Bernhard G. Humm

Applied Artificial Intelligence

An Engineering Approach

Second Edition

Applied Artificial Intelligence

An Engineering Approach

Bernhard G. Humm

This book is for sale at http://leanpub.com/AAI

This version was published on 2020-05-28

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

Preface

Why yet another book on Artificial Intelligence?

It is true that hundreds of publications on Artificial Intelligence (AI) have been published within the last decades - scientific papers and text books. Most of them focus on the theory behind AI solutions: logic, reasoning, statistical foundations, etc. However, little can be found on engineering AI applications.

Modern, complex IT applications are not built from scratch but by integrating off-the-shelf components: libraries, frameworks, and services. The same applies, of course, for AI applications. Over the last decades, numerous off-the-shelf components for AI base functionality such as logic, reasoning, and statistics have been implemented - commercial and open source. Integrating such components into user friendly, high-performance, and maintainable AI applications requires specific engineering skills. This book focuses on those skills.

My own professional background is that of a software engineer. After under-/postgraduate studies of Computer Science in Germany and a Ph.D. program in Australia, I worked for more than a decade in a large German software company. In extensive teams, we developed custom software for clients: multinational banks, credit-card issuers, tour operators, telecommunication providers, fashion companies and ATC authorities. My tasks were as diverse as the industry sectors and technologies used. They ranged from development, software architecture, project management to managing a division and running the company's research department.

After re-entering university as a professor 15 years ago, a common theme of my courses has been the professional development of software according to engineering principles and practices. AI was an old love of mine but my industry projects hat little relation to AI. Noticing that AI applications - powerful image processing, speech analysis and generation etc. - were rapidly entering the market attracted my interest, again. Via R&D projects, funded by industry and the public, I gradually built up expertise in engineering AI applications. In teams with colleagues, postgraduate students and industry partners we developed applications for a hotel portal, a library, an art museum, the cancer department of a hospital, the borderline unit of a psycho-therapeutic clinic, and a robot manufacturer. However diverse the industry sectors, many approaches and technologies can be used across the projects in order to build maintainable AI applications with a good user experience that meet functional and non-functional requirements, particularly high performance requirements. Overall, we are combining general software engineering skills with AI expertise.

When I perceived a growing demand for AI applications in the business and consumer markets in 2014, I started a new M.Sc. course "Applied Artificial Intelligence" at Darmstadt University of Applied Sciences. This book reflects topics of this course. I am constantly learning: from project experience, from my colleagues and partners, from my students, and hopefully also from you, the readers of this book. So please, don't hesitate to contact me under bernhard.humm@h-da.de when you agree or disagree with my findings.

To support you to come to grips with the topics I have added questions and exercises in all chapters. They are indicated by a pencil symbol.

Finally, I would like to thank my friend and project partner Prof Dr Paul Walsh from NSilico Lifescience Ltd., Ireland, for valuable comments and hints.

Bernhard Humm, Darmstadt, Germany, 2016 and 2020

Copyright Notice

All photos in figures of this book are published under the Creative Commons (CC) License.

- Cover image: CC Wikimedia Commons, Author Meritxell.canela, Bag_of_words.JPG[1]
- Fig. 1.2: CC Wikimedia Commons, Author Artaxerxes, 28 June 2008, Our_Community_Place_-Sandbox.jpg[2]
- Fig. 1.4: CC-BY Wikimedia Commons, Author James the photographer, 14 June 2007, Deep_-Blue.jpg[3]
- Fig. 1.5: CC Wikimedia Commons, Author Raysonho @ Open Grid Scheduler / Grid Engine, 7 April 2011, IBMWatson.jpg[4]
- Fig. 1.6: CC Wikimedia Commons, Author LG Electronics, 26 May 2016, Lee_Se-Dol.jpg[5]
- Fig. 1.6: CC Wikimedia Commons, Author Wesalius, 10 Sept 2018, Lee_Sedol_vs_AlphaGo_-Game_4.jpg[6]
- Fig. 2.6: CC Wikimedia Commons, Author Stephen Milborrow, 1 March 2011, CART_tree_-titanic_survivors.png[7]
- Fig. 2.11: CC Wikimedia Commons, Author Sven Behnke, CC BY-SA 4.0, Deep Learning.jpg[8]
- Fig. 2.20: CC Wikimedia Commons, Author Walber, 22 November 2014, Precisionrecall.svg[9]
- Fig. 3.2: Wikimedia Commons, Painter Michelangelo Buonarroti (1457-1564) {PD-Italy}{PD-Canada}, File:The_Creation_of_Adam.jpg[10]
- Fig. 3.10: CC Wikimedia Commons, Author Max Schmachtenberg, Source lod-cloud.net[11], Lod-cloud_colored_1000px.png[12]
- Fig. 7.3: CC Wikimedia Commons, Author Mbroemme5783, 4 October 2012, Face_Capture.PNG[13]
- Fig. 7.4: CC Wikimedia Commons, Author R Walters, 22 July 2007, Restoration.jpg[14]
- Fig. 7.5: CC Wikimedia Commons, Author OpenStax College, Source cnx.org[15], 5 April 2013, 113abcd_Medical_Imaging_Techniques.jpg[16]

[1] https://commons.wikimedia.org/wiki/File:Bag_of_words.JPG
[2] https://commons.wikimedia.org/wiki/File:Our_Community_Place_Sandbox.jpg
[3] https://commons.wikimedia.org/wiki/File:Deep_Blue.jpg
[4] https://commons.wikimedia.org/wiki/File:IBMWatson.jpg
[5] https://commons.wikimedia.org/wiki/File:Lee_Se-Dol.jpg
[6] https://commons.wikimedia.org/wiki/File:Lee_Sedol_B_vs_AlphaGo_W_-_Game_4.jpg
[7] https://commons.wikimedia.org/wiki/File:CART_tree_titanic_survivors.png
[8] https://commons.wikimedia.org/w/index.php?curid=82466022
[9] https://commons.wikimedia.org/wiki/File:Precisionrecall.svg
[10] https://commons.wikimedia.org/wiki/File:The_Creation_of_Adam.jpg
[11] http://lod-cloud.net/
[12] https://commons.wikimedia.org/wiki/File:Lod-cloud_colored_1000px.png
[13] https://commons.wikimedia.org/wiki/File:Face_Capture.PNG
[14] https://commons.wikimedia.org/wiki/File:Restoration.jpg
[15] http://cnx.org/content/col11496/1.6/
[16] https://commons.wikimedia.org/wiki/File:113abcd_Medical_Imaging_Techniques.jpg

- Fig. 7.6: Wikimedia Commons, Author KUKA Roboter GmbH, Bachmann; Source KUKA Roboter GmbH, Zugspitzstr. 140, D-86165 Augsburg, Germany, Dep. Marketing, www.kuka-robotics.com; 2003, KUKA_robot_for_flat_glas_handling.jpg[17]
- Fig. 7.7: CC Wikimedia Commons, Author Robert Basic, Source Audi A8 2013 Uploaded by AVIA BavARia, 17 October 2013, Audi_A8_2013_(11209949525).jpg[18]
- Fig. 7.8: Wikimedia Commons, Author NASA/JPL/Cornell University, Maas Digital LLC, Source photojournal.jpl.nasa.gov[19], February 2003, NASA_Mars_Rover.jpg[20]
- Fig. 7.9: CC Wikimedia Commons, Author Vazquez88, 5 February 2013, Motion_Capture_-with_Chad_Phantom.png[21]

[17] https://commons.wikimedia.org/wiki/File:KUKA_robot_for_flat_glas_handling.jpg
[18] https://commons.wikimedia.org/wiki/File:Audi_A8_2013.jpg
[19] http://photojournal.jpl.nasa.gov/catalog/PIA04413
[20] https://commons.wikimedia.org/wiki/File:NASA_Mars_Rover.jpg
[21] https://commons.wikimedia.org/wiki/File:Motion_Capture_with_Chad_Phantom.png

1. Introduction

How relevant is *Artificial Intelligence*?

When I wrote the first edition of this book in 2015, Artificial Intelligence (AI) was hardly noticeably in public. For many people, AI was nothing more than a burst bubble of a 20th century hype. However, even then AI was *relevant and ubiquitous* in IT applications of business and consumer markets.

A few examples of AI in everyday use were and still are:

- Speech control for smart-phones, navigation systems in cars, etc.
- Face recognition in cameras
- Learning spam filters in e-mail clients
- AI in computer games
- Semantic Internet search, including question answering; See Fig. 1.1.

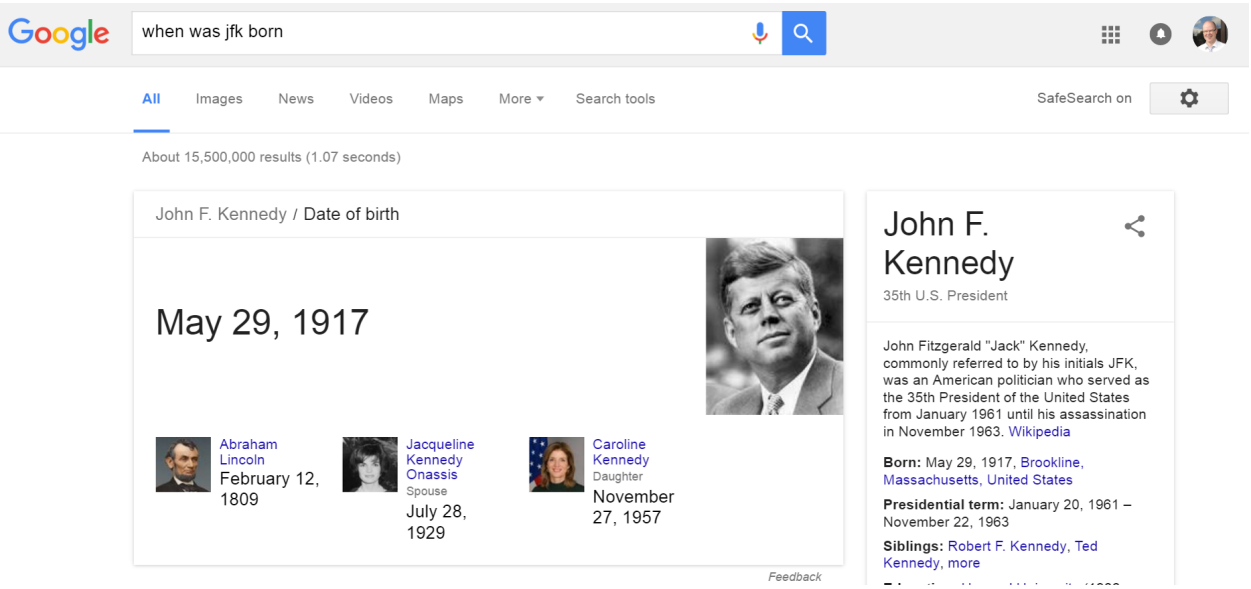


Fig. 1.1: AI in everyday use: Google's question answering "When was JFK born?"

Commercial examples are:

- Business intelligence
- Sentiment analysis
- Robotics

- Industrial computer vision
- Self-driving cars, drones, rockets (military and commercial)

The public perception of AI has changed drastically in the last few years. AI has re-gained an enormous attention in public debate - like in the 1980s. Nobody questions the importance of AI any more; instead, many people raise exaggerated, unrealistic claims and fears - which is common in hype topics.

I expect AI to continue to change our daily lives and labor markets in an enormous way, just like technology has changed the daily lives of generations since the 19th century with ever increasing speed.

How are AI applications developed?

While most AI publications, such as scientific papers and text books, focus on the theory behind AI solutions, little can be found on engineering AI applications. What kinds of AI libraries, frameworks and services already exist? Which ones should be chosen in which situation? How to integrate them into maintainable AI applications with a good user experience? How to meet functional and non-functional requirements, in particular high performance?

The focus of this book is to answer those kinds of questions for software developers and architects.

1.1 Overview of this Book

The remainder of this chapter presents a definition of AI as well as a brief AI history. Chapters 2 and 3 present the main AI approaches: machine learning (non-symbolic) and knowledge representation (symbolic). Chapter 4 gives guidelines for the architecture of AI applications. The remaining chapters focus on individual AI areas: (5) information retrieval, (6) natural language processing, and (7) computer vision. Chapter 8 concludes the book.

Inspired by one of my AI projects, I use application examples from the domain of art museums throughout the book. Additionally, I present real-world examples from other domains.

At the end of each chapter I repeat the main themes in form of questions that you, the reader, can use as a *quick check*.

The appendix contains product tables as well as source code examples that can be used as starting point for your own developments.

1.2 What is AI?

Encyclopaedia Britannica[1] defines AI as follows.

[1] https://www.britannica.com/technology/artificial-intelligence

"Artificial intelligence (AI), the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings."

Note that this definition does not claim or even postulate that AI applications *are* intelligent nor that AI is comparable ore even equivalent to human intelligence.

What are tasks commonly associated with intelligent beings?

- **Perceiving**: Seeing, listening, sensing, etc.
- **Learning, knowing, reasoning**: Thinking, understanding, planning etc.
- **Communicating**: Speaking, writing, etc.
- **Acting**

See Fig. 1.2.

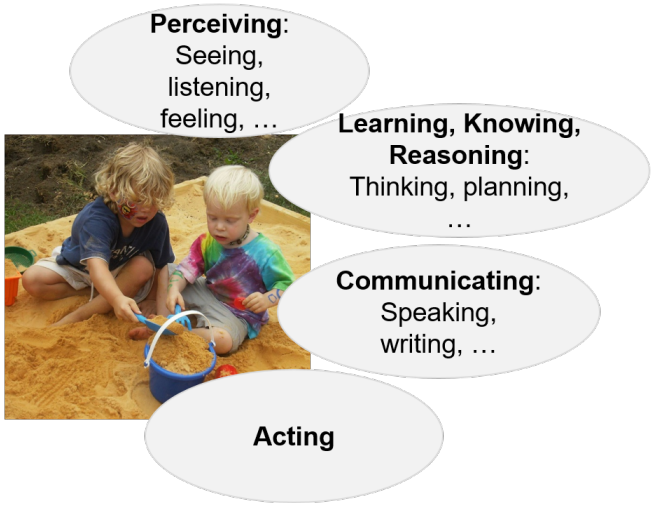


Fig. 1.2: Tasks commonly associated with intelligent beings

The different areas of AI can be structured according to those behaviors. See the circles in the "landscape of AI" (inspired by an AI Spektrum poster[2]) in Fig. 1.3.

[2] https://www.sigs-datacom.de/order/poster/Landkarte-KI-Poster-AI-2018.php

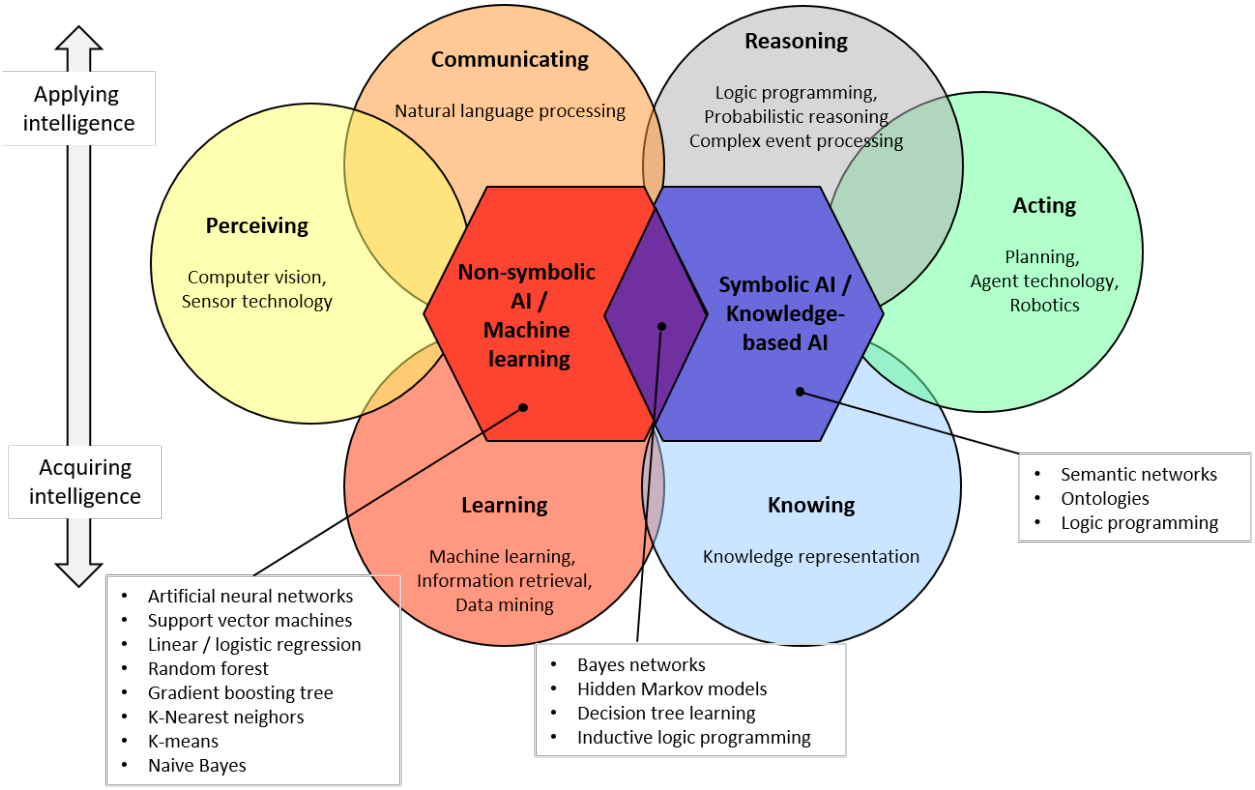


Fig. 1.3: The AI landscape

- **Perceiving** covers the AI areas of *computer vision* and *sensor technology*;
- **Communicating** covers the area of *natural language processing (NLP)*;
- **Learning** covers *machine learning (ML), information retrieval (IR)* and *data mining*;
- **Knowing** covers *knowledge representation*;
- **Reasoning** covers *logic programming, probabilistic reasoning*, and *complex event processing (CEP)*;
- **Acting** covers planning, agent technology, and robotics.

Please note, that various different namings and groupings of AI areas can be found in literature.

Two fundamentally different approaches to AI can be distinguished (depicted as hexagons in Fig. 1.3): - **Symbolic AI / knowledge based AI**: Knowledge is represented explicitly (with symbols) in a human-readable way, e.g., with semantic networks, ontologies, or logic programming languages (see the boxes with technologies attached to the approaches in Fig. 1.3). - **Non-symbolic AI / machine learning**: Knowledge is implicit in form of numbers, e.g., as weights in artificial neural networks, support vector machines, in linear / logistic regression etc.

Both approaches, symbolic and non-symbolic, have been around from the various beginnings of AI in the 1950s. In the first decades of AI research and practice, symbolic approaches were most prominent and showed the most convincing results. However, in the last century this has shifted and today, non-symbolic approaches - particularly machine learning - are most prominent.

Both approaches have advantages and disadvantages. Non-symbolic approaches require little upfront knowledge, just (many) samples for training. They exhibit good behavior, e.g., in classifying situations, also with noisy data. However, the reasoning behind decisions cannot be explained to humans. In contrast, the reasoning behind symbolic approaches is explicit and can be retraced by humans. However, explicit knowledge engineering is required upfront and reasoning under uncertainty is challenging.

Both approaches have been applied in all areas of AI, but symbolic approaches are commonly used for knowing, reasoning, and acting whereas non-symbolic approaches are commonly used perceiving, communicating, and learning.

I expect both approaches to stay in the future. Hybrid approaches combine the advantages of symbolic and non-symbolic AI; I expect them to gain increasing importance. Hybrid approaches include Bayes networks, hidden Markov models, decision tree learning etc.

1.3 A Brief History of AI

The term "Artificial Intelligence" was coined at the *Dartmouth Workshop* in 1956. Members of this workshop were John McCarthy, Marvin Minsky, Claude Shannon, Allen Newell, Herbert Simon, and others. However, Alan Turing (1912 – 1954) with his fundamental work on computability and the so-called *Turing Test* for assessing intelligent behavior had already laid ground for AI decades before.

The 1960s - 1980s saw unprecedented *AI hype*, triggered by enthusiastic promises of quick AI results. Examples of such promises are:

"Within a generation the problem of creating 'artificial intelligence' will be substantially solved." (Marvin Minsky, 1967)

"Within 10 years computers won't even keep us as pets." (Marvin Minsky, 1970).

or

"Machines will be capable of doing any work that a man can do." (Herbert Simon, 1985) (Citations from (American Heritage, 2001)).

This hype resulted in massive funding of AI projects, particularly in the US.

The effect of those wildly exaggerated promises was the same as in all hype. When people started to notice that the most sophisticated AI applications failed to perform tasks that are easily accomplished by small children they threw the baby out with the bathwater. The disillusionment of the unrealizable expectations resulted in massive cuts in funding and a collapse of the AI market. The 1980s - 1990s have sometimes been called the *AI winter*.

From then on, unnoticed and often not under the term AI, AI methods and technologies have matured enormously driven by major technology companies. For example, Google co-founder Larry Page[3] said in 2006:

[3] http://bigdata-madesimple.com/12-famous-quotes-on-artificial-intelligence-by-google-founders

"We want to create the ultimate search engine that can understand anything. Some people could call that artificial intelligence" .

This development of AI applications by major technology drivers lead to the situation today where AI is relevant and ubiquitous in everyday applications. Moreover, AI has become a hype topic, again. This is promoted by continuous media attention, science fiction movies, and bold statements by individuals in the AI community that are remarkably similar to the ones in the 1970s:

"Artificial intelligence will reach human levels by around 2029. Follow that out further to, say, 2045, we will have multiplied the intelligence, the human biological machine intelligence of our civilization a billion-fold." (Ray Kurzweil, 2005[4])

I personally cannot see the slightest evidence for such bold claims and regard them as pure science fiction. Such exaggerated claims create much media attention but may eventually lead to another AI winter. A most profound response to such claims has been written by Rodney Brooks in 2017[5] which I recommend reading.

1.4 Impacts of AI on Society

Software applications and in particular AI applications may have massive impacts on society. I believe that software engineers must be aware of those implications and act responsibly.

Automation technology has always had enormous impact on the labor market. We can see this development particularly since the the industrial revolution in the 19th century. With the advent of information technology in the 20th century, the speed of innovation has accelerated. AI technology is a continuation of this highly dynamic process, with ever increasing innovation speed. Like in earlier technology shifts, groups of jobs will become obsolete and new jobs will emerge.

There are numerous predictions about the implications on AI on the labor market. Some predict a replacement rate of human labor by robots for certain business sectors of up to 99%[6]. Some predict that fears of massive displacement of workers are unfounded in the longer term[7].

To quote Mark Twain (or Niels Bohr maybe others), "It's Difficult to Make Predictions, Especially About the Future"[8] ;-). I personally assume that the ever increasing automation technology will, in fact, decrease the need for human labor since we cannot (and should not) increase consumption forever. In this case it will be necessary to come to a societal agreement of how to distribute the wealth generated by machines. Some discuss a basic income[9] as one model.

Also, as a society, we must come to an agreement, which decisions we may leave to machines and which ones we must not. Alan Bundy pointedly and correctly states in an CACM viewpoint[10]: "Smart

[4] https://www.brainyquote.com/quotes/ray_kurzweil_591137
[5] http://rodneybrooks.com/the-seven-deadly-sins-of-predicting-the-future-of-ai/
[6] https://futurism.com/will-artificial-intelligence-take-job-experts-weigh
[7] https://ec.europa.eu/epsc/sites/epsc/files/ai-report_online-version.pdf
[8] https://quoteinvestigator.com/2013/10/20/no-predict/
[9] https://en.wikipedia.org/wiki/Basic_income
[10] https://cacm.acm.org/magazines/2017/2/212436-smart-machines-are-not-a-threat-to-humanity/

machines are not a threat to humanity. Worrying about machines that are too smart distracts us from the real and present threat from machines that are too dumb".

More and more decisions are delegated to machines although, of course, no piece of software is perfect. This may be fine for self-driving cars, at least as soon as they drive safer than average humans (which I expect in the foreseeable future). But it may be dangerous in areas where decisions by machines may have far-reaching impacts that no human can foresee. One of those areas may be high-frequency trading where machines autonomously make selling and buying decisions, but the impact on the global financial market is unclear. Even more critical are autonomous weapons that are programmed to detect alleged attacks automatically and to start a counter-strike without human intervention.
An open letter from AI and robotics researchers on autonomous weapons[11] in which this issue is discussed was signed by more than 30,000 people.

Computer scientists developing AI applications must be aware of their impact and act responsibly. Current keywords in the debate are responsible AI, ethical AI, human-centered AI or explainable AI. We will cover aspects of this in the book.

1.5 Prominent AI Projects

Some AI projects have become milestones in AI history due to their public visibility. I shortly mention just three examples.

In 1997, *IBM Deep Blue* beat the chess world champion Garry Kasparov. This was a milestone since chess is considered one of the most complex board games.

See Fig. 1.4.

[11] https://futureoflife.org/open-letter-autonomous-weapons/

Fig. 1.4: IBM Deep Blue beats the chess champion Garry Kasparov

However, when people examined the computational algorithms used in the Deep Blue Application, people quickly realized the difference between the IBM approach and the way human chess players act intelligently. For people who believed in machines being intelligent like humans, this was a disillusionment. For practitioners this was no disillusionment at all but an important milestone in the development of applications which exhibit behavior of human intelligence.

In 2011, IBM succeeded with another important AI project: *IBM Watson*. While Deep Blue targeted a most specific ability, namely playing chess, IBM Watson was able to answer natural language questions about general knowledge. The media highlight of this project was beating the human champions at the popular US quiz show Jeopardy! This was remarkable since not only questions about history and current events, the sciences, the arts, popular culture, literature, and languages needed to be answered but also play on words as well as execution speed and strategy needed to be considered.

See Fig. 1.5.

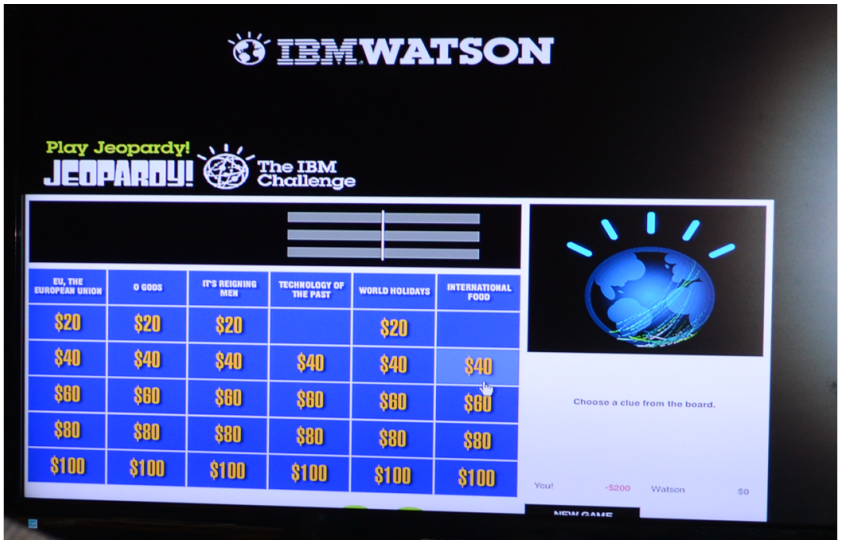


Fig. 1.5: IBM Watson for the Jeopardy! quiz show

Go is a complex board game that requires intuition, creative and strategic thinking. It has long been considered a difficult challenge in AI and is considerably more difficult to solve than chess. In 2015, AlphaGo, a computer Go program developed by Google DeepMind, played a five-game Go match with the 18-time world champion Lee Sedol[12]. The received high media attention. AlphaGo won all but the fourth game. This was considered another breakthrough in AI.

See Fig. 1.6.

Fig. 1.6: Lee Sedol, Go match with AlphaGo

To conclude, AI applications have made enormous progress in the last two decades - applications that were unthinkable in the late 20th century. All those applications exhibit behavior of human

[12] https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol

intelligence in certain situations (playing games, answering questions, driving cars, etc.) without necessarily imitating human intelligence as such. Looking inside the implementation of such applications may be disappointing to people hoping to gain insights in the very nature of human intelligence. They are IT applications and follow engineering practices of IT applications. And they may be useful in every-day's life.

The following chapters give insights into engineering such AI applications.

1.6 Further Reading

There are numerous publications on AI, scientific papers and text books. I present only a very short selection.

Stuart Russell's and Peter Norvig's book: "Artificial Intelligence: A Modern Approach" (Russell and Norvig, 2013) is the standard textbook on AI. It gives detailed insights in AI mechanisms and the theories behind them - issues which I do not focus on in this book. It may be used as an excellent source for further reading when you need to know more about those issues.

Marc Watson has written a number of books on practical aspects of AI: "Practical Artificial Intelligence Programming with Java" (Watson, 2013), "Practical Semantic Web and Linked Data Applications, Java, Scala, Clojure, and JRuby Edition" (Watson, 2010), "Practical Semantic Web and Linked Data Applications, Common Lisp Edition" (Watson, 2011). While they are some of the relatively few books dealing with the development aspects of AI applications, the focus is still somewhat different from the focus of this book. Watson introduces concrete frameworks and tools in concrete programming languages. In contrast, I rather give an overview of the tools available in form of services maps and product maps, together with hints as to which tools to use in which context and how to integrate them. Insofar, also Watson's book may be used for further reading in case you have decided on specific languages and tools covered by him.

Also, there are a number of recommendable online courses on AI - with a similar focus as Russell and Norvig's standard text book (2013). Examples are:

- Udacity: Intro to Artificial Intelligence[13]
- Coursera: Introduction to Artificial Intelligence[14]
- edX: Introduction to Artificial Intelligence[15]

1.7 Quick Check

The quick check shall help you assessing whether you have understood the main topics of this chapter. Answer the following questions.

[13] https://www.udacity.com/course/cs271
[14] https://www.coursera.org/learn/introduction-to-ai
[15] https://www.edx.org/course/introduction-to-artificial-intelligence-ai-3

1. What does the term “Artificial Intelligence” mean?
2. What are main areas of AI?
3. Sketch the history of AI
4. In what sense is AI today relevant and ubiquitous?
5. What are potential impacts of AI applications on society?
6. Name a few prominent AI projects

2. Machine Learning

Machine learning (ML) is currently the most prominent AI area, representing non-symbolic AI. Fig. 2.1. shows ML in the AI landscape.

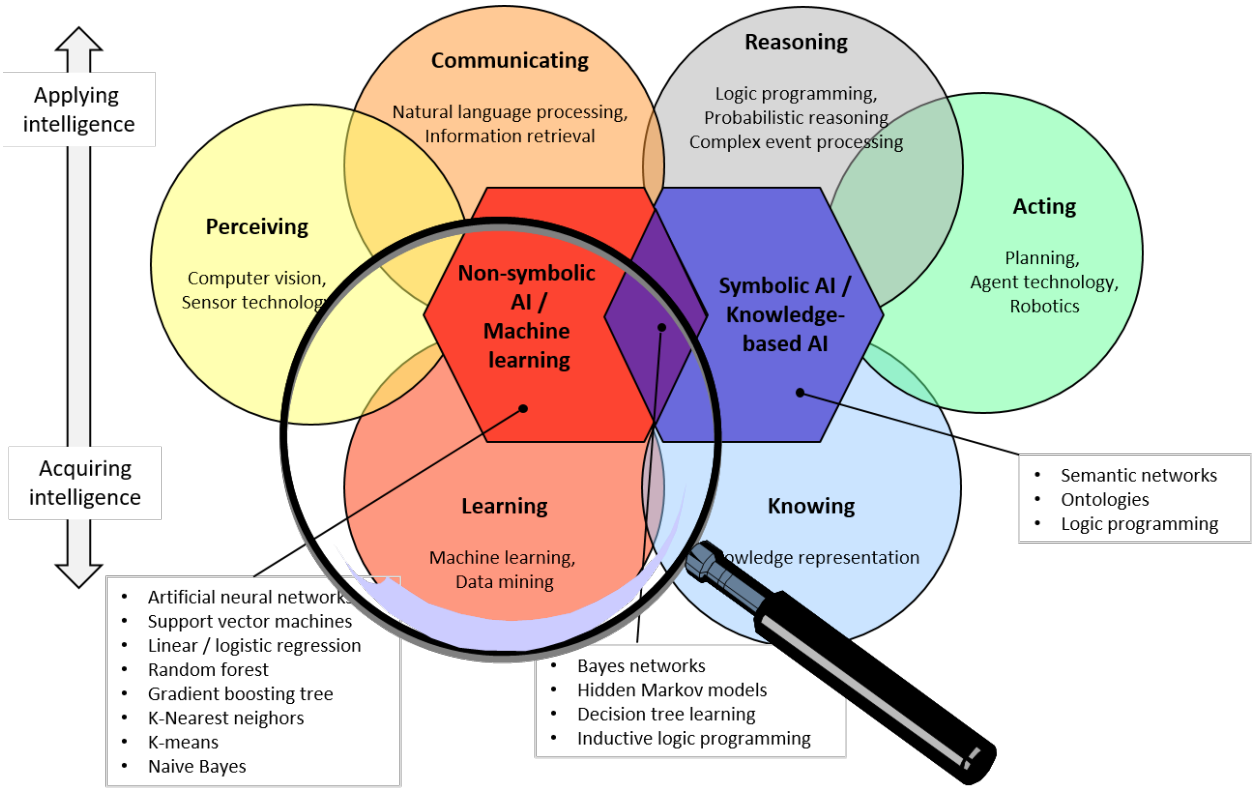


Fig. 2.1: ML in the AI landscape

What does ML mean?

ML allows for making decisions or predictions based on models which have been generated automatically ("learned") from sample data.

The models are generated automatically in a *learning phase* (a.k.a. *training phase*). Explicitly programming instructions for decisions or predictions, as in classic programming, is not necessary in a machine learning setting. Rather machine learning algorithms process data to learn patterns within them.

When to use ML?

You should use ML in problem domains where it is not possible or feasible to explicitly program a solution, e.g., if there is no known algorithm or explicit rules to solve the problem (e.g., identifying faces in images) or if the behavior depends on too many factors (e.g., spam filtering).

Never use ML when there is an explicit algorithm or explicit rules for solving a problem, e.g., in business applications where company regulations or legal requirements need to be implemented (taxation laws etc.).

2.1 ML Applications

ML approaches are used in numerous applications in daily use. Here are a few examples.

Spam Filtering

Spam filters based on machine learning are used in state-of-the-art email clients. The task is to classify incoming emails as spam or not spam. See Fig. 2.1.

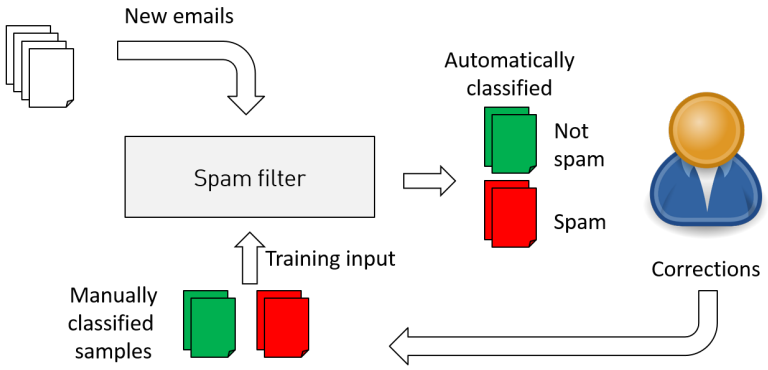


Fig. 2.1: Spam filtering

The ML spam filter is trained with manually classified samples: spam emails and non-spam emails. After the training phase, the spam filter automatically classifies emails in spam / non-spam. The user may correct ML decisions. Those corrections are used as new training samples that improve the correctness of future classifications.

Stock Market Analysis and Prediction

Consider the task of predicting the development of shares on the stock market in order to recommend sales and purchases. See Fig. 2.2.

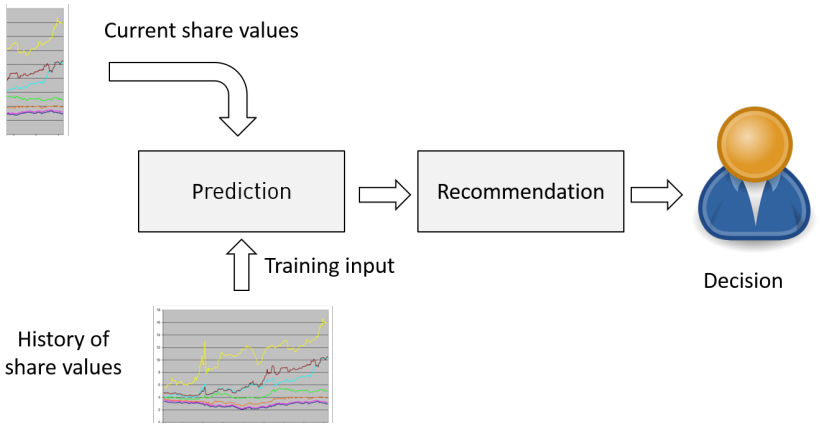


Fig. 2.2: Stock market analysis

The prediction model is continuously trained with the history of share values. It is used for predicting the development of shares based on the current share values. The predictions can then be used to create recommendations.

Fraud Detection

Consider the task of detecting fraud in financial transactions, e.g., suspicious credit card payments. See Fig. 2.3.

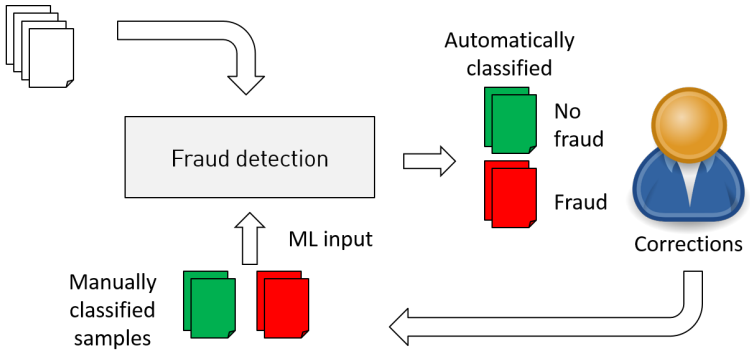


Fig. 2.3: Fraud detection

The overall process is similar to spam classification. The ML application is trained with manually classified samples and then used to automatically identify fraud. Human clerks validate whether, in fact, there is a case of fraud or not. Manual corrections of the automatic classification results are used to further train the ML application. In contrast to email spam classification, one sample does not necessarily consist of a single item but possibly of a series of transactions.

Recommender Systems

In recommender systems, the task is to recommend suitable products to customers on the basis of their previous purchasing behavior ("Customers who bought this product were also interested in those products..."). See Fig. 2.4.

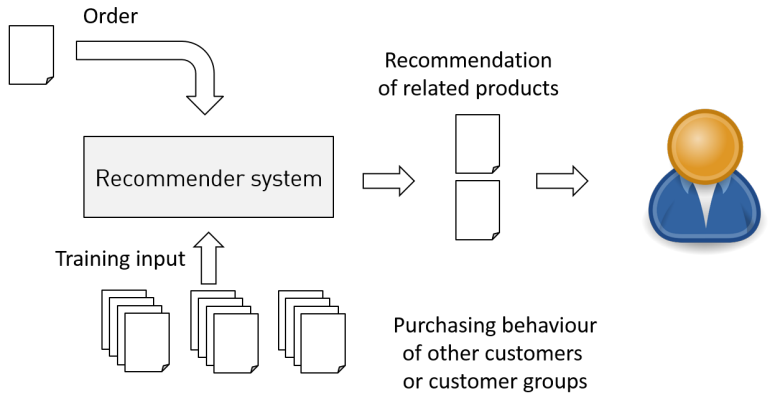


Fig. 2.4: Recommender systems

The ML recommender system is trained with the purchasing behavior of a large number of customers and associated customer groups. Then, based on a concrete order of a customer, related products are recommended.

2.2 ML Areas and Tasks

Machine learning is a wide area. There are numerous groupings and names for ML subareas and tasks in literature. Fig. 2.5 shows a simple way of categorizing machine learning.

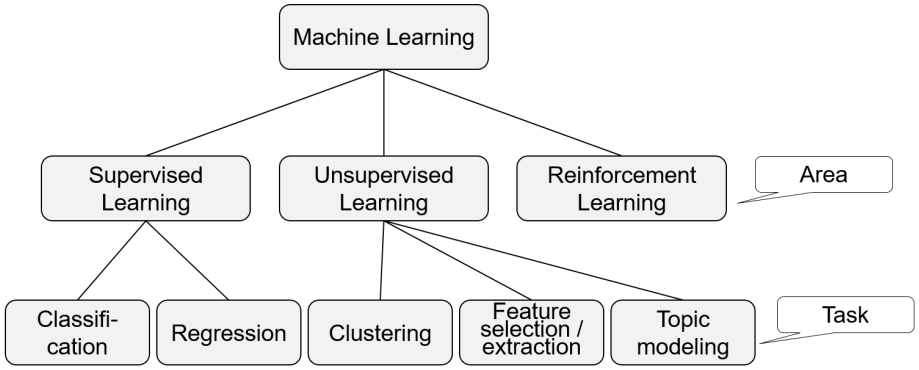


Fig. 2.5: ML areas and tasks

Three areas of machine learning can be distinguished: *supervised learning*, *unsupervised learning*, and *reinforcement learning*. With supervised learning, one can perform the tasks *classification* and *regression*; with unsupervised learning one can perform the tasks *clustering*, *feature selection / extraction*, and *topic modeling*.

Supervised Learning

Supervised learning assumes a (human) supervisor or trainer who trains the ML application with samples.

- *Setting*: Sample inputs and desired outputs as specified by the supervisor
- *Goal*: A model that maps new (unknown) inputs to correct outputs

Consider, e.g., the task of *classification*.

- *Setting*: A set of sample input data records (training data set) are given which are classified into a number of pre-defined classes
- *Goal*: A model that correctly classifies new input records.
- *Examples*:
  - Spam filtering. Input: Email meta-data and content; Classes: "spam" and "not spam"
  - Fraud detection. Input: financial transactions; Classes: "fraud" and "not fraud"

Now consider the task of *regression*.

- *Setting*: A set of sample input data records are given with continuous (not discrete) output values, e.g., floating point numbers
- *Goal*: A model that predicts output values for new inputs as closely as possible
- *Example*: Stock value prediction.
  - Input: History of values of a share
  - Output: Predicted values in the future

Unsupervised Learning

Unsupervised learning aims at detecting correlations in (large) data sets that are hard to detect by humans. There is no human supervisor who defines what correct classifications are. A related term is *data mining*.

- *Setting*: A set of data records are given
- *Goal*: Find meaningful structures within the data

Consider, e.g., the task of *clustering*.

- *Setting*: A set of input records are given
- *Goal*: Identify clusters, i.e. groups of similar records, where the similarity metric is initially unknown
- *Example* customer segregation: Groups of customers shall be identified that exhibit similar purchasing behavior. Those customer groups can then be targeted individually.

Now consider the tasks of *feature selection* and *feature extraction*.

- *Setting*: A set of input data records with attributes ("features") are given

- *Goal of feature selection*: Find a subset of the original attributes that are equally well suited for a classification / regression / clustering task. The idea is to automatically distinguish meaningful attributes from attributes that can be omitted without losing information when performing a certain ML task.
- *Goal of feature extraction*: Like in feature selection, the goal is to identify features that are meaningful for a certain ML task. While in feature selection, a subset of the original feature set is to be selected, in feature extraction a new set of features is to be constructed from the original feature set. As a result, various original features may be combined into one.
- *Example*: customer segregation; what features of a customer record are meaningful for customer segregation (e.g., number of sales) and which ones are not (e.g., name).

Finally, consider the task of *topic modeling*.

- *Setting*: A set of text documents are given.
- *Goal*: Find topics that occur in several documents and classify the documents accordingly.
- *Example*: Automatic extraction of a meaningful vocabulary for a specific field from a set of articles in this field.

Reinforcement Learning

Reinforcement learning is the form of learning which is most similar to the way humans learn.

- *Setting*: An agent interacts with a dynamic environment in which it must reach a goal
- *Goal*: Improving the agent's behavior
- *Example*: A chess-playing bot attempts the goal of winning chess games. Reinforcement learning can be applied when having a chess-playing bot playing many games against itself (or against other chess playing bots or humans). While playing, the effects of decisions are used to optimize parameters for decision taking.

In the development of AI applications, various ML tasks are often combined. For example, unsupervised learning is often applied before supervised learning. Via unsupervised learning, e.g., feature extraction, the input data is pre-processed. Usually this is done in cycles of automatic ML processes followed by manual quality assurance and adaptation processes. Once a set of meaningful features are identified, then supervised learning can be applied successfully.

2.3 ML Approaches

Over the many years of AI research, many different ML approaches have been developed and optimized. In the next sections, I will briefly introduce a few prominent ones.

Decision Tree Learning

Decision trees can be used for supervised learning: for classification as well as for regression. A decision tree is usually a tree in which internal nodes represent tests on attributes values of a data set; Branches represent the possible outcomes of the tests; Leaf nodes represent classes in a classification setting resp. numerical values in a regression setting.

See Fig. 2.6 for a simple example from the Decision tree learning page in Wikipedia[1].

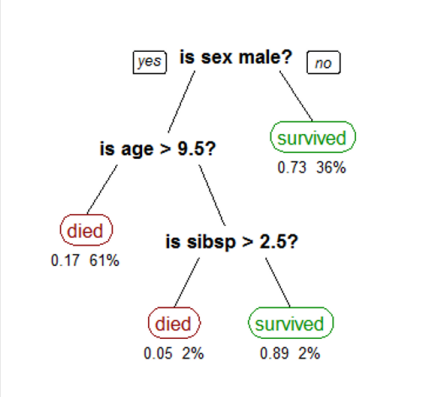


Fig. 2.6: Decision tree learning

This example decision tree shows the survival of passengers on the Titanic, depending on their sex, age and the number of spouses or siblings aboard (sibsp). The figures under the leaves show the probability of survival and the percentage of observations in the leaf.

In the training phase, a decision tree can be automatically generated from a set of sample data. Then, the decision tree can be used for classifying new data records simply by applying the tests from the root node to a leaf node.

The main advantage of decision trees is that they are easy to understand and interpret. People are able to understand decision tree models after a brief explanation and domain experts can validate generated decision trees. However, in complex scenarios where values are uncertain or many attributes are correlated, decision trees tend to become overly complex and cannot easily be validated by humans any more.

Artificial Neural Networks

Artificial neural networks (ANN) are inspired by nature. In the human brain and nervous system, neurons receive signals through synapses located on the dendrites. When the signals received are strong enough (surpass a certain threshold), then the neuron is activated and emits an electronic signal - it “fires”. This signal might be sent to another synapse, and might activate other neurons.

[1] https://en.wikipedia.org/wiki/Decision_tree_learning

And where does learning come into play? If an input of a neuron is repeatedly causing the neuron to fire, the synapse changes chemically, reducing its resistance. This adaptation has an effect on the neuron's activation behavior.

See Fig. 2.7.

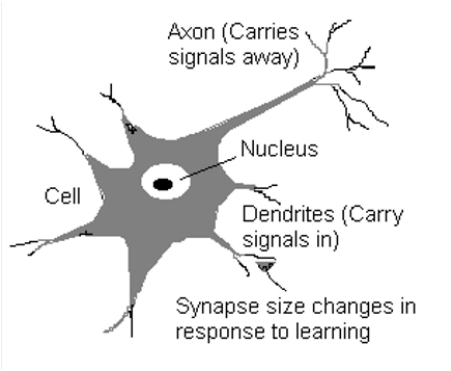


Fig. 2.7: Neuron (Galkin, 2016)

Artificial neurons simulate neurons in nature, however in a highly abstracted way. Artificial neurons consist of inputs (like synapses), which are multiplied by weights (strength of the respective signals), and then computed by a mathematical function. A simple example is the weighted sum of all inputs. This function determines the activation of the artificial neuron. Another function computes the output of the artificial neuron, sometimes dependent on a certain threshold.

See Fig. 2.8.

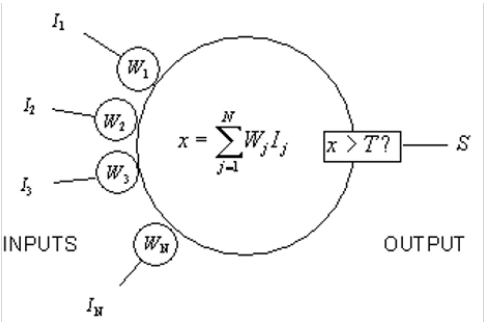


Fig. 2.8: Artificial Neuron (Galkin, 2016)

Finally, artificial neural networks (ANN) combine artificial neurons in order to process information. They consist of an input layer, an output layer, and possibly a number of intermediate layers of neurons.

See Fig. 2.9 for a simple example.

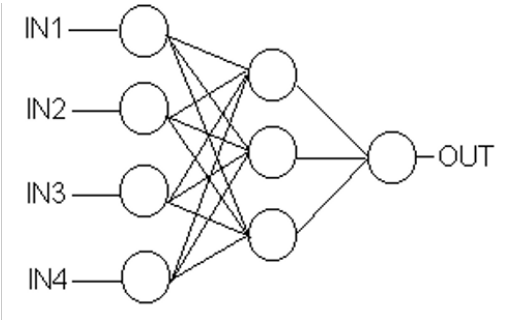


Fig. 2.9: Artificial neural network (Galkin, 2016)

For example, an artificial neural network could be used for recognizing characters from an image (OCR: Optical Character Recognition). In a simple setting where each image depicts exactly one character, each pixel of the image could be associated with one input neuron of the neural network; Each possible character (a-z, A-Z) could be associated with an output neuron.

In the *training phase*, manually classified images are fed to the neural network, one by another. The pixels of the images are connected to the input neurons and the manually identified characters to the output neurons. During the training phase, the weights and thresholds of the artificial neurons are adapted so that the input neurons optimally fit to the respective output neurons.

Then, in the *prediction phase*, the trained artificial network can be used to automatically classify new images, i.e., recognize the character on the image.

Fig. 2.10 shows an ANN playground[2] with the state-of-the-art ML tool Tensorflow. You can experiment with varying numbers of layers, neurons, etc. and get a feeling on how ANN behave on various problems.

[2] http://playground.tensorflow.org

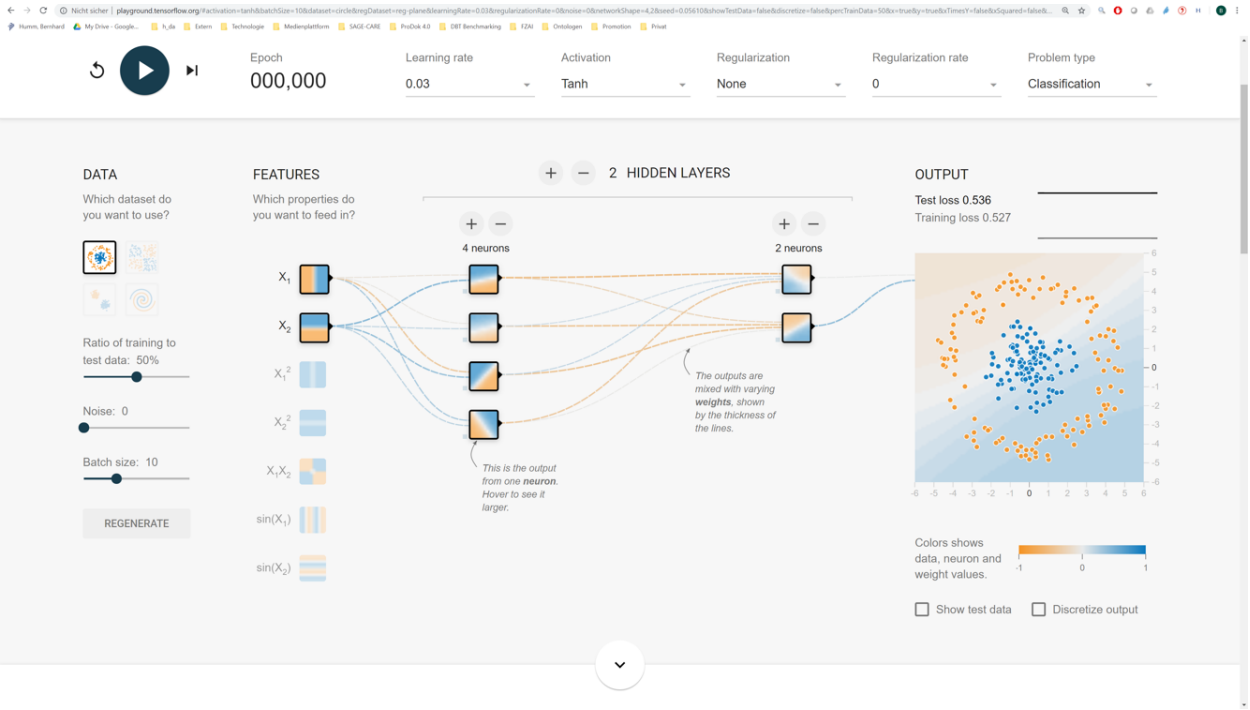


Fig. 2.10: ANN playground with Tensorflow

Artificial neural networks are popular since they are relatively easy to use and do not require a deep background in statistics. They can handle large amounts of data, implicitly detecting complex relationships between attributes.

An important disadvantage is that artificial neural networks behave as black-box systems. Users cannot explain how learning from input data was actually performed and, hence, cannot explain classifications made by the artificial neural network.

Support Vector Machines are a popular alternative to ANNs and share their advantages and disadvantages.

Deep Learning

Deep learning is a popular architecture of ANNs with a cascade of several layers. Each layer performs a specific tasks and each successive layer uses the output from the previous layer as input. Examples for individual tasks may be feature extraction (unsupervised ML), and classification (supervised ML). Deep learning is currently the most common approach for image recognition and voice recognition tasks. The first layers in the cascade perform low-level, pixel-oriented tasks including feature selection; later layers perform increasingly complex tasks like high-level recognition. See Fig. 2.11 for an illustration.

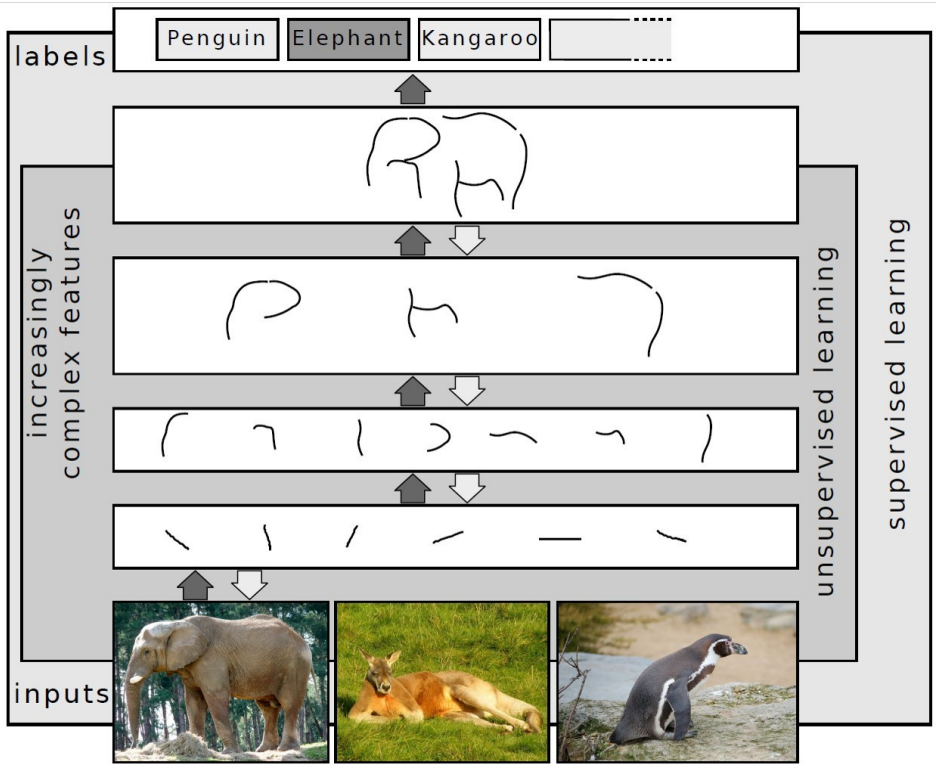


Fig. 2.11: Deep learning

In Chapter 7 we present deep learning for computer vision more in detail.

Transfer Learning

Transfer learning is a technique of re-using pre-trained deep neural networks, adding and training additional layers for specific tasks. This approach makes sense since training large ANNs may use an enormous amount of computing power and require a very large number of training data. For example, for classifying specific dog races, you can re-use a pre-trained ANN model like ResNet[3] and add layers for classifying the specific races. Then, you need to train just the added layers with specific training data.

In Chapter 7 we present transfer learning for computer vision more in detail.

Bayesian Networks

A *Bayesian network* is a directed acyclic graph (DAG) with random variables as nodes and their conditional dependencies as edges.

For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used for diagnostic purposes, computing the probabilities of the presence of various diseases. See Fig. 2.12.

[3] https://keras.io/applications/#resnet

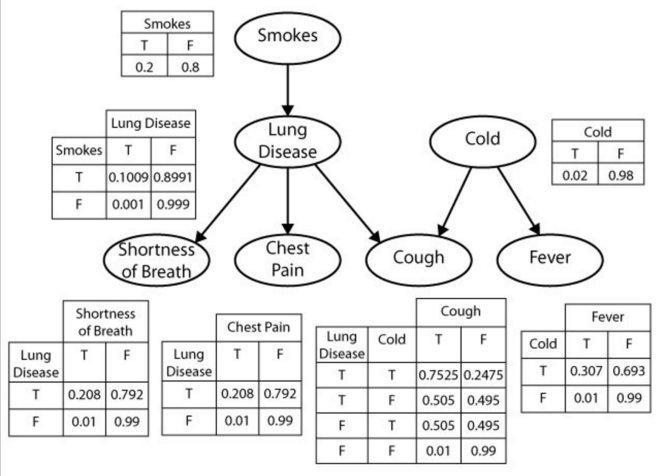


Fig. 2.12: Bayesian network example (Goodman and Tenenbaum, 2016)

In the example from Goodmann and Tenenbaum (2016), the disease "cold" may lead to symptoms like "cough" and "fever", whereas a lung disease may lead to symptoms like "shortness of breath", "chest pain", and "cough". Those dependencies are modeled in the Bayesian network as edges. Additionally, the probabilities of each individual situation (random variable) as well as the conditional probabilities (e.g., probability of fever in case of a cold) are denoted. Also the relationships between causes and diseases (e.g., smoking may cause lung diseases) can be modeled.

Defining the random variables (here: "causes", "diseases", "symptoms") and their dependencies is a manual task to be performed by experts, e.g., medical doctors. The probabilities may be derived from medical statistics.

After modeling the Bayesian network, it can then be used to compute the probability of certain diseases, given the patient's symptoms. The computation is based on Bayes' theorem on conditional probabilities.

Bayesian networks have the advantage over artificial neural networks and support vector machines that the reasoning behind a recommendation can be explained (Explainable AI). This, of course, is possible only in domains where conditions and dependencies are known and can be modeled explicitly.

Overview of ML Approaches

There are many more ML approaches than the ones which I briefly introduced above. *Inductive logic programming*, for example, is one of the early approaches. It allows learning logic rules from examples, e.g., learning the rule *parent(p1, p2) and female(p2) -> daughter(p2, p1)* from facts about a large number of people, containing the predicates "parent", "female" and "daughter".

A major disadvantage of such ML approaches purely based on logic is that the sample data must be 100% correct. One incorrect data record or a rare special case would invalidate correct rules, learned otherwise.

Since in many application scenarios, training data is prone to errors, noise, or other data quality problems, most ML approaches used today are probabilistic ones. They simply ignore rare exceptions as noise. Artificial neural networks, support vector machines and Bayes networks are prominent examples of probabilistic methods. Even decision trees can deal with probabilities.

There is not *the* single best ML approach. Some approaches allow for explaining the ML results, like Bayesian networks and decision tree learning, others do not. Some approaches take into account many hidden, implicit dependencies between attributes of the data without the need of explicit modeling, like artificial neural networks and support vector machines. Selecting the right ML approach for a specific use case requires experience and often a great deal of experimenting and testing.

Fig. 2.13 can help making such a decision by classifying prominent ML approaches to the main ML tasks.

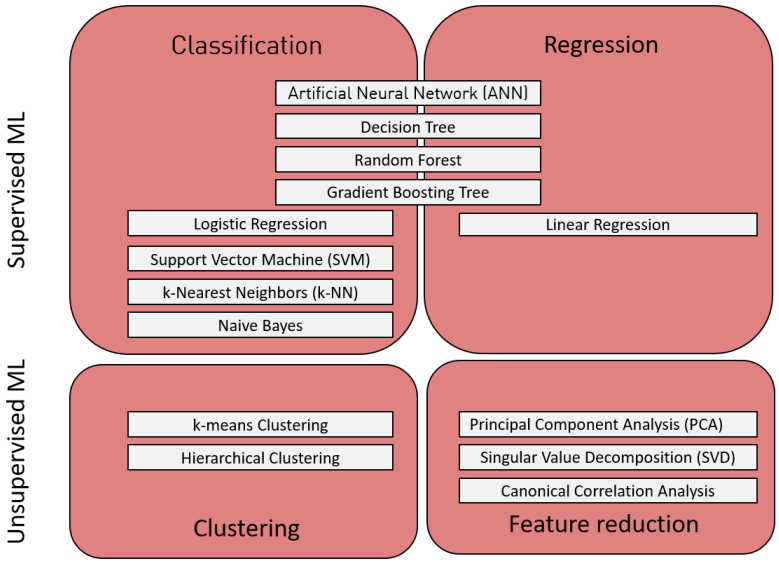


Fig. 2.13: Overview of ML approaches

To get further guidance in selecting a suitable ML approach for a specific tasks, various players in the ML area have provided so-called "ML cheat sheets". See a selection:

- Microsoft[4]
- SAS[5]
- Becoming Human[6]
- Peekaboo[7]
- The R Trader[8]

[4] https://docs.microsoft.com/de-de/azure/machine-learning/media/algorithm-cheat-sheet/machine-learning-algorithm-cheat-sheet.svg
[5] http://www.7wdata.be/wp-content/uploads/2017/04/CheatSheet.png
[6] https://becominghuman.ai/cheat-sheets-for-ai-neural-networks-machine-learning-deep-learning-big-data-678c51b4b463
[7] http://peekaboo-vision.blogspot.com/2013/01/machine-learning-cheat-sheet-for-scikit.html
[8] http://www.thertrader.com/wp-content/uploads/2018/03/Picture3.jpg

2.4 Example: Classifying Customers using Decision Tree Learning

I will show a simple classification example with decision trees. I will use the free Basic Edition of RapidMiner Studio[9], an integrated development environment for machine learning. In this example, a customer rating is to be performed. See Fig. 2.14 for a subset of the data set, taken from a RapidMiner tutorial.

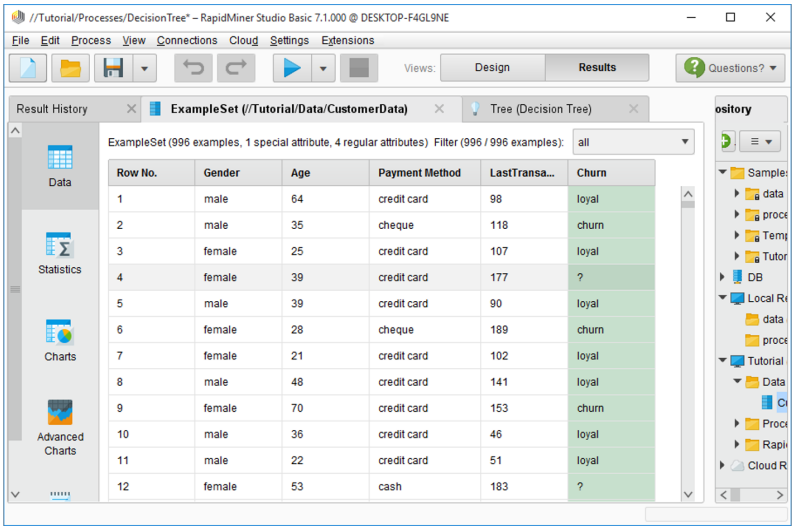


Fig. 2.14: Customer sample data

Customers are specified by a number of attributes, e.g., “gender”, “age”, “payment method” and “last transaction”. Those are the *input attributes* for machine learning.

The goal is to classify customers as “loyal” or “churn” in order to address them individually, e.g., via special offers. “Churn” is the *output attribute* which is to be predicted by the ML application. The value “churn” indicates the expected loss of a customer. Some of the customers are already classified (e.g., rows Nos. 1-3), others are not yet classified (e.g., row No. 4). A decision tree shall be generated from the customers already classified in order to predict the classification of the new ones. This is done by configuring a ML process. See Fig. 2.15.

[9] https://rapidminer.com/products/studio/

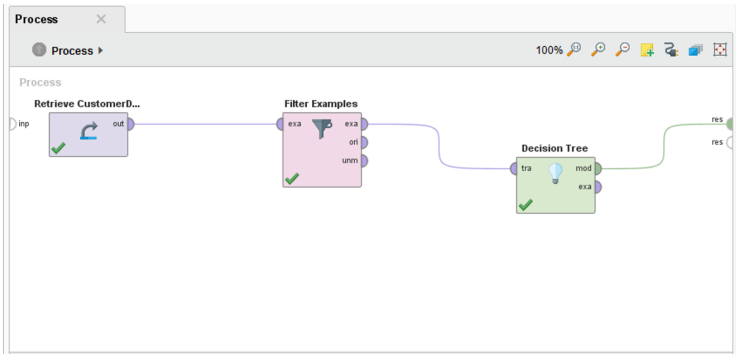


Fig. 2.15: ML process

The simple ML process consists of three steps, identified by boxes that are connected. The connections represent data flow. The first process step is the retrieval of the customer data. The next step is to filter those customer records that are already classified. From those records, a decision tree is being generated and returned as a result.

Fig. 2.16 shows the decision tree generated from the customer records.

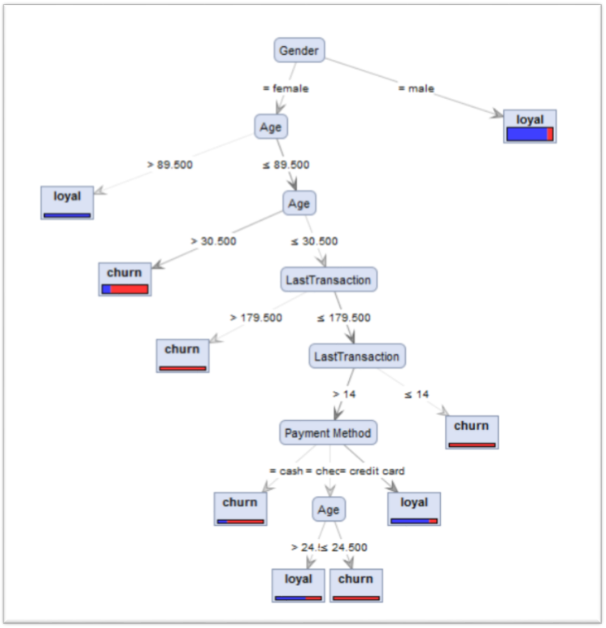


Fig. 2.16: Decision tree

The decision tree generated can be inspected by humans. The tree indicates that male customers have the tendency to be loyal. For female customers of a certain age group (> 89.5), this is also true. For other age groups, loyalty depends on last transaction and payment method.

Now, this decision tree may be applied to the customers not yet classified. The result is shown in Fig. 2.17.

Row No.	Churn	prediction(Churn)	confidence(l...	confidence(...	Gender	Age	Payment Me...	LastTransaction
1	?	churn	0.202	0.798	female	39	credit card	177
2	?	churn	0.202	0.798	female	53	cash	183
3	?	churn	0.202	0.798	female	33	credit card	194
4	?	churn	0.202	0.798	female	71	credit card	27
5	?	loyal	0.890	0.110	male	81	cash	153
6	?	churn	0.202	0.798	female	54	cheque	146
7	?	loyal	0.890	0.110	male	63	credit card	102
8	?	churn	0.202	0.798	female	58	credit card	176
9	?	churn	0.202	0.798	female	45	credit card	150
10	?	churn	0.202	0.798	female	33	credit card	144
11	?	loyal	0.890	0.110	male	40	credit card	82
12	?	loyal	0.890	0.110	male	36	credit card	91
13	?	churn	0.202	0.798	female	72	credit card	158

Fig. 2.17 Classification result

Each customer who was not previously classified (labeled with "?") is now classified either "loyal" or "churn", together with a confidence, e.g., 0.798.

Can we really trust the classification by the decision tree? I discuss validating the ML result in the following section.

2.5 ML Methodology

The ML Process and Data Flow

The process for implementing a supervised ML application consists of two major phases: the training phase and the productive use. See Fig. 2.18 for a simplified overview as BPMN diagram[10].

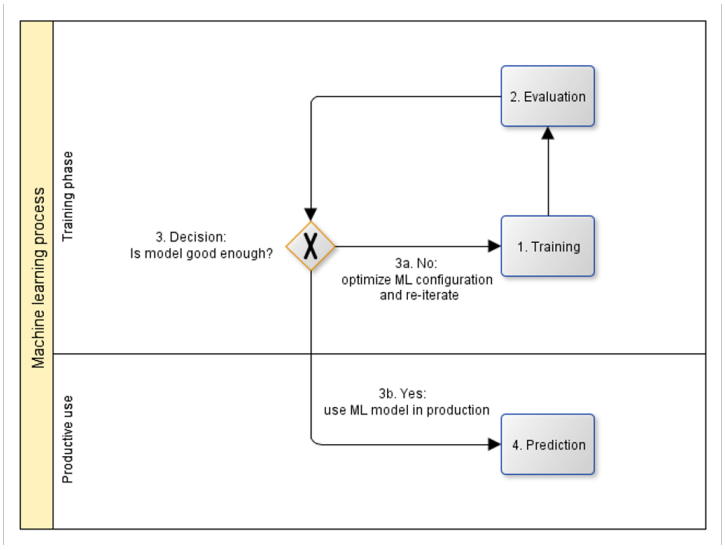


Fig. 2.18: The ML process simplified

[10] https://www.omg.org/spec/BPMN

In the training phase, a training data set is used to generate an ML model using some ML approach. But can we really trust this model? Is it capable of making good predictions when used in production? To find out, the model's prediction performance is evaluated, i.e., how accurately the model actually predicts. The evaluation is done on a test data set. If the model is good enough, then it can be used for predictions in productive use. However, usually the first model is not nearly good enough for productive use. Then, the ML configuration needs to be adopted, e.g., a different ML approach chosen or parameterised differently, and the training / evaluation cycle starts again - until the model is finally good enough.

Fig. 2.19 zooms into the ML process and gives an overview of the data flow in supervised ML.

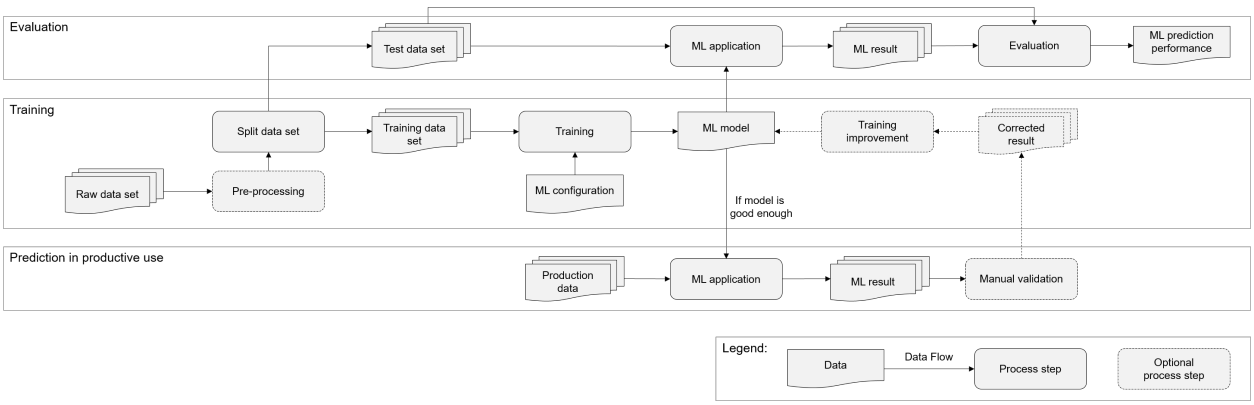


Fig. 2.19: Data flow in supervised ML

In the training phase, the raw data set may be pre-processed, e.g., by normalizing certain values. The input data set is in tabular form. Each row is a training sample. Columns are input attributes (so-called "features") and the expected output (so-called "labels"). The input data set is then split into a training data set and a test data set. The training process step uses the training data set and the ML configuration in order to generate the ML model.

In the evaluation phase, this ML model is applied to the test data set, using the input attributes only. The ML result is compared with the expected output (evaluation) and the prediction performance is measured.

If the model is not yet good enough, then the ML approach needs to be adjusted. For example, pre-processing can be fine-tuned, the ML approach can be changed (e.g., from decision tree to ANN), the parameters of the ML-approach (so-called "hyperparameters") may be adjusted, e.g., the number of layers and neurons in an ANN or the maximum depth of a decision tree. Then, training and evaluation is re-iterated.

If the model is good enough, i.e., the prediction performance is adequate, then the previously trained ML can be used productively. For this, the model is used to perform the ML application on production data. The computed result can be used in the AI application, e.g., displayed in a user interface. In case the users perform a manual validation of the computed results, corrected results may be fed back for training improvement.

Prediction Performance Measures

Can we really trust the results of an ML application? How confident are we that results are correct? Before using an ML application in production it is important to get a feeling of how good its predictions actually are.

Confusion Matrix

A *confusion matrix* is the basis for measuring the prediction performance of ML applications for classification. The name indicates that it allows to realize if the ML application is confusing two classes, i.e. commonly mislabeling one as another. See Fig. 2.20.

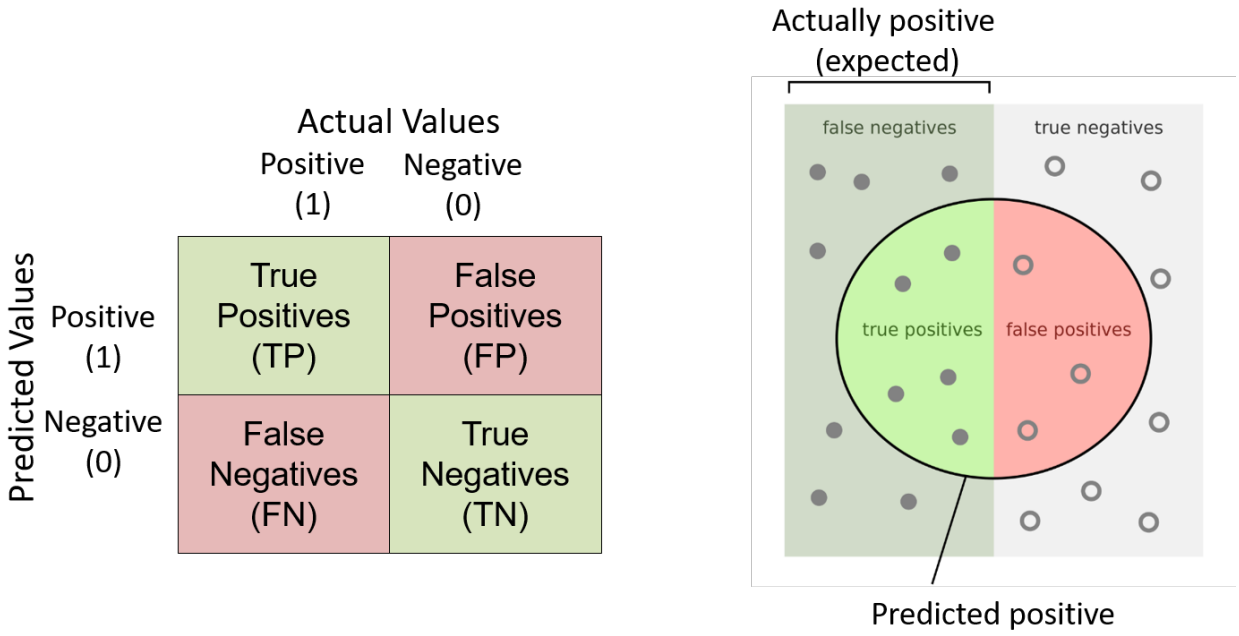


Fig. 2.20: Confusion matrix

The rows relate to the values predicted by the ML application, for a binary classification positive (1) and negative (0). The columns relate the actual values, i.e., what should have been predicted. The cells of the matrix contain 4 numbers:

- *True positives (TP)*: correctly predicted positive (1)
- *True negatives (TN)*: correctly predicted negative (0)
- *False positive (FP)*: predicted positive (1), but actually negative (0)
- *False negative (FN)*: predicted negative (0), but actually positive (1)

Consider the example of an ML application which predicts cancerous tumors of patients based on images. There are two outcomes of the classification:

- Malignant: cancer predicted
- Benign: no cancer predicted

See Fig. 2.21 for an exemplary confusion matrix for cancer prediction.

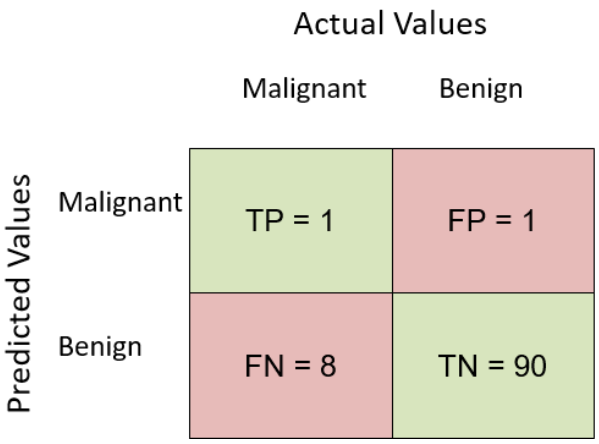


Fig. 2.21: Confusion matrix: example cancer prediction

In this example, out of 100 predictions, 91 were correct (TP = 1, TN = 90) and 9 were incorrect (FP=1, FN = 8). Is this a good prediction performance? In the following section we compare various prediction performance measures.

Accuracy

Accuracy is a simple, intuitive and frequently used prediction performance measure. It simply puts the number of correct predictions in relation to all predictions. It is defined as follows.

$$Accuracy = \frac{|correctPredictions|}{|allPredictions|}$$

For binary classifications, accuracy can be computed as follows.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy is a percentage, i.e. the value is always between 0 and 1. The higher the value the better. Whether an accuracy of 75% is already sufficient or, say, 99.9% is required depends entirely on the application use case. It is important to specify the required accuracy (or other performance measure depending on the use case) before implementing the ML application. This enables you to decide after the evaluation, whether or not the prediction performance is good enough to use the application in production.

In the cancer prediction example above, the accuracy can be computed as follows.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{1 + 90}{1 + 90 + 1 + 8} = 0.91$$

An accuracy of 91% looks like a decent prediction performance, does it not? However, let us have a closer look. In the example, 8 out of 9 malignant tumors were wrongly diagnosed as benign. If doctors had trusted the ML classification, those patients would not have undergone treatment and may have died because their cancer had not been detected. This is a catastrophe!

How can it be that such a fatal misclassification rate results in such a high accuracy value? The reason is that the data set is *imbalanced*: 91% of the sample cases are benign and only 9% are malignant - and we are interested in the minority class malignant.

Precision, Recall and F measure

The prediction performance measures Precision, Recall and F measure are better suited for imbalanced data sets than accuracy is.

Precision, a.k.a. *Positive predictive value (PPV)*, measures how precise the result is when the ML application predicts the class of interest, e.g., malignant. It is defined as the ratio of true positives to all positively predicted samples.

$$Precision = \frac{TP}{TP + FP}$$

Recall, a.k.a. *true positive rate (TPR), sensitivity, power*, or *probability of detection*, is the probability that a positive sample will be correctly classified. It is defined as the ratio of true positives to all actually positive samples.

$$Recall = \frac{TP}{TP + FN}$$

Both, precision and recall are percentages, i.e., values between 0 and 1. The higher the value the better. In the cancer classification example above, the following values for precision and recall can be computed.

$$Precision = \frac{TP}{TP + FP} = \frac{1}{1 + 1} = 0.5$$

When the ML application predicts a malignant tumor, it is correct in 50% of the cases.

$$Recall = \frac{TP}{TP + FN} = \frac{1}{1 + 8} = 0.11$$

The ML application correctly identifies only 11% of all malignant tumors, and misses out on the other 89%.

Precision and recall are mutually conflicting measures. Optimizing just one measure will impair the other. For example, it is possible to ensure a recall of 100% by simply implementing a trivial predictor that predicts malignant in all cases. Theoretically, you do not miss out on any actually malignant case, but such a predictor is obviously not useful at all.

Therefore, precision and recall must be balanced. The *F measure*, a.k.a *F1 score*, is defined as the harmonic mean of both measures.

$$F = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Like precision and recall, F measure is a value between 0 and 1 - the higher the better. In the cancer classification example above, the F measure can be computed as follows.

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} = 2 \cdot \frac{0.5 \cdot 0.11}{0.5 + 0.11} = 0.18$$

Clearly, the low F measure of 18% much better represents the catastrophic prediction performance of the ML application than the high accuracy value of 91%, both based on the same confusion matrix.

Which prediction performance measure to chose for classification tasks?

Selecting a suitable prediction performance measure and required value for a concrete classification use case is a difficult task. As a rule of thumb, accuracy as a simple and easy-to-understand measure is usually a good choice when dealing with balanced datasets.

As the example above indicates, accuracy should never be used with heavily unbalanced datasets. In those cases consider using F measure. However, also F measure may have flaws in certain application scenarios. Notice that F measure just focuses on one class of interest and does not take true negatives into account at all. For the cancer prediction example this may be adequate since the use case is about treating cancer and not healthy people. In other application scenarios this might not be appropriate at all.

There are many other prediction performance measures, all with advantages and disadvantages. Another popular prediction performance measure is AUC (Area Under the receiver operating characteristic Curve). Other measures are Cohen's Kappa, Bookman Informedness, logarithmic loss, specificity, prevalence, positive/negative likelihood ratio, positive/negative predictive value, and more. For definitions see the Wikipedia page Confusion Matrix[11].

Prediction performance measures for regression: MAE, MSE and RMSE

Confusion matrices and the prediction performance measures introduced above relate to classification tasks where a prediction either matches the actual class or not. In regression tasks, where

[11] https://en.wikipedia.org/wiki/Confusion_matrix

numeric values are predicted, the question is how close the predicted values are to the actual values. See Fig. 2.22.

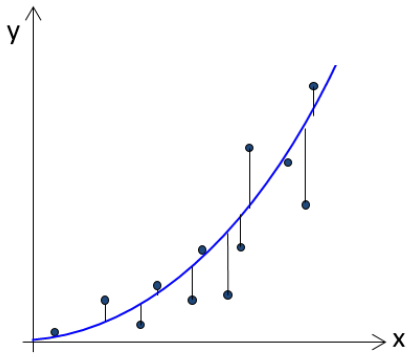


Fig. 2.22: Errors between regression function and data points

The most commonly used prediction performance measures for regression tasks are the mean absolute error (MAE) and the rooted mean squared error (RMSE).

The *mean absolute error (MAE)* is defined as the mean of the absolute values of the differences between predicted values $\hat{f}_i$ and actual values y_i.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|\hat{f}_i - y_i\right|$$

The MAE is a value greater or equal zero - the lower the better. The measuring unit of the MSE is the one of the values of the regression task. E.g., if house prices in USD are measured, then the measuring unit of the MAE is USD. It tells you the average regression error.

The *rooted mean squared error (RMSE)* is defined as the rooted mean of all squares of differences between predicted and actual values.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{f}_i - y_i)^2}$$

Like MAE, the RMSE is a value greater or equal zero and its measuring unit is the one of the values of the regression task. So, both are similar. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. Therefore, the RMSE is preferred if large errors are particularly undesirable. Note that RMSE values are greater than or equal to MAE values.

Often, also the *mean squared error (MSE)* is used. It simply omits the square root in the formula for RMSE. MSE values are not as easily interpreted as MAE or RMSE values since they are squares. In the house price prediction example above, the measuring unit of the MSE is USD^2.

k-fold Cross-Validation

It is important that the training set used to train an ML model is disjoint from the test set used to evaluate the model. See the process step "split data set" in Fig. 2.19. But be aware that the way the data set is split has an effect on the training as well as on the evaluation. Imagine if in the cancer classification example, by chance, all samples of class "benign" end up in the training set and all samples of the class "malignant" end up in the test set. Then the training step will not result in a reasonable ML model and the evaluation step will not meaningfully evaluate it.

It is a good idea to randomly shuffle the input data set before splitting it into training and test set. But still there is some element of chance in this procedure. *k-fold cross-validation* is a method for largely removing this element of chance. See Fig. 2.23.

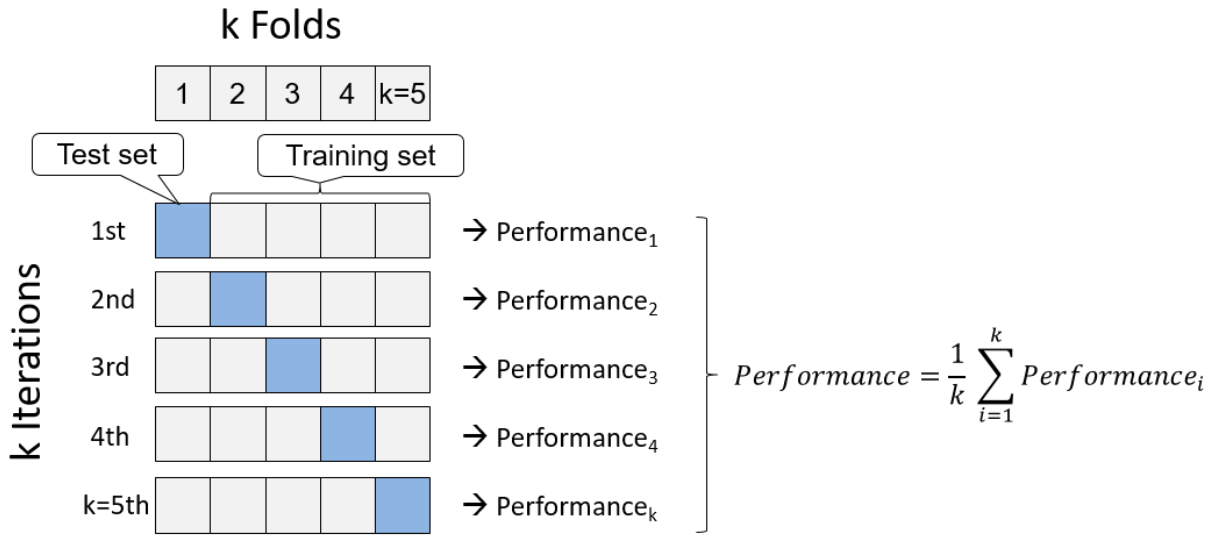


Fig. 2.23: k-fold cross validation

The general procedure is as follows.

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each group, i.e., k times do:
    1. Take the group as the test set
    2. Take all remaining k-1 groups together as the training set
    3. Train a new model with the training set
    4. Evaluate this model with the test set and compute the prediction performance
    5. Store the prediction performance and continue with the loop
4. Return the average of the k computed prediction performances as the result.

Many ML libraries and toolkits contain easy-to-use components for k-fold cross validation. k can be set as a parameter. A typical value is k=10.

Take as an example the RapidMiner process for customer rating introduced above. RapidMiner offers a validation process which performs k-fold cross-validation. This can be used to evaluate the accuracy of the decision tree for the customer rating example above. See Fig. 2.24.

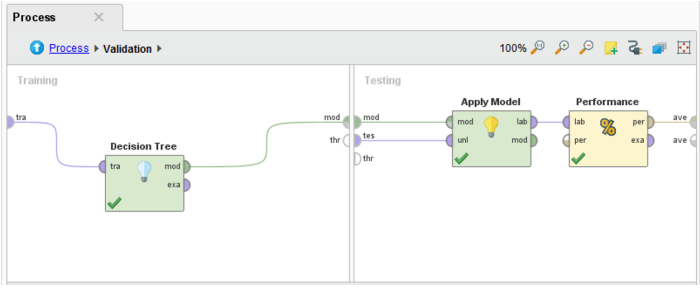


Fig. 2.24: Cross validation

The validation process consists of two sub-processes. The first sub-process on the left hand side represents the training phase. Input is the training data set. Output of the decision tree module is the ML model, i.e., the decision tree.

The second sub-process on the right hand side represents the evaluation phase. Input to the model application is the decision tree as well as the test data set. The result of the validation process is the confusion matrix and the average accuracy. See Fig. 2.25.

accuracy: 83.89% +/- 3.49% (mikro: 83.89%)

	true loyal	true churn	class precision
pred. loyal	509	76	87.01%
pred. churn	69	246	78.10%
class recall	88.06%	76.40%	

Fig. 2.25: Accuracy

The result of cross-validation is displayed as a confusion matrix, showing the number of true positives, true negatives, false positives and false negatives. For each class (“loyal” and “churn”), the precision and recall values are displayed. Additionally, the total accuracy is shown which is 83.89%.

Bias and Variance - Overfitting and Underfitting

A central point in the ML process as depicted in Figures 2.18 and 2.19 is optimizing the ML configuration if the ML model is not yet good enough. But how to do this in a meaningful way? To acquire an understanding of the problems at hand we introduce the concepts of bias and variance / overfitting and underfitting for regression tasks. Please note that those concepts equally apply to classification tasks.

An ML model can be seen as a generalization of the training data. In supervised ML, we assume that there is a real function f with the features as input and the target as output. However, f is unknown. I f was known, then there would be no need for machine learning at all. The ML model we get from training we call $\hat{f}$.

$\hat{f}$ should be as close as possible to f. In practice, it is impossible that $\hat{f} = f$. This is due to missing data, noise, etc. Those factors lead in any case to some error, which is sometimes called *irreducible error*. Take the cancer prediction example. The image data is just an extremely small subset of all possible information about a patient and the image data itself is subject to all sorts of noise due to the physics of the camera, image processing, storage, etc.

So, the goal is to find a function $\hat{f}$ which is close enough to f. To illustrate this, see Figures 2.26 and 2.27.

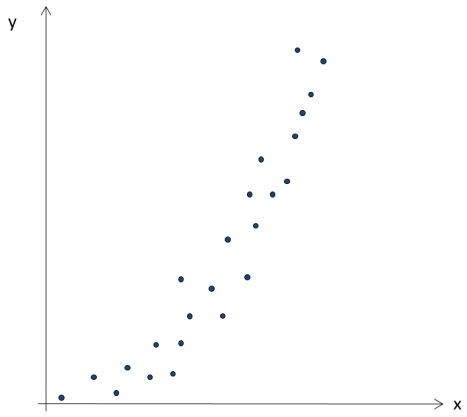


Fig. 2.26: Example data for regression task

Fig. 2.26 shows sample data points for a simple regression task with just one input (x axis) and one output (y axis). The question is: which function $\hat{f}$ best fits the sample data points?

Let us simply look for a polynomial function that fits the data points as closely as possible. The question now is: of which degree should the polynomial be? 0 (constant), 1 (linear), 2 (quadratic) or any higher degree? Fig. 2.27 shows three different options for $\hat{f}$.

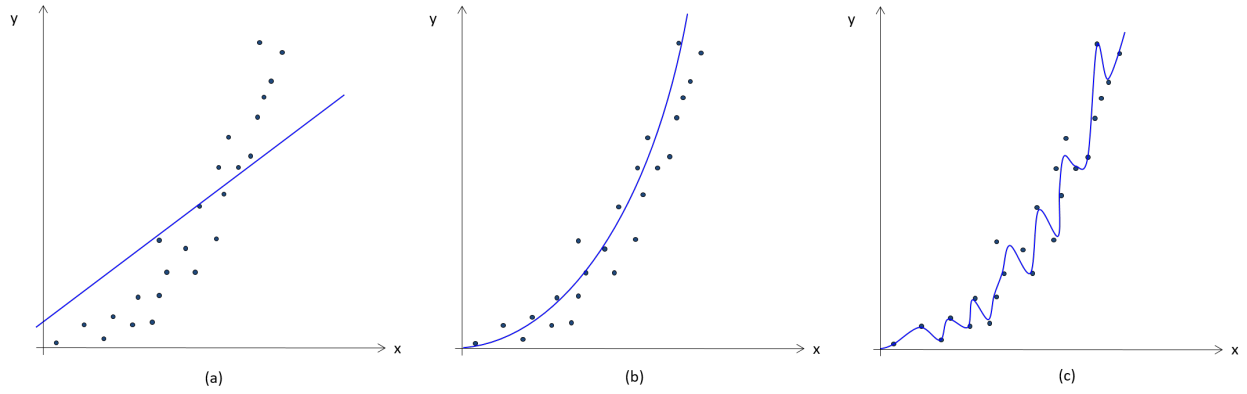


Fig. 2.27: Regression polynomials (a) degree = 1 (linear), (b) degree = 2 (quadratic), (c) degree >> 3

Fig. (a) shows a linear approximation (degree = 1), Fig. (b) a quadratic approximation (degree = 2), and Fig (c) a polynomial with a degree much higher than 3.

Which one is best?

If we simply look at the regression error with respect to the training data, e.g. using the MAE measure, then (c) is the best since the polynomial nearly meets every data point of the training data. But we can assume that the data contains some noise and the real (unknown) function f is not as twisty.

Without knowing the application domain we cannot really make an informed decision. However, by visually inspecting the data points there is a clear indication that the polynomial (a) (degree = 1) is too simple and the polynomial (c) (degree >> 3) is too complex. It looks like the quadratic polynomial (b) (degree = 2) best approximates the real function f and should be used as $\hat{f}$.

Bias is the error (e.g., measured by MAE or RMSE) between the ML model $\hat{f}$ and the data points in the training data set. In Fig. 2.27, (a) has the highest bias and (c) the lowest bias, since the complex polynomial (c) fits the data points much more closely than the simple polynomial (a).

Variance indicates how much the ML model $\hat{f}$ changes if the training data is changed, e.g., by taking a different training set in k-fold cross validation. In the example, the twisty polynomial (c) will look very different if the training data points change, whereas the simple polynomial (a) will not change much if some other noisy data points are used for training.

Bias and variance are mutually conflicting and should both be avoided. If an ML model is too simple then bias is high and variance is low. We will get bad prediction performance measures on a test data set. We call this *underfitting*.

On the other hand, if an ML model is too complex then variance is high and bias is low. We also will get bad prediction performance measures on a test set. Measuring prediction performance on the training set would reveal good results since the ML model learns noise in the data - but the prediction performance on the training set is not relevant. We call this *overfitting*.

Underfitting is as bad as overfitting. The art of machine learning is finding the *appropriate* model complexity - neither too simple nor too complex. See Fig. 2.28 (adopted from Dataquest[12]).

[12] https://www.dataquest.io/blog/learning-curves-machine-learning

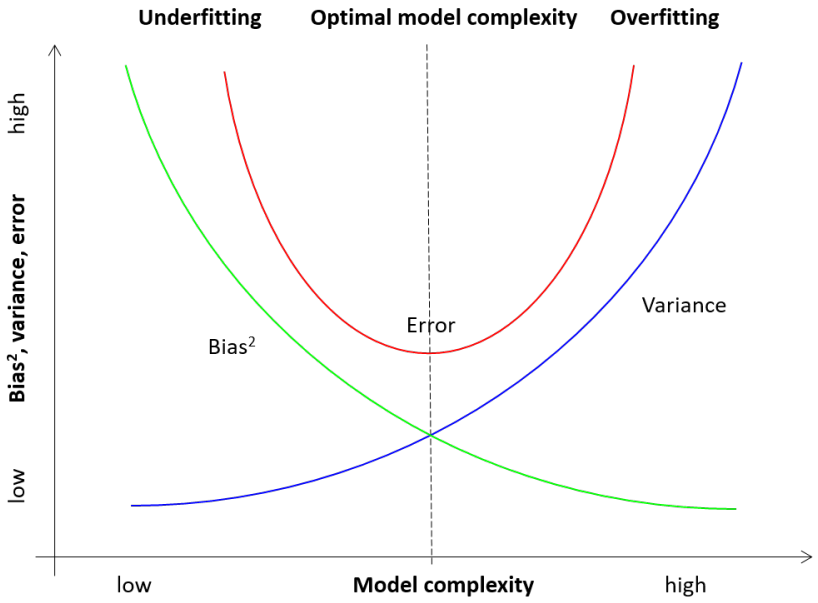


Fig. 2.28: Model complexity

The total error of an ML model (red curve) stems from errors from bias (blue curve) and variance (green curve). Both in underfitting scenarios (low model complexity, high bias, low variance) and in overfitting scenarios (high model complexity, low bias, high variance), the total error is high. However, the total error is minimized in scenarios where bias and variance are balanced.

Optimizing ML configuration

In the simple example shown in Fig. 2.27, it is possible to guess the appropriate model complexity by visually inspecting the data points. In realistic ML scenarios with dozens, sometimes thousands of features, this is impossible. How to optimize an ML configuration in practice if the prediction performance is not yet good enough?

As said before, the ML approach can be changed (e.g., from decision tree to ANN) or the hyperparameters may be adjusted, e.g., the number of layers and neurons in an ANN or the maximum depth of a decision tree. Optimizing the ML configuration is often a question of *experience* (see, e.g., the ML cheat sheets mentioned above) as well as trial and error.

The trial and error approach is obviously not satisfying. You can also view finding the best ML configuration as an optimization task with the goal to reach the best prediction performance. Many ML toolsets provide features for performing such an optimization. See Fig. 2.29 for the AutoModel feature in RapidMiner.

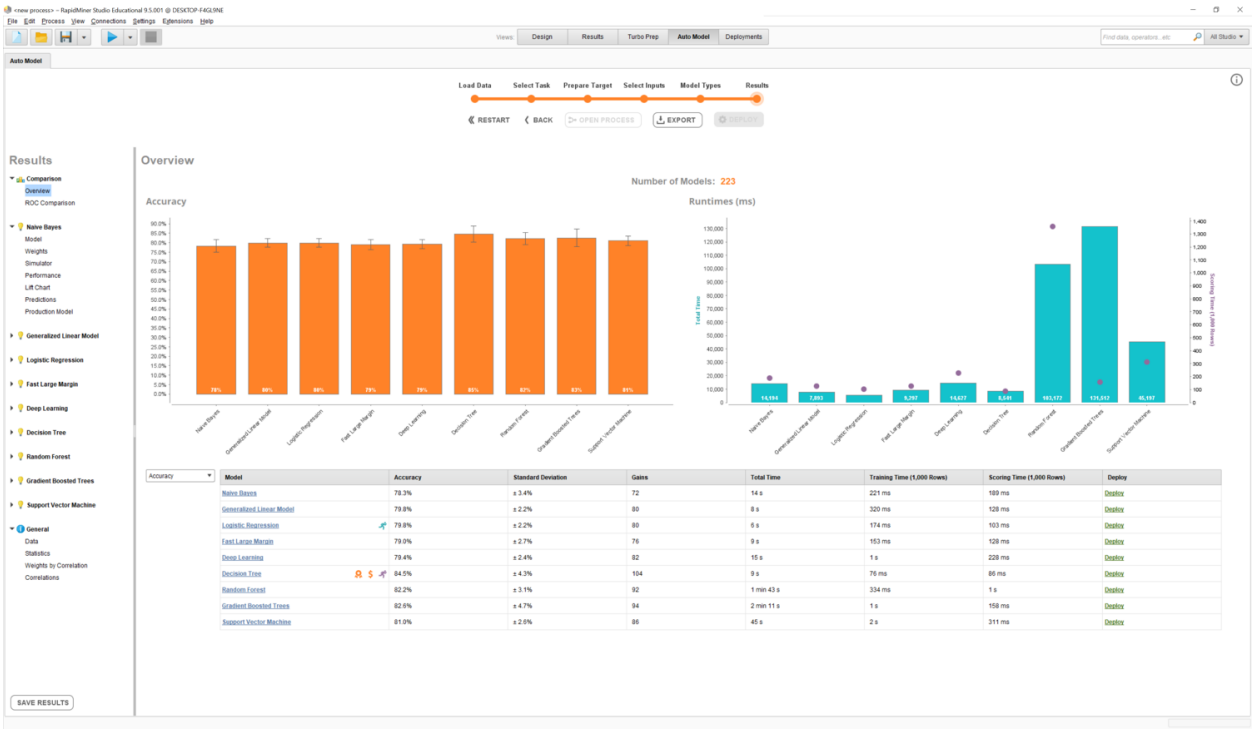


Fig. 2.29: AutoModel feature in RapidMiner

Here, the customer classification example from Section 2.4 is used. 9 different ML approaches are tried out, including deep learning, decision tree, and support vector machine. For all approaches, different hyperparameters are tried out and the best prediction performances are displayed. With an accuracy of 85%, decision tree turns out to be the best approach. The Decision tree approach also exhibits the best runtime performance (76 ms for training and 76 ms for scoring 1000 samples).

However, since training complex ML models may consume a considerable amount of computing power, optimizing the ML configuration will be even more costly. In the next section we present a method for optimizing ML configurations in a more informed way.

Learning Curve Analysis

Learning Curves

A general rule of thumb in machine learning is: the more training samples the better. Therefore, you will always use all training data available to train the best model.

In a *learning curve analysis*, however, you deliberately omit larger parts of the training data. Why would you do this? In order to get a feeling for the progression of the ML prediction performance with more and more training data. See Fig. 2.30 for a schematic diagram of a *learning curve* for a regression task (adopted from Dataquest[13]).

[13] https://www.dataquest.io/blog/learning-curves-machine-learning

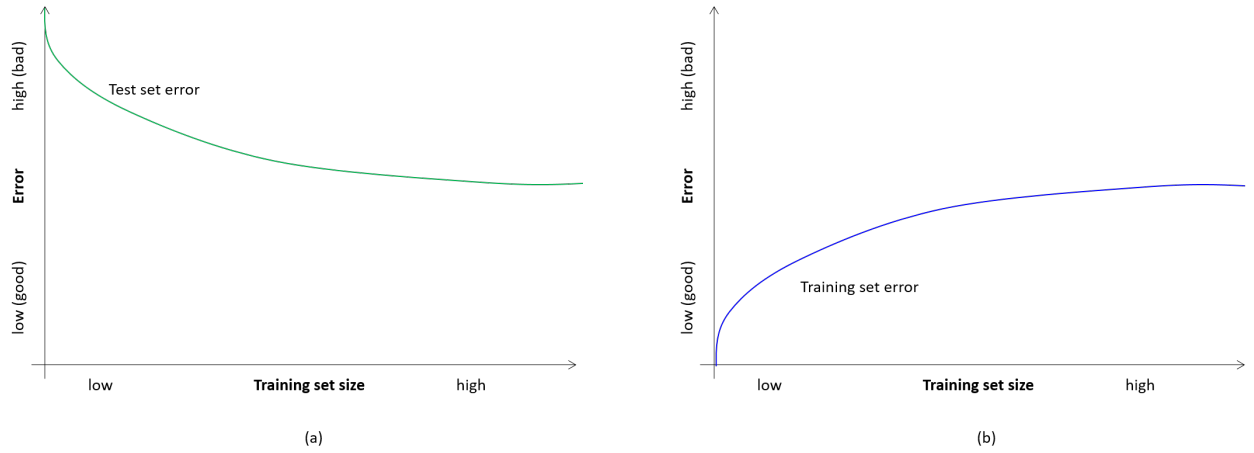


Fig. 2.30: Learning curves: (a) test set (b) training set

On the x axis of the learning curve, the number of training samples is charted, e.g., N=1, 10, 50, 100, 500, 1000, etc. On the y axis, the respective ML prediction performance is shown, e.g., the MSE. For a regression task, a high value is bad and a low value is good. As outlined above, what you are really interested in is optimizing the ML prediction performance on a test data set. This is shown in Fig. 2.30 (a). The more training data is used, the better the ML model and, consequently, the better the prediction performance. The curve is falling with an asymptotic approximation to some optimum.

Fig. 2.30 (b) shows the learning curve, however showing the prediction performance on the training data set instead of the test data set. The curve is constantly rising with an asymptotic approximation to some value. This is because the effect of overfitting is diminishing with an increasing number of training samples. Why would you look at the training set when you are actually interested in optimizing the prediction performance on the test set? This is because interpreting the progression of both curves as well as the relationship between the curves may gives you hints how to optimize the ML configuration in a meaningful way.

Interpreting Learning Curves

Take a look at the learning curves for training and test set in Fig. 2.31 (a).

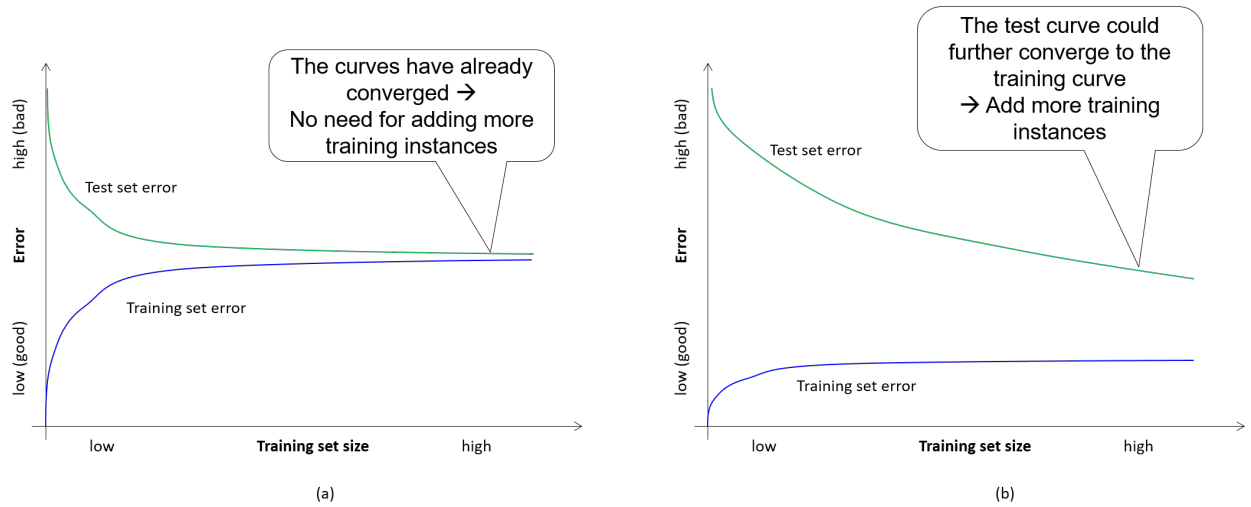


Fig. 2.31: Learning curve analysis

Both curves have nearly converged. It is obvious that adding more training data with the same ML configuration will not have a significant effect on the prediction performance. If you are happy with the prediction performance achieved, i.e., the model is good enough, then you can use it in production.

If, however, the error of the test set is too high, then ML configuration needs to be adapted. The high error indicates a high bias. The fact that adding more training data does not much change the prediction performance indicates a low variance. High bias and low variance together indicates that we are in an underfitting situation. Therefore, we need to takes measures for increasing the model complexity.

Assume that after increasing the model complexity, we get learning curves as shown in Fig. 2.31 (b). You can make two observations: (1) the training set error is converging to a much lower (better) value than before. (2) The test set error is already lower (better) than before, but has not converged yet. This shows that increasing the model complexity has already had a positive effect and adding more training data will additionally increase the prediction performance.

Increasing and Decreasing Model complexity

When the learning curve analysis has revealed that you are in an underfitting or overfitting situation, which measures can be taken to increase respectively decrease model complexity? You have to adjust the hyperparameters in your ML configuration. For example, you can add or remove features, have more or less layers in an ANN, have a higher or lower maximum tree size in a decision tree etc. See Fig. 2.32 for selected measures for increasing and decreasing model complexity.

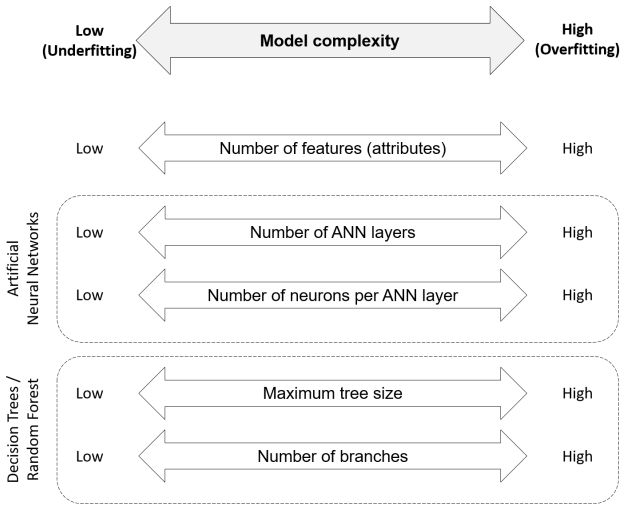


Fig. 2.32: Increasing and Decreasing Model complexity

Learning Curve Analysis for Classification Tasks

The learning curve examples above relate to regression tasks. For classification tasks, learning curve analysis is the same approach. The only difference is the prediction performance measure, e.g., accuracy or F1 score instead of MAE or RMSE. Here, high values are good and low values are bad. Therefore, both learning curves are reversed. See Fig. 2.33.

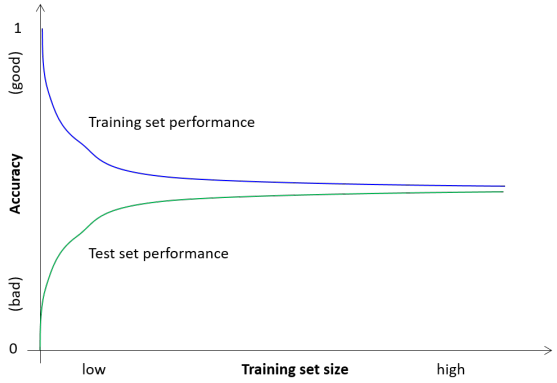


Fig. 2.33: Learning curve analysis

2.6 Services Maps and Product Maps

Services maps and product maps are diagrams that I present for each AI area in the respective chapter of this book. What are those maps and what can they be used for?

A *services map* depicts groups of functionality that AI products offer. For machine learning you may, e.g., use an ML library for programming ML applications or an ML development environment

for graphically orchestrating ML applications. So, “ML library” and “ML development environment” are services for machine learning.

Different tools and tool suites (commercial and open source) offer different services that often overlap. A *product map* maps the services to specific products. For example, TensorFlow is an ML library. In fact, the product map depicts a table with tools as rows and services as columns.
For each product map, such a table can be found in the appendix of this book.

Use of Services Maps and Product Maps

You may use services maps and product maps for selecting suitable products for an AI application development project. I recommend the following steps. See Fig. 2.34 for a method for selecting AI products.

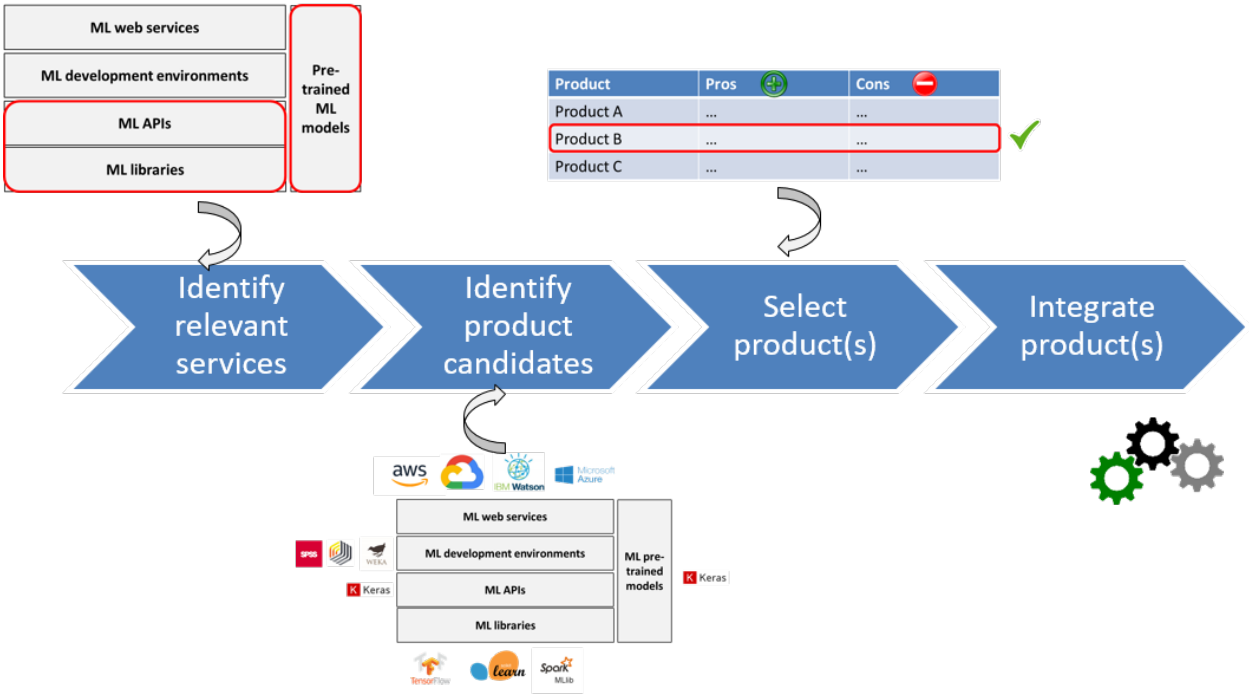


Fig. 2.34: A method for selecting AI products

1. *Identify relevant services*: For a concrete project, mark the services which are relevant. For example, in an application domain where there are pre-trained models available, they may be relevant.
2. *Identify product candidates*: Consult the product map and retrieve products that cover the relevant services. Those are potential product candidates.
3. *Select product(s)*: Now select a product or a set of products (best-of-breed approach). This step, of course, requires a lot of expertise. In many organizations there are best practices for

product evaluation processes. Usually, evaluation criteria are first defined and then the product candidates are evaluated against those criteria. If there are too many candidates, then a shortlist is prepared first.

4. *Integrate product(s)*: Integrate the selected product(s) into your AI application.

Note: The integration of different products can be expensive. If there is a tool suite which matches all requirements, favor it over a best-of-breed solution. On the other hand, try to avoid vendor lock-in. Use products with open, standardized interfaces which can be replaced later if needed.

ML Services Map

Fig. 2.35 gives an overview of service categories of ML products.

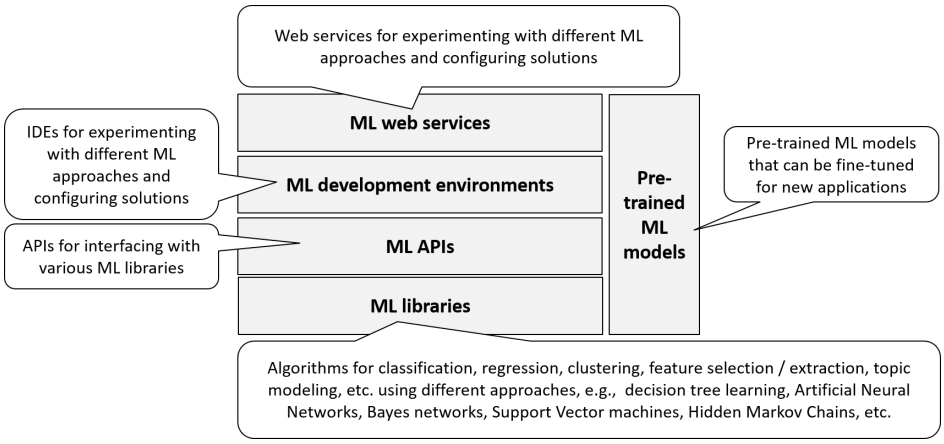


Fig. 2.35: ML services map

ML libraries provide algorithms for classification, regression, clustering, feature selection / extraction, topic modeling, etc. using different approaches, e.g., decision tree learning, artificial neural networks, Bayesian networks, inductive logic programming, support vector machines, hidden Markov chains, etc. They are implemented in a certain programming language, e.g., Python, Java or C/C++, and can be used in AI applications that are implemented in a compatible programming language.

ML APIs provide an ML programming interface to various ML libraries, e.g., Keras running on top of TensorFlow, CNTK, or Theano.

ML Development Environments allow experimenting with different ML approaches, testing the performance, and configuring solutions. Some of them have visual programming features to configure ML processing steps. They usually allow exporting solutions that can then be included in an AI applications as libraries.

ML web services provide similar functionality as ML development environments, but need not be installed locally. Instead, they can be used via the Web. This implies the necessity for uploading the ML data sets to some cloud storage.

Finally, *ML pre-trained models* can be used in ML libraries, APIs, development environments and web services to perform transfer learning.

ML Product Map

Fig. 2.36 shows an overview of ML products, each assigned to the respective service category.

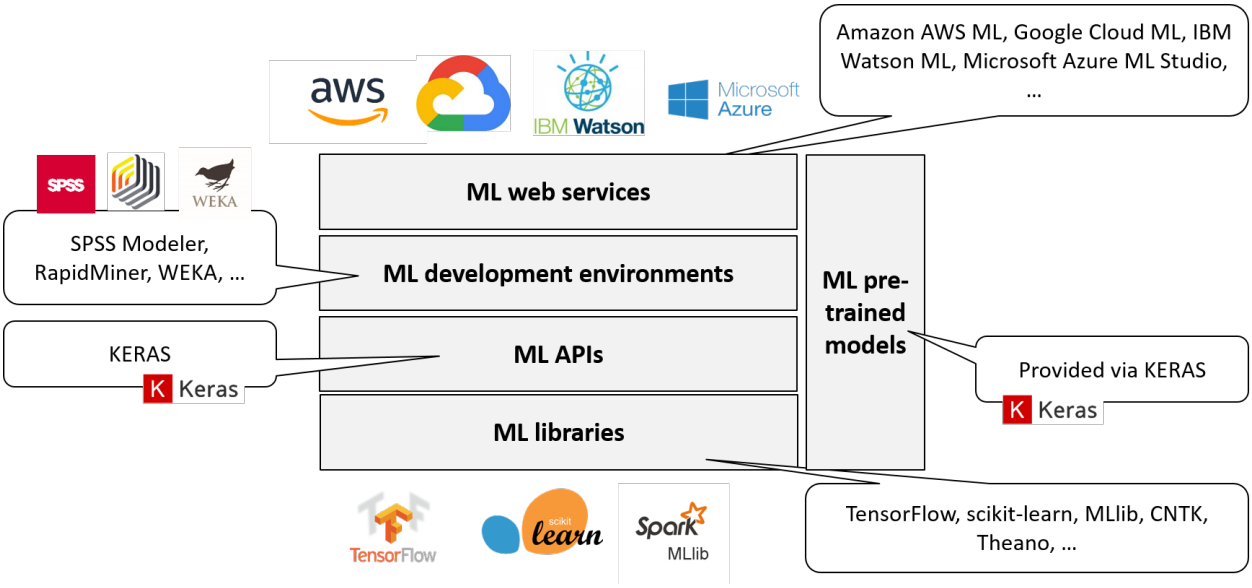


Fig. 2.36: ML product map

Examples for ML libraries are TensorFlow[14], scikit-learn[15], MLlib[16], CNTK[17], and Theano[18]. Keras[19] is an example for an ML API. Examples for ML development environments are SPSS Modeler[20], RapidMiner[21], and WEKA[22]. Examples for ML web services are Amazon AWS ML[23], Google Cloud ML[24], IBM Watson ML[25], and Microsoft Azure ML[26]. Keras[27] also bundles ML pre-trained models like ResNet[28].

[14] https://www.tensorflow.org/
[15] http://scikit-learn.org/
[16] http://spark.apache.org/mllib/
[17] https://docs.microsoft.com/en-us/cognitive-toolkit/
[18] http://deeplearning.net/software/theano/
[19] https://keras.io/
[20] http://www-01.ibm.com/software/analytics/spss/products/modeler/
[21] https://rapidminer.com/
[22] http://www.cs.waikato.ac.nz/ml/weka/
[23] https://aws.amazon.com/de/machine-learning/
[24] https://cloud.google.com/products/ai/
[25] https://www.ibm.com/cloud/machine-learning
[26] https://azure.microsoft.com/de-de/services/machine-learning/
[27] https://keras.io/
[28] https://keras.io/applications/#resnet

2.7 Engineering ML Applications

Methodology

Integrating an ML component in an AI application requires some experience. So summarize, I recommend using the following methodological steps as a guideline. See Fig. 2.37.

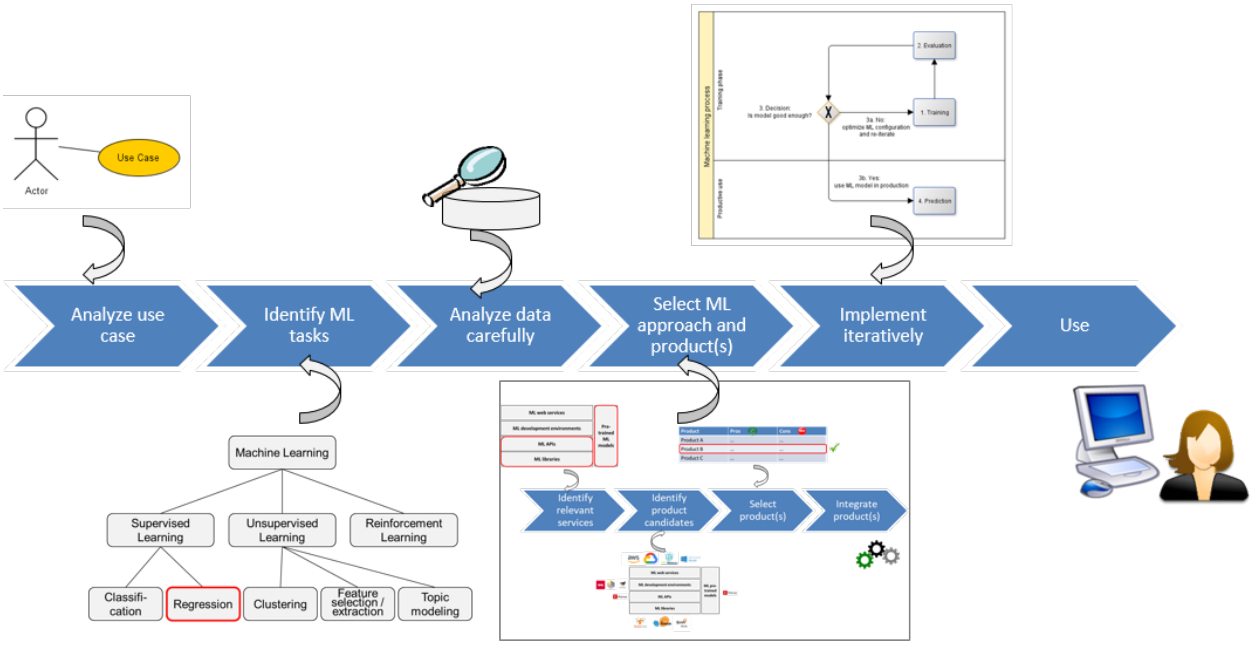


Fig. 2.37: A methodology for developing ML applications

1. *Analyze use case*: As in engineering any IT application, the first step is to carefully analyze the use case, i.e., the stakeholders, the goals of the application to be implemented and the users' requirements.
2. *Identify ML tasks*: Based on the user requirements, the relevant ML tasks should be identified, e.g., classification, regression, topic mining, etc.
3. *Analyze data carefully*: As in all data-intensive tasks, it is most important to intensively work with the data sets at hand. It is essential to understand the meaning of the data entities and their attributes in order to develop suitable ML applications. Statistics and unsupervised ML may be used to better understand the data.
4. *Select ML approach and product(s)*: Which of the various ML approaches are appropriate for solving the task: decision trees, artificial neural networks, support vector machines, Bayesian networks, ...? Which products are best suited? See above the method for selecting a product for a certain application use case based on the services map and product map.

5. *Implement iteratively*:
    1. Often, the raw training data needs to be pre-processed, e.g., unnecessary attributes need to be removed (e.g., via feature selection), attributes combined (e.g., via feature extraction), values normalized, data records with quality problems removed, etc. See also Section 4.3 for a description on semantic ETL.
    2. Then, the pre-processed data set is used for training the ML model.
    3. Before using the ML model in production it should be validated in order to measure the expected prediction performance.
    4. If the prediction performance of the ML model is not sufficient, then pre-processing should be considered or the ML configuration should be adapted. Incrementally, training and validation is performed again. This iterative process comes to an end when a sufficient accuracy is reached.
6. *Use*: Finally, the ML application may be used productively. If, during productive use, a human validation of the results is planned, then the manually corrected results may be used for re-training the ML model.

Caveats: Biased ML

ML applications exhibit a lack of robustness (see e.g. Baeza-Yates, 2018 and Mueller-Birn, 2019). Firstly, training data can only reflect a subset of situations in which the ML application is used in practice. Secondly, training data is historical and does not necessarily reflect current circumstances. Additionally, there are various forms of biases influencing ML applications. See Fig. 2.38.

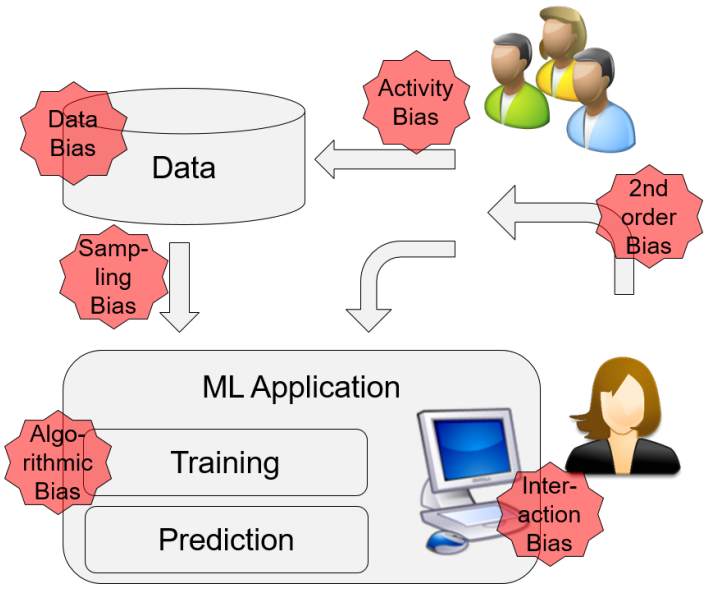


Fig. 2.38: Bias in ML applications (adopted from Baeza-Yates, 2018)

In the sections above we discussed *algorithmic bias*, e.g. using too simple or too complex models leading to underfitting or overfitting. However, there are many other sources of bias in the life cycle of an ML application.

Most problematic is *data bias* since ML applications are data-driven. Increasingly, there are examples of data bias in ML applications reported in media (also see O'Neil, 2016, Mueller-Birn, 2019). People with an Afro-American background in the US were systematically disadvantaged by software for predictive police work and legal risk assessment in court. Software for assigning students to schools in the US discriminated against children from low-income families. Research has shown that the use of facial recognition software in public places leads to systematic misjudgments. So, discrimination in the past is perpetuated in the future.

Closely related to data bias is *activity bias*, particularly if the data for training ML applications is based on crowd-sourced web data. Only a very small percentage of web users generate the vast majority of content, including deliberate misinformation a.k.a. "fake news". Data used for training ML applications is often called "ground truth" but it is important to note that this does not necessarily reflect the truth.

Also closely related to data bias is *sampling bias*. Many medical applications, e.g., rely on data from western, educated, industrialized, rich and democratic societies (WEIRD). WEIRD societies represent as much as 80 % of study participants but only 12 % of the world's population, i.e., they are not representative of humans as a species.

Algorithmic bias we discussed in detail in the sections above, including means of detecting and handling. Also relevant is *interaction bias*, induced by the user interface (UI) of the ML application and the way users interact with it. The way in which ML predictions are presented in a UI and how well they are explained influences how users understand them. People frequently believe that AI applications exhibit human-like intelligence. They assume that good performance in one context assures reliability in another, which is not necessarily the case - depending how the application is trained.

Bias begets bias. This feedback effect is called *second order bias* and may lead to a vicious cycle. In some ML applications, user feedback is directly fed back to training. In other situations the use of ML applications influences opinion which leads to new crowd-sourced data which indirectly influences ML applications.

How do deal with bias when developing ML applications?

First of all it is important to be aware of the problem. Data bias is outside the sphere of influence of engineers of ML applications. However, data sampling usually is. Discussing and evaluating data bias with domain experts is a first step. As shown in the sections above, there are methods for dealing with algorithmic bias. However, rarely discussed in literature is the role of the UI in presenting predictions of an ML application. Explaining the prediction criteria and indicating the confidence in predictions will help humans to better deal with recommendations generated by a machine.

2.8 Quick Check

Answer the following questions.

1. What is ML?
2. When to use ML / when not to use ML?
3. Name applications of ML.
4. Explain the following ML areas: supervised learning, unsupervised learning, reinforcement learning.
5. Explain the following ML tasks: classification, regression, clustering, feature selection / extraction, topic modeling
6. Explain the following ML approaches: decision tree learning, Artificial Neural Networks, Bayes networks, deep learning. Name other approaches.
7. Explain the process for supervised ML. How to use training data and test data?
8. Explain the measures accuracy, precision, recall, F1 score, MAE, RSME and MSE
9. Explain k-fold cross-validation
10. What is bias and variance?
11. Explain overfitting and underfitting
12. What are learning curves? How to interpret them?
13. How to increase / decrease model complexity?
14. How does learning curve analysis work for classification (instead of regression)?
15. Explain the concepts of services maps and product maps. How can they be used for selecting products in a project?
16. Explain the main services of ML
17. Name prominent products for each ML service
18. How to proceed in engineering an ML application (methodology)?
19. What are sources of bias in ML applications and how to deal with them?

Assignments

Kaggle[29] is a prominent provider for ML competitions. Data sets are often provided by companies who also pay a price for the best submission. Submissions are ranked according to the prediction performance achieved.

1. Participate in the Kaggle classification competition Titanic: Machine Learning from Disaster[30]. You will have to register at Kaggle to participate in the competition. You find the data (train.csv, test.csv, and as a submission example gender-submission.csv) as well as a description of the fields and the goal of the competition on the website. You can use any ML tool you like, e.g., RapidMiner. If you want to program your solution in Python you may follow a tutorial[31]. Submit your solution to Kaggle and compare yourself with other committers.

[29] https://www.kaggle.com
[30] https://www.kaggle.com/c/titanic/
[31] https://www.kaggle.com/sashr07/kaggle-titanic-tutorial

2. Participate in the Kaggle regression competition Housing Prices Competition for Kaggle Learn Users[32] and proceed as in the first assignment. If you like to program your solution in Python, you may follow the tutorial[33].

[32] https://www.kaggle.com/c/home-data-for-ml-course/overview
[33] https://www.kaggle.com/dansbecker/basic-data-exploration

3. Knowledge Representation

AI applications are based on knowledge. Therefore, a central question to all AI applications is how to represent the knowledge relevant for the application domain.

Fig. 3.1. shows *knowledge representation* in the AI landscape.

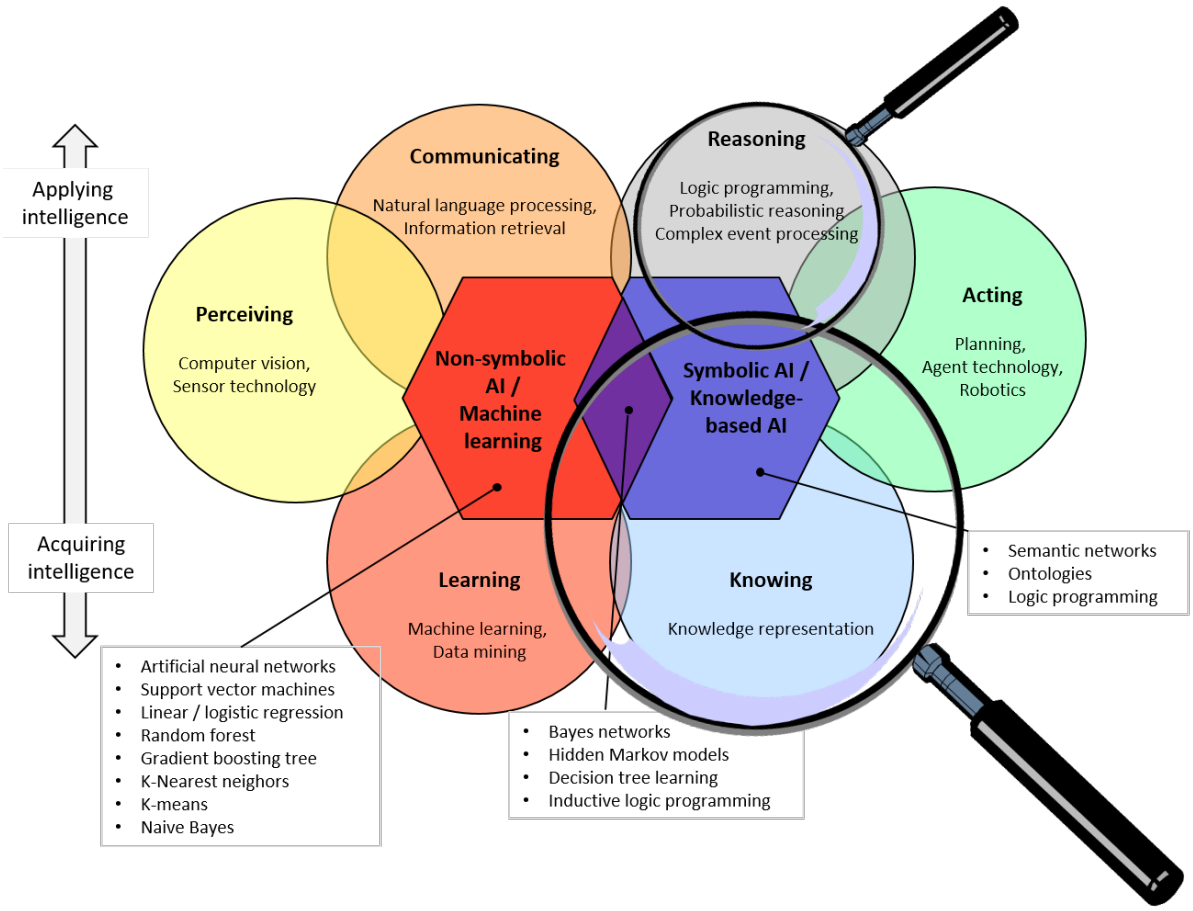


Fig. 3.1: Knowledge representation in the AI landscape

Knowledge representation is part of symbolic AI and deals with the abilities of knowing and reasoning.

Inspired by one of my own projects, the digital collection[1] of one of the leading arts museums in Germany, I will use arts as the sample application domain for this chapter and the remainder of the

[1] https://sammlung.staedelmuseum.de

book.

See Fig. 3.2.

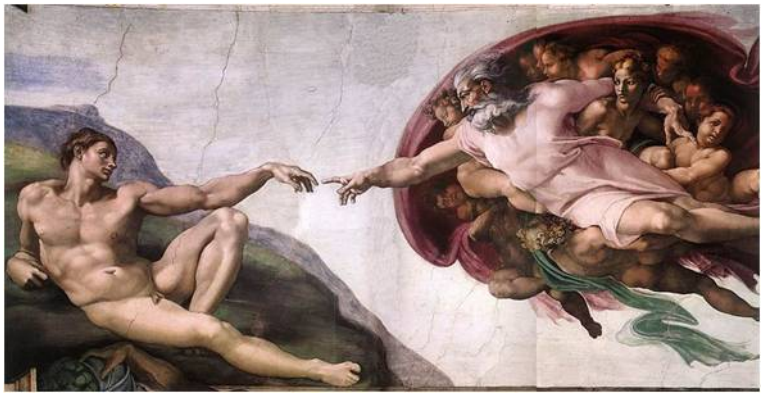


Fig. 3.2: Michelangelo (1475 – 1564): Creation of Adam, Sistine Chapel, Rome, Italy

Relevant questions for knowledge representation are:

- How to *represent knowledge*, e.g., "Michelangelo is a Renaissance artist"
- How to *reason over knowledge*, e.g., "Someone who has created paintings is an artist"
- How to *answer questions*, e.g., "Which Renaissance artists lived in Italy?"

3.1 Ontology

In this book, I use the term "ontology" for represented knowledge with the following definition.

An *ontology* is the representation of knowledge of a particular application domain, i.e., a *representation of the relevant concepts and their relationships*.

The term "ontology" was originally coined in philosophy and is now also used in Computer Science. See, e.g., (Busse et al., 2015). Let us look at some specific examples of knowledge about creative arts (Fig. 3.3).

- Michelangelo was an Italian artist.
- He created many paintings, e.g., the „Creation of Adam" in the Sistine Chapel, Rome.
- He also created sculptures like David.
- He belonged to the artistic movement of Renaissance.
- People who create paintings are painters.
- Painters are artists.

Fig. 3.3: Examples of arts facts

Those pieces of knowledge are represented informally as English sentences. Parts of those sentences are marked with different colors. The terms "artist", "painting", "sculpture", and "artistic movement"

are marked red (solid lines). They represent concept types a.k.a *classes*. The terms with dashed red lines are concept instances a.k.a. *individuals*: "Michelangelo" is an artist, "Creation of Adam" is a painting, "David" is a sculpture, and "Renaissance" is an artistic movement. Marked in blue are *relationships*: Michelangelo "created" David as well as Creation of Adam and he "belonged to" the High Renaissance movement. Finally, marked in green are general *rules*: Every individual who has created a painting is a painter; every individual belonging to the class painter also belongs to the class artist.

See Fig. 3.4.

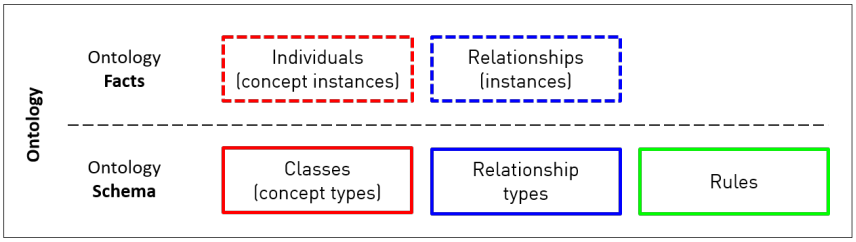


Fig. 3.4: Ontology facts and ontology schema

An ontology consists of *facts* and an ontology *schema*. Facts are based on the schema. Facts are individuals (concept instances, e.g., Michelangelo) and concrete relationships between those individuals (e.g., Michelangelo created David; dashed lines in the figure). The schema specifies the type level (solid lines in the figure). Classes are the concept types for individuals (e.g., artist for Michelangelo). Relationship types, e.g., "created" define the kinds of relationships that can be specified between individuals. Finally, *rules* are part of an ontology schema.

An ontology of some form is in the center of many AI applications. Extending the ontology, i.e., adding new concepts, means extending the AI application.

Ontology reasoning

An ontology is more powerful than a traditional relational database. In addition to the knowledge that is stated explicitly, knowledge may be derived via *reasoning*.

Consider the following explicitly stated facts and rules:

- Michelangelo is a person.
- Michelangelo has created the painting „Creation of Adam".
- Every person that has created a painting is a painter.

From this explicitly stated knowledge, the following fact may be derived via reasoning:

- Michelangelo is a painter.

I leave it to the reader to find an explanation why the reasoning result is, in fact, correct. Take a while to think about it. What is the intuitive, common sense, explanation? Can you also formulate the explanation more formally? How would a computer program (a reasoning engine) have to be constructed in order to derive the new fact automatically?

Implementing a reasoning engine is beyond the scope of this book. This book focuses on engineering AI applications using off-the-shelf components like a reasoning engine (Much like a book on engineering JavaEE applications will not explain how the Java compiler is implemented). For further reading refer, e.g., to (Russell and Norvig, 2013).

Ontology Querying

Knowledge representation is no end in itself but a means to an end: answering relevant questions of the application domain. Therefore, ontology engines include a query interface that allow to formulate queries and retrieve results.

Consider the following questions and answers regarding the facts specified above:

- Q1: Who has painted “Creation of Adam”? A1: Michelangelo
- Q2: Has Michelangelo painted “Creation of Adam”? A2: Yes
- Q3: Is Michelangelo a painter? A3: Yes
- Q4: Which painters exist? A4: Michelangelo (and more painters, according to the facts specified)

The four questions are of different nature.

Q1 (Who has painted “Creation of Adam”?) can be answered by matching an explicitly stated fact (Michelangelo painted „Creation of Adam“) and returning the matching individual (“Michelangelo”).

Q2 (Has Michelangelo painted “Creation of Adam”?) is similar but expects a Boolean answer (here: Yes).

Q3 (Is Michelangelo a painter?) also expects a Boolean answer but cannot be answered by simply matching explicitly stated facts. Instead, ontology reasoning is required to derive the answer: Michelangelo is a person and painted „Creation of Adam“; every person who has created a painting is a painter ⇒ Michelangelo is a painter. Hence, the answer is yes.

Q4 (Which painters exist?) also involves reasoning but expects a set of individuals as an answer. In case the ontology only contained the facts mentioned above then the result would, indeed, be just {Michelangelo}. If, however, paintings by Raphael, Leonardo da Vinci, etc. were listed then the set would contain those artists as well.

Open versus Closed World Assumption

Q4 (Which painters exist?) leads to an interesting question: How to ensure that the ontology is complete, i.e., actually contains all painters?

In an ontology for our solar system it is relatively easy to list all planets (even though the definition of what is a planet may change over time: Pluto is, today, considered a dwarf planet only). In contrast, for the Art Ontology it is next to impossible to list all painters that ever lived. Many of them are not known anymore. And who should decide who was a "real" painter and who just scribbled a few sketches?

Other critical questions are counting questions (How many painters exist?) and negated questions (Which painters have not created a sculpture?). The answers to those questions may be wrong if the ontology is incomplete – even if all facts in the ontology are, indeed, correct.

Therefore, the correct interpretation of ontology query results depends on assumptions that we make on the completeness of the ontology[2].

In the *Closed World Assumption (CWA)* it is assumed that the ontology is complete. This is a comfortable situation and makes the interpretation of query results easy. Assume e.g., that the Art Ontology is restricted to one particular museum and is complete (closed world assumption). In this case, the answer to Q4 (Which painters exist?) may be "There are exactly the painters X,Y,Z for which we currently have paintings in our museum"; the counting query (How many painters are there?) may be answered "There are exactly n painters currently exhibiting in our museum"; the answer to the negated query (Which painters have not created a sculpture?) can be answered as "For painters X,Y,Z we have currently no sculpture in our exhibition".

In the *Open World Assumption (OWA)* it is assumed that the ontology is not complete and new findings may be added as facts at any time. This does not mean that questions about enumerations, counting, and negations cannot be asked at all. However, the results must be interpreted differently than in the CWA. For example, the result to Q4 (Which painters exist?) can be interpreted as "painters that we know of …" instead of "all painters …"; The result of the counting question (How many painters are there?) can be interpreted as "there are at least n painters that we know of"; the answer to the negated question (Which painters have not created a sculpture?) may be interpreted as "for painters X,Y,Z there is currently no sculpture known".

The distinction between OWA and CWA is not technical - it is a matter of business application logic! It is about interpreting the results of an ontology query adequately. When developing an AI application it is essential to figure out under which assumption the ontology has been constructed in order to interpret results adequately.

[2]The situation is, of course, further complicated if we even cannot assume that the specified facts are correct. There are various approaches for dealing with uncertainty (see (Russell and Norvig, 2013)). In practice, it is advisable to try to avoid this situation by quality assurance measures if at all possible.

3.2 Knowledge Representation Approaches

In the last sections, I introduced ontologies, ontology reasoning and querying informally. In order to use knowledge in an AI application one needs concrete formalisms and implementations of such formalisms. In this section, I briefly mention important knowledge representation formalisms. They have been developed over the years in AI research and they are, today, more or less used in practice. For more details see (Russell and Norvig, 2013).

Predicate Logic

Predicate logic is a mathematical formalism which is the foundation of many knowledge representation formalisms. Facts and rules are specified in form of predicates, quantifiers and Boolean operations. In the following example, `painted(p, x)` and `painter(p)` are predicates; The universal quantifier ("for all", denoted as an inverted letter "A") and the existential quantifier ("there exists", denoted as a rotated letter "E") are used; As a Boolean operation, the implication ("if … then", denoted as an arrow) is used.

See Fig. 3.5.

painted (Michelangelo, CreationOfAdam)
$\forall$p: ($\exists$ x: painted (p, x)) $\rightarrow$ painter (p)

Fig. 3.5: Predicate logic

Interpretation: Michelangelo painted "Creation of Adam"; Every person who painted something is a painter.

Frames

Frames is a knowledge representation mechanism which influenced early object-orientation and, hence, is familiar to most software engineers today.

See Fig. 3.6 for an example.

Michelangelo
Type: Person
born: 1475
in: Italy

CreationOfAdam
type: Painting
artist:Michelangelo
…

Fig. 3.6: Frame example

Interpretation: Michelangelo is a Person, born 1475 in Italy. "Creation of Adam" is a painting by Michelangelo. In frame systems, rules and reasoning can be implemented algorithmically in form of if-needed / if-added actions.

Semantic Nets

Semantic nets have been popular because of their intuitive graphical representation. In Fig. 3.7, individuals are represented as oval shapes, and relationships as arrows.

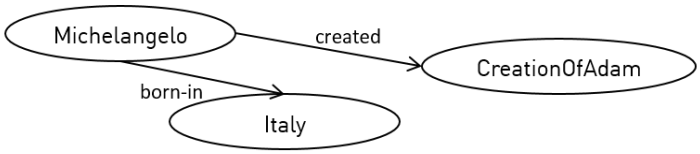


Fig. 3.7: Semantic net example

Interpretation: Michelangelo was born in Italy and created the painting "Creation of Adam".

Rules

In rule-based languages, knowledge is expressed in form of facts and rules. In the following example (Fig. 3.8), `is_a` and `painted` are predicates, `?p` and `?x` are variables, `person` and `painter` are classes, and `->` denotes an implication. The two conditions on the left side of the implication are implicitly conjunctive, i.e., connected by the Boolean AND operator.

```
(?p is_a person)
(?p painted ?x)
->
(?p is_a painter)
```

Fig. 3.8: Rules

Interpretation: If ?p is a person who painted something then ?p is a painter.

General-Purpose Data Stores

General-purpose data stores are rarely mentioned in AI literature but commonly used in AI applications in practice. Examples are relational databases and NoSQL databases including object databases, graph databases, key-value stores, search indexes and document stores.

I have co-edited a book on Corporate Semantic Web (Ege et al., 2015) in which 18 AI applications are presented which are in corporate use. More than half of the applications use general-purpose data stores for representing application knowledge. The architects of those applications made this decision due to performance reasons, ease of use, and for better integration into the corporate IT application landscape.

3.3 Semantic Web Standards

In the following sections I introduce RDF and SPARQL as sample languages and technologies for knowledge representation. Those languages have been standardized by the World Wide Web Consortium (W3C) as part of the Semantic Web initiative[3] and have gained some use in practice. I use those languages for examples in this book. However, neither the W3C standards nor other knowledge representation mechanisms and technologies can be considered as *the* de facto standard in today's AI applications.

Resource Description Framework (RDF)

RDF[4] stands for Resource Description Framework. RDF as well as the languages RDFS[5] and OWL[6] built on top of RDF are based on predicate logic and semantic nets.

I will explain RDF briefly. For a good, practice-oriented introduction see (Allemang and Hendler, 2011).

Resources and URIs

An *RDF resource* represents anything which is relevant in an application domain, e.g., the individual "Michelangelo", the class "Painter", the relationship type "created", etc.

In RDF, every resource is identified by a *Uniform Resource Identifier* (URI[7]).

Example: Wikidata's entry for Michelangelo

```
<https://www.wikidata.org/entity/Q5592>
```

RDF Namespaces

Qualified names introduce namespaces to URIs in order to reduce typing effort, to increase readability, and to avoid name clashes. With the definition of a prefix

```
@prefix wd: <http://www.wikidata.org/entity/> .
```

the URI above can briefly be specified as:

[3] http://www.w3.org/2001/sw/
[4] http://www.w3.org/RDF/
[5] https://www.w3.org/2001/sw/wiki/RDFS
[6] https://www.w3.org/2001/sw/wiki/OWL
[7] http://www.w3.org/TR/webarch/#identification

```
wd:Q5592
```

A default namespace can be specified as follows:

```
@prefix : <http://www.wikidata.org/entity/> .
```

Then, the URI above can be specified without prefix at all:

```
:Q5592
```

Note that the URI does not need to be a cryptic ID but can also be a symbolic name, e.g.

```
:Michelangelo
```

RDF Triples

The main construct for specifying facts in RDF is a *triple* consisting of *subject, predicate*, and *object*. Subject and predicate must be RDF resources. The object may be an RDF resource but may also be a literal value in form of an XML data type[8].

Examples:

```
wd:Q5592 rdfs:label "Michelangelo" .
wd:Q5592 rdf:type :person .
wd:Q5592 :date_of_birth 1475 .
wd:Q5592 :movement wd:Q1474884 .
```

The namespaces for this and the following examples are:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix wd: <http://www.wikidata.org/entity/> .
@prefix : <http://h-da.de/fbi/artontology/> .
```

In the first example triple, the subject is `wd:Q5592`, the predicate is `rdfs:label` and the object is `"Michelangelo"`. Every triple must be terminated by a period.

Fig. 3.9 shows the example triples displayed as a semantic net.

[8] http://www.w3.org/TR/webarch/#formats

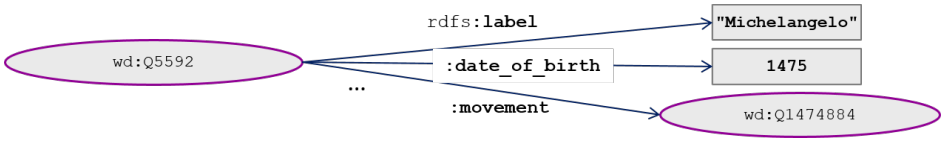


Fig. 3.9: RDF triples displayed as a semantic net

The triples may be read as: Michelangelo is a person who was born 1475 and belongs to the artistic movement of Renaissance (wd:Q1474884).

To avoid repeating the subject `dbpedia:Michelangelo` three times, an abbreviated notation may be used. The subject is stated only once and the triples with the same subject are combined via semicolons. A period terminates the group of triples.

The following example is semantically identical to the three triples above.

```
wd:Q5592 rdfs:label "Michelangelo" ;
         rdf:type :person ;
         :date_of_birth 1475 ;
         :movement wd:Q1474884 .
```

Classes

Classes represent concept types and are defined by RDF triples with `rdf:type` as predicate and `rdfs:Class` as object.

Example:

```
:person rdf:type rdfs:Class .
```

Meaning: Person is a class.

You can also use `a` as an abbreviation for `rdf:type`. So, the triple above can be written as

```
:person a rdfs:Class .
```

Individuals, i.e. concept instances, are specified by RDF triples with `rdf:type` as predicate and the class as object.

Example:

```
wd:Q5592 a :person .
```

Meaning: Michelangelo is a person.

Properties

A relationship type is defined as an RDF property.

Example:

```
:date_of_birth a rdf:Property .
```

Meaning: `date_of_birth` is a property (relationship type).

An RDF property can be used as a predicate in an RDF triple.

```
wd:Q5592 :date_of_birth 1475 .
```

Meaning: Michelangelo's date of birth is in 1475.

RDF and Object-Orientation

Classes and properties of RDF (and RDFS / OWL built on top of RDF) have some of similarities to classes and associations in object-oriented design and programming. However, there are also fundamental differences. In object-orientation, every instance belongs to exactly one class. In RDF, there is no need for specifying a class and an `rdf:type` relationship. Zero, one, but also several `rdf:type` relationships may be specified. Furthermore, there is no type checking for subjects, predicates and objects.

For details see, e.g. (Allemang and Hendler, 2011).

Other Serialization Syntaxes

The RDF syntax introduced above is called Turtle[9]. It shall be noted that other RDF serialization syntaxes exist: N-Triples[10] and RDF/XML[11].

Linked Data

Linked Data[12] is an initiative to create, interlink, and share ontologies for a wide range of application areas. Linked data is based on the W3C Semantic Web technologies introduced above. A large number of most comprehensive ontologies has been created and are publicly available – see the Linked Open Data Cloud[13] in Fig. 3.10.

[9] http://www.w3.org/TR/2014/REC-turtle-20140225/
[10] http://www.w3.org/TR/2014/REC-n-triples-20140225/
[11] http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/
[12] http://linkeddata.org/
[13] https://lod-cloud.net

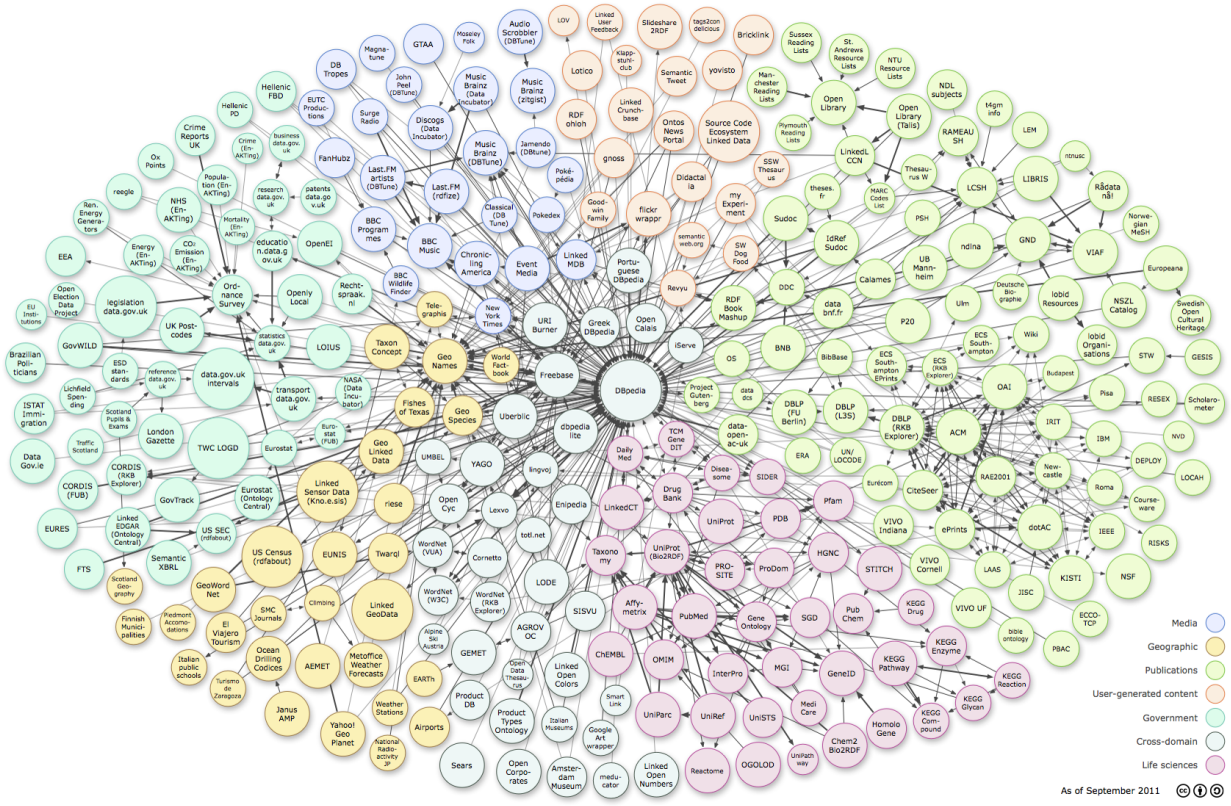


Fig. 3.10: Linked Open Data (LOD) Cloud

Prominent linked data ontologies are WikiData[14], DBpedia[15], and YAGO[16].

Schema.org[17] is an open initiative originally initiated by the major search engine providers Google, Microsoft, Yahoo and Yandex. It is based on RDF and provides a vocabulary for semantically specifying things talked about on web pages, including persons, organizations, locations, products, events, etc. Providers of websites can enrich their human-readable content by machine-readable semantic markup. Google introduced the term *knowledge graph* as Google's collection of such data which is used for semantic features of the search engine, e.g., the Google info boxes. Meanwhile, the term knowledge graph is also used for other large-scale ontologies with a focus on facts, including the Facebook graph and the Microsoft Office graph. Wikidata, DBpedia and YAGO can also be regarded as knowledge graphs in this sense.

Projects like OpenStreetMap[18] and GeoNames[19] use different technology but also follow the idea of linked data.

Linked data is important: Due to the community editing and network effect, the coverage of topics is enormous. However, not every large linked data set is a suitable ontology for a concrete AI

[14] https://www.wikidata.org
[15] http://wiki.dbpedia.org/
[16] https://yago-knowledge.org/
[17] https://schema.org
[18] https://www.openstreetmap.org/
[19] http://www.geonames.org/

application. In section “Tips and Tricks” I give hints on ontology selection and creation.

Example: Art Ontology

In this book, I use an Art Ontology in RDF as an example. The Art Ontology is a custom subset of Wikidata. Wikidata itself is a crowd-sourced community effort to provide the information for Wikipedia’s info boxes and one of the most comprehensive publicly available knowledge graph.

Wikidata provides a crawling API for extracting custom ontologies for AI applications. The art ontology crawler has been developed in one of my research projects (Humm 2020). The crawler code is open source under GitHub[20].

The Art Ontology consists of ca. 1.9 million RDF triples, representing

- ca. 180,000 artworks (paintings, drawings and sculptures), e.g., Mona Lisa
- 27,000 motifs, e.g., mountains
- 20,000 artists, e.g., Leonardo da Vinci
- 7,300 locations, e.g., Louvre Museum
- 720 materials, e.g., oil paint
- 320 genres, e.g., portrait
- 270 artistic movements, e.g., Renaissance.

Ontology Schema

Fig. 3.11 shows the Art Ontology schema as UML[21] class diagram. Please note that RDF does not provide means for specifying required attributes of classes and their datatypes as well as associations between classes. However, the Art Ontology crawler ensures that this schema is met and AI applications using the Art Ontology can rely on it.

[20] https://github.com/hochschule-darmstadt/openartbrowser/blob/master/scripts/data_extraction/art_ontology_crawler.py
[21] http://www.uml.org/

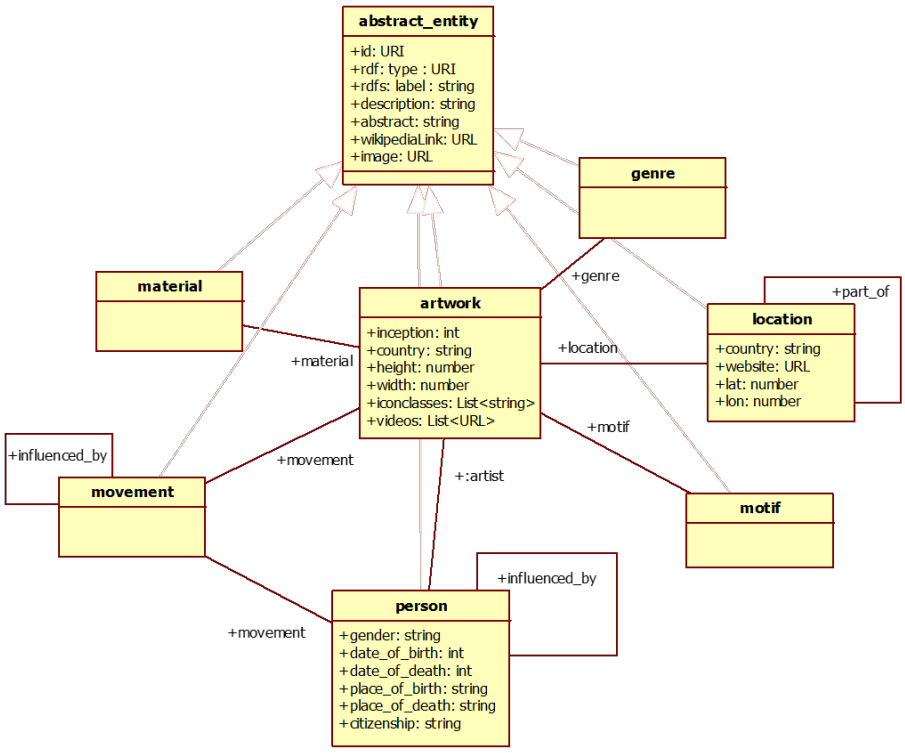


Fig. 3.11: Art Ontology schema

The Art Ontology schema consists of 7 classes. `artwork` is the central class with associations to the other classes: `person`, `movement`, `material`, `genre`, `location`, and `motif`, connecting artworks with their creators, the depicted motifs, the location where the artwork is exhibited, the material, the artistic movement and the genre of the artwork. All classes share common attributes which are denoted in the superclass `abstract_entity`: `id`, `label`, `description`, `abstract`, `image`, and `wikipediaLink`. Some classes have additional attributes, e.g., `gender`, `date_of_birth`, `date_of_death`, `place_of_birth`, `place_of_death` and `citizenship` for class `person`.

This is a subset of the entries for Mona Lisa (Q12418).

```
wd:Q12418 a :artwork;
    rdfs:label "Mona Lisa" ;
    :description "oil painting by Leonardo da Vinci" ;
    :image <https://upload.wikimedia.org/wikipedia/commons/e/ec/Mona_Lisa%2C_by_Leon\
ardo_da_Vinci%2C_from_C2RMF_retouched.jpg> ;
    :artist wd:Q762 ;
    :location wd:Q19675 ;
    :genre wd:Q134307 ;
    :movement wd:Q1474884 ;
    :inception 1503 ;
    :country "France" ;
```

```
12      :height 77 ;
13      :width 53 .
```

The label of the painting is “Mona Lisa”, the artist is Leonardo da Vinci (Q762), the location is the Louvre Museum in Paris (Q19675), the genre is portrait (Q134307), the artistic movement is High Renaissance (Q1474884), the year of inception is 1503, and the size of the painting is 77x53 cm.

Ontology Editor

Protégé[22] is an open source ontology editor. Fig. 3.12 shows as an example, the entry of Mona Lisa in the Art Ontology in Protégé.

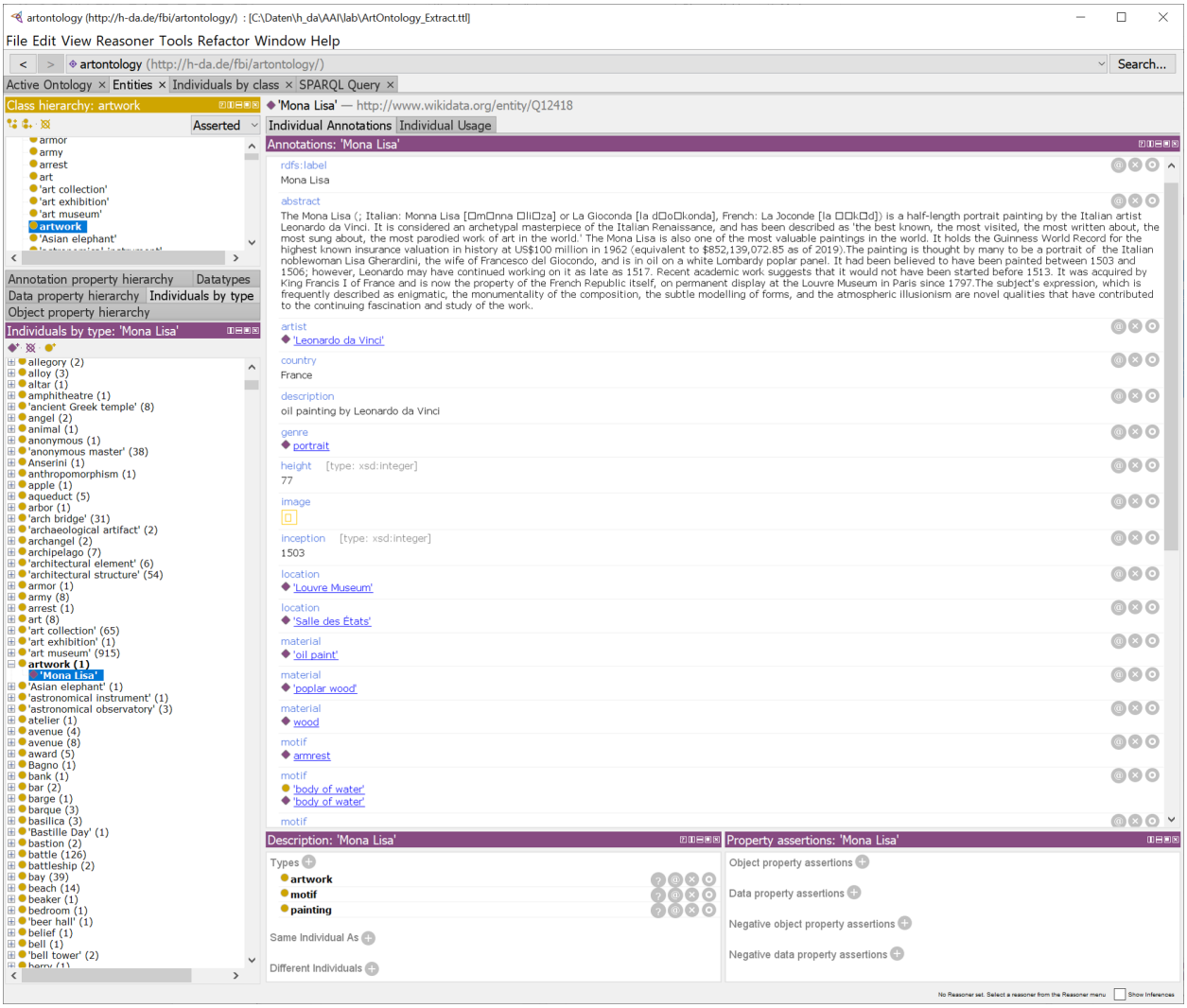


Fig. 3.12: Protégé

[22] http://protege.stanford.edu/

3.4 Querying Ontologies

SPARQL[23] is the query language for RDF and is also standardized by the W3C.

SPARQL Namespaces

SPARQL also supports namespaces but with slightly different a syntax than RDF. See, e.g., some relevant namespaces. The keyword `PREFIX` is used to define a namespace prefix which is separated by a colon from the full namespace.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX : <http://h-da.de/fbi/artontology/>
```

Simple SPARQL Queries

SPARQL uses the keywords `SELECT` and `WHERE` similar to SQL[24]. The query conditions are expressed as RDF triples in which variables can be used. Variables start with a question mark.

See e.g. the following SPARQL query resembling the question "When was Leonardo da Vinci born?"

```
SELECT ?d
WHERE {
   ?p rdfs:label "Leonardo da Vinci";
      :date_of_birth ?d .
}
```

In this query, the variable for holding the date of birth is `?d`.
The query consists of two triples – as you can see, the abbreviated RDF Notation linking several triples with the same subject via a semicolon can be used in SPARQL, too. Each triple poses a restriction on the query – implicitly AND connected. We query for some entry `?p` with label `"Leonardo da Vinci"` and some `date_of_bith` entry `?d`. If such a combination can be found in the ontology then it is returned as the query result.

Fig. 3.13 shows this query and its result executed in the SPARQL server Apache Jena Fuseki[25] after loading the Art Ontology.

[23] http://www.w3.org/TR/sparql11-query/
[24] http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=53681
[25] https://jena.apache.org/documentation/fuseki2/

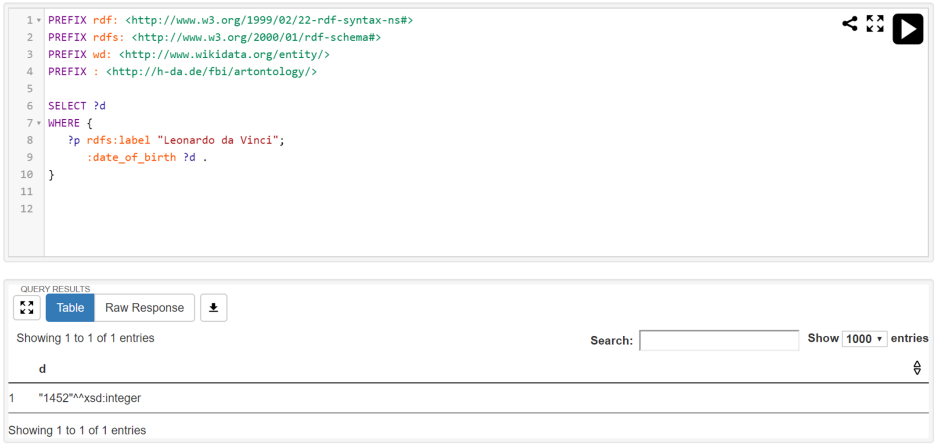


Fig. 3.13: A simple SPARQL query in Fuseki

Since the Art Ontology is an extract of Wikidata, you can execute a similar query at the Wikidata query service[26] (See Fig. 3.14). Since Wikidata uses cryptic IDs for all properties (e.g., `wdt:P569` for `date_of_birth`, there is a query helper suggesting parts of the query to be completed. Try it yourself!

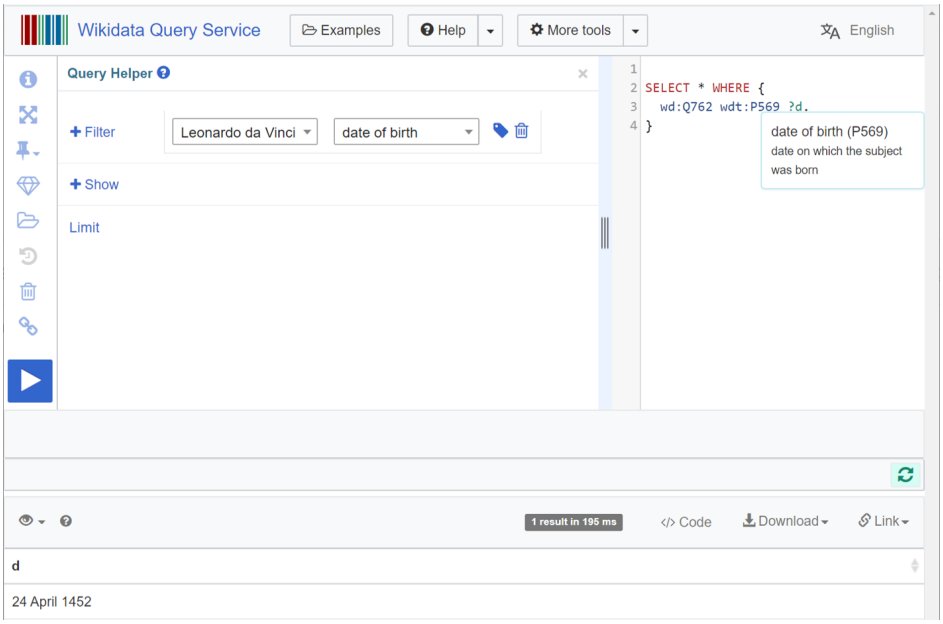


Fig. 3.14: A simple SPARQL query at the Wikidata query service

Assume we are interested in artists who were born in Paris.

[26] https://query.wikidata.org

```
SELECT  ?l
WHERE {
   ?p a :person ;
   :place_of_birth "Paris" ;
   rdfs:label ?l .
```

This query consists of 3 triples. We query for instances of class `person` with `place_of_birth "Paris"` and return their labels. Executed on the Art Ontology, this query will result in 690 artists, including Édouard Manet, Claude Monet, and Jacques Rousseau.

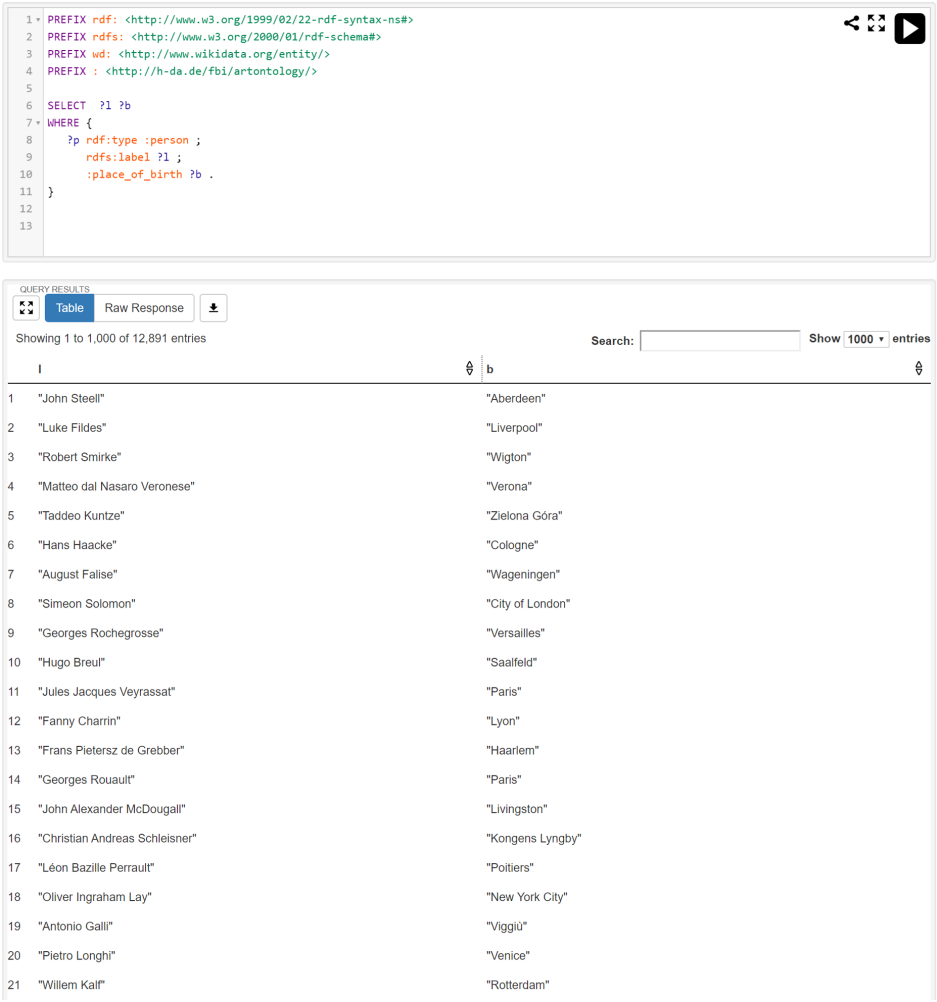
Multiple Query Result Variables

When you are interested in multi-value query results, then multiple result variables may be specified in the `SELECT` clause of a SPARQL query.

The following query, e.g., lists artist with their corresponding place of birth.

```
SELECT  ?l ?b
WHERE {
   ?p a :person ;
      rdfs:label ?l ;
      :place_of_birth ?b .
}
```

The query result is a set of pairs `?l`, `?b`. See Fig. 3.15.



```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX : <http://h-da.de/fbi/artontology/>

SELECT  ?l ?b
WHERE {
   ?p rdf:type :person ;
      rdfs:label ?l ;
      :place_of_birth ?b .
}
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 1,000 of 12,891 entries

Search: Show 1000 entries

	l	b
1	"John Steell"	"Aberdeen"
2	"Luke Fildes"	"Liverpool"
3	"Robert Smirke"	"Wigton"
4	"Matteo dal Nasaro Veronese"	"Verona"
5	"Taddeo Kuntze"	"Zielona Góra"
6	"Hans Haacke"	"Cologne"
7	"August Falise"	"Wageningen"
8	"Simeon Solomon"	"City of London"
9	"Georges Rochegrosse"	"Versailles"
10	"Hugo Breul"	"Saalfeld"
11	"Jules Jacques Veyrassat"	"Paris"
12	"Fanny Charrin"	"Lyon"
13	"Frans Pietersz de Grebber"	"Haarlem"
14	"Georges Rouault"	"Paris"
15	"John Alexander McDougall"	"Livingston"
16	"Christian Andreas Schleisner"	"Kongens Lyngby"
17	"Léon Bazille Perrault"	"Poitiers"
18	"Oliver Ingraham Lay"	"New York City"
19	"Antonio Galli"	"Viggiù"
20	"Pietro Longhi"	"Venice"
21	"Willem Kalf"	"Rotterdam"

Fig. 3.15: Multi-variable query

If all used variables shall be returned then `SELECT *` may be used as in SQL.

Distinct Query Results

As in SQL, `SELECT DISTINCT` avoids duplicates in the result set.

The following query lists countries of artworks in the Art Ontology.

```
SELECT DISTINCT  ?c
WHERE {
   ?p a :artwork ;
      :country ?c .
}
```

Path Expressions

Path expressions are a convenient abbreviation for expressing traversals in RDF graphs. Consider the following query for artworks by Paris-born artists.

```
SELECT  ?n ?l
WHERE {
   ?a a :artwork ;
      rdfs:label ?l ;
      :creator/:place_of_birth "Paris" ;
      :creator/rdfs:label ?n .
}
```

The two conditions that the artwork `?a` has a `creator` and this creator has as `place_of_birth "Paris"` can be conveniently expressed as

```
?a :creator/:place_of_birth "Paris" .
```

Note that path expressions a just convenience. The same condition could have been expressed by 2 triples:

```
?a :creator ?p .
?p :place_of_birth "Paris" .
```

Transitive Closures

SPARQL provides a powerful construct to query transitive closures. Consider the question to find artists that have been influenced by Paul Cézanne, directly or indirectly.

First consider the following SPARQL query.

```
SELECT *
WHERE {
  ?p a :person;
     :influenced_by/rdfs:label "Paul Cézanne" .
}
```

It will return all persons which, according to the facts in the Art Ontology, have directly been influenced by Paul Cézanne: 4 artists, including Pablo Picasso, and Vincent van Gogh.

If we are additionally interested in artists who were influenced by those 4, including the ones influenced by them and so forth (1..n times the `influenced_by` relationship), then the query condition has to be modified as follows.

```
?p :influenced_by+/rdfs:label "Paul Cézanne"
```

The plus sign in `influenced_by+` denotes a transitive closure (1..n). This will result in 15 artists, including Pablo Picasso and Vincent van Gogh as before, but also Salvador Dalí, Paul Klee, Wassily Kandinsky, Franz Marc, and others.

If you want to additionally include Paul Cézanne in the result set (0..n: reflexive and transitive closure), then you have to use a star sign as in `influenced_by*`.

Advanced Features

There are numerous advanced SPARQL features which are important for practical use but which I will not go into details in this book:

- `ASK` queries: for checking a condition (Boolean result)
- `FILTER`: for expressing additional conditions, e.g., on datatypes
- `OPTIONAL`: for specifying optional values
- `EXISTS` / `NOT EXISTS`: for defining negation queries
- `GROUP BY`, `HAVING`: for aggregations
- `ORDER BY`: for sorting query results
- Subqueries: for nesting queries

For those and other features refer to the SPARQL specification[27].

3.5 Rule-based Reasoning

Overview

Reasoning is a major AI technique to derive new knowledge from existing facts in an ontology. An intuitive and, hence, popular way of specifying reasoning behavior is via *rules*. Rules are of the form "IF condition(s) THEN conclusion(s)". Rule-based reasoning has a long history in AI. In particular, the Prolog[28] programming language (PROgramming in LOgic) was, apart from Lisp, the most popular AI programming language in the 1980s.

Two approaches to rule-based reasoning can be distinguished: *forward chaining* and *backward chaining*. In forward chaining, the reasoning engine starts with the facts and applies all specified rules in order to find all possible conclusions. Queries on this extended ontology will include the logic specified in the rules. Backward chaining operates inversely. Starting from a particular query, those rules are applied which are necessary for answering the query. Backward chaining is sometimes called *goal-directed* reasoning whereas forward chaining is called *data-driven* reasoning.

[27] http://www.w3.org/TR/sparql11-query/
[28] https://en.wikipedia.org/wiki/Prolog

Which approach is better? This, of course, depends on the application use case, in particular the types of queries asked and the branching rate of rules. Some technologies provide one strategy only (e.g., Prolog provides backward chaining), others combine forward and backward chaining (e.g., Drools Expert[29]).

For the Semantic Web, there are various approaches for rule-based reasoning, more or less adopted in practice. SWRL (Semantic Web Rule Language)[30] is a W3C Member Submission since 2004 but not a W2C standard and not widely adopted in practice. OWL[31] reasoning is based on set theory. In contrast to rule-based approaches, reasoning logic is specified via set operations like restrictions, intersections, complements, and unions. Some RDF triple store implementations like Apache Jena provide their own rule language like Jena rules[32]. Jena rules provide forward and backward chaining. GraphDB[33] provides proprietary rules with forward chaining.

SPARQL Update[34] provides means for updating ontologies based on existing facts. This mechanism may be used for forward chaining rule-based reasoning. Since SPARQL update statements are an intuitive extension of SPARQL queries and standardized by the W3C, I will present them in the following section.

SPARQL Update

Assume you want to query the Art Ontology for artists who are painters as well as sculptors. However, in the Art Ontology schema there is no direct representation of the concepts painter and sculptor. But you may infer this information from the Art Ontology with the following rules:

1. A person who created a painting is considered a painter.
2. A person who created a drawing is considered a painter.
3. A person who created a sculpture is considered a sculptor.

Those rules can easily been expressed using SPARQL INSERT statements.

[29] https://www.drools.org
[30] https://www.w3.org/Submission/SWRL
[31] https://www.w3.org/TR/2012/REC-owl2-overview-20121211
[32] https://jena.apache.org/documentation/inference
[33] http://graphdb.ontotext.com/documentation/standard/reasoning.html
[34] https://www.w3.org/TR/sparql11-update

```
INSERT {?c a :painter}
WHERE {
  ?a a/rdfs:label "painting" ;
     :artist ?c .
}


INSERT {?c a :painter}
WHERE {
  ?a a/rdfs:label "drawing" ;
     :artist ?c .
}


INSERT {?c a :sculptor}
WHERE {
  ?a a/rdfs:label "sculpture" ;
     :artist ?c .
}
```

SPARQL INSERT statements allow providing RDF triples after the `INSERT` keyword, containing SPARQL variables that are matched according to the conditions specified in the `WHERE` part. The `WHERE` part can contain everything that can be specified in SPARQL queries. In the example above, it is assumed that the Art Ontology contains the Wikidata type information, e.g., `wd:Q3305213` (painting), `wd:Q93184` (drawing) or `wd:Q860861` (sculpture) for artworks. The `WHERE` conditions are straight forward, using path expressions.

When experimenting with SPARQL INSERT statements in the Fuseki web app, make sure that the SPARQL endpoint is set to `update`. See Fig. 3.16.

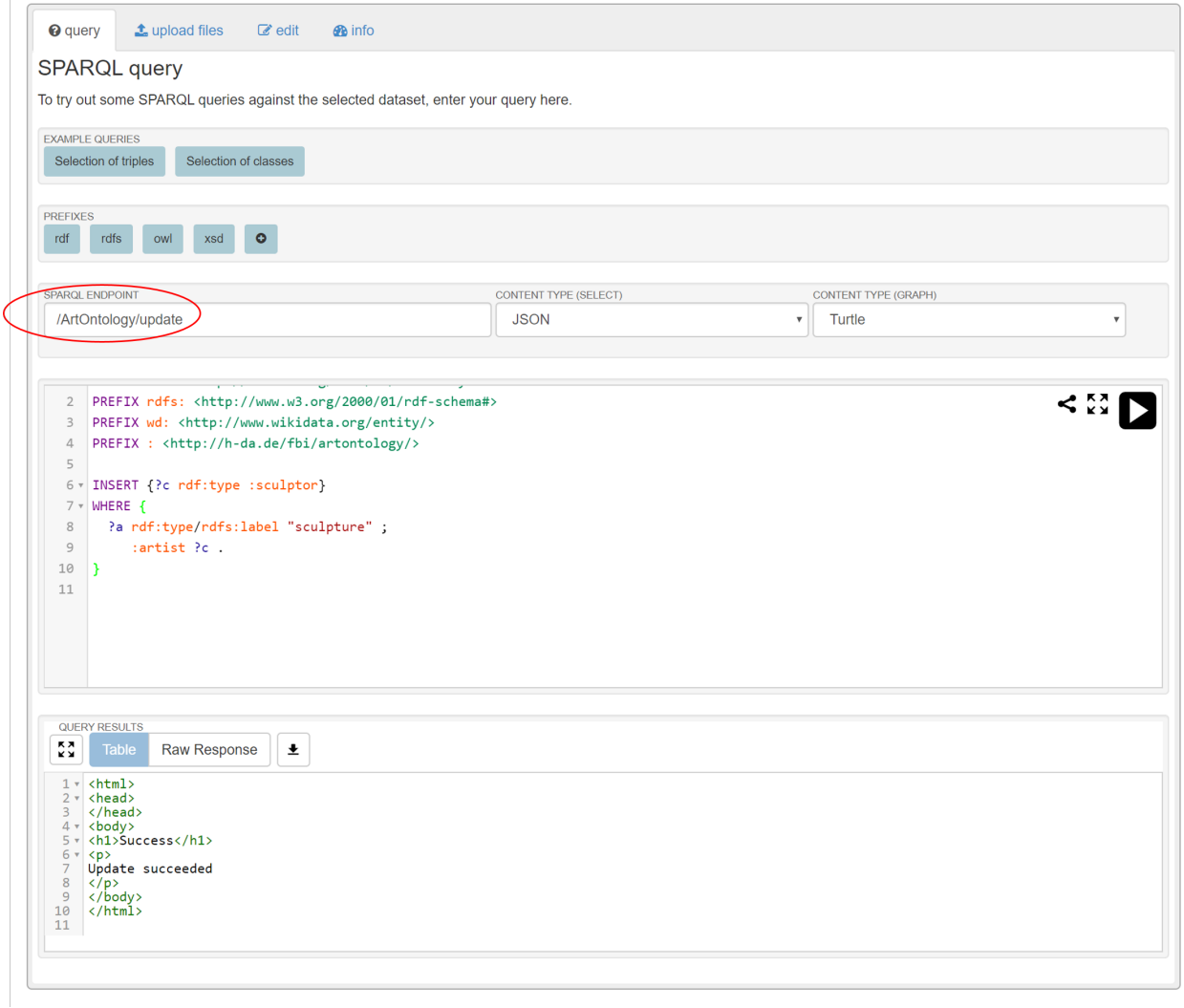


Fig. 3.16: SPARQL INSERT statement

After execution of the update operation, a success message is returned.

After executing the 3 `INSERT` statements specified above, the query for artists who are both, painters and sculptors can be easily and intuitively be specified and executed as shown in Fig. 3.17.

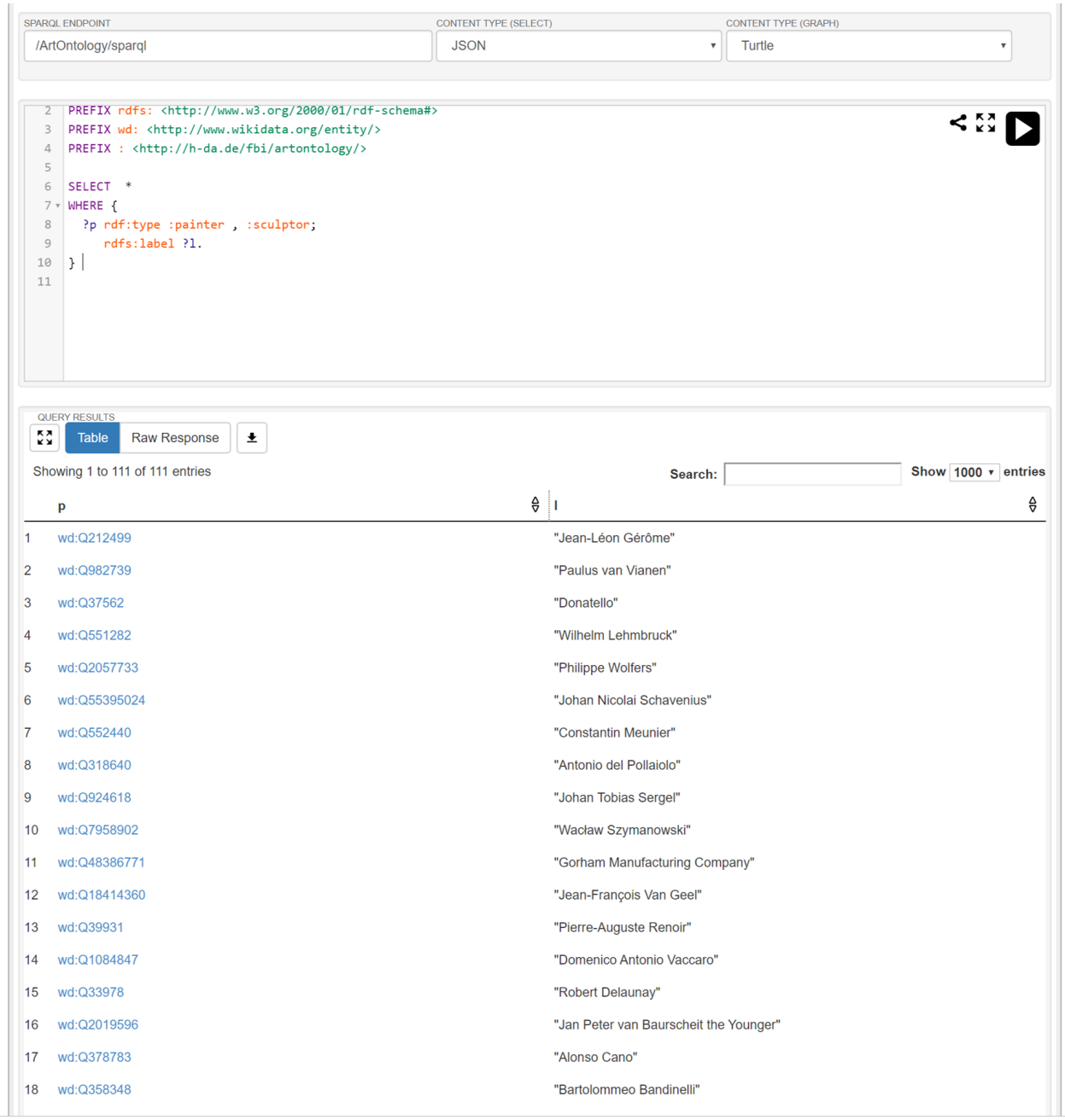


Fig. 3.17: Querying inferred facts: painters and sculptors

In this SPARQL query, both RDF abbreviation notations are used: using a semicolon for chaining several predicates for the same subject and using a comma for chaining several objects for the same predicate.

When you wish to inspect the RDF triples generated, you may use a SPARQL `CONSTRUCT` query instead of an `INSERT` statement. See Fig. 3.18.

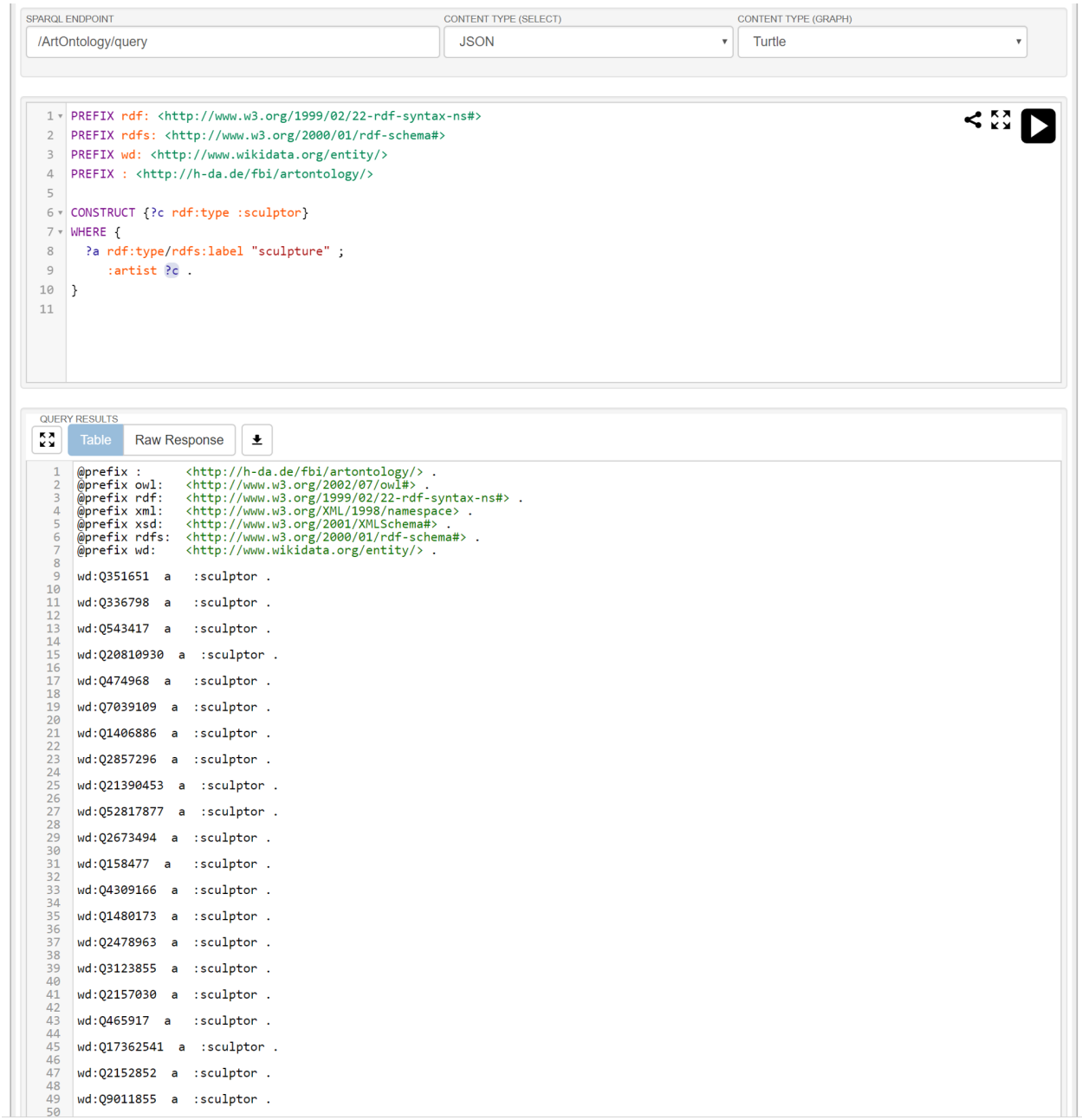


Fig. 3.18: Querying inferred facts: painters and sculptors

Please note that the SPARQL endpoint is set to query, as for other SPARQL queries.

3.6 Knowledge Representation Services and Product Maps

Knowledge Representation Services Map

Fig. 3.19 shows the services map for knowledge representation.

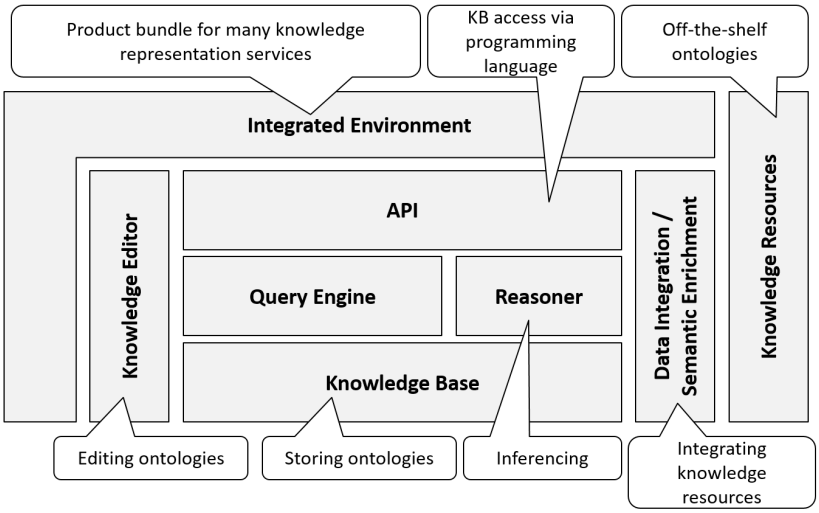


Fig. 3.19: Knowledge representation services map

- A *knowledge base* allows the storage and retrieval of ontologies, i.e. knowledge structures of all kinds. It is usually the core of an AI application.
- *Query engine* and *reasoning engine (a.k.a. reasoner)* usually come with a knowledge base product. But since they can often be plugged in and replaced, I have added them as separate services.
- Since AI applications are often implemented in a general-purpose programming language like Python, an Application Programmers' Interface (*API*) is needed to access to the knowledge base and its reasoners.
- A *Knowledge Editor* allows editing ontologies. Ontologies may be imported / exported between knowledge editor (development time) and knowledge base (run-time). Standard formats like RDF may be used for importing / exporting.
- *Knowledge resources* include off-the-shelf ontologies like, e.g., Wikidata, that may be used in an AI application.
- *Data Integration / Semantic Enrichment* are services that allow integrating various knowledge resources or subsets thereof. For example, the Art Ontology described above is a custom subset of Wikidata.
- *Integrated environments* are tool suites that bundle various development-time and run-time knowledge representation services.

Knowledge Representation Product Map

Fig. 3.20 shows the knowledge representation product map.

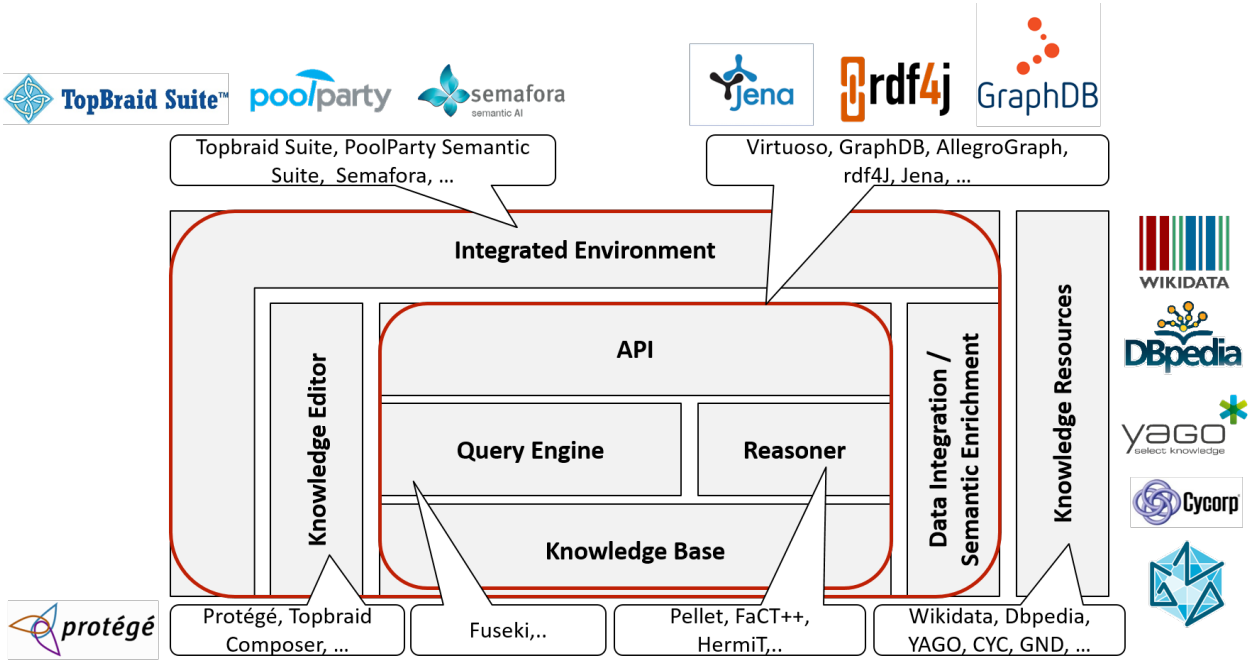


Fig. 3.20: Knowledge representation product map

Virtuoso, GraphDB, rdf4J, Apache Jena are bundles that include knowledge base, reasoning / query engines and Java APIs. Pellet, FaCT++, and HermiT are reasoning engines that may be plugged into other products. Protégé and Topbraid Composer are knowledge editors; Topbraid Suite, PoolParty and Semafora are integrated environments that include knowledge editors and runtime-components. Examples for knowledge resources are WikiData, DBpedia, YAGO, CYC and GND.

More products and details can be found in the appendix.

3.7 Tips and Tricks

Ontology - Make or Buy?

Make or buy? This is common a question in the technology selection phase of an IT project. It also applies to the ontology in an AI application project.

As outlined in the section on Linked Data, there are thousands of ontologies available, many with a public license, with thousands or even hundreds of thousands of facts each. An area with particularly well-established public ontologies are life sciences; See, e.g., The OBO Foundry[35] with ontologies

[35] http://www.obofoundry.org/

for symptoms, diseases, medications, clinical procedures, genes / proteins, etc. etc. There are even *ontology search engines*, e.g., BARTOC[36] .

Surprisingly however, when being confronted with concrete requirements for an AI application, it turns out that rarely an off-the-shelf ontology is sufficient. In none of my previous projects in various areas (medicine, manufacturing industry, tourism, libraries, creative arts, and software engineering) I could simply take and use an off-the-shelf ontology without adaptation. Other practitioners have made similar experiences (Ege et al., 2015), (Hoppe et al. 2018).

I think Bowker and Star (1999) give a good explanation for this dilemma. They write: "Classifications that appear natural, eloquent, and homogeneous within a given human context appear forced and heterogeneous outside of that context". Different use cases require different structuring of the same knowledge.

Also, the quality of some public ontologies is mediocre. For example, in Wikidata the class wife is in a transitive `rdfs:subClassOf` relationship with class animal (accessed 2019-04-03). A wife being regarded as an animal?! How can such an embarrassing misconception happen? This is because in Wikidata, the following relationship chain is modeled: wife is subclass of woman, is subclass of female, is subclass of Homo sapiens, is subclass of omnivore, is subclass of animal. Each individual `subClassOf` relationship could be considered reasonable; as a relationship chain it is just embarrassing.

When implementing the scripts for extracting the Art Ontology from Wikidata, we put quite some effort in data cleansing measures, e.g., eliminating birth and death dates that are not valid, birth and death places that are no locations, artist's movements that are no artistic movements etc.

Wikidata is still a suitable source for the Art Ontology example in this book and for web apps like openArtBrowser[37]. However, in a project for one of the leading German Arts museums (Staedel digital collection[38]), the use of general-purpose knowledge graphs like Wikidata was out of question due to their quality deficiencies.

Of course, it is generally advisable to favor an off-the-shelf ontology over a custom-made one. Advantages are:

- Less costs for creating and maintaining the ontology
- Potentially higher quality due to contribution and use by many communities (as mentioned above, this is not always the case)

However, developing a custom ontology for a specific use case, e.g., within a corporation is not as expensive as one might think. Experience from practice shows that an experienced team can model about 1,000 concepts within a week (Hoppe, 2015).

[36] http://bartoc.org
[37] https://openartbrowser.org
[38] https://sammlung.staedelmuseum.de

Pre-Processing Off-the-Shelf Ontologies

When analyzing off-the-shelf ontologies for a concrete application use case, the ontologies' scope, structure and quality should be taken into account. Sometimes one or several ontologies can be identified that are quite suitable but do not fit perfectly. In this case a pre-processing step, much like the *ETL (Extraction, Transformation, Loading)* step in Data Warehouses, is recommendable. Ontology pre-processing may include the following activities:

1. Transforming technical data formats, e.g., from tabular format CSV to RDF
2. Transforming ontology schemas, e.g., from `wd:Q3305213` to `:artwork`
3. Quality enhancement, e.g., omitting birth dates that are not valid
4. Integrating multiple ontologies, e.g., removing duplicates

Pre-processing may be supported by services of type "Data Integration / Semantic Enrichment" in the Services Map. From my point of view, this important step is not discussed enough in the AI literature. See also the chapter on AI Application Architecture.

One final remark regarding selecting off-the-shelf ontologies: do not worry too much about technical formats. Converters between XML, CSV, XLS, RDF, database dumps, etc. are available open source or can easily be developed.

Deciding on Technology

There are numerous mature products for knowledge representation available – commercial and open source. So, when it comes to selecting knowledge representation technologies and products, there is no make versus buy decision to be taken – just like you would not implement your own database management system for a business information system.

In the Section 2.6 I recommend a general method for selecting suitable products. Here I give some tips and tricks regarding selecting knowledge representation products:

Check carefully what is *really* needed in your application use case. In many practical use case scenarios, much less is required than what traditional AI frameworks offer. In AI literature, there is much discussion about the expressive power of the OWL full standard compared to OWL light and other standards. In most of my projects, none of the OWL reasoning facilities were required at all. For many application use case scenarios, reasoning over hierarchies is enough (e.g., an artist who lived in Florence also lived in Italy since Florence is in Italy).

On the other hand, *high performance* requirements are common in real-world applications. However, traditional AI knowledge representation products exhibit poor performance due to their sophisticated reasoning facilities. Therefore, high-performance solutions like RDBMS and NOSQL data stores including search index solutions are recommended. Requirements like, e.g., reasoning over hierarchies can often easily be implemented using clever indexing strategies. Those solutions also deliver additional services like, e.g., full-text search which are not provided by classic AI knowledge representation products.

So, my recommendation is to not unnecessarily narrow the product candidates to traditional AI and Semantic Web technology when implementing AI applications.

3.8 Quick Check

Answer the following questions.

1. What is the purpose of knowledge representation?
2. Give examples for classes, individuals, relationships, and rules.
3. What is an ontology?
4. Name different knowledge representation approaches.
5. What does reasoning mean? Give an example. How does it work?
6. Explain the open world assumption and closed world assumption. What are the differences? What are implications of the assumption made?
7. What is a resource in RDF? How are namespaces used? What is an RDF triple?
8. How are classes declared in RDF? How are individuals (instances) assigned to classes?
9. How are properties declared in RDF? How are properties used?
10. Which serialization syntaxes exist for RDF?
11. What is linked data? Give examples of off-the-shelf ontologies.
12. How are simple queries structured in SPARQL?
13. How is the result set of SPARQL queries structured?
14. What advanced features does SPARQL offer?
15. How can rules be implemented with SPARQL?
16. Name knowledge representation services (services map).
17. Name a few products that implement those services (product map).
18. How to select ontologies for a concrete application use cases?
19. How to integrate various off-the-shelf ontologies?
20. How to select knowledge representation products for a concrete AI application project?

4. AI Application Architecture

What characterizes an AI application? Is it the use of a particular technology like a rule engine or an agent framework? Or is it maybe a particular architectural choice like a blackboard architecture?

In my opinion, it is not the implementation characteristics that are relevant for characterizing an AI application. It is mainly the application use case that is relevant: AI applications exhibit behavior of human intelligence - whatever is under the hood.

Although there are cases for rule engines, reasoners, agent frameworks, and blackboard architectures, many AI application in practice are, from an architectural point of view, similar to classic business information systems. Traditional software engineering principles and techniques apply, including separation of concerns, information hiding, layering, component-orientation, etc. Common issues like performance, security, maintainability, cost-effectiveness are important. And, equally, a sound development methodology is important. When following an agile development approach, this includes e.g., early releases and prototypes, regular user feedback and quality assurance, etc.

To rephrase: *AI applications are computer applications and therefore classic software engineering principles apply.*

4.1 AI Reference Architecture

A *reference architecture* is a blueprint for concrete application architectures. In a reference architecture, the benefits of many concrete application architectures are distilled. The architect in a development project may use a reference architecture as a starting point for developing the concrete application architecture.

Fig. 4.1 shows a reference architecture for AI applications.

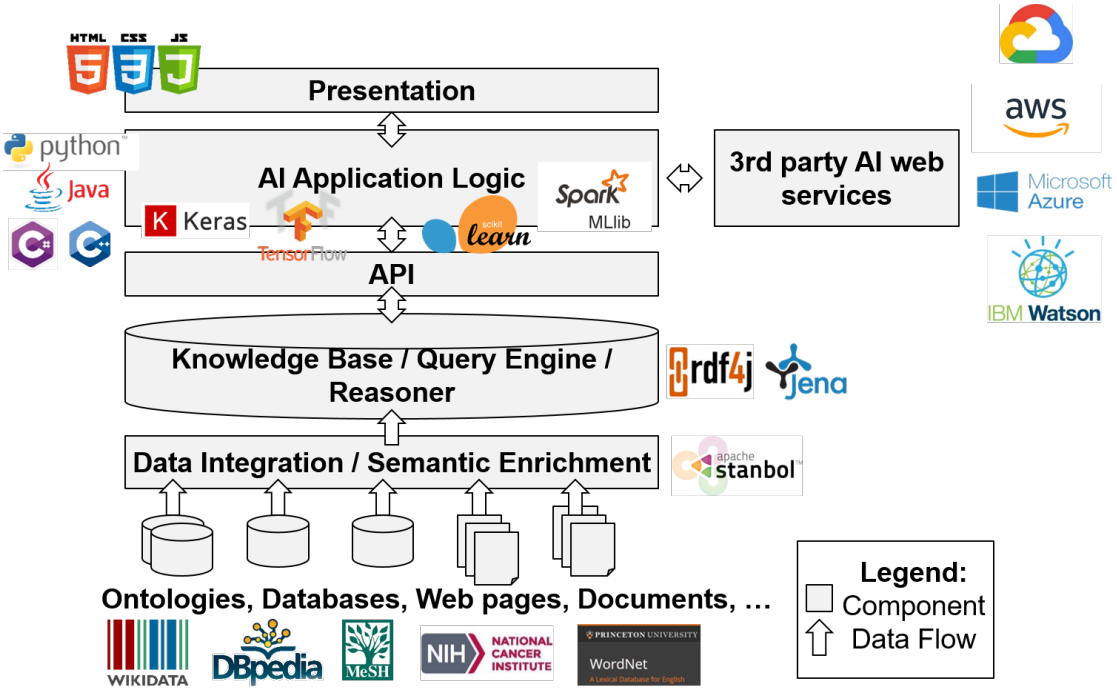


Fig. 4.1: AI reference architecture

The reference architecture is structured as a *layered architecture* similar to the classic three-layer-architecture of business information systems.

The *presentation layer* implements the (graphical) user interface which, in the case of a web apps or mobile apps, may be implemented with state-of-the-art UI technology like HTML/CSS and JavaScript. The UI layer communicates with the application logic layer via an application programmer's interface (API), e.g., using REST (representational state transfer).

The *application logic layer* implements the intelligence of the AI application, e.g., in form of intelligent agents. The AI application logic is usually implemented in a general-purpose programming language. Within the last years, Python is evolving as the primary AI programming language (as Lisp and Prolog were in the 1980s). But object-oriented languages like Java, C# and C++ are also in common use. Often, powerful libraries and frameworks for AI tasks like machine learning, language processing, image processing, etc. are utilized. Examples are Keras, TensorFlow, Scikit-learn and Spark.

Instead of including AI libraries, 3rd party AI web services can be used. Major vendors like Google, Amazon, Microsoft, and IBM offer AI suites with web services for machine learning, language processing, image processing etc.

The underlying data of the AI application is stored in a *knowledge base* which is accessed by the application logic layer via an API. As outlined in the chapter on knowledge representation, technologies with reasoning engines (like, e.g., Apache Jena) may be used. However, also classic storage technologies like RDBMS or NoSQL databases are often used.

Finally, data may be loaded into the knowledge base from various sources, e.g., ontologies, databases,

web pages, documents etc. Those data may be *integrated and semantically enriched* (see the chapter on knowledge representation).

In a concrete AI application, each of those layers may be developed differently. Also, depending on the application use case, individual layers may be missing entirely. For example, in applications where knowledge items are created by the users, the data integration layer is not needed. In an application where the reasoning capabilities of the knowledge base are sufficient, an explicit application logic layer may be omitted. In an embedded AI application, e.g, a robot, a graphical user interface is not needed.

4.2 Application Example: Virtual Museum Guide

Let us consider a concrete application scenario: a *virtual museum guide*. The task of the virtual museum guide is to guide users through a virtual museum - much like a human museum guide who guides visitors though the physical museum.

The degree of intelligence of this virtual museum guide may vary a lot. In the simplest form, the guide may offer the users displays of paintings along with descriptive text. The paintings may simply be presented in a fixed order which has been curated by a human.

In the most complex form, the virtual museum guide tells stories about the paintings, answers natural-language questions of the users (possibly via speech input and output), and adapts the selected paintings and stories to the users' background. For example, children will be told different stories than adults.

While one would probably not consider the simple virtual museum guide as intelligent, the complex one definitely exhibits behavior of human intelligence: understanding, speaking, telling stories, answering questions, etc.

Fig. 4.2 shows a potential architecture of a virtual museum guide application.

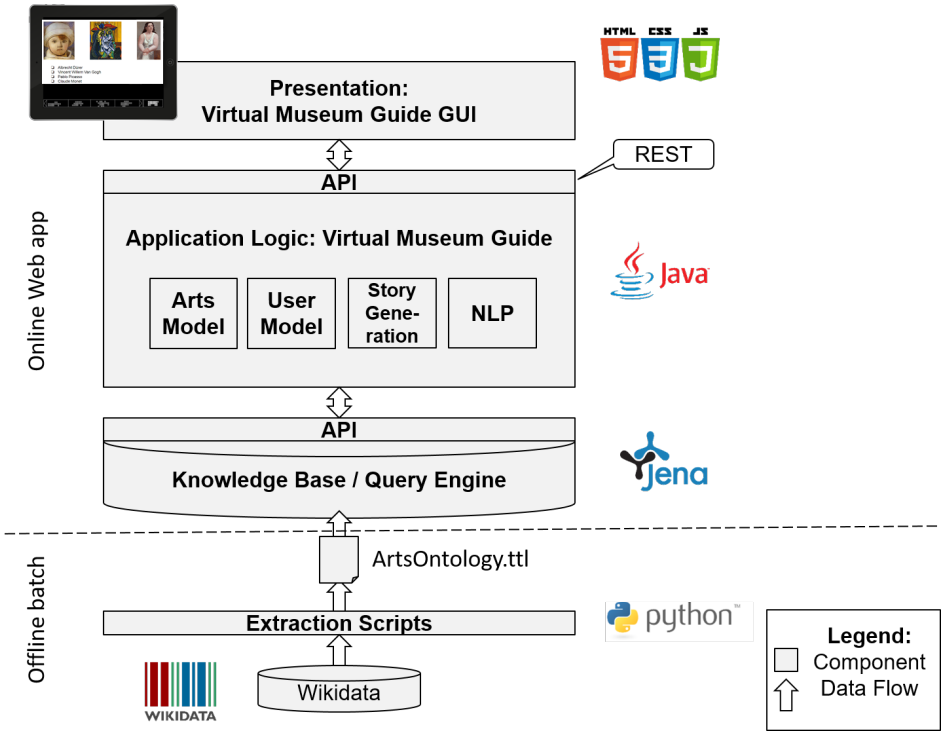


Fig. 4.2: Example architecture: Virtual museum guide application

In this architecture, the virtual museum guide GUI is implemented with HTML5 / CSS and JavaScript including state-of-the-art libraries. The application logic of the virtual museum guide is implemented in Java including libraries like Eclipse rdf4j. Sub-components are:

- Arts model: for representing artworks and their content
- User model: for representing the current user and his / her background
- Story generation: for generating stories about artworks suitable for the current user
- Natural language processing (NLP): for generating voice output and analyzing voice input

The knowledge base is implemented using Eclipse RDF4J (API and knowledge base including reasoner and SPARQL query engine). The Art Ontology is loaded into RDF4J at system start. In an offline step, it is extracted beforehand via Python scripts from Wikidata.

4.3 Data Integration / Semantic Enrichment

The data integration aspect is, in my opinion, not treated enough in AI literature. The knowledge in AI applications often stems from various data sources (see the chapter on knowledge representation). This is similar in the field of Business Intelligence (BI) where the process of integrating data from various sources into a data warehouse (DWH) is often called *ETL (Extraction, Transformation, Loading)*.

ETL can be seen as an architectural pattern where business information systems (data sources) are separated form business intelligence systems. ETL is a pipeline for extracting, transforming, and loading data in a format highly optimized for its use (analytic application).

The ETL architectural pattern also is suitable for the data integration of AI applications. Since in AI applications, data is often semantically enriched, I use the term *Semantic ETL*.

Semantic ETL consists of the following steps.

1. *Extraction* of data from source systems: These could be files, websites, databases, SPARQL endpoints, etc., e.g., the DBpedia SPARQL endpoint.
2. *Filtering* irrelevant data and data of insufficient quality; e.g., selecting only paintings, sculptures and the respective artists from Wikidata; selecting English descriptions only and filtering attributes with wrong datatypes.
3. *Technical format transformation*: transforming from the source formats to the target format, e.g., from JSON to RDF
4. *Data schema transformation*: transforming from the data schemas of the source format to a target data schema, e.g., renaming `wd:Q3305213` to `:artwork`
5. *Semantic enrichment*: heuristically integrating semantic information from various data sources, e.g., Michelangelo's birth and death date from GND, his influences from YAGO, and his paintings from Wikidata
6. *Performance tuning*: optimizing the data storage according to the application use cases, e.g., normalizing data and indexing for high-performance access
7. *Loading*: storing data in the target knowledge base, e.g., rdf4j.

4.4 Application Logic / Agents

In many AI publications, e.g., (Russell and Norvig, 2013), *agents* are described as a metaphor for the central component of an AI application which exhibits intelligent behavior.

Fig. 4.3 by Russell and Norvig (1995) illustrates the concept of an agent.

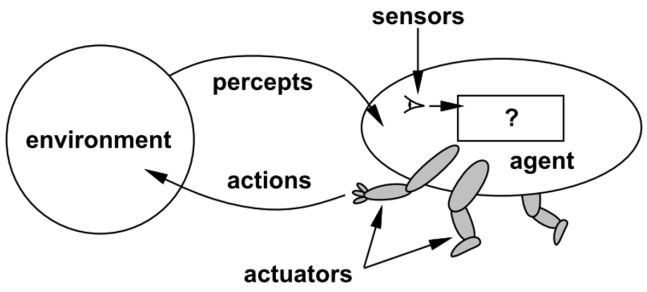


Fig. 4.3: The concept of an agent (Russell and Norvig, 1995, Fig. 2.1)

An agent interacts with an environment and via sensors it perceives the environment. The agent logic then reasons over its perceptions and its internal expert knowledge and plans respective actions.

Via actuators it executes those actions. The executed actions may, in turn, have an effect on the environment which is perceived, again, by the agent.

Fig. 4.4 shows examples of agents, from simple to complex.

Agent	Environment	Percepts and Actions
Thermostat	Room and heating	Measures temperature and adjusts heating accordingly
Softbot	Internet	Crawls web pages, extracts information (e.g., compares prices), makes recommendations or even purchases autonomously
Robot	Home	Performs simple household tasks like vacuum cleaning
Human	World	Performs all sorts of things, some better, some worse for the environment ;-)

Fig. 4.4: Examples of agents

Is it appropriate to call the virtual museum guide an agent? In its simple form (pre-defined guides) one might intuitively say "no". In its complex form (story telling) the answer surely is "yes".

In my opinion, however, this question is not too relevant. Much more relevant is the question whether the agent metaphor is beneficial for designing the virtual museum guide application. And this question may well be answered with "yes". Thinking of the virtual museum guide as an agent may lead to an architecture where perceptions are separated from actions and where the agent establishes a model of all past perceptions that is used for planning the next actions. And the separation of those concerns may well be a good architectural decision.

Agent Frameworks

Agent frameworks provide a base architecture and offer services for developing the agent logic of an AI application. A number of agent frameworks implement a plug-in architecture where framework components and custom components can be integrated. Some frameworks specify domain-specific languages (DSL) for the agent logic. Usually, APIs for integrating code in different programming languages are provided.

See, e.g., the architecture of Cougaar[1] in Fig. 4.5.

[1] http://www.cougaar.world

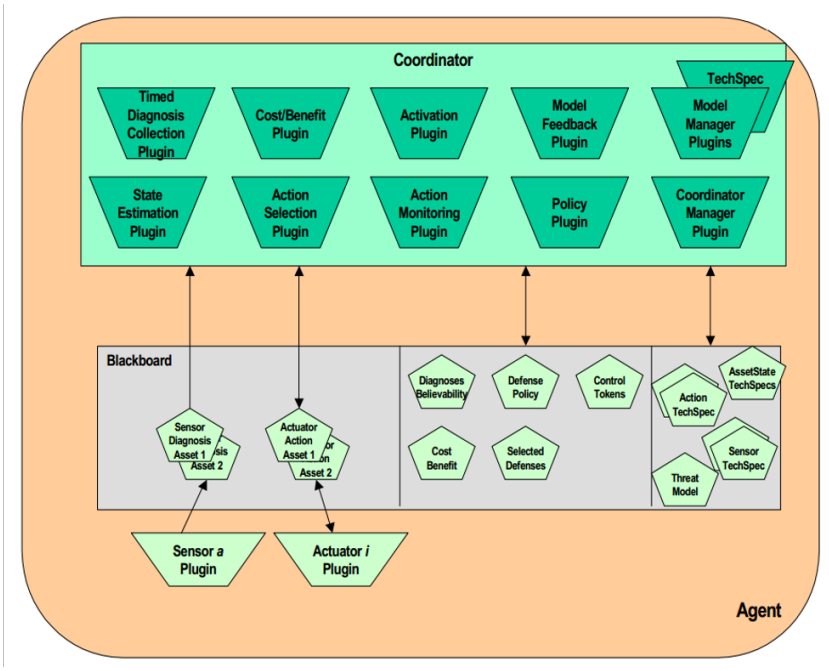


Fig. 4.5: An agent framework example: Cougaar (More et al., 2004)

In Cougaar, coordinator components like a Cost/Benefit Plugin provide the agent logic. The blackboard component is a shared repository for storing information on current problems, suggestions for solving the problems as well as (partial) solutions. Sensors and actuator components may be plugged in. Sensors regularly update information on the blackboard.

Other agent frameworks are JaCaMo[2], JADE[3], JIAC[4], AgentFactory[5], and the Jadex BDI Agent System[6]. For details see the appendix.

When to use an Agent Framework?

In my opinion, the metaphor of an agent is useful when designing an AI application. The separation of sensors from actuators as well as the separation of a model of the application domain and the environment from the agent logic is good architectural practice. However, the use of an agent framework is not always recommended. This is because every framework involves a learning curve and adds new technological dependencies to the project.

If the the agent logic is sufficiently complex and the services provided by the framework are suitable for the application use case then the costs for introducing an agent framework may well be justified. However, if this is not the case, a traditional component-based software architecture is adequate. The architectural recommendations following the agent metaphor can still be implemented to some degree.

[2] http://jacamo.sourceforge.net
[3] http://jade.tilab.com
[4] http://www.jiac.de/agent-frameworks
[5] https://sourceforge.net/projects/agentfactory
[6] http://sourceforge.net/projects/jadex/

4.5 Presentation

The (graphical) user interface of an AI application is not AI-specific. As in all IT applications, it is of essential importance for the user's experience of the application. See the comprehensive literature for developing user-centric applications.

4.6 Programming Languages

Within the last years, Python is being established as the major AI programming language. This development has been supported by major players publishing their AI libraries and frameworks in Python, e.g., Google TensorFlow. Also, there are still numerous AI libraries available in traditional object-oriented programming languages like Java, C#, and C++. The traditional AI programming languages of the 1980, Lisp and Prolog, only play a niche role in today's AI application development. But they have a major influence on the design of modern dynamic programming languages like Python, R, Julia and others.

For making a sound programming language decision in an AI application development project, one should consider various aspects:

- Which technology stack offers the best resources (runtime platform, libraries, developers' tools, etc.)?
- Are there enough developers familiar with the technology?
- Where is the best support (User groups etc.)

To conclude: for developing AI applications, all software engineering principles for the development of large-scale, complex IT systems apply.

4.7 Quick Check

Answer the following questions.

1. What characterizes an AI application?
2. What are the main components of the AI reference architecture?
3. Could you speak of an AI application if none of these technologies are used: reasoning engine, machine learning framework, agent framework?
4. What is an agent? Give examples.
5. Which services do agent frameworks offer?
6. In which situations is the use of an agent framework recommended? In which not?
7. Which programming language should be used for developing an AI application?

5. Information Retrieval

Information retrieval (IR) allows retrieving relevant documents matching an information need. Fig. 5.1 shows IR in the AI landscape.

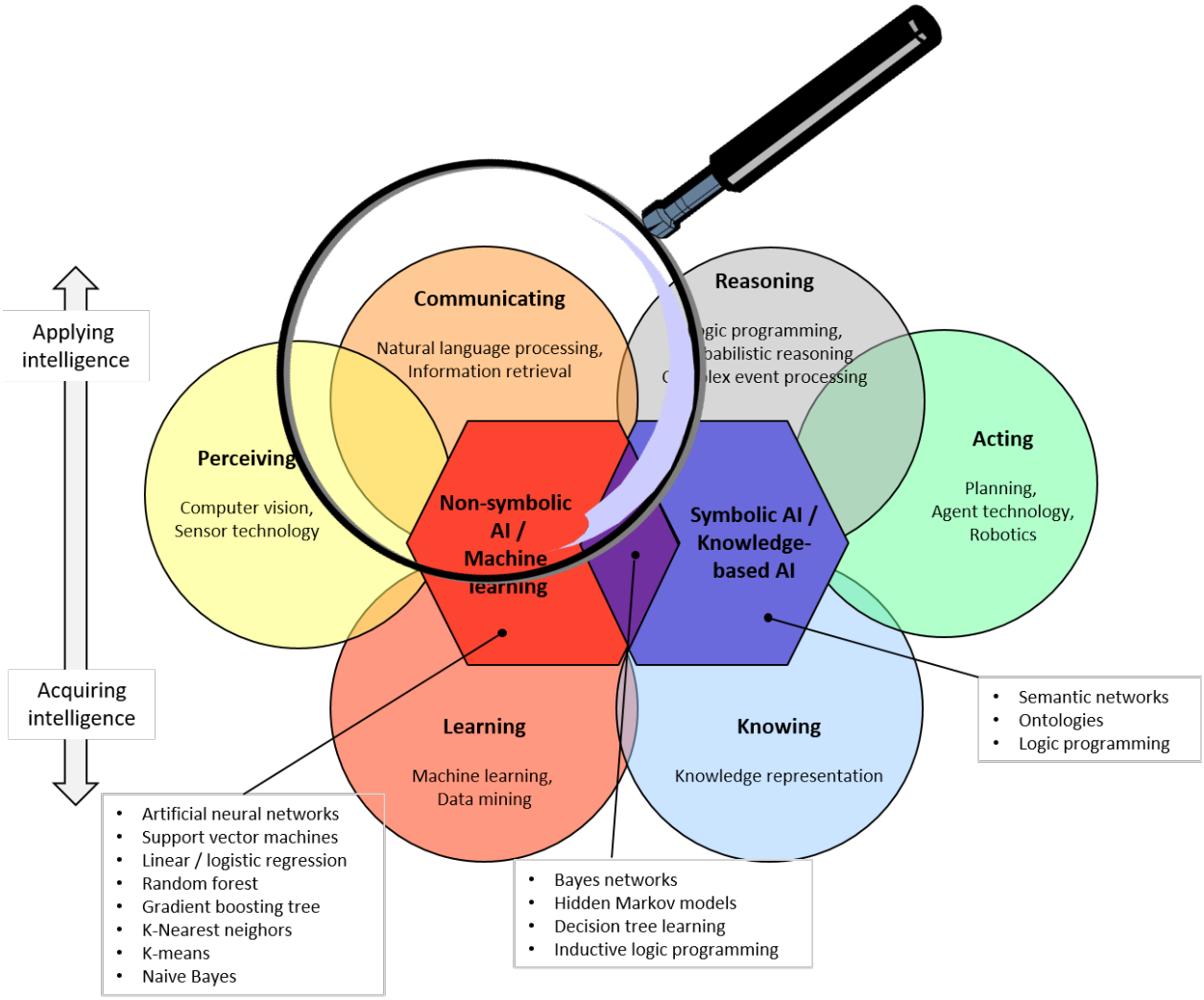


Fig. 5.1: Information retrieval in the AI landscape

Together with natural language processing, it can be assigned to the ability “communicating”.

Information Retrieval can be described by its input/output behavior as follows.

- *Input*: an information need, e.g., specified via a search text
- *Output*: a set of relevant documents from a (potentially very large) collection that satisfy the information need, e.g., text documents, images, audio files, videos, etc.

The basis for performing information retrieval is indexed metadata of those documents.

The most prominent examples of information retrieval systems are *web search engines* like Google[1], Yahoo![2], and Yandex[3].

Information retrieval may be considered a simple form of AI. Sometimes it is considered a subarea of natural language processing. In fact, the term "information retrieval" is even overstated insofar as simply data (documents) are retrieved – not information. Therefore, a more suitable term would be "document retrieval".

However, information retrieval is of enormous end-user value. Web search engines are the major facilitators of the World Wide Web. Also, in many applications, the integration of an information retrieval component may considerably increase the user experience. Examples are full-text search and semantic autosuggest features. Furthermore, there are mature open source libraries for information retrieval that can easily be included in applications.

Because of those reasons, I decided to dedicate a chapter of this book to information retrieval. Every AI application developer should be familiar with information retrieval.

5.1 Information Retrieval Services Map

Fig. 5.2 shows the information retrieval services map.

[1] https://www.google.com
[2] https://www.yahoo.com/
[3] https://www.yandex.com/

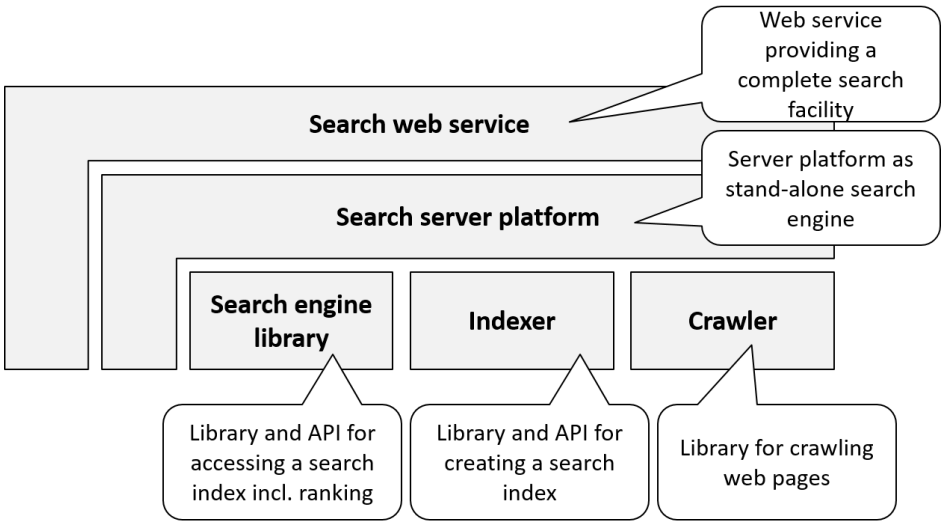


Fig. 5.2: Information retrieval services map

The basic way of employing information retrieval in an application is to include an indexer and a search engine library. An *indexer* is software for indexing a collection of documents and for storing those indexes. The indexer is implemented in a programming language like Java and accessible via an API. Indexing is an offline-process, usually implemented as a batch. The *search engine library* can then be used online to access this index. It can be accessed via a search query API.

If the documents to be indexed are not available initially but have to be retrieved first, then a *crawler* may be used. A crawler is a library for visiting web pages in order to extract data. This data may then be indexed and searched for. Web search engines work like this.

In case, the application is implemented in a different programming language, a *search server platform* may be used. It allows starting a server process on an operating system which can then be accessed by applications via a programming language independent interface, e.g., HTTP / REST. Like the search engine library, documents must be indexed for the search server platform before it can be used for querying.

Finally, an existing search engine can be included in an application as a *search web service* . All prominent search engines like Google, Yahoo!, and Yandex offer web services.

5.2 Information Retrieval Product Map

Fig. 5.2 shows the information retrieval product map.

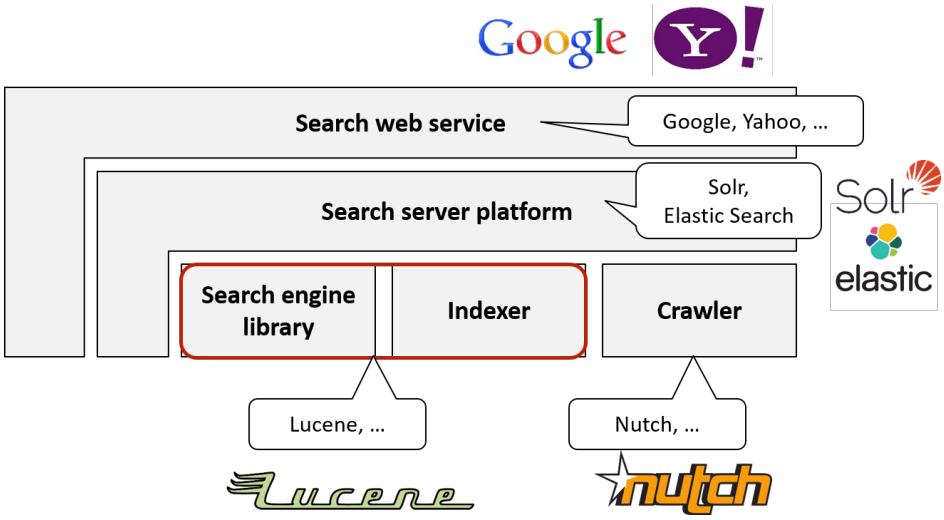


Fig. 5.2: Information retrieval product map

Apache Lucene[4] is *the* state-of-the-art open source search engine library and indexer. Lucene is implemented in Java and is used in numerous applications. Ports to other languages exist, e.g., PyLucene for Python.

Apache Nutch[5] is a web crawler.

There are two most prominent search server platforms, both built on top of Lucene: Apache Solr[6] and Elasticsearch[7]. Both provide similar functionality, are mature, and have been used in numerous large-scale applications.

All prominent search engines like Google, Yahoo!, and Yandex offer web services to access the search, e.g., https://developer.yahoo.com/search-sdk/

More products and details can be found in the appendix.

5.3 Tips and Tricks

Developers are spoiled for choice among the various options in the information retrieval services map. So what information retrieval service options are most suitable for a given situation?

Integrating a search service like Google is the natural choice if the application is to offer a general web search. In this case, the legal conditions of the search service APIs should be studied and compared carefully. Costs may incur. It should be evaluated whether the runtime performance is sufficient for the particular use case.

In the case of scenarios where documents to be retrieved are not available on the web, but are application-specific, the search server platforms or libraries must be used. Search server platforms

[4] https://lucene.apache.org/
[5] http://nutch.apache.org/
[6] https://lucene.apache.org/solr/
[7] https://www.elastic.co/products/elasticsearch

as well as libraries offer extremely high performance, also with very large data sets. For example, in one of my projects we use Apache Lucene and are able to search 10 million documents (book metadata) in less than 30 ms.

When is a search server platform suitable? When should a developer use a library instead?

Apache Lucene as a library is easily included in Java applications. The API is well documented and a working prototype may be implemented within a few hours. This makes the library solution particularly attractive for Java applications.

If other programming languages are used for implementing the application, a search server platform must be used. E.g. for C#, SolrNet[8] may be used to conveniently access a Solr server. Also, there are reasons for using a search server platform even when the application is implemented in Java. This is because search server platforms offer additional services, e.g. for system administrators. Those services include monitoring, clustering, etc. Therefore, the issues of administration and operation should also be taken into account before making a decision between search server platform and library.

5.4 Application Example: Semantic Autosuggest Feature

A *semantic autosuggest* feature is a good example of how information retrieval may improve the user experience considerably with relatively little implementation effort. The concept of *autosuggest* (a.k.a. *autocomplete*) is well-known from web search engines like Google. While the user is typing a search string, terms are suggested in a drop-down menu from which the user can choose.

Semantic autosuggest extends this feature by utilizing semantic information, e.g., term categories. See Fig. 5.3 for an example in the openArtBrowser[9] (Humm, 2020).

[8] https://github.com/mausch/SolrNet
[9] https://openartbrowser.org

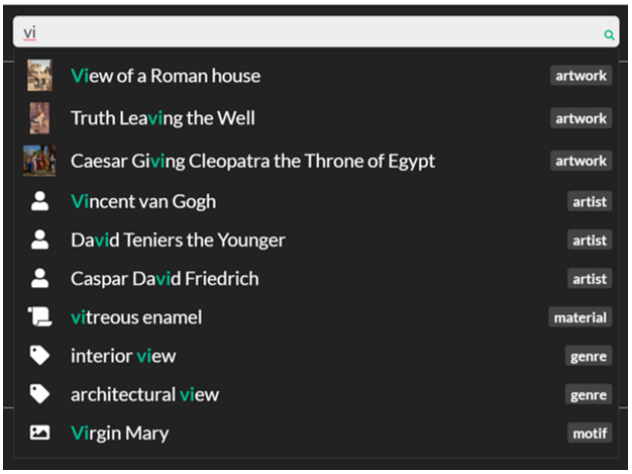


Fig. 5.3: Application example: Semantic autosuggest

OpenArtBrowser is a web app for educating in visual art, fascinating users for paintings, drawings and sculptures. It provides a search feature with semantic autosuggest. In the example shown in Fig. 5.3, the user is typing the letters "vi...". Various artworks, artists, materials, genres and motifs are displayed which contain the letters "vi" (case-insensitive), grouped according to their semantic category. The matching letters "vi" are highlighted (in green). A sophisticated heuristic ranking selects a limited number (here 10) of suggestions from a potentially very large number of matches, e.g., the artist Vincent van Gogh, the motif Virgin Mary, and the artwork View of a Roman House.

By selecting one of the suggested terms, the user also selects a semantic category (artist, artwork, motif, genre, etc.). The search will then be refined accordingly using the search term and the semantic category.

OpenArtBrowser and its semantic autosuggest feature is based on the Art Ontology described in Chapter 3. The semantic AutoSuggest feature was implemented using ElasticSearch. An ngram index was created from the ArtOntology. An autocomplete widget of some JavaScript library like JQuery UI[10] was used in the HTML client. From the web client, the ElasticSearch server was invoked to query the terms.

The implementation of the entire semantic autosuggest feature involves less than 100 lines of code.

5.5 Quick Check

Answer the following questions.

1. What does the term information retrieval mean?

[10] https://jqueryui.com/autocomplete/

2. What are the main services of information retrieval tools?
3. Name state-of-the-art information retrieval tools and technologies.
4. When to use which technology?
5. Explain semantic autosuggest. How can it be implemented?

6. Natural Language Processing

Natural Language Processing (NLP) is the AI area which deals with processing natural languages like English, German, etc. Fig. 6.1 shows NLP in the AI landscape.

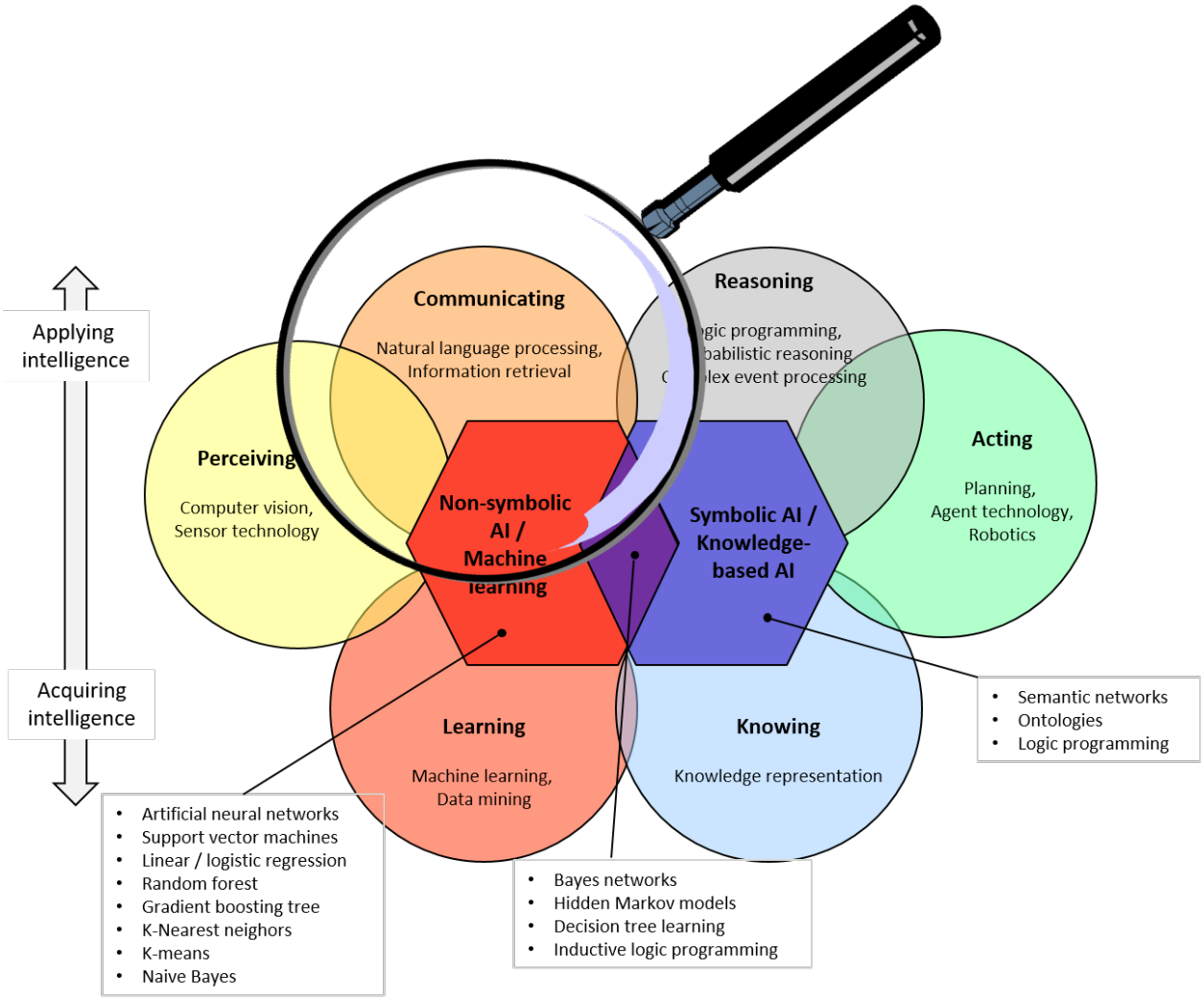


Fig. 6.1: NLP in the AI landscape

NLP can be assigned to the ability of “communicating”. It is a broad area that covers many aspects. Examples are:

- Spell checking and grammar checking of written texts, e.g., as in word processors
- Classifying texts, e.g. according to the topic
- Understanding written text, e.g., the sentiment of Twitter tweets (positive, neutral, negative)
- Understanding speech, e.g., voice control of a navigation system
- Translating texts, e.g., between English and German
- Answering natural language questions, e.g., in a specific domain like medicine
- Summarizing written texts, e.g., news
- Generating texts, e.g., story telling
- Generating voice, e.g., routing information in a navigation system

Due to the diversity of NLP, different subareas are distinguished. There is no commonly agreed classification but often the following NLP subareas are mentioned:

- *Information Retrieval (IR)* supports retrieving documents for a specific information need. As explained in the last chapter, the term "Document Retrieval" would be more suitable. Information Retrieval is usually considered a subarea of NLP.
- *Information Extraction (IE)* deals with the understanding of written and spoken text. This includes the analysis of texts and transformation into a knowledge representation that can be queried. Sentiment analysis is a simple form of information extraction.
- *Question Answering (QA)* generates natural language answers to natural language questions, in written or spoken form.
- *Machine Translation* allows translating texts between different natural languages.
- *Text Generation* supports the generation of written or spoken texts. Text summaries and story telling are forms of text generation.

6.1 The Big Picture

Fig. 6.2 shows the big picture of NLP as seven levels of language understanding, adopted from (Harriehausen, 2015).

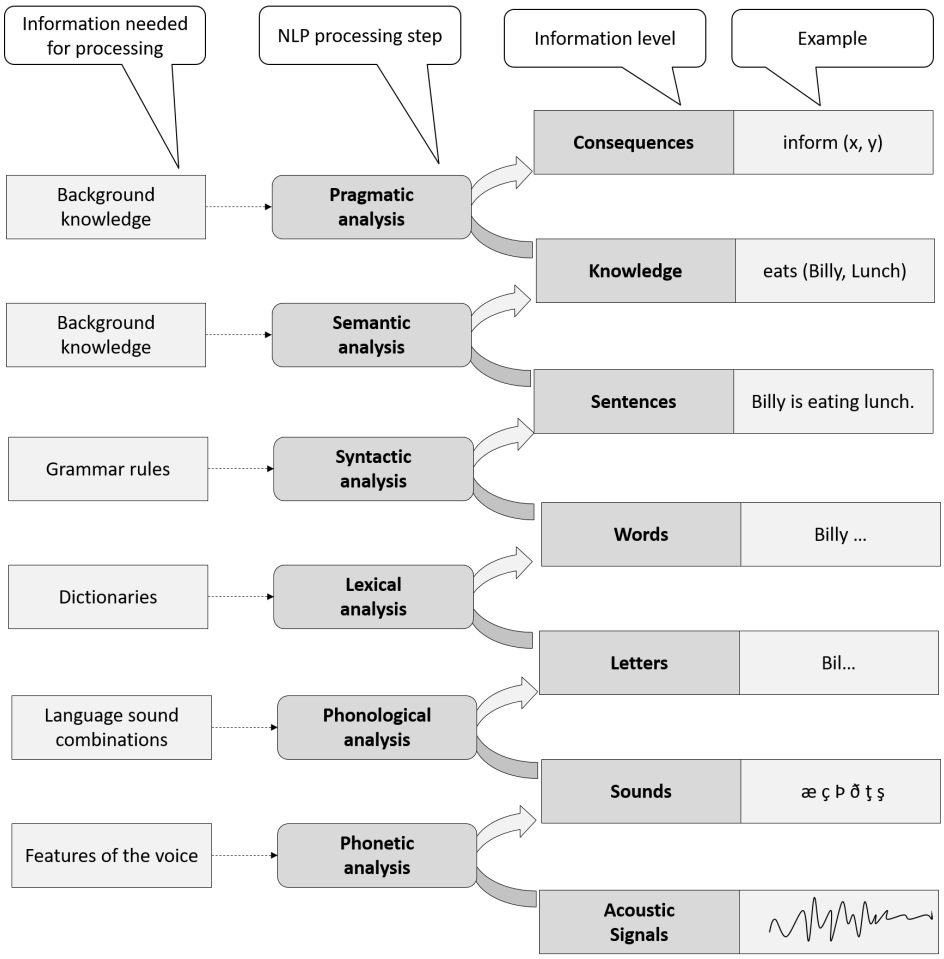


Fig. 6.2: The 7 levels of language understanding, adopted from (Harriehausen, 2015)

The figure shows information levels and NLP processing steps to raise the level of understanding. On the lowest level there are acoustic signals. *Phonetic analysis* uses features of the voice to extract sounds. *Phonological analysis* uses sound combinations of specific languages in order to extract letters. *Lexical analysis* may use dictionaries to extract individual words. *Syntactic analysis* (parsing) uses grammar rules in order to extract sentences and their structure (parse tree). *Semantic analysis* uses background knowledge to represent the knowledge in a text. Finally, *pragmatic analysis* may draw conclusions and consequences for actions.

In most AI applications, only some NLP processing steps are relevant. When dealing with written texts, then phonetic and phonological analysis are not necessary. Also, semantic and pragmatic analysis may be simple or even irrelevant, depending on the application use case.

In this chapter, I focus on lexical, syntactic and semantic analysis which is used in most NLP applications. I first explain a simple approach, namely the bag-of-words model. I then explain individual processing steps for lexical and syntactic analysis (from letters to sentences) that may lead to a deeper semantic analysis.

6.2 Simple Approach: Bag-of-words Model

The *bag-of-words (BoW) model* is a simple NLP approach which delivers surprisingly good results in certain application scenarios like text classification and sentiment analysis. In the BoW, a text is represented as the *bag (multiset)* of its words, disregarding grammar and even word order but only keeping the multiplicity of the words.

Consider the following example text.

```
John likes to watch movies. Mary likes movies too.
```

The bag of words, represented in JSON, is:

```
BoW = {"John":1,"likes":2,"to":1,"watch":1,"movies":2,"Mary":1,"too":1};
```

The word `John` appears once in the text, the word `likes` twice etc.

Machine Learning with Bags of Words

In the simplest form, vectors resulting of bags of words can be used in supervised ML approaches as described in Chapter 2. Consider the ML task of classification with texts t1, … tn and classes A, B, C. Then the data set for ML training consists of each distinct word in all texts as features (attributes as classification input) and the classes as labels (classification output). See Fig. 6.3 with the example text above as t1.

ID	John	likes	to	watch	movies	Mary	too	…	label
t1	1	2	1	1	2	1	1	…	A
t2	0	0	2	0	0	0	1	…	C
…	…	…	…	…	…	…	…	…	…
tn	0	1	2	2	3	0	0	…	A

Fig. 6.3: ML classification data from bags of words

Now, any ML approach suitable for classification can be used, e.g. Artificial Neural Networks, Decision Trees, Support Vector Machines, k-nearest Neighbor etc.

tf-idf

As a rule of thumb, a term appearing often in a text is more important than a term appearing rarely. However, there are exceptions to this rule of thumb. Consider so-called *stop words* like “the”, “a”,

"to" etc. which are most common in English texts but add little to the semantics of the text. In information retrieval, stop words are usually ignored.

How to deal with this in the BoW model which mainly deals with word counts?

One approach is to remove stop words before computing the bag of words. This approach much depends on the selection of the right stop words.

There is another elegant, general approach which avoids fixed stop word lists: *term frequency - inverse document frequency (tf-idf). Term frequency* is based on the count of a particular word in a concrete text as shown in the example above. *Document frequency* considers the count of a particular word in an entire *corpus*, i.e., a large set of texts.

tf-idf puts the term frequency in relation to the document frequency. So, a word like "to" which appears often in all texts but not more often in the text under consideration will not have a particularly high tf-idf and, therefore, will not be considered important. In contrast, a word like "movies" which occurs twice in the short text above but not particularly often in texts in general will have a high tf-idf and, therefore, will be considered important for this text. This matches the intuition.

There are various formulas for computing tf-idf in practical use, which are more meaningful than a simple quotient of the word counts. See e.g. the Wikipedia entry on tf-idf[1]. NLP libraries conveniently provide implementations of tf-idf.

The ML classification performance can be improved by using the tf-idf values instead of the simple word counts in the training data.

N-gram Model

The simple BoW model as explained above treats each individual word independently. The word order gets ignored completely. The *n-gram model* is a simple improvement which takes combinations of up to n successive words into account. N is usually relatively small, e.g., 2 or 3.

See Fig. 6.4. with an extension of the example in Fig. 6.3 to a 2-gram model.

ID	John	likes	to	...	John likes	likes to	to watch	...	label
t1	1	2	1	...	1	1	1	...	A
t2	0	0	2	...	0	0	1	...	C
...	...	...	...	...	...	...	...	...	...
tn	0	1	2	...	0	1	2	...	A

Fig. 6.4: ML classification with 2-gram model

[1] https://en.wikipedia.org/wiki/Tf%E2%80%93idf

n-gram models can be combined with tf-idf by simply computing the tf-idf values for the n-grams.

The BoW model is simple and relatively easy to implement. Despite its simplicity, it delivers good prediction performance for many application use cases, particularly when combined with extensions like tf-idf or n-grams, and particularly with large training data sets.

Obviously, the number of features (attributes) in the BoW model can get extremely large, particularly when using n-grams. Hundreds of thousands of features are possible. Particularly with huge training data sets this can cause major performance problems in the ML training and prediction phases. Feature selection mechanisms from unsupervised ML may be used to reduce the number of features and, hence, alleviate those performance problems.

6.3 Deeper Semantic Analysis: From Letters to Sentences

The bags-of-words model is a simple approach, based on counting words. This is most different from the way humans understand texts. Despite its simplicity it delivers surprisingly good results for simple NLP tasks like text classification. However, it is obvious that complex NLP tasks like question answering require a deeper semantic analysis of texts than simply counting words.

In this section I will present some approaches: from letters to sentences.

Tokenization

Tokenization is the step of grouping letters into words. This step seems primitive: looking for blank characters seems to be enough. However, tokenization is a little more complicated. Consider the following example sentence:

```
1 My dog also likes eating sausage.
```

Following the primitive tokenization approach, the last word identified would be `sausage.`. However, in fact, the last word is `sausage` and the period `'.'` is a separate token. So, the correct tokenization result is as follows (Fig. 6.5).

Fig. 6.5: Tokenization example

Sentence splitting

Sentence splitting identifies whole sentences. Sentences are terminated by periods (full stops). However, simply looking for the next period is not enough. Consider the following sample sentence.

```
Interest rates raised by 0.2 percent.
```

Obviously, the point in `0.2` is part of a floating point number and does not terminate the sentence. Other cases to be considered are abbreviations like `e.g.`, ellipsis (`...`), etc.

Stemming, Part-of-speech (PoS) Tagging

Stemming means reducing a word to its root word. E.g., `eat` is the root word of `eating`. *Part of speech (PoS)* is the grammatical category of a word. E.g., `eating` is the gerund or the present participle of the verb `to eat`. *PoS Tagging* is the step of identifying the PoS of a word.

Fig. 6.6 shows the PoS tagging result of the sentence `My dog also likes eating sausage.`

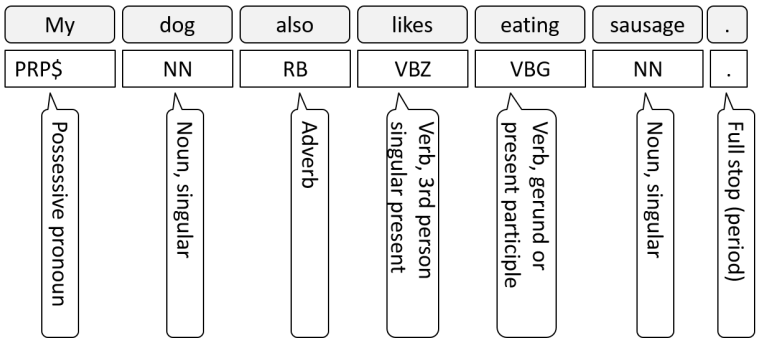


Fig. 6.6: PoS tagging example

In this figure, the Penn Treebank tag set[2] is used. E.g., Verb, gerund or present participle is marked as `VBG`. The Penn Treebank tag set is a de-facto standard used by many PoS tagging tools.

Note: Tokenization and stemming are often pre-processing steps before applying a BoW model. They may improve prediction performance and, at the same time, reduce the number of features.

Parsing

Parsing is the step of analyzing the grammar of a sentence. The result is the sentence structure, usually denoted as a tree. Fig. 6.7 shows the parsing result for the sentence `My dog also likes eating sausage.`

[2] http://www.clips.ua.ac.be/pages/mbsp-tags

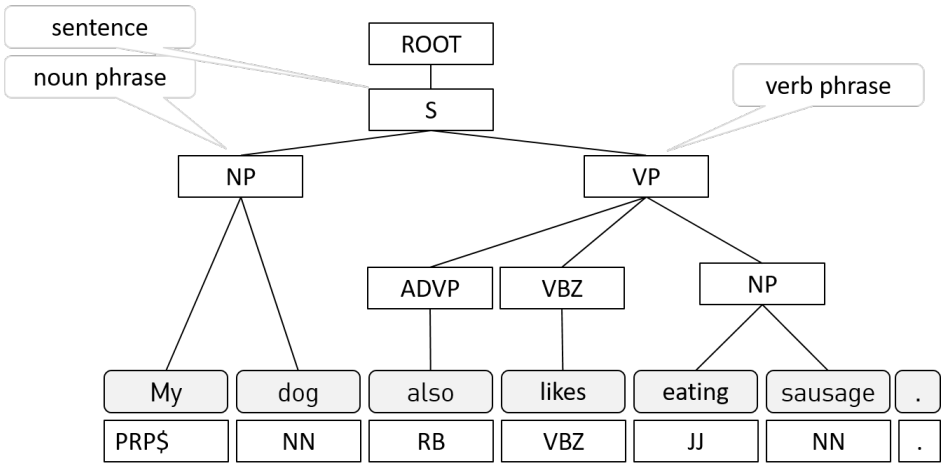


Fig. 6.7: Parsing

Again, the the Penn Treebank tag set is used. E.g., `NP` stands for noun phrase and `VP` for verb phrase.

Parsing of most natural language sentences is highly ambiguous. As humans, we rarely notice this ambiguity. Our brain combines the syntactic analysis and the semantic analysis and chooses the "obvious" meaning, i.e., the most likely variant. However, we also sometimes stumble on ambiguities in the language. Many jokes play with misunderstandings based on ambiguities. For example[3]:

"I want to be a millionaire. Just like my dad!" "Wow, your dad's a millionaire?" "No, but he always wanted to be."

If you technically parse natural language sentences you may be surprised of how many different interpretations of the same sentence are valid. Consider the following example sentence:

```
1 I saw the man on the hill with a telescope.
```

Fig. 6.8, adopted from AllThingsLinguistic[4], shows five different, valid interpretations of this sentence.

[3] http://www.ijokes.eu/index.php/joke/category/misunderstanding?page=2
[4] http://allthingslinguistic.com/post/52411342274/how-many-meanings-can-you-get-for-the-sentence-i

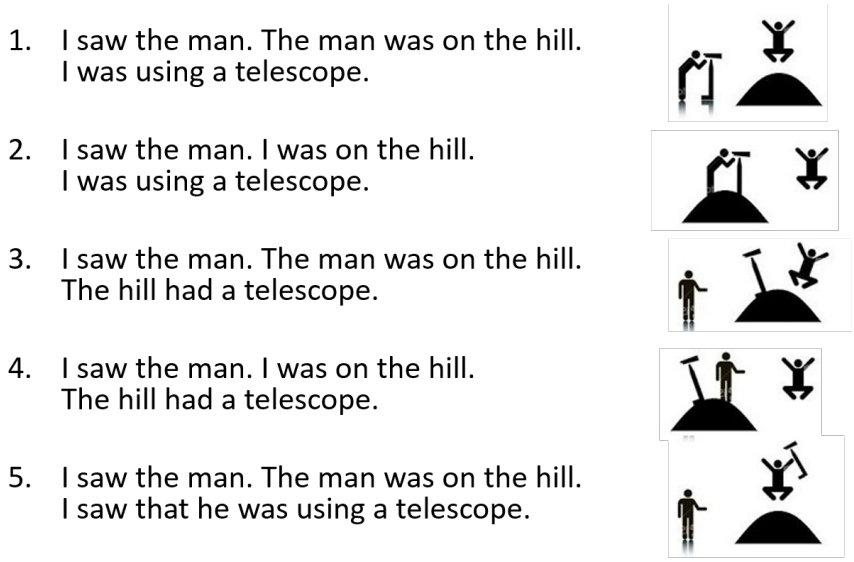


Fig. 6.8: Parsing ambiguity

As an exercise, you may construct a parse tree for each interpretation of the sentence.

Early NLP parsers were rule-based. They mechanically applied grammar rules to sentences. They had enormous difficulties with the multiple alternative parse trees, as well as with grammatically incorrect sentences. Most modern NLP parsers are statistics-based. They produce the most likely parse result according to statistics and can also deal with grammatically incorrect sentences, as we humans do.

6.4 Services and Product Maps

NLP Services Map

Fig. 6.9 shows the NLP services map.

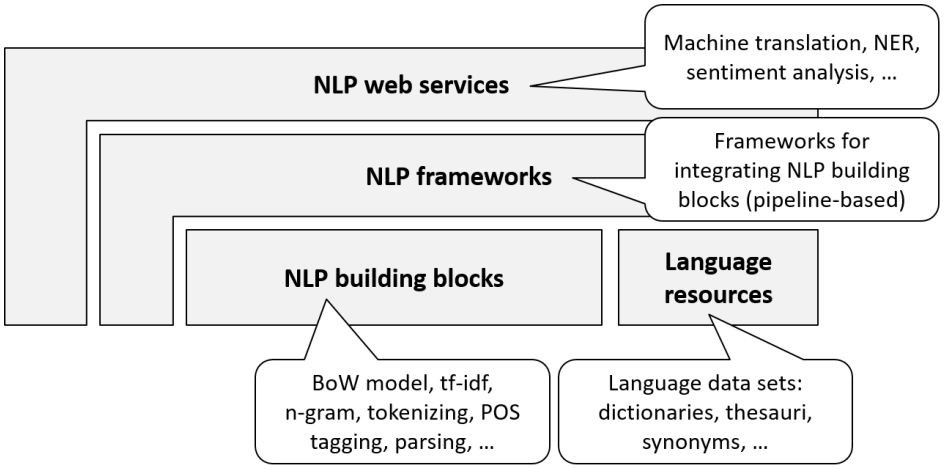


Fig. 6.9: NLP services map

When developing an AI application with NLP facilities, you very rarely build basic NLP *building blocks* from scratch. Class libraries with powerful and well-established building blocks for BoW model, tf-idf, n-gram, tokenization, sentence splitting, PoS tagging, parsing etc. exist and can be integrated into your application. Additionally, *language resources* like dictionaries may be used.

When building complex custom NLP applications, the use of an NLP framework is recommended. They usually follow a pipeline approach allowing to plug in off-the-shelf NLP building blocks. NLP frameworks are powerful and highly customizable. However, they require a certain level of expertise, both on NLP concepts as described above as well as framework-specifics.

For a number of NLP tasks, entire solutions may be integrated into an AI application as a web service. Examples are translation services, voice-to-text transformation services, named entity recognition, sentiment analysis etc. Including an NLP web service is, of course, the easiest and least effort solution. However, you should check licenses, performance, privacy and availability issues involved.

NLP Product Map

Fig. 6.10 shows the NLP product map.

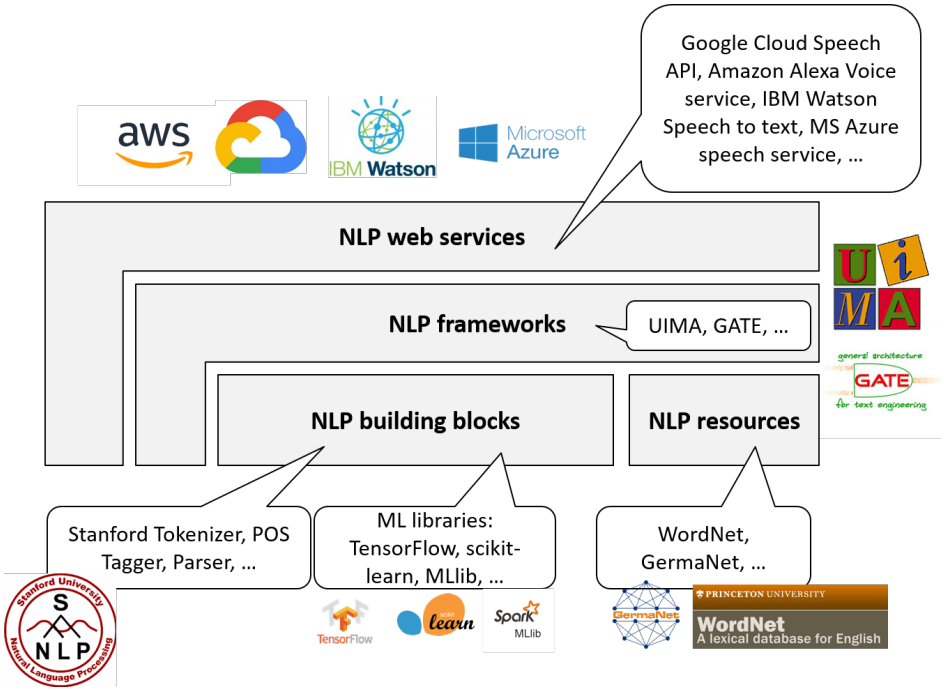


Fig. 6.10: NLP product map

Apache UIMA[5] and GATE[6] are the most widely used NLP frameworks. While GATE allows experimenting with NLP using a graphical desktop application, UIMA is more suitable for software developers offering plug-ins for IDEs like Eclipse.

There are numerous NLP building blocks, e.g. from the Stanford University NLP group[7]. Many of them can be plugged into UMIA and GATE. However, sometimes wrappers are needed like from uimaFIT[8] and DKPro[9]. ML libraries like TensorFlow[10], scikit-learn[11] and MLlib[12], offer functionality for the BoW model, tf-idf and n-grams.

The most prominent NLP language resource for the English language is WordNet[13].

There are also numerous NLP web services from various providers, e.g., Amazon Alexa Voice service[14], Google Cloud Speech API[15], Google Translate API[16], IBM Watson NLP[17], and MS Azure Speech Services[18].

[5] https://uima.apache.org/
[6] https://gate.ac.uk/
[7] http://nlp.stanford.edu/software/
[8] https://uima.apache.org/uimafit.html
[9] https://www.ukp.tu-darmstadt.de/research/current-projects/dkpro/
[10] https://www.tensorflow.org/
[11] http://scikit-learn.org/
[12] http://spark.apache.org/mllib/
[13] https://wordnet.princeton.edu/
[14] https://developer.amazon.com/de/alexa-voice-service
[15] https://cloud.google.com/speech
[16] https://cloud.google.com/translate
[17] https://cloud.ibm.com/catalog/services/natural-language-understanding
[18] https://azure.microsoft.com/de-de/services/cognitive-services/speech

6.5 Examples

I will briefly introduce one prominent example for each NLP service category in the next sections, namely WordNet (NLP resource), Stanford Parser (NLP building block), UIMA (NLP framework), and Dandelion API (NLP web service).

More NLP products and details can be found in the appendix.

NLP Resources: WordNet

WordNet[19] is a state-of-the-art lexical database for the English language. It lists over 150,000 English words: nouns, verbs, adjectives and adverbs. For each word, different meanings ("senses") are distinguished. For example, 7 different noun senses and one verb sense of the word "dog" are listed, including the animal as well as minced meat (as in "hot dog").

Fig. 6.11 shows a screenshot of the WordNet online search[20].

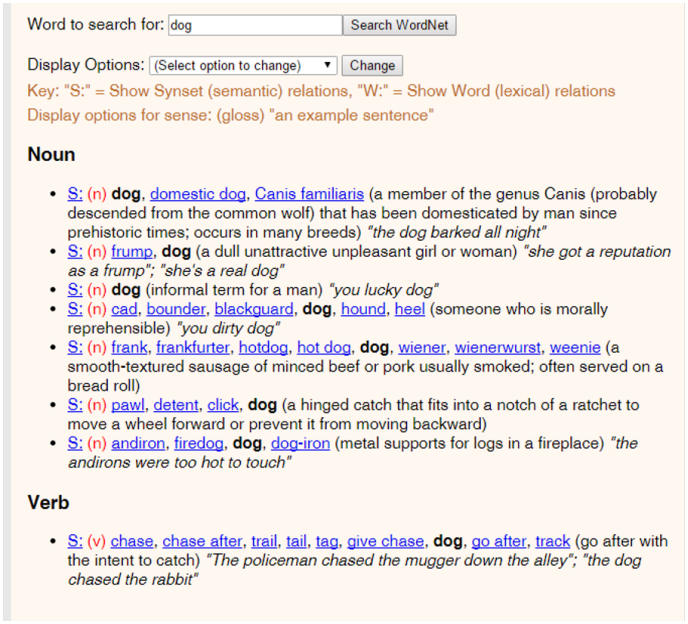


Fig. 6.11: WordNet example: Senses of the word "dog"

For each word sense, a description and different relationships are specified.

- Synonyms, e.g., "Canis familiaris" and "Domestic" dog for the "animal" sense of the word "dog"
- Hypernyms (broader terms), e.g., "mammal" and "animal"
- Hyponyms (narrower terms), e.g., "Puppy", "Hunting dog", "Poodle", etc.

[19] https://wordnet.princeton.edu
[20] http://wordnetweb.princeton.edu/perl/webwn?s=dog

See Fig. 6.12.

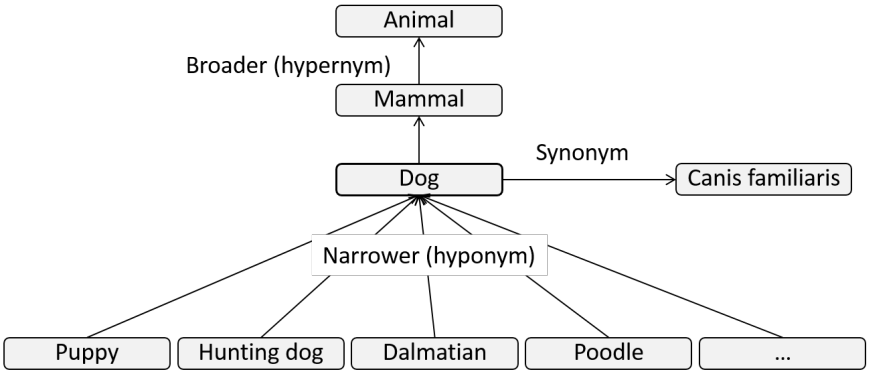


Fig. 6.12: WordNet example: Relations of the word "dog"

WordNet is open source under a BSD license. It can be used in AI applications in various forms. A set of "standoff files" can be downloaded and can be used in applications of any programming language. The WordNet database can be downloaded as a binary for Windows, Unix, and Linux. It can be integrated into applications of any programming language using operating system calls. Finally, the online version of WordNet can be integrated via HTTP.

Which integration type is recommended? As usual, integrating the online service is the least-effort approach. If a permanent Internet connection is guaranteed and the performance is sufficient, then this is recommended. Working with the raw files offers the most flexibility but requires considerable implementation effort. In most cases, working with the locally installed WordNet database is the solution of choice: good performance, no dependency on the remote system and relatively small implementation overhead.

NLP Building blocks: Stanford Parser

The Stanford Parser[21] is a state-of-the-art statistical NLP parser. It supports different natural languages, namely English, French, Spanish, German, and Chinese. It is implemented in Java and is published open source under the GNU General Public License.

See Fig. 6.13 for a screenshot of the online parser[22].

[21] http://nlp.stanford.edu/software/lex-parser.shtml
[22] http://nlp.stanford.edu:8080/parser

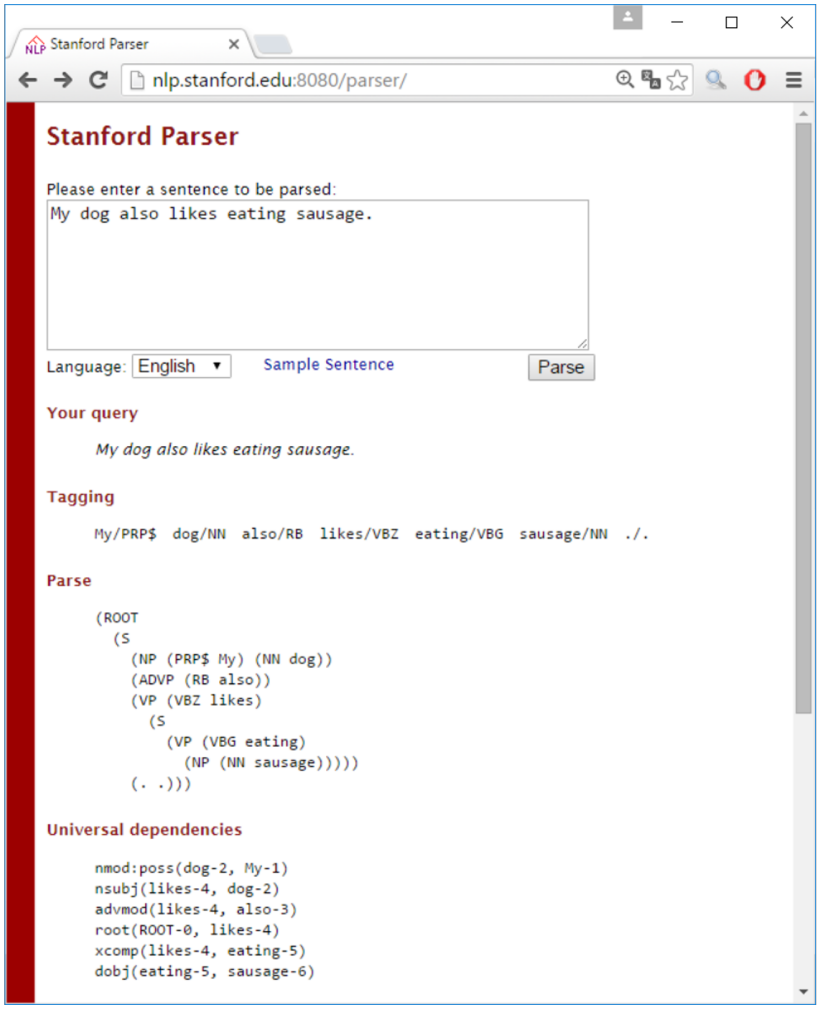


Fig. 6.13: Stanford parser example

The sample sentence `My dog also likes eating sausage.` is parsed. The PoS tagging and parsing results are displayed. There are two representations of the parsing result: a syntax tree representation and a typed dependency representation. The typed dependencies representation is usually easier to understand by people without linguistic expertise who want to extract textual relations from a text.

The Stanford parser can be integrated into Java applications as a JAR file. To integrate it into applications implemented in other programming languages, it can be invoked on the operating system level. Alternatively, extensions or ports exist for a number of programming languages, including PHP, Python, Ruby, and C#. Finally, the online parser can be invoked via HTTP.

NLP Frameworks: Apache UIMA

Apache UIMA (Unstructured Information Management Architecture)[23] is a powerful, mature NLP framework. It is used in many corporate applications, e.g. in IBM Watson.

[23] https://uima.apache.org/

UIMA allows NLP components to be integrated into a pipeline. Components implement NLP processing steps, e.g., tokenization, sentence splitting, PoS tagging, parsing, semantic analysis etc. Each component implements interfaces defined by the framework and provides self-describing metadata via XML descriptor files. The framework manages these components and the data flow between them. Components are written in Java or C++. The framework is also available in both programming languages. Third party components like the Stanford parser can be plugged into UIMA as well. For this, wrappers are needed as offered by uimaFIT[24] and DKPro[25].

See Fig. 6.14.

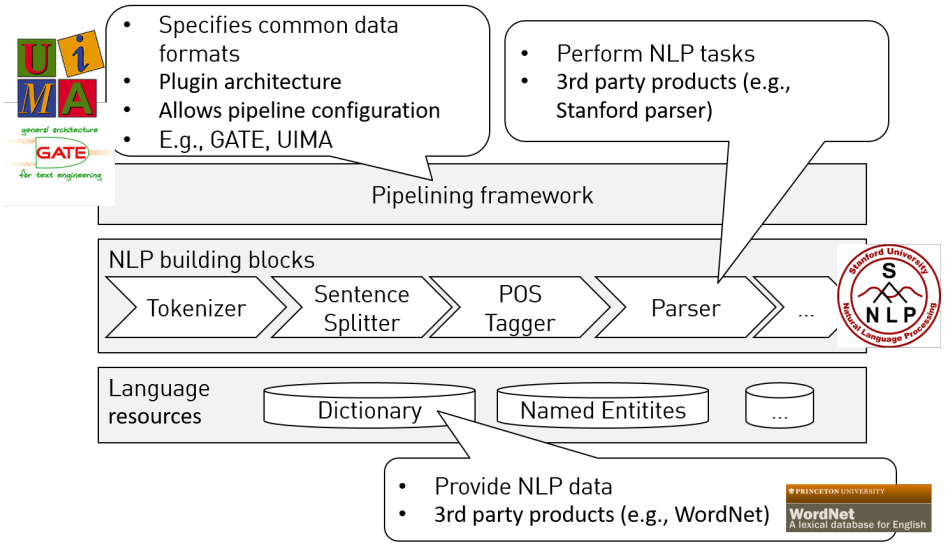


Fig. 6.14: NLP frameworks

UIMA is open source under the Apache license. The interfaces are approved as an OASIS[26] standard.

NLP Services: Named Entity Recognition with Dandelion API

There are numerous NLP services for completely different NLP tasks. As an example, I pick *Named Entity Recognition (NER)*. NER is a sub task of information extraction, locating and classifying elements in a text as persons, organizations, locations, etc.

Dandelion API[27] is a web service for semantic texts analytics, including NER. See a screenshot of an example in Fig. 6.15.

[24] https://uima.apache.org/uimafit.html
[25] https://www.ukp.tu-darmstadt.de/research/current-projects/dkpro/
[26] https://www.oasis-open.org/committees/uima
[27] https://dandelion.eu

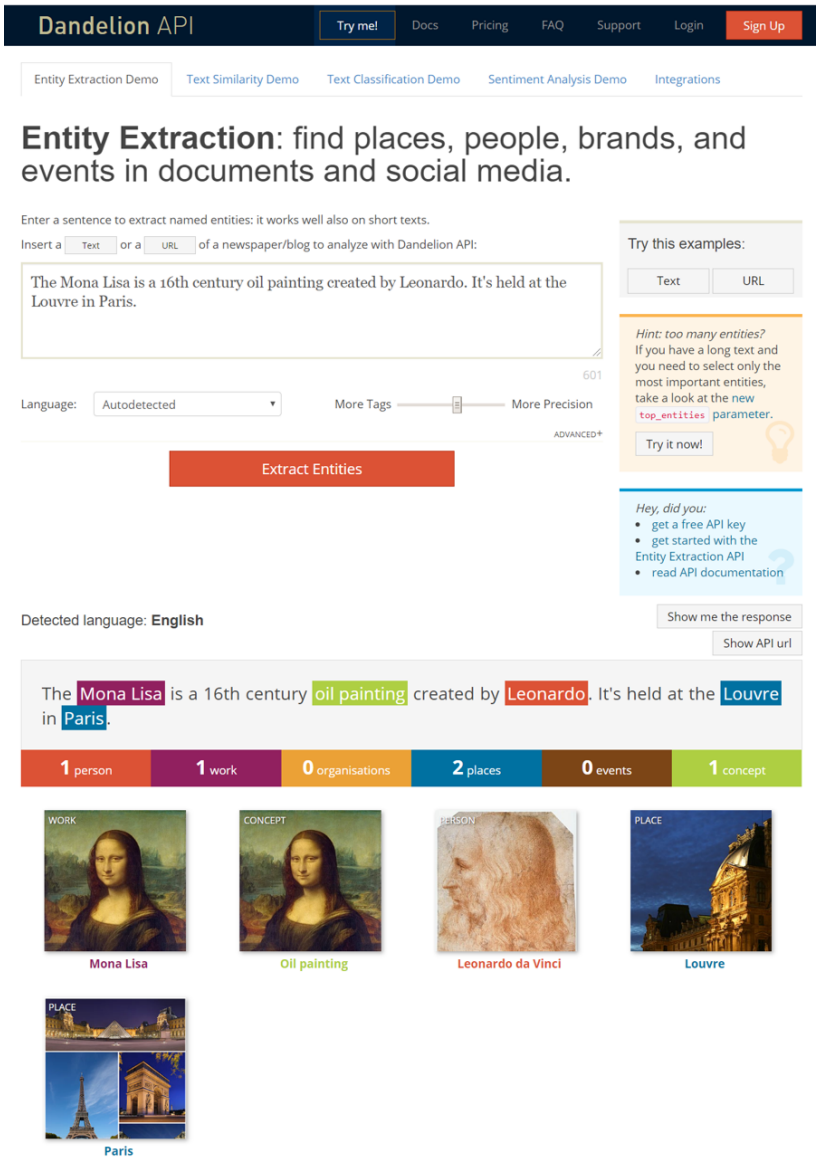


Fig. 6.15: NER example

In this example, the following text is analyzed:

```
The Mona Lisa is a 16th century oil painting created by Leonardo. It's held at the L\
ouvre in Paris.
```

Dandelion detected the language English and the following named entities:

1. Work Mona Lisa[28] with respective DBpedia link

[28] http://dbpedia.org/resource/Mona_Lisa

2. Concept Oil painting[29]
3. Person Leonardo da Vinci[30]
4. Place Louvre[31]
5. Place Paris[32]

The Dbpedia links allow retrieving additional information about the named entities, e.g., the birth date and death date of Leonardo da Vinci. The Dandelion API provides a JSON file containing all this information including including confidence scores for each named entity detected.

Dandelion can be configured to provide higher precision or more tags (higher recall). When favoring more tags, then the following additional named entity is identified:

Concept Tudor period[33]

This is a wrong identification. Although Leonardo da Vinci lived during the Tudor period, this period applies to England and not to Italy. This shows that NER, like all AI approaches, may produce erroneous results; just like humans who can misunderstand words in texts.

6.6 Quick Check

Answer the following questions.

1. Name and explain different areas of NLP.
2. Explain the levels of language understanding.
3. Explain the bag-of-words model, tf-idf and the n-gram model.
4. What is tokenization, sentence splitting, PoS tagging, and parsing?
5. What do language resources offer to NLP? Give examples.
6. What do NLP frameworks offer? Give example.
7. What do NLP web services offer? Give examples.

[29] http://dbpedia.org/resource/Oil_painting
[30] http://dbpedia.org/resource/Leonardo_da_Vinci
[31] http://dbpedia.org/resource/Louvre
[32] http://dbpedia.org/resource/Paris
[33] http://dbpedia.org/resource/Tudor_period

7. Computer Vision

Computer vision (CV) is a wide field of AI. It is all about processing images: still images and moving images (videos) / analyzing and generating. Relevant questions are:

- How to retrieve and classify images and videos, e.g., which photos depict certain people?
- What is depicted in an image / which situation takes place in a video, e.g., is there a danger of a vehicle collision?
- How to generate images / videos from an internal representation, e.g., in a computer game?

Fig 7.1. shows computer vision in the AI landscape.

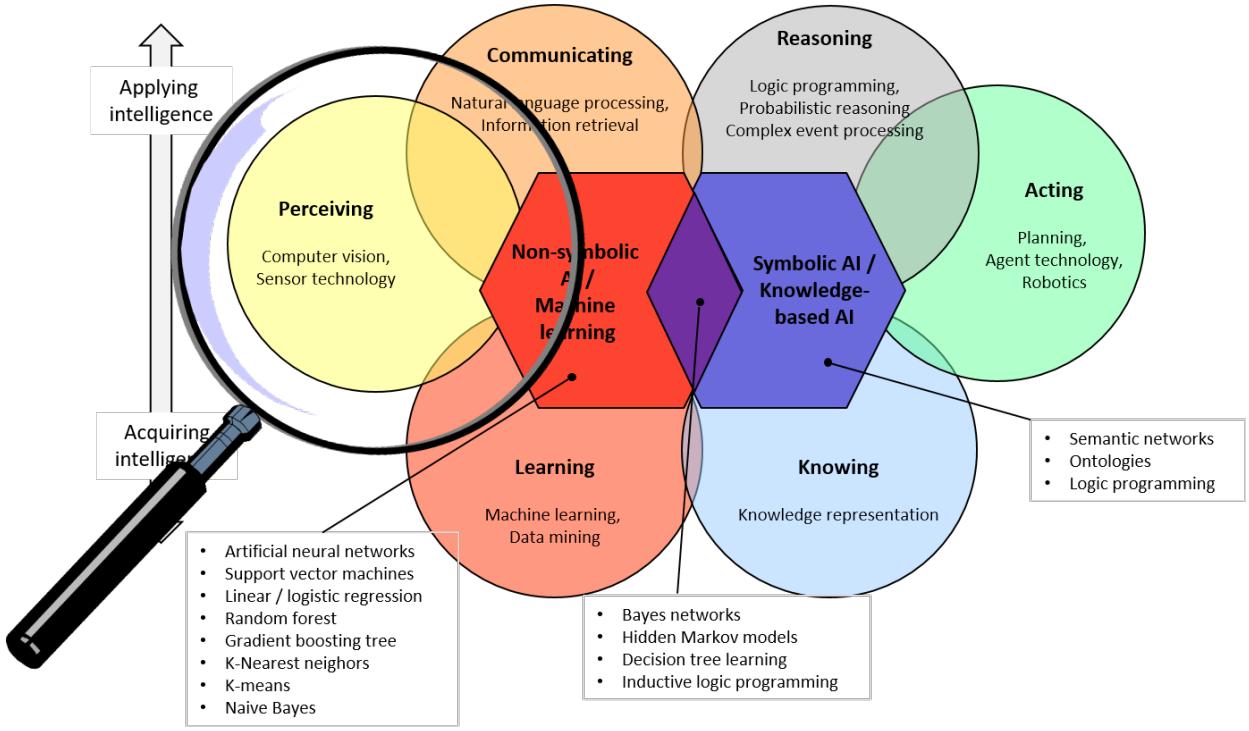


Fig. 7.1: Computer vision in the AI landscape

Computer vision can be assigned to the ability of “perceiving”. In the following section, I will briefly introduce prominent computer vision applications.

7.1 Computer Vision Applications

Optical Character Recognition (OCR)

Optical Character Recognition (OCR) is the extraction of text from images of typed, handwritten or printed text.

See Fig. 7.2. for the reconstruction of the text from the Wikipedia article on Michelangelo from a screenshot using FreeOCR[1].

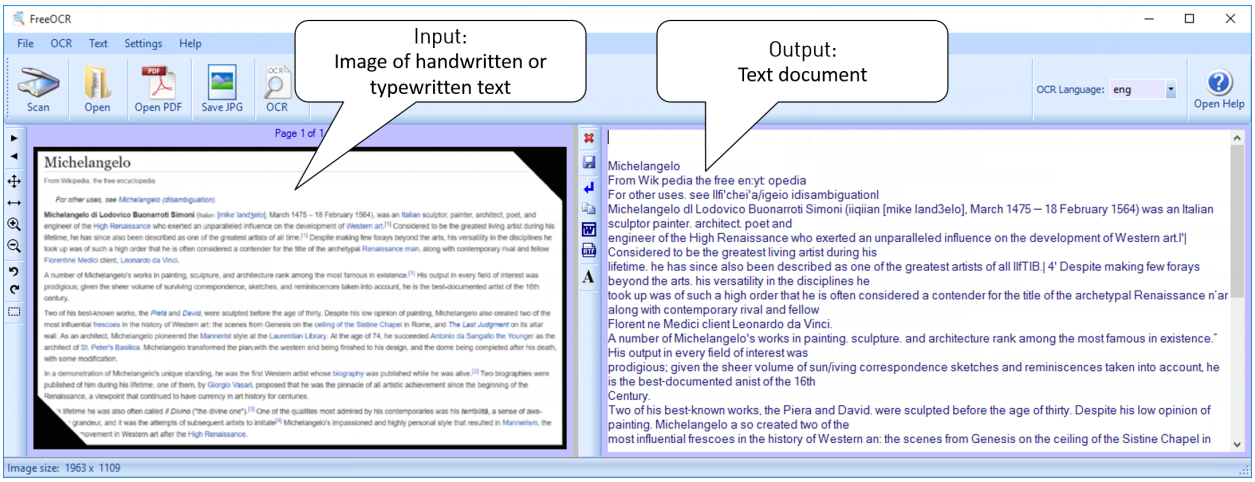


Fig. 7.2: OCR Example

Use cases include the following ones.

- Scanning addresses on letters
- Automatic number plate recognition
- Evaluating manually filled-out forms, e.g. in polls or surveys
- Data entry for business information systems, e.g. from invoices, bank statements and receipts etc.
- Checking documents, e.g. passports, drivers licenses, etc.
- Archiving and retrieving text documents for which only scans are available, e.g. Google Books
- Converting handwriting in real-time (pen computing)

Face Recognition

Face recognition is the detection of human faces in images or videos, as well as the identification of the respective persons.

See Fig. 7.3.

[1] http://www.freeocr.net

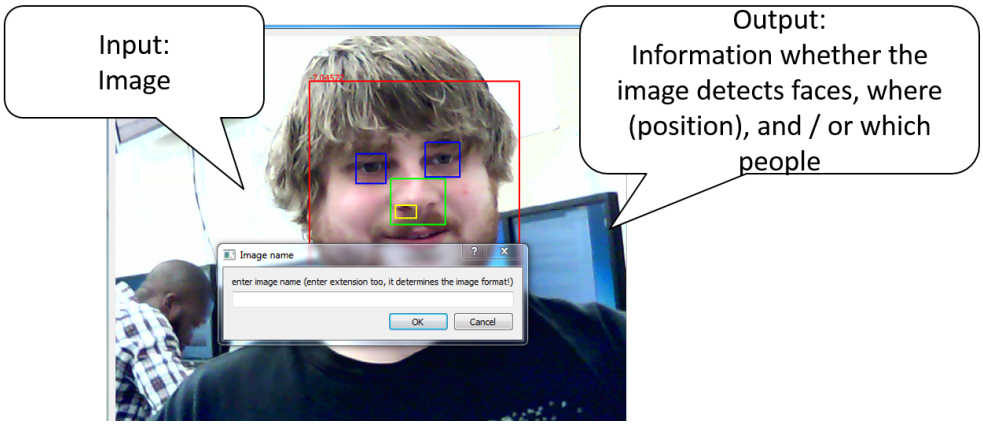


Fig. 7.3: Face Recognition

Use cases include the following ones.

- Identifying friends and family members in a private photo collection
- Face detection in cameras to focus and measure exposure on the face of the person being photographed
- Biometrical identification of people, e.g., for airport security
- Retrieving photos of prominent people from the Web

Image Processing

Image processing is the wide area of post-processing digital images, particularly photos, for several reasons. Examples are:

- Changing color, brightness, contrast, smoothing, noise reduction, etc.
- Retouching, e.g., removing red eyes
- Slicing, i.e., dividing an image into different sections to be used individually
- Image restoration; see Fig. 7.4.

Fig. 7.4: Image restoration

Medical Applications

Computer vision has numerous uses cases in medicine. Examples are:

- Generating images from examination procedures like PET[2]/CT[3], MRI[4], ultrasound images (2D and 3D) for inspection by doctors
- Detecting anomalies in PET/CT, MRI, ultrasound images etc. automatically.

See Fig. 7.5.

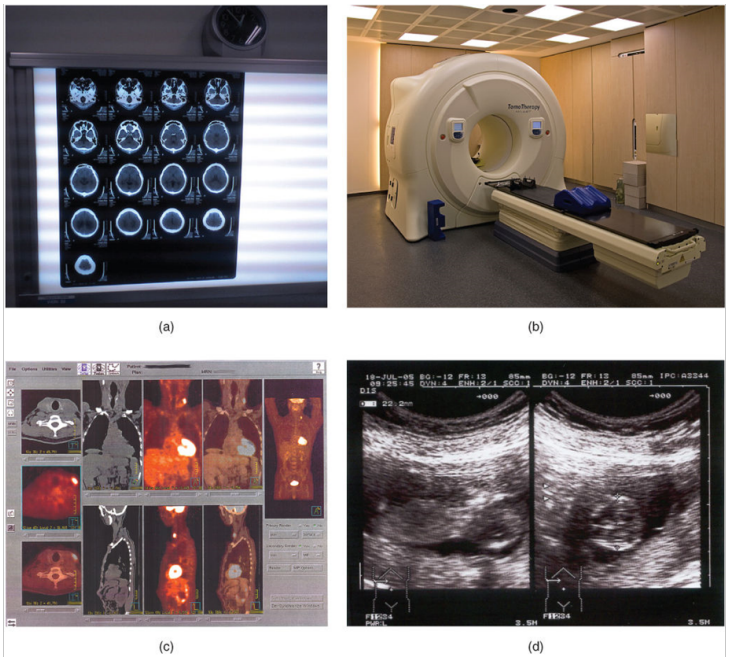


Fig. 7.5: Medical computer vision applications

Industrial and Agricultural Applications

Computer vision is increasingly used for automating processes in industry and agriculture. Examples are:

- Quality management in manufacturing processes
- Robot control in manufacturing processes
- Sorting fruits in agriculture

[2] Positron_Emission_Tomography
[3] Computed_Tomography
[4] Magnetic_Resonance_Imaging

See Fig. 7.6.

Fig. 7.6: Industrial computer vision application

Automotive Applications

Computer vision applications are, today, state-of-the-art in modern cars. Examples are:

- Parking aid and automatic parking
- Collision warning
- Road sign detection
- Autonomous driving

See Fig. 7.7.

Fig. 7.7: Automotive computer vision application

Military, Aviation and Aerospace Applications

Also in military and aviation and aerospace industries, computer vision is applied. Examples are:

- Collision detection
- Drone and missile navigation

- Detection of enemy soldiers or vehicles
- Autonomous vehicles (see Fig. 7.8)

Fig. 7.8: Autonomous space vehicle

Computer Games and Cinema

Computer games are mostly visual. Also in modern cinema, computer vision is heavily used for special effects. Examples are:

- Generation of moving images in computer games
- Generation of images and scenes in animated movies
- Motion capturing for digitizing videos taken from human actors; see Fig. 7.9.

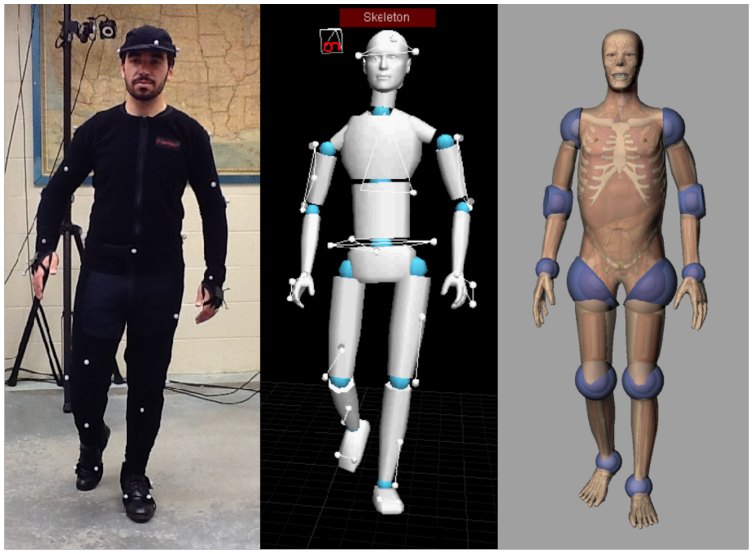


Fig. 7.9: Motion capturing

7.2 Computer Vision Tasks and Approaches

As computer vision is a wide field, there are many groups of tasks that may or may not be relevant in a concrete project. This is a simple grouping of computer vision tasks:

1. *Image acquisition*: Obtaining image data from light-sensitive cameras, ultra-sonic cameras, radars, range sensors, PET/CT/MRI/ultrasound devices, etc. The image data may be 2D or 3D / still images or sequences (video).
2. *Pre-processing*: Preparing the image data for further processing, e.g. by scaling, noise reduction, contrast enhancement, etc. Approaches include filtering and transformation algorithms.
3. *Feature extraction*: Identifying lines, edges, ridges, corners, texture, etc. in images. In use are specific algorithms, e.g., for edge detection.
4. Segmentation: Identifying image regions of particular interest, e.g. faces in photos. Machine learning approaches are used.
5. *High-level processing*: Application-specific image processing, e.g., classification, image recognition, scene analysis, etc. Machine learning approaches as well as other AI approaches for decision making are used.
6. *Image Generation*: generating images (still or moving) from an internal representation (usually 3D). Specific rendering algorithms are used.

7.3 Services and Product Maps

Fig. 7.10 shows the services map for computer vision.

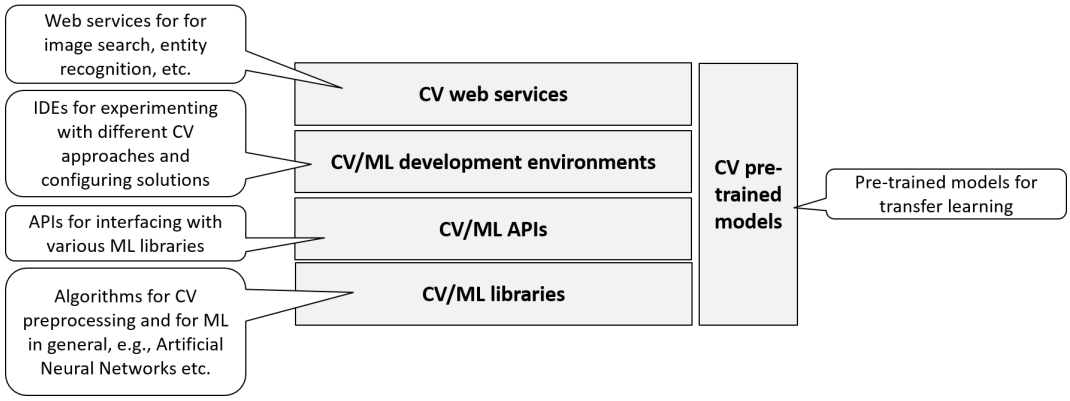


Fig. 7.10: Computer vision services map

Like for machine learning, there are products available on the library level, the framework level, and the web service level.

- *CV / ML algorithms and libraries*: Algorithms for image pre-processing and feature extraction as well as machine learning libraries.

- *CV / ML APIs*: APIs for interfacing with various ML libraries
- *CV / ML development environments / frameworks*: IDEs[5] and frameworks for experimenting with different computer vision approaches and configuring solutions.
- *CV web services*: Web services for image search, named entity recognition, etc.
- *CV pre-trained models*: Pre-trained ML models for transfer learning CV tasks.

Fig. 7.11 shows the product map for computer vision.

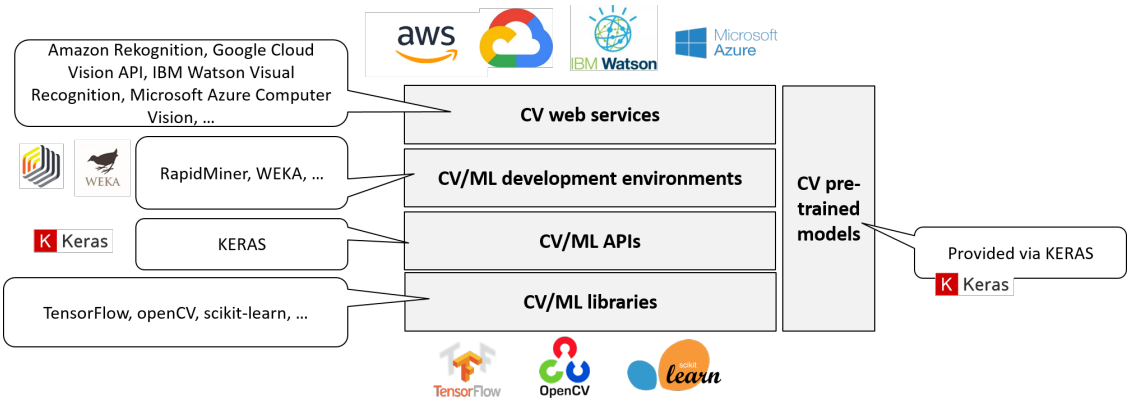


Fig. 7.11: Computer vision product map

TensorFlow[6] and OpenCV[7] are examples for CV / ML libraries. RapidMiner[8] is an IDE for machine learning. Keras[9] is a Python ML library, interfacing to TensorFlow, CNTK, or Theano. Examples for CV web services are: Autokeyword[10] and clarifai[11] for entity recognition, tineye[12] and Google image search[13] for image retrieval. The major vendors Google, Amazon, IBM and Microsoft offer web services for CV tasks.

More products and details can be found in the appendix.

7.4 Examples

In this section I present a few CV technologies by example.

Example: OCR with Deep Learning using TensorFlow (Yalçın, 2018)

TensorFlow[14] is an open source Python library for machine learning. It was developed by members of Google's Machine Intelligence research organization.

[5] Integrated_Development_Environments
[6] https://www.tensorflow.org
[7] http://opencv.org
[8] https://rapidminer.com
[9] https://keras.io
[10] http://autokeyword.me
[11] http://www.clarifai.com
[12] https://www.tineye.com
[13] https://www.google.de
[14] https://www.tensorflow.org/

The simple OCR (Object Character Recognition) example is taken from the online tutorial[15] (Yalçın, 2018). The task is to recognize digits from images where each image contains exactly one digit. See Fig. 7.12.

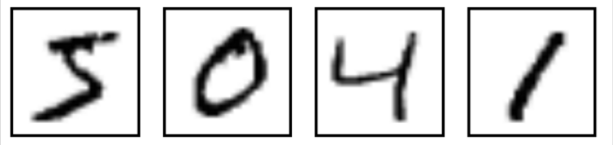


Fig. 7.12: Sample images with digits (Tensorflow, 2016)

MNIST

The images are taken from the MNIST[16] database of handwritten digits, available for learning computer vision techniques. Each image is 28 pixels by 28 pixels. Those pixels can be interpreted as an 28x28 array of numbers. See Fig. 7.13.

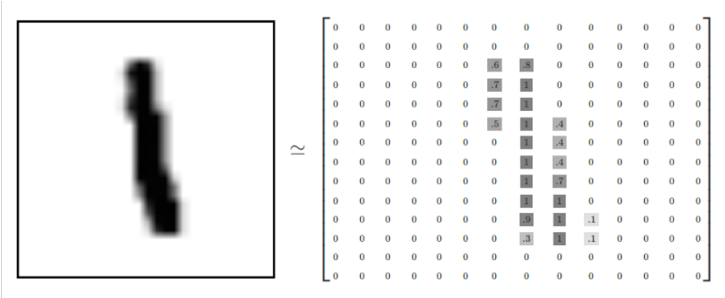


Fig. 7.13: Interpreting an image as array of pixels (Tensorflow, 2016)

The array may be flattened as a vector of 28x28 = 784 numbers which will be used as input for machine learning. 55.000 training images are available, all categorized with the digit (0...9) they represent. See Fig. 7.14. The categories are *one-hot encoded*. This means that there are 10 columns in the training data set, one for each digit. If, e.g., the image depicts the digit 5 then in the column for digit 5 there will be an entry 1 and in all other columns there will be an entry 0.

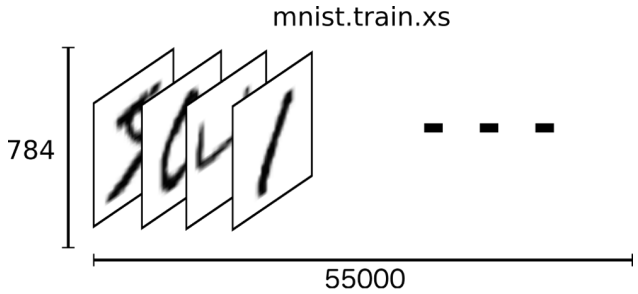


Fig. 7.14: Representation of MNIST training data (Tensorflow, 2016)

[15] https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d
[16] http://yann.lecun.com/exdb/mnist

Deep Learning

Deep learning has become the de-facto standard for image processing and is used and explained in this tutorial. See Fig. 7.15 for a typical deep learning network topology for image processing.

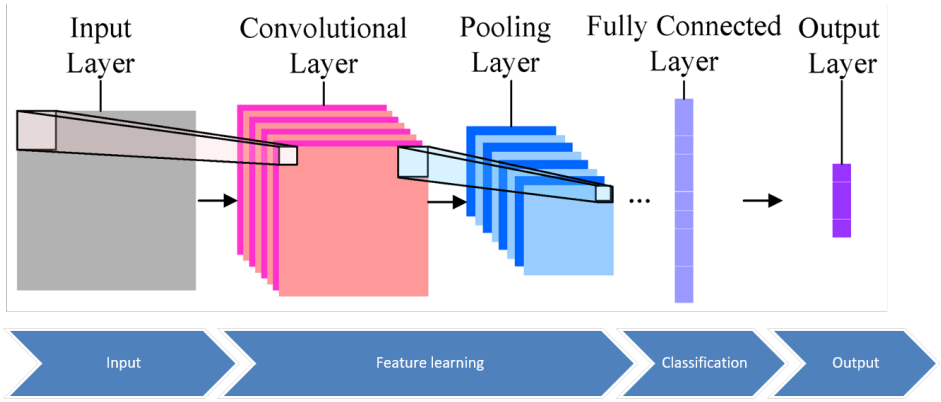


Fig. 7.15: Deep learning for image processing, adapted from (Ananthram, 2018)

The input layer of the deep neural network are neurons representing each pixel value of the images to be classified: 784 values in the MNIST example. The output of the network are neurons representing each class, i.e., one neuron for each digit 0…9. Between input layer and output layer there are several *convolutional layers, pooling layers* and *fully connected layers*. Each output of one layer is input to the next layer. This type of deep neural network is also called *convolutional neural network (CNN)*.

Convolutional layers are used to extract features from the images, e.g., edges. The idea is to use a small pixel filter (light blue 3x3 matrix in Fig. 7.16) which is successively compared with the pixels of the image (blue 5x5 matrix in Fig. 7.16). The comparison is performed simply by computing the dot product: the higher the dot product, the better the match. The comparison is performed step by step, eg., with a one pixel shift at each step. The result is a smaller matrix than the original pixel matrix (yellow 3x3 matrix in Fig. 7.16). The resulting matrix preserves the relationship between different parts of an image while decreasing the complexity.

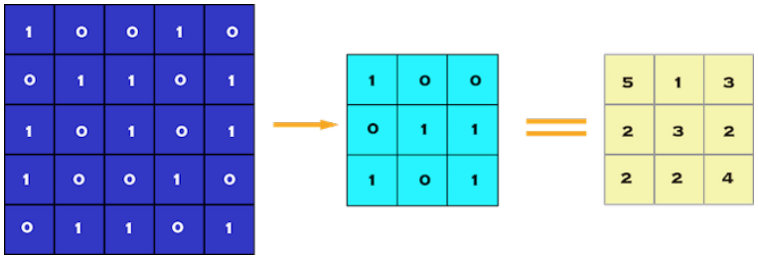


Fig. 7.16: Convolutional layer (Yalçın, 2018)

It is common to insert a pooling layer after each convolutional layer. A pooling layer further decreases complexity by considering parts of a matrix (differently colored 2x2 matrices in Fig. 7.17) and computing a simple aggregation like the maximum of numbers in this part (*max pooling*). The

resulting matrix is smaller, e.g., 2x2 in Fig. 7.17.

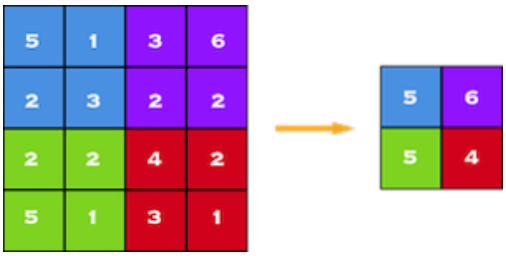


Fig. 7.17: Pooling layer (Yalçın, 2018)

After successions of convolutional layers and pooling layers for reducing complexity while focusing on relevant features, a set of fully connected layers are used for classification. As the name suggests, in fully-connected layers, each neuron of one layer is connected with all neurons of the next layer. See Fig. 7.18.

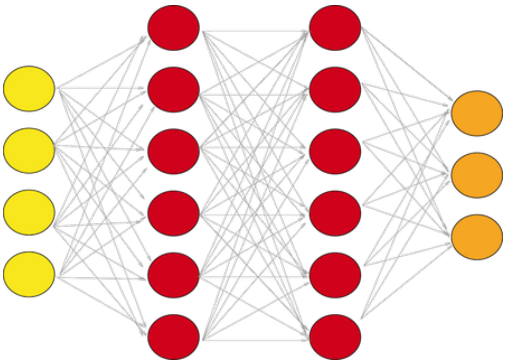


Fig. 7.18: Fully connected layers (Yalçın, 2018)

Keras and TensorFlow

I will now explain parts of the Keras and TensorFlow code from (Yalçın, 2018). Keras and Tensorflow allow importing and downloading the MNIST dataset directly from their API.

```
import tensorflow as tf
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

It is important to access the shape of the dataset to channel it to the convolutional layers. This is done using the `shape` attribute of `numpy` arrays.

```
x_train.shape
```

The result is `(55000, 28, 28)`. `55000` represents the number of images in the train dataset and `(28, 28)` represents the size of the image: 28 x 28 pixels.

To be able to use the dataset in Keras, the data must be normalized as it is always required in neural networks. This can be achieved by dividing the RGB codes by 255. Furthermore, the data format required by the API must be met. Here, the three-dimensional arrays must be converted to four-dimensional arrays.

```
# Reshaping the array to 4-dims so that it can work with the Keras API
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)        # Making sure that the values are float so that we can get\
 decimal points after division
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')        # Normalizing the RGB codes by dividing it to the \
max RGB value.
x_train /= 255
x_test /= 255
```

The architecture of the simple deep neural network has the following layers:

1. A convolutional layer
2. A max pooling layer
3. A flatten layer to avoid overfitting by disregarding some of the neurons
4. 2 dense layers for classification
5. A flatten layer to convert 2D arrays to a 1D array.

The following code implements this architecture.

```
# Creating a Sequential Model and adding the layers
model = Sequential()
model.add(Conv2D(28, kernel_size=(3,3), input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten()) # Flattening the 2D arrays for fully connected layers
model.add(Dense(128, activation=tf.nn.relu))
model.add(Dropout(0.2))
model.add(Dense(10,activation=tf.nn.softmax))
```

The following code specifies an optimizer and loss function that uses a metric for training.

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Now the model can be trained. The value `epochs` specifies how often all training data shall be used.

```
model.fit(x=x_train,y=y_train, epochs=10)
```

Finally, you may evaluate the trained model as follows.

```
model.evaluate(x_test, y_test)
```

The result is an accuracy of 98.5% on the test dataset. This is quite a good result for such a simple model and only 10 epochs of training. If, however, an error of 0.1% is not tolerable, then the model can be optimized, e.g., by experimenting with more epochs, different optimizers or loss functions, more layers, different hyperparameters etc.

The trained model can now be used to predict the digit in an unknown image.

```
image_index = 4444
plt.imshow(x_test[image_index].reshape(28, 28),cmap='Greys')
pred = model.predict(x_test[image_index].reshape(1, img_rows, img_cols, 1))
print(pred.argmax())
```

In this example, the digit `9` is returned which is, indeed the correct classification of the image with index `4444`. See Fig. 7.19.

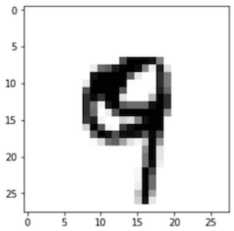


Fig. 7.19: Image example (Yalçın, 2018)

Using a ML library like TensorFlow requires a considerably deeper understanding of the algorithms at hand than working with a ML IDE like RapidMiner. However, it allows optimizing an application more specifically.

Example: Transfer Learning with Keras (Ananthram, 2018)

Transfer Learning

Training a deep neural network from scratch is possible for small projects. However, most applications require the training of very large neural networks and this takes huge amounts of processed data and computational power. Both are expensive.

This is where *transfer learning* comes into play. In transfer learning, the pre-trained weights of an already trained model (e.g., trained on millions of images belonging to thousands of classes, on several high power GPU's for several days) are used for predicting new classes. The advantages are obvious:

1. There is no need of an extremely large training dataset.
2. Not much computational power is required, as pre-trained weights are used and only the weights of the last few layers have to be learned.

To understand how transfer learning works re-consider the architecture of a convolutional neural network in Fig. 7.15. Feature learning takes place in a succession of convolutional layers and pooling layers. E.g., the filters on the first few layers may learn to recognize colors and certain horizontal and vertical lines. The next few layers may recognize trivial shapes using the lines and colors learned in the previous layers. Then the next layers may recognize textures, then parts of objects like legs, eyes, noses etc.

The classification as such takes place in the few fully connected layers at the end. When classifying new things, e.g., individual dog breeds, then all the pre-trained features like colors, lines, textures etc. may be re-used and only classifying the dog breeds need to be trained. All this helps in making the training process much faster, requiring much less training data compared to training the neural network from scratch.

Keras and MobileNet

MobileNet[17] is a pre-trained model which gives reasonably good image classification accuracy while occupying relatively little space (17 MB). In this example, the ML library Keras[18] is used. Keras supports transfer learning by accessing several pre-trained models via the API.

Building the model requires the following steps:

1. Importing the pre-trained model and adding the dense layers
2. Loading training data
3. Training the model.

I will explain those steps in the next sections.

Importing Pre-trained Model and Adding Layers

The following Python code imports MobileNet. Mobilenet's last layer consists of 1000 neurons, one for each class for which it was originally trained. Since we want to train the network for different classes, e.g., dog breeds, we have to discard the last layer. If the dog breed classifier is to identify 120 different breeds, then we need 120 neurons in the final layer. This can be done using the following code.

[17] https://keras.io/applications/#mobilenet
[18] https://keras.io

```
base_model=MobileNet(weights='imagenet',include_top=False) #imports the mobilenet mo\
del and discards the last 1000 neuron layer.
x=base_model.output
x=GlobalAveragePooling2D()(x)
x=Dense(1024,activation='relu')(x) #we add dense layers so that the model can learn \
more complex functions and classify for better results.
x=Dense(1024,activation='relu')(x) #dense layer 2
x=Dense(512,activation='relu')(x) #dense layer 3
preds=Dense(120,activation='softmax')(x) #final layer with softmax activation
```

Next we make a model based on the architecture provided.

```
model=Model(inputs=base_model.input,outputs=preds)
```

As we will be using the pre-trained weights, we have to set all the weights to be non-trainable.

```
for layer in model.layers:
    layer.trainable=False
```

Loading Training Data

The following code loads the training data, which must be in a particular format, from a folder.

```
train_datagen=ImageDataGenerator(preprocessing_function=preprocess_input)
train_generator=train_datagen.flow_from_directory('path-to-the-main-data-folder',
                                                  target_size=(224,224),
                                                  color_mode='rgb',
                                                  batch_size=32,
                                                  class_mode='categorical',
                                                  shuffle=True)
```

Training the Model

The following code performs compiling and training of the model on the dataset.

```
model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
step_size_train=train_generator.n//train_generator.batch_size
model.fit_generator(generator=train_generator,
                    steps_per_epoch=step_size_train,
                    epochs=10)
```

The model can now be evaluated and used for predicting classes in new images.

Example: Named Entity Recognition with the Autokeyword.me Web Service

Web services for image tagging like Autokeyword.me[19] allow classifying and tagging images of any kind. See Fig. 7.20.

Fig. 7.20: Tagging example with Autokeyword.me

In this example, the painting "Isabella" by John Everett Millais (1829-1896)[20] is analyzed. Tagging results include concepts like woman and man as well as attributes like attractive and smiling. They are all provided with a degree of probability, indicated by a bar at the circles. For example, the

[19] http://autokeyword.me
[20] https://upload.wikimedia.org/wikipedia/commons/8/82/John_Everett_Millais_-_Isabella.jpg

category man is more probable than boy and, indeed, there are men but no single boy depicted in the painting. So, the category “boy” is a false positive. There are also false negatives, e.g., the dogs on the painting were not recognized.

Using an image tagging web service is easy and simply requires invoking an API. However, if the results are not suitable for an application use case, there is no way of optimizing the solution like when using a library or a framework.

7.5 Quick Check

Answer the following questions.

1. Name applications of computer vision
2. What are the main tasks and approaches in computer vision?
3. Which libraries / frameworks / web services can be used for computer vision?
4. Explain deep learning. What are convolutional layers? What are pooling layers? What are dense layers?
5. Explain transfer learning
6. What are the advantages / disadvantages of using a web service compared to a library or framework?

8. Complex Event Processing

Complex event processing (CEP) allows processing streams of event data and deriving conclusions from them.

Fig. 8.1. shows CEP in the AI application landscape.

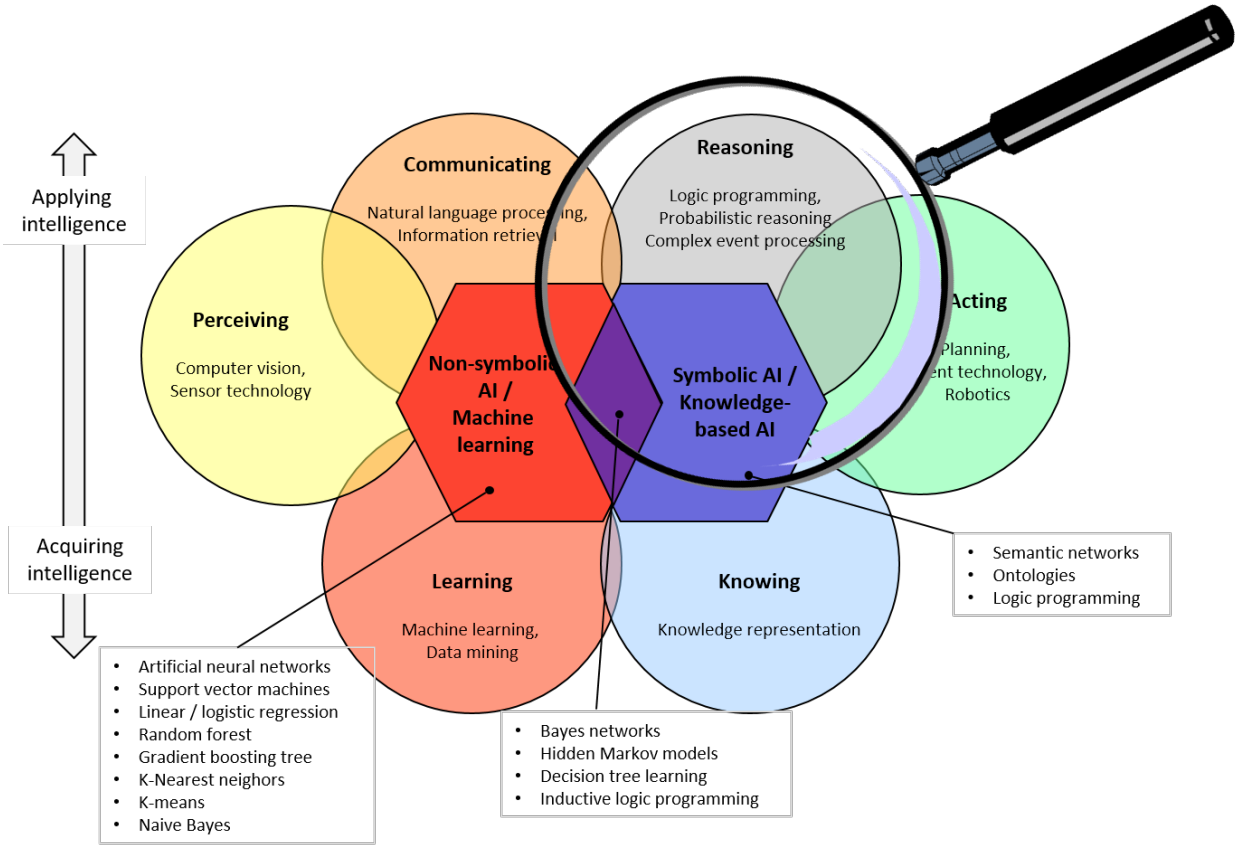


Fig. 8.1: CEP in the AI landscape

CEP can be assigned to the ability of “reasoning”.

Examples of CEP applications are:

1. Detecting fraud based on payment transaction data
2. Making selling / buying decision based on stock market feeds
3. Switching active traffic signs based on traffic data
4. Making purchase recommendations based on click stream analysis in webshops

All those applications have in common that complex decisions (e.g., fraud alert, selling / buying etc.) are made in real-time based on events (e.g., payment transactions, stock market feeds etc.).

8.1 Foundations

What is an *event* in this context? An event is something notable that happens. What is notable entirely depends on the application use case. Examples are:

1. A financial transaction
2. An airplane landing
3. A sensor outputs a reading
4. A change of state in a database
5. A key stroke performed by a user of a web app
6. A historic event, e.g., the French Revolution

An event takes place in the real world. An *event object* is a data record that represents an event.

Examples are:

1. A Purchase order record
2. A stock tick message
3. A record with an RFID sensor reading

An *event type (a.k.a. event class)* specifies the common structure of related event objects, i.e., their attributes and data types, e.g., `PurchasingEvent` with attributes `timestamp`, `buyer`, `product` and `price`.

In CEP literature, often the term “event” is also used for event objects and event types. From the context it is usually clear whether the real-world event is meant, the concrete data record or its type.

A *CEP engine* allows specifying *CEP rules* in a *CEP rule language* and executing them. CEP rules specify patterns that can match events in an *event stream*. *Message brokers* are platforms to manage streams of messages, in particular event objects.

Fig. 8.2. illustrates the difference between data access in database management systems (DBMS) and CEP engines.

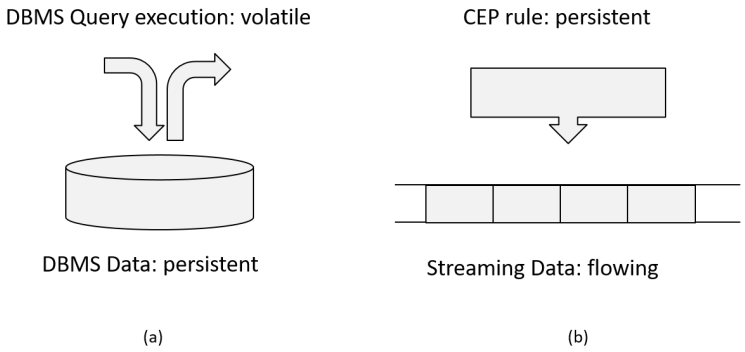


Fig. 8.2: Data access in (a) DBMS versus (b) CEP engine

A DBMS stores persistent data. A query, e.g., formulated in SQL, executes instantaneously and returns a query result based on the current state of the persistent data. In contrast, the data source for CEP is a flowing stream of events. CEP rules are persistent, with the CEP engine constantly trying to match CEP rules with the events. Whenever a CEP rule matches, a higher (complex) event is being generated which may trigger certain actions.

8.2 Application Example: Fault Detection in the Smart Factory

I will explain CEP with the application example of one of my research projects: fault detection in the smart factory (Beez et al., 2018).
The use case is visual quality inspection in glass production. A glass inspection machine consists of cameras and lights as well as servers with inspection software. An error may occur in every component of the setup, including the interconnections, and on both the software level and the hardware level. Errors in the inspection machine may lead to uninspected glass or glass of unknown quality and thus impact the plant yield. Common inspection machine errors are camera failures, network errors, lighting issues, or incorrect configuration of external system parameters. Often, hints for these kinds of problems can be found provided that the distributed log information the system generates at runtime is considered as a whole.

Hereafter, I describe some typical CEP rules in this application context. We assume log messages from various components of the inspection machine to to be normalized to a common event format and provided by a message broker. Parts of CEP rules are presented in pseudo code, oriented at CEP languages like for Apache Flink.

Filtering

Filtering is a simple form of CEP; see Fig. 8.3.

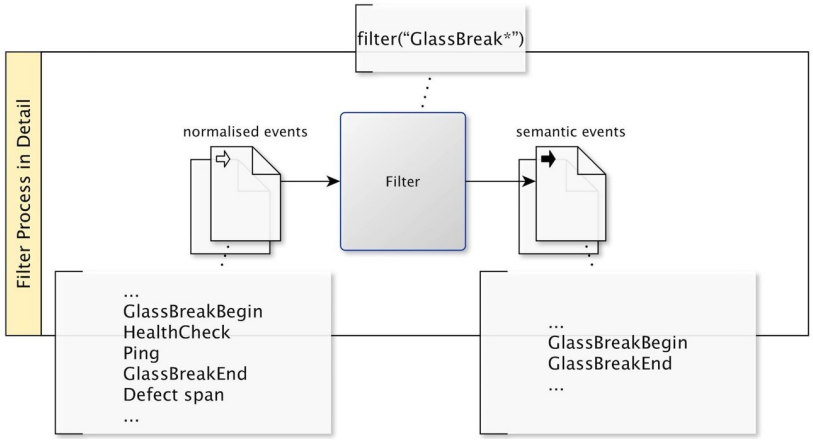


Fig. 8.3: CEP filtering (Kaupp et al., 2017)

Assuming that events have types like `GlassBreakBegin`, `GlassBreakEnd`, `DefectSpan` etc., then the CEP rule `filter("GlassBreak*")` will filter all events of which type matches the regular expression `"GlassBreak*"`. In the example, events with type `GlassBreakBegin`, `GlassBreakEnd` are filtered which indicate a break in the glass being manufactured.

Pattern Matching

Pattern matching allows detecting patterns in successions of events. Consider the pattern shown in Fig. 8.4 being applied to the stream of events resulting form the filtering rule from Fig. 8.3.

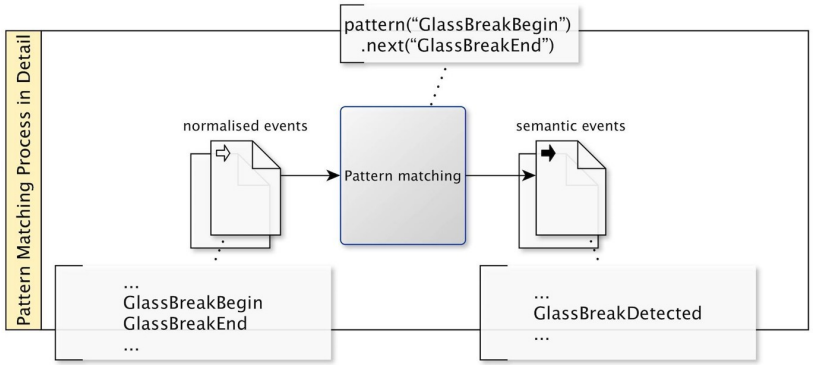


Fig. 8.4: CEP pattern matching (Kaupp et al., 2017)

The condition `pattern("GlassBreakBegin".next("GlassBreakEnd")` in the CEP rule matches if a `GlassBreakBegin` event is immediately followed by a `GlassBreakEnd` event. In this case, a higher-level semantic event `GlassBreakDetected` can be generated and inserted into a queue of the message broker.

Value Progression Analysis

Fig. 8.5 shows a *value progression analysis*, analyzing the successive change of values in events.

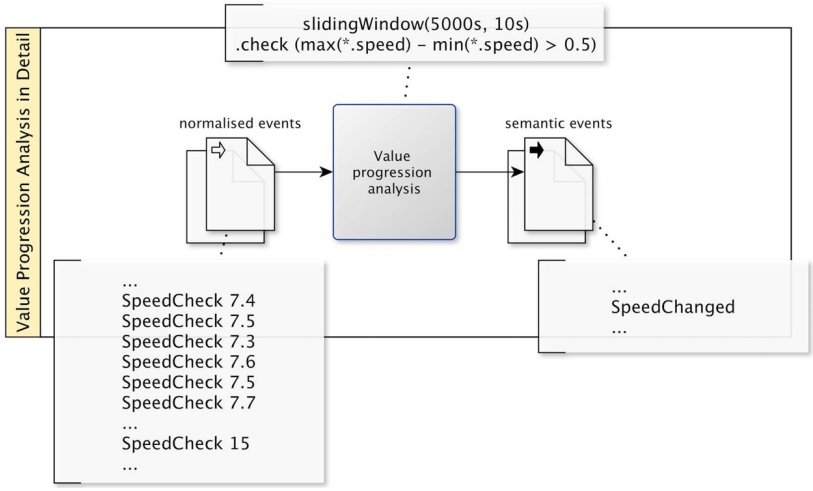


Fig. 8.5: CEP value progression analysis (Kaupp et al., 2017)

Each `SpeedCheck` event delivers a snapshot of the speed of the conveyor belt. The condition `slindingWindow(5000s, 10s).check (max(*.speed) - min(*.speed > 0.5)` in the CEP rule uses a sliding window. A sliding window is a subset of the last events in a certain time period (here 500 seconds) on an event stream. The second parameter (`10s`) indicates how often the window is considered. Within this time frame, the speed values of all events are considered. If the difference in speed exceeds a threshold (here `0.5`), then a new event `SpeedChanged` is generated and inserted into a queue of the message broker. A speed change may affect defect detection and, therefore, is important semantic information.

Time Progression Analysis

Fig. 8.6 shows a *time progression analysis*, analyzing the timing of events.

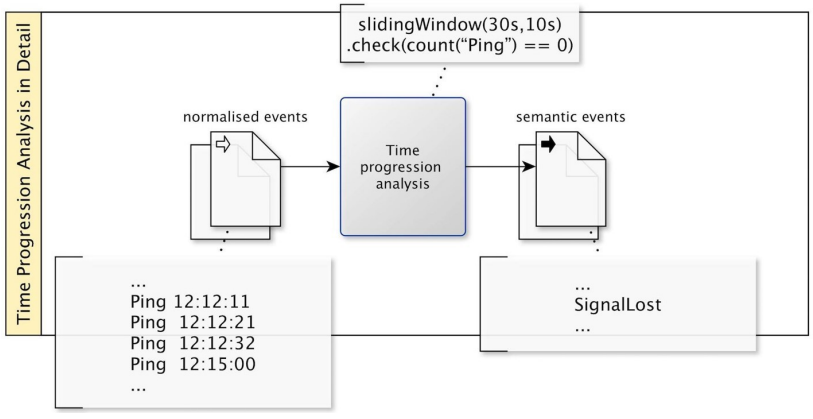


Fig. 8.6: CEP time progression analysis (Kaupp et al., 2017)

Each component within the glass inspection machine regularly sends `Ping` events. If no `Ping` event occurs within half a minute, it is considered as a loss of signal. The condition `slidingWindow(30s, 10s).check(count("Ping")==0)` in the CEP rule uses a sliding window of 30 seconds, which is

checked every 10 seconds. If no `Ping` event occurred within the time window, then a `SignalLost` event is generated.

Semantic Enrichment

CEP rules using filters, pattern matching, value progression analysis and time progression analysis can be applied iteratively. This allows incremental semantic enrichment, from primitive low level events to high-level semantic events. See Fig. 8.7.

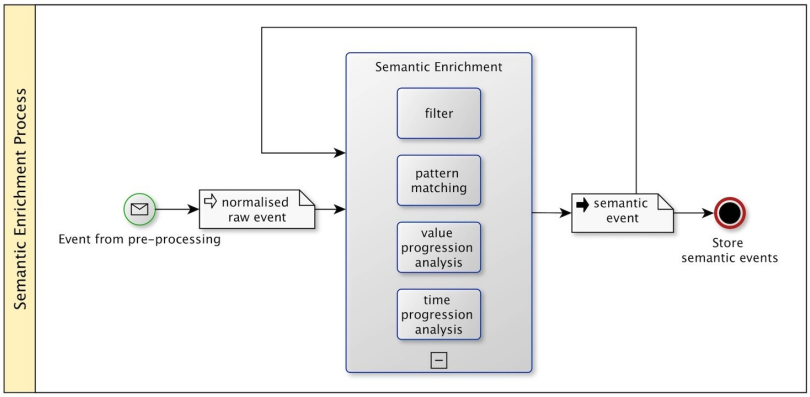


Fig. 8.7: CEP semantic enrichment (Kaupp et al., 2017)

Semantic events generated by CEP rules can be used as input for other CEP rules. For example, high conveyor belt speed typically implies a thinner glass layer. With the glass thickness, the properties of the defects change, and in the transition between different thicknesses many defects may occur. The semantic event `SpeedChanged` introduced above may then be used for generating a semantic `ThicknessChanged` event.

8.3 Services Map and Product Map

Fig. 8.8 shows the CEP services map.

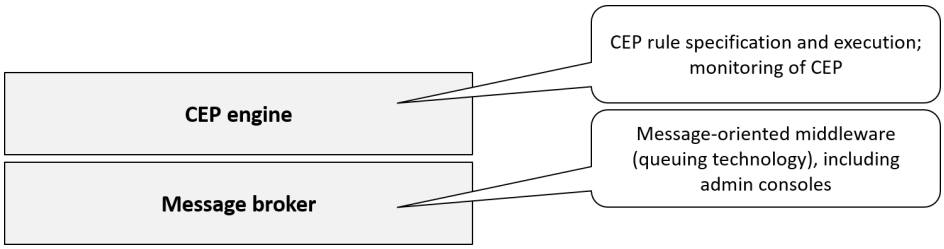


Fig. 8.8: CEP Services Map

Message brokers implement a message-oriented middleware, providing message queuing technology. The allow managing streams of event messages. They also provide consoles for administering

message queues. CEP engines allow specifying CEP rules and executing them. They also allow monitoring CEP.

Fig. 8.9 shows the CEP product map.

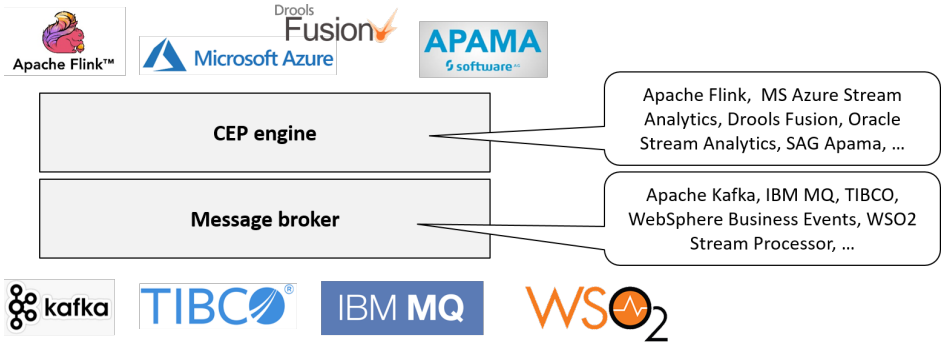


Fig. 8.9: CEP Product Map

All major IT vendors provide message brokers and CEP engines. Also, there are several production-ready open source solutions. Examples of message brokers are Apache Kafka, IBM MQ, TIBCO, WebSphere Business Events, and WSO2 Stream Processor. Examples of CEP engines are Apache Flink, MS Azure Stream Analytics, Drools Fusion, Oracle Stream Analytics, and SAG Apama.

8.4 Quick Check

Answer the following questions.

1. What is CEP?
2. What is an event, event object and event type?
3. What is a message broker?
4. What is a CEP engine?
5. Explain the difference between data access in a DBMS versus a CEP engine?
6. Explain filter, pattern matching, value progression analysis and time progression analysis.
7. How can CEP be used for semantic enrichment?
8. Name prominent CEP products

9. Conclusions

Artificial Intelligence is relevant and ubiquitous. Speech controlled personal assistants, face recognition in cameras, learning spam filters in e-mail clients and AI in computer games are examples of AI applications in everyday use.

After all, AI applications are IT applications and software engineering skills are required for their development. "Applied Artificial Intelligence – An Engineering Approach" focuses on exactly those skills.

Many AI applications have implications on our society. Increasing automation in manufacturing and agricultural industries has a strong impact on the workforce. The use of autonomous vehicles will raise new legal issues when severe accidents will occur. Most critically, the advent of lethal autonomous weapons systems raise most severe ethical issues. See for example an open letter[1] signed by more than 1,000 AI professionals to "ban an offensive autonomous weapons beyond meaningful human control". So, developers of AI applications have to consider impacts of those applications on our society and act responsibly.

Finally, I would like to express my wish that this book will be a living document. Dear reader, I am most interested in your comments and opinions. Please, question and discuss the opinions stated in this book and, thereby, help improve it.

I look forward to your comments (bernhard.humm@h-da.de).

[1] http://futureoflife.org/open-letter-autonomous-weapons/

Literature

(Allemang and Hendler, 2011) Dean Allemang, James Hendler: "Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL". 2nd Edition Morgan Kaufmann Publishers Inc., San Francisco, 2011.

(American Heritage, 2001) "Behind the Cutting Edge: The Myth Of Artificial Intelligence". American Heritage, Volume 52, Issue 1, 2001. Online resource[2] (accessed 2018/07/14)

(Ananthram, 2018) Aditya Ananthram: "Deep Learning For Beginners Using Transfer Learning In Keras" Towards Data Science[3], Oct 17, 2018 (accessed 2020/03/24).

(Baeza-Yates, 2018) Ricardo Baeza-Yates: "Bias on the Web". Communications of the ACM, June 2018, Vol 61, No. 6, pp 54-61, 10.1145/3209581.

(Beez et al., 2018) Ulrich Beez, Lukas Kaupp, Tilman Deuschel, Bernhard G. Humm, Fabienne Schumann, Jürgen Bock, Jens Hülsmann: "Context-Aware Documentation in the Smart Factory". In: Thomas Hoppe, Bernhard G. Humm, Anatol Reibold (Eds.): Semantic Applications - Methodology, Technology, Corporate Use. pp. 163-180. Springer Verlag, Berlin, 2018. ISBN 978-3-662-55432-6.

(Bowker and Star, 1999) G. C. Bowker, S. L. Star: "Sorting Things Out: Classification and its consequences". MIT Press, Cambridge, 1999.

(Busse et al., 2015) Johannes Busse, Bernhard Humm, Christoph Lübbert, Frank Moelter, Anatol Reibold, Matthias Rewald, Veronika Schlüter, Bernhard Seiler, Erwin Tegtmeier, Thomas Zeh: "Actually, What Does "Ontology" Mean? A Term Coined by Philosophy in the Light of Different Scientic Disciplines". Journal of Computing and Information Technology - CIT 23, 2015, 1, 29–41.

(Ege et al., 2015) Börteçin Ege, Bernhard Humm, Anatol Reibold (Editors): "Corporate Semantic Web – Wie semantische Anwendungen in Unternehmen Nutzen stiften" (in German). Springer-Verlag, 2015.

(Galkin, 2016) Ivan Galkin: "Crash Introduction to Artificial Neural Networks". University of Massachusetts Lowell. Online resource[4] (accessed 2016/05/01).

(Goodman and Tenenbaum, 2016) Noah D. Goodman, Joshua B. Tenenbaum: "Probabilistic Models of Cognition". Online resource[5] (accessed 2016/05/25).

(Harriehausen, 2015) Bettina Harriehausen-Mühlbauer: "Natural Language Processing". Lecture Script, Hochschule Darmstadt - University of Applied Sciences, Computer Science Department. Darmstadt, Germany.

[2] https://www.americanheritage.com/content/myth-artificial-intelligence
[3] https://towardsdatascience.com/keras-transfer-learning-for-beginners-6c9b8b7143e
[4] http://ulcar.uml.edu/~iag/CS/Intro-to-ANN.html
[5] https://probmods.org/

(Hoppe, 2015) Thomas Hoppe: “Modellierung des Sprachraums von Unternehmen” (in German). In: Börteçin Ege, Bernhard Humm, Anatol Reibold (Editors): “Corporate Semantic Web – Wie semantische Anwendungen in Unternehmen Nutzen stiften” . Springer-Verlag, 2015.

(Hoppe et al., 2018) Thomas Hoppe, Bernhard G. Humm, Anatol Reibold (Eds.): Semantic Applications - Methodology, Technology, Corporate Use. Springer Verlag, Berlin, 2018. ISBN 978-3-662-55432-6.

(Humm, 2020) Bernhard G. Humm: Fascinating with Open Data: openArtBrowser. In Adrian Paschke, Clemens Neudecker, Georg Rehm, Jamal Al Qundus, Lydia Pintscher (Eds.): Proceedings of the Conference on Digital Curation Technologies (Qurator 2020). CEUR Workshop Proceedings Vol-2535, urn:nbn:de:0074-2535-7. Berlin, Germany, 2020.

(Kaupp et al., 2017) Lukas Kaupp, Ulrich Beez, Bernhard G. Humm, Jens Hülsmann: “From Raw Data to Smart Documentation: Introducing a Semantic Fusion Process”. In: Udo Bleimann, Bernhard Humm, Robert Loew, Stefanie Regier, Ingo Stengel, Paul Walsh (Eds): Proceedings of the Collaborative European Research Conference (CERC 2017), pp 83-94, Karlsruhe, Germany, 22-23 September 2017. ISSN: 2220-4164.

(More et al., 2004) Dana Moore, Aaron Helsinger, David Wells: “Deconfliction in ultra-large MAS: Issues and a potential architecture”. In Proceedings of the First Open Cougaar Conference, pp. 125-133. 2004.

(Mueller-Birn, 2019) Claudia Mueller-Birn: “What Is Beyond the ‘Human-Centered’ Design of AI-driven Systems?” Medium.com[6], Nov 2019 (accessed 2020-01-29).

(O’Neil 2016) Cathy O’Neil: “Weapons of Math Destruction”. Crown Books, 2016. ISBN 0553418815

(Russell and Norvig, 1995) Stuart Russell, Peter Norvig: “Artificial Intelligence: A Modern Approach”, Prentice-Hall, 1995.

(Russell and Norvig, 2013) Stuart Russell, Peter Norvig: “Artificial Intelligence: A Modern Approach”, 3rd Edition. Pearson, 2013.

(Tensorflow, 2016) Tensorflow:”MNIST for Beginners” Tutorial[7] (accessed 2016-05-01).

(Watson, 2010) Marc Watson: “Practical Semantic Web and Linked Data Applications, Java, Scala, Clojure, and JRuby Edition”. www.markwatson.com[8], 2010 (accessed 2016/05/25).

(Watson, 2011) Marc Watson: “Practical Semantic Web and Linked Data Applications, Common Lisp Edition”. www.markwatson.com[9], 2011 (accessed 2016/05/25).

(Watson, 2013) Marc Watson: “Practical Artificial Intelligence Programming with Java”. LeanPub, 2013.

(Yalçın, 2018) Orhan Gazi Yalçın: “Image Classification in 10 Minutes with MNIST Dataset”. Towards Data Science[10], Aug 19, 2018 (accessed 2020/03/24).

[6] https://medium.com/@clmb/what-is-beyond-the-human-centered-design-of-ai-driven-systems-10f90beb9574
[7] https://www.tensorflow.org/versions/r1.0/get_started/mnist/beginners
[8] https://www.markwatson.com/books/
[9] https://www.markwatson.com/books/
[10] https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d

Appendix: Product Tables

The following tables list products for different AI areas in addition to the ones listed in the respective product maps. The assignment to service categories are indicated by asterisks.

Machine Learning

Machine learning product table

Product	ML library	ML dev. env.	ML web service	Pre-trained model
aisolver[11]	*			
Amazon AWS Machine Learning[12]			*	
Apache Mahout[13]	*			
bigml[14]			*	
Caffe[15]	*			
CURRENNT[16]	*			
Deeplearning4j[17]	*			
eblearn[18]	*			
ELKI Data Mining[19]	*	*		
Encog[20]	*			
Fast Artificial Neural Network Library[21]	*			
Google Cloud Machine Learning[22]			*	
IBM Watson Machine Learning[23]			*	
Jaden[24]	*	*		
Java Neural Network Framework Neuroph[25]	*	*		
Joone[26]	*	*		
KERAS[27]				*

[11] http://sourceforge.net/projects/aisolver
[12] https://aws.amazon.com/de/machine-learning
[13] https://mahout.apache.org
[14] https://bigml.com
[15] http://caffe.berkeleyvision.org
[16] http://sourceforge.net/projects/currennt
[17] http://deeplearning4j.org
[18] http://sourceforge.net/projects/eblearn
[19] https://elki-project.github.io
[20] http://www.heatonresearch.com/encog
[21] http://sourceforge.net/projects/fann
[22] https://cloud.google.com/products/machine-learning
[23] https://www.ibm.com/cloud/machine-learning
[24] http://sourceforge.net/projects/jaden
[25] http://sourceforge.net/projects/neuroph
[26] http://sourceforge.net/projects/joone
[27] https://keras.io

Machine learning product table

Product	ML library	ML dev. env.	ML web service	Pre-trained model
KNIME[28]	*	*		
Microsoft Azure Machine Learning Studio[29]			*	
MLlib (Apache Spark)[30]	*			
MS Cognitive Toolkit[31]	*			
OpenNN - Open Neural Networks Library[32]	*			
Orange[33]	*	*		
procog[34]			*	
R [35]	*			
RapidMiner[36]	*	*		
scikit-learn[37]	*			
Shogun[38]	*			
SPSS Modeler[39]	*	*		
TensorFlow[40]	*			
Theano[41]	*			
Torch[42]	*			
WEKA[43]	*	*		

Knowledge Representation

Knowledge representation product table

Product	Integrated Environment	Knowledge Editor	API	Query Engine	Reasoner	Integration / Enrichment	Knowledge Base	Knowledge Resources
AllegroGraph[44]			*	*	*		*	
Apache Jena[45]			*	*	*		*	

[28] https://www.knime.com
[29] https://azure.microsoft.com/de-de/services/machine-learning-studio
[30] http://spark.apache.org/mllib
[31] https://docs.microsoft.com/en-us/cognitive-toolkit
[32] http://sourceforge.net/projects/opennn
[33] http://orange.biolab.si
[34] http://precog.com
[35] https://www.r-project.org
[36] https://rapidminer.com
[37] http://scikit-learn.org
[38] http://www.shogun-toolbox.org
[39] https://www.ibm.com/analytics/spss-statistics-software
[40] https://www.tensorflow.org
[41] http://deeplearning.net/software/theano
[42] http://torch.ch
[43] http://www.cs.waikato.ac.nz/ml/weka
[44] http://franz.com/agraph/allegrograph
[45] https://jena.apache.org

Knowledge representation product table

Product	Integrated Environment	Knowledge Editor	API	Query Engine	Reasoner	Integration / Enrichment	Knowledge Base	Knowledge Resources
Apache Jena Fuseki[46]				*				
Apache Stanbol[47]			*	*	*	*	*	
Cognitum FluentEditor[48]		*						
CYC[49]								*
DBpedia[50]								*
dotNetRDF[51]			*					
Ecliplse rdf4j[52]			*	*	*		*	
FaCT++[53]					*			
GND[54]								*
GraphDB[55]				*	*		*	
HermiT[56]					*			
i-Views Knowledge Graph Platform[57]	*	*	*	*	*		*	
Neo4J[58]							*	
OpenLink Virtuoso[59]			*	*	*		*	
PoolParty[60]	*	*	*	*	*	*	*	
Protégé[61]		*						
RacerPro[62]					*			
Semafora[63]	*	*	*	*	*		*	

[46] https://jena.apache.org/documentation/fuseki2
[47] https://stanbol.apache.org
[48] http://www.cognitum.eu/semantics/FluentEditor
[49] http://www.cyc.com
[50] https://wiki.dbpedia.org
[51] https://www.dotnetrdf.org
[52] https://rdf4j.org
[53] http://owl.man.ac.uk/factplusplus
[54] https://www.dnb.de/DE/Professionell/Standardisierung/GND/gnd_node.html
[55] http://www.ontotext.com/products/ontotext-graphdb
[56] http://hermit-reasoner.com
[57] https://i-views.com/de/knowledge-graph-plattform
[58] https://neo4j.com
[59] https://virtuoso.openlinksw.com
[60] https://www.poolparty.biz
[61] http://protege.stanford.edu
[62] http://franz.com/agraph/racer
[63] https://www.semafora-systems.com

Knowledge representation product table

Product	Integrated Environment	Knowledge Editor	API	Query Engine	Reasoner	Integration / Enrichment	Knowledge Base	Knowledge Resources
Topbraid Composer[64]		*						
Topbraid EDG[65]	*	*	*	*	*	*	*	
Wikidata[66]								*
YAGO[67]								*

AI Application Architecture

AI application architecture product table

Product	Presentation	Application Logic	API	Knowledge Base / Query Engine / Reasoner	Data Integration / Semantic Enrichment	3rd party AI web services
AllgegroGraph[68]			*	*		
Amazon AWS AI[69]						*
Apache Any23[70]					*	
Apache Jena[71]			*	*		
Apache Nutch[72]					*	
Apache Stanbol[73]			*	*	*	
C#[74]		*				
C++[75]		*				
Clojure / Lisp[76]		*				
Eclipse rdf4j[77]			*	*		

[64] http://www.topquadrant.com/tools/ide-topbraid-composer-maestro-edition
[65] http://www.topquadrant.com/product/TB_Suite.html
[66] https://www.wikidata.org
[67] https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago
[68] https://franz.com/agraph/allegrograph
[69] https://aws.amazon.com/ai
[70] https://any23.apache.org
[71] https://jena.apache.org
[72] http://nutch.apache.org
[73] https://stanbol.apache.org
[74]
[75]
[76]
[77] http://rdf4j.org

AI application architecture product table

Product	Presentation	Application Logic	API	Knowledge Base / Query Engine / Reasoner	Data Integration / Semantic Enrichment	3rd party AI web services
Google Cloud AI[78]						*
GraphDB[79]				*		
HTML/CSS[80]	*					
IBM Watson AI Services[81]						*
Java[82]		*				
JavaScript[83]	*					
Keras[84]		*				
Microsoft Azure AI[85]						*
Python[86]		*				
scikit-learn[87]		*				
Spark MLlib[88]		*				
TensorFlow[89]		*				
Virtuoso[90]			*	*	*	

Agent Frameworks

Agent frameworks product table

Product	Description
Product	Description
Agent Factory[91]	The Agent Factory Framework is an open source collection of tools, platforms, and languages that support the development and deployment of multi-agent systems.
Cougaar (Cognitive Agent Architecture)[92]	Cougaar is a java-based architecture for the construction of large-scale distributed agent-based applications

[78] https://cloud.google.com/products/ai
[79] https://www.ontotext.com/products/graphdb
[80]
[81] https://www.ibm.com/watson/products-services
[82]
[83]
[84] https://keras.io
[85] https://www.microsoft.com/AI
[86]
[87] https://scikit-learn.org
[88] https://spark.apache.org/mllib
[89] https://www.tensorflow.org
[90] http://virtuoso.openlinksw.com
[91] https://sourceforge.net/projects/agentfactory
[92] http://www.cougaar.world

Agent frameworks product table

Product	Description
JaCaMo[93]	JaCaMo is a framework for Multi-Agent Programming
JADE (Java Agent DEvelopment Framework)[94]	JADE (Java Agent DEvelopment Framework) is a software Framework fully implemented in the Java language. It simplifies the implementation of multi-agent systems through a middle-ware
Jadex BDI Agent System[95]	Jadex is a Belief Desire Intention (BDI) reasoning engine that allows for programming intelligent software agents in XML and Java
JIAC (Java-based Intelligent Agent Componentware)[96]	JIAC (Java-based Intelligent Agent Componentware) is a Java-based agent architecture and framework that eases the development and the operation of large-scale, distributed applications and services.
osBrain (General-Purpose Multi-Agent Python Module)[97]	osBrain is a general-purpose multi-agent system module written in Python and developed by OpenSistemas.

Information Retrieval

Information retrieval product table

Product	Search engine library	Indexer	Crawler	Search server platform	Search web service
Apache Lucene[98]		*			
Apache Nutch[99]			*		
Apache Solr[100]	*	*		*	
Elastic Search[101]	*	*		*	
Google Search[102]					*
Yahoo Search[103]					*
Yandex Search[104]					*

[93] http://jacamo.sourceforge.net
[94] http://jade.tilab.com
[95] http://sourceforge.net/projects/jadex
[96] http://www.jiac.de/agent-frameworks
[97] https://github.com/opensistemas-hub/osbrain
[98] http://lucene.apache.org/core
[99] http://nutch.apache.org
[100] http://lucene.apache.org/solr
[101] http://www.elasticsearch.org
[102] https://www.google.com
[103] https://yahoo.com
[104] https://www.yandex.com

Natural Language Processing

Natural language processing product table

Product	NLP building block	NLP framework	NLP web service	NLP resource
Amazon Alexa Voice service[105]			*	
Apache UIMA (Unstructured Information Management Architecture)[106]		*		
Bing Translate API[107]			*	
Botkit[108]		*		
ChatterBot[109]	*	*		
Chrome Web Speech API[110]		*		
Dandelion API[111]			*	
DBpediaSpotlight[112]			*	
DeepL[113]			*	
GATE (General Architecture for Text Engineering)[114]		*		
Germanet[115]				*
Google Cloud Speech API[116]			*	
Google Dialogflow[117]	*		*	
Google Translate API[118]			*	
IBM Watson NLP[119]			*	
IBM Watson Speech-to-Text[120]			*	
MS Azure Speech Services[121]			*	
MS Bot Framework[122]		*	*	

[105] https://developer.amazon.com/de/alexa-voice-service
[106] https://uima.apache.org
[107] http://msdn.microsoft.com/en-us/library/dd576287.aspx
[108] https://botkit.ai
[109] https://github.com/gunthercox/ChatterBot
[110] https://developers.google.com/web/updates/2013/01/Voice-Driven-Web-Apps-Introduction-to-the-Web-Speech-API?hl=en
[111] https://dandelion.eu
[112] https://www.dbpedia-spotlight.org
[113] https://www.deepl.com/translator
[114] https://gate.ac.uk
[115] http://www.sfs.uni-tuebingen.de/GermaNet
[116] https://cloud.google.com/speech
[117] https://dialogflow.com
[118] https://cloud.google.com/translate
[119] https://cloud.ibm.com/catalog/services/natural-language-understanding
[120] https://www.ibm.com/watson/services/speech-to-text
[121] https://azure.microsoft.com/de-de/services/cognitive-services/speech
[122] https://dev.botframework.com

Natural language processing product table

Product	NLP building block	NLP framework	NLP web service	NLP resource
Natural Language Toolkit[123]		*		
Ontotext[124]			*	
Pandorabots[125]	*	*		
RASA[126]	*	*		
scikit-learn[127]	*			
spaCy[128]	*	*	*	*
Spark MLlib[129]	*			
Stanford EnglishTokenizer[130]	*			
Stanford NER[131]	*			
Stanford Parser[132]	*			
Stanford POS Tagger[133]	*			
TensorFlow[134]	*			
TextRazor[135]	*		*	
Wikimeta API[136]			*	
Wordnet[137]				*
Yandex Translate API[138]			*	

Computer Vision

Computer vision product table

Product	CV/ML library	CV/ML development environment	CV web service	CV Pre-trained model
Amazon Rekognition[139]			*	
Autotag[140]			*	
BetaFace[141]			*	

[123] https://www.nltk.org
[124] https://www.ontotext.com
[125] https://home.pandorabots.com
[126] https://rasa.com
[127] https://scikit-learn.org
[128] https://spacy.io
[129] https://spark.apache.org/mllib
[130] http://nlp.stanford.edu/software/tokenizer.shtml
[131] http://nlp.stanford.edu/software/CRF-NER.shtml
[132] http://nlp.stanford.edu/software/lex-parser.shtml
[133] http://nlp.stanford.edu/software/tagger.shtml
[134] https://www.tensorflow.org
[135] https://www.textrazor.com
[136] https://www.programmableweb.com/api/wikimeta
[137] http://wordnet.princeton.edu
[138] https://api.yandex.com/translate
[139] https://aws.amazon.com/rekognition
[140] http://autokeyword.me/demo/?key=common
[141] http://www.betaface.com

Computer vision product table

Product	CV/ML library	CV/ML development environment	CV web service	CV Pre-trained model
BoofCV[142]	*			
Clarifai[143]			*	
DenseNet[144]				*
encog[145]	*			
Google Cloud Vision API[146]			*	
IBM Watson Visual Recognition[147]			*	
Inception[148]				*
Keras[149]	*			
MobileNet[150]				*
MS Azure Computer Vision[151]			*	
NASNet[152]				
Neuroph (Java Neural Network Framework)[153]	*	*		
OpenCV[154]	*			
Pan-o-Matic [155]	*			
RapidMiner[156]	*	*		
ResNet[157]				*
SURF[158]	*			
TensorFlow[159]	*			
Theano[160]	*			
tineye[161]			*	
Torch[162]	*			
VGG19[163]				

[142] http://boofcv.org
[143] http://www.clarifai.com
[144] https://keras.io/applications/#densenet
[145] http://www.heatonresearch.com/encog
[146] https://cloud.google.com/vision
[147] https://www.ibm.com/watson/services/visual-recognition
[148] https://keras.io/applications/#inceptionv3
[149] https://keras.io
[150] https://keras.io/applications/#mobilenetv2
[151] https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision
[152] https://keras.io/applications/#nasnet
[153] http://neuroph.sourceforge.net
[154] http://opencv.org
[155] http://aorlinsk2.free.fr/panomatic/?p=home
[156] https://rapidminer.com
[157] https://keras.io/applications/#resnet
[158] http://people.ee.ethz.ch/~surf
[159] https://www.tensorflow.org
[160] http://deeplearning.net/software/theano
[161] https://www.tineye.com
[162] http://torch.ch
[163] https://keras.io/applications/#vgg19

Computer vision product table

Product	CV/ML library	CV/ML development environment	CV web service	CV Pre-trained model
WEKA[164]	*	*		

Complex Event Processing

CEP product table

Product	Message broker	CEP engine
Amazon Kinesis[165]	*	
Apache ActiveMQ[166]	*	
Apache Flink[167]		*
Apache Kafka[168]	*	
Apache Qpid[169]	*	
Apache Storm[170]		*
Drools Fusion[171]		*
Eclipse Mosquitto MQTT Broker[172]	*	
Esper[173]		*
EVAM Streaming Analytics[174]		*
Fuse Message Broker[175]	*	
IBM MQ[176]	*	
Informatica RulePoint[177]		*
JBoss Messaging[178]	*	
Microsoft StreamInsight[179]		*
MS Azure Stream Analytics[180]		*
Open Message Queue[181]	*	
Oracle Stream Analytics[182]	*	*
RabbitMQ[183]	*	

[164] http://www.cs.waikato.ac.nz/ml/weka
[165] https://aws.amazon.com/kinesis
[166] http://activemq.apache.org
[167] https://flink.apache.org
[168] https://kafka.apache.org
[169] https://qpid.apache.org
[170] https://storm.apache.org
[171] https://www.drools.org
[172] https://mosquitto.org
[173] http://www.espertech.com/esper
[174] http://evam.com
[175] https://www.jboss.org/products/amq.html
[176] https://www.ibm.com/products/mq
[177] https://www.informatica.com/products/data-integration/real-time-integration/rulepoint-complex-event-processing.html
[178] http://labs.jboss.org/jbossmessaging
[179] https://msdn.microsoft.com/en-us/library/ee391416(v=sql.111).aspx
[180] https://docs.microsoft.com/de-de/azure/stream-analytics/stream-analytics-introduction
[181] https://javaee.github.io/openmq
[182] http://www.oracle.com/technetwork/middleware/complex-event-processing/overview/index.html
[183] https://www.rabbitmq.com

CEP product table

Product	Message broker	CEP engine
Redis[184]	*	
SAG Apama[185]		*
SAP ESP[186]		*
SAS ESP[187]		*
Siddhi[188]		*
TIBCO[189]	*	*
VIATRA-CEP[190]		*
WebSphere Business Events[191]	*	*
WSO2 Stream Processor[192]		*

[184] https://redis.io
[185] http://apamacommunity.com
[186] https://www.sap.com/products/complex-event-processing.html
[187] https://www.sas.com/en_us/software/event-stream-processing.html
[188] http://siddhi.sourceforge.net
[189] https://www.tibco.com/products/event-driven-applications
[190] https://wiki.eclipse.org/VIATRA/CEP
[191] https://www-01.ibm.com/software/integration/wbe
[192] https://docs.wso2.com/display/SP400/Stream+Processor+Documentation

About the Author

Bernhard G. Humm is a professor for software engineering and project management at the Computer Science Department of Hochschule Darmstadt - University of Applied Sciences, Germany. He received a Ph.D. from the University of Wollongong, Australia and the degree Dipl.-Inform. from Kaiserslautern University, Germany. His research focus is on applied AI and software architecture. He is the managing director of the institute of applied informatics, Darmstadt (aIDa) and Ph.D. coordinator. He is running several national and international research projects in co-operation with industry and research organizations and is publishing his results regularly. Before re-entering university as a professor, he worked in the IT industry for 11 years as a software architect, chief consultant, IT manager and head of the research department of a large German software company. His clients were in the industry sectors finance, tourism, trade, and aviation.

Contact:
Prof. Bernhard G. Humm, Dipl.-Inform., Ph.D. Computer Science (AUS)
Hochschule Darmstadt - University of Applied Sciences, Computer Science Department
Haardring 100, 64295 Darmstadt, Germany
bernhard.humm@h-da.de, www.fbi.h-da.de/~b.humm