аналитической модели и ее поведения на независимых данных с наиболее равномерным использованием имеющихся данных. В основе метода лежит разделение исходного множества данных на k примерно равных

блоков, например на пять (k = 5). Затем на (k – 1), т.е. на четырех блоках производится обучение модели, а пятый блок используется для тестирования. Процедура повторяется k раз, при этом на каждом проходе для проверки выбирается новый блок, а обучение производится на оставшихся [20].

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lr	Logistic Regression	0.7706	0.8165	0.6782	0.7024	0.6901	0.5080	0.5082	0.4500
ada	Ada Boost Classifier	0.7706	0.8112	0.7471	0.6771	0.7104	0.5212	0.5229	0.0800
ridge	Ridge Classifier	0.7662	0.7420	0.6437	0.7089	0.6747	0.4929	0.4943	0.0100
lda	Linear Discriminant Analysis	0.7576	0.8170	0.6667	0.6824	0.6744	0.4814	0.4814	0.0200
gbc	Gradient Boosting Classifier	0.7403	0.8116	0.7126	0.6392	0.6739	0.4591	0.4610	0.1300
et	Extra Trees Classifier	0.7273	0.7915	0.6322	0.6395	0.6358	0.4179	0.4179	0.2000
catboost	CatBoost Classifier	0.7229	0.8123	0.7011	0.6162	0.6559	0.4256	0.4281	1.6600
rf	Random Forest Classifier	0.7186	0.8107	0.6667	0.6170	0.6409	0.4101	0.4110	0.2500
xgboost	Extreme Gradient Boosting	0.7143	0.7802	0.7011	0.6040	0.6489	0.4103	0.4136	0.2200
lightgbm	Light Gradient Boosting Machine	0.7100	0.7907	0.7011	0.5980	0.6455	0.4027	0.4063	0.0800
knn	K Neighbors Classifier	0.7056	0.7310	0.5747	0.6173	0.5952	0.3644	0.3650	0.0100
nb	Naive Bayes	0.6797	0.7349	0.3793	0.6226	0.4714	0.2606	0.2771	0.0000
dt	Decision Tree Classifier	0.6407	0.6390	0.6322	0.5189	0.5699	0.2665	0.2703	0.0000
dummy	Dummy Classifier	0.6234	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
qda	Quadratic Discriminant Analysis	0.5411	0.5002	0.3793	0.3882	0.3837	0.0183	0.0183	0.0000
svm	SVM - Linear Kernel	0.4589	0.5500	0.9195	0.4040	0.5614	0.0799	0.1386	0.0100

Рисунок 21. Значения показателей

По умолчанию показатели оцениваются с помощью кросс-валидации по 10 блокам. Количество блоков можно изменить, поменяв значение параметра *fold*.

Таблица по умолчанию отсортирована по «*Accuracy*» от самого высокого значения, к самому низкому. Порядок сортировки также можно изменить с помощью параметра *sort*.

Создать модель в любом модуле *PyCaret* так просто, что нужно просто написать *create_model*. На вход функция принимает

один параметр, т.е. имя модели, передаваемое в виде строки. Эта функция возвращает таблицу с кросс-валидированными оценками и объект обученной модели.

```
adaboost = create_model('ada')
```

В переменной «*adaboost*» хранится объект обученной модели, который возвращает функция *create_model*. Доступ к исходным атрибутам обучаемого объекта можно получить с помощью функции *period* (.) после переменной.

В *PyCaret* больше шестидесяти готовых к использованию алгоритмов с открытым исходным кодом. Полный список оценщиков (моделей), доступных в *PyCaret*, можно найти по ссылке:
https://pycaret.gitbook.io/docs/get-started/functions/others#models

Функция *tune_model* используется для автоматической настройки гиперпараметров модели машинного обучения. *PyCaret* использует *random grid search* в определенном пространстве поиска. Функция возвращает таблицу с кросс-валидированными оценками и объект обученной модели.

```
tuned_adaboost = tune_model(adaboost)
```

Функция *tune_model* в модулях обучения без учителя, таких как *pycaret.nlp*, *pycaret.clustering* и *pycaret.anomaly*, может использоваться совместно с модулями обучения с учителем. Например, модуль *NLP* (*Natural Language Processing*) в *PyCaret* может использоваться для настройки параметра *number of topics* путем оценки объективной функции или функции потерь из модели с учителем, такой как «*Accuracy*» или «*R2*».

```
ensemble_adaboost = ensemble_model(adaboost)
```

Также *PyCaret* предоставляет функции *blend_models* и *stack_models* для объединения нескольких обученных моделей.

Оценить производительность и провести диагностику обученной модели машинного обучения можно с помощью

функции *plot_model*. Она принимает в себя объект обученной модели и тип графика в виде строки.

```
plot_model(adaboost, plot = 'auc')
plot_model(adaboost, plot = 'boundary')
plot_model(adaboost, plot = 'pr')
plot_model(adaboost, plot = 'vc')
```

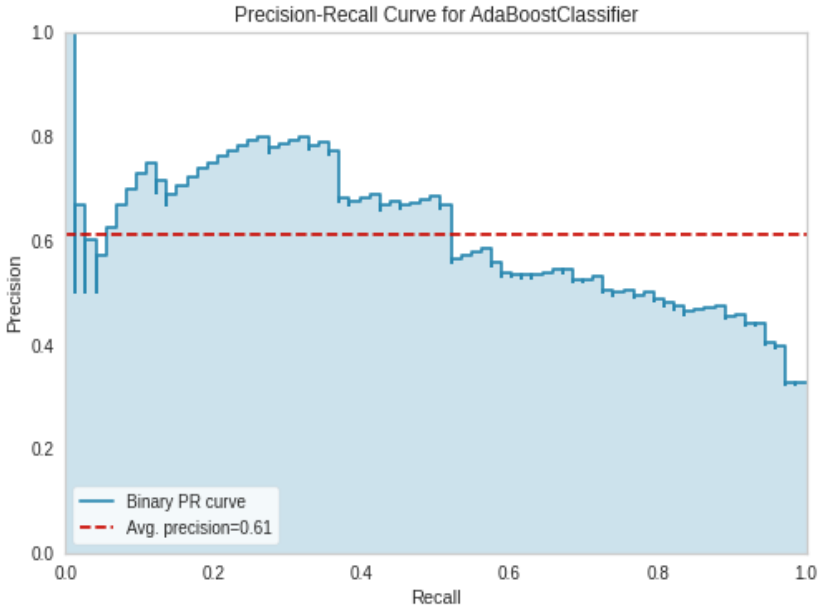

Рисунок 22. Использование *plot_model*

По ссылке можно узнать больше о визуализации в *PyCaret*: https://pycaret.gitbook.io/docs/get-started/functions

Также можно использовать функцию *evaluate_model*, чтобы увидеть графики с помощью пользовательского интерфейса *notebook*.

```
evaluate_model(adaboost)
```

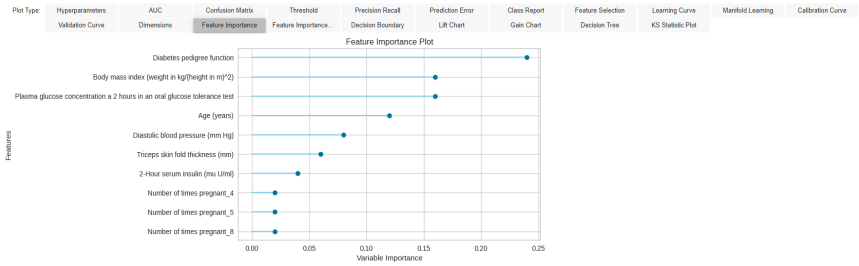

Рисунок 23. Использование *evaluate_model*

Когда данные нелинейные, что случается в реальной жизни достаточно часто, мы неизменно видим, что древовидные модели работают гораздо лучше, чем простые гауссовские модели. Однако это происходит за счет потери интерпретируемости, поскольку древовидные модели не обеспечивают простых коэффициентов, как линейные модели. *PyCaret* реализует *SHAP* (*SHapley Additive exPlanations*) с помощью функции *interpret_model*.

```
dt = create_model('dt')
interpret_model(dt)
```

До этого момента, результаты основывались на кросс-валидации по *K*-блокам на обучающем датасете (по умолчанию 70%). Для того, чтобы увидеть прогнозы и производительность модели на тестовом/*hold-out* датасете, используется функция *predict_model*.

```
predict_model(adaboost)
```

Функция *predict_model* используется для составления прогноза невидимого датасета. На практике, функция *predict_model* будет использоваться итеративно, каждый раз на новой невидимой части датасете.

После окончания обучения весь пайплайн, содержащий все преобразования препроцессинга и объект обученной модели, можно сохранить в бинарном *pickle*-файле.

```
save_model (adaboost, model_name = 'ada_for_deployment')
```

На официальном сайте библиотеки представлены титуриалы по темам, в рамках самостоятельной работы студентам предлагается изучить их и апробировать:

- Regression https://github.com/pycaret/pycaret/blob/master/tutorials/Regression%20Tutorial%20Level%20Beginner%20-%20REG101.ipynb
- Multiclass Classification https://github.com/pycaret/pycaret/blob/master/tutorials/Multiclass%20Classification%20Tutorial%20Level%20Beginner%20-%20MCLF101.ipynb
- Natural Language Processing https://github.com/pycaret/pycaret/blob/master/tutorials/Natural%20Language%20Processing%20Tutorial%20Level%20Beginner%20-%20NLP101.ipynb

Также возможно прохождение других уроков по ссылке: https://pycaret.gitbook.io/docs/get-started/tutorials

Дополнительным заданием в изучении *low-code* платформ может стать изучение систем по ссылке: https://serokell.io/blog/top-no-code-platforms

2.4. Кейс библиотека FER

Следующее задание посвящено знакомству с библиотекой *Face Expression Recognition* (*FER*), направленную на распознавание человеческих эмоций по фото.

Распознавание эмоций — горячая тема в сфере искусственного интеллекта. К наиболее интересным областям применения подобных технологий можно отнести: распознавание состояния водителя, маркетинговые исследования, системы видеоаналитики для умных городов, человеко-машинное взаимодействие, мониторинг учащихся, проходящих *online*-курсы, носимые устройства и др.

Многие современные роботы, в том числе голосовые помощники, такие как Алиса, *Siri*, *Alexa* уже давно и достаточно успешно способны имитировать поведение человека и отвечать

сентиментально. Это также относится к когнитивно-поведенческой терапии, которая занимается тревожными расстройствами у пациентов, которые постоянно испытывают эмоциональное напряжение.

Для банка анализ эмоционального фона клиентов может помочь в увеличении продаж продуктов, а также объективной оценке сервиса и услуг. Также, анализ эмоций сотрудника может помочь в выявлении проблем в бизнес-процессах, а также в предотвращении принятия фатальных решений в управлении сложной системой. Поэтому, учитывая масштаб экосистем, применение подобных технологий выглядит крайне актуальным и перспективным.

Основная задача любой системы распознавания эмоций состоит в том, чтобы изолировать полярность входных данных (текст, речь, выражение лица), чтобы понять, является ли представленное первичное настроение положительным, отрицательным или нейтральным. Основываясь на этом первоначальном анализе, алгоритмы затем часто копают глубже, анализируют интенсивность, чтобы определить такие эмоции, как удовольствие, счастье, отвращение, гнев, страх, удивление и т.д.

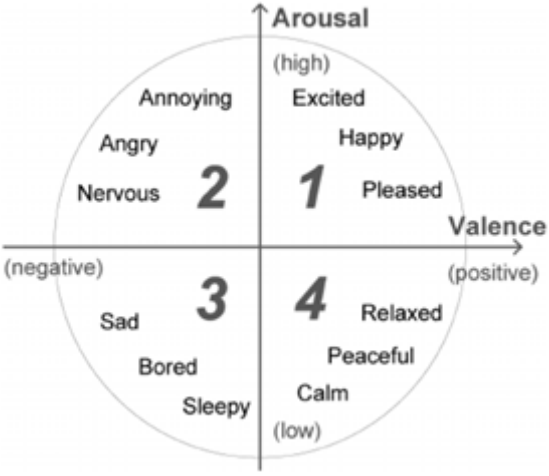

Рисунок 24. Круговая диаграмма эмоций

Такой анализ имеет два основных этапа исследований. Во-первых, это количественная оценка входных данных для алгоритмов для чтения и обработки, во-вторых, это психологические исследования, которые помогают определить, какое выражение обозначает конкретную эмоцию.

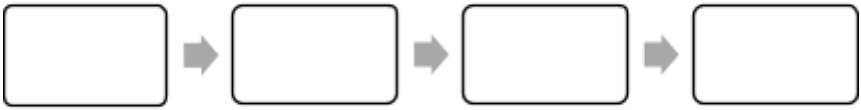

Рисунок 25. Алгоритм определения эмоции

С точки зрения вычислительных систем, когнитивная наука — это изучение научных процессов, происходящих в человеческом мозге. Она представляет Междисциплинарное научное направление, объединяющее теорию познания, когнитивную психологию, нейрофизиологию, когнитивную лингвистику, невербальную коммуникацию и теорию искусственного интеллекта [21].

Целью когнитивной науки является понимание и моделирование человеческого интеллекта с использованием всего спектра результатов и методологий смежных дисциплин. Предполагается, что, создавая компьютерные системы на основе знаний о человеческом интеллекте, машины смогут имитировать обучение и развивать интеллектуальные модели поведения, подобные человеческим.

Когнитивная наука действует на трех различных уровнях анализа:

1. Вычислительная теория: на этом уровне определяются цели анализа и передаются в компьютерную систему. Это может быть имитация речи или распознавание эмоций.

2. Представление и алгоритмы: в общих терминах машинного обучения (*Machine Learning* или сокращенно *ML*) это этап обучения. Здесь машине представляются идеальные сценарии ввода и вывода и вводятся в действие алгоритмы, которые в конечном итоге будут отвечать за преобразование ввода в вывод.
3. Аппаратная реализация: это заключительная фаза когнитивистики. Это внедрение алгоритма в реальном мире и анализ траектории его дальнейшего развития в отношении исследований человеческого мозга.

Распознавание эмоций на изображении. Разберем пример.

Face expression recognition (более известная как *FER*) представляет собой *Python* библиотеку с открытым исходным кодом, используемую для анализа настроений изображений. Проект построен на версии, которая использует сверточную нейронную сеть (*Convolutional neural network* или *CNN*), веса которой представлены в файле *HDF*5, присутствующем в исходном коде: https://github.com/justinshenk/fer. При необходимости, модель можно переобучить с помощью конструктора *FER* при вызове и инициализации модели.

MTCNN (*Multi-task Cascaded Neural Network*) является параметром конструктора. Это техника для распознавания лиц. Когда установлено значение «True», модель *MTCNN* используется для обнаружения лиц, а когда установлено значение «Ложь», функция использует классификатор *OpenCV Haar Cascade* по умолчанию.

detect_emotions(): эта функция используется для классификации обнаруженных эмоций и регистрирует выходные данные по шести категориям, а именно: «*angry*», «*disgust*», «*fear*», «*happy*», «*sad*», «*surprise*», «*neutral*». Каждая эмоция вычисляется, и результат оценивается по шкале от 0 до 1.

Чтобы установить *FER* и импортировать сопутствующие библиотеки, в новом проекте *Google Colab* необходимо воспользоваться уже хорошо известном нам командой *import*:

```
!pip install FER
from fer import FER
import matplotlib.pyplot as plt
%matplotlib inline
```

Для удобства тестирования работы сети будем использовать команду *!curl*, которая поможет по ссылке на картинку. Чтобы скопировать *URL* картинки, требуется кликнуть правой кнопкой мыши по картинке из Интернета и выбрать пункт всплывающего меню «Копировать *URL* картинки». Важно следить, чтобы *URL* вел на конкретную картинку (например, https://www.crushpixel.com/big-static19/preview4/handsome-man-laughing-inside-3373500.jpg).

Загруженную картинку назовем *predict.jpg* с помощью конструкции *--output*. Считаем полученную картинку.

```
!curl
https://www.crushpixel.com/big-static19/preview
4/handsome-man-laughing-inside-3373500.jpg
--output predict.jpg
test_image_one = plt.imread("predict.jpg")
```

Теперь используем библиотеку *FER*, обратим внимание на оставленные с помощью # комментарии в коде, они помогут при дальнейшей работе с кодом и облегчат последующее чтение алгоритма:

```
emo_detector = FER(mtcnn=True) # Когда
установлено значение «True», модель MTCNN
используется для обнаружения лиц, а когда
установлено значение «Ложь», функция использует
классификатор OpenCV Haar Cascade по умолчанию.
captured_emotions =
emo_detector.detect_emotions(test_image_one)
#Эта функция используется для классификации
обнаруженных эмоций и регистрирует выходные
данные по шести категориям, а именно: «angry»,
«disgust», «fear», «happy», «sad», «surprise»,
```

```
«neutral». Каждая эмоция вычисляется, и результат оценивается по шкале от 0 до 1.
    print(captured_emotions)      #визуализация вывода
    plt.imshow(test_image_one)    #визуализация картинки
```

Получаем результат распознавания.

```
[{'box': (469, 218, 501, 501), 'emotions': {'angry': 0.04, 'disgust': 0.0, 'fear': 0.06, 'happy': 0.88, 'sad': 0.02, 'surprise': 0.0, 'neutral': 0.0}}]
<matplotlib.image.AxesImage at 0x7f417b916750>
```

Рисунок 26. Результат работы алгоритма по распознаванию эмоций

Для вывода преобладающей эмоции на фото можно использовать следующий код:

```
    dominant_emotion, emotion_score = emo_detector.top_emotion(test_image_one)
    print(dominant_emotion, emotion_score)
```

Меняя ссылку на картинку или загружая собственную, можно проверить качество работы модели.

Благодаря подобным библиотекам и инструментам машинное обучение перестает быть чем-то далеким, помогает проводить анализ данных и выбор модели с минимальными трудозатратами!

2.5. Кейс обнаружение пожаров по фото

Алгоритмы ИИ могут решать важные задачи по сохранению объектов инфраструктуры и жизни людей. Например, распознавание пожаров, удаленный вызов пожарных бригад и оповещения людей.

Поставим следующую задачу: по снимку определить, есть ли на фото пожар или его нет. Для решения этой задачи воспользуемся датасетом с платформы *Kaggle* [22]:

https://www.kaggle.com/phylake1337/fire-dataset.

Изучим структуру каталога: есть два каталога (две папки), в одной находятся снимки с возгоранием, по второй размещены фотографии, на которых пожара нет.

После скачивания и разархивирования данных, их необходимо загрузить в *Google Colab* любым удобным способом.

Для работы нам потребуются следующие библиотеки:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import os
import tensorflow as tf
from tensorflow.keras.preprocessing import image
sns.set_style('darkgrid')
```

Получим датафрейм с путями к файлам и метками классов. Датафрейм (*DataFrame*) — двумерная структура данных, содержащая строки и столбцы. Можно сказать, что датафрейм является аналогом привычной нам таблицы. Датафрейм один из наиболее часто использующихся объектов библиотеки *Pandas* [23]. Далее представим код на *Python*, показывающий как получить нужный для нашей задачи датафрейм.

```
#create an empty DataFrame
df = pd.DataFrame(columns=['path','label'])
for dirname, _, filenames in os.walk('/content/drive/MyDrive/fire_dataset/fire_images'):
    for filename in filenames:
        df = df.append(pd.DataFrame([[os.path.join(dirname,
```

```
            filename),'fire']],columns=['path','
            label']))
for      dirname,      _,      filenames      in
os.walk('/content/drive/MyDrive/fire_dataset/no
n_fire_images'):
    for filename in filenames:
                                              df     =
df.append(pd.DataFrame([[os.path.join(dirname,
filename),'non_fire']],columns=['path','label']
))
    #print(os.path.join(dirname, filename))
    #shuffle the dataset for redistribute the
labels
    df                                            =
df.sample(frac=1).reset_index(drop=True)
    df.head(10)
```

Далее следует проанализировать данные. Самый простой способ — визуализировать их. Например, посмотрим количество объектов каждого класса. Построим диаграмму для части нашего датасета (фото, на которых пожар есть).

```
    fig    =    make_subplots(rows=1,    cols=2,
specs=[[{"type": "xy"}, {"type": "pie"}]])
    fig.add_trace(go.Bar(x
=df['label'].value_counts().index,y=df['label']
.value_counts().to_numpy(),marker_color=['darko
range','green'],showlegend=False),row=1,col=1)
    fig.add_trace(go.Pie(

values=df['label'].value_counts().to_numpy(),

labels=df['label'].value_counts().index,

marker=dict(colors=['darkorange','green'])),
        row=1, col=2)
```

Любым удобным способом отобразим сами картинки, например:

```
label = 'fire' #label for images with fire
data = df[df['label'] == label]
sns.set_style('dark')
pics = 6 #set the number of pics
fig,ax =
plt.subplots(int(pics//2),2,figsize=(15,15))
plt.suptitle('Images with Fire')
ax = ax.ravel()
for i in range((pics//2)*2):
    path =
data.sample(1).loc[:,'path'].to_numpy()[0]
    img = image.load_img(path)
    img = image.img_to_array(img)/255
    ax[i].imshow(img)
    ax[i].axes.xaxis.set_visible(False)
    ax[i].axes.yaxis.set_visible(False)
```

Аналогично поступим для изображений без пожара. Нужно изменить лэйбл вывода.

Необходимо проверить размер нескольких изображений — это важно для нормализации данных. В области машинного обучения под нормализацией данных понимают процедуру предобработки входной информации, при которой значения признаков во входном векторе приводятся к некоторому заданному диапазону. Необходимость нормализации обусловлена тем, что значения признаков могут изменяться в очень большом диапазоне и отличаться друг от друга на несколько порядков из-за различий по физическому смыслу. Дисбаланс между значениями признаков может вызвать неустойчивость работы модели, ухудшить результаты обучения [24].

Для проверки размера изображений в нашей задаче требуется написать следующий код:

```
def shaper(row):
```

```
                                        shape        =
image.load_img(row['path']).size
            row['height'] = shape[1]
            row['width'] = shape[0]
            return row
        df = df.apply(shaper,axis=1)
        df.head(5)
```

Так как все изображения имеют разный размер, можно визуализировать эти данные для простоты дальнейшего анализа.

```
        sns.set_style('darkgrid')
        fig,(ax1,ax2,ax3)                                =
plt.subplots(1,3,gridspec_kw={'width_ratios':
[3,0.5,0.5]},figsize=(15,10))
        sns.kdeplot(data=df.drop(columns=['path','l
abel']),ax=ax1,legend=True)
        sns.boxplot(data=df,y='height',ax=ax2,color
='skyblue')
        sns.boxplot(data=df,y='width',ax=ax3,color=
'orange')
        plt.suptitle('Distribution     of     image
shapes')
        ax3.set_ylim(0,7000)
        ax2.set_ylim(0,7000)
        plt.tight_layout()
```

Создадим бэтчи данных тензорного изображения с увеличением данных в реальном времени.

```
        from    tensorflow.keras.preprocessing.image
import ImageDataGenerator

        generator = ImageDataGenerator(
            rotation_range= 20,
            width_shift_range=0.1,
            height_shift_range=0.1,
            shear_range = 2,
            zoom_range=0.2,
```

```
        rescale = 1/255,
        validation_split=0.2,
    )
    train_gen =
generator.flow_from_dataframe(df,x_col='path',y
_col='label',images_size=(256,256),class_mode='
binary',subset='training')
    val_gen =
generator.flow_from_dataframe(df,x_col='path',y
_col='label',images_size=(256,256),class_mode='
binary',subset='validation')
```

Нормируем метки классов.

```
    class_indices = {}
    for key in train_gen.class_indices.keys():

class_indices[train_gen.class_indices[key]] =
key

    print(class_indices)
```

Теперь установлено соответствие 0 — пожар, 1 — отсутствие пожара.

```
    {0: 'fire', 1: 'non_fire'}.
```

Переходим к этапу создания модели. Импортируем нужные нам модули из библиотек.

```
    from tensorflow.keras.models import
Sequential
    from tensorflow.keras.layers import Conv2D,
MaxPool2D, Flatten, Dense
```

Теперь можно собрать сверточную нейронную сеть. Сверточная нейронная сеть (*convolutional neural network*, *CNN*) — это архитектура искусственных нейронных сетей, применяющаяся в задах распознавания образов. *CNN* использует операцию свертки, в процессе выполнения которой каждый фрагмент изображения умножается на матрицу свертки поэлементно, после чего просуммированный результат записывается в аналогичную

позицию выходного изображения. Сверточные нейронные сети обеспечивают частичную устойчивость к изменениям масштаба, смещениям, поворотам, смене ракурса и прочим искажениям. На данный момент сверточная нейронная сеть и ее модификации считаются лучшими по точности и скорости алгоритмами нахождения объектов. Начиная с 2012 года, нейросети занимают первые места на известном международном конкурсе по распознаванию образов *ImageNet* [24].

В рамках нашей задачи выполним:

```
model = Sequential()
model.add(Conv2D(filters=32,kernel_size = (2,2),activation='relu',input_shape = (256,256,3)))
model.add(MaxPool2D())
model.add(Conv2D(filters=64,kernel_size=(2, 2),activation='relu'))
model.add(MaxPool2D())
model.add(Conv2D(filters=128,kernel_size=(2 ,2),activation='relu'))
model.add(MaxPool2D())
model.add(Flatten())
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation = 'relu'))
model.add(Dense(1,activation = 'sigmoid'))
```

Посмотрим суммарные параметры и другие характеристики сети:

```
model.summary()
```

Далее нужно скомпилировать модель и перейти к ее обучению.

```
from tensorflow.keras.metrics import Recall,AUC
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
```

```
    model.compile(optimizer='adam',loss='binary
_crossentropy',metrics=['accuracy',Recall(),AUC
()])
    early_stoppping                                =
EarlyStopping(monitor='val_loss',patience=5,res
tore_best_weights=True)
    reduce_lr_on_plateau                           =
ReduceLROnPlateau(monitor='val_loss',factor=0.1
,patience=5)
    model.fit(x=train_gen,batch_size=32,epochs=
15,validation_data=val_gen,callbacks=[early_sto
ppping,reduce_lr_on_plateau])
```

Протестируем сеть на тестовом наборе данных.

```
    eval_list                                      =
model.evaluate(val_gen,return_dict=True)
```

Загрузим любую картинку для проверки. Сделать это можно с помощью специализированных команд, например:

```
    !curl
https://static.dw.com/image/58715155_303.jpg
--output predict.jpg
```

Считаем и визуализируем картинку.

```
    img = image.load_img('predict.jpg')
    img
```

Нормируем.

```
    img = image.img_to_array(img)/255
    img = tf.image.resize(img,(256,256))
    img = tf.expand_dims(img,axis=0)

    print("Image Shape",img.shape)
```

Распознаем.

```
    prediction                                     =
int(tf.round(model.predict(x=img)).numpy()[0][0
])
```

```
    print("The predicted value is: ",prediction,"and the predicted label is:",class_indices[prediction])
```

Данная модель может подойти и для решения предыдущей задачи.

2.6. Кейс NLP

Обработка естественного языка (*Natural Language Processing*, *NLP*) — пересечение машинного обучения и математической лингвистики, направленное на изучение методов анализа и синтеза естественного языка [25].

При наличии огромного количества информационных ресурсов все чаще встает задача подтверждения опубликованных данных.

Задача: определить настоящая ли новость.

Скачиваем с платформы *Kaggle* датасет по ссылке [26]: https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset.

Аналогично с предыдущим кейсом, требуется загрузить данные в *Google Colab*. Затем нужно импортировать нужные для решения задачи библиотеки.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.graph_objects as go
```

После импорта библиотек, считаем данные из *csv*-файлов и разметим классы. *CSV* (*Comma-Separated Values*) это файл в текстовом формате, содержащий построчное представление табличных данных, причем в строке данные разных полей в таблице разделены знаками (чаще всего запятыми) [27].

```
    fake_news_data = pd.read_csv("../content/drive/MyDrive/news/Fake.csv")
```

```
    true_news_data =
pd.read_csv("../content/drive/MyDrive/news/True
.csv")

    true_news_data['class'] = 0
    fake_news_data['class'] = 1
```

Собираем считанные данные в единый датафрейм.

```
    news_data =
pd.concat([true_news_data,fake_news_data])
    news_data =
news_data.sample(frac=1.0).reset_index(drop=Tru
e)
```

Визуализируем количество представителей каждого класса.

```
    colors = ['gold', 'mediumturquoise']
    labels = ['REAL','FAKE']
    values =
news_data['class'].value_counts()/news_data['cl
ass'].shape[0]
    fig = go.Figure(data=[go.Pie(labels=labels,
values=values, hole=.3)])
    fig.update_traces(hoverinfo='label+percent'
, textinfo='percent', textfont_size=20,

marker=dict(colors=colors,
line=dict(color='#000000', width=2)))
    fig.update_layout(
        title_text="Target Balance",
        title_font_color="white",
        legend_title_font_color="yellow",
        paper_bgcolor="black",
        plot_bgcolor='black',
        font_color="white",
    )
    fig.show()
```

Посмотреть количество можно с помощью следующего кода:

```
news_data['class'].value_counts()
```

Далее посмотрим количество представителей каждого класса с разбивкой по разделам.

```
news_data.subject.groupby(news_data['class'
]).value_counts()
```

Далее будут использованы библиотеки, которые требуют дополнительной установки, сделать это достаточно просто:

```
!pip install texthero
!pip install -U spacy
```

Импортируем установленные библиотеки

```
import texthero as hero
```

Эта библиотека поможет провести нормализацию текстовых данных. Чтобы это сделать, необходимо провести следующие манипуляции с текстом:

- приведение к единому реестру
  ```
  news_data['text'] =
  hero.lowercase(news_data['text'])
  ```
- удаление цифр
  ```
  news_data['text'] =
  hero.remove_digits(news_data['text'])
  ```
- удаление иллюстраций
  ```
  news_data['text'] =
  hero.remove_punctuation(news_data['text'])
  ```
- удаление различий в написании
  ```
  news_data['text'] =
  hero.remove_diacritics(news_data['text'])
  ```
- удаление всех видов скобок
  ```
  news_data['text'] =
  hero.remove_brackets(news_data['text'])
  news_data['text'] =
  hero.remove_angle_brackets(news_data['text
  '])
  news_data['text'] =
  hero.remove_curly_brackets(news_data['text
  '])
  ```

```
news_data['text'] =
hero.remove_round_brackets(news_data['text
'])
news_data['text'] =
hero.remove_square_brackets(news_data['tex
t'])
```

- удаление стоп-слов

```
news_data['text'] =
hero.remove_stopwords(news_data['text'])
```

- удаление пробелов

```
news_data['text'] =
hero.remove_whitespace(news_data['text'])
news_data['text'] =
news_data['text'].apply(lambda x: '
'.join([word for word in x.split() if
len(word) > 2 and word.isalpha() and word
!= 'reuters']))
```

Затем нужно импортировать дополнительные библиотеки, помогающие провести процесс лемматизации. Лемматизация (*lemmatization*) — это метод морфологического анализа, который сводится к приведению словоформы к ее первоначальной словарной форме (лемме). Метод лемматизации обычно применяется в поисковых алгоритмах в процессе схематизации веб-документов при их индексировании. В результате лемматизации от словоформы отбрасываются флективные окончания и возвращается основная или словарная форма слова [28].

В машинном обучении также присутствует понятие алгоритма *Stemming*. Алгоритм *Stemming* работает, вырезая суффикс из слова. В более широком смысле отсекает начало или конец слова. Лемматизация является более мощной операцией и учитывает морфологический анализ слов. Он возвращает лемму, которая является базовой формой всех ее флективных форм. Для создания словарей и поиска правильной формы слова необходимы глубокие лингвистические знания. Стемминг — это общая

операция, а лемматизация — интеллектуальная операция, в которой правильная форма будет выглядеть в словаре. Следовательно, лемматизация помогает в формировании лучших возможностей машинного обучения [29].

В рамках решаемой задачи для проведения лемматизации выполним следующее:

```
import nltk
from nltk.stem import WordNetLemmatizer
nltk.download('all')
news_data['text'] = news_data['text'].apply(lambda x: ' '.join([WordNetLemmatizer().lemmatize(word) for word in x.split()]))
```

С помощью следующего кода можно построить облако слов:

```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
wordcloud = WordCloud(
    width = 3000,
    height = 2000,
    background_color = 'black',
    stopwords = STOPWORDS).generate(str(news_data.loc[news_data['class'] == 0,"text"].values))

fig = plt.figure(
    figsize = (10, 10),
    facecolor = 'k',
    edgecolor = 'k')
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```

Далее проводим процесс векторизации. Векторизация позволит ускорить выполнение кода.

```
    from sklearn.feature_extraction.text import
TfidfVectorizer
    Tfidf_vect = TfidfVectorizer()
    Train_content_tfidf                          =
Tfidf_vect.fit_transform(news_data['text'])
```

Процесс первичной подготовки текстовых данных завершен. Теперь можно переходить к построению моделей. В данном кейсе мы попробуем сравнить качество работы логистической регрессии и один из видов рекуррентных сетей — *LSTM*.

Рекуррентная нейронная сеть (англ. *recurrent neural network*, *RNN*) — вид нейронных сетей, где связи между элементами образуют направленную последовательность [30].

Логистическая регрессия представляет собой контролируемый алгоритм классификации обучения, используемый для прогнозирования вероятности целевой переменной. Природа целевой или зависимой переменной дихотомична, что означает, что будет только два возможных класса [].

Для реализации логистической регрессии используем следующие библиотеки:

```
    from       sklearn.linear_model       import
LogisticRegression
    from      sklearn.model_selection      import
StratifiedKFold,cross_val_score
```

Разделяем данные на тестовые и тренировочные.

```
    from      sklearn.model_selection      import
train_test_split
    X_train,X_test,Y_train,Y_test                =
train_test_split(Train_content_tfidf,news_data[
'class'],test_size=0.2)
```

Построим модель логистической регрессии.

```
    LR = LogisticRegression()
    LR.fit(X_train,Y_train)
    Y_pred = LR.predict(X_test)
```

Проверим точность модели.

```
    from sklearn import metrics
```

```
    print(f'        Accuracy        Score        :
{round(metrics.accuracy_score(Y_test,Y_pred)*10
0,2)}%')
```

Построим матрицу ошибок.

```
    sns.heatmap(metrics.confusion_matrix(Y_test
,Y_pred), annot=True, fmt='d',cmap='YlGnBu')
    plt.xlabel('Predicted')
    plt.ylabel('Truth')
    plt.show()
```

Переходим к модели *LSTM*. Долгая краткосрочная память (*Long short-term memory*; *LSTM*) — особая разновидность архитектуры рекуррентных нейронных сетей, способная к обучению долговременным зависимостям [32].

В нашей задаче нужно импортировать библиотеки *Keras* и *TensorFlow*:

```
    import tensorflow as tf
    from tensorflow import keras
```

Затем нужно провести токенизацию. Токенизация — разбиение текста на слова и не-слова (то есть знаки препинания, границы абзацев и т.п). Полезность токенизации в машинном обучении — прямое донесение до нейронной сети факта, что человек (чьим действиям сеть надо научить подражать) воспринимает текст как поток слов, а не поток букв [32].

```
    MAX_NB_WORDS = 10000
    tokenizer                                    =
keras.preprocessing.text.Tokenizer(MAX_NB_WORDS
) # Selecting top 10000 words
    tokenizer.fit_on_texts(news_data['text'])
    train_data                                   =
tokenizer.texts_to_sequences(news_data['text'])
    word_index = tokenizer.word_index
```

На гистограмме можно увидеть распределение токенов. Нужно выполнить следующий код:

```
    sns.histplot([len(x)      for      x      in
train_data],bins=1000)
```

Подготовим данные на вход нейронной сети и разделим их на обучающую и тестовую выборку.

```
MAX_SEQUENCE_LENGTH = 600
EMBEDDING_DIM = 30
train_data = keras.preprocessing.sequence.pad_sequences(train_data, maxlen=MAX_SEQUENCE_LENGTH)
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(train_data,news_data['class'], test_size=0.2)
```

Построим, скомпилируем и обучим модель.

```
model = keras.Sequential()
model.add(keras.layers.Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=MAX_SEQUENCE_LENGTH))
model.add(keras.layers.LSTM(units=128))
model.add(keras.layers.Dense(1,activation='sigmoid'))
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
model.summary()
model.fit(X_train, Y_train, epochs=5,batch_size=1000, validation_split=0.1,verbose=1)
```

Проверим точность на тестовом наборе.

```
print(model.evaluate(X_test,Y_test, verbose=1))
```

И построим матрицу ошибок.

```
y_predicted = model.predict(X_test)
y_predicted_labels = [np.round(i) for i in y_predicted]
```

```
    sns.heatmap(tf.math.confusion_matrix(labels
=Y_test,predictions=y_predicted_labels),
annot=True, fmt='d',cmap='YlGnBu')
    plt.xlabel('Predicted')
    plt.ylabel('Truth')
    plt.show()
```

Выполнение кейса завершено. Если представлять общую схему алгоритма, подходящего для выполнения нашей задачи, то она будет следующей:

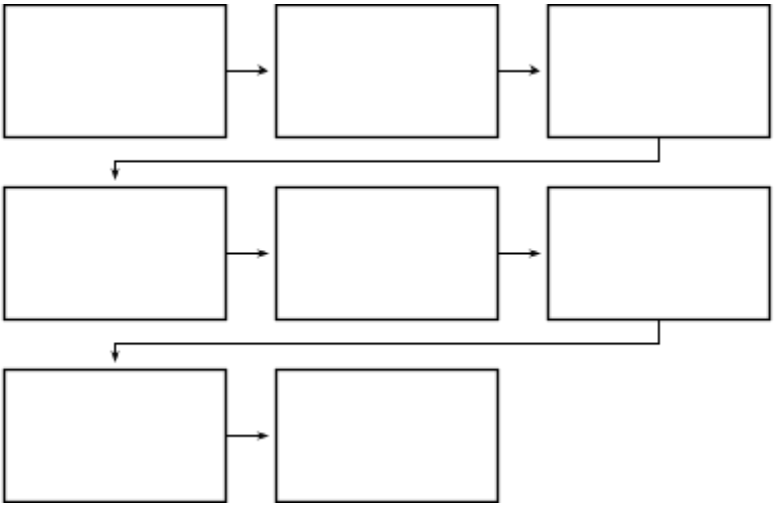

Рисунок 27. *NLP*

В рамках реализации проекта по теме можно использовать датасеты с сайта *Kaggle*, анализировать готовый код к выбранному датасету, производить его разбор, реализацию и выступление с результатами.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА ПО КУРСУ

1. Джонс М. Т. Программирование искусственного интеллекта в приложениях / М. Тим Джонс; Пер. с англ. Осипов А. И. – М.: ДМК Пресс, 2011. — 312 с.: ил.
2. [Электронный ресурс] А.Тьюринг, «Может ли машина мыслить?». Доступ — свободный. Дата последнего обращения: 22.01.2022. URL: http://www.ict.nsc.ru/jspui/bitstream/ICT/885/5/CantheMachinethink.pdf
3. Шолле Франсуа. Глубокое обучение на Python. — СПб.: Питер, 2018. — 400 с.: ил. — (Серия «Библиотека программиста»).
4. [Электронный ресурс] Станкевич Л.А. Интеллектуальные системы и технологии: учебник и практикум для среднего профессионального образования. Москва: Издательство Юрайт, 2022. — 397 с. URL: https://urait.ru/viewer/intellektualnye-sistemy-i-tehnologii-495988
5. [Электронный ресурс] BIG Picture. Режим доступа: свободный. Дата последнего обращения: 01.02.2022 г. URL: https://bigpicture.ru/kartiny-kotorye-narisoval-iskusstvennyj-intellekt-sozdannyj-google/
6. [Электронный ресурс] Google AI Experiments. Режим доступа: свободный. Дата последнего обращения: 02.02.2022 г. URL: https://experiments.withgoogle.com/collection/ai

7. [Электронный ресурс] IBM. Режим доступа: свободный. Дата последнего обращения: 02.02.2022 г. URL: https://www.ibm.com/us-en?lnk=m
8. [Электронный ресурс] Дата последнего обращения: 05.02.2022 г. URL: https://www.instapainting.com/ai-painter
9. [Электронный ресурс] Дата последнего обращения: 06.02.2022 г. URL: https://neurohive.io/ru/gotovye-prilozhenija/avatary-deepbrain-dlya-obshheniya-s-klientami/
10. [Электронный ресурс] Дата последнего обращения: 06.02.2022 г. URL: https://life-prog.ru/1_18238_sistemi-s-intellektualnim-interfeysom.html
11. Тельнов Ю.Ф. Издательство: Москва Год издания: 2004. — 82 с.
12. [Электронный ресурс] Theory & Practice. Дата последнего обращения: 06.02.2022 г. URL: https://theoryandpractice.ru/posts/17550-chto-takoe-iskusstvennyy-intellekt-ii-opredelenie-ponyatiya-prostymi-slovami
13. Перевод с французского Г.П. Гаврилова, П.П. Пермякова, А.А. Ивановой под редакцией Г.П. Гаврилова. Логический подход к искусственному интеллекту. От модальной логики к логике баз данных. Москва. «Мир». — 1998.
14. [Электронный ресурс] *Kaggle*. Дата последнего обращения: 06.02.2022 г. URL: https://www.kaggle.com/
15. [Электронный ресурс] *Teachable machine*. Дата последнего обращения: 06.02.2022 г. URL: https://teachablemachine.withgoogle.com/
16. [Электронный ресурс] *Google Colab*. Дата последнего обращения: 06.02.2022 г. URL: https://colab.research.google.com/
17. [Электронный ресурс] Хабр. Молчание вентиляторов. Google Colab, Javascript и TensorflowJS. Дата последнего

обращения: 08.02.2022 г. URL: https://habr.com/ru/company/avito/blog/488936/
18. [Электронный ресурс] Код. Журнал Яндекс Практикума. Дата последнего обращения: 09.02.2022 г. URL: https://thecode.media/jupyter/
19. [Электронный ресурс] Хабр. Представляем PyCaret: открытую low-code библиотеку машинного обучения на Python. Дата последнего обращения: 09.02.2022 г. URL: https://habr.com/ru/company/otus/blog/497770/
20. [Электронный ресурс] Loginom. Предобработка данных (Data Preprocessing). Дата последнего обращения: 11.02.2022 г. URL: https://wiki.loginom.ru/articles/data-preprocessing.html
21. [Электронный ресурс] Хабр. Руководство по распознаванию эмоций на изображении с использованием Python/ Дата последнего обращения: 09.02.2022 г. URL: https://habr.com/ru/post/650041/
22. [Электронный ресурс] *Kaggle*. *Fire dataset*. Дата последнего обращения: 09.02.2022 г. URL: https://www.kaggle.com/phylake1337/fire-dataset/code
23. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html
24. [Электронный ресурс] Хабр. Сверточная нейронная сеть, часть 1: структура, топология, функции активации и обучающее множество. Дата последнего обращения: 14.02.2022 г. URL: https://habr.com/ru/post/348000/
25. [Электронный ресурс] Университет ИТМО. Обработка естественного языка. Дата последнего обращения: 14.02.2022 г. URL: https://neerc.ifmo.ru/wiki/
26. [Электронный ресурс] *Kaggle*. *Fake & real news*. Дата последнего обращения: 09.02.2022 г. URL: https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset/code

27. [Электронный ресурс] Справочная служба *Microsoft*. Дата последнего обращения: 16.02.2022 г. URL: https://support.microsoft.com/ru-ru/office
28. [Электронный ресурс] Журнал *PromoPult*. Словарь терминов. Дата последнего обращения: 18.02.2022 г. URL: https://promopult.ru/libraryЛемматизация
29. [Электронный ресурс] *CoderLessions*. Дата последнего обращения: 18.02.2022 г. URL: https://coderlessons.com/tutorials/mashinnoe-obuchenie/uchebnik-nltk/5-stemming-i-lemmatizatsiia
30. [Электронный ресурс] Университет ИТМО. Рекуррентные нейронные сети. Дата последнего обращения: 18.02.2022 г. URL: https://neerc.ifmo.ru/wiki/
31. [Электронный ресурс] *CoderLessions*. Дата последнего обращения: 19.02.2022 г. URL: https://coderlessons.com/tutorials/python-technologies/uznaite-mashinnoe-obuchenie-s-python/mashinnoe-obuchenie-logisticheskaia-regressiia
32. [Электронный ресурс] Хабр. *LSTM* – сети долгой краткосрочной памяти. Дата последнего обращения: 19.02.2022 г. URL: https://habr.com/ru/company/wunderfund/blog/331310/