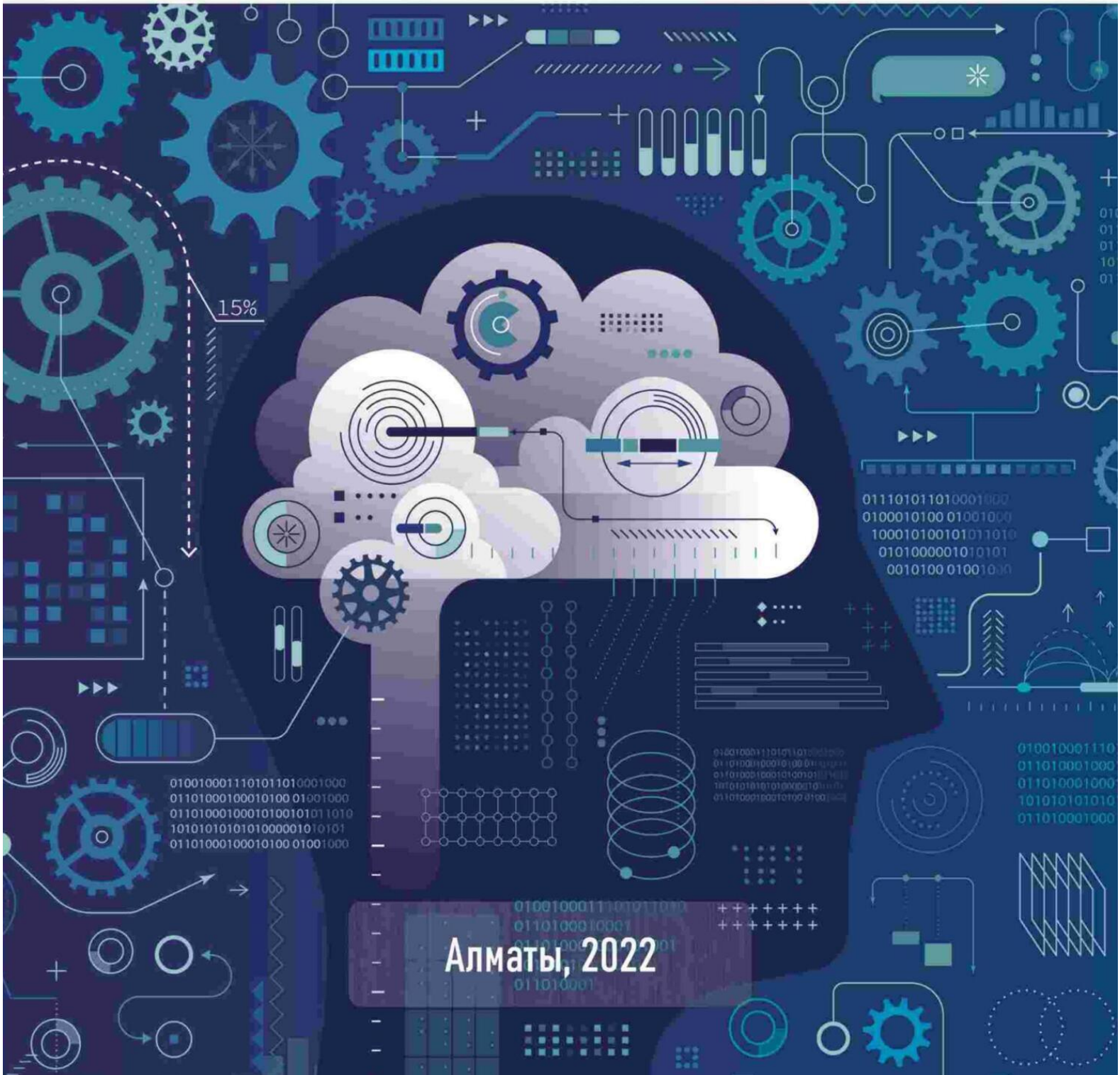
Р.И. Мухамедиев, Е.Н. Амиргалиев

ВВЕДЕНИЕ В МАШИННОЕ ОБУЧЕНИЕ



Алматы, 2022

Равиль Мухамедиев

Введение в машинное обучение

«Автор»
2023

Мухамедиев Р. И.

Введение в машинное обучение / Р. И. Мухамедиев — «Автор», 2023

Учебник поможет студентам различных специальностей освоить современные технологии машинного обучения и практически использовать их в работе и научных проектах. В настоящем пособии даются весьма краткие теоретические и относительно подробные практические сведения о применении отдельных алгоритмов классификации и регрессии. Для практического освоения материала достаточно базовых навыков работы с языком Python. При этом освоение возможностей основных библиотек, таких как matplotlib, numpy, pandas, sklearn происходит в процессе решения задач. Используя полученные знания и навыки, студенты смогут решать широкий круг задач классификации, регрессии, анализировать влияние отдельных признаков на работу классификаторов и регрессионных моделей, снижать размерность данных, визуализировать результаты и оценивать качество моделей машинного обучения. Издание рекомендовано УМО РУМС.

Содержание

Едилхан Амиргалиев, Равиль Мухамедиев
Введение в машинное обучение

Министерство образования и науки Республики Казахстан

Рекомендован к изданию УМО РУМС

Рецензенты:
Барахнин В. Б., д.т.н., профессор, НГУ, Россия, Новосибирск
Никульчев Е. В., д.т.н., профессор, МИРЭА, Россия, Москва
Маткаримов Б. Т., , д.т.н., профессор, Назарбаев университет, Казахстан, Нурсултан

Эту книгу мы посвящаем памяти наших родителей Валерии, Ильгиза, Несипхана и Задан, которые дали нам жизнь, любовь и возможность стать теми, кто мы есть.
Равиль Ильгизович Мухамедиев,
Едилхан Несипханович Амиргалиев

Предисловие

Авторы по роду своей педагогической деятельности и в рамках своих научных исследований довольно часто соприкасались с проблемами искусственного интеллекта, распознавания образов, в том числе машинного обучения. Естественно, нам приходилось поглубже изучать эти проблемы, чтобы успешно оперировать методами машинного обучения, которые становятся «модными» инструментами, применяемыми многими специалистами для решения оптимизационных задач в «плохо формализованных» областях исследования.

Машинное обучение наиболее быстро развивающая часть науки о данных. Новые, успешные модели появляются ежегодно, а их модификации и примеры приложений практически ежедневно. Поэтому написание какого-нибудь подробного пособия сопряжено с трудностями выбора такого содержания, которое с одной стороны не будет слишком поверхностным изложением самых последних достижений, а с другой стороны не погрузиться в разбор моделей, которые на практике используются уже не столь часто. Авторы постарались следовать от простого к сложному, уделив особое внимание аппарату искусственных нейронных сетей, поскольку последние достижения в области искусственного интеллекта связаны с моделями глубоких нейронных сетей. Вместе с тем разбор особенностей различных моделей хоть и полезен, но не достаточен для практических применений. Поэтому авторы включили специальные разделы, описывающие оценку качества моделей, предобработку данных и оценку параметров, которые в той или иной мере присутствуют в каждом приложении машинного обучения. На наш взгляд имеющийся набор сведений достаточен для успешного старта на пути применения машинного обучения на практике. Учебник обогащен материалами на основе опыта преподавания авторами предметов по искусственному интеллекту, распознаванию образов и классификации, компьютерному зрению, обработке естественного языка и машинному обучению в ведущих вузах Республики Казахстан: КазНУ имени аль-Фараби, КазНИТУ имени К. Сатпаева, Международном университете информационных технологий, Университете имени Сулеймана Демиреля, Казахстанско-Британском Техническом университете, а также материалами и результатами, полученными в рамках научных исследований по выполненным научным проектам грантового и программно-целевого финансирования в течение последних 10 лет.

Любая книга – это большой и часто длительный труд, который не мог бы состояться без помощи многих людей. Авторы выражают искреннюю признательность рецензентам В. Б. Барахнину, Е. В. Никульчеву и Б. Т. Маткаримову, потратившим драгоценное время для улучшения качества текста и давшим ценные советы по содержанию учебника. Большую работу по коррекции текста и тестированию примеров книги проделали Адилхан Сымагулов, Марина Елис, Ян Кучин. Ян Кучин оказал большую помощь в рамках проекта по созданию классификатора литологических типов урановых скважин РК. Рустам Мусабаев представил результаты эксперимента, посвященные высокопроизводительным вычислениям.

Не претендуя на полноту изложения всех возможностей методов машинного обучения, авторы в данной книге необходимый материал подали в двух частях, как введение в теорию и введение в практику машинного обучения. Практическая часть насыщена лабораторными занятиями и разборами кодов примеров, что поможет читателям поглубже освоить методы применения машинного обучения. Архив с примерами программ читатели могут скопировать по адресу: https://www.dropbox.com/s/xtxicveo5lwmu8z/ML_book_ExamplesLabs_v.1.0.zip?dl=0.

Авторы надеются, что учебник окажет помощь преподавателям вузов, магистрантам, докторантам и многим разработчикам, занимающимся прикладными задачами.

За все ошибки и неточности, которые заинтересованный читатель увидит в тексте, несут ответственность только авторы. Мы будем признательны за конструктивные заме-

чания и уточнения, которые читатели могут направить на адреса электронной почты: mukhamediev.ravil@gmail.com, amir_ed@mail.ru. Дополнительный материал авторы планируют размещать на сайте geoml.info.

Введение

Машинное обучение (Machine Learning – ML) – направление науки, относящееся к большой области, называемой искусственным интеллектом. Это направление исследований развивается уже несколько десятков лет. Оно обеспечивает потребности практики в тех ситуациях, когда строгая математическая модель задачи отсутствует или является неприемлемо сложной. В рамках этого направления рассматривают алгоритмы, которые способны обучаться, то есть, по существу, находить закономерности в данных. ML как научное направление изучает методы кластеризации, классификации и регрессии. В результате для специалистов по разработке программного обеспечения предлагаются методы обработки данных, которые реализуют часть интеллектуальных способностей, присущих человеку. К их числу относится способность обучаться, переобучаться, классифицировать объекты реального мира, предсказывать на основе накопленного опыта. В настоящее время именно с ML связано наибольшее количество ожиданий по реализации «умных» программ и сервисов (Smart Services). Например, по оценкам Gartner в 2017 году (рисунок 1.1), ML порождает наибольшие ожидания в развитии технологий. Более того, значительная часть новых технологий связана с ML.

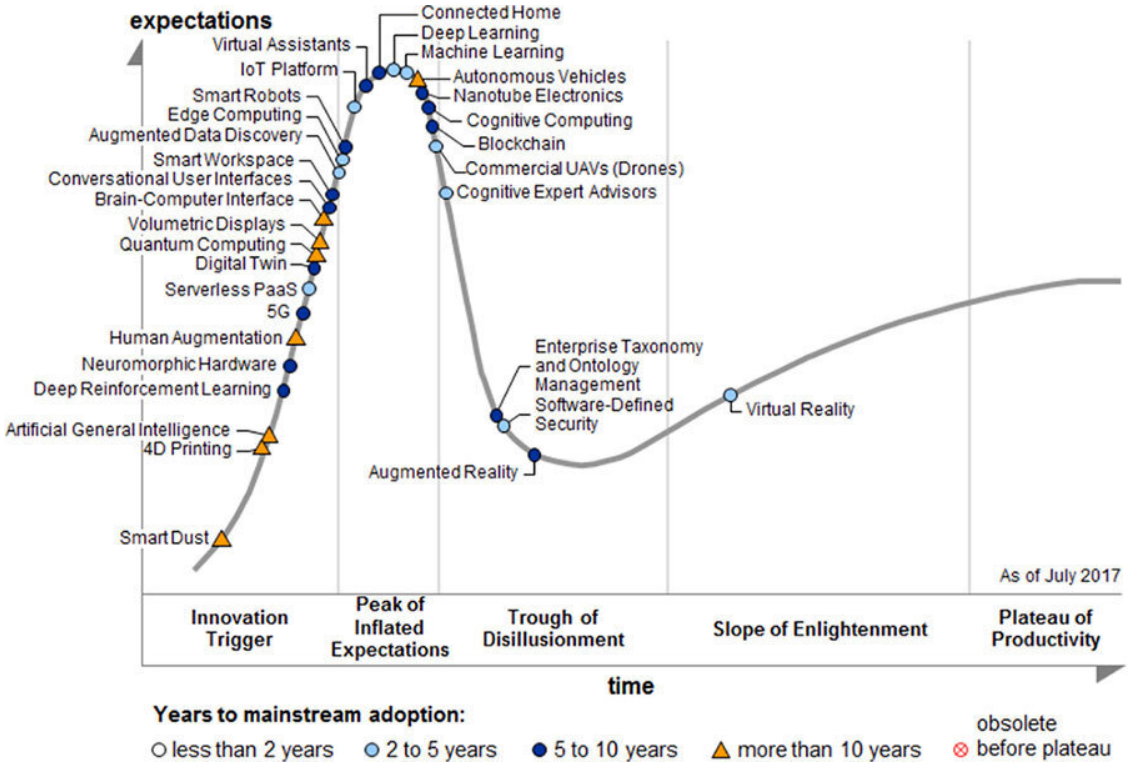


Рисунок 1.1. Инновационные триггеры, ожидания, разочарования и продуктивность технологий [1]

Организациям и исследователям, занимающимся разработкой наукоемких технологий, рекомендуется рассматривать следующие научные области: Smart Dust, Machine Learning, Virtual Personal Assistants, Cognitive Expert Advisors, Smart Data Discovery, Smart Workspace, Conversational User Interfaces, Smart Robots, Commercial UAVs (Drones), Autonomous Vehicles,

[1] http://www.gartner.com/newsroom/id/3412017

Natural-Language Question Answering, Personal Analytics, Enterprise Taxonomy and Ontology Management, Data Broker PaaS (dbrPaaS) и Context Brokering.

Таким образом, ML из сферы научных исследований перешло в сферу инженерных дисциплин. Знание ML необходимо системным аналитикам, инженерам программного обеспечения, разработчикам встроенных систем, программистам. Общие понятия о ML должны быть также у специалистов по управлению.

В настоящее время существует несколько программных систем и библиотек программ, реализующих алгоритмы машинного обучения с той или иной степенью гибкости. Например, система RapidMiner [2], один из лучших интегрированных пакетов, обеспечивает подготовку данных, создание моделей и тем самым интеграцию их в бизнес-процессы организации. Matlab, широко известный пакет прикладных программ и язык программирования компании MathWorks, предоставляет несколько сотен функций для анализа данных – от дифференциальных уравнений и линейной алгебры до математической статистики и рядов Фурье. GNU Octave использует совместимый с Matlab язык высокого уровня и в целом имеет высокую совместимость с Matlab. Это позволяет использовать и его для прототипирования систем машинного обучения. Функции Octave доступны онлайн [3], загрузить систему можно по ссылке [4]. Отметим, что Octave содержит несколько предустановленных библиотек, список которых можно просмотреть по ссылке https://octave.sourceforge.io/packages.php.

Однако наиболее часто упоминается язык программирования Python и ряд библиотек, использующих его для реализации алгоритмов машинного обучения. Например, развитые библиотеки программ по машинному обучению могут быть вызваны из среды Anaconda (https://www.anaconda.com/), основой которой является язык Python. Библиотеки numpy, matplotlib, pandas, sklearn, предустановленные в Anaconda, используются в данном пособии в качестве практической основы для решения задач классификации и регрессионного анализа.

Настоящая книга состоит из двух основных частей.

В первой части, которую можно назвать «теоретической», мы рассматриваем модели машинного обучения, основные метрики оценки качества работы алгоритмов ML, задачи и методы подготовки данных и т.п. В ней приводятся примеры и необходимые пояснения обсуждаемых моделей. Материал этой части может составить основу лекционного курса по машинному обучению. Эта часть состоит из семи глав.

В первой главе машинное обучение рассматривается в контексте дисциплин искусственного интеллекта (ИИ). Несложная классификация дисциплин ИИ дает понимание места и роли ML в задачах обработки данных.

Во второй главе обсуждаются математические модели классических алгоритмов машинного обучения. В эту группу мы, разумеется, включили не все возможные алгоритмы, однако представленные алгоритмы дают представление о разнообразии классических моделей ML.

В третьей главе мы достаточно подробно обсуждаем методы оценки качества классификации и регрессии.

Четвертая глава посвящена методам и средствам предобработки табличных данных.

Пятая глава кратко описывает специфические задачи обработки больших объемов данных.

Шестая глава содержит введение в модели глубокого обучения.

Седьмая глава посвящена еще до конца не решенному вопросу объяснения результатов работы моделей ML.

---

[2] https://rapidminer.com/

[3] Octave online. – https://octave-online.net/ (2017-04-01).

[4] Octave download. – https://www.gnu.org/software/octave/download.html (2017-04-01).

Вторая часть включает методические рекомендации по порядку выполнения лабораторных работ, достаточно объемный практикум машинного обучения и описание проектной работы. Каждая лабораторная работа содержит необходимые пояснения и одну или несколько задач. Выполнение этих задач позволит учащимся получить хорошие навыки в использовании библиотек машинного обучения и решении практических задач. Дополнительная глава описывает практическую задачу по интерпретации данных электрического каротажа скважин по добыче урана и ставит несколько задач по обработке этих данных. Материалы этой главы можно использовать для выполнения проекта по применению машинного обучения в задачах добычи полезных ископаемых.

Любая книга не свободна от недостатков. Как говаривал незабвенный Козьма Прутков, «нельзя объять необъятное». Множество интересных вопросов машинного обучения остались за рамками книги. Однако авторы надеются, что представленный материал покроет некоторый дефицит в систематическом, практико ориентированном изложении сведений о классических методах машинного обучения, а лабораторные работы позволят студентам овладеть практическими навыками, необходимыми для решения задач машинного обучения на базовом уровне.

Часть I. Математические модели и прикладные методы машинного обучения

1. Искусственный интеллект и машинное обучение. Составные части искусственного интеллекта

Искусственный интеллект (ИИ) – это любые программно-аппаратные методы, которые имитируют поведение и мышление человека. ИИ включает машинное обучение, обработку естественного языка (Natural Language Processing – NLP), синтез текста и речи, компьютерное зрение, робототехнику, планирование и экспертные системы [5]. Схематично компоненты ИИ показаны на рисунке 1.1.

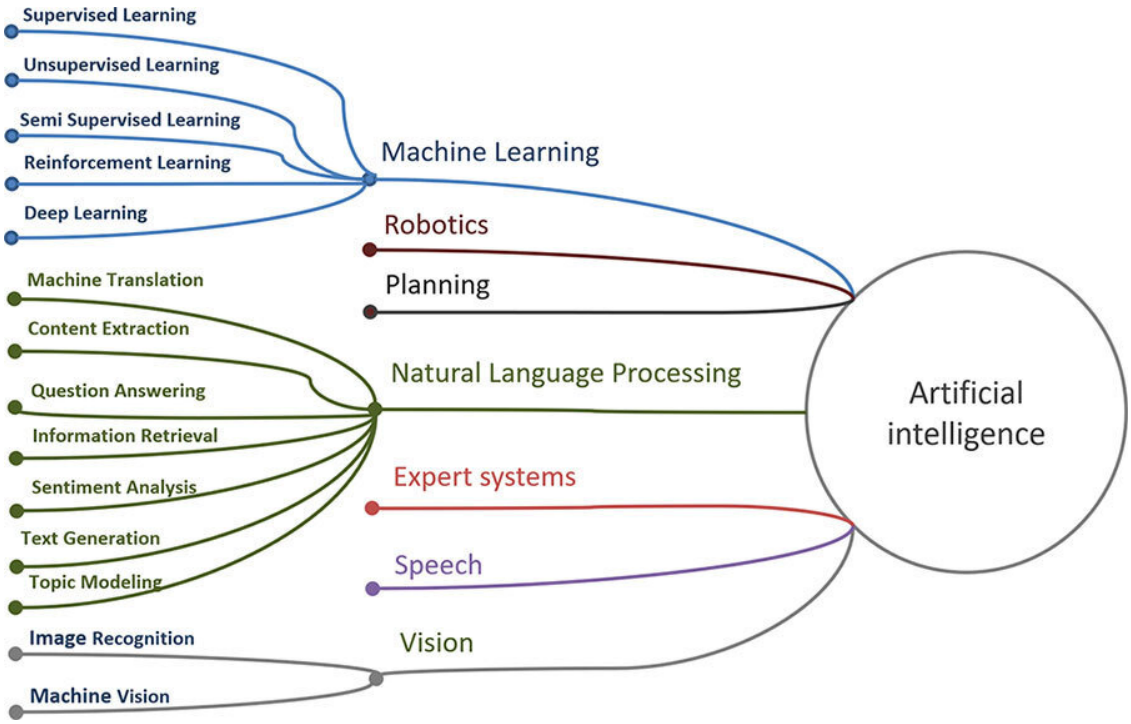


Рисунок 1.1. Подразделы искусственного интеллекта

Машинное обучение как дисциплина, являющаяся частью обширного направления, именуемого «искусственный интеллект», реализует потенциал, заложенный в идее ИИ. Основное ожидание, связанное с ML, заключается в реализации гибких, адаптивных, «обучаемых» алгоритмов или методов вычислений.

> *Примечание. «Метод вычислений» – термин, введенный Д. Кнутом для отделения строго обоснованных алгоритмов от эмпирических методов, обоснованность которых часто подтверждается практикой.*

[5] The Artificial Intelligence (AI) White Paper. – https://www.iata.org/contentassets/b90753e0f52e48a58b28c51df023c6fb/ai-white-paper.pdf (2021-02-23).

В результате обеспечиваются новые функции систем и программ. Согласно определениям, приведенным в [6]:

– Машинное обучение (ML) – это подмножество методов искусственного интеллекта, которое позволяет компьютерным системам учиться на предыдущем опыте (то есть на наблюдениях за данными) и улучшать свое поведение для выполнения определенной задачи. Методы ML включают методы опорных векторов (SVM), деревья решений, байесовское обучение, кластеризацию k-средних, изучение правил ассоциации, регрессию, нейронные сети и многое другое.

– Нейронные сети (NN) или искусственные NN являются подмножеством методов ML, имеющим некоторую косвенную связь с биологическими нейронными сетями. Они обычно описываются как совокупность связанных единиц, называемых искусственными нейронами, организованными слоями.

– Глубокое обучение (Deep Learning -DL) – это подмножество NN, которое обеспечивает расчеты для многослойной NN. Типичными архитектурами DL являются глубокие нейронные сети, сверточные нейронные сети (CNN), рекуррентные нейронные сети (RNN), порождающие состязательные сети (GAN), и многое другое.

Перечисленные компоненты ИИ показаны на рисунке 1.2.

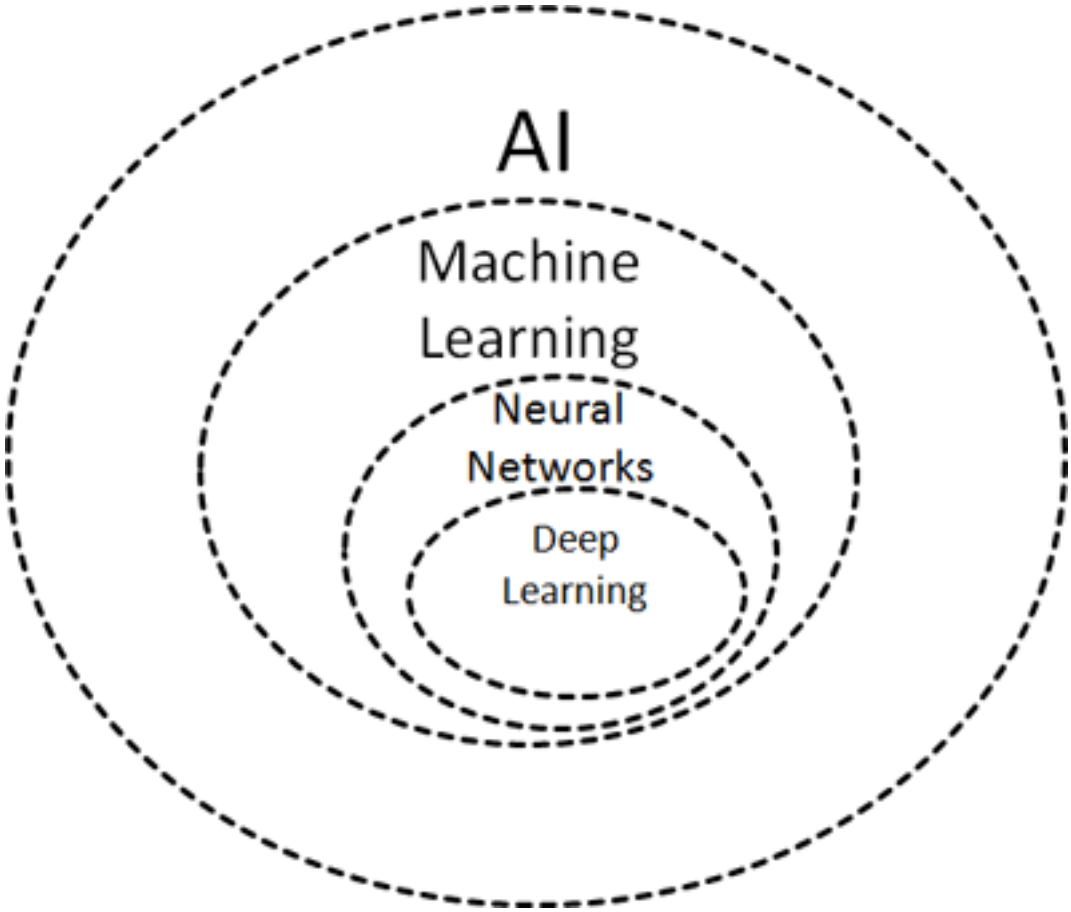


Рисунок 1.2. Искусственный интеллект и машинное обучение

[6] Nguyen G. et al. Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: A survey // Artificial Intelligence Review. – 2019. – Т. 52. – № 1. – С. 77–124.

Сегодня машинное обучение успешно применяется для решения задач в медицине [7, 8], биологии [9], робототехнике, городском хозяйстве [10] и промышленности [11, 12], сельском хозяйстве [13], моделировании экологических [14] и геоэкологических процессов [15], при создании системы связи нового типа [16], в астрономии [17], петрографических исследованиях [18, 19], геологоразведке [20], обработке естественного языка [21, 22] и т.д.

1.1. Машинное обучение в задачах обработки данных

Массивы накопленных или вновь поступающих данных обрабатываются для решения задач регрессии, классификации или кластеризации.

В первом случае задача исследователя или разработанной программы - используя накопленные данные, предсказать показатели изучаемой системы в будущем или восполнить пробелы в данных.

Во втором случае, используя размеченные наборы данных, необходимо разработать программу, которая сможет самостоятельно размечать новые, ранее не размеченные наборы данных.

В третьем случае исследователь имеет множество объектов, принадлежность которых к классам, как и сами классы, не определена. Необходимо разработать систему, позволяющую определить число и признаки классов на основании признаков объектов.

Таким образом, задача обработки данных называется регрессией, когда по некоторому объему исходных данных, описывающих, например, предысторию развития процесса, необходимо определить его будущее состояние в пространстве или времени или предсказать его

---

[7] Joseph A. Cruz and David S. Wishart. Applications of Machine Learning in Cancer Prediction and Prognosis // Cancer Informatics. – 2006. – Vol. 2. – P. 59–77.

[8] Miotto R. et al. Deep learning for healthcare: Review, opportunities and challenges // Briefings in Bioinformatics. – 2017. – Т. 19. – № 6. – С. 1236–1246.

[9] Ballester, Pedro J. and John BO Mitchell. A machine learning approach to predicting protein–ligand binding affinity with applications to molecular docking // Bioinformatics. – 2010. – Vol. 26. – № 9. – P. 1169–1175.

[10] Mahdavinejad, Mohammad Saeid, Mohammadreza Rezvan, Mohammadamin Barekatain, Peyman Adibi, Payam Barnaghi, and Amit P. Sheth. Machine learning for Internet of Things data analysis: A survey // Digital Communications and Networks. – 2018. – Vol. 4. – Issue 3. – P. 161–175.

[11] Farrar, Charles R. and Keith Worden. Structural health monitoring: A machine learning perspective. – John Wiley & Sons, 2012. – 66 p.

[12] Lai J. et al. Prediction of soil deformation in tunnelling using artificial neural networks // Computational Intelligence and Neuroscience. – 2016. – Т. 2016. – С. 33.

[13] Liakos, Konstantinos et al. Machine learning in agriculture: A review // Sensors. – 2018. – 18(8). – P. 2674.

[14] Friedrich Recknagel. Application of Machine Learning to Ecological Modelling // Ecological Modelling. – 2001. – Vol. 146. – P. 303–310.

[15] Татаринов В. Н., Маневич А. И., Лосев И. В. Системный подход к геодинамическому районированию на основе искусственных нейронных сетей // Горные науки и технологии. – 2018. – № 3. – С. 14–25.

[16] Clancy, Charles, Joe Hecker, Erich Stuntebeck, and Tim O'Shea. Applications of machine learning to cognitive radio networks // Wireless Communications, IEEE. – 2007. – Vol. 14. – Issue 4. – P. 47–52.

[17] Ball, Nicholas M. and Robert J. Brunner. Data mining and machine learning in astronomy // Journal of Modern Physics D. – 2010. – Vol. 19. – № 7. – P. 1049–1106.

[18] R. Muhamediyev, E. Amirgaliev, S. Iskakov, Y. Kuchin, E. Muhamedyeva. Integration of Results of Recognition Algorithms at the Uranium Deposits // Journal of ACIII. – 2014. – Vol. 18. – № 3. – P. 347–352.

[19] Амиргалиев Е. Н., Искаков С. Х., Кучин Я. В., Мухамедиев Р. И. Методы машинного обучения в задачах распознавания пород на урановых месторождениях // Известия НАН РК. – 2013. – № 3. – С. 82–88.

[20] Chen Y., Wu W. Application of one-class support vector machine to quickly identify multivariate anomalies from geochemical exploration data // Geochemistry: Exploration, Environment, Analysis. – 2017. – Т. 17. – № 3. – С. 231–238.

[21] Hirschberg J., Manning C. D. Advances in natural language processing // Science. – 2015. – Т. 349. – № 6245. – С. 261–266.

[22] Goldberg Y. A primer on neural network models for natural language processing // Journal of Artificial Intelligence Research. – 2016. – Т. 57. – С. 345–420.

состояние при ранее не встречавшемся сочетании параметров; классификацией, когда определенный объект нужно отнести к одному из ранее определенных классов, и кластеризацией, когда объекты разделяются на заранее не определенные группы (кластеры).

В случаях, когда нет строгих формальных методов для решения задач регрессии, классификации и кластеризации, используются методы ML [23].

В настоящее время методы ML делят на пять классов [24, 25, 26, 27, 28]: обучение без учителя (Unsupervised Learning – UL) [29] или кластерный анализ, обучение с учителем (Supervised Learning – SL) [30], полууправляемое обучение, включая самообучение (Semi-supervised Learning – SSL), обучение с подкреплением (Reinforcement Learning – RL) и глубокое обучение (Deep Learning). Методы машинного обучения решают задачи регрессии, классификации, кластеризации и снижения размерности данных (рисунок 1.3).

Задачи кластеризации и снижения размерности решают с использованием методов UL, когда множество заранее не обозначенных объектов разбивается на группы путем автоматической процедуры, исходя из свойств этих объектов [31, 32]. Указанные методы позволяют выявлять скрытые закономерности в данных, аномалии и дисбалансы. Однако в конечном счете настройка этих алгоритмов все же требует экспертного оценивания.

---

[23] Под методом машинного обучения мы будем понимать реализацию алгоритма или некоторой модели вычислений, которая решает задачу классификации, регрессии или кластеризации с использованием «обучающихся» алгоритмов.

[24] Taiwo Oladipupo Ayodele. Types of Machine Learning Algorithms // New Advances in Machine Learning. – 2010. – P. 19–48.

[25] Hamza Awad Hamza Ibrahim et al. Taxonomy of Machine Learning Algorithms to classify realtime Interactive applications // International Journal of Computer Networks and Wireless Communications. – 2012. – Vol. 2. – № 1. – P. 69–73.

[26] Muhamedyev R. Machine learning methods: An overview // CMNT. – 2015. – 19(6). – P. 14–29.

[27] Goodfellow I. et al. Deep learning. – Cambridge: MIT press, 2016. – Т. 1. – № 2.

[28] Nassif A. B. et al. Speech recognition using deep neural networks: A systematic review // IEEE Access. – 2019. – Т. 7. – С. 19143–19165.

[29] Hastie T., Tibshirani R., Friedman J. Unsupervised learning. – New York: Springer, 2009. – P. 485–585.

[30] Kotsiantis, Sotiris B., I. Zaharakis, and P. Pintelas. Supervised machine learning: A review of classification techniques // Emerging Artificial Intelligence Applications in Computer Engineering. – IOS Press, 2007. – P. 3–24.

[31] Jain A. K., Murty M. N., Flynn P. J. Data clustering: A review // ACM computing surveys (CSUR). – 1999. – Т. 31. – № 3. – С. 264–323.

[32] Wesam Ashour Barbakh, Ying Wu, Colin Fyfe. Review of Clustering Algorithms. Non-Standard Parameter Adaptation for Exploratory Data Analysis // Studies in Computational Intelligence. – 2009. – Vol. 249. – P. 7–28.

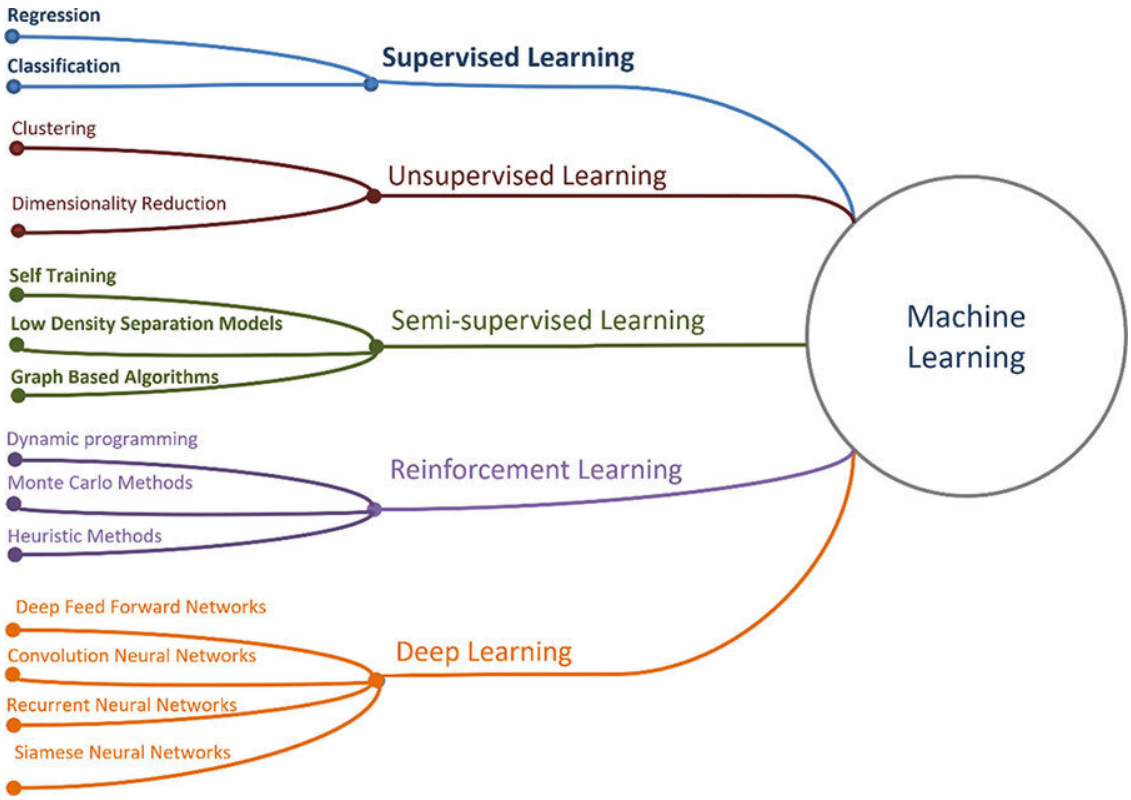
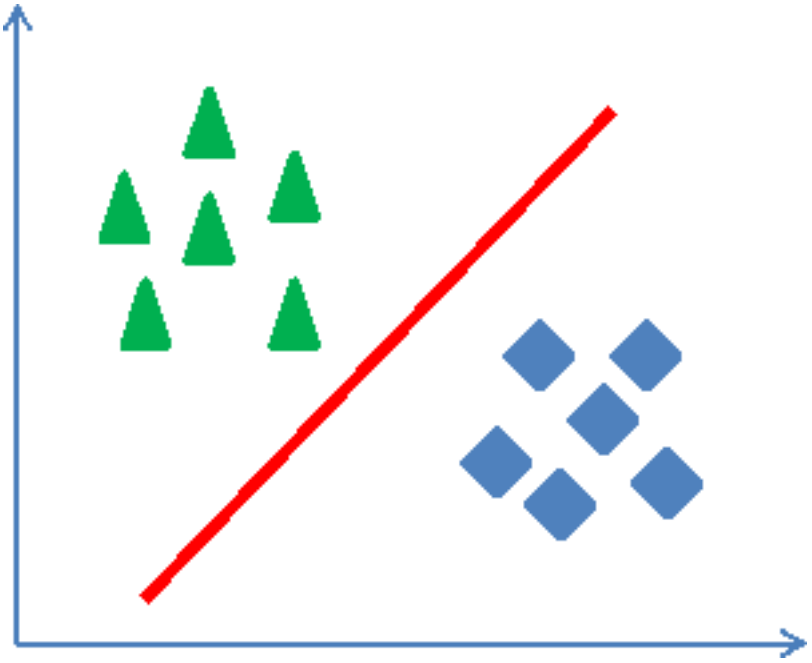


Рисунок 1.3. Основные классы методов машинного обучения [33]

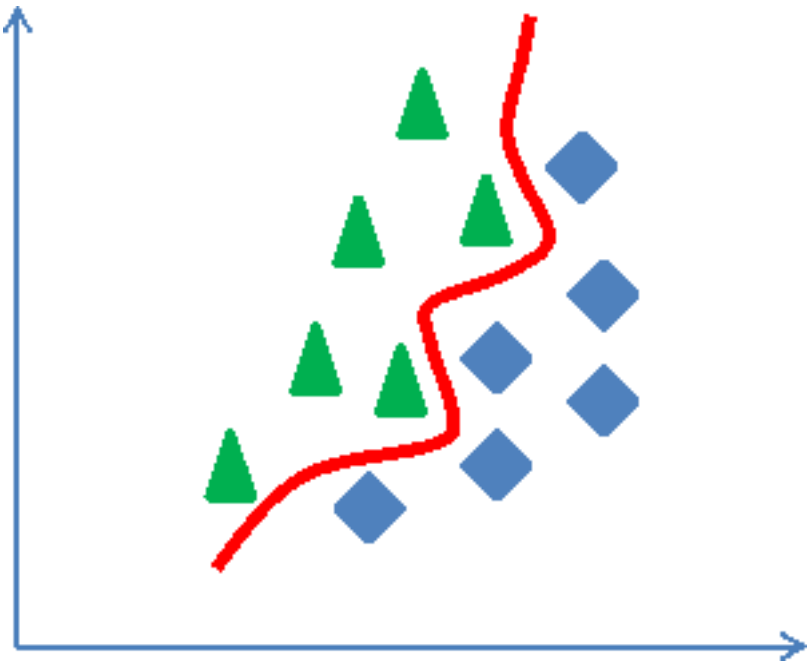
Методы SL решают задачу классификации или регрессии. Задача классификации возникает тогда, когда в потенциально бесконечном множестве объектов выделяются конечные группы некоторым образом обозначенных объектов. Обычно формирование групп выполняется экспертом. Алгоритм классификации, используя эту первоначальную классификацию как образец, должен отнести следующие не обозначенные объекты к той или иной группе, исходя из свойств этих объектов.

Методы SL часто разделяются на линейные и нелинейные в зависимости от формы (гиперплоскости или гиперповерхности), разделяющей классы объектов. В двумерном случае линейные классификаторы разделяют классы единственной прямой, тогда как нелинейные классификаторы – линией (рисунок 1.4).

[33] Mukhamediev R. I. et al. From Classical Machine Learning to Deep Neural Networks: A Simplified Scientometric Review // Applied Sciences. – 2021. – Т. 11. – №. 12. – С. 5541.



a)



b)

Рисунок 1.4. Линейный (a) и нелинейный (b) классификаторы

В таблице 1.1 перечислены пять классов методов машинного обучения и выделены алгоритмы, которые рассматриваются в нижеследующих разделах.

Таблица 1.1. Методы машинного обучения для анализа данных

Класс методов ML	Метод/алгоритм
UL	k-means [34] Principal Component Analysis (PCA), Isometric Mapping (ISOMAP) [35], Locally Linear Embedding (LLE) [36], t-distributed stochastic neighbor embedding (t-SNE) [37], kernel principal component analysis (KPCA) [38], multidimensional scaling (MDS) [39]
SL classic	k-Nearest Neighbor (k-NN) [40, 41, 42],

Более детальная иерархическая классификация классических методов машинного обучения приведена в приложении 2.

1.2. Программное обеспечение для решения задач машинного обучения

Библиотеки машинного обучения можно разделить на две большие группы: базовые библиотеки, реализующие широкую гамму классических алгоритмов машинного обучения, импорт и экспорт данных и их визуализацию, и библиотеки, предназначенные для создания и работы с моделями глубокого обучения. В приведенном ниже перечне выделены пакеты, которые далее используются при выполнении задач настоящего учебника.

Базовые библиотеки:

Обработка массивов и матриц – numpy

Обработка данных, включая импорт и экспорт данных – pandas, pytables

Анализ данных – scipy, scikit-learn, opencv

Визуализация данных- matplotlib, bokeh, seaborn

Многоцелевые – sympy, cython

Пакеты для работы с моделями глубокого обучения (Deep Learning frameworks):

Caffe/Caffe2, CNTK, DL4J, Keras, Lasagne, mxnet, PaddlePaddle, TensorFlow, Theano, Torch, Trax

Таблица 1.2 кратко описывает наиболее часто применяемые пакеты программ.

Таблица 1.2. Пакеты программ, применяемые для решения задач машинного обучения

Библиотека	Назначение	Примечание
numpy	Высокоэффективные матричные операции.	Предустановлен в Anaconda.
pandas	Импорт-экспорт из файлов разного формата, включая таблицы Excel и базы данных, агрегация данных в виде data frame, визуализация данных.	Предустановлен в Anaconda.
scikit-learn	Решение задач машинного обучения: классификация, регрессия, кластеризация, снижение размерности, настройка моделей машинного обучения.	Предустановлен в Anaconda.
matplotlib	Визуализация двумерных изображений и графиков.	Предустановлен в Anaconda.
TensorFlow (https://www.tensorflow.org/)	Высокоэффективные тензорные вычисления, в том числе с применением графических процессоров.	Требуется установка в простейшем случае pip install tensorflow.
Keras (https://keras.io/)	Высокоуровневый программный интерфейс для реализации моделей нейронных сетей, работающий как надстройка над TensorFlow, CNTK или Theano.	Требуется установка pip install keras. В настоящее время является частью текущей версии TensorFlow.
Trax	Альтернатива связке Keras-TensorFlow.	Текущая версия, вероятно, не работает в среде Windows. Пакет находится в стадии разработки.

1.3. Схема настройки системы машинного обучения

Применение методов машинного обучения в задачах, для которых строгая математическая модель отсутствует, а имеются только экспертные оценки, часто бывает оптимальным способом решения. Обучаемая система, в частности, искусственная нейронная сеть, способна воспроизвести закономерность, которую сложно или невозможно формализовать. В задачах «обучения с учителем» часто затруднительно определить качество экспертных оценок. К таким задачам относятся, в частности, и задачи выявления рисков заболеваний, оценки качества продуктов, распознавания речи, предсказания уровня котировок акций на финансовых рынках, распознавания литологических типов на урановых месторождениях по данным электрического каротажа. Несмотря на то, что эксперты задают перечень актуальных признаков объектов, диапазоны измеряемых физических величин могут перекрываться, а экспертные оценки могут быть противоречивыми или содержать ошибки. В качестве такого примера на рисунке 1.5 показаны точки, соответствующие породам (по экспертным оценкам), или, иначе говоря, литологическим типам (песок, гравий, глина и т.п.), в пространстве трех видов электрического каротажа (кратко обозначены ИК, ПС, КС) для одного из урановых месторождений Казахстана.

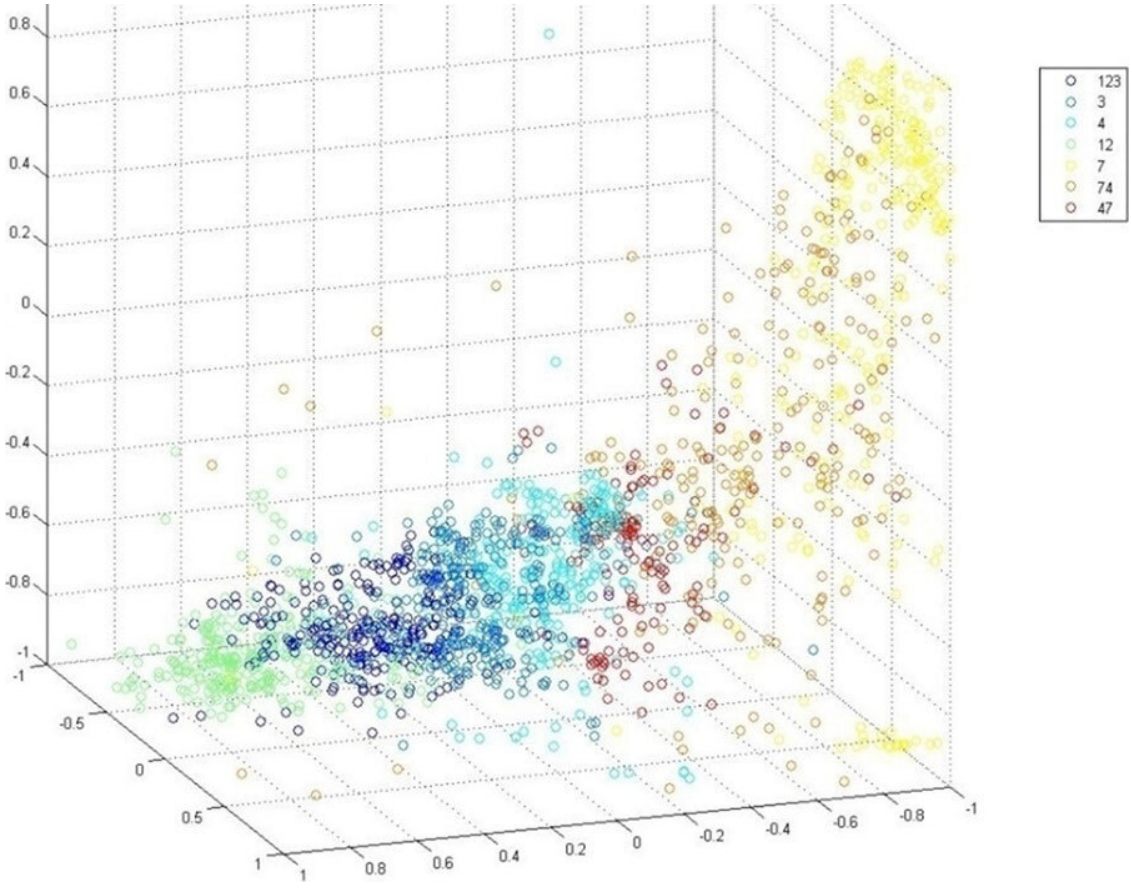


Рисунок 1.5. Ответы экспертов в трехмерном (ИК, КС и ПС) пространстве признаков

Примечание. Подробнее о задаче классификации литологических типов на урановых месторождениях с применением методов машинного обучения рассказывается в монографии [34].

Номера пород, приведенных на рисунке и обозначенных разными цветами, описываются в главе «Проект по созданию классификатора литологических типов на основании каротажных данных урановых скважин РК».

Видно, что точки, соответствующие разным литологическим типам, существенно перемешаны в пространстве признаков и, соответственно, не могут быть разделены простыми (например, линейными) способами.

Кроме этого, данные, представленные для классификации, могут содержать аномальные значения и ошибки, связанные с физическими особенностями процессов их получения. Соответственно, и обученная система может интерпретировать данные с ошибками.

В процессе разработки комплекса программ обработки данных инженер по данным выполняет анализ применимости методов машинного обучения, определяет способы подготовки данных для использования указанных методов, выполняет сравнение алгоритмов с целью выявления лучшего алгоритма, решающего задачу.

Общая схема настройки методов машинного обучения на решаемую задачу приведена на рисунке 1.6.

В соответствии с этой схемой нам необходимо определить саму задачу, которая должна быть решена с помощью машинного обучения. Затем собрать данные, предобработать их,

[34] Мухамедиев Р. И. Методы машинного обучения в задачах геофизических исследований. – Рига, 2016. – 200 с. – ISBN 978-9934-14-876-7.

выбрать алгоритмы или методы, обучить или настроить методы, оценить результаты. В задачах обучения с учителем данные должны быть разделены на тренировочную (train), тестовую (test) и для некоторых задач проверочную (validation) части. Перечисленные этапы на самом деле части итеративного процесса, который инженер по данным повторяет с целью добиться наилучшего результата работы. Этот процесс не обязательно приводит к наилучшему результату, но его цель – добиться лучшего из возможных при тех данных, которые имеются в распоряжении исследователя.

Рисунок 1.6. Циклический процесс настройки модели машинного обучения для решения задачи

1.4. Контрольные вопросы

1. Искусственный интеллект – это часть обширного направления, называемого «искусственные нейронные сети»?

2. Глубокое обучение как направление исследований и разработок – часть машинного обучения?

3. Чем отличаются алгоритмы «обучения с учителем» от кластеризации?

4. Что такое линейный классификатор и чем он отличается от нелинейного?

5. Процесс настройки модели машинного обучения – это _____?

6. Укажите типы машинного обучения, относящиеся к классу «обучение с учителем» (Supervised Learning).

7. Какие библиотеки машинного обучения используются в данном пособии?

8. Укажите типы машинного обучения, относящиеся к классу «обучение без учителя» (Unsupervised Learning).

9. Вы получили заданный набор обучающих данных. Что делать, если результаты работы алгоритма машинного обучения не удовлетворяют потребностям практики?

2. Классические алгоритмы машинного обучения

2.1. Формальное описание задач машинного обучения

Формальная постановка задачи машинного обучения (задача обучения по примерам или задача обучения с учителем) заключается в следующем [35].

Пусть имеются два пространства: Ob (пространство допустимых объектов), Y (пространство ответов или меток) и (целевая) функция.

Определено отображение y: Ob → Y, которое задано лишь на конечном множестве объектов (обучающей выборке (прецедентах) (sample set)) размером m:

$$y^{(1)}(ob_1), y^{(2)}(ob_2), \ldots, y^{(m)}(ob_m),$$

то есть известны метки объектов $ob_1, ob_2, \ldots, ob_m$. Требуется построить алгоритм A («обучить»), который по объекту ob определяет значение y(ob) или «достаточно близкое» значение, если допускается неточное решение.

Другими словами, зная значения целевой функции на обучающей выборке, требуется найти удовлетворительное приближение к ней в виде A.

При конечном множестве Y = {1, 2,…, l} задачу называют задачей классификации (на l непересекающихся классов). В этом случае можно считать, что множество X разбито на классы $C_1, \ldots, C_l$, где Ci = {ob Ob | y(ob) = i} при i{1, 2,…, l}:

$$Ob = \bigcup^{1}_{i=1} C_i.$$

При Y = {(α1,…,αl) |α1,…,αl {0,1 говорят о задаче классификации на l пересекающихся классов. Здесь i-й класс – Ci = {ob Ob | y(ob) = (α1,…,αl), αi = 1}.

Для решения задачи, то есть поиска оптимального алгоритма A, вводится функция потерь или функция стоимости (cost function) J(A(ob), y(ob)), которая описывает, насколько «плох» ответ A(ob) по сравнению с верным ответом y(ob). В задаче классификации можно считать, что

$$J(A(ob), y(ob)) = \begin{cases} \gg 0, A\ (ob) \neq y(ob) \\ 0, A(ob) = y(ob) \end{cases},$$

а в задаче регрессии

$$J(A(ob), y(ob)) = | A(ob) - y(ob) |$$

или

$$J(A(ob), y(ob)) = (A(ob) - y(ob))2.$$

Возникает закономерный вопрос: что же такое объект? В задачах машинного обучения объект – это некоторое множество параметров (признаков). Если некоторую сущность можно

---

[35] Дьяконов А. Г. Анализ данных, обучение по прецедентам, логические игры, системы WEKA, RapidMiner и MatLab (Практикум на ЭВМ кафедры математических методов прогнозирования): учебное пособие. – М.: Изд. отдел факультета ВМК МГУ им. М. В. Ломоносова, 2010.

описать конечным набором параметров, то она может рассматриваться как объект в машинном обучении, причем ее физическая природа не имеет значения. Параметры могут задаваться исследователем, исходя из его представлений о наилучшем описании объекта, так, как это делается в «классических» задачах машинного обучения, или, с другой стороны, формироваться путем выполнения некоторой процедуры так, как это делается в глубоком обучении.

Таким образом, каждый объект ob описывается конечным набором (входных) параметров или свойств (input values or features) $x_1,x_2,\ldots x_n$, одинаковым для каждого $ob_i \in Ob$, а y называется целевой переменной (целевым параметром) (target value) в задаче регрессии или классом в задаче классификации.

Алгоритм A может описываться конечным набором параметров $\theta_i \in \theta$ или, как часто говорится при описании нейронных сетей, весов (weights) $w_i \in W$.

Задача обучения по примерам рассматривается как задача оптимизации, которую решают путем настройки множества параметров θ алгоритма A так, чтобы минимизировать значение функции стоимости $J(\theta)$ по всем примерам m.

В задаче регрессии алгоритм A часто называется функцией гипотезы, а функция стоимости определяется как сумма квадратов разности «предсказываемого» алгоритмом (функцией гипотезы) значения и реального значения y по множеству примеров m. При этом подбирается такая функция гипотезы $h_\theta(x)$, которая при некотором наборе параметров $\theta_i \in \theta$ обеспечивает минимальное значение $J(\theta)$.

$$J(\theta) = min \frac{1}{2m} \sum_{i=1}^{m} (h_\theta (x^{(i)}) - y^{(i)})^2, \qquad \text{(Eq. 2.1)}$$

где m – множество обучающих примеров или объектов; $x^{(i)}$ – значение параметров или свойств для i-го объекта; $y^{(i)}$ – фактическое значение объясняемой или целевой переменной для i-го примера; h_θ – функция гипотезы, которая может быть линейной ($h_\theta = \theta_0 + \theta_1 x$) или нелинейной (например, квадратичная функция гипотезы одной переменной – ($h_\theta = \theta_0 + \theta_1 x + \theta_2 x^2$).

Например, если мы рассматриваем задачу прогнозирования стоимости автомобиля, исходя из года его производства, то год производства будет являться входной переменной или свойством (x), а стоимость – целевой переменной (y) (рисунок 2.1).

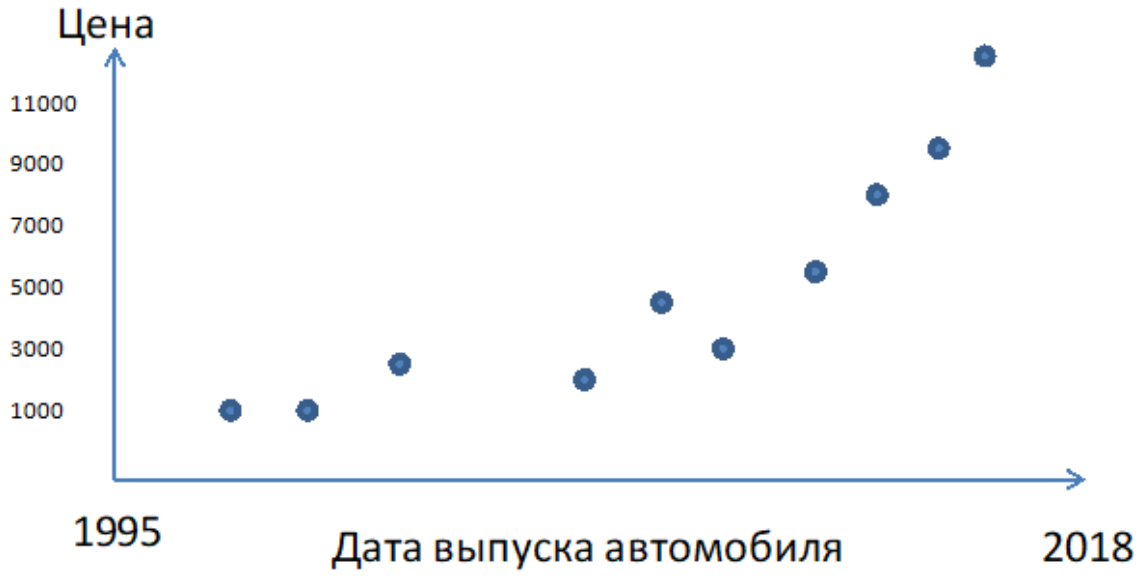


Рисунок 2.1. Зависимость стоимости автомобиля от года выпуска

В таком случае мы решаем задачу регрессии одной переменной. Случай регрессии многих переменных возникает тогда, когда мы будем учитывать кроме года выпуска объем двигателя, количество посадочных мест, марку и т.п. Перечисленные параметры образуют множество свойств или входных параметров, которые определяют единственную целевую переменную – стоимость.

Забегая вперед, можно сказать, что для подбора параметров θ_i необходимо, чтобы параметры $x_j \in X$ (в многомерном случае), описывающие объекты, были выражены единицами одинаковой размерности и примерно одинаковой величины. Чаще всего путем нормализации стремятся представить все параметры в виде чисел в диапазоне $0 \le x \le 1$ или $-1 \le x \le 1$. Вообще говоря, выбор функции нормализации зависит от класса задачи. Кроме того, в процессе предварительной обработки данных могут быть использованы методы, обеспечивающие исключение аномальных значений, исключение шумов (например, высокочастотных) путем сглаживания и т.п. Выбор этих методов также зависит от класса задачи. После того как параметры нормализованы и очищены от аномальных значений, а также исключены объекты, которые определены не полностью (то есть объекты, для которых часть свойств неизвестна), выполняется поиск функции гипотезы $h_\theta(x)$, которая минимизирует стоимость $J(\theta)$.

2.2. Линейная регрессия одной переменной

Задача линейной регрессии формулируется как поиск минимальной функции стоимости (см. формулу 2.1) при условии, что функция гипотезы является линейной $h_\theta = \theta_0 + \theta_1 x$. Очевидно, что подобная функция соответствует линии в двумерном пространстве (рисунок 3.1а). Для нахождения оптимальной функции $h_\theta(x)$ применяется алгоритм градиентного спуска (gradient descent), суть которого заключается в последовательном изменении параметров θ_0, θ_1 с использованием выражения:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1), \quad \text{(Eq. 2.2)}$$

где α – параметр обучения; а $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ является производной функции стоимости по θ_j. Знак := означает присваивание, в отличие от знака равенства (=), применяемого в алгебраических выражениях.

При этом шаги алгоритма выполняются так, что вначале происходит одновременное изменение обоих параметров на основании выражения 2.2 и только затем использование их для расчета новых значений функции стоимости. Другими словами, алгоритмическая последовательность одного из шагов цикла для случая двух параметров, выраженная на псевдокоде, будет следующей:

$$temp\theta_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \qquad \text{(Eq. 2.3)}$$

$$temp\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \qquad \text{(Eq. 2.4)}$$

$$\theta_0 := temp\theta_0;$$

$$\theta_1 := temp\theta_1.$$

Отметим, что выражение функции гипотезы можно преобразовать следующим образом:

$$h_\theta = \theta_0 + \theta_1 x = \theta_0 * 1 + \theta_1 x_1 = \theta_0 x_0 + \theta_1 x_1$$

и записать в виде:

$$h_\theta = \theta_0 x_0 + \theta_1 x_1$$

с учетом того, что $x_0 = 1$. Последнее выражение позволяет вычислять функцию гипотезы путем матричного умножения матрицы X, первая колонка которой всегда состоит из единиц, на вектор θ.

С учетом дифференцирования выражения 1.3 и 1.4 можно переписать в виде:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)} - y^{(i)})\, x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)} - y^{(i)})\, x_1^{(i)}.$$

В зависимости от параметра обучения α алгоритм может достигать минимума (сходиться) или же при слишком большом α не сходиться.

Наиболее простой в реализации, но не оптимальный по времени выполнения пакетный алгоритм градиентного спуска (Batch Gradient Descent) использует все обучающие примеры на каждом шаге алгоритма. Вместо алгоритма градиентного спуска для нахождения параметров θ_j можно использовать матричное выражение:

$$\Theta = (X^T X)^{-1} X^T y, \qquad \text{(Eq. 2.5)}$$

где θ – вектор параметров; $(X^TX)^{-1}$ – обратная матрица X^TX; X^T – транспонированная матрица X.

Преимуществом матричных операций является то, что нет необходимости подбирать параметр α и выполнять несколько итераций алгоритма. Недостаток связан с необходимостью получения обратной матрицы, сложность вычисления которой пропорциональна $O(n^3)$, а также с невозможностью получения обратной матрицы в некоторых случаях.

Рассмотрим пример.

Решим гипотетическую задачу нахождения параметров линейной регрессии методом градиентного спуска. Во-первых, подключим необходимые библиотеки:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time
```

Отметим, что библиотека time позволит нам рассчитать время выполнения программы. Ее применение будет понятно из нижеследующего кода. Сформируем обучающее множество, состоящее из 30 примеров:

```
xr=np.matrix(np.linspace(0,10,30))
x=xr.T
#значения функции зададим в виде следующего выражения
y=np.power(x,2)+1
#Построим график (рисунок) командами
plt.figure(figsize=(9,9))
plt.plot(x,y,'.')
```

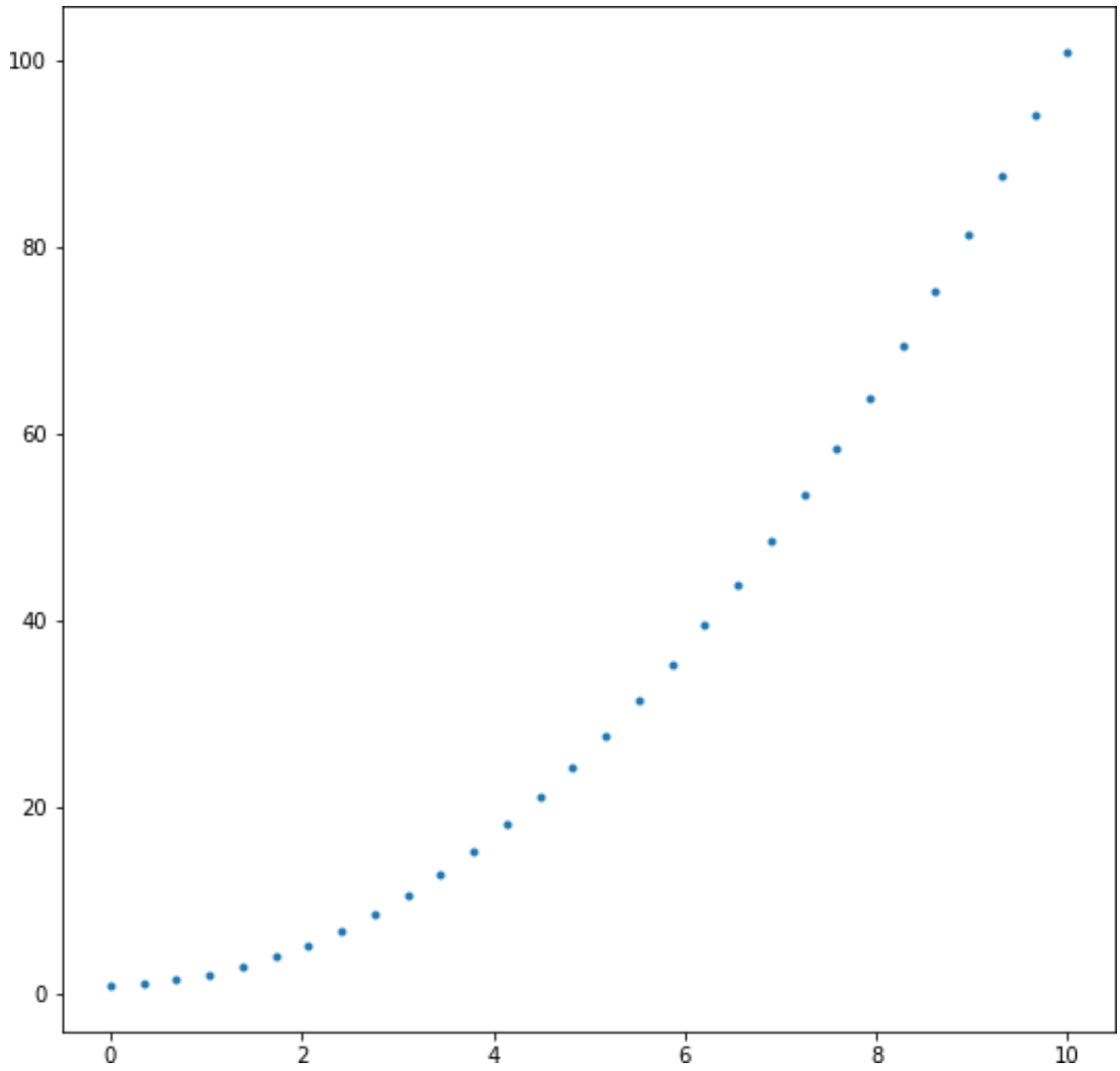


Рисунок 2.2. График функции $y=x_2+1$

В нашем случае мы задали фиксированное множество примеров (m = 30), однако в дальнейшем мы можем его изменить. Для того чтобы программа воспринимала любое множество примеров, определим его, используя метод size:

```
m=x.size
#сформируем первую колонку матрицы X, состоящую из единиц
on=np.ones([m,1])
#и сформируем матрицу X, объединив колонки
X=np.concatenate((on,x),axis=1)
```

Это матрица, в первой колонке которой стоят единицы, а во второй – значения $x^{(1)}$, $x^{(2)}$, …, $x^{(m)}$. Затем зададим абсолютно произвольно начальные значения коэффициентов регрессии:

```
theta=np.matrix('0.1;1.3')
#и рассчитаем значения функции гипотезы
h=np.dot(X,theta)
#дополним предыдущий график регрессионной прямой
```

```
plt.plot(x,h)
```

Получим график вида:

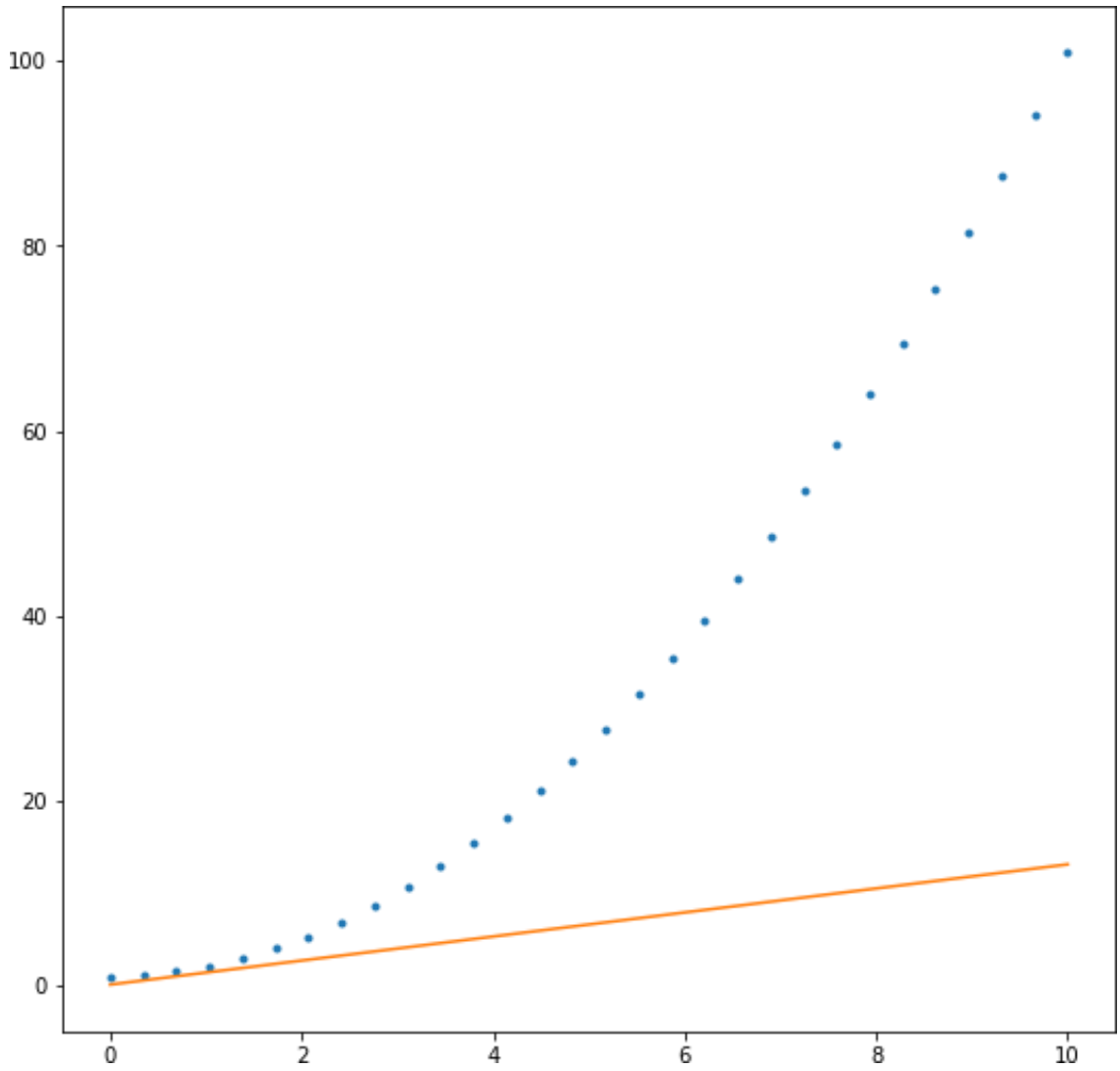


Рисунок 2.3. Начальное положение прямой регрессии

На графике видно, что прямая функция гипотезы далека от идеальной. Применим алгоритм градиентного спуска для нахождения оптимальных значений параметров регрессионной прямой (функции гипотезы):

```
t0=time.time()
alpha=0.05
iterations=500
for i in range(iterations):
theta=theta-alpha*(1/m)*np.sum(np.multiply((h-y),x))
h=np.dot(X,theta)
t1=time.time()
#Построим графики
plt.figure(figsize=(9,9))
plt.plot(x,y,'.')
```

```
plt.plot(x,h,label='regressionByIteration')
leg=plt.legend(loc='upper right',shadow=True,fontsize='x-small')
leg.get_frame().set_facecolor('#0055DD')
leg.get_frame().set_facecolor('#eeeeee')
leg.get_frame().set_alpha(0.5)
plt.show()
```

Получим следующий график регрессионной прямой:

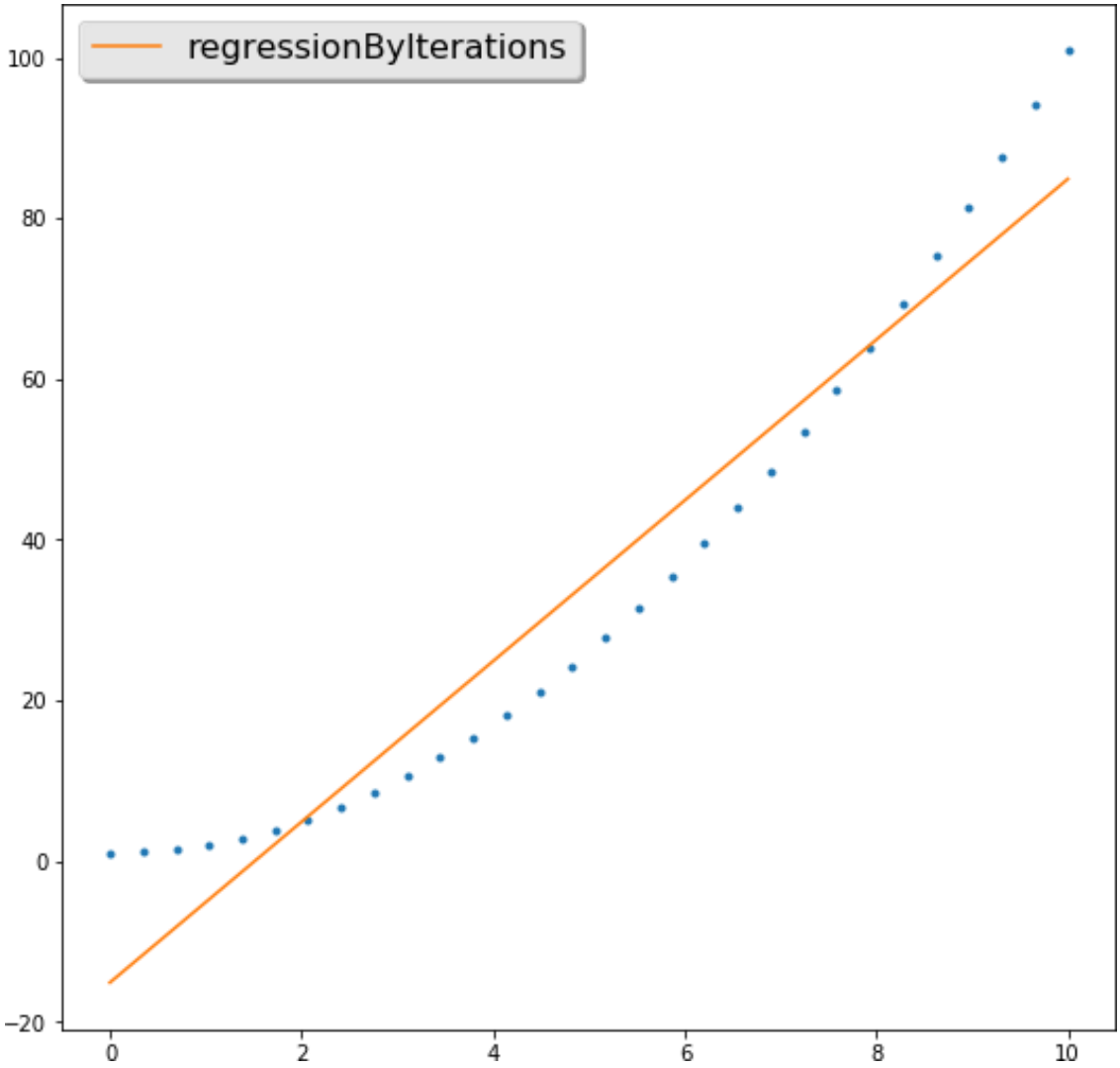


Рисунок 2.4. Результат выполнения алгоритма градиентного спуска

```
#рассчитаем среднеквадратическую ошибку
mse=np.sum(np.power((h-y),2))/m
print('regressionByIteration mse= ', mse)
#и распечатаем длительность выполнения цикла градиентного спуска
print('regressionByIterations takes ',(t1-t0))
```

Получим примерно следующий вывод:
regressionByIterations mse = 63.270782365456206
regressionByIterations takes 0.027503490447998047

В качестве небольшого дополнения рассчитаем показатели точности регрессии с применением библиотеки метрик sklearn.

```
from sklearn.metrics import mean_squared_error, r2_score
y_predict = h
y_test=y
print("Mean squared error: {:.2f}".format(mean_squared_error(y_test,y_predict)))
print("r2_score: {:.2f}".format(r2_score(y_test, y_predict)))
```

Получим следующие результаты:
Mean squared error: 63.27
r2_score: 0.93

Подробнее о метриках точности классификации и регрессии см. в разделе «3. Оценка качества методов ML».

2.3. Полиномиальная регрессия

В отличие от линейной регрессии, полиномиальная регрессия оперирует нелинейной функцией гипотезы вида $h_\theta = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \ldots + \theta_n x^n$, что позволяет строить экстраполирующие кривые (гиперповерхности) сложной формы. Однако с увеличением числа параметров существенно возрастает вычислительная сложность алгоритма. Кроме этого, существует опасность «переобучения», когда форма кривой или гиперповерхности становится слишком сложной, практически полностью подстроившись под обучающее множество, но дает большую ошибку на тестовом множестве. Это свидетельствует о том, что алгоритм потерял способность к обобщению и предсказанию. В случае переобучения, когда классификатор теряет способность к обобщению, применяют регуляризацию, снижающую влияние величин высокого порядка:

$$J(\theta) = min \frac{1}{2m} [\sum_{i=1}^{m} (h_\theta (x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2]. \qquad \text{(Eq. 2.6)}$$

Увеличение коэффициента λ приводит к повышению степени обобщения алгоритма. В пределе при очень большом значении λ функция гипотезы превращается в прямую или гиперплоскость. Практически это означает, что алгоритм будет вести себя слишком линейно, что также неоптимально. Задача исследователя – подобрать коэффициент регуляризации таким образом, чтобы алгоритм был не слишком линеен и в то же время обладал достаточной способностью к обобщению.

Вычисление параметров регрессии методом градиентного спуска выполняется так же, как и ранее, за исключением небольшого дополнения в виде регуляризационного коэффициента, так что для j-го параметра на каждом шаге цикла вычисляется значение:

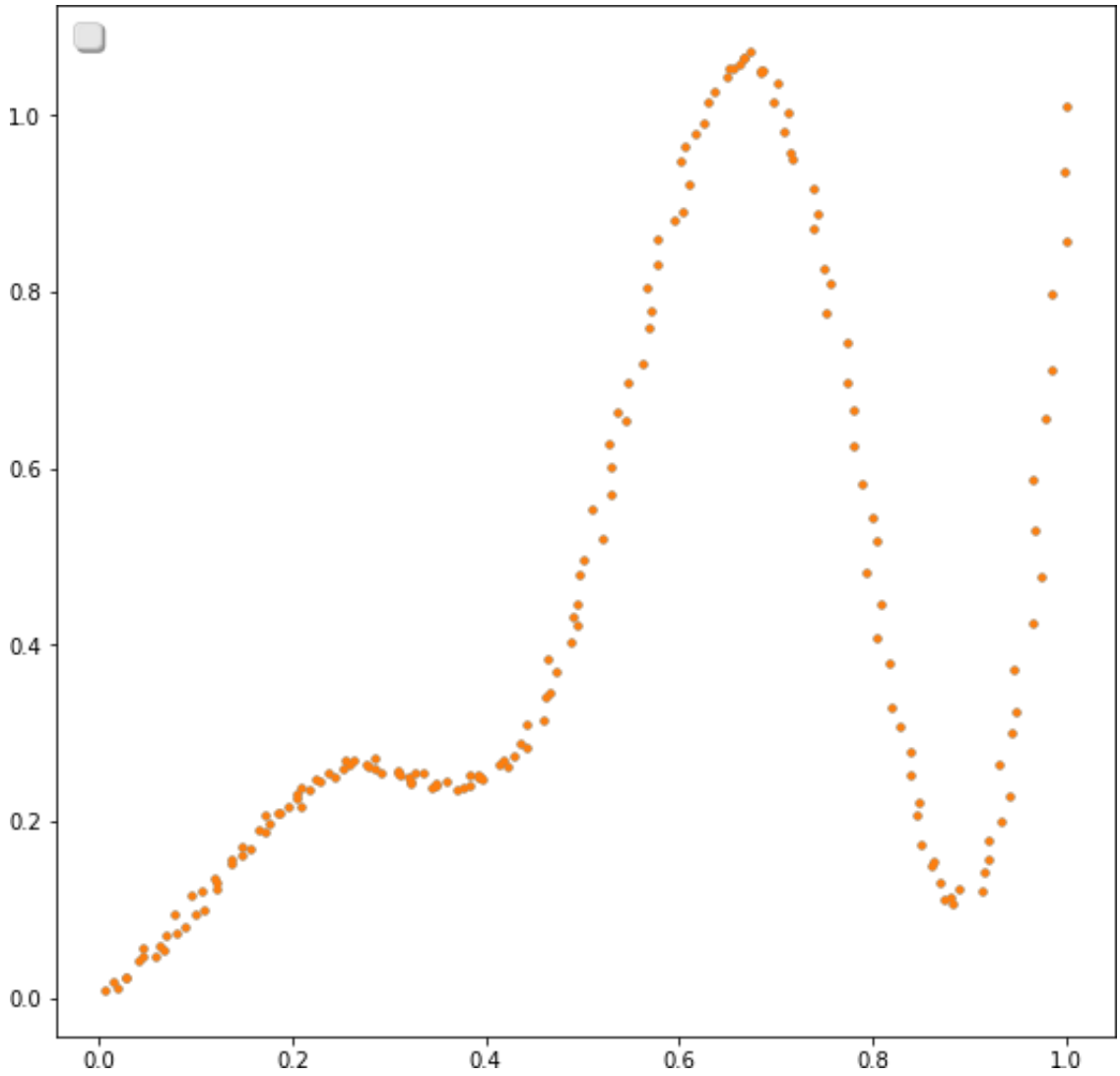
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta (x^{(i)} - y^{(i)}) \, x_j^{(i)} - \frac{\lambda}{m} \theta_j. \qquad \text{(Eq. 2.7)}$$

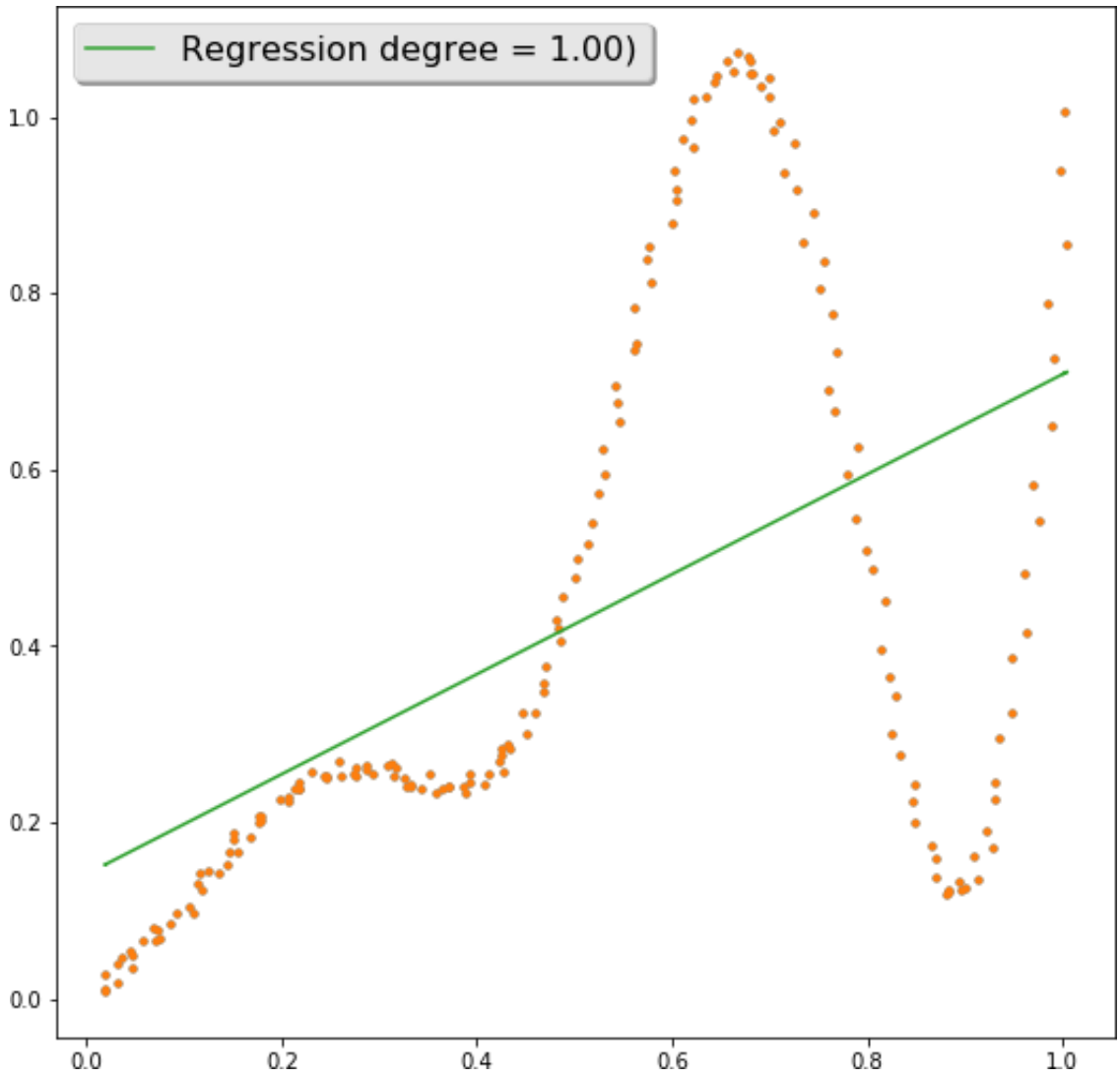
Рассмотрим пример.

В качестве исходных данных синтезируем набор данных в соответствии с выражением:

$$y = x^2 \, sin(\, 4\pi x) + x.$$

Добавив некоторый случайный коэффициент с помощью np.array([np.random.rand(x.size)]).T/50, получим примерно следующее (рисунок 2.5):




Regression degree = 1.00)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

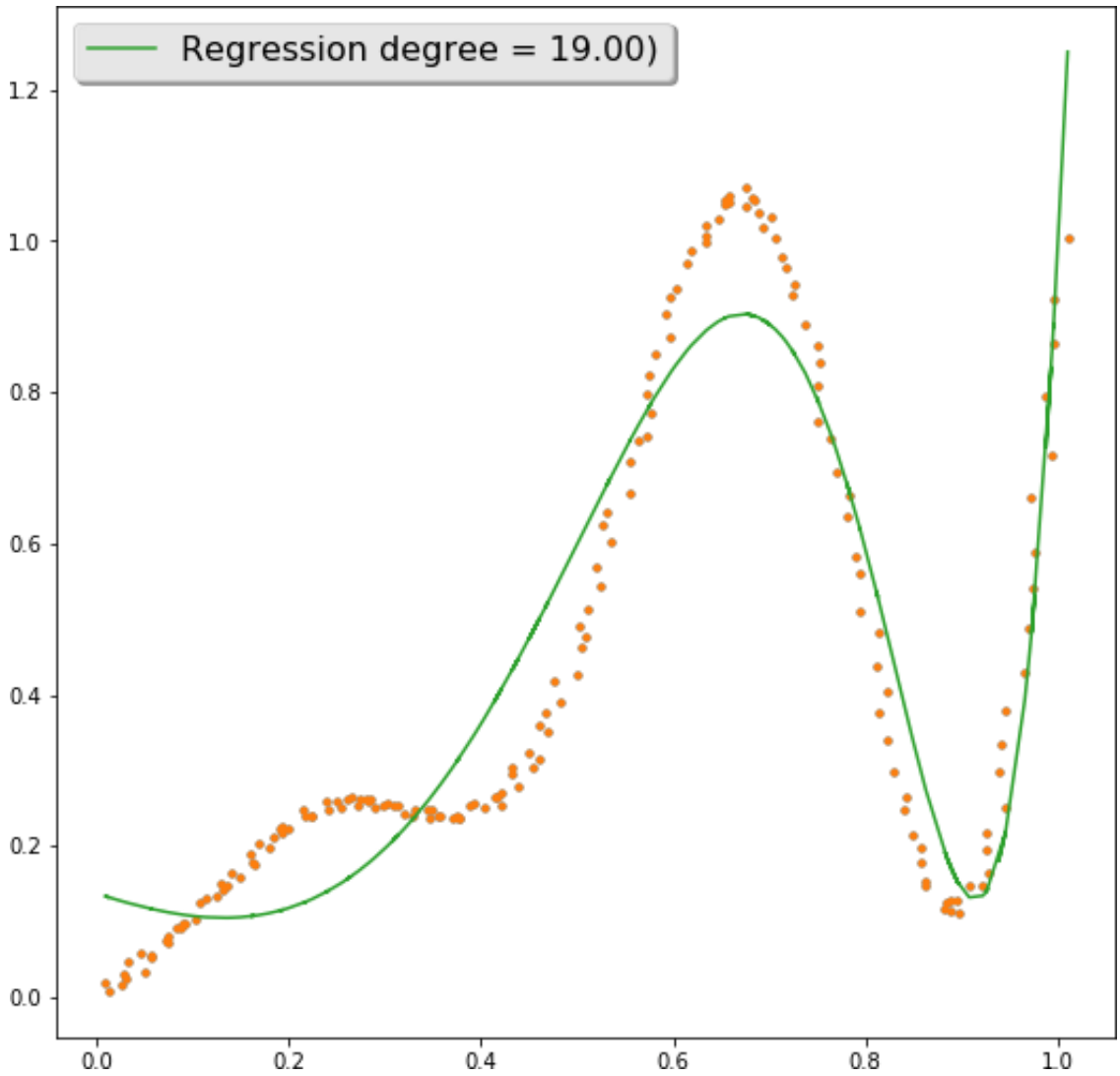


Рисунок 2.5. Исходные данные и результаты выполнения алгоритма полиномиальной регрессии

Введем переменную degree, означающую коэффициент регрессии. Например, при degree = 1 получим обычную линейную регрессию (r2_score = 0.27). Увеличивая степень регрессии, можем добится значительно лучших результатов. Например, при degree = 19 r2_score = 0.90. Использование коэффициента регуляризации lambda_reg позволяет «сглаживать» регрессионную кривую. Фрагмент программы, обеспечивающей расчет параметров полиномиальной регрессии, **приведен** ниже:

```
xr=np.array([np.linspace(0,1,180)])
x=xr.T
print(x.size)
y=f(x)
(x,y)=plusRandomValues(x,y) #добавление случайных величин
plt.figure(figsize=(9,9))
plt.plot(x,y,'.')
m=x.size
degree=19 #коэффициент регрессии
lambda_reg=0.00001
on=np.ones([m,1])
```

```
    X=on
    #расчет степеней свободной переменной в соответствии со сте-
пенью регрессии degree
    for i in range(degree):
        xx=np.power(x, i+1)
        X=np.concatenate((X,xx),axis=1)
    theta=np.array([np.random.rand(degree+1)])
    h=np.dot(X,theta.T)
    t0=time.time()
    alpha=0.5
    iterations=100000
    for i in range(iterations):
          theta=theta-alpha*(1/m)*np.dot((h-y).T,X)  -(lambda_reg/
m)*theta
        h=np.dot(X,theta.T)
    t1=time.time()
    plt.plot(x,y,'.')
    plt.plot(x,h,         label='Regression         degree         =
{:0.2f})'.format(degree))
    leg=plt.legend(loc='upper left',shadow=True,fontsize=16)
    leg.get_frame().set_facecolor('#0055DD')
    leg.get_frame().set_facecolor('#')
    leg.get_frame().set_alpha(0.9)
    plt.show()
```

2.4. Классификаторы. Логистическая регрессия

Несмотря на присутствующее в названии данного метода слово «регрессия», цель данного алгоритма не восстановление значений или предсказание. Алгоритм применяется в случае, если необходимо решить задачу классификации. В случае логистической регрессии речь идет о задаче бинарной классификации, то есть отнесении объектов к классу «негативных» или «позитивных», вследствие чего набор обучающих примеров построен таким образом, что $y \in \{0,1\}$.

В этом случае от функции гипотезы требуется выполнение условия $0 \leq h_\theta(x) \leq 1$, что достигается применением сигмоидальной (логистической) функции:

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}, \quad \text{(Eq. 2.8)}$$

Где θ – вектор параметров.

Можно записать также

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n), \quad \text{(Eq. 2.9)}$$

где n – число параметров (свойств или признаков) объектов; $g(z)$ – сигмоидальная или логистическая функция.

В сокращенном виде $h_\theta(x) = g(\theta^T x)$.

Отметим, что сигмоидальная функция широко применяется и в нейронных сетях в качестве активационной функции нейронов, поскольку является непрерывно дифференцируемой и тем самым гарантирует сходимость алгоритмов обучения нейронной сети. Примерный вид сигмоиды показан в разделе «Активационные функции».

Функция $h_\theta(x)$ может рассматриваться как вероятность того, что объект является «позитивным» ($h_\theta(x)\geq 0.5$) или «негативным» ($h_\theta(x)<0.5$). В сложных случаях, требующих нелинейной границы разделения, например, в виде окружности (рисунок 2.6), необходимо добавить дополнительные параметры, например, квадратные степени исходных параметров:

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 {x_1}^2 + \theta_4 {x_2}^2) \qquad \text{(Eq. 2.10)}$$

или их произведения и т.п.

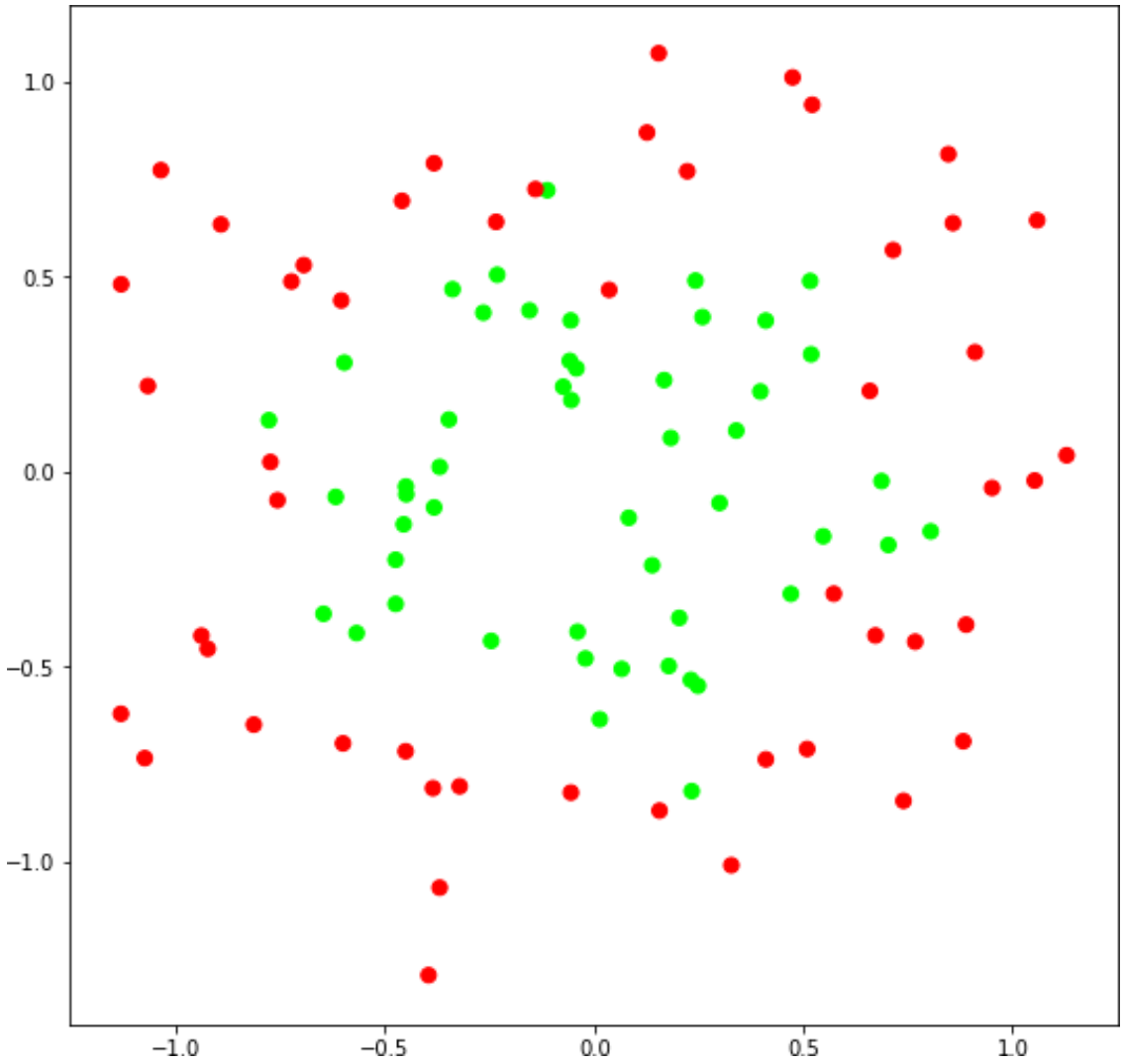


Рисунок 2.6. Объекты, для которых необходима нелинейная граница разделения

Подбор параметров θ после выбора функции гипотезы выполняется так, чтобы минимизировать функцию стоимости вида:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta \left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right). \qquad \text{(Eq. 2.11)}$$

Из двух частей функции стоимости, объединенных знаком +, вычисляется фактически только одна, так как в задаче классификации y может принимать только два значения: 1 и 0.

То есть в случае, если y = 0, стоимость для i-го примера принимает вид:

$$C = -log(1 - h_\theta(x^{(i)})),$$

а при y = 0

$$C = -log(h_\theta(x^{(i)})).$$

Таким образом, при минимальном значении функции стоимости в обоих случаях достигается максимизация вероятности принадлежности объекта к положительному классу для «положительных» объектов и минимизация вероятности для «отрицательных» объектов. Подругому логистический классификатор называется классификатором максимизации энтропии (maximum-entropy classification – MaxEnt).

Как и в случае с линейной регрессией, минимизация функции стоимости достигается с помощью алгоритма градиентного спуска (gradient descent), но также применяются Conjugate gradient [36], BFGS, L-BFGS или lbfgs [37].

Логистический классификатор может быть применен и в отношении нескольких классов. В этом случае для каждого класса классификатор настраивается отдельно. Класс, к которому принадлежит новый объект, вычисляется расчетом значений всех функций гипотез и выбором из них максимального значения $max h_\theta^{(i)}(x)$, где i – номер класса. Другими словами, объект принадлежит к тому классу, функция гипотезы которого максимальна.

Как и в случае с линейной регрессией, для увеличения обобщающей способности алгоритма применяют регуляризацию (последнее слагаемое в нижеследующей формуле), которая позволяет уменьшить влияние величин высокого порядка:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} log h_\theta\ (x^{(i)}) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}) + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \quad \text{(Eq. 2.12)}$$

Интересно, что производная функции стоимости логистической регрессии ровно такая же, как и производная функции стоимости линейной регрессии (вывод см., например, в [38]). Следовательно, алгоритм градиентного спуска будет работать так же, как и для линейной регрессии (формула 1.5), с тем отличием, что значение функции гипотезы будет вычисляться по формуле 2.8.

Пример. Построим линейный классификатор на основе логистической регрессии. Вначале сгенерируем набор данных и разделим его на тренировочное и тестовое множества:

```
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.model_selection import train_test_split
```

[36] Martin Fodslette Møller. A scaled conjugate gradient algorithm for fast supervised learning // Neural Networks. – 1993. – Vol. 6. – Issue 4. – P. 525–533.

[37] Dong C. Liu, Jorge Nocedal. On the limited memory BFGS method for large scale optimization // Mathematical Programming. – 1989. – Vol. 45. – Issue 1–3. – P. 503–528.

[38] Derivative of Cost Function for Logistic Regression. – https://medium.com/mathematics-behind-optimization-of-cost-function/derivative-of-log-loss-function-for-logistic-regression-9b832f025c2d

```
dataset = make_circles(noise=0.2, factor=0.5, random_state=1)
X_D2, y_D2 = dataset
plt.figure(figsize=(9,9))
plt.scatter(X_D2[:,0],X_D2[:,1],c=y_D2,marker='o',
    s=50,cmap=ListedColormap(['#FF0000','#00FF00']))
X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, test_size=.4,
random_state=42)
```

В результате получим распределение данных, показанное на рисунке 1.3.

Вызовем необходимые библиотеки и методы:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix,
classification_report
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
```

Последние две строки необходимы для оценки точности работы классификатора (см. раздел «Оценка качества методов ML»).

Разработаем логистическую функцию logisticFunction(X,theta) и функцию, **обеспечивающую** оценку объекта на основе предсказанного значения гипотезы, – logRegPredictMatrix(h,threshold). Как показано выше, функция гипотезы принимает значение от 0 до 1. Для того чтобы получить оценку принадлежности объекта к классу (1 – «положительный», 0 – «отрицательный»), необходимо для каждого значения гипотезы вычислить номер класса («предсказать») по правилу predicted = 0 If h <threshold и predicted = 1 If h >= threshold. Обычное значение порога threshold=0.5.

Функция, вычисляющая значения коэффициентов логистической регрессии первого порядка:

```
def logisticRegressionByNumpy(X,y):
    m=y.size
    X=np.concatenate((np.ones([m,1]),X), axis=1)
    theta=np.array(np.random.rand(X.shape[1]))
    h=logisticFunction(X,theta)
    alpha=0.05
    iterations=1500
    lambda_reg=0.01
    for i in range(iterations):
     theta=theta - alpha*(1/m) *np.dot(X.T,(h-y))-(lambda_reg/
m)*theta
        h=logisticFunction(X,theta)
    return theta,h
```

Вызов функции и вывод показателей качества можно выполнить:

```
theta,h=logisticRegressionByNumpy(X_train,y_train)
predicted_train=logRegPredictMatrix(h,threshold=0.50)
```

```
    matrix_train = confusion_matrix(y_train,
predicted_train)#,labels)
    print('Logistic regression')
    print('Results on train set')
    print('Accuracy on train set:
{:.2f}'.format(accuracy_score(y_train, predicted_train)))
    print('Conf. matrix on the train \n', matrix_train)
    print('Classification report at train set\n',
                classification_report(y_train, predicted_train,
target_names = ['not 1', '1']))
```

В результате получим на тренировочном множестве значение accuracy = 0.57, а на тестовом 0.4. Другими словами, точность предсказания нашей функции хуже, чем при случайном выборе классов! Подобный результат вполне предсказуем, поскольку мы попытались использовать прямую там, где требуется как минимум окружность.

Исправить положение можно, используя регрессию второго порядка в соответствии с выражением (2.10). В предыдущей функции достаточно изменить одного оператора:

X=np.concatenate((np.ones([m,1]),X,X**2), axis=1)

После этого мы получим значение accuracy на тренировочном и тестовом множествах, равное 0.9, что вполне приемлемо для нашей задачи.

Необходимость подбора значимых параметров и формирования новых параметров является одним из недостатков логистической регрессии.

Вторым недостатком данного метода является то, что он предназначен для решения задач бинарной классификации.

Третья проблема, вытекающая из структурных свойств графического представления логистической регрессии, заключается в том, что она не способна напрямую решать некоторые классические логические задачи.

Для преодоления этих недостатков используются искусственные нейронные сети. Однослойные нейронные сети способны решать задачу мультиклассовой классификации, а многослойные нейронные сети успешно преодолевают все три ограничения.

> *Примечание.* Программный код примера MLF_logReg_Python_numpy_002.ipynb, описанного в этом разделе, можно получить по ссылке
>
> https://www.dropbox.com/s/vlp91rtezr5cj5z/MLF_logReg_Python_numpy_002.ipynb?dl=0

2.5. Контрольные вопросы

Что такое объект в задачах машинного обучения?

Как в общем виде записать функцию стоимости в задаче классификации?

Как в общем виде записать функцию стоимости в задаче регрессии?

Приведите выражение для функции гипотезы линейной регрессии одной переменной.

Как вычислить значения коэффициентов линейной регрессии? Укажите оба способа вычисления.

Приведите выражение функции стоимости логистической регрессии. Каково будет значение функции стоимости, если y = 0, h = 0, m = 2?

Каково назначение регуляризации?

Каковы недостатки логистической регрессии?

Какие алгоритмы применяются для минимизации значения функции стоимости логистической регрессии?

Чем отличается сигмоидальная функция от логистической?
Какие значения принимает логистическая функция?

2.6. Искусственные нейронные сети

2.6.1. Вводные замечания

Искусственные нейронные сети (Artificial Neural Networks – ANN – ИНС) – аппарат, который активно исследуется начиная с 40-х годов прошлого столетия. ИНС как часть теории коннективизма прошли значительный путь от эпохи завышенных ожиданий, через период разочарований (в 70-х годах) до широко применяемой технологии в настоящее время. Связь между биологическими нейронами и возможностями их моделирования с помощью логических вычислений установлена в работе Warren S. McCulloch, Walter Pitts [39], в работе Розенблатта [40] описана модель персептрона. Недостатки однослойного персептрона отражены в книге М. Минского и С. Пейперта [41, 42]. В этой книге подробно рассмотрены ограничения однослойной нейронной сети и доказано, что она не способна решать некоторые классические логические задачи, в частности, обозначена знаменитая проблема неразрешимости функции XOR для однослойной нейронной сети. Преодолеть этот недостаток можно было путем использования многослойных нейронных сетей. Однако в конце 60-х годов было еще неясно, как обучать многослойные нейронные сети.

В 1974 году был предложен алгоритм, который впоследствии получил название «алгоритм обратного распространения» (backpropagation) [43, 44], или «алгоритм обратного распространения ошибки», пригодный для автоматического подбора весов (обучения) многослойного персептрона или многослойной нейронной сети прямого распространения. Этот алгоритм стал базой для бурного развития нейросетевых методов вычислений.

> *Примечание*. Первенство в разработке алгоритма окончательно не установлено. Считается, что он был впервые описан А. И. Галушкиным и независимо Полом Вербосом в 1974 году. Далее алгоритм развивался усилиями как отечественных ученых, так и зарубежных групп, которые, собственно, и ввели термин backpropagation в 1986 году. Метод несколько раз переоткрывался разными исследователями.

Значительный вклад в теорию коннективизма внесли советские и российские ученые [45, 46, 47, 48], доказавшие возможность решения классических вычислительных задач в нейросетевом базисе, тем самым заложив фундаментальную основу построения нейрокомпьютеров.

---

[39] Warren S. McCulloch, Walter Pitts. A logical calculus of the ideas immanent in nervous activity // The bulletin of mathematical biophysics. – 1943. – Vol. 5. – Issue 4. – P. 115–133.

[40] Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain // Psychological Review. – 1958. – Vol. 65 (6). – P. 386–408.

[41] Minsky M. L., Papert S. A. Perceptrons: An Introduction to Computational Geometry. – MIT, 1969. – 252 p.

[42] Marvin Minsky, Seymour Papert. Perceptrons, expanded edition. – The MIT Press, 1987. – 308 p.

[43] Werbos P. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. – Harvard University, 1974. – 38 p.

[44] Werbos P. J. Backpropagation: past and future // IEEE International Conference on Neural Networks. – San Diego, 1988. – Vol. 1. – P. 343–353.

[45] Нейрокомпьютеры: учеб. пособие для вузов. – М.: Изд-во МГТУ им. Н. Э. Баумана, 2004. – 320 с.

[46] Галушкин А. И. Решение задач в нейросетевом логическом базисе // Нейрокомпьютеры: разработка, применение. – М.: Радиотехника, 2006. – № 2. – С. 49–71.

[47] Ясницкий Л. Н. Введение в искусственный интеллект: учебное пособие для вузов. – М.: Академия, 2008. – 176 с.

[48] Галушкин А. И. Нейронные сети: основы теории. – Горячая линия – Телеком, 2010. – 496 с.

Примечание. Коннективизм или коннекционизм – это подход к изучению человеческого познания, который использует математические модели, известные как коннекционистские сети или искусственные нейронные сети. Часто они бывают в виде тесно связанных между собой нейронных процессоров [49].

Наиболее популярная архитектура ANN – сеть прямого распространения, в которой нелинейные элементы (нейроны) представлены последовательными слоями, а информация распространяется в одном направлении (Feed Forward Neural Networks) [50]. В 1989 году в работах G. Gybenco [51], K. Hornik [52] и др. показано, что такая сеть способна аппроксимировать функции практически любого вида. Однако в тот период теоретическая возможность была существенно ограничена вычислительными мощностями. Преодолеть этот разрыв удалось в 90-х годах, когда были предложены сети новой архитектуры, получившие впоследствие название глубоких нейронных сетей. В результате в последние годы получены впечатляющие результаты в разработке и применении новых классов сетей и так называемого глубокого обучения [53], которые состоят из множества слоев разного типа, обеспечивающих не просто классификацию, но, по существу, выявление скрытых свойств объектов, делающих такую классификацию высокоточной. Общее количество различных классов нейронных сетей превысило 27 [54]. Введение в новые архитектуры сетей приведено в разделе «Глубокое обучение».

Применение аппарата ANN направлено на решение широкого круга вычислительно сложных задач, таких как оптимизация, управление, обработка сигналов, распознавание образов, предсказание, классификация.

2.6.2. Математическое описание искусственной нейронной сети

Рассмотрим ANN с прямым распространением сигнала. В такой сети отдельный нейрон представляет собой логистический элемент, состоящий из входных элементов, сумматора, активационного элемента и единственного выхода (рисунок 2.7).

---

[49] Connectionism. Internet Encyclopedia of Philosophy. –https://iep.utm.edu/connect/#:~:text=Connectionism%20is%20an%20approach%20to,%2C%20neuron%2Dlike%20processing%20units

[50] David Saad. Introduction. On-Line Learning in Neural Networks. – Cambridge University Press, 1998. – P. 3–8.

[51] Cybenco G. Approximation by superpositions of a sigmoidal function // Mathematics of Control, Signals, and Systems. – 1989. – Vol. 4. – P. 304–314.

[52] Hornik K. et al. Multilayer feedforward networks are universal approximators // Neural Networks. – 1989. – Vol. 2. – P. 359–366.

[53] Schmidhuber, Jürgen. Deep learning in neural networks: An overview // Neural Networks. – 2015. – Vol. 61. – P. 85–117.

[54] http://www.asimovinstitute.org/neural-network-zoo/ – THE NEURAL NETWORK ZOO POSTED ON SEPTEMBER 14, 2016 BY FJODOR VAN VEEN

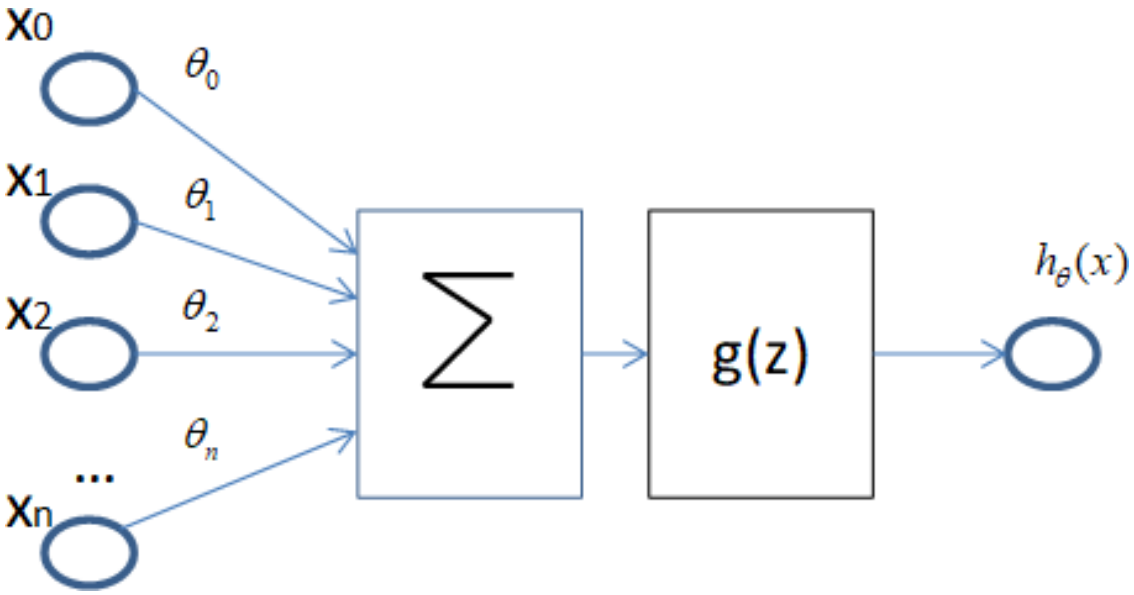


Рисунок 2.7. Схема классического нейрона

Выход нейрона определяется формулами:

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \ldots + \theta_n x_n,$$

$$h_\theta(x) = g(z),$$

где g(z) – сигмоидальная функция.

Выражение функции гипотезы классического нейрона идентично выражению функции гипотезы логистической регрессии (Eq. 2.9).

Часто в качестве активационной функции применяется сигмоидальная функция, описанная в разделе «Логистическая регрессия».

В последнее время в литературе веса θ нейронной сети чаще обозначают символом w, подчеркивая тем самым преемственность естественных нейронных сетей и искусственных нейронных сетей, где широко используется понятие синаптического коэффициента или веса (weight). Кроме того, такое обозначение показывает разницу между множеством параметров или весов (W) и гиперпараметрами модели. Гиперпараметры определяют общие свойства модели, и к ним относят коэффициент обучения, алгоритм оптимизации, число эпох обучения, количество скрытых слоев сети, количество нейронов в слоях и т.п.

Для упрощения схемы сумматор и активационный элемент объединяют, тогда многослойная сеть может выглядеть так, как показано на рисунке 1.5. Сеть содержит четыре входных нейрона, четыре нейрона в скрытом слое и один выходной нейрон.

На рисунке входные нейроны обозначены символом x, нейроны скрытого слоя – символами $a_1^{[1]}$, $a_1^{[1]}$, $a_2^{[1]}$, $a_3^{[1]}$, $a_0^{[1]}$ и выходного слоя – символом $a_1^{[2]}$. Если нейронная сеть имеет несколько слоев, то первый слой называют входным, а последний – выходным. Все слои между ними называются скрытыми. Для нейронной сети с L-слоями выход входного или нулевого слоя нейронов определяется выражением $a^{[0]} = x$.

На входе следующего или первого скрытого слоя имеем

$$z^{[1]} = w^{[1]} a^{(0)}.$$

Выход первого слоя:

$$a^{[1]} = g\left(z^{[1]}\right).$$

Для любого нейрона j, находящегося в скрытом слое i:

$$a_j^{[i]} = g\left(w_j^{[i]} a^{[i-1]} + w_{j0}^{[i]} a_0^{[i]}\right). \qquad \text{(Eq. 2.13)}$$

В этом выражении значение bias и его вес упомянуты отдельно как произведение $w_{j0}^{[i]} a_0^{[i]}$,

где $w_j^{[i]}$ – вектор весов нейрона j.

Для выходного слоя:

$$a^{[L]} = h_w(x) = g(z^{[L]}). \qquad \text{(Eq. 2.14)}$$

Например, для сети на рисунке 2.8 выход каждого нейрона скрытого слоя можно рассчитать так же, как и для одиночного нейрона:

$$a_1^{[1]} = g(w_{10}^{[1]} x_0 + w_{11}^{[1]} x_1 + w_{12}^{[1]} x_2 + w_{13}^{[1]} x_3). \qquad \text{(Eq. 2.15)}$$
$$a_2^{[1]} = g(w_{20}^{[1]} x_0 + w_{21}^{[1]} x_1 + w_{22}^{[1]} x_2 + w_{23}^{[1]} x_3).$$
$$a_3^{[1]} = g(w_{30}^{[1]} x_0 + w_{31}^{[1]} x_1 + w_{32}^{[1]} x_2 + w_{33}^{[1]} x_3).$$

Выход нейронной сети определяется выражением:

$$a^{[2]} = h_\theta(x) = g(w_{10}^{[2]} a_0^{[1]} + w_{11}^{[2]} a_1^{[1]} + w_{12}^{[2]} a_2^{[1]} + w_{13}^{[2]} a_3^{[1]}). \qquad \text{(Eq. 2.16)}$$

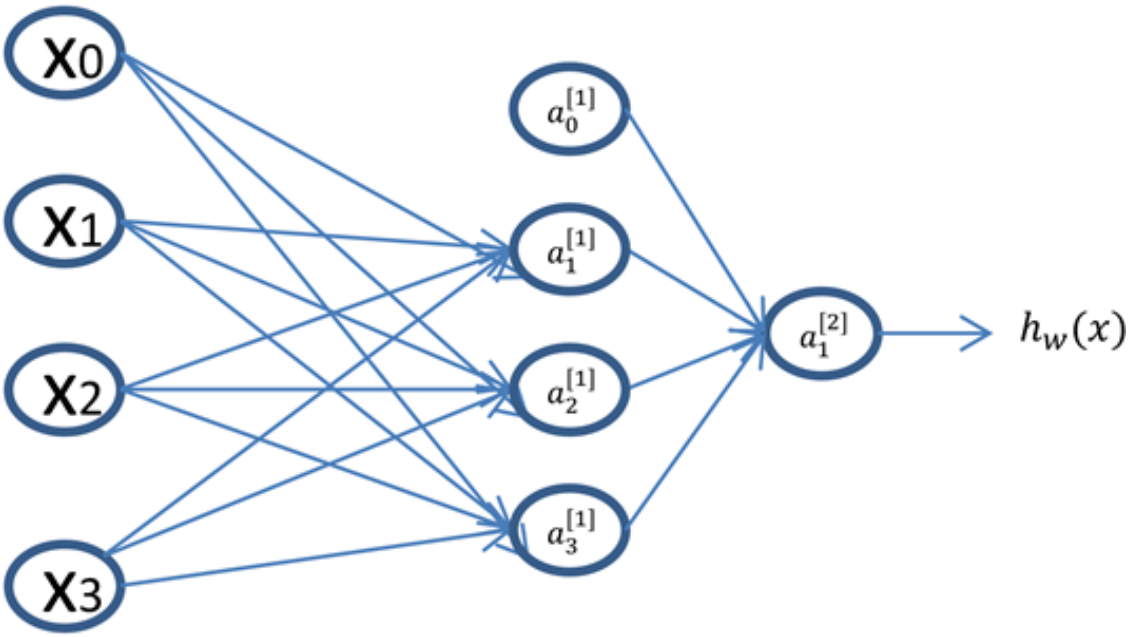


Рисунок 2.8. Схема многослойной сети с одним скрытым слоем

Для настройки весов *w* нейронной сети (обучения сети) используют функцию стоимости, напоминающую функцию стоимости для логистической регрессии (Eq. 2.12).

$$J(W) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k{}^{(i)} \log\left(h_\Theta(x^{(i)})\right)_k + \left(1 - y_k{}^{(i)}\right)\log\left(1 - h_\Theta(x^{(i)})\right)_k\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}(w_{ji}^{l})^2, \qquad \text{(Eq. 2.17)}$$

где L – количество слоев нейронной сети; s_l – количество нейронов в слое l; K – количество классов (равно количеству нейронов в выходном слое); *W* – матрица весов.

Достоинством нейронной сети является возможность классификации с несколькими классами. В случае классификации объектов одного класса, то есть тогда, когда мы должны отделить условно «положительные» объекты от всех остальных, количество нейронов в выходном слое может быть равным и 1 (рисунок 1.5). В этом случае принадлежность объекта к классу «положительных» определяется значением функции гипотезы, то есть если $h_\Theta(x^{(i)}) > 0.5$, то объект принадлежит к искомому классу. Однако чаще, в том числе с целью унификации, используется метод голосования («победитель забирает все»), когда сеть имеет в выходном слое 2 нейрона для двух классов объектов (рисунок 1.6), три для трех и т.д.

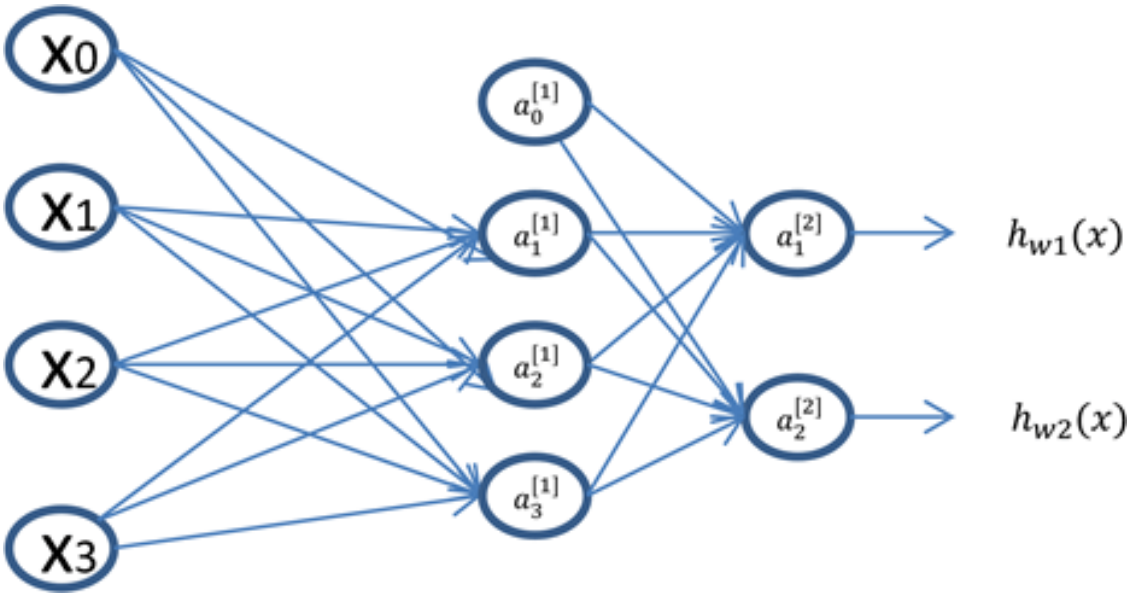


Рисунок 2.9. Схема многослойной сети с двумя выходами

Для обучения, то есть минимизации функции ошибки многослойной ИНС, используют алгоритм обратного распространения ошибки (Backpropagation of errors – BPE) [55] и его модификации, направленные на ускорение процесса обучения.

2.6.3. Алгоритм обратного распространения ошибки

Суть алгоритма BPE заключается в следующем. Для тренировочного набора примеров

$$\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$$

устанавливаем выход первого слоя нейронов:

$$a^{[0]} = x^{(i)}$$

Шаг 1. Выполняем этап прямого распространения сигнала по слоям сети, то есть вычисляем сигнал на выходе сети, выполняя расчет для каждого нейрона в каждом слое, как показано в выражениях 1.4, 1.5. Результаты в виде выходных значений нейронов сети $a^{[0]},a^{[1]},\ldots,a^{[L]}$ сохраняем в промежуточном хранилище (кэш).

Шаг 2. Используя полученный результат на выходе сети $a^{[L]} = h_w{}^{(i)}(x)$, и необходимое для данного примера выходное значение $y^{(i)}$, рассчитываем ошибку выходного слоя:

$$dz^{[L]} = y^{(i)} - a^{[L]},$$

где L – номер выходного слоя нейронной сети.

[55] Werbos P. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. – Harvard University, 1974. – 38 p.

Шаг 3. «Возвращаем» ошибку, распространяя ее обратно по сети с учетом значения производной:

$$dz^{[L-1]} = w^{[L]} dz^{[L]} .* g'\left(a^{[L-1]}\right),$$

где знак * – символ поэлементного умножения; g' – производная.

Производная сигмоидальной активационной функции:

$$g'\left(z^{[L-1]}\right) = a^{[L-1]} .* \left(1 - a^{[L-1]}\right).$$

Для любого скрытого слоя сети:

$$dz^{[i]} = w^{[i+1]} dz^{[i+1]} .* g'\left(a^{[i]}\right).$$

В случае сигмоидальной активационной функции:

$$dz^{[i]} = w^{[i+1]} dz^{[i+1]} * a^{[i]} * \left(1 - a^{[i]}\right).$$

Рассчитанное значение градиентов ошибки $dz^{[1]}, dz^{[2]}, \ldots, dz^{[L]}$ также сохраняем в кэше.
Шаг 4. Модифицируем веса сети с учетом значения ошибки для всех слоев $I \in L$:

$$w^{[i]} = w^{[i]} + \rho * dz^{[i]} * a^{[i-1]}, \qquad \text{(Eq. 2.18)}$$

где i – номер слоя сети; ϱ – параметр обучения (learning rate) ($0 < \varrho < 1$); $\Theta^{(i)}$ – матрица весов слоя i; $dz^{[i]}$ – рассчитанное значение ошибки i-го слоя (точнее говоря, градиент ошибки).

Получив измененные значения весов, повторяем шаги 1–4 до достижения некоторого минимального значения ошибки либо заданное количество раз.

Процесс обучения искусственной нейронной сети можно представить в виде следующей схемы (рисунок 2.10):

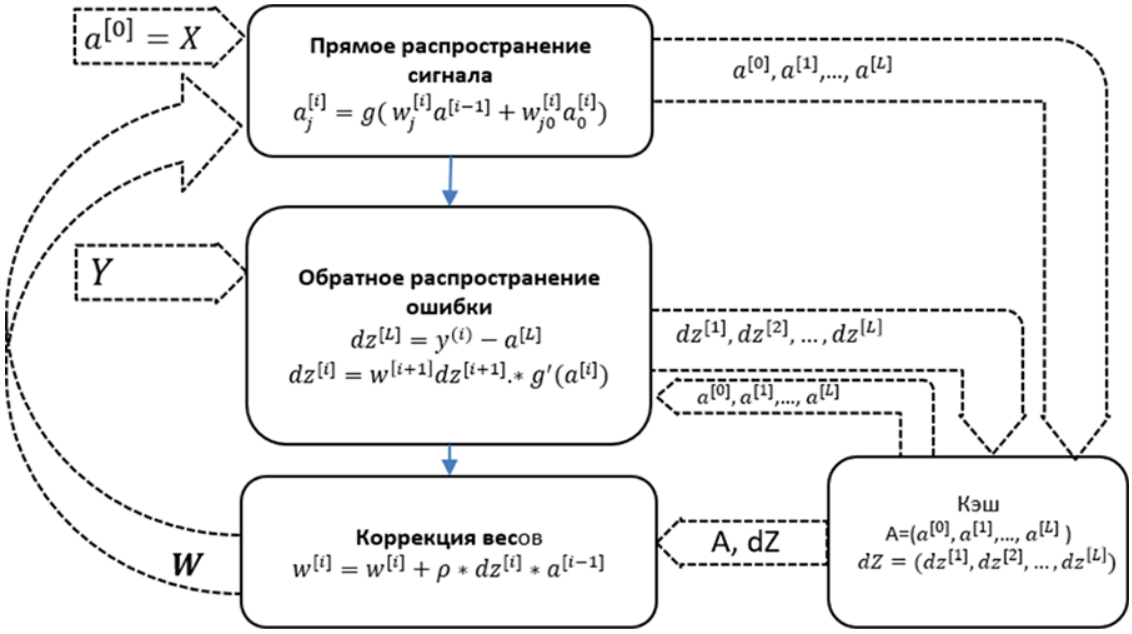


Рисунок 2.10. Итеративный процесс обучения искусственной нейронной сети

Рассмотрим пошаговый пример расчета прямого распространения сигнала, обратного распространения ошибки и коррекции весов.

Пошаговый пример расчета алгоритма обратного распространения ошибки

В этом примере (рисунок 2.11) веса нейронной сети будем обозначать символом w, смещения b. Номер слоя, как и ранее, указываем верхним индексом в квадратных скобках для того, чтобы не путать с индексом обучающего примера, номер нейрона в слое – нижним индексом. Выход нейрона по-прежнему обозначаем символом a.

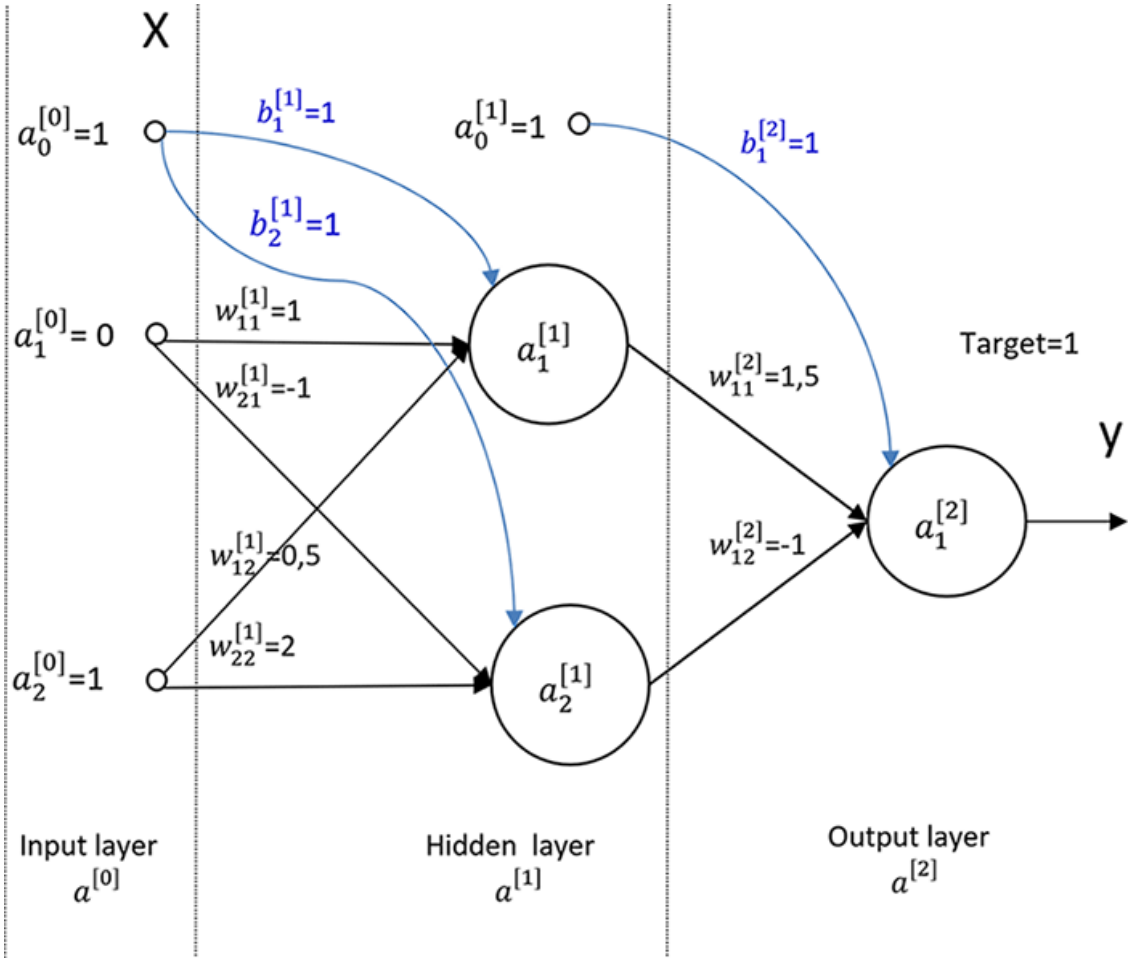


Рисунок 2.11. Пример нейронной сети с одним скрытым слоем

Входной слой с его входами x для единообразия последующих матричных операций обозначаем как нулевой слой – $a^{[0]}$. В нашем примере x1 = 0, x2 = 1, тогда $a_1^{[0]}$ = x1 = 0 и $a_2^{[0]}$ = x2 = 1. Смещение (bias) во всех слоях $a_1^{[l]} = 1$.

На вход сети, таким образом, подается вектор [1,0,1], а на выходе сети необходимо получить y=1.

Шаг 1. Прямое прохождение сигнала.

Рассмотрим прямое прохождение сигнала от входа к выходу:

$$z_1^{[1]} = b_1^{[1]} * a_0^{[0]} + w_{11}^{[1]} * a_1^{[0]} + w_{12}^{[1]} * a_2^{[0]} = 1*1 + 1*0 + 0.5*1 = 1.5$$

$$z_2^{[1]} = b_2^{[1]} * a_0^{[0]} + w_{21}^{[1]} * a_1^{[0]} + w_{22}^{[1]} * a_2^{[0]} = 1*1 + (-1)*0 + 2*1 = 3$$

$$a_1^{[1]} = g(z_1^{[1]}) = g(1.5) = 0.81757$$

$$a_2^{[1]} = g\left(z_2^{[1]}\right) = g(3) = 0.95257$$

Выход нейронной сети:

$$a_1^{[2]} = g\left(b_1^{[2]} * a_0^{[1]} + w_{11}^{[2]} * a_1^{[1]} + w_{12}^{[2]} * a_2^{[1]}\right) =$$
$$a_1^{[2]} = g(1 * 1 + 1{,}5 * 0.81751 + (-1) * 0.95257) = 0.78139$$

Шаг 2. Расчет ошибки выходного слоя.

Сеть должна давать значение $y^{(1)} = 1$, однако получена величина 0.78139. Ошибка, с которой сеть «предсказывает» наш единственный пример, равна разнице между ожидаемым значением и полученным результатом.

$$dz^{[L]} = y^{(i)} - a^{[L]}$$
$$dz_1^{[2]} = y^{(1)} - a_1^{[2]} = 1 - 0.78139 = 0.21861$$

Шаг 3. Обратное распространение ошибки.

Полученную ошибку нужно «распространить обратно» для того, чтобы скорректировать веса сети. Для этого рассчитаем градиенты ошибок нейронов скрытого слоя, используя выражение

$$dz^{[L-1]} = w^{[L-1]} dz^{[L]} .* g'\left(a^{[L-1]}\right) = w^{[L-1]} dz^{[L]} .* a^{[L-1]} * \left(1 - a^{[L-1]}\right).$$

Получим

$$dz_1^{[1]} = dz_1^{[2]} .* w_{11}^{[2]} * \ a_1^{[1]} * (1 - a_1^{[1]}) = 0.048907262419244965$$
$$dz_2^{[1]} = dz_2^{[2]} .* w_{12}^{[2]} \ * \ a_2^{[1]} * \left(1 - a_2^{[1]}\right) = -0.009876050115013725$$

Теперь у нас все готово для того, чтобы, используя градиенты ошибок, пересчитать веса нейронной сети.

Шаг 4. Коррекция весов нейронной сети.

Установим для нашего учебного примера большой коэффициент обучения (learning rate) ro = 0.5. Отметим, что в реальных случаях ro редко превышает 0.1. Здесь мы использовали относительно большое значение, чтобы увидеть значимые изменения весов уже на первой итерации.

Используем выражение (Eq. 2.18) для расчета измененных весов сети:

$$w_{11}^{[2]} = w_{11}^{[2]} + ro * dz_1^{[2]} * a_1^{[1]} = 1.5 + 0.5 * 0{,}21861 * 0.81757 = 1.58936$$
$$w_{12}^{[2]} = w_{12}^{[2]} + ro * dz_1^{[2]} * a_2^{[1]} = -1 + 0.5 * 0{,}21861 * 0.95257 = -0.89588$$
$$b_1^{[2]} = b_1^{[2]} + ro * dz_1^{[2]} * a_0^{[1]} = 1 + 0.5 * 0.21861 * 1 = 1.10930$$

для скрытого слоя:

$$w_{11}^{[1]} = w_{11}^{[1]} + ro * dz_1^{[1]} * a_1^{[0]} = 1 + 0.5 * 0.04891 * 0 = 1$$
$$w_{12}^{[1]} = w_{12}^{[1]} + ro * dz_1^{[1]} * a_2^{[0]} = 0.5 + 0.5 * 0.04891 * 1 = 0.524453$$
$$w_{21}^{[1]} = w_{21}^{[1]} + ro * dz_2^{[1]} * a_1^{[0]} = -1$$
$$w_{22}^{[1]} = w_{22}^{[1]} + ro * dz_2^{[1]} * a_2^{[0]} = 1.99153$$
$$b_1^{[1]} = b_1^{[1]} + ro * dz_1^{[1]} = 1.02445$$
$$b_2^{[1]} = b_2^{[1]} + ro * dz_2^{[1]} = 0.99506$$

Используя скорректированные значения весов, повторим расчет прямого прохождения сигнала и получим значение ошибки выходного слоя:

$$dz_1^{[2]} = 0.17262$$

Видно, что ошибка стала значительно меньше.

После третьей итерации $dz_1^{[2]} = 0.14184$

Примечание. Расчет двух итераций алгоритма BPE с применением Python-numpy приведен в MLF_Example_Of_BPE – https://www.dropbox.com/s/tw6zwht3d5pd4zf/MLF_Example_Of_BPE.html?dl=0

Пример, приведенный выше, является иллюстрацией прямого и обратного хода алгоритма так, что каждый обучающий пример и каждый синаптический коэффициент рассчитываются по отдельности. На практике этапы алгоритма для сети из L-слоев реализуются в матричном виде следующим образом:

$$Z^{[1]} = b^{[1]} + W^{[1]} * X$$
$$A^{[1]} = g(Z^{[1]})$$
$$Z^{[2]} = b^{[2]} + W^{[2]} * A^{[1]}$$
$$A^{[2]} = g(Z^{[2]})$$

…

$$Z^{[l]} = b^{[l]} + W^{[l]} * A^{[l-1]}$$
$$A^{[l]} = g(Z^{[l]}),$$

где $W^{[i]}$ – матрица весов i-го слоя нейронной сети; X – матрица обучающих примеров размерностью n x m (n – число параметров, m – количество обучающих примеров).

Расчет алгоритма градиентного спуска для нейронной сети в матричном виде:

$$dZ^{[L]} = Y^{(i)} - A^{[L]}$$
$$dW^{[l]} = \frac{1}{m} dZ^{[L]} A^{[l-1]T}$$
$$db^{[L]} = dZ^{[L]}$$

…

$$dZ^{[i]} = W^{[i+1]} dZ^{[i+1]} .* g'(A^{[i]})$$
$$db^{[i]} = dZ^{[i]}$$

Примечание. Важно отметить, что алгоритм обратного распространения «требует», чтобы начальные значения весов были небольшими случайными величинами. То есть начальная инициализация требует нарушения «симметрии» для того, чтобы нейроны сети изменяли свои веса «индивидуально». Нулевые значения неприемлемы, поскольку градиент ошибки также будет нулевым и веса не будут корректироваться. Большие значения, сравнимые по величине со значениями, подаваемыми на вход сети, приведут к тому, что алгоритм не будет сходиться. Приведенный выше пример начальных значений весов и смещений является исключительно учебным.

Выражения, приведеные выше, говорят о том, что на вход сети подаются все обучающие примеры «одновременно» и значения градиентов ошибки рассчитываются сразу для всех примеров. Этот процесс составляет одну эпоху обучения. Batch Gradient Descent – это процесс обучения, когда все обучающие примеры используются одновременно. Нескольких десятков или сотен эпох обычно достаточно для достижения оптимальных значений весов матриц $W^{[i]}$.

Однако, когда количество примеров очень велико, примеры разбиваются на группы, которые можно поместить в оперативную память компьютера, и эпоха обучения включает последовательную подачу этих групп. При этом возможны два подхода [56]:

Stochastic Batch Gradient Descent – когда группа включает лишь один пример, выбираемый случайно из множества обучающих примеров.

Mini Batch Gradient Descent – когда группа включает некоторое количество примеров.

Примечание. Для ускорения обучения рекомендуется подбирать размер группы равный степени двойки – 8, 16, 32, …, 1024 – в идеале так, чтобы пакет примеров мог быть помещен в кэш-память процессора.

При применении современных пакетов машинного обучения программисту не приходится заботиться о выполнении алгоритма BPE. Он реализуется путем выбора того или иного оптимизационного алгоритма (solver). Часто применяются lbfs, adam. Например, загрузка многослойного персептрона (multilayer perceptron – MLP) и создание объекта классификатора осуществляются следующим образом:

```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(hidden_layer_sizes = [10, 10], alpha = 5, random_state = 0, solver='lbfgs')
```

[56] Batch, Mini-Batch & Stochastic Gradient Descent. – https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a

Пример применения MLPClassifier приведен в разделе 2.8 Пример простого классификатора.

2.6.5. Активационные функции

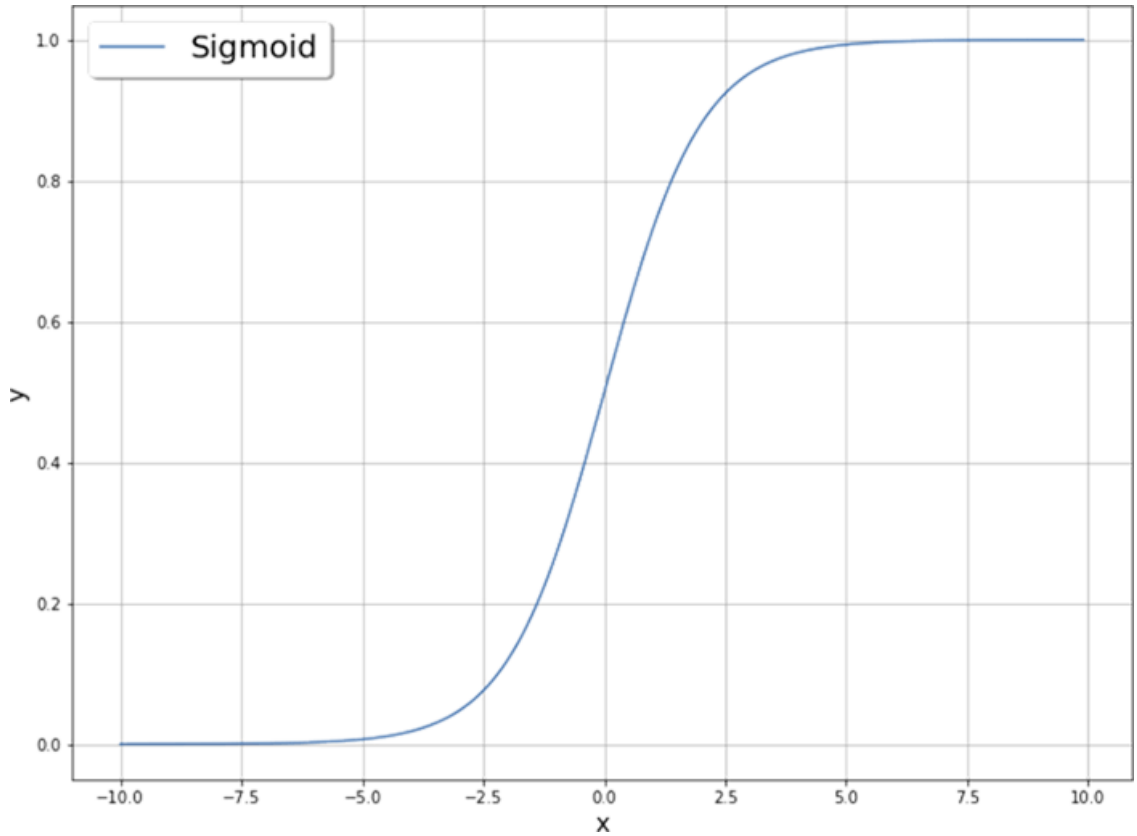
Нелинейная активационная функция играет фундаментальную роль в процессе обучения нейронной сети. Именно ее применение позволяет нейронной сети обучаться сложным закономерностям, содержащимся в исходных данных. Кроме уже упомянутой сигмоидальной функции часто используются и несколько других активационных функций (рисунок 2.12), описываемых уравнениями

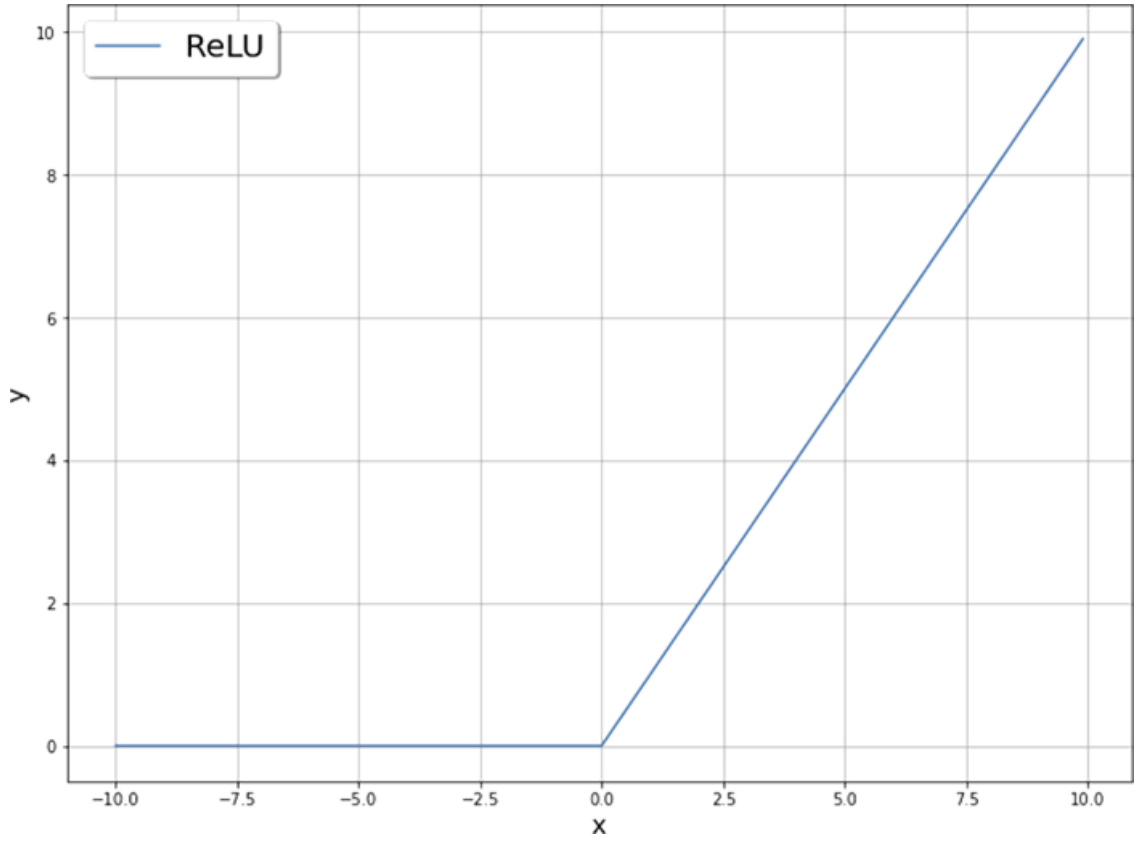
$$\text{Sigmoid: } y = \frac{1}{1+e^{-x}}$$

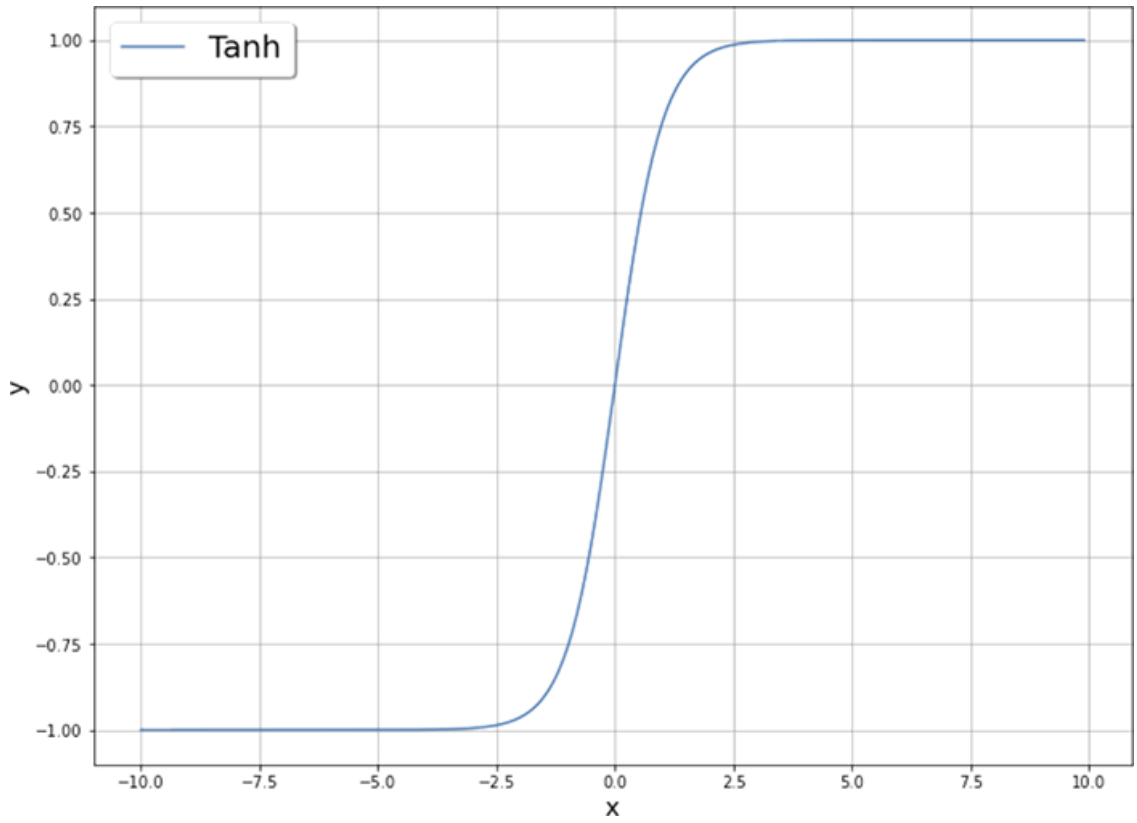
$$\text{Tanh: } y = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

$$\text{ReLU: } y = \max(0, x)$$

$$\text{Leaky ReLU: } y = \max(0.01x, x)$$




ReLU
y
x
10
8
6
4
2
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Tanh
y
x
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

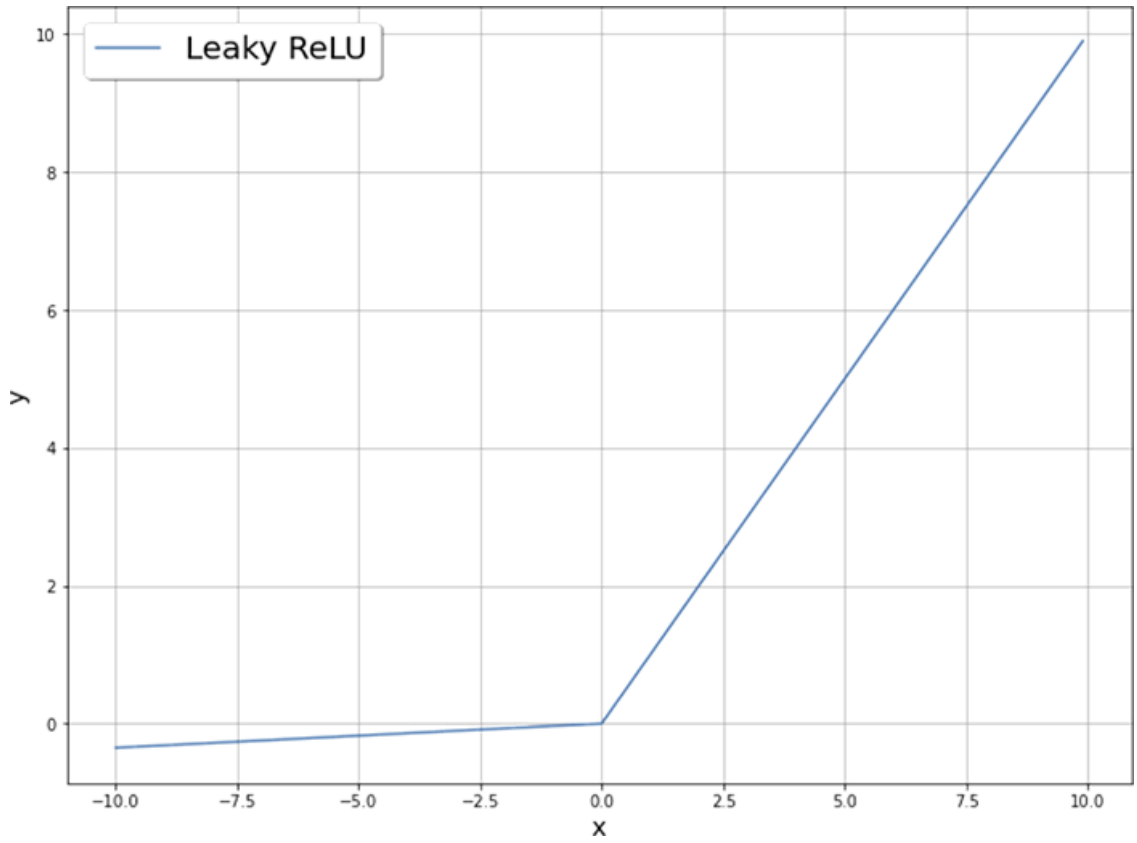


Рисунок 2.12. Активационные функции, применяемые в нейронных сетях

Резонный вопрос: «Почему исследователи используют несколько видов активационных функций?» Ответ, следующий: вычислительные затраты на расчеты результатов весьма велики, особенно в крупномасштабных сетях. Как известно, расчет выхода каждого слоя нейронной сети выполняется с использованием активационной функции. А в процессе выполнения алгоритма обратного распространения ошибки используется производная активационной функции. И в том, и в другом случае ReLU имеет большое преимущество с точки зрения вычислительных затрат. Следовательно, нейронная сеть будет обучаться значительно быстрее. С другой стороны, использование сигмоидальной функции для выходного слоя нейронной сети позволяет вычислять оценку вероятности принадлежности к классу, поскольку она принимает значения в диапазоне от 0 до 1.

2.7. Контрольные вопросы

Какие ученые оказали существенное влияние на развитие коннективизма?

Коннективизм или коннекционизм – в чем отличие этих двух терминов?

Приведите схему классического нейрона.

Приведите схему многослойной сети прямого распространения.

Как вычисляется выход многослойной нейронной сети прямого распространения?

Приведите функцию стоимости многослойной сети прямого распространения.

Сколько основных шагов в алгоритме обратного распространения? В чем их назначение?

Каково назначение кэша в процессе выполнения алгоритма обратного распространения ошибки?

Что такое эпоха обучения нейронной сети?

Укажите, какие виды процессов обучения нейронной сети применяются на практике.

В чем заключается сходство и отличие активационных функций, применяемых в нейронных сетях?

В чем заключается сходство активационных функций, применяемых в нейронных сетях?

В чем заключается преимущество активационной функции ReLU?

Какая активационная функция удобна для реализации бинарного классификатора?

Какими должны быть начальные значения весов и смещений в нейронной сети?

2.8. Пример простого классификатора

Рассмотрим интересную задачу классификации изображений, представленную в качестве примера применения TensorFlow [57]. TensorFlow в нашем решении мы используем лишь для загрузки данных, а в качестве классификатора применим упомянутый выше MLPClassifier. Суть задачи заключается в том, что необходимо классифицировать предметы одежды по их монохромным изображениям в низком разрешении (28 x 28). Набор данных Fashion-MNIST содержит 60 000 изображений для обучения и 10 000 для тестирования, начиная от футболок и брюк и заканчивая сумками и туфлями. Всего 10 классов изображений. Классы, пронумерованные от 0 до 9, и их описание показаны на рисунке 2.13.

[57] Обучи свою первую нейросеть: простая классификация. – https://www.tensorflow.org/tutorials/keras/classification

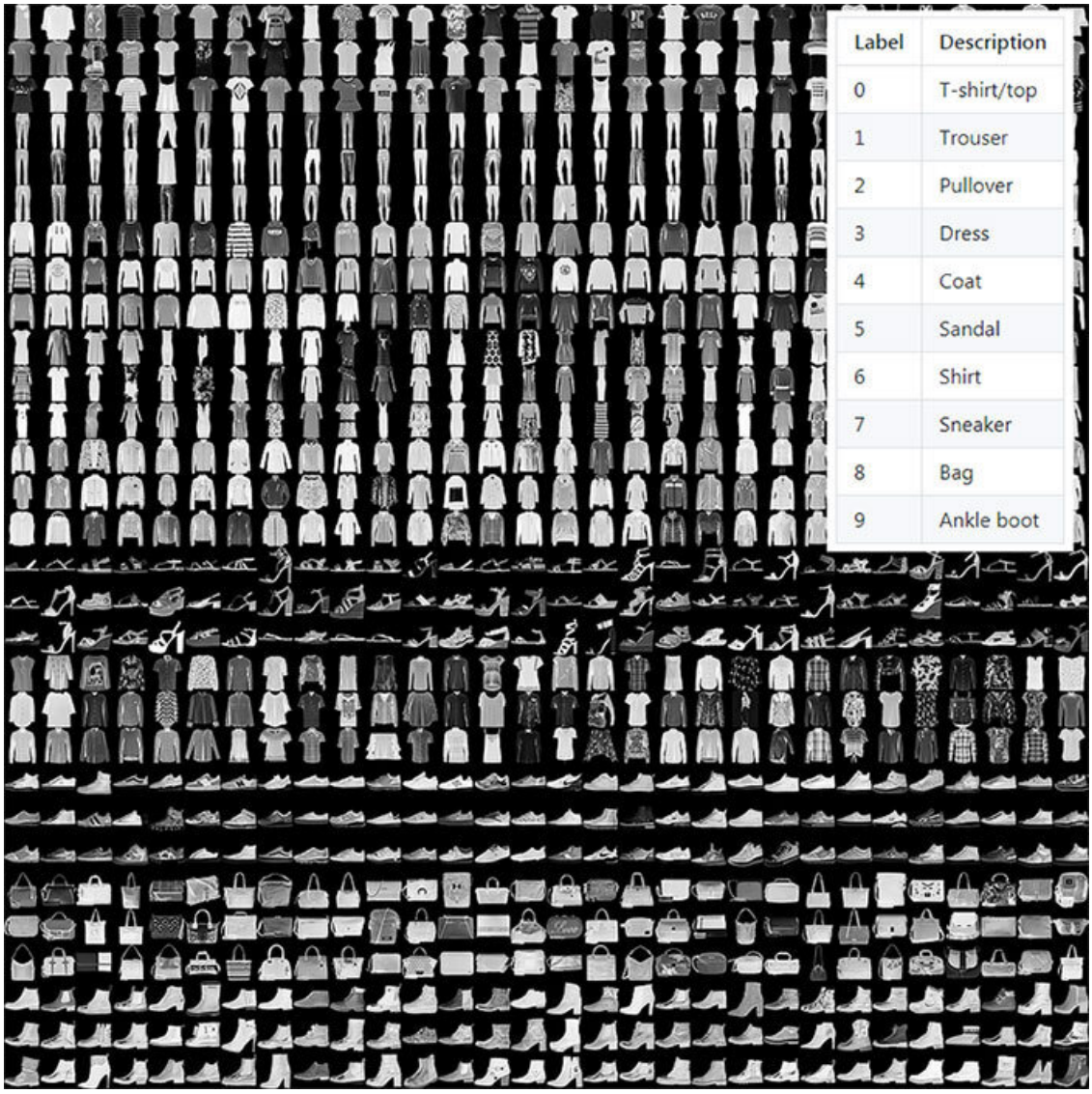


Рисунок 2.13. Образцы Fashion-MNIST

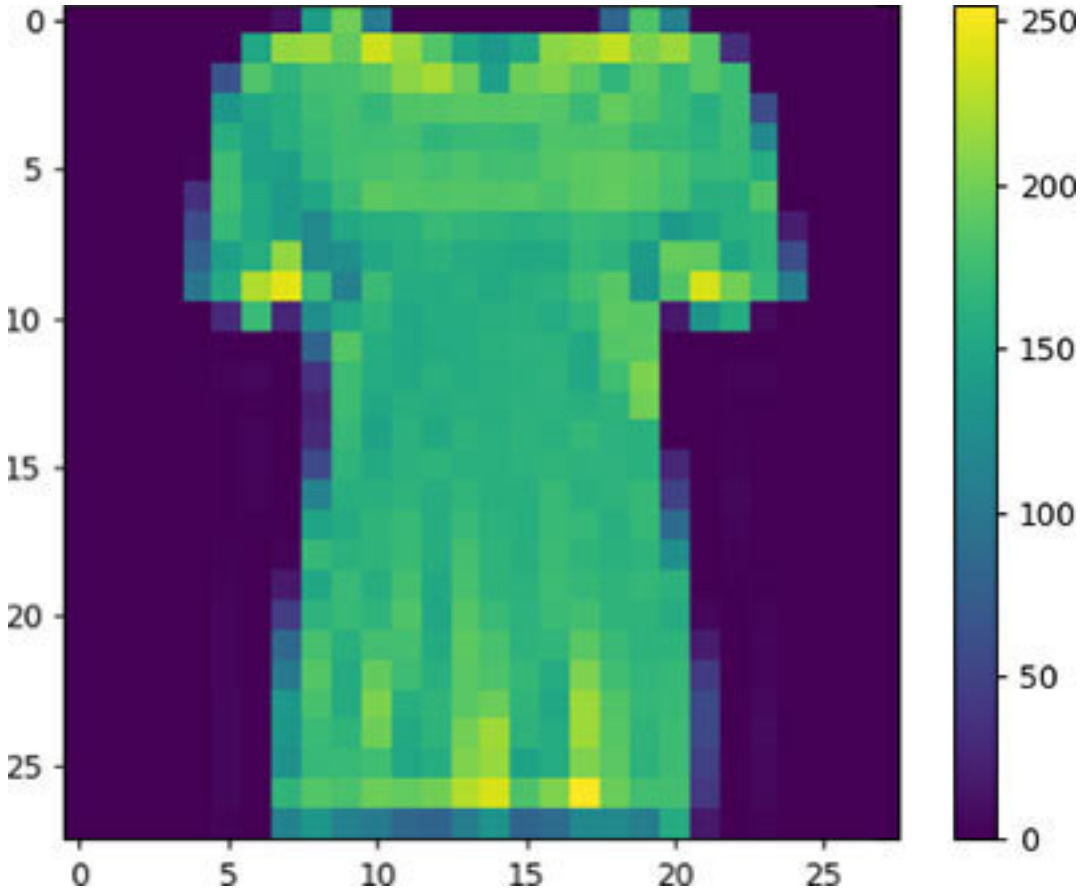
Fashion-MNIST разработан в дополнение к классическому набору данных MNIST, который часто используют как «Hello, World» для отладки методов машинного обучения в задачах компьютерного зрения. MNIST содержит изображения рукописных цифр (0, 1, 2 и т.д.) в формате, идентичном формату изображений одежды набора Fashion-MNIST. Для современных программ компьютерного зрения MNIST стал «слишком прост», поэтому применение более сложного набора данных полезно для отладки систем машинного обучения.

Загрузить набор данных можно, используя keras. Предварительно потребуется загрузить необходимые библиотеки:

```
# TensorFlow и tf.keras
import tensorflow as tf
from tensorflow import keras
# Вспомогательные библиотеки
import numpy as np
import matplotlib.pyplot as plt
```

Теперь можно загрузить набор данных и посмотреть одно из изображений:

```
fashion_mnist = keras.datasets.fashion_mnist
(X_train1, y_train),(X_test1,y_test)= fashion_mnist.load_data()
plt.figure()
plt.imshow(X_train1[10])
plt.colorbar()
plt.grid(False)
plt.show()
```



Как видно, диапазон изменения яркости пикселя – от 0 до 255. Если подать такие значения на вход нейронной сети, качественные результаты классификации существенно упадут. Поэтому все значения нужно нормировать так, чтобы на вход сети поступили значения в диапазоне от 0 до 1, просто разделив каждое значение на 255:

```
X_train1=X_train1/255.0
X_test1=X_test1/255.0
```

Следующее, что нам необходимо сделать в процессе предобработки, – это преобразовать двумерные массивы изображений 28 x 28 в одномерные векторы. Каждый такой вектор станет набором входных параметров размерностью 784:

```
X_train=np.reshape(X_train1,(X_train1.shape[0],X_train1.shape[1]*X_train1.shape[2]))
X_test=np.reshape(X_test1,(X_test1.shape[0],X_test1.shape[1]*X_test1.shape[2]))
```

В результате матрица X_train размерностью (60 000, 28, 28) будет преобразована в матрицу размером (60 000, 784), которую можно подать на вход нейронной сети для тренировки.

```
from sklearn.neural_network import MLPClassifier
```

```
clf = MLPClassifier(hidden_layer_sizes = [15, 15,],
      alpha = 0.01,random_state = 0,
      solver='adam').fit(X_train, y_train)
```

Обучение нейронной сети может занять несколько минут. Затем можно оценить качественные показатели классификатора командами:

```
predictions=clf.predict(X_test)
print('Accuracy of NN classifier on training set: {:.2f}'
    .format(clf.score(X_train, y_train)))
print('Accuracy of NN classifier on test set: {:.2f}'
    .format(clf.score(X_test, y_test)))
print(classification_report(y_test,predictions))
matrix = confusion_matrix(y_test, predictions)
print('Confusion matrix on test set\n',matrix)
```

Значение accuracy может быть примерно следующим:
Accuracy of NN classifier on training set: 0.89
Accuracy of NN classifier on test set: 0.86

Изменяя количество нейронов в слоях сети и параметр регуляризации alpha (например, hidden_layer_sizes = [75, 75], alpha = 0.015), можно несколько улучшить результат:
Accuracy of NN classifier on training set: 0.91
Accuracy of NN classifier on test set: 0.88

Примечание. Программу данного раздела MLF_MLP_Fashion_MNIST_001.ipynb можно получить по ссылке – https://www.dropbox.com/s/ryk05tyxwlhz0m6/MLF_MLP_Fashion_MNIST_001.html?dl=0

2.9. Алгоритм k ближайших соседей (k-Nearest Neighbor – k-NN)

Алгоритм [58, 59] основан на подсчете количества объектов каждого класса в сфере (гиперсфере) с центром в распознаваемом (классифицируемом) объекте. Классифицируемый объект относят к тому классу, объектов у которого больше всего в этой сфере. В данном методе предполагается, что веса выбраны единичными для всех объектов.

Если веса не одинаковы, то вместо подсчета количества объектов можно суммировать их веса. Таким образом, если в сфере вокруг распознаваемого объекта 10 эталонных объектов класса А весом 2 и 15 ошибочных/пограничных объектов класса Б весом 1, то классифицируемый объект будет отнесен к классу А.

Веса объектов в сфере можно представить как обратно пропорциональные расстоянию до распознаваемого объекта. Таким образом, чем ближе объект, тем более значимым он является для данного распознаваемого объекта. Расстояние между классифицируемыми объектами может рассчитываться как расстояние в декартовом пространстве (эвклидова метрика), но можно использовать и другие метрики: манхэттенскую (Manhattan), метрику Чебышева (Chebyshev), Минковского (Minkowski) и др.

---

[58] Dudani, Sahibsingh A. The Distance-Weighted k-Nearest-Neighbor Rule // Systems, Man, and Cybernetics. – 1976. – Vol. SMC-6. – Issue 4. – P. 325–327.

[59] K-Nearest Neighbors algorithm. – http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm (2012-07-05).

В итоге метрический классификатор можно описать так:

$$a(u; X^l) = arg\max_{y \in Y} \sum_{i=1}^{l} [y_u^{(i)} = y] w(i,u), \text{(Eq. 2.19)}$$

где w(i, u) – вес i-го соседа распознаваемого объекта u; a(u;Xl) – класс объекта u, распознанный по выборке Xl.

Радиус гиперсферы может быть как фиксированным, так и изменяемым, причем в случае с изменяемым радиусом радиус для каждой точки подбирается так, чтобы количество объектов в каждой сфере было одинаковым. Тогда при распознавании в областях с разной плотностью выборки количество «соседних» объектов (по которым и происходит распознавание) будет одинаковым. Таким образом, исключается ситуация, когда в областях с низкой плотностью не хватает данных для классификации.

В целом это один из самых простых, но часто неточных алгоритмов классификации. Алгоритм также отличается высокой вычислительной сложностью. Объем вычислений при использовании эвклидовой метрики пропорционален квадрату от числа обучающих примеров.

Рассмотрим пример.

Загрузка соответствующей библиотеки и создание классификатора выполняются командами:

```
from sklearn import neighbors
clf = neighbors.KNeighborsClassifier(n_neighbors=5, weights='distance')
```

Используем уже упомянутый ранее набор данных Fashion-MNIST. Однако в связи с тем, что скорость обучения и особенно классификации KNeighborsClassifier значительно ниже, чем MLP, будем использовать только часть набора: 10 000 примеров для обучения и 2000 для тестирования:

```
X_train1=X_train1[0:10000,:,:]
y_train=y_train[0:10000]
X_test1=X_test1[0:2000,:,:]
y_test=y_test[0:2000]
```

Процесс обучения классификатора:

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors = 5, weights='distance')
clf.fit(X_train, y_train)
```

Вывод результатов выполняется практически так же, как и в предыдущем примере. Качественные показатели классификатора, примерно следующие:

```
Accuracy of kNN classifier on training set: 1.00
Accuracy of kNN classifier on test set: 0.82
```

Примечание. Программу MLF_KNN_Fashion_MNIST_001.ipynb, использованную в данном разделе, можно получить по ссылке – https://www.dropbox.com/s/ei1tuaifi2zj2ml/MLF_KNN_Fashion_MNIST_001.html?dl=0

2.10. Алгоритм опорных векторов

Алгоритм опорных векторов (Support Vector Machines) [60] относится к группе граничных методов: он определяет классы при помощи границ областей. В основе метода лежит понятие плоскостей решений. Плоскость решения разделяет объекты с разной классовой принадлежностью. В пространствах высоких размерностей вместо прямых необходимо рассматривать гиперплоскости – пространства, размерность которых на единицу меньше, чем размерность исходного пространства. В R3, например, гиперплоскость – это двумерная плоскость.

Метод опорных векторов отыскивает образцы, находящиеся на границах классов (не меньше двух), т.е. опорные векторы, и решает задачу нахождения разделения множества объектов на классы с помощью линейной решающей функции. Метод опорных векторов строит классифицирующую функцию f(x) в виде:

$$f(x) = sign(\langle w, s\rangle + b),$$

где *#w,s#* – скалярное произведение; w – нормальный (перпендикулярный) вектор к разделяющей гиперплоскости; b – вспомогательный параметр, который равен по модулю расстоянию от гиперплоскости до начала координат. Если параметр b равен нулю, гиперплоскость проходит через начало координат.

Объекты, для которых f(x) = 1, попадают в один класс, а объекты с f(x) = -1 – в другой.

С точки зрения точности классификации лучше всего выбрать такую прямую, расстояние от которой до каждого класса максимально. Такая прямая (в общем случае – гиперплоскость) называется оптимальной разделяющей гиперплоскостью. Задача состоит в выборе w и b, максимизирующих это расстояние.

В случае нелинейного разделения существует способ адаптации машины опорных векторов. Нужно вложить пространство признаков Rn в пространство H большей размерности с помощью отображения: φ = Rn → H. Тогда решение задачи сводится к линейно разделимому случаю, т.е. разделяющую классифицирующую функцию вновь ищут в виде: *f(x)=sign(#w,ϕ(x)#+b)*.

Возможен и другой вариант преобразования данных – перевод в полярные координаты:

$$\begin{cases} x_1 = r\cos(\phi), \\ x_2 = r\sin(\phi). \end{cases}$$

В общем случае машины опорных векторов строятся таким образом, чтобы минимизировать функцию стоимости вида:

$$J(\Theta) = C\sum_{i=1}^{m} y^{(i)} S_1(\Theta^T, f_k{}^{(i)}(x^{(i)})) + (1 - y^{(i)}) S_0(\Theta^T, f_k{}^{(i)}(x^{(i)})) + \frac{1}{2}\sum_{j=1}^{m}\theta_j^2, \quad \text{(Eq. 2.20)}$$

где S_1 и S_0 – функции, заменяющие $log(h_\theta)$ и $log(1–h_\theta)$ в выражении для логистической регрессии (f2) (обычно это кусочно-линейные функции); f_k – функция ядра, выполняющая отображение φ и определяющая значимость объектов обу-

[60] Support vector machine. – http://en.wikipedia.org/wiki/Support_vector_machine (2012-02-22).

чающего множества в пространстве признаков. Часто используется гауссова функция $f_k^{(i)}(x^{(i)}) = exp(\frac{|x-x^{(i)}|^2}{2\delta^2})$, которая для любого x позволяет оценить его близость к $x^{(i)}$ и тем самым формировать границы между классами, более близкие или более отдаленные от опорного объекта, устанавливая значение δ, C – регуляризационный параметр (C=1/λ).

Существенным недостатком классификатора является значительное возрастание времени обучения при увеличении количества примеров. Другими словами, алгоритм обладает высокой вычислительной сложностью.

Рассмотрим пример.

Подключение алгоритма и создание классификатора выполняются командами:

```
from sklearn.svm import SVC
clf = SVC(kernel = 'rbf', C=1)
```

Используем еще раз набор данных Fashion-MNIST. Скорость обучения и особенно классификации SVC значительно ниже, чем MLP, поэтому, как и в случае с KNeighborsClassifier, будем использовать только часть набора: 10 000 примеров для обучения и 2000 для тестирования. Обучение классификатора со стандартными параметрами:

```
from sklearn.svm import SVC
clf = SVC(kernel = 'rbf',C=1).fit(X_train, y_train)
```

В результате получим примерно следующие значения accuracy:

```
Accuracy of SVC classifier on training set: 0.83
Accuracy of SVC classifier on test set: 0.82
```

Отметим, что, применив поиск оптимальных параметров классификатора (см. далее раздел «Подбор параметров по сетке»), можно получить значение accuracy, близкое к 0.87.

> *Примечание*. Ноутбук MLF_SVC_Fashion_MNIST_001.ipynb, реализующий упомянутый пример, можно загрузить по ссылке – https://www.dropbox.com/s/0p1i1dqk8wqwp5x/MLF_SVC_Fashion_MNIST_001.html?dl=0

Набор классификаторов scikit-learn включает кроме упомянутых алгоритмов еще и GaussianProcessClassifier, DecisionTreeClassifier, GaussianNB и др. Сравнение между собой классификаторов, имеющих стандартные параметры, описано в классическом примере [61], который можно рекомендовать как первую ступень в разработке программы выбора лучшего классификатора.

[61] Classifier comparison. – https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

2.11. Статистические методы в машинном обучении. Наивный байесовский вывод

2.11.1. Теорема Байеса и ее применение в машинном обучении

Машинное обучение использует теорию вероятности для предсказания и классификации. Особенностью ML является создание алгоритмов, способных обучаться. Способ обучения в данном случае заключается в использовании статистических закономерностей. Одна из таких относительно простых возможностей – использование теоремы Байеса.

Напомним, что теорема Байеса говорит о том, что если известна априорная вероятность гипотезы A – P(A), априорная вероятность гипотезы B – P(B) и условная вероятность наступления события B при истинности гипотезы A – P(B|A), то мы можем рассчитать условную вероятность гипотезы A при наступлении события B:

$$P(A|B) = \mathrm{P(B|A)}\frac{\mathrm{p(A)}}{\mathrm{P(B)}}. \qquad \text{(Eq. 2.21)}$$

Рассмотрим пример.

Предположим, что нам известна статистика дворовых игр в футбол и погода, при которых они состоялись, например, в таком виде:

	Play	Weather
0	no	Rainy
1	no	Rainy
2	no	Rainy
3	yes	Rainy
4	yes	Rainy
5	no	Sunny
6	no	Sunny
7	yes	Sunny
8	yes	Sunny
9	yes	Sunny
10	yes	Overcast
11	yes	Overcast
12	yes	Overcast
13	yes	Overcast

То есть мы имеем информацию о количестве игр (14) и сведения о трех видах погоды, при которой они проходили: sunny – солнечно, rainy – дождливо, overcast – пасмурно. Попробуем рассчитать, состоится ли очередная игра, если на улице солнечно (sunny). Для этого нам нужно рассчитать вероятность того, что игра состоится ('yes') при условии 'Sunny', то есть нам нужно рассчитать:

P('yes'|'Sunny').

Другими словами, мы хотим оценить вероятность справедливости гипотезы, что A = 'yes' – игра состоится при условии, что B = 'Sunny'.

Для такого расчета нам нужно вычислить априорные вероятности того, что погода солнечная – P('Sunny') и что игра вообще состоится P('yes'). Кроме этого, рассчитать условную вероятность того, что погода является солнечной при состоявшейся игре P('Sunny'|'yes'). Тогда в соответствии с теоремой Байеса мы сможем рассчитать искомую вероятность:

P('yes'|'Sunny') = P('Sunny'|'yes') * P('yes') / P('Sunny')

Используя таблицу, легко посчитать оценки указанных вероятностей. Положим, что:

A_value = 'yes'

B_hypothes = 'Sunny'

Тогда цель нашего расчета – получить значение величины:

P(A_value|B_hypothes) = P('yes'|'Sunny') = P('Sunny'|'yes') * P('yes') / P('Sunny')

Рассчитаем условную вероятность:

P('Sunny'|'yes') = 3 / 9 = 0.33

Рассчитаем априорные вероятности солнечной погоды и того, что игра состоится:
P('Sunny') = 5 / 14 = 0.36
P('yes') = 9 / 14 = 0.64
Подставив полученные значения, получим:
P('yes'|'Sunny') = 0.33 * 0.64 / 0.36 = 0.60.

2.11.2. Алгоритм Naïve Bayes

Однако как быть, если игра зависит не только от погоды, но и от других условий, например, готовности поля, здоровья игроков и т.п.? В этом случае вывод классификатора можно строить на отношении условных вероятностей следующим образом:

$$NBI_1 = \frac{P(\prime yes\prime)}{P(\prime no')} \prod_i^F \frac{P(c_i|'yes')}{P(c_i|'no\prime)}, \quad \text{(Eq. 2.22)}$$

где NBI_1 – вывод наивного байесовского классификатора (Naïve Bayes Inference); ci – i-е свойство или признак из F (features), влияющий на вывод классификатора. Отметим, что если P('yes')= P('no'), то первый сомножитель будет равен 1. Это означает, что если априорные вероятности исходов одинаковы, то формула упрощается к виду:

$$\mathrm{NBI}_2 = \prod_i^F \frac{P(c_i|'yes')}{P(c_i|'no\prime)}. \qquad \text{(Eq. 2.22.1)}$$

Оценки вероятностей вычисляются следующим образом:

$$P(c_i|'yes') = \frac{freq(c_i,'yes')}{N}, \quad \text{(Eq. 2.23)}$$

где freq – частота; N – частота всех случаев данного класса. Примером служит выражение P('Sunny'|'yes') = 3 / 9 = 0,33.

В выражении Eq. 2 величина NBI принимает значения от 0 до +∞. Если NBI < 1, то это свидетельствует в пользу отрицательной гипотезы ('no'). Если NBI > 1, то это свидетельство того, что текущее сочетание условий дает возможность положительного вывода ('yes'). Отметим, что если мы используем выражение Eq. 2, то мы должны примириться с неравновесностью такого вывода.

Кроме того, если некоторые признаки встречаются только в сочетании с 'yes' или 'no', то мы можем получить ошибку вывода, когда произведение обращается в ноль либо происходит деление на ноль. Третья проблема связана с тем, что оценки условных вероятностей обычно имеют небольшое значение, и если их много, то итоговое произведение может стать меньше машинного нуля. Эти вычислительные недостатки разрешаются путем сглаживания и использования суммы логарифмов вместо произведения вероятностей. Чаще всего для исключения деления на ноль применяется сглаживание по Лапласу, например, для положительной гипотезы:

$$P(c_i|'yes') = \frac{freq(c_i,'yes')+1}{N+F}. \qquad \text{(Eq. 2.23.1)}$$

В этом выражении F – количество свойств или параметров. В примере ниже F = 2 – параметры: погода (Weather) и состояние поля (Field). В свою очередь, N – частота всех случаев для данного класса, то есть для нашего примера это количество случаев, когда игра состоялась, – 9.

Применение логарифмов позволяет перейти от произведения отношений вероятности к суммам логарифмов этих отношений, так как $\log(a*b) = \log(a) + \log(b)$. Тогда вывод классификатора можно рассчитать следующим образом:

$$NBI_{log} = \log\left(\frac{P('yes')}{P('no')}\right) + \sum_i^F \log\left(\frac{P(c_i|'yes')}{P(c_i|'no')}\right). \qquad \text{(Eq. 2.24)}$$

Применение логарифмов позволяет работать с очень небольшими значениями вероятностей. Второе преимущество заключается в том, что при применении логарифмов шкала вывода будет равномерной в диапазоне от -∞ до +∞. Величина NBI_{log} будет либо больше 0, что означает верность положительной гипотезы, либо меньше 0, что означает справедливость отрицательной гипотезы (рисунок 2.14).

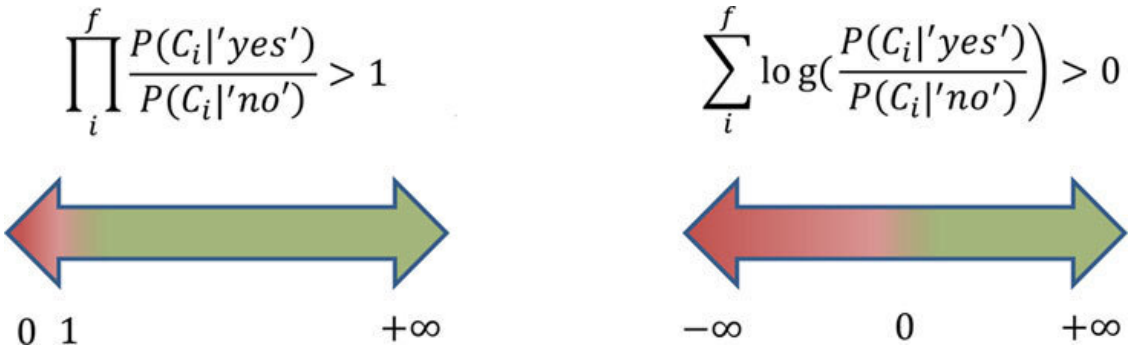


Рисунок 2.14. Шкала вывода алгоритма Naïve Bayes при использовании выражений Eq. 2.21 (слева) и Eq. 2.24 (справа)

Обучение алгоритма Naïve Bayes выполняется просто путем расчета оценок вероятностей (Eq. 2.23, 2.24). После этого вывод обеспечивается по формуле Eq. 2.24.

Рассмотрим пример.

За основу возьмем данные, приведенные в предыдущем параграфе. Добавим еще одно свойство – состояние игрового поля Field. Теперь набор данных содержит два свойства (Weather, Field) и целевую колонку Play:

	Play	Weather	Field
0	no	Rainy	bad
1	no	Rainy	bad
2	no	Rainy	bad
3	yes	Rainy	bad
4	yes	Rainy	good
5	no	Sunny	good
6	no	Sunny	good
7	yes	Sunny	good
8	yes	Sunny	bad
9	yes	Sunny	good
10	yes	Overcast	bad
11	yes	Overcast	good
12	yes	Overcast	bad
13	yes	Overcast	good

По-прежнему будем предсказывать возможность игры, но уже не только в зависимости от погоды, но и принимая во внимание состояние поля (bad, good):

(P('yes'|'Sunny' & 'good').

Так же, как и ранее:

P('Sunny'|'yes') = 3 / 9 = 0.33

В дополнение рассчитаем:

P('Sunny'|'no') = 2 / 5 = 0.4

P('good'|'yes') = 5 / 9 = 0.5555

P('good'|'no') = 2 / 5 = 0.4

Результат с использованием выражения Eq. 2.1:

P('yes'|'Sunny' & 'good') = [P('Sunny'|'yes') / P('Sunny'|'no')] * [P('good'|'yes') / P('good'|'no')] = 1.574,

то есть в предположении, что априорная вероятность того, что игра состоится – P('yes'), равна априорной вероятности того, что игра не состоится – P('no'), получаем значение больше 1, и, следовательно, игра состоится.

Примечание. Поэкспериментировать с NBA можно путем решения задач ML_Lab01.2_NaiveBayesSimpleExampleByPython – https://www.dropbox.com/sh/oto9jus54r4qv7x/AAAcOtl9SE-i6b1zViwMP6Wga?dl=0

2.11.3. Положительные и отрицательные свойства Naïve Bayes

Положительные стороны

Классификация, в том числе многоклассовая, выполняется легко и быстро. Когда допущение о независимости выполняется, Naïve Bayes Algorithm (NBA) превосходит другие алгоритмы, такие как логистическая регрессия (logistic regression), и при этом требует меньший объем обучающих данных.

NBA лучше работает с категорийными признаками, чем с непрерывными. Для непрерывных признаков предполагается нормальное распределение, что является достаточно сильным допущением.

Отрицательные стороны

Если в тестовом наборе данных присутствует некоторое значение категорийного признака, которое не встречалось в обучающем наборе данных, тогда модель присвоит нулевую вероятность этому значению и не сможет сделать прогноз. Это явление известно под названием «нулевая частота» (zero frequency). Данную проблему можно решить с помощью сглаживания. Одним из самых простых методов является сглаживание по Лапласу (Laplace smoothing).

Хотя NBA является хорошим классификатором, значения спрогнозированных вероятностей не всегда являются достаточно точными. Поэтому не следует слишком полагаться на результаты, возвращенные методом predict_proba.

Еще одним ограничением NBA является допущение о независимости признаков. В реальности наборы полностью независимых признаков встречаются крайне редко.

Наивный байесовский метод называют наивным из-за допущений, которые он делает о данных. Во-первых, метод предполагает независимость между признаками или свойствами. Во-вторых, он подразумевает, что набор данных сбалансирован, то есть объекты разных классов представлены в наборе данных в одинаковой пропорции. На практике ни первое, ни второе предположения полностью не выполняются: чаще всего признаки связаны между собой, а реальные наборы данных редко бывают сбалансированными. Типичным примером является задача поиска окончания предложения, например: «Очень сухо, солнечно и жарко в Сахаре». Если полагать что мы должны найти последнее слово (Сахара), то его вероятность, исходя из сочетания слов, должна быть выше, чем, например, Магадан. Но с точки зрения независимости слов алгоритм может с равной вероятностью поставить в качестве окончания предложения любой город или место. Другой пример может быть связан с наборами данных о проникновении в закрытую компьютерную сеть. В таких наборах данных примеров зафиксированных проникновений обычно намного меньше, чем стандартных транзакций. Это приводит к тому, что алгоритм становится излишне «оптимистичным» или, наоборот, «пессимистичным».

Еще одно неудобство связано с тем, что если в тестовом наборе присутствует значение признака, которое не встречалось в обучающем наборе, то модель присвоит этому значению нулевую вероятность или нулевую частоту и не сможет сделать прогноз. С этим недостатком борются, применяя сглаживание по Лапласу так, как описано выше.

2.11.4. Приложения наивного байесовского алгоритма

Мультиклассовая классификация в режиме реального времени. NBA очень быстро обучается, поэтому его можно использовать для обработки данных в режиме реального времени. NBA обеспечивает возможность многоклассовой классификации.

Классификация текстов, фильтрация спама, анализ тональности текста, определение авторства, поиск информации, устранение неоднозначности слов . При решении задач автоматической обработки текстов часто используется статистическая модель естественного языка, NBA ей идеально соответствует. Поэтому алгоритм находит широкое применение в задаче идентификации спама в электронных письмах, анализа тональности текста (sentiment analysis), поиска информации, соответствующей запросу (information retrieval), определения авторства текста (author identification), устранения неоднозначности слов (word disambiguation).

Рекомендательные системы. NBA – один из методов, который эффективно применяется в решении задач совместной фильтрации (collaborative filtering) [62]. То есть алгоритм позволяет реализовать рекомендательную систему. В рамках такой системы информация о товарах или услугах отфильтровывается на основании спрогнозированного мнения пользователя о ней. Совместная фильтрация подразумевает, что пользователь относится к некоторой типичной группе пользователей, а прогноз вычисляется с учетом большого количества мнений пользователей.

2.12. Композиции алгоритмов машинного обучения. Бустинг

Представим ситуацию, что мы имеем несколько простых алгоритмов классификации, дающих результат лишь немного лучше случайного выбора. Оказывается, что, используя группу из нескольких таких алгоритмов, можно получить хороший результат, строя итоговый алгоритм так, чтобы каждый простой алгоритм, включаемый в группу, компенсировал недостатки предыдущего.

Суть градиентного бустинга, введенного в [63], заключается в том, что после расчета оптимальных значений коэффициентов регрессии и получения функции гипотезы $h_\theta(x)$ с помощью некоторого алгоритма (a) рассчитывается ошибка и подбирается, возможно, с помощью другого алгоритма (b) новая функция $h_{b\theta}(x)$ так, чтобы она минимизировала ошибку предыдущего:

$$h_\theta(x^{(i)}) + h_{b\theta}(x^{(i)}) - y^{(i)} \to min.$$

Иными словами, речь идет о минимизации функции стоимости вида:

---

[62] **Коллаборативная_фильтрация. – ru.wikipedia.org/wiki/Коллаборативная_фильтрация;** https://en.wikipedia.org/wiki/Collaborative_search_engine

[63] Friedman, Jerome H. Greedy function approximation: a gradient boosting machine // Annals of Statistics. – 2001. – P. 1189–1232.

$$J_b = \sum_{i=1}^{m} L(y^{(i)}, h_\theta(x^{(i)}) + h_{b\theta}(x^{(i)})),$$

где L – функция ошибки, учитывающая результаты работы алгоритмов a и b. Для нахождения минимума функции $J_b(\theta)$ используется значение градиента функции следующим образом.

Пусть мы имеем некоторую функцию ошибки:

$$L(y^{(i)}, h_\theta(x^{(i)}))_{i=1}^{m}.$$

Учитывая, что минимизация функции $J_b(h_{b\theta}(x^{(i)}))_{i=1}^{m}$ достигается в направлении антиградиента функции ошибки, алгоритм (b) настраивается так, что целевым значением является не $(y^{(i)})_{i=1}^{m}$, а антиградиент $(-L'(y^{(i)}, h_\theta(x^{(i)}))_{i=1}^{m})$, то есть при обучении алгоритма (b) вместо пар $(x^{(i)}, y^{(i)})$ используются пары $(x^{(i)}, -L'(y^{(i)}, h_\theta(x^{(i)}))$. Если $J_b(\theta)$ все еще велико, подбирается третий алгоритм (c) и т.д.

При этом, как указывается в [64], «во многих экспериментах наблюдалось практически неограниченное уменьшение частоты ошибок на независимой тестовой выборке по мере наращивания композиции. Более того, качество на тестовой выборке часто продолжало улучшаться даже после достижения безошибочного распознавания всей обучающей выборки. Это перевернуло существовавшие долгое время представления о том, что для повышения обобщающей способности необходимо ограничивать сложность алгоритмов. На примере бустинга стало понятно, что хорошим качеством могут обладать сколь угодно сложные композиции, если их правильно настраивать».

При решении задач классификации наиболее эффективным считается бустинг над деревьями решений. Одной из самых популярных библиотек, реализующих бустинг над деревьями решений, является XGBoost (Extreme Gradient Boosting). Загрузка библиотеки и создание классификатора выполняются командами:

```
import xgboost
clf = xgboost.XGBClassifier(nthread=1)
```

Применим XGBClassifier для решения задачи Fashion-MNIST:

```
clf = xgboost.XGBClassifier(nthread=4,scale_pos_weight=1)
clf.fit(X_train, y_train)
```

nthread – количество потоков, которое рекомендуется устанавливать не по количеству процессорных ядер вычислительной системы.

Результат, который получен в этом случае:

[64] Бустинг. – http://www.machinelearning.ru/wiki/index.php?title=Бустинг

```
Accuracy of XGBClassifier on training set: 0.88
Accuracy of XGBClassifier on test set: 0.86
```

Важной особенностью является нечувствительность к нормировке данных. То есть если мы будем рассматривать исходные данные изображения в их первозданном виде, исключив операторы:

```
##X_train1=X_train1/255.0
##X_test1=X_test1/255.0
```

Мы получим те же самые показатели качества, что и для нормированных данных.

> *Примечание*. При проведении экспериментов с большим набором данных нужно учесть, что алгоритм довольно долго обучается. В частности, при решении задачи Fashion-MNIST время обучения превышает 10 минут. Программу, решающую задачу Fashion-MNIST с помощью XGBoost (MLF_XGBoost_Fashion_MNIST_001), можно загрузить по ссылке https://www.dropbox.com/s/frb01qt3slqkl6q/MLF_XGBoost_Fashion_MNIST_001.html?dl=0

2.13. Снижение размерности данных. Метод главных компонент

Метод главных компонент (Principal Component Analysis – PCA) – один из «классических» способов уменьшения размерности данных, причем таким образом, чтобы минимизировать потери информации. С его помощью можно выяснить, какие из свойств объектов наиболее влиятельны в процессе принятия классификации. Однако он вполне успешно применяется для сжатия данных и обработки изображений. В машинном обучении метод часто применяется как один из способов понижения размерности до двух или трех с целью отображения объектов классификации или регрессии в виде, понятном для человека, или для ускорения обучения путем «отбрасывания» тех свойств данных, которые менее существенны, то есть вносят меньший вклад в распределение данных. Метод восходит к работам Пирсона и Сильвестра [65, 66].

Суть метода заключается в том, что ведется поиск ортогональных проекций с наибольшим рассеянием (дисперсией), которые и называются главными компонентами. Другими словами, ведется поиск ортогональных проекций с наибольшими среднеквадратическими расстояниями между объектами. Для дальнейшего изложения нам потребуются два нестрогих определения.

Определение 1. В теории вероятностей и математической статистике мера линейной зависимости двух случайных величин называется ковариацией. Ковариационная матрица, определяющая такую зависимость, рассчитывается следующим образом:

$$S = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^{T}.$$

[65] Pearson K. On lines and planes of closest fit to systems of points in space // Philosophical Magazine. – 1901. – Vol. 2. – P. 559–572.

[66] Sylvester J. J. On the reduction of a bilinear quantic of the nth order to the form of a sum of n products by a double orthogonal substitution // Messenger of Mathematics. – 1889. – Vol. 19. – P. 42–46.

Иначе, учитывая, что X – матрица параметров размерностью m x n (m – количество случайных величин, n – количество параметров или измерений, их определяющих), мы можем записать:

$$S = (1/m) * X \cdot X^T.$$

Определение 2. Ненулевой вектор, который при умножении на некоторую квадратную матрицу превращается в самого же себя с числовым коэффициентом, называется собственным вектором матрицы. Другими словами, если задана квадратная матрица S, то ненулевой вектор *v* называется собственным вектором матрицы, если существует число *w* – такое, что:

$$S \cdot v = w \cdot v.$$

Число *w* называют собственным значением или собственным числом матрицы S. Алгоритм расчета главных компонент включает два этапа:

Рассчитывается ковариационная матрица S, которая по определению является квадратной матрицей размера n x n, где n – число свойств.

Рассчитывается матрица собственных векторов V размерностью n x n, состоящая из n собственных векторов матрицы, каждый из которых состоит из n компонентов.

Фактически мы получаем n ортогональных измерений, в которых распределены величины $x^{(i)}$.

Из образовавшихся n главных компонент выбирают первые k, которые обеспечивают минимальные потери данных, так, что теряются минимальные отклонения в данных (variation). Вообще говоря, это означает, что данные можно восстановить с ошибкой не меньшей, чем указанные потери.

Другими словами, можно сократить матрицу V, уменьшив тем самым число ортогональных проекций вектора x. Обозначим сокращенную матрицу Vreduced. Затем можно умножить сокращенную матрицу на транспонированную матрицу X:

Z= Vreduced*X.T.

Так мы получим новую матрицу Z, содержащую проекции X на сокращенный набор измерений. Тем самым часть измерений будет потеряна, размерность новой матрицы Z будет меньше X, однако при этом можно отбрасывать малозначимые проекции, вдоль которых значения $x^{(i)}$ меняются незначительно.

Рассмотрим простой пример преобразования двумерного набора данных в одномерный. На рисунке 2.15а слева показан синтетический набор данных, где каждая из 200 точек является объектом в пространстве двух признаков. Набор получен командой:

```
X = np.dot(np.random.random(size=(2, 2)), np.random.normal(size=(2, 200))).T
```

Рассчитаем ковариационную матрицу, собственное число и матрицу собственных векторов командами:

```
S=(1/X.shape[1])*np.dot(X.T,X) #covariance matrix
w, v = np.linalg.eigh(S)
```

Используя первый или второй вектор матрицы v, мы можем получить два набора взаимно ортогональных значений – z и zz:

```
vreduced=v[:,1]
vreduced1=v[:,0]
```

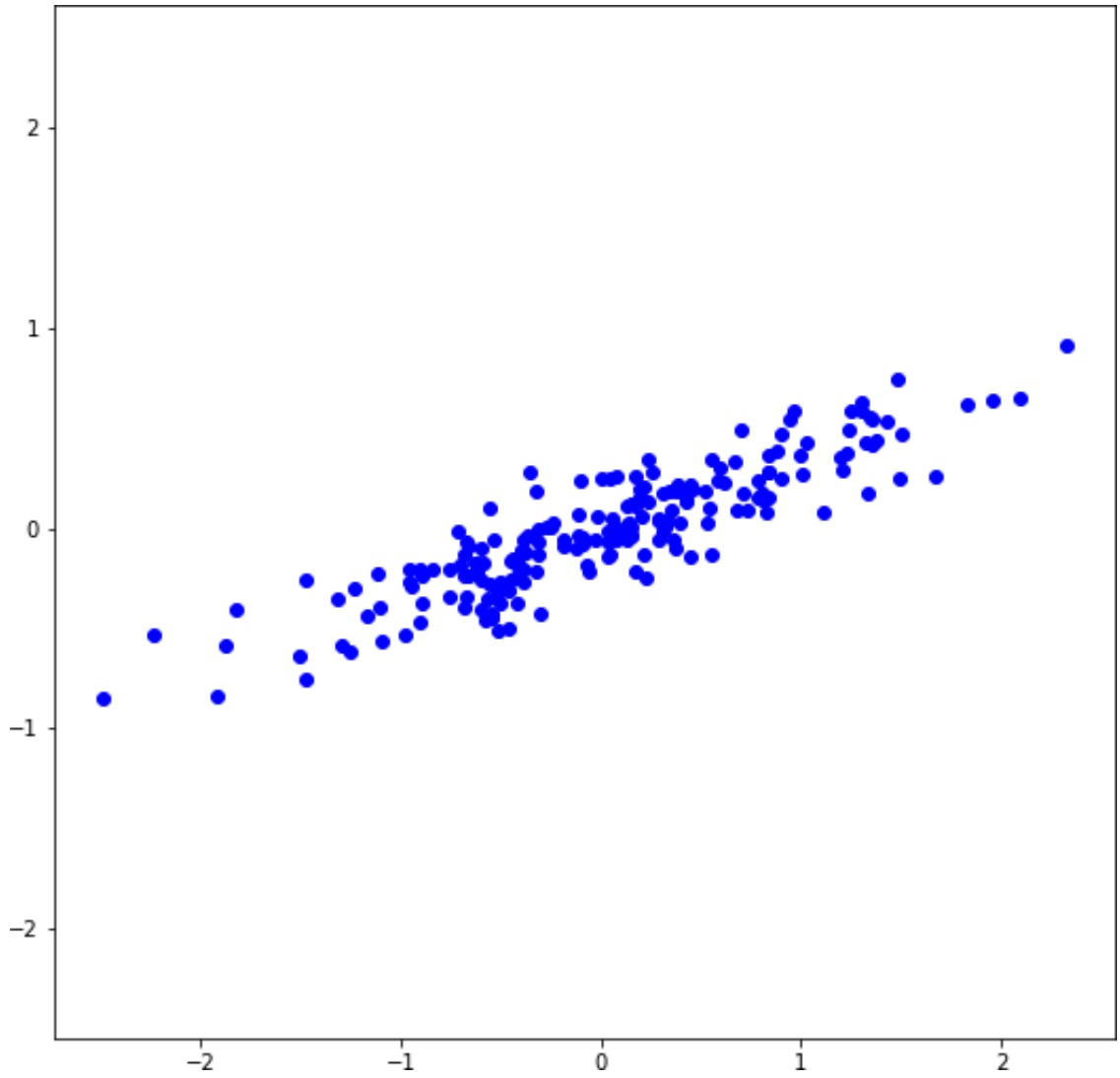
```
z=np.dot(vreduced,X.T)
zz=np.dot(vreduced1,X.T)
```

Видно, что дисперсия распределения объектов вдоль горизонтальной оси значительно больше, чем вдоль вертикальной (рисунок 2.15b). Фактически объекты, расположенные на горизонтальной и вертикальной осях, и являются одномерным представлением исходного набора. Видно, что, исключая вертикальную ось (рисунок 2.15b) полностью (вторая главная компонента), мы теряем относительно небольшое количество информации.

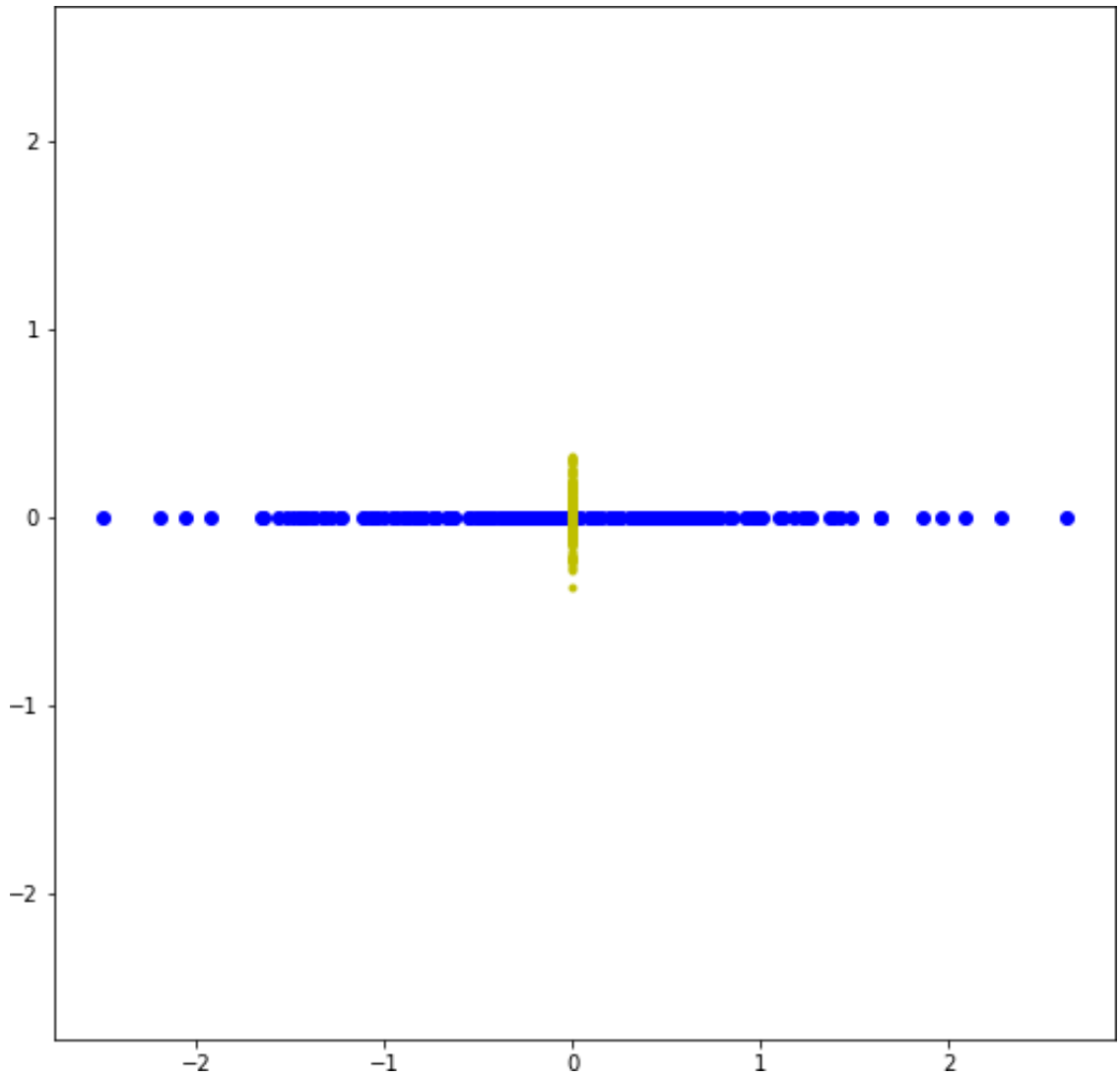
Заметим, что объекты можно вновь неточно восстановить в пространстве двух признаков, выполнив обратное преобразование:

Xa= Vreduced*Z.

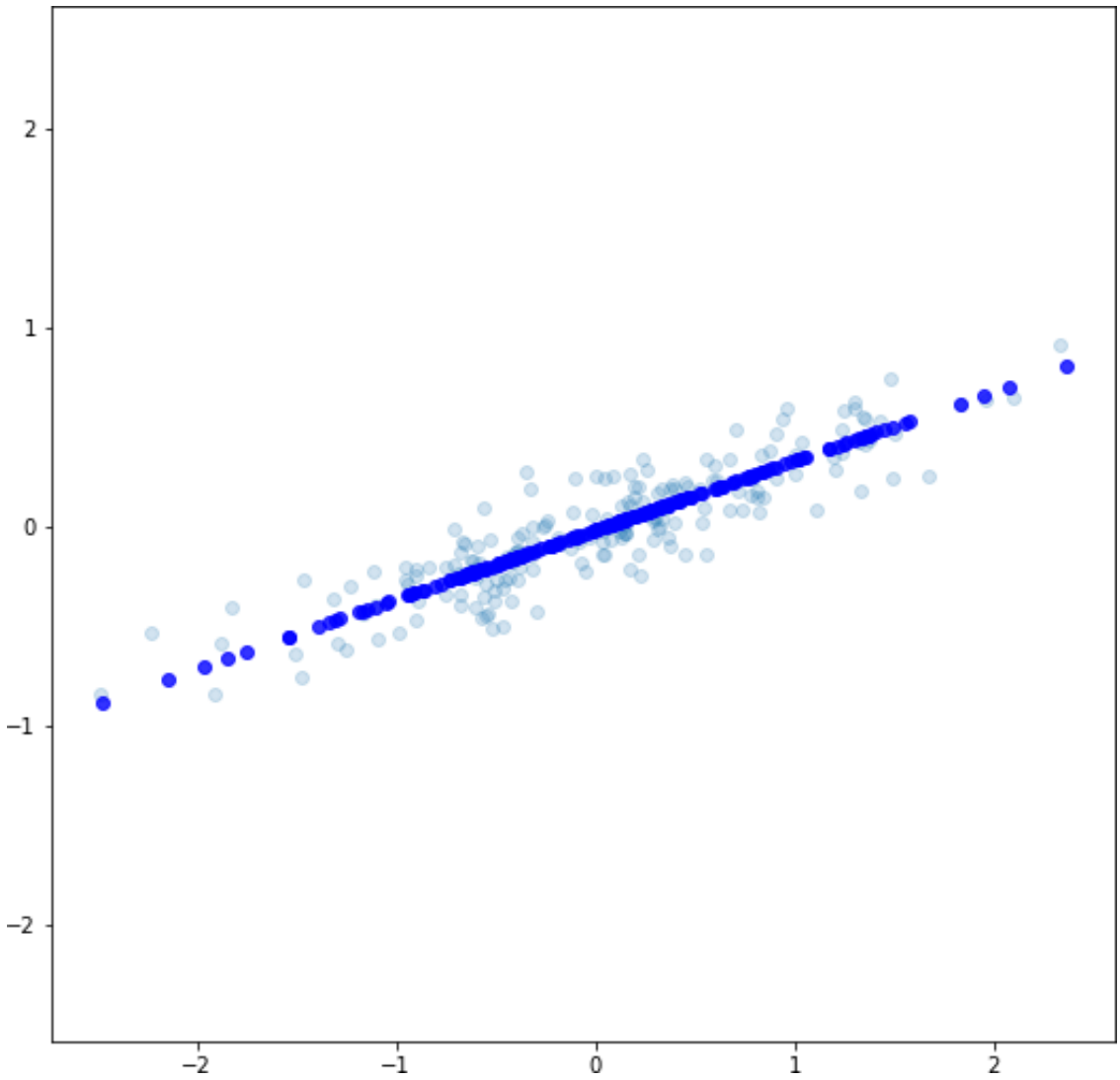
Однако информацию, относящуюся ко второй главной компоненте, мы, конечно, потеряем (рисунок 2.15c).



а) Исходный набор данных, где каждый объект имеет два свойства



b) Отображение объектов на взаимно перпендикулярные оси (первую и вторую главную компоненты)



c) Восстановление объектов в двумерном пространстве признаков. Исходное распределение объектов показано полупрозрачными точками

Рисунок 2.15. Преобразование данных при применении PCA

На первый взгляд (рисунок 2.15c) может показаться, что задача PCA является задачей линейной регрессии, однако это не совсем так. Отличие в том, что в задаче линейной регрессии среднеквадратическое расстояние определяется вдоль оси y (оси меток), а в PCA – перпендикулярно главной компоненте (рисунок 2.16).

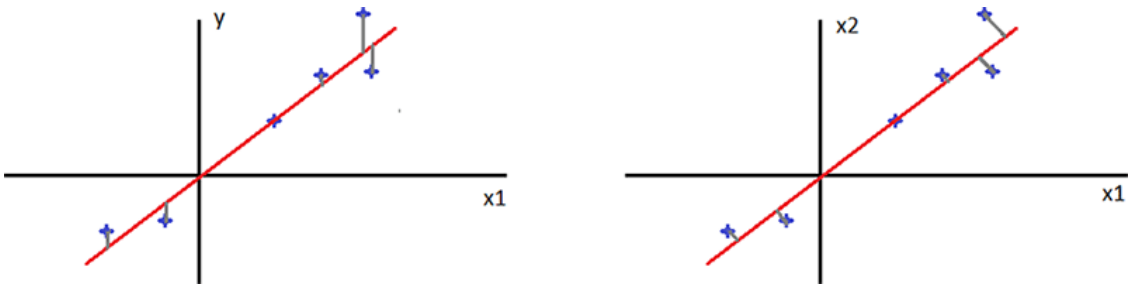


Рисунок 2.16. Представление задач линейной регрессии (слева) и PCA (справа)

Примечание. Полный текст программы расчета главных компонент приведен в MLF_PCA_numpy_001.ipynb – https://www.dropbox.com/s/65y1z7svf7epx1q/MLF_PCA_numpy_001.html?dl=0

Библиотека scikit-learn имеет в своем составе модуль PCA, с помощью которого можно вычислить главные компоненты и найти количество главных компонент, необходимых для обеспечения заданной вариативности новых параметров z.

Примечание. Закрепить навыки работы с PCA в составе библиотеки scikit-learn можно, выполнив задания лабораторной работы ML_lab08_Principal Component Analysis – https://www.dropbox.com/sh/xnjiztxoxpqwos3/AADoUPfNeMnEXapbqb3JHHvla?dl=0

2.14. Контрольные вопросы

Какие параметры регулируют работу алгоритма k-NN и позволяют улучшить качество классификации?

Что такое ядро в алгоритме опорных векторов?

Приведите выражение функции стоимости алгоритма опорных векторов.

Как обучается алгоритм Naïve Bayes?

Укажите достоинства алгоритма Naïve Bayes.

Укажите недостатки алгоритма Naïve Bayes.

Что дает сглаживание по Лапласу в алгоритме Naïve Bayes?

Чем помогает применение логарифмов в алгоритме Naïve Bayes?

Что такое бустинг?

В чем заключается преимущество бустинга над деревьями решений?

Что такое PCA?

Каково минимальное количество главных компонент, получаемых с помощью PCA?

3. Оценка качества методов ML

Для решения конкретной задачи с помощью ML необходимо выбрать соответствующий метод, который дает наилучший результат.

> *Примечание*. Под методом машинного обучения мы понимаем в данном случае реализацию алгоритма или некоторой модели вычислений, которая решает задачу классификации, регрессии или кластеризации.

Для выбора такого метода требуются некоторые показатели, позволяющие оценить методы ML и сравнить их между собой.

> *Примечание*. Программу, которая реализует большую часть примеров данного раздела, можно получить по ссылке – https://www.dropbox.com/s/nc1qx6tjw11t5gs/MLF_Evaluation001.ipynb?dl=0

При этом, как правило, на начальном этапе отбираются методы, удовлетворяющие ограничениям по вычислительной мощности, объему и характеристикам данных, которые есть в распоряжении специалиста по обработке данных. Например, методы глубокого обучения, решающие сложные задачи машинного обучения с высокой точностью, можно использовать, если в распоряжении исследователя имеются большие по объему данные и значительные вычислительные мощности. С другой стороны, если количество примеров меньше числа свойств, то затруднено применение машин опорных векторов (SVM), поскольку они подвержены в таком случае переобучению. Таким образом, отобрав некоторое множество методов для решения задачи и изменяя их параметры (например, коэффициент регуляризации, число слоев нейронных сетей и т.п.), необходимо оценивать результаты их работы, используя один или несколько показателей.

> *Примечание*. Рекомендуется выбрать одну, возможно, интегральную метрику для оценки качества.

К числу таких показателей можно отнести метрики качества, кривые оценки качества, способность к обучению и скорость обучения и решения задачи.

В общем случае метрики оценки качества зависят от предметной области и цели, поставленной перед системой ML, и могут задаваться исследователем. Например, для поисковых машин, выполняющих поиск информации в интернете, это может быть удовлетворенность пользователей (user satisfaction) в результатах поиска, для систем электронной коммерции – доход (amount of revenue), для медицинских систем – выживаемость пациентов (patient survival rates) и т.п. Однако есть некоторый базовый набор метрик, которые применяются достаточно часто при оценке качества алгоритмов классификации, регрессии и кластеризации.

Назначение метрик качества – дать оценку, показывающую, насколько классификация или предсказание, выполненное с применением методов ML, отличается от таковой, выполненной экспертами или другим алгоритмом. При этом часто применяют простейшую метрику – процент (доля) правильно классифицированных примеров. Для оценки ошибок первого и второго рода применяют также еще несколько важных показателей: «точность» (precision), «полноту» (recall), и обобщающие показатели – меры F1 и F (F1 score и F-score).

> *Примечание*. Напомним, что ошибкой первого рода называется ошибка, состоящая в опровержении верной гипотезы, а ошибкой второго рода называется ошибка, состоящая в принятии ложной гипотезы.

Их применение особенно важно в случае неравных по объему классов, когда количество объектов одного типа значительно превосходит количество объектов другого типа. Часто упоминаемый перечень метрик оценки классификаторов, следующий:

Accuracy
Precision
Recall
F1 score
F-score
Area Under the Curve (AUC)

Кроме этого, на практике часто применяются специальные кривые:
1. Precision-Recall curve
2. ROC curve

Кроме метрик оценки качества важным показателем применяемого метода ML является его способность обучаться, то есть улучшать свои показатели точности при увеличении числа примеров. Может оказаться, что метод, который показывает очень хорошие результаты на тренировочном множестве примеров, дает неудовлетворительный результат на тестовом множестве, то есть не обладает нужной степенью обобщения. Баланс между способностью обобщения и точностью может быть найден с помощью «кривых обучаемости», которые в общем случае могут показать, способен ли тот или иной метод улучшать свой результат так, чтобы показатели качества как на тренировочном, так и на тестовом множестве были примерно равны и удовлетворяли требованиям предметной области исследования.

Третий показатель, который становится особенно важным в задачах с большим объемом данных, – скорость обучения и классификации. Методы ускорения работы алгоритмов ML в задачах с большими данными рассматриваются в разделе «Машинное обучение в задачах с большим объемом данных».

3.1. Метрики оценки качества классификации

В настоящее время в задачах машинного обучения для оценки качества классификации наиболее часто используется доля правильных ответов (accuracy) или Correct Classification Rate (CCR) – относительное количество корректно классифицированных объектов (процент или доля правильно классифицированных объектов):

$$Ac = \frac{N_t}{N},$$

где N_t – количество корректно классифицированных объектов; N – общее число объектов.

Этот показатель является весьма важным, однако если количество объектов в классах существенно неравное (так называемые неравномерные, или «перекошенные», классы – skewed classes), то может случиться так, что очень плохой классификатор будет давать большое значение Ac. Например, если объектов 1-го типа 90% от всего числа объектов, а объектов 2-го типа только 10%, то классификатору достаточно отвечать всегда, что он распознал объект 1-го типа, и доля правильных ответов достигнет 90%. Таким образом, даже если алгоритм никогда правильно не распознает объект 2-го класса, он все равно будет иметь высокий показатель Ac.

При этом, если распознавание объектов 2-го класса исключительно важно, показатель Ac будет попросту вводить в заблуждение. Для того чтобы избежать подобной неадекватной оценки, рассматривается еще несколько важных показателей: «точность» (precision), «полнота» (recall), и обобщающий показатель – F1 score (гармоническое среднее или мера F1), которые рассчитываются с помощью следующих выражений:

$$\text{Precision: } P = \frac{TP}{(TP+FP)}.$$

$$\text{Recall: } \quad R = \frac{TP}{(TP+FN)}.$$

$$\text{F1 score: } F1\ score = \frac{2*P*R}{(P+R)}.$$

Поясним приведенные выражения.

Рассмотрим случай классификации двух классов (или одного класса номер 1 (positive) и всех остальных классов, которым присвоим номер 0 (negative)). В этом случае возможны следующие ситуации:

		Реальный класс (Actual class)	
		1	0
Предсказанный класс (Predicted class)	1	True positive (TP)	False positive (FP)
	0	False negative (FN)	True negative (TN)

Случаи True positive (TP) и True negative (TN) являются случаями правильной работы классификатора, т.е. предсказанный класс совпал с реальностью. Соответственно, False negative (FN) и False positive (FP) – случаи неправильной работы. FN или ошибка первого рода возникает тогда, когда объект классификации ошибочно отнесен к негативному классу, являясь на самом деле позитивным. Эту ошибку можно рассматривать как признак излишне пессимистического (осторожного) классификатора, т.е. ML-модель предсказала отрицательный результат, когда он является на самом деле положительным. FP или ошибка второго рода, наоборот, признак излишне оптимистического, или неосторожного, классификатора, то есть ML-модель предсказала положительный результат, когда он является на самом деле отрицательным.

Precision (P) будет показывать часть правильно распознанных объектов заданного класса по отношению к общему числу объектов, принятых классификатором за объекты заданного класса. С другой стороны, Recall (R) будет показывать отношение правильно распознанных объектов к общему числу объектов данного класса.

Оба показателя – и P, и R – отражают «путаницу» классификатора. Однако R показывает, насколько классификатор оптимистичен в своих оценках или как часто он «любит» (высокое значение R) присоединять объекты другого класса (negative) к заданному, в то время как P показывает, насколько классификатор «строг» в своих оценках, насколько часто он «отбрасывает» (высокое значение P) объекты нужного (positive) класса. Разумеется, желательно, чтобы оба этих показателя стремились к 1, однако, как правило, в сложных случаях классификации результаты работы балансируют между значениями P и R, то есть большое значение P характерно при малом значении R, и наоборот. На рисунках 3.1a и 3.1b приведены примеры

двух линейных классификаторов с высокими значениями precision и recall, где положительные объекты показаны черными точками, отрицательные – желтыми, а граница между классами – красной прямой.

Конец ознакомительного фрагмента.