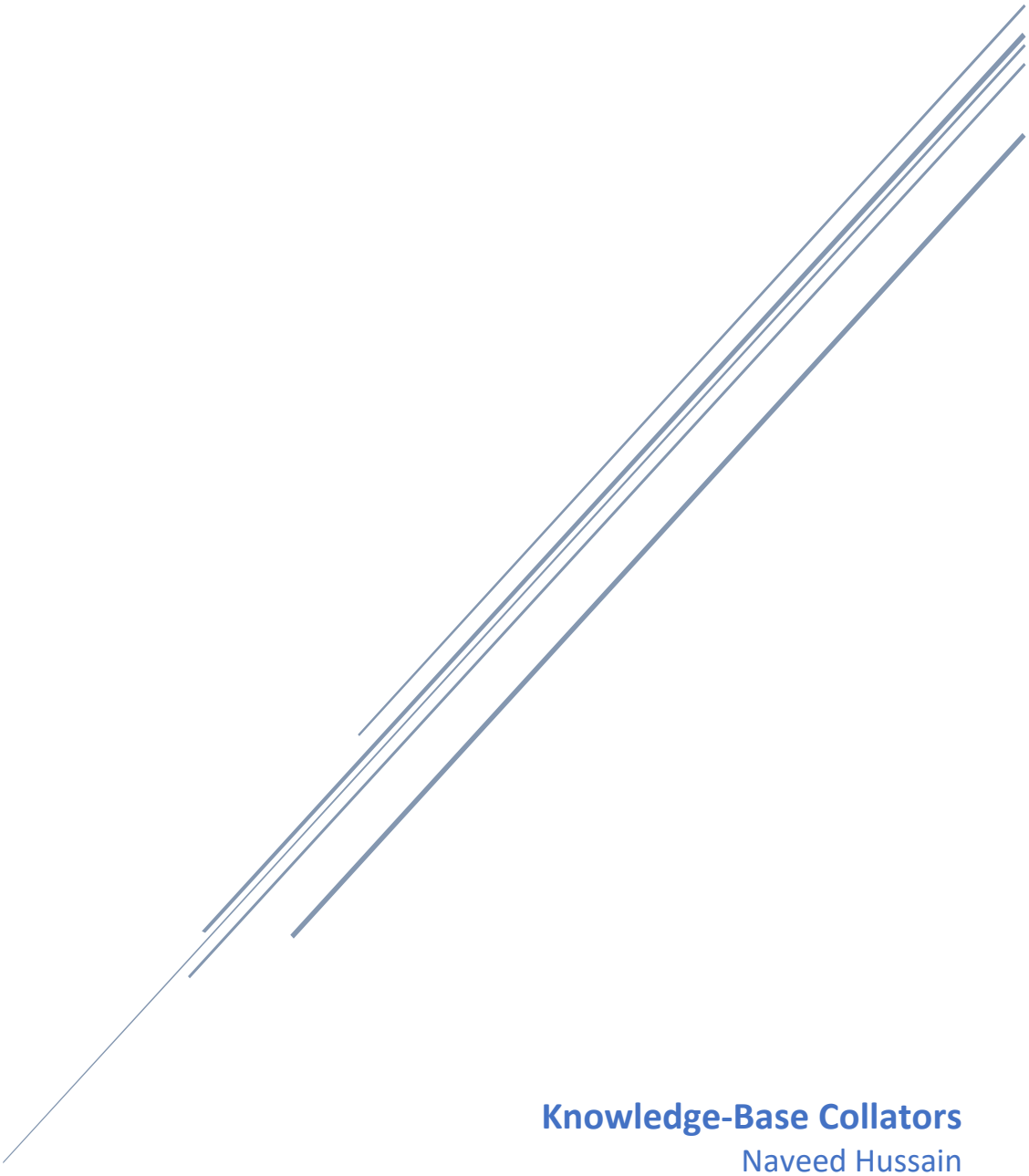
DATA SCIENCE & DEEP LEARNING ABC…

"There's a joke that says a data scientist is someone who knows more statistics than a computer scientist and more computer science than a statistician (I didn't say it was a good joke)"



Knowledge-Base Collators
Naveed Hussain
Faris Haddad
Robin Lester
Cloud Solution Architects for Data & AI at Microsoft UK

Table of Contents

Contents

Modern AI Quotes

The Core Currency of any business going forward will be the ability to convert their ***Data into AI*** that drives competitive advantage.

Every Developer can be an ***AI Developer***, and ***Every Company*** can become an ***AI Company***.

I believe over the next decade computing will become even more ubiquitous and ***intelligence will become ambient***. This will be made possible by an ever-growing network of connected devices, incredible computing capacity from the cloud, insights from big data, and intelligence from machine learning.

Satya Nadella

Microsoft CEO

Introduction

Artificial intelligence and machine learning, are at the top of the list of new technologies that enterprises want to embrace for all kinds of reasons. But it all reduces to the same problem: Sorting through the increasing amounts of data coming into their environments and finding patterns that will help them to run their businesses more efficiently, to make better businesses decisions, and ultimately to make more money.

Artificial Intelligence Needs Big Data, and Big Data Needs AI

The cloud has created a democratised platform where everyone has access to the same compute, storage, and analytics. The real differentiator for enterprises will be the data they generate, and more importantly, the value the enterprises derive from that data. Given that, it will be data that businesses compete on, and the challenge there will be who will have the best data, be able to most quickly derive the best insight and make the best business decisions based on those insights.

Artificial intelligence and big data have formed a truly symbiotic relationship, and they need each other to bring to fruition what both are promising.

To further illustrate that artificial intelligence and big data are intertwined, consider these recent quotes from two highly regarded thought leaders in this space:

“Throughout the business world, every company these days is basically in the data business, and they’re going to need AI to civilize and digest big data and make sense out of it.” *(Kevin Kelly, co-founder of Wired)*

“In the past, AI’s growth was stunted due to limited data sets, representative samples of data rather than real-time, real-life data and the inability to analyse massive amounts of data in seconds. Today, there’s real-time, always-available access to the data and tools that enable rapid analysis. This has propelled AI and machine learning and allowed the transition to a data-first approach. Our technology is now agile enough to access these colossal datasets to rapidly evolve AI and machine-learning applications.” *(Bernard Marr, noted AI author and speaker).*

For now, 99% of the economic value created by AI comes from supervised learning systems, according to Ng, an AI thought leader and an adjunct professor of computer science at Stanford University. These algorithms require human teachers and tremendous amounts of data to learn. It's laborious, but a proven process.

AI algorithms, for example, can now recognize images of cats, although they required thousands of labelled images of cats to do so; and they can understand what someone is saying, although leading speech recognition systems needed 50,000 hours of speech -- and their transcripts -- to do so.

Data is the competitive differentiator for what AI can do today -- not algorithms, which, once trained, can be copied.

"There's so much open source, word gets out quickly, and it's not that hard for most organizations to figure out what algorithms organizations are using," said Ng.

Platforms

Enterprises like Google, Facebook, LinkedIn and Microsoft have for several years embraced data-driven AI and machine learning techniques and built their own internal frameworks and platforms that enable them to quickly take advantage of them. But as the technologies began to cascade into more mainstream enterprises, the complexity of software and systems were throwing obstacles in front of initiatives aimed at leveraging AI and machine learning for the good of the business.

There are myriad frameworks on the market that enterprises can take advantage of, from Azure ML, TensorFlow, Shogun and Theano libraries and Torch and Caffe frameworks to the Apache Singa and Vele platforms. The problem is that many enterprises don't have the time or resources to pull all that together themselves to create enterprise-grade, easy-to-use systems.

This structural mismatch of capabilities deployment means that data scientists at many of these enterprises are spending so much time pulling together the systems themselves – configuring and managing the databases and data management systems – that they're aren't doing what their jobs demand, which is coding and building algorithms that will enable their businesses to take advantage of AI and machine learning. You need effective toolsets to try to bring valued-added data and leverage ML to operate more efficiently as a business and you don't have the effective toolset on the operational side to support this

What is needed are commercial platforms that automate and operationalize the processes around AI that take much of the grunt work out of building the systems out of their hands; where they can put this in a mission-critical way, and not forcing mathematicians to do process supervision or to deploy microservices.

Microsoft provides the platforms that solved these issues, where a lot of the repeated patterns of building out an ML pipeline are systematized and done in a way that you could leverage system engineers for the operational infrastructure and let data scientists focus on the data science. What is meant by operational is that within a unified platform, you're able to do data cleansing, intergeneration model training, model evaluation and model deployment and audit?

This is the most efficient use of the data science team's time and the best use of the entire enterprise's time because the things that are being deployed and being used are being managed in an operational context.

As enterprises get more comfortable with the idea of leveraging AI and machine learning, demand for automated platforms also grows, fuelled in part by the widespread development and availability of open-source tools. This, in turn, is also cascading down to mid-tier and smaller enterprises.

Now ML platforms and the open source frameworks like Anaconda's Python distribution are all out there and are rapidly improving with more and more programmers out there who know how to leverage them. This is bringing down the bar on the math side, where you don't really have to know about the innards of the machinery, you can just understand how it fits into a broader ensemble that you're trying to build. As that barrier to entry from the mathematical and the library perspective is dramatically reduced, it's opening it up to thinking about ML and how they can use it from a development standpoint. Ten years ago, if you were building an ML pipeline you'd have to get deep into math and understand what's going on and sometimes build out you're own computational or algorithmic primitive. Now that's taken care of.

Artificial intelligence is no longer the exclusive domain of PhDs. Now, thanks to a new generation of easier-to-use tools and platforms, tech professionals can start building and deploying AI solutions within their projects. Today the idle ground where big data analytics is finally within reach of the average engineer or programming techie, there is now a large middle ground where smart non-data scientists can be very productive with applied machine learning even on big and real-time data streams. To achieve the big data and AI goals, you still need to understand extract, transform and load concepts and what machine learning is and can do, but you certainly don't need to program low-level parallel linear algebra in MapReduce anymore.

Chapter 01 – General Theory of Data Science

What is Analytics?

Traditional Analytics

Business Information resulting from the systematic analysis of data or statistics

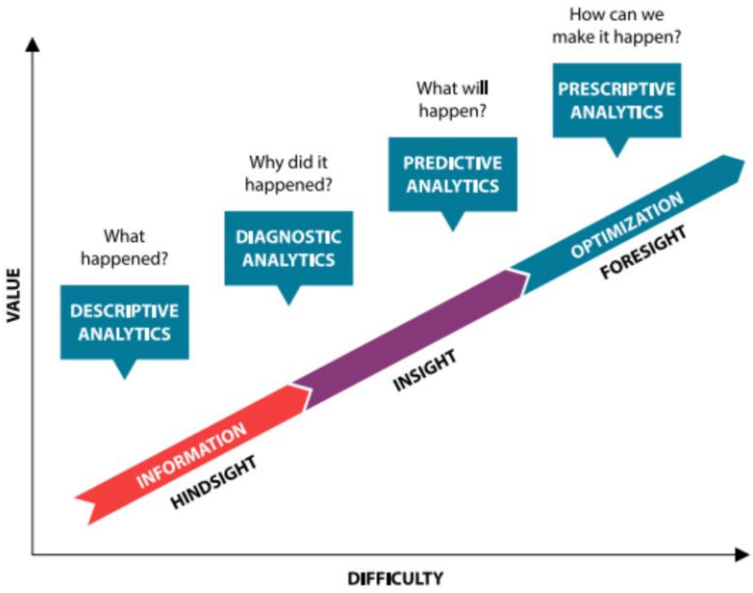
Traditional Analytics is the discovery, interpretation, and communication of meaningful patterns in data. Especially valuable in areas rich with recorded information, analytics relies on the simultaneous application of statistics, computer programming and operations research to quantify performance.

Advanced Analytics

A broad category of inquiry that can be used to help drive changes and improvements in business practices... Predictive analytics, data mining, big data analytics, and location intelligence are just some of the analytical categories that fall under the heading of advanced analytics

Advanced Analytics is the autonomous or semi-autonomous examination of data or content using sophisticated techniques and tools, typically beyond those of traditional business intelligence (BI), to discover deeper insights, make predictions, or generate recommendations.

Analytics Type Comparison



Descriptive Analytics

Business Intelligence and Data Mining

The simplest class of analytics which allows you to condense data into more useful nuggets of information. What is happening now based on incoming data. To mine the analytics, you typically use a real-time dashboard and/or email reports.

Diagnostic Analytics

Root-Cause Analysis

A look at past performance to determine what happened and why. The result of the analysis is often an analytic dashboard.

Predictive Analytics

Forecasting

It can only what might happen in future because it is probabilistic in nature. An analysis of likely scenarios of what might happen. The deliverables are usually a predictive forecast.

Prescriptive Analytics

Simulation & Optimisation

It helps you achieve the best outcomes and helps understanding how by adding spice to manipulate the future. This type of analysis reveals what actions should be taken. This is the most valuable kind of analysis and usually results in rules and recommendations for next steps.

Data Science (DS)

Data science is the discipline of drawing conclusions from data using computation. There are three core aspects of effective data analysis: exploration, prediction, and inference

Data science is an ***interdisciplinary field*** about processes and systems to extract knowledge or ***insights from data*** in various forms, either ***structured, semi-structured, unstructured or multi-structured***, which is a continuation of some of the data analysis fields such as ***Mathematics, Statistics, Artificial Intelligence, Machine Learning, Deep Learning, Data Mining & Predictive Analytics***, like **Knowledge Discovery in Databases (KDD)**.

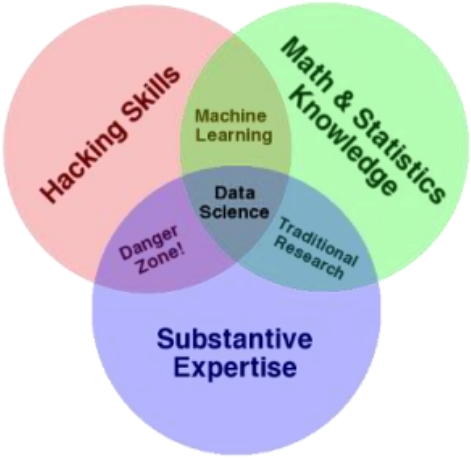
What is Data Science?

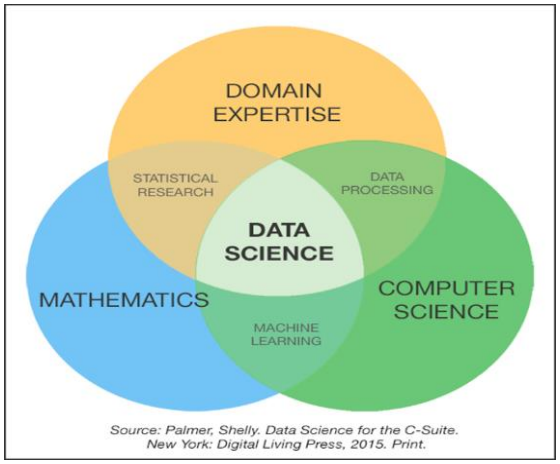
As Data Science is an interdisciplinary field of processes and systems to extract knowledge or insights from data in various forms, either structured or unstructured. Data science is concerned with extracting knowledge and insights from a wide variety of data sources to analyse patterns or predict future behaviour. It draws from a wide array of disciplines including statistics, computer science, mathematics, machine learning, and data mining.

Data science is not a single science as much as it is a collection of various scientific disciplines integrated for analysing data. These disciplines include various statistical and mathematical techniques, including:

- Computer science
- Data engineering
- Visualization
- Domain-specific knowledge and approaches

There are tons of blogs, articles, diagrams, and other information channels that aim to define this new and still-fuzzy term 'Data Science,' and it will still be some years before we achieve consensus. At least, for now, there is some agreement surrounding the main ingredients; Drew Conway summarizes them nicely in this Venn diagram:





- **Statistics** is perhaps the most obvious component, as Data Science is partially about analysing data using summary statistics (e.g., averages, standard deviations, correlations, etc.) and more complex mathematical tools. This is supplemented by
- **Machine learning** which is defined as a "Field of study that gives computers the ability to learn without being explicitly programmed". Machine learning explores the study and construction of algorithms that can learn from and make predictions on data. Such algorithms operate by building a model from example inputs to make data-driven predictions or decisions. Typically, the output of Machine Learning is a certain number of features that are important to a given business problem. It can provide insight when evaluated in the context of
- **Domain Knowledge** and therefore essential to identify and explore the questions that will drive business actions. It is the one ingredient that's not generalizable across different segments of the industry (disciplines or domains) and as such, a Data Scientist must acquire new Domain Knowledge for each new problem that she/he encounters.

Tasks of Data Science

Basic data science tasks are:

- Data collection
- Data Cleaning
- Data Analysis
    - Statistical Techniques
    - Machine Learning
    - Neural Networks
    - and Deep Learning
- and Data Visualization

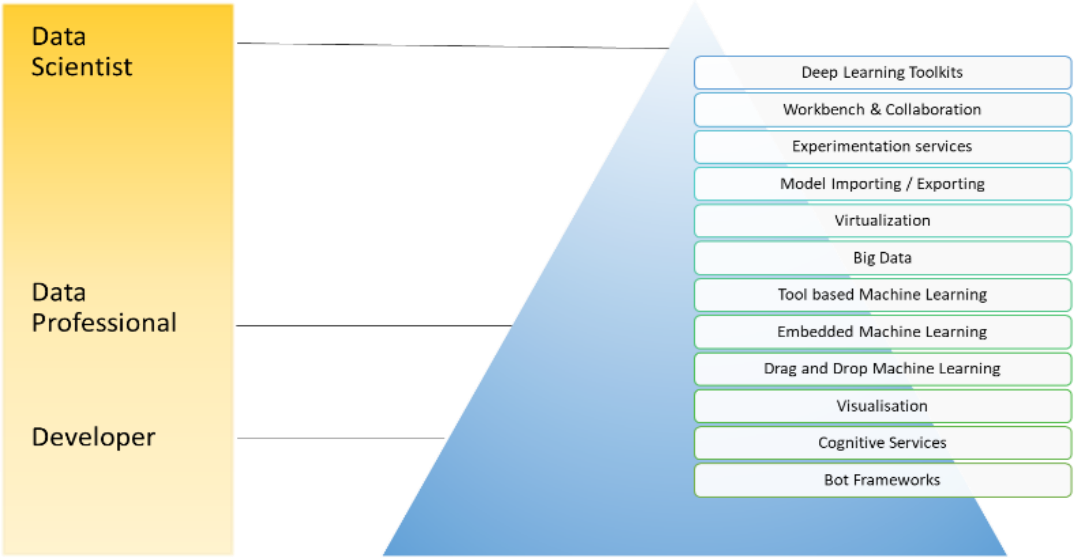
Data Scientist

An individual employed to ***Analyse & Interpret Complex Digital Data***, such as the usage statistics of a website, especially to ***Assist a Business*** in its ***Decision-Making***.

With the advent of cheaper storage technology, more and more data has been collected and stored permitting previously unfeasible processing and analysis of data. With this analysis came the need for various techniques to make sense of the data. These large sets of data, when used to analyse data and identify trends and patterns, become known as **big data**.

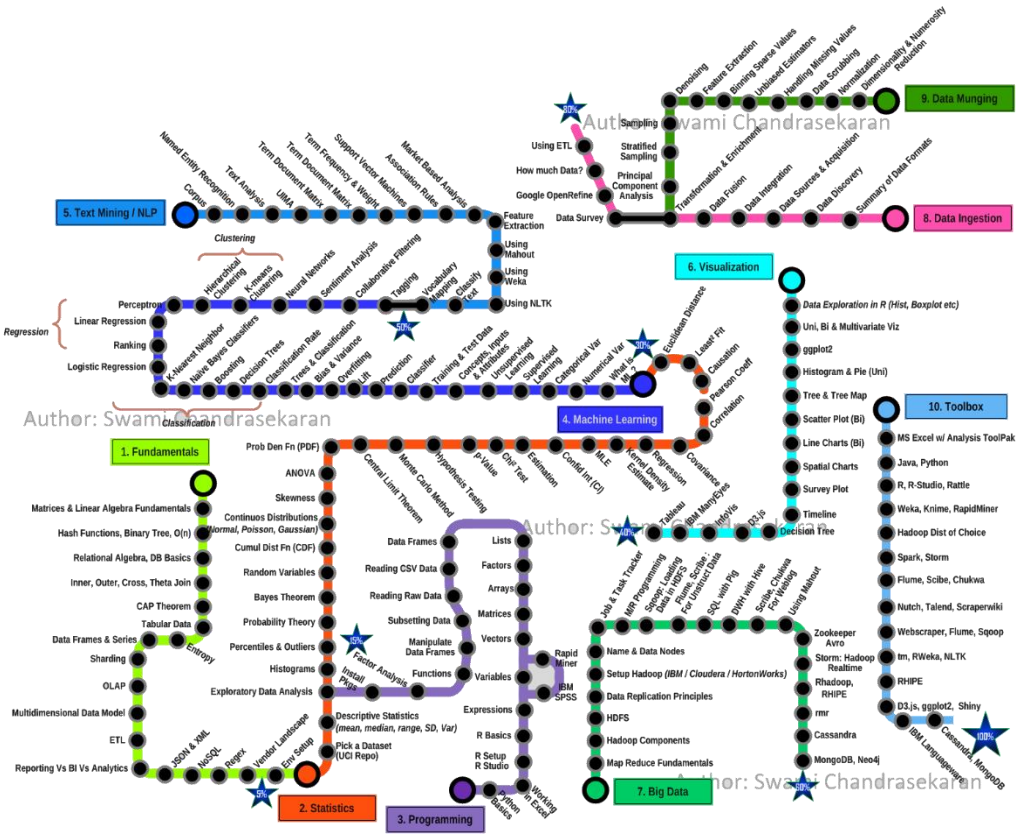
This, in turn, gave rise to cloud computing and concurrent techniques such as **map-reduce,** which distributed the analysis process across many processors, taking advantage of the power of parallel processing.

The process of analysing big data is not simple and evolves to the specialization of developers who were known as **data scientists**. Drawing upon a myriad of technologies and expertise, they can analyse data to solve problems that previously were either not envisioned or were too difficult to solve.



"Silicon Valley technology companies are hiring data scientists to help them glean insights from the terabytes of data that they collect every day".

Data scientists use their data and analytical ability to find and interpret rich data sources; manage large amounts of data despite hardware, software, and bandwidth constraints; merge data sources; ensure consistency of datasets; create visualizations to aid in understanding data; build mathematical models using the data; and present and communicate the data insights/findings.



The route to becoming a data scientist is not an easy one as Swami Chandrasekaran illustrates in Becoming a Data Scientist – Curriculum via Metro map.

Why would you hire a data scientist?

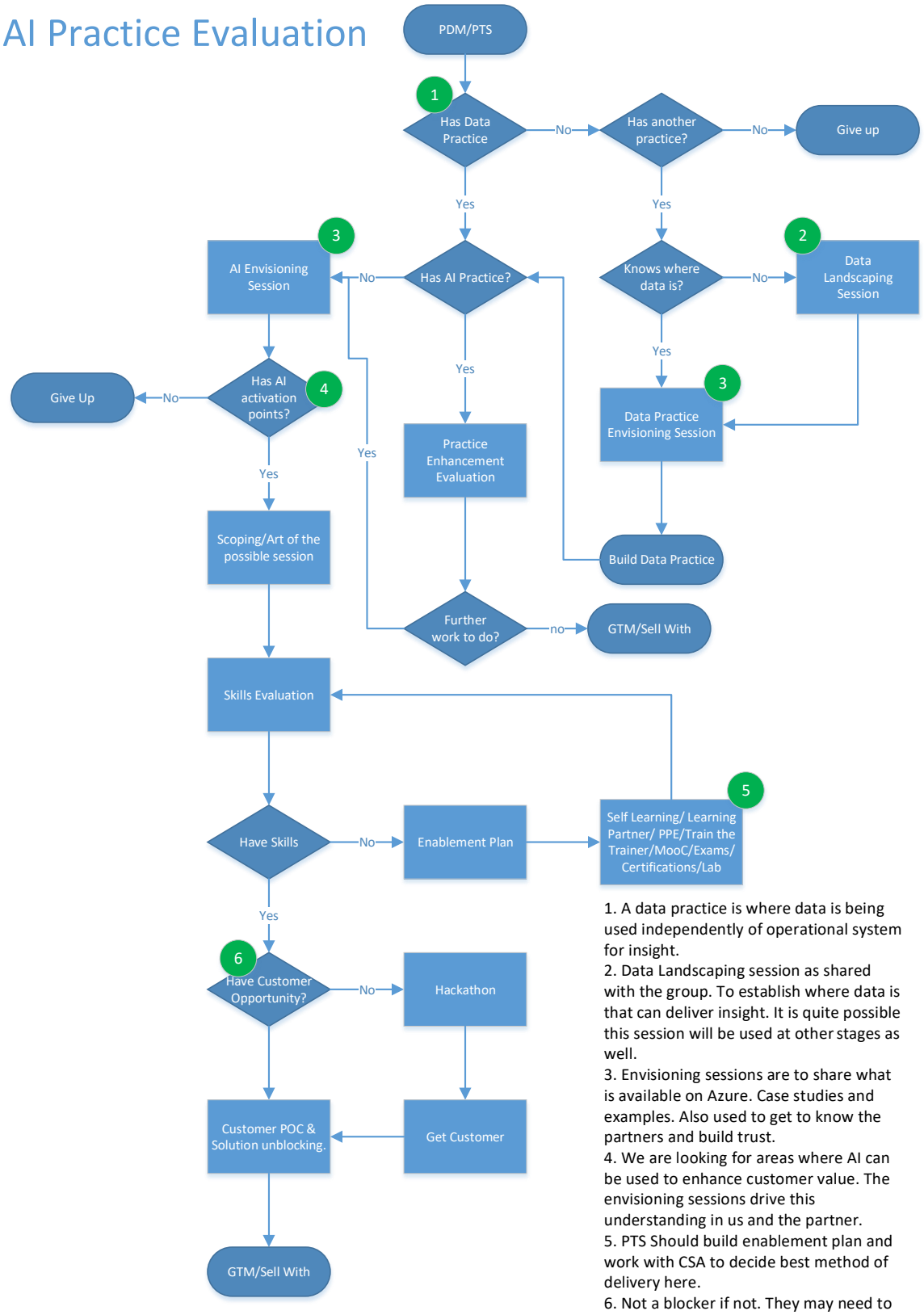
So, all this talk about data science is great and all, but why should you hire one, Because, A data scientist can help you turn raw data into information.

The Data Science Process

So now that you have a good understanding of what a data scientist can do for you. What are the steps in a data science project?

- Ask an interesting question
- Get the data
- Explore the data
- Model the data
- Communicate and visualize the results

AI Practice Evaluation Criteria



1. A data practice is where data is being used independently of operational system for insight.
2. Data Landscaping session as shared with the group. To establish where data is that can deliver insight. It is quite possible this session will be used at other stages as well.
3. Envisioning sessions are to share what is available on Azure. Case studies and examples. Also used to get to know the partners and build trust.
4. We are looking for areas where AI can be used to enhance customer value. The envisioning sessions drive this understanding in us and the partner.
5. PTS Should build enablement plan and work with CSA to decide best method of delivery here.
6. Not a blocker if not. They may need to have something to 'show' before they can sell.

Problems solved using Data Science

The various data science techniques that we will illustrate have been used to solve a variety of problems. Many of these techniques are motivated to achieve some economic gain, but they have also been used to solve many pressing social and environmental problems.

Problem domains where these techniques have been used include finance, optimizing business processes, understanding customer needs, performing DNA analysis, foiling terrorist plots, and finding relationships between transactions to detect fraud, among many other data-intensive problems.

Data Science Approach

Data science is concerned with the processing and analysis of large quantities of data to create models that can be used to make predictions or otherwise support a specific goal. This process often involves the building and training of models. The specific approach to solve a problem is dependent on the nature of the problem. However, in general, the following are the high-level tasks that are used in the analysis process:

- **Acquiring the data** before we can process the data, it must be acquired. The data is frequently stored in a variety of formats and will come from a wide range of data sources.
- **Cleaning the data** Once the data has been acquired, it often needs to be converted to a different format before it can be used. In addition, the data needs to be processed or cleaned, to remove errors, resolve inconsistencies, and otherwise put it in a form ready for analysis.
- **Analysing the data** This can be performed using many techniques including:
    - **Statistical analysis** This uses a multitude of statistical approaches to provide insight into data. It includes simple techniques and more advanced techniques such as regression analysis.
    - **AI analysis** These can be grouped as machine learning, neural networks, and deep learning techniques:
        - Machine learning approaches are characterized by programs that can learn without being specifically programmed to complete a specific task Neural networks are built around models patterned after the neural connection of the brain Deep learning attempts to identify higher levels of abstraction within a set of data.
- **Text analysis** This is a common form of analysis, which works with natural languages to identify features such as the names of people and places, the relationship between parts of the text, and the implied meaning of the text.
- **Data visualization** This is an important analysis tool. By displaying the data in a visual form, a hard-to-understand set of numbers can be more readily understood.
- **Video, image, and audio processing and analysis** This is a more specialized form of analysis, which is becoming more common as better analysis techniques are discovered and faster processors become available. This contrasts with the more common text processing and analysis tasks. Complementing this set of tasks is the need to develop applications that are efficient. The introduction of machines with multiple processors and GPUs contributes significantly to the result. While the exact steps used will vary by application, understanding these basic steps provides the basis for constructing solutions to many data science problems.

Chapter 02 – Theory of Data Science Process

Introduction

In Data Science, it's often more fun and exciting to focus on the technologies, the algorithms, and visualizations in the project. But you should start with focusing on the process you'll follow. The platform should always follow Process.

The initial thought is that a Data Science project is like any other technology project. But Data Science, unlike other IT efforts, has specific elements that are exploratory and experiment-based, which many organizations are unfamiliar with.

Enterprise data science teams are generally quite diverse, comprising of individuals with varied backgrounds and training, and often situated across geographical boundaries. Standardizing on data science projects and project artefacts can, therefore, be a particularly important tool in improving collaboration, consistency and efficiency across such teams.

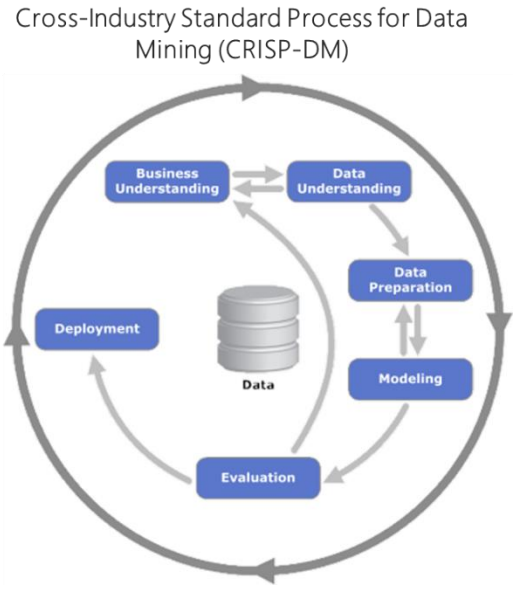
So how do you explain the project, implement it, and keep it on track? A process is needed. A process specifies a detailed sequence of activities necessary to perform specific business tasks. It is used to standardize procedures and establish best practices. Processes give you a place to start, a roadmap, and a way to explain to your stakeholders what you're going to do and the order you'll do it. In addition, a process compresses information into shorter information so that you can keep tabs as you work through it. Then you can decompress that information for each step, assign it to the right people and teams, and parallelize work where possible. So, it's important to think through a process, create and edit it, test it, and adjust based on reality. It doesn't mean you *have* to follow it – but it gives you a defined way to start.

A recent Forrester study to explore the use of big data analytics and data science platforms in greater depth. Forrester sought to quantify the impact that data science platforms have on the organizations that use them, and whether the use of more advanced and centralized platforms translates into better business results.

Once you have these many data scientists to manage you rapidly become concerned about efficiency and effectiveness. That's a huge investment in high priced talent that needs to show a good ROI. Also, in this environment, it's likely that you have from several hundred to thousands of models that direct core business functions to develop and maintain.

It's easy to see that if everyone is freelancing in R (or Python) that managing for consistency of approach and quality of outcome, not to mention the ability for collaboration around a single project is almost impossible. This is what's driving the largest companies onto common platforms with drag-and-drop consistency and efficiency. Much of the work that data scientists do will revolve around centralized platforms that help to organize not just the data and the tools, but data scientists themselves.

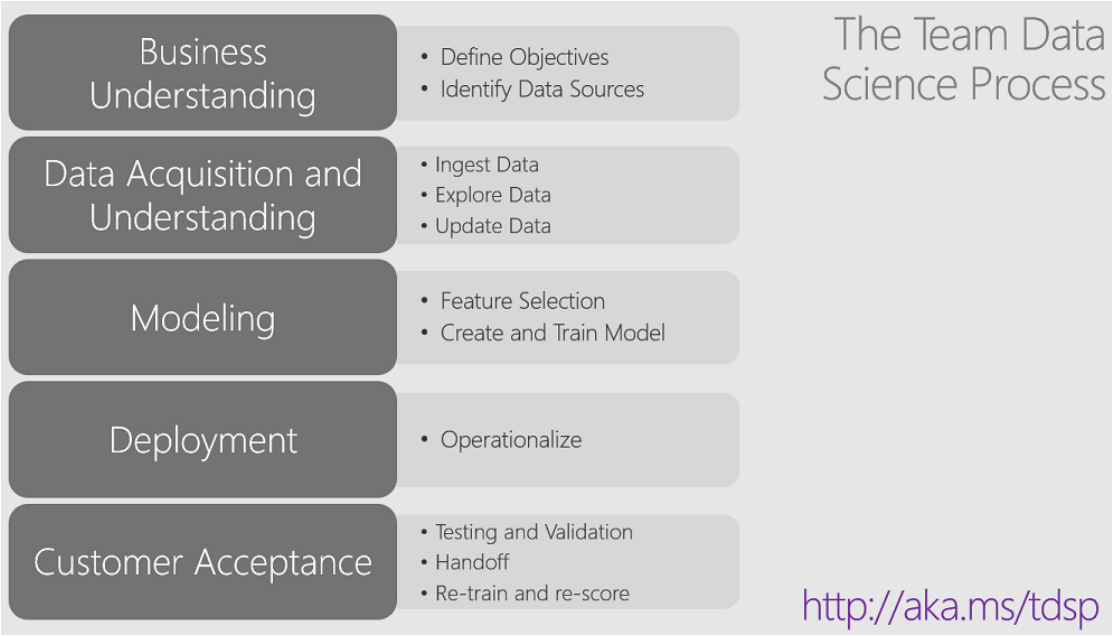
For years, the primary process a Data Scientist would follow was CRISP-DM. It's a great process involving many phases you'll recognize from Business Intelligence frameworks. The methodology itself was conceived in 1996.



"CRISP-DM remains the most popular methodology for analytics, data mining, and data science projects, with 43% share in latest KDnuggets Poll, but a replacement for unmaintained CRISP-DM is long overdue." *Industry veteran Gregory Piatetsky of KDNuggets*

But there are a couple of issues with it: It is quite general - covers all aspects of a client project, from business understanding to final deployment of a solution and highlights the iterative nature of data science project phases, but it is just a high-level description of the phases. Does not help you execute a team. Hints at but does not prescribe output or organization. It also assumes that every project will have a Machine Learning or at least predictive component – not always necessary in Advanced Analytics. It is no longer being maintained and the framework itself has not been updated to address issues of working with new technologies, such as Big Data and the team nature of things. CRISP-DM also neglects aspects of decision making.

This led Microsoft to invent the Team Data Science Process (TDSP), a process to make enterprise DS teams more efficient. It handles the same kind of work as the CRISP-DM but adds in other phases and fleshes out the team aspect of the process.



It's an **open source** agile, iterative, data science methodology to improve collaboration and team learning. The launch of the methodology is accompanied by a set of utilities meant to help companies better organize its data.

Its aimed at including Big Data as a data source. As previously stated, the Data Understanding can be more complex. But in an Advance Analytics project, there are lots of things that can be done by a team, not all of whom are 6-year PhD's in Machine Learning – such as Data Wrangling, visualizations, and other steps.

TDSP is an agile, iterative, data science process for executing and delivering machine learning and advanced analytics solutions. It is designed to improve collaboration and efficiency in enterprise data science teams. TDSP has four components:

- A standard data science lifecycle definition.
- A standardized project structure, including project documentation and reporting templates. A standard project structure, including a well-defined directory hierarchy and a list of output artefacts in a standard document template structure that is stored in a versioned repository.
- A shared and distributed analytics infrastructure for project execution, e.g. compute and storage infrastructure, code repositories, etc.
- Tools for data science project tasks, e.g. collaborative version control and code review, data exploration and modelling, work planning, etc. These simplify adherence to the process by automatically producing project artefacts and providing scripts for common tasks such as the creation and management of repositories and shared analytics resources.

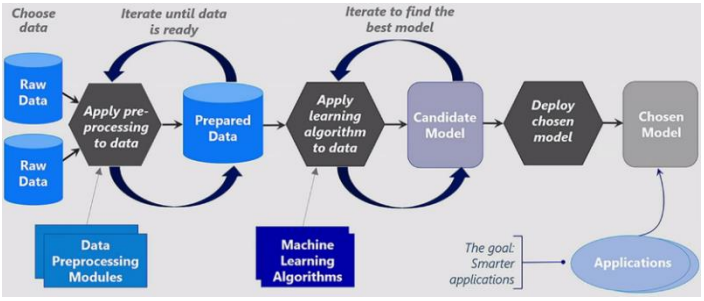
We have a 2-day workshop with hands-on activities that develop proficiency in AI-oriented workflows leveraging Azure Machine Learning Workbench and Services, the Team Data Science Process, Visual Studio Team Services, and Azure Container Services. These labs assume an introductory to intermediate knowledge of these services, and if this is not the case, then you should spend the time working through the pre-requisites.

https://azure.github.io/LearnAI-Bootcamp/proaidev_bootcamp

TDSP resources on Azure

We provide documentation and end-to-end data science process walkthroughs and templates using different platforms and tools on Azure, such as Azure ML, HDInsight, Microsoft R server, SQL-server, Azure Data Lake etc.

Here are instructions on how to execute data science life cycle steps in Azure ML.



Understanding the Process of Collecting, Cleaning, Analysis, Modelling and Visualizing Data

As data scientists work on huge sets of apparently disparate information to unveil surprising insights in fields as varied as accounting and law enforcement, the process they follow is a mystery to most outside the field.

Your car insurance costs less if you pay your bill on time. That's because insurance industry data scientists found that people that pay bills promptly are less likely to be in accidents. How did they even think to ask that question? How did they accumulate the accident data and compare it to the billing information to establish the correlation? What other revelations are buried in those numbers?

Still, it's not the mysteries they unveil so much as that process itself that defines the field of data science.

In the past business and government turned to statisticians for answers when big numbers were involved. But large and complex datasets, representational reporting challenges, and data-driven inquiry all wrought changes that made "statistics" an outmoded description of what researchers were doing.

In 1997, University of Michigan statistics professor C.F. Jeff Wu went through the trouble of setting down what distinguished the modern practices that were evolving from traditional aspects of statistics. In a talk, he titled "Statistics = Data Science?" He both gave data science its name and outlined the basic process that describes the field today.

He identified three aspects of data science which differentiate it from pure statistics:

- Data Collection
- Data Modelling and Analysis
- Problem Solving and Decision Support

But while those three steps provide a high-level overview of what data scientists do daily, there's still a lot of mystery when it comes to the details of the process.

The data science process is a recursive one; arriving at the end will take a good data scientist back to the beginning again to refine each of the steps based on the information they uncovered.

But each round begins with a question.

Step 1. Ask an Interesting Question

Whether it originates in the mind of the investigating data scientist or as a request from other parties, every inquiry starts as a question to be answered.

> ***Is there a business goal to achieve?***
>
> ***Some object of scientific interest that would be helpful to discover.***
>
> ***What parameters would the ideal answer fulfil?***

Step 2. Design a Data Collection Program

In many cases, data scientists work with existing data sets collected during other investigations. But the way that data is gathered and stored can limit the questions that may be answered from it and relevant data is not always immediately available.

With the question in mind, the data scientist will decide *how* to gather the information required to answer it:

- Establish whether the data exists in the real world and is relevant to the question
- Devise a collection scheme to acquire it
- Logistical considerations
- Cost?
- Privacy issues.
- Coordinate with departments or agencies needed for collection program liaison

Step 3. Collect and Review the Data

Even the best-designed data collection system will result in some quirks and oddities in the data as it becomes available– typos, falsification, or frequently misunderstood questions on badly designed forms can all present data sets that are less than factual.

As the data is collected, the data scientist will review it to revisit the collection program and get a feel for the set:

- Store the incoming data in a way that will allow further modelling and reporting
- Join data from multiple sources in a relevant and logical manner
- Check for anomalies or unusual patterns caused by the collection process itself, or do they reflect the topic of investigation? Possible to correct, or do they require a new collection scheme?

Step 4. Process the Data

Either due to anomalies found in step 3 or just the general and common necessity of cleaning up messy raw data, the data scientist will have to “wrangle” it before moving further into the modelling process.

Also known as “munging” this hard-to-define step is one of the ways that data scientists make the magic happen—bringing skills and intuition to bear to take messy, incoherent information and shuffle it into clean, accessible sets.

Decide on the tools to use to comb through the raw data

Tools: ML workbench, R, Python, SQL

Devise scripts to correct issues or reformat the data

Store the munged data as a fresh data set or use programmatic pre-processing for each subsequent query

Step 5. Model and Analyse the Data Sets

With all the important groundwork complete, the data scientist will get down to the fun stuff— diving into a clean data set and applying the pick-and-shovel algorithms that will pluck meaning from it:

- Build a data model to fit the question
- Validate the model against the actual collected data
- Perform the necessary statistical analyses
- Machine-learning or recursive analysis
- Regression testing and other classical statistical analysis techniques
- Compare results against other techniques or sources

Step 6. Visualize and Communicate the Results

The most challenging part of the data scientist’s job is taking the results of the investigation and presenting them to the public or internal consumers of information in a way that makes sense and can be easily communicated:

Graph or chart the information for presentation

- Interactive, allowing users to explore directly?
- Tools: R, Python, Tableau, Excel, Power BI

Tell a story to fit the results

Interpret the data to describe the real-world sources in a plausible manner

Assist decision-makers in using the results to drive their decisions

- Answer follow-up questions
- Present the same data in different formats for specific purposes: reports, websites, compliance purposes

The process is rarely linear. Each step can push a data scientist back to previous steps before reaching the end of the process, forcing them to revisit their methods, techniques, or even to reconsider whether the original question was the right one in the first place.

And, having finally come to a definitive result, the data scientist will almost always find that the answer simply sparks more questions: the process begins again!

Example case study:

In this section, this process demonstrated through an example case study:

When a non-technical supervisor asks you to solve a data problem, the description of your task can be quite ambiguous at first. It is up to you, as the data scientist, to translate the task into a concrete problem, figure out how to solve it and present the solution back to all your stakeholders. We call the steps involved in this workflow the "Data Science Process." This process involves several important steps:

Frame the problem: Who is your client? What exactly is the client asking you to solve? How can you translate their ambiguous request into a concrete, well-defined problem?

Collect the raw data needed to solve the problem: Is this data already available? If so, what parts of the data are useful? If not, what more data do you need? What kind of resources (time, money, infrastructure) would it take to collect this data in a usable form?

Process the data (data wrangling): Real, raw data is rarely usable out of the box. There are errors in data collection, corrupt records, missing values and many other challenges you will have to manage. You will first need to clean the data to convert it to a form that you can further analyse.

Explore the data: Once you have cleaned the data, you must understand the information contained within at a high level. What kinds of obvious trends or correlations do you see in the data? What are the high-level characteristics and are any of them more significant than others?

Perform in-depth analysis (machine learning, statistical models, algorithms): This step is usually the meat of your project, where you apply all the cutting-edge machinery of data analysis to unearth high-value insights and predictions.

Communicate results of the analysis: All the analysis and technical results that you produce are of little value unless you can explain to your stakeholders what they mean, in a way that's comprehensible and compelling. Data storytelling is a critical and underrated skill that you will build and use here.

So how can you help the VP of Sales at hotshot.io? In the next few sections, we will walk you through each step in the data science process, show you how it plays out in practice.

Step 1 of 6: Frame the problem (a.k.a. "ask the right questions")

The VP of Sales at hotshot.io, where you just started as a data scientist, has asked you to help optimize the sales funnel and improve conversion rates. Where do you start?

- You start by asking a lot of questions.
- Who are the customers, and how do you identify them?
- What does the sales process look like right now?
- What kind of information do you collect about potential customers?
- What are the different tiers of service right now?

Your goal is to get into your client's (the VP in this case) head and understand their view of the problem as well as you can. This knowledge will be invaluable later when you analyse your data and present the insights you find within.

Once you have a reasonable grasp of the domain, you should ask more pointed questions to understand exactly what your client wants you to solve. For example, you ask the VP of Sales, *"What does optimizing the funnel look like for you? What part of the funnel is not optimized right now?"*

She responds, *"I feel like my sales team is spending a lot of time chasing down customers who won't buy the product. I'd rather they spent their time with customers who are likely to convert. I also want to figure out if there are customer segments who are not converting well and figure out why that is."*

Bingo! You can now see the data science in the problem. Here are some ways you can frame the VP's request for data science questions:

1. What are some important customer segments?
2. How do conversion rates differ across these segments? Do some segments perform significantly better or worse than others?
3. How can we predict if a prospective customer is going to buy the product?
4. Can we identify customers who might be on the fence?
5. What is the return on investment (ROI) for different kinds of customers?

Spend a few minutes and think about any other questions you'd ask.

Now that you have a few concrete questions, you go back to the VP Sales and show her your questions. She agrees that these are all important questions but adds: *"I'm particularly interested in having a sense of how likely a customer is to convert. The other questions are pretty interesting too!"* You make a mental note to prioritize questions 3 and 4 in your story.

The next step for you is to figure out what data you have available to answer these questions. Stay tuned, we'll talk about that next time!

Step 2 of 6: Collect the right data

You've decided on your very first data science project for hotshot.io: predicting the likelihood that a prospective customer will buy the product.

Now's the time to start thinking about data. What data do you have available to you?

You find out that most of the customer data generated by the sales department are stored in the company's CRM software, and managed by the Sales Operations team. The backend for the CRM tool is a SQL database with several tables. However, the tool also provides a very convenient web-based API that returns data in the popular JSON format.

What data from the CRM database do you need? How should you extract it? What format should you store the data in to perform your analysis?

You decide to roll up your sleeves and dive into the SQL database. You find that the system stores detailed identity, contact and demographic information about customers, in addition to details of the sales process for each of them. You decide that since the dataset is not too large, you'll extract it to CSV files for further analysis.

As an ethical data scientist concerned with both security and privacy, you are careful not to extract any personally identifiable information from the database. All the information in the CSV file is anonymized and cannot be traced back to any specific customer.

In most data science industry projects, you will be using data that already exists and is being collected. Occasionally, you'll be leading efforts to collect new data, but that can be a lot of engineering work and it can take a while to bear fruit.

Well, now you have your data. Are you ready to start diving into it and cranking out insights? Not yet. The data you have collected is still 'raw data' — which is highly likely to contain mistakes, missing and corrupt values. Before you draw any conclusions from the data, you need to subject it to some data wrangling, which is the subject of our next section.

Step 3 of 6: How to process (or "wrangle") your data

As a brand-new data scientist at hotshot.io, you're helping the VP of Sales by predicting which prospective customers are likely to buy the product. To do so, you've extracted data from the company's CRM into CSV files.

But, despite all your work, you're not ready to use the data yet. First, you need to make sure the data is clean! Data cleaning and wrangling often take up the bulk of time in a data scientist's day-to-day work, and it's a step that requires patience and focus.

First, you need to look through the data that you've extracted and made sure you understand what every column means. One of the columns is called 'FIRST_CONTACT_TS', representing the date and time the customer was first contacted by hotshot.io. You automatically ask the following questions:

- Are there missing values i.e. being there customers without a first contact date? If not, why not? Is that a good or a bad thing?
- What's the time zone represented by these values? Do all the entries represent the same time zone?
- What is the date range? Is the date range valid? For example, if hotshot.io has been around since 2011, are there dates before 2011? Do they mean anything special or are they mistakes? It might be worth verifying the answer with a member of the sales team.

Once you have uncovered missing or corrupt values in your data, what do you do with them? You may throw away those records completely, or you may decide to use reasonable default values (based on feedback from your client). There are many options available here, and as a data scientist, your job is to decide which of them makes sense for your specific problem.

You'll have to repeat these steps for every field in your CSV file: you can begin to see why data cleaning is time-consuming. Still, this is a worthy investment of your time, and you patiently ensure that you get the data as clean as possible.

This is also a time when you make sure that you have all the critical pieces of data you need. To predict which future customers will convert, you need to know which customers have converted in the past. Conveniently enough, you find a column called 'CONVERTED' in your data, with a simple 'Yes/No' value.

Finally, after a lot of data wrangling, you're done cleaning your dataset, and you're ready to start drawing some insights from the data. Time for some exploratory data analysis!

Step 4 of 6: Explore your data

You've extracted data and spent a lot of time cleaning it up.

And now, you're finally ready to dive into the data! You're eager to find out what information the data contains, and which parts of the data are significant in answering your questions. This step is called exploratory data analysis.

What are some things you'd like to explore? You could spend days and weeks of your time aimlessly plotting away. But you don't have that much time. Your client, the VP of Sales, would love to present some of your results at the board meeting next week. The pressure is on!

You look at the original question: predict which prospects are likely to convert. What if you split the data into two segments based on whether the customer converted or not and examine differences between the two groups? Of course!

Right away, you start noticing some interesting patterns. When you plot the age distributions of customers on a histogram for the two categories, you notice that there are many customers in their early 30s who seem to buy the product and far fewer customers in their 20s. This is surprising since the product targets people in their 20s. Hmm, interesting …

Furthermore, many of the customers who convert were targeted via email marketing campaigns as opposed to social media. The social media campaigns make little difference. It's also clear that customers in their 20s are being targeted mostly via social media. You verify these assertions visually through plots, as well as by using some statistical tests from your knowledge of inferential statistics.

The next day, you walk up to the VP of Sales at her desk and show her your preliminary findings. She's intrigued and can't wait to see more! We'll show you how to present your results to her in our next section.

Step 5 of 6: Analyse Your Data in Depth

In the previous section, we explored a dataset to find a set of factors that could solve your original problem: predicting which customers at hotshot.io will buy the product. Now you have enough information to create a model to answer that question.

To create a predictive model, you must use techniques from machine learning. A machine learning model takes a set of data points, where each data point is expressed as a *feature vector.*

How do you produce these feature vectors? In our EDA phase, we identified several factors that could be significant in predicting customer conversion age and marketing method (email vs. social media). Notice an important difference between the two factors we've talked about: age is a numeric value whereas marketing method is a categorical value. As a data scientist, you know how to treat these values differently and how to correctly convert them to features.

Besides features, you also need labels. Labels tell the model which data points correspond to each category you want to predict. For this, you simply use the CONVERTED field in your data as a Boolean label (converted or not converted). 1 indicates that the customer converted, and 0 indicates that they did not.

Now that you have features and labels, you decide to use a simple machine learning classifier algorithm called logistic regression. A classifier is an instance of a broad category of machine learning techniques called '*supervised learning,*' where the algorithm learns a model from labelled examples. Contrary to supervised learning, *unsupervised learning* techniques extract information from data without any labels supplied.

You choose logistic regression because it's a technique that's simple, fast and it gives you not only a binary prediction about whether a customer will convert or not but also a probability of conversion. You apply the method to your data, tune the parameters, and soon, you're jumping up and down at your computer.

The VP of Sales is passing by, notices your excitement and asks, *"So, do you have something for me?"* And you burst out, *"Yes, the predictive model I created with logistic regression has a TPR of 95% and an FPR of 0.5%!"*

She looks at you as if you've sprouted a couple of extra heads and are talking to her in Martian.

You realize you haven't finished the job. You need to do the last critical step, which is making sure that you communicate your results to your client in a way that is compelling and comprehensible for them.

Step 6 of 6: Visualize and Communicate Your Findings

You now have an amazing machine learning model that can predict, with high accuracy, how likely a prospective customer is to buy Hotshot's product. But how do you convey its awesomeness to your client, the VP of Sales? How do you present your results to her in a form that she can use?

Communication is one of the most underrated skills a data scientist can have. While some of your colleagues (engineers, for example) can get away with being siloed in their technical bubbles, data scientists must be able to communicate with other teams and effectively translate their work for maximum impact. This set of skills is often called *'data storytelling.'*

So what kind of story can you tell based on the work you've done so far? Your story will include important conclusions that you can draw based on your exploratory analysis phase and the predictive model you've built. Crucially, you want the story to answer the questions that are most important to your client!

First and foremost, you take the data on the current prospects that the sales team is pursuing, run it through your model, and rank them in a spreadsheet in the order of most to least likely to convert. You provide the spreadsheet to your VP of Sales.

Next, you decide to highlight a couple of your most relevant results:

> **Age:** We're selling a lot more top prospects in their early 30s, rather than those in their mid-20s. This is unexpected since our product is targeted people in their mid-20s!
>
> **Marketing methods:** We use social media marketing to target people in their 20s, but email campaigns to people in their 30s. This appears to be a significant factor behind the difference in conversion rates.

The following week, you meet with her and walk her through your conclusions. She's ecstatic about the results you've given her! But then she asks you, *"How can we best use these findings?"*

Technically, your job as a data scientist is about analysing the data and showing what's happening. But as part of your role as the interpreter of data, you'll be often called upon to make recommendations about how others should use your results.

In response to the VP's question, you think for a moment and say, *"Well, first, I'd recommend using the spreadsheet with prospect predictions for the next week or two to focus on the most likely targets and see how well that performs. That'll make your sales team more productive right away and tell me if the predictive model needs more fine-tuning.*

Second, we should also investigate what's happening with our marketing and figure out whether we should be targeting the mid-20s crowd with email campaigns or making our social media campaigns more effective."

The VP of Sales nods enthusiastically in agreement and immediately sets you up for a meeting with the VP of Marketing so you can demonstrate your results to him. Moreover, she asks you to send a couple of slides summarizing your results and recommendations, so she can present them at the board meeting.

You've successfully finished your first data science project at work, and you finally understand what your mentors have always said: data science is not just about the techniques, the algorithms or the math. It's not just about the programming and implementation. It's a truly multi-disciplinary field, one that requires the practitioner to translate between technology and business concerns. This is what makes the career path of data science so challenging, and so valuable.

Chapter 03 – General Theory of Artificial Intelligence

Artificial Intelligence (AI)

Human Intelligence exhibited by Machines (Abstract-Thinking, Self-Reasoning & Knowledge Representation)

The theory and development of computer systems (Self-Learning) able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages.

Machine Learning (ML)

An Approach (Mathematical & Statistical) to Achieve Artificial Intelligence

Machine learning is an application of artificial intelligence (AI) that provides systems with the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

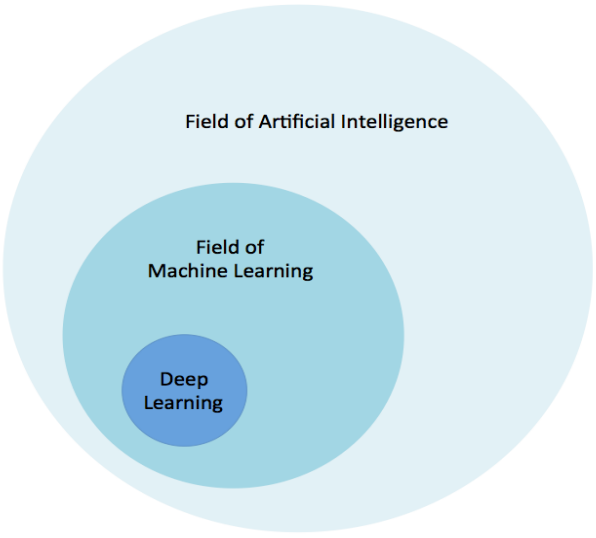
Deep Learning (DL)

A Technique for Implementing Machine Learning

Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms.

Relationship between AI, ML & DL

The field of AI is broad and has been around for a long time. Deep learning is a subset of the field of machine learning, which is a subfield of AI



You can think of deep learning, machine learning and artificial intelligence as a set of Russian dolls nested within each other, beginning with the smallest and working out. Deep learning is a subset of machine learning, and machine learning is a subset of AI, which is an umbrella term for any computer program that does something smart. In other words, all machine learning is AI, but not all AI is machine learning, and so forth.

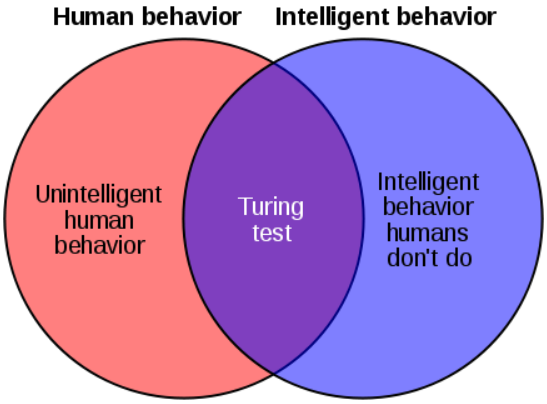
Cognitive Computing (CC)

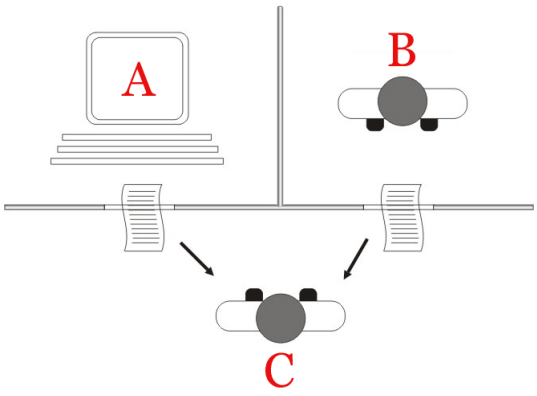
Cognitive computing (CC) describes technology platforms that, broadly speaking, are based on the scientific disciplines of artificial intelligence and signal processing. These platforms encompass machine learning, reasoning, natural language processing, speech recognition and vision (object recognition), human-computer interaction, dialogue and narrative generation, among other technologies.

Turing Test

A test for intelligence in a computer, requiring that a human being should be unable to distinguish the machine from another human being by using the replies to questions put to both

The ***"standard interpretation"*** of the Turing Test, in which player C, the interrogator, is given the task of trying to determine which player – A or B – is a computer and which is a human. The interrogator is limited to using the responses to written questions to make the determination.





History of AI, ML, DL & CC

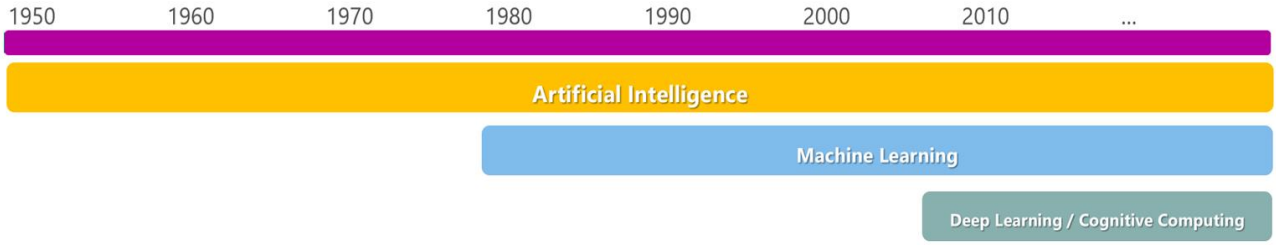
History of AI

Beginning in the 1950s, modern AI focused on what was called strong AI, which referred to AI that could generally perform any intellectual task that a human could. The lack of progress in strong AI eventually led to what's called weak AI or applying AI techniques to narrower problems. Until the 1980s, AI research was split between these two paradigms. But, around 1980, machine learning became a prominent area of research, its purpose to give computers the ability to learn and build models so that they could perform activities like prediction within specific domains.

Building on research from both AI and machine learning, deep learning emerged around 2000. Computer scientists used neural networks in many layers with new topologies and learning methods. This evolution of neural networks has successfully solved complex problems in various domains.

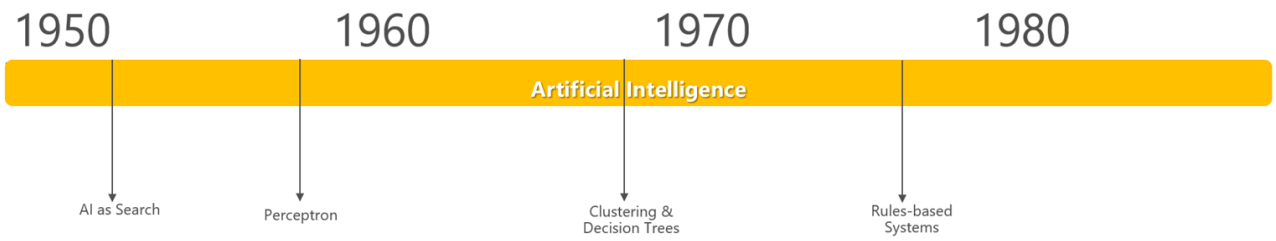
In the past decade, cognitive computing has emerged, the goal of which is to build systems that can learn and naturally interact with humans. Cognitive computing was demonstrated by IBM Watson by successfully defeating world-class opponents at the game Jeopardy.

Foundation of AI



AI as a Search

Most AI can be solved through ***brute-force*** search (***depth-first or breadth-first*** search). However, basic search quickly suffers considering the search space for moderate problems. One of the earliest examples of AI as the search was the development of a checkers-playing program. Arthur Samuel built the first such program on the IBM 701 Electronic Data Processing Machine, implementing an optimization to search trees called ***alpha-beta pruning***. His program also recorded the reward for a specific move, allowing the application to learn with each

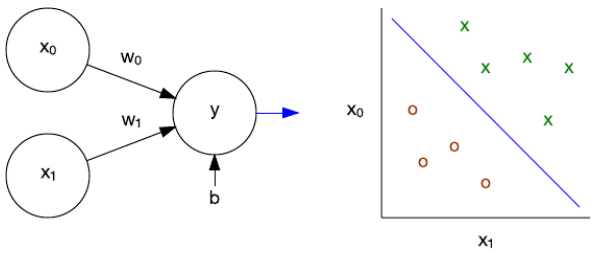


game played (making it the first self-learning program). To increase the rate at which the program learned, Samuel programmed it to play itself, increasing its ability to play and learn.

Although you can successfully apply search to many simple problems, the approach quickly fails as the number of choices increases. Take the simple game of tic-tac-toe as an example. At the start of a game, there are nine possible moves. Each move results in eight possible countermoves, and so on. The full tree of moves for tic-tac-toe contains (unoptimized for rotation to remove duplicates) is 362,880 nodes. If you then extend this same thought experiment to chess or Go, you quickly see the downside of search.

Perceptrons

The Perceptron was an early supervised learning algorithm for single-layer neural networks. Given an input feature



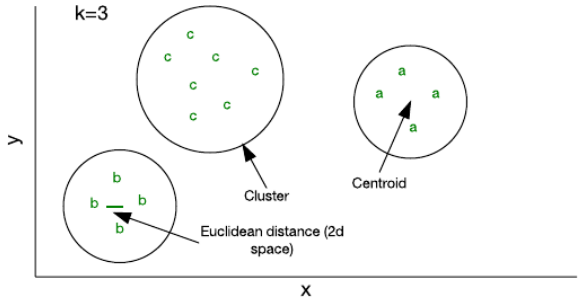
vector, the perceptron algorithm could learn to classify inputs as belonging to a specific class. Using a training set,

the network's weights and bias could be updated for linear classification. The perceptron was first implemented for the IBM 704, and then on custom hardware for image recognition.

As a linear classifier, the perceptron was capable of linear separable problems. The key example of the limitations of the perceptron was its inability to learn an exclusive OR (XOR) function. Multilayer Perceptrons solved this problem and paved the way for more complex algorithms, network topologies, and deep learning.

Clustering algorithms

With Perceptrons, the approach was supervised. Users provided data to train the network, and then test the network against new data. Clustering algorithms take a different approach called unsupervised learning. In this model, the algorithm organizes a set of feature vectors into clusters based on one or more attributes of the data.



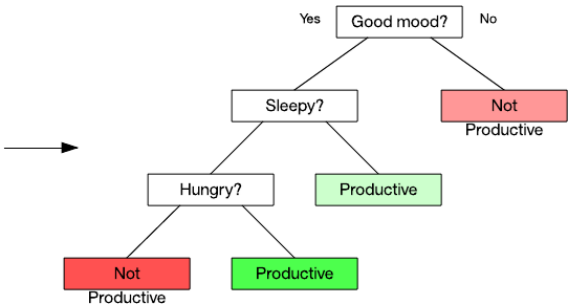
One of the simplest algorithms that you can implement in a small amount of code is called k-means. In this algorithm, k indicates the number of clusters in which you can assign samples. You can initialize a cluster with a random feature vector, and then add all other samples to their closest cluster (given that each sample represents a feature vector and a Euclidean distance used to identify "distance"). As you add samples to a cluster, its centroid—that is, the centre of the cluster—is recalculated. The algorithm then checks the samples again to ensure that they exist in the closest cluster and ends when no samples change cluster membership.

Although k-means is relatively efficient, you must specify k in advance. Depending on the data, other approaches might be more efficient, such as hierarchical or distribution-based clustering.

Decision trees

Closely related to clustering is the decision tree. A decision tree is a predictive model of observations that lead to some conclusion. Conclusions are represented as leaves on the tree, while nodes are decision points where an observation diverges. Decision trees are built from decision tree learning algorithms, where the data set is split into subsets based on attribute value tests (through a process called recursive partitioning).

Sleepy	Hungry	Good mood	Productive?
No	No	No	No
No	No	Yes	Yes
No	Yes	No	No
No	Yes	Yes	Yes
Yes	No	No	No
Yes	No	Yes	Yes
Yes	Yes	No	No
Yes	Yes	Yes	No



Consider the example in the following figure. In this data set, we can observe when someone was productive based on three factors. Using a decision tree learning algorithm, we can identify attributes by using a metric (one

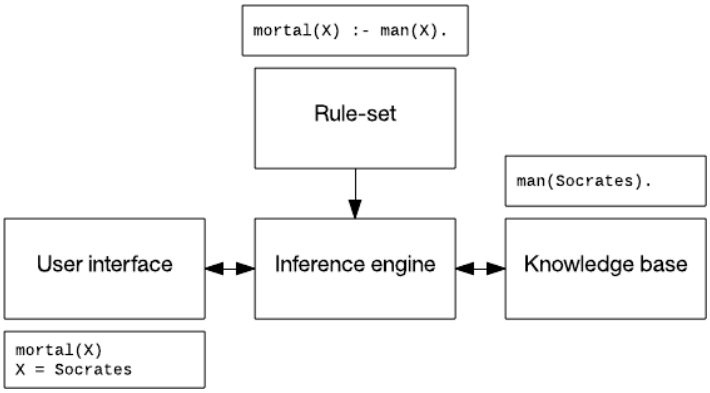
example is information gain). In this example, the mood is a primary factor in productivity, so the data set split according to whether "good mood" is Yes or No. The No side is simple: It's always non-productive. But, the Yes side requires us to split the data set again based on the other two attributes. The data set is colourized to illustrate where observations led to the leaf nodes.

A useful aspect of decision trees is their inherent organization, which gives you the ability to easily (and graphically) explain how you classified an item. Popular decision tree learning algorithms include C4.5 and the Classification and Regression Tree.

Rules-based systems

The first system built on rules and inference, called Dendral, was developed in 1965, but it wasn't until the 1970s that these so-called "expert systems" hit their stride. A rules-based system is one that stores both knowledge and rules and uses a reasoning system to draw conclusions.

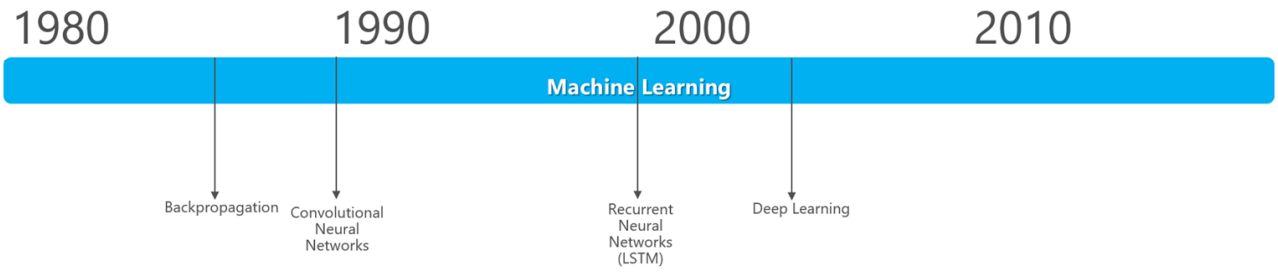
A rules-based system typically consists of a rule set, a knowledge base, an inference engine (using forward or backward rule chaining), and a user interface. In the following figure, we use a piece of knowledge ("Socrates was a man"), a rule ("if a man, then mortal"), and an interaction on who is mortal.



Rules-based systems have been applied to speech recognition, planning and control, and disease identification. One system developed in the 1990s for monitoring and diagnosing dam stability, called Kaleidos, is still in operation today.

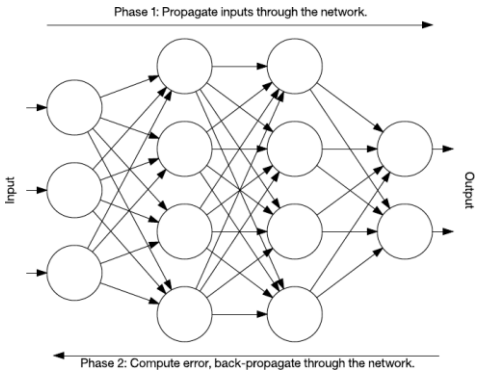
History of Machine Learning

Machine learning is a subfield of AI and computer science that has its roots in statistics and mathematical optimization. Machine learning covers techniques in supervised and unsupervised learning for applications in prediction, analytics, and data mining. It is not restricted to deep learning, and in this section, we explore some of the algorithms that have led to this surprisingly effective approach.



Back-propagation

The true power of neural networks is their multilayer variant. Training single-layer Perceptrons is straightforward, but the resulting network is not very powerful. The question became, how can we train networks that have multiple layers? This is where back-propagation came in.



Back-propagation is an algorithm for training neural networks that have many layers. It works in two phases. The first phase is the propagation of inputs through a neural network to the final layer (called feed-forward). In the second phase, the algorithm computes an error and then back-propagates this error (adjusting the weights) from the final layer to the first.

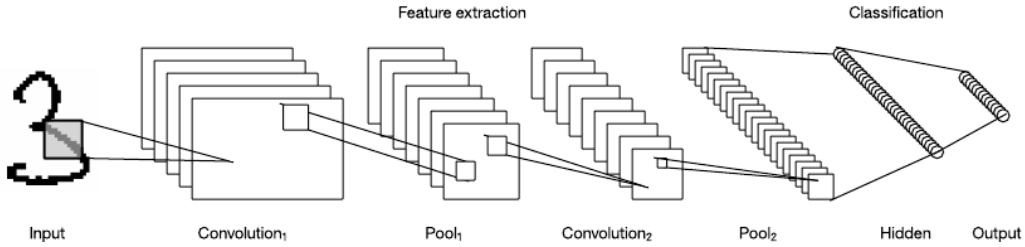
During training, intermediate layers of the network organize themselves to map portions of the input space to the output space. Back-propagation, through supervised learning, identifies an error in the input-to-output mapping and then adjusts the weights accordingly (with a learning rate) to correct this error. Back-propagation continues to be an important aspect of neural network learning. With faster and cheaper computing resources, it continues to be applied to larger and denser networks.

Convolutional neural networks

Convolutional neural networks (CNNs) are multilayer neural networks that take their inspiration from the animal visual cortex. The architecture is useful in various applications, including image processing. The first CNN was created by Yann LeCun, and at the time, the architecture focused on handwritten character-recognition tasks like reading postal codes.

The LeNet CNN architecture is made up of several layers that implement feature extraction, and then classification. The image is divided into receptive fields that feed into a convolutional layer that extracts features from the input image. The next step is pooling, which reduces the dimensionality of the extracted features (through down-sampling) while retaining the most important information (typically through max pooling). The algorithm then performs another convolution and pooling step that feeds into a fully connected, multilayer perceptron. The final output layer of this network is a set of nodes that identify features of the image (in this case, a node per identified number). Users can train the network through back-propagation.

The use of deep layers of processing, convolutions, pooling, and a fully connected classification layer opened the door to various new applications of neural networks. In addition to image processing, the CNN has been successfully applied to video recognition and many tasks within natural language processing. CNNs have also been efficiently implemented within GPUs, greatly improving their performance.



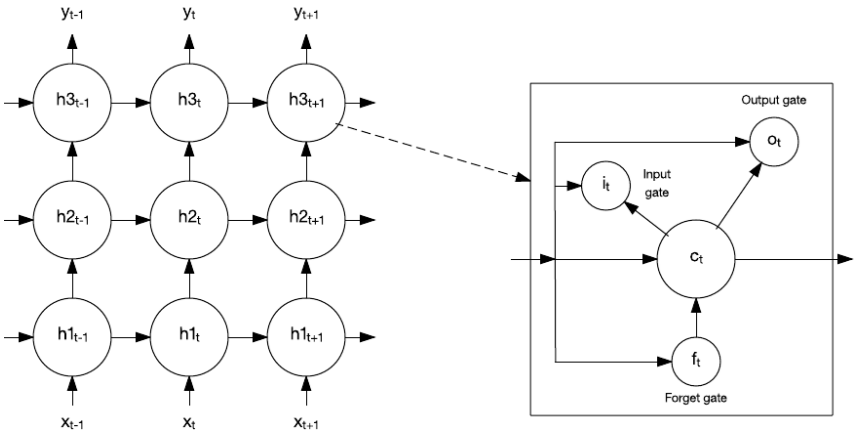
Long short-term memory (LSTM)

Recall in the discussion of back-propagation. that the network being trained was feed-forward. In this architecture, users feed inputs into the network and propagate them forward through the hidden layers to the output layer. But, many other neural network topologies exist. One, which I investigate here, allows connections between nodes to form a directed cycle. These networks are called recurrent neural networks, and they can feed backwards to prior layers or to subsequent nodes within their layer. This property makes these networks ideal for time series data.

In 1997, a special kind of recurrent network was created called the long short-term memory (LSTM). The LSTM consists of memory cells that within a network remember values for a short or long time.

A memory cell contains gates that control how information flows into or out of the cell. The input gate controls when new information can flow into the memory. The forget gate controls how long an existing piece of information is retained. Finally, the output gate controls when the information contained in the cell is used in the output from the cell. The cell also contains weights that control each gate. The training algorithm, commonly back-propagation-through-time (a variant of back-propagation.), optimizes these weights based on the resulting error.

The LSTM has been applied to speech recognition, handwriting recognition, text-to-speech synthesis, image captioning, and various other tasks.

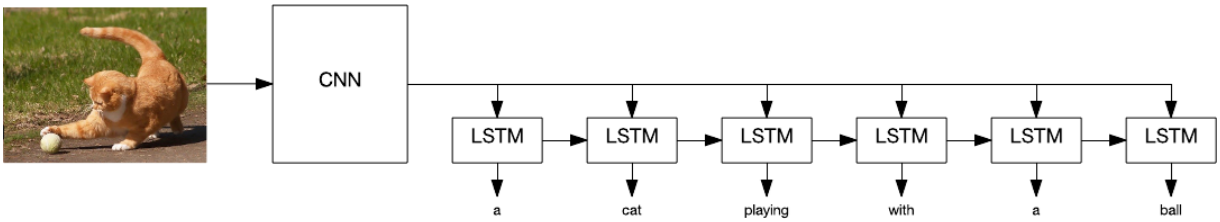


Deep learning

Deep learning is a relatively new set of methods (CNN)that's changing machine learning in fundamental ways. Deep learning isn't an algorithm, per se, but rather a family of algorithms that implement deep networks with

unsupervised learning. These networks are so deep that new methods of computation, such as GPUs, are required to build them (in addition to clusters of compute nodes).

This article has explored two deep learning algorithms so far: CNNs and LSTMs. These algorithms have been combined to achieve several surprisingly intelligent tasks. As shown in the following figure, CNNs and LSTMs have been used to identify, and then describe in natural language a picture or video.



Deep learning algorithms have also been applied to facial recognition, identifying tuberculosis with 96 percent accuracy, self-driving vehicles, and many other complex problems.

However, despite the results of applying deep learning algorithms, problems exist that we have yet to solve. A recent application of deep learning to skin cancer detection found that the algorithm was more accurate than a board-certified dermatologist. But, where dermatologists could enumerate the factors that led to their diagnosis, there's no way to identify which factors a deep learning program used in its classification. This is called deep learning's black box problem.

Another application, called Deep Patient, was able to successfully predict disease given a patient's medical records. The application proved to be considerably better at forecasting disease than physicians—even for schizophrenia, which is notoriously difficult to predict. So, even though the models work well, no one can reach into the massive neural networks to identify why.

Cognitive computing

AI and machine learning are filled with examples of biological inspiration. And, while early AI focused on the grand goals of building machines that mimicked the human brain, cognitive computing is working toward this goal.

Cognitive computing, building on neural networks and deep learning, is applying knowledge from cognitive science to build systems that simulate human thought processes. However, rather than focus on a singular set of technologies, cognitive computing covers several disciplines, including machine learning, natural language processing, vision, and human-computer interaction.

An example of cognitive computing is IBM Watson, Microsoft Cognitive Services, which demonstrated state-of-the-art question-and-answer interactions on *Jeopardy,* but that IBM has since extended through a set of web services. These services expose application programming interfaces for visual recognition, speech-to-text, and text-to-speech function; language understanding and translation; and conversational engines to build powerful virtual agents.

Machine Learning Relationship

Machine learning has a relationship with several areas:

Statistics It uses the elements of data sampling, estimation, hypothesis testing, learning theory, and statistical-based modeling, to name a few **Algorithms and computation**: It uses the basic concepts of search, traversal, parallelization, distributed computing, and so on from basic computer science **Database and knowledge discovery**: For its ability to store, retrieve, and access information in various formats **Pattern recognition**: For its ability to find interesting patterns from the data to explore, visualize, and predict

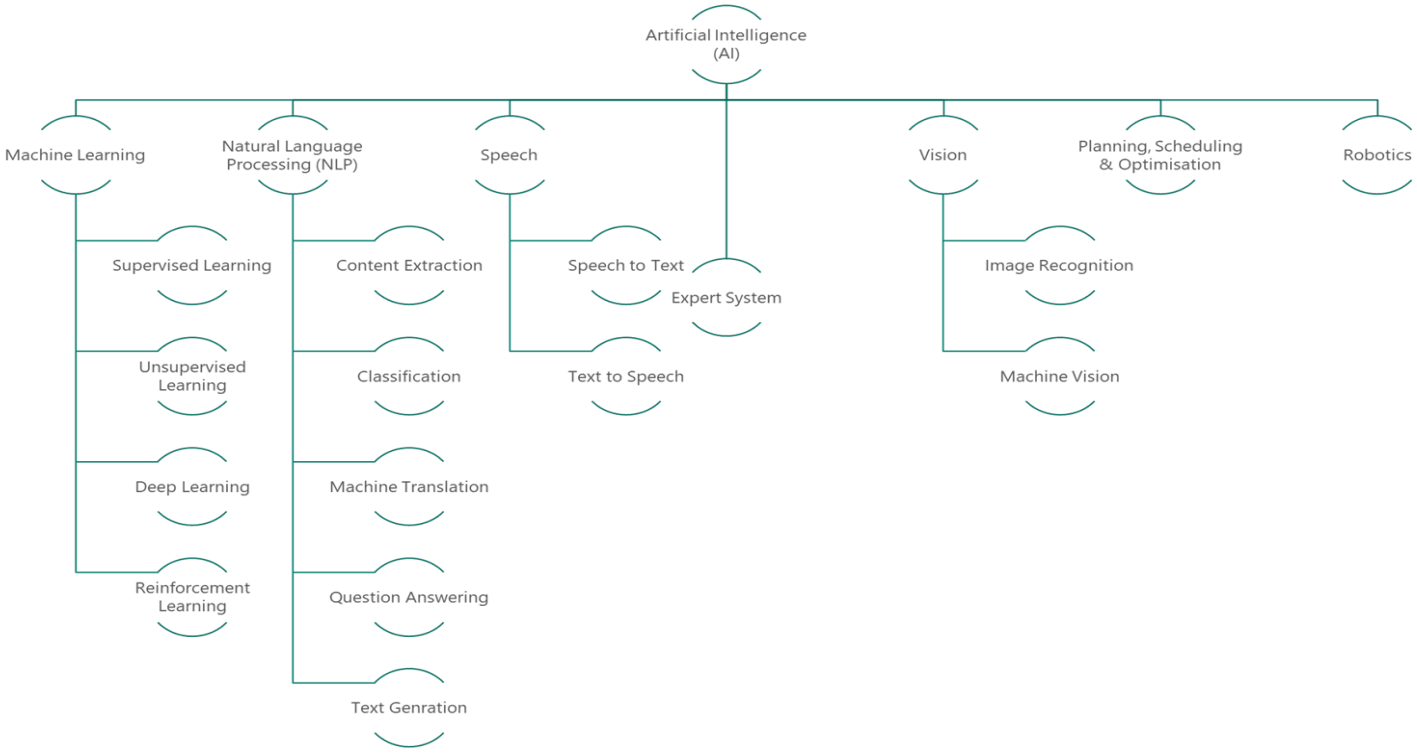
Artificial intelligence Though it is considered a branch of artificial intelligence, it also has relationships with other branches, such as heuristics, optimization, evolutionary computing, and so on

What is Not Machine Learning?

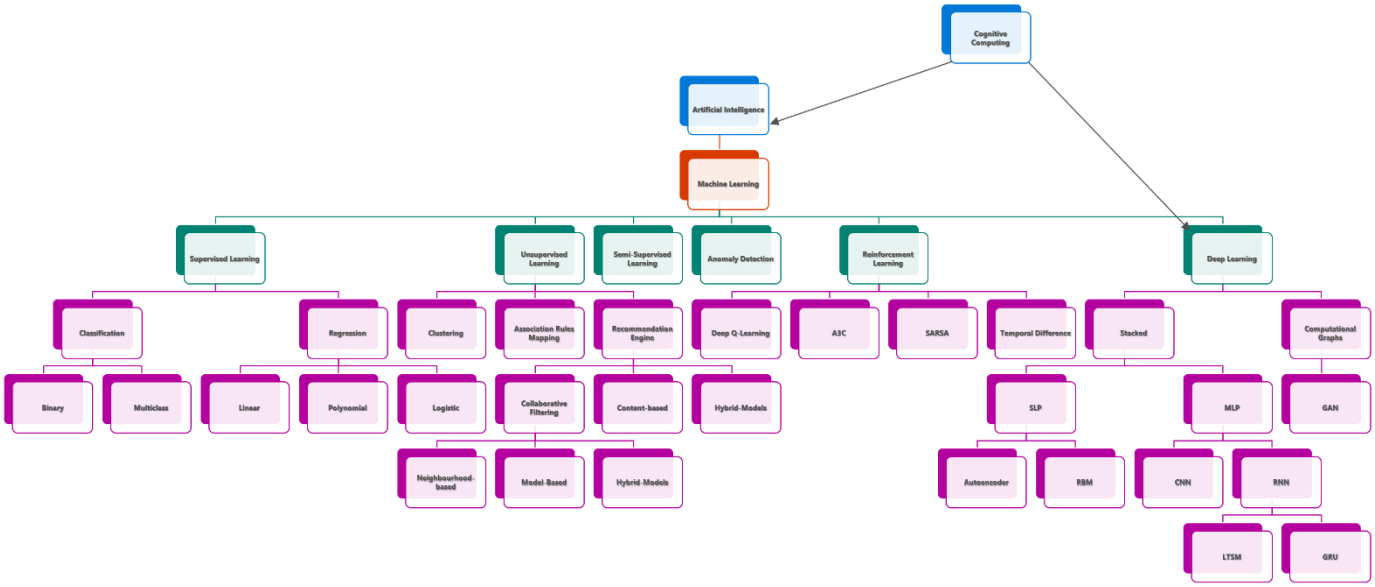
It is important to recognize areas that share a connection with machine learning but cannot themselves be considered part of machine learning. Some disciplines may overlap to a smaller or larger extent, yet the principles underlying machine learning are quite distinct:

- **Business intelligence (BI) and reporting**: Reporting **key performance indicators** (**KPI's**), querying OLAP for slicing, dicing, and drilling into the data, dashboards, and so on that form, the central components of BI are not machine learning.
- **Storage and ETL**: Data storage and ETL are key elements in any machine learning process, but, by themselves, they don't qualify as machine learning.
- **Information retrieval, search, and queries**: The ability to retrieve data or documents based on search criteria or indexes, which form the basis of information retrieval, are not really machine learning. Many forms of machine learning, such as semi-supervised learning, can rely on the searching of similar data for modelling, but that doesn't qualify to search as machine learning.
- **Knowledge representation and reasoning**: Representing knowledge for performing complex tasks, such as ontology, expert systems, and semantic webs, do not qualify as machine learning.

Practices of AI



Hierarchy of Artificial Intelligence

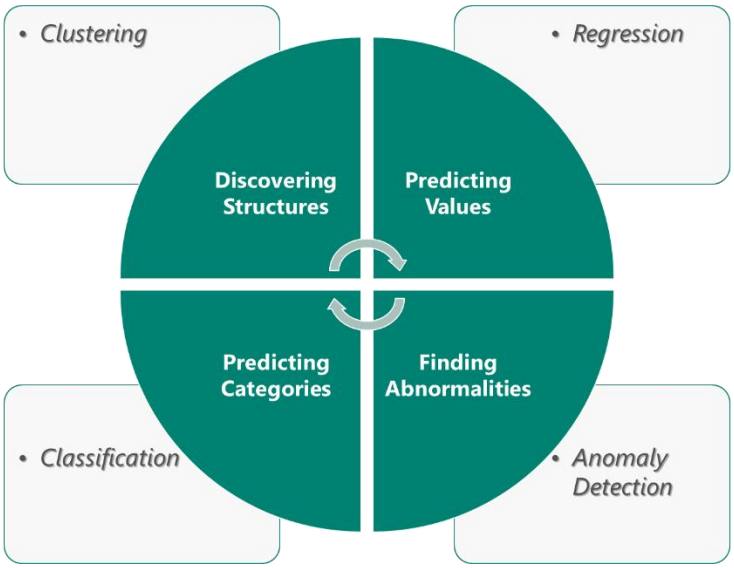


Problem Space Selection Criteria

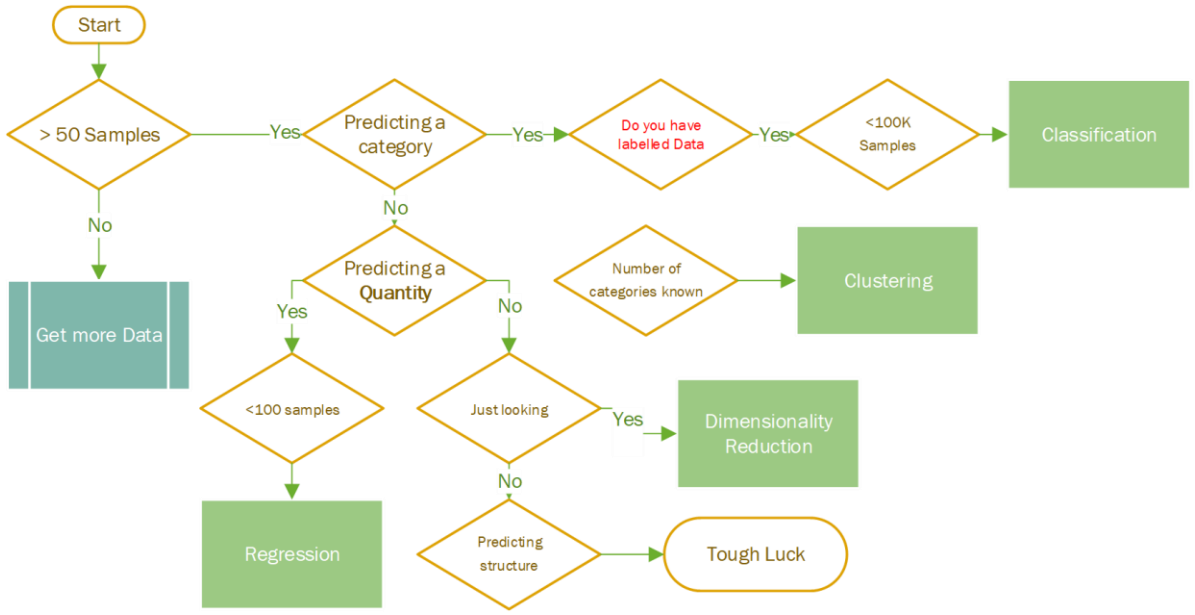
S#	???	Type Selection	Example
1.	***How much or How many***	Regression	Predicting Price

2.	***Which Category***	Classification	Email Spam Filtration
3.	***Which Groups***	Clustering	Customer Segmentation
4.	***Is it odd, weird or any abnormality***	Anomaly Detection	Preventive Maintenance
5.	***Which Option***	Recommendations	Movie Recommendations or Online Purchase suggestions
6.	***How Deep & Complex***	Deep Learning	Image Analysis (Computer Vision), Text Analytics (Speech Synthesis)

ML Type Selection Criteria



What type of Problem my domain is?



Walking Skeleton for Algorithms

Supervised Learning

All data is labelled, and the algorithms learn to predict the output from the input data

Supervised learning is the machine learning task of inferring a function from labelled training data. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and the desired output value (also called the supervisory signal).

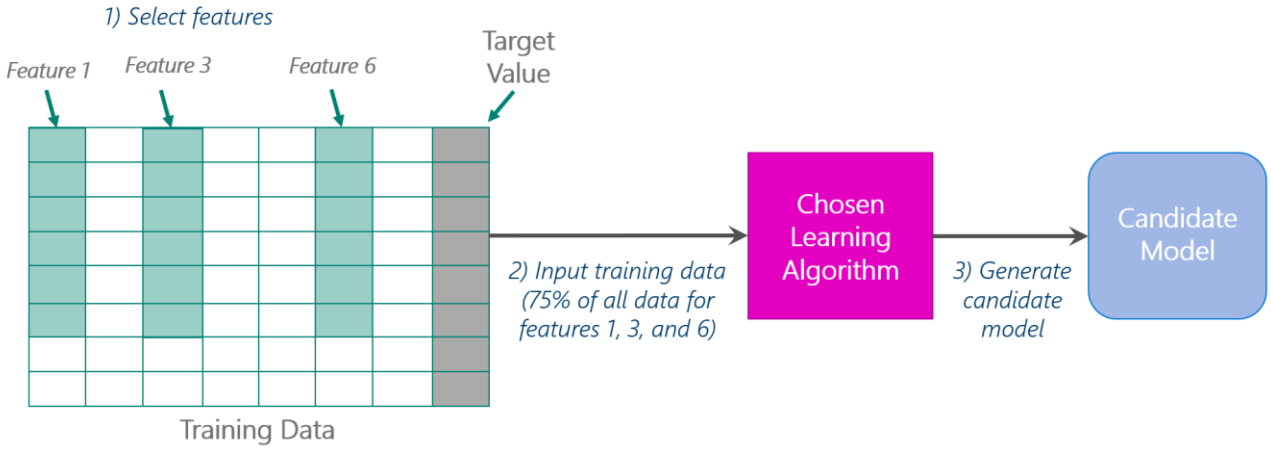
A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way.

The majority of practical machine learning uses supervised learning. Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

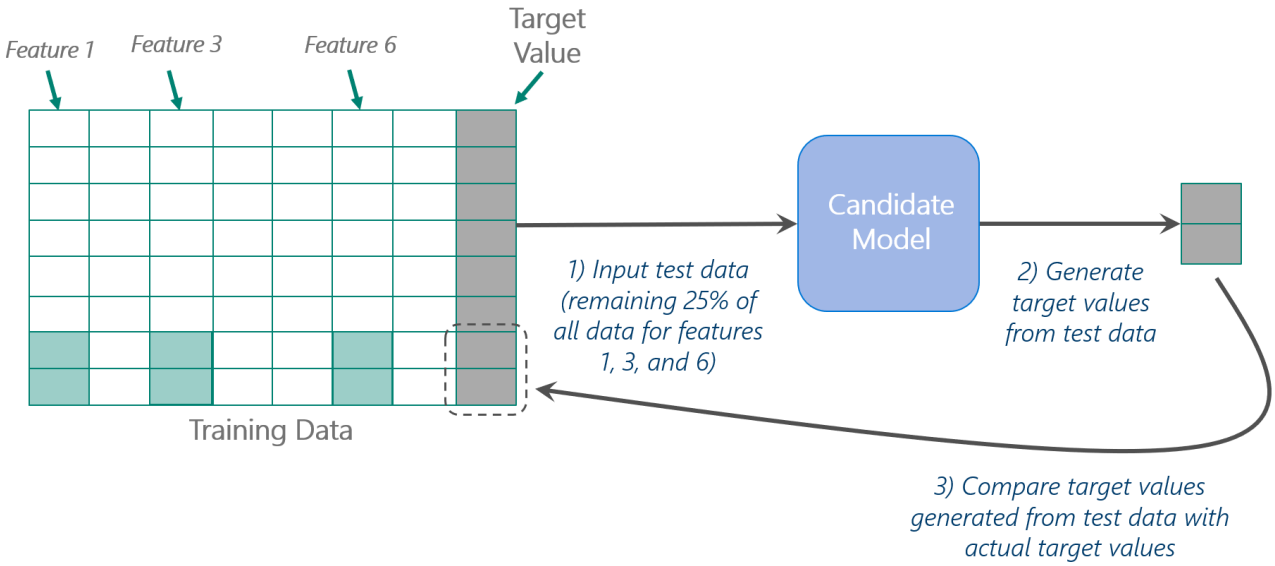
$$Y = f(X)$$

The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.

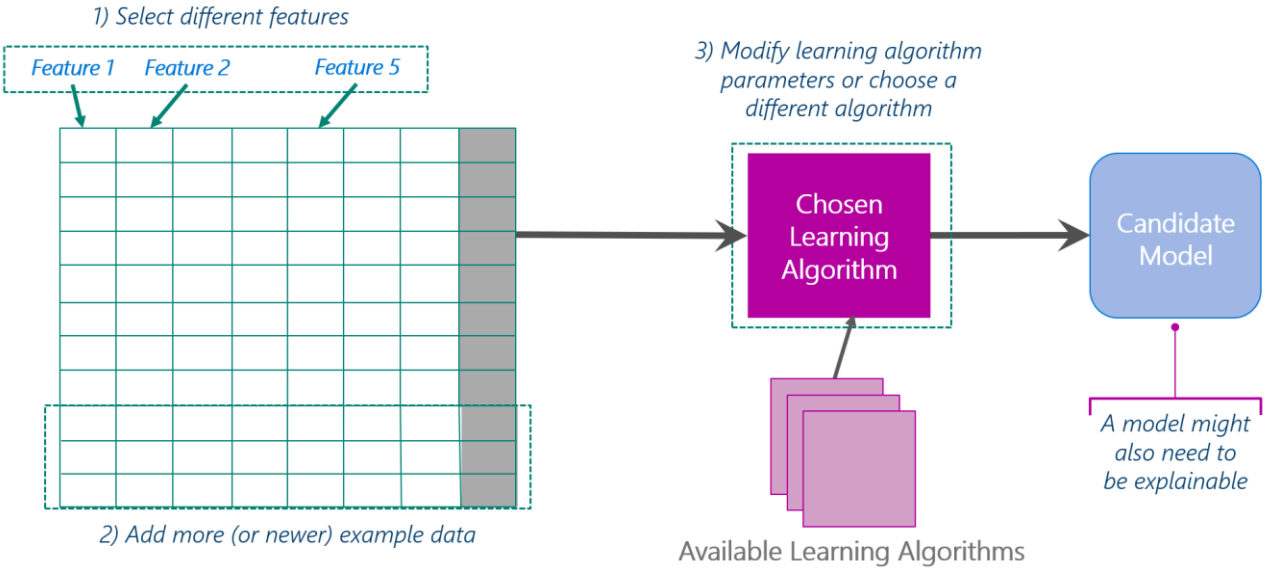
It is called supervised learning because the process of an ***algorithm*** learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers; the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.



Training Model



Testing Model



Updating Model

Supervised learning problems can be further grouped into regression and classification problems.

- **Classification** A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”.
- **Regression** A regression problem is when the output variable is a real continuous value, such as “dollars” or “weight”.

Some common types of problems built on top of classification and regression include recommendation and time series prediction respectively.

Some popular examples of supervised machine learning algorithms are:

- Linear regression for regression problems.
- Random forest for classification and regression problems.
- Support vector machines for classification problems.

Unsupervised Learning

All data is unlabelled, and the algorithms learn to inherent structure from the input data

Unsupervised machine learning is the machine learning task of inferring a function to describe hidden structure from "unlabelled" data (a classification or categorization is not included in the observations). Since the examples given to the learner are unlabelled, there is no evaluation of the accuracy of the structure that is output by the relevant algorithm—which is one way of distinguishing unsupervised learning from supervised learning and reinforcement learning.

A central case of unsupervised learning is the problem of density estimation in statistics, though unsupervised learning encompasses many other problems (and solutions) involving summarizing and explaining key features of the data.

Unsupervised learning is where you only have input data (X) and no corresponding output variables.

The goal for unsupervised learning is to model the underlying structure or distribution in the data to learn more about the data.

These are called unsupervised learning because unlike supervised learning above there are no correct answers and there is no teacher. Algorithms are left to their own devices to discover and present the interesting structure in the data.

Unsupervised learning problems can be further grouped into clustering and association problems.

- **Clustering** A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behaviour.
- **Association** An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Some popular examples of unsupervised learning algorithms are:

- K-Means for clustering problems.
- Apriori algorithm for association rule learning problems.

Semi-Supervised Learning

Some data is labelled but most of it is unlabelled and a mixture of supervised and unsupervised techniques can be used

Semi-supervised learning is a class of supervised learning tasks and techniques that also make use of unlabeled data for training – typically a small amount of labelled data with a large amount of unlabeled data. Semi-supervised learning falls between unsupervised learning (without any labelled training data) and supervised learning (with completely labelled training data). Many machine-learning researchers have found that unlabelled data when used in conjunction with a small amount of labelled data, can produce considerable improvement in learning accuracy. The acquisition of labelled data for a learning problem often requires a skilled human agent (e.g. to transcribe an audio segment) or a physical experiment (e.g. determining the 3D structure of a protein or determining whether there is oil at a location). The cost associated with the labelling process thus may render a fully labelled training set infeasible, whereas acquisition of unlabelled data is relatively inexpensive. In such situations, semi-supervised learning can be of great practical value. Semi-supervised learning is also of theoretical interest in machine learning and as a model for human learning.

Problems where you have a large amount of input data (X) and only some of the data is labelled (Y) are called semi-supervised learning problems.

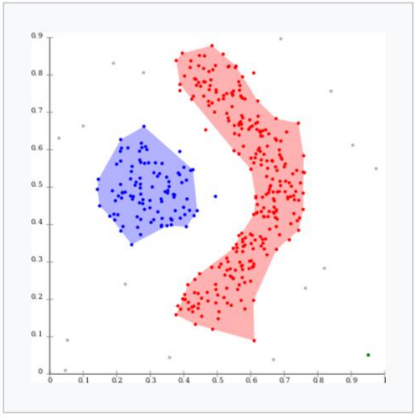
These problems sit in between both supervised and unsupervised learning.

A good example is a photo archive where only some of the images are labelled, (e.g. dog, cat, person) and the majority are unlabelled.

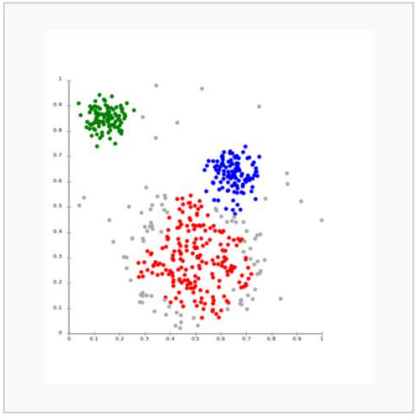
Many real-world machine learning problems fall into this area. This is because it can be expensive or time-consuming to label data as it may require access to domain experts. Whereas unlabelled data is cheap and easy to collect and store.

You can use unsupervised learning techniques to discover and learn the structure in the input variables.

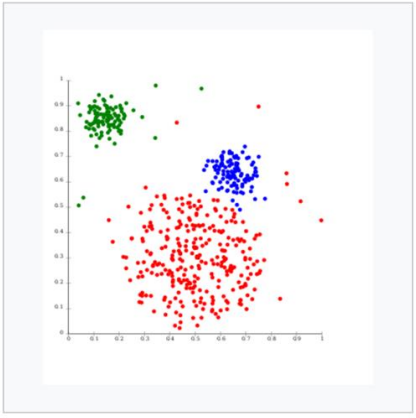
You can also use supervised learning techniques to make best guess predictions for the unlabelled data, feed that data back into the supervised learning algorithm as training data and use the model to make predictions on new unseen data.



Density-based clustering with DBSCAN.



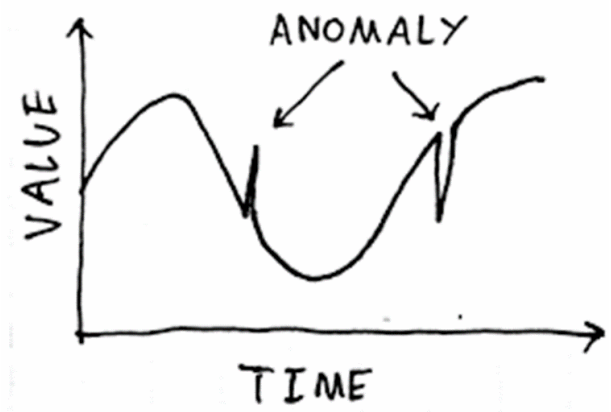
DBSCAN assumes clusters of similar density, and may have problems separating nearby clusters

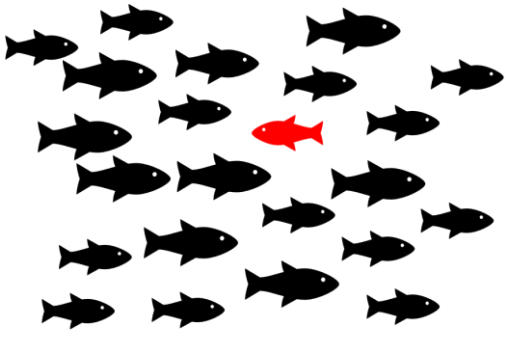


OPTICS is a DBSCAN variant that handles different densities much better

Anomaly Detection

The process of identifying rare or unexpected items or events in a dataset that do not conform to other items in the dataset





In data mining, anomaly detection (also outlier detection) is the identification of items, events or observations which do not conform to an expected pattern or other items in a dataset. Typically, the anomalous items will translate to a problem such as bank fraud, a structural defect, medical problems or errors in a text. Anomalies are also referred to as outliers, novelties, noise, deviations and exceptions.

Outliers to a normal data pattern, Machines breaking down, Perfect storm, Superwave, cannot be one-off but must be a recurring phenomenon over a span of time.

Popular Techniques for Anomaly Detection:

- Several anomaly detection techniques have been proposed in the literature. Some of the popular techniques are:
- Density-based techniques (k-nearest neighbour (k-nn), local outlier factor and many more variations of this concept).
- Subspace and correlation-based outlier detection for high-dimensional data.
- One class support vector machines.
- Replicator neural networks.
- Cluster analysis-based outlier detection.
- Deviations from association rules and frequent item sets.
- Fuzzy logic based outlier detection.
- Ensemble techniques, using feature bagging, score normalization and different sources of diversity.

The performance of different methods depends a lot on the dataset and parameters, and methods have little systematic advantages over another when compared across many datasets and parameters.

Categories of Anomaly Detection

- Unsupervised anomaly detection
- Supervised anomaly detection
- Semi-Supervised anomaly detection

Three broad categories of anomaly detection techniques exist. Unsupervised anomaly detection techniques detect anomalies in an unlabelled test data set under the assumption that most of the instances in the dataset are normal by looking for instances that seem to fit least to the remainder of the dataset. Supervised anomaly detection techniques require a data set that has been labelled as "normal" and "abnormal" and involves training a classifier (the key difference to many other statistical classification problems is the inherently unbalanced nature of outlier detection). Semi-supervised anomaly detection techniques construct a model representing normal behaviour from a given normal training dataset and then testing the likelihood of a test instance to be generated by the learnt model.

Reinforcement Learning

Reinforcement learning is Ground-hog Day for algorithms

Neural networks have become well known for recent advances in such diverse fields as computer vision, machine translation and time series prediction – but reinforcement learning may be their killer app.

Reinforcement learning is goal-oriented. RL algorithms learn how to attain a complex objective or maximize along a dimension over many steps, starting from a blank slate, and under the right conditions, they achieve superhuman performance.

Reinforcement algorithms with deep learning at their core are currently beating expert humans at numerous Atari video games. While that may sound trivial, it's a vast improvement over their previous accomplishments. Two reinforcement learning algorithms – Deep-Q learning and A3C – have been implemented in a deep learning that can play Doom game already.

In time, we expect reinforcement learning to perform better in more ambiguous, real-life environments while choosing from an arbitrary number of possible actions, rather than from the limited options of a video game. When people talk about building robot armies, this is what they mean.

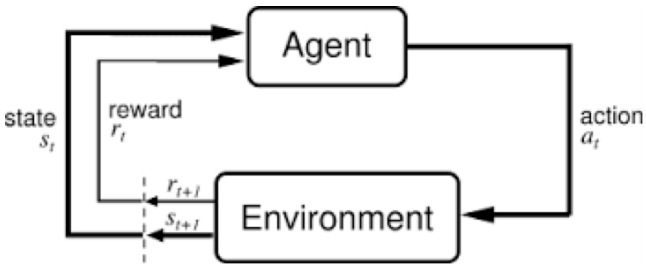
Reinforcement learning is based on agents, environments, states, actions and rewards, all of which we'll explain.

An agent takes actions; for example, a drone making a delivery, or Super Mario navigating a video game.

A state is a situation in which the agent finds itself; i.e. a specific place and moment, a configuration that puts the agent in relation to other significant things such as tools, obstacles, enemies or prizes.

An action is almost self-explanatory, but it should be noted that agents choose among a list of possible actions. In video games, the list might include running right or left, jumping high or low, crouching or standing still. In the stock markets, the list might include buying, selling or holding any one of an array of securities and their derivatives. When handling aerial drones, alternatives would include many different velocities and accelerations in 3D space.

A reward is a feedback by which we measure the success or failure of an agent's actions. For example, in a video game, when Mario touches a coin, he wins points. An agent sends output in the form of actions to the environment, and the environment returns the agent's new state as well as rewards.



In the feedback loop above, the subscripts denote time steps t and t+1, each of which refers to different states: the state at moment t, and the state at moment t+1. Unlike other forms of machine learning – such as supervised and unsupervised learning – reinforcement learning can only be thought about sequentially in terms of state-action pairs that occur one after the other.

Reinforcement learning judge's actions by the results they produce. It is goal oriented, and its aim is to learn sequences of actions that will lead it to achieve its goal. In video games, the goal is to finish the game with the most points, so each additional point obtained throughout the game will affect the agent's subsequent behaviour; i.e. the agent may learn that it should shoot battleships, touch coins or dodge meteors to maximize its score.

In the real world, the goal might be for a robot to travel from point A to point B, and every inch the robot is able to move closer to point B could be counted like points.

RL differs from both supervised and unsupervised learning by how it interprets inputs. We can illustrate their difference by describing what they learn about a "thing."

Unsupervised learning: That thing is like this other thing. (Similarities w/o names, and the inverse: anomaly detection)

Supervised learning: That thing is a "double bacon cheeseburger". (Labels, putting names to faces…)

Reinforcement learning Eat that thing because it tastes good and will keep you alive. (Actions based on short- and long-term rewards.)

One way to imagine an autonomous RL agent would be as a blind person attempting to navigate the world with their ears and a white cane. Agents have small windows that allow them to perceive their environment, and those windows may not even be the most appropriate way for them to perceive what's around them.

(In fact, deciding *which types* of feedback your agent should pay attention to is a hard problem to solve, and glossed over by algorithms that are learning how to play video games, where the kinds of feedback are limited and well defined. These video games are much closer to the sterile environment of the lab, where ideas about reinforcement learning were initially tested.)

The goal of reinforcement learning is to pick the best-known action in any state, which means the actions must be ranked, assigned values relative to one another.

Since those actions are state dependent, what we are really gauging is the value of state-action pairs; i.e. an action taken from a certain state, something you did somewhere.

If the action is marrying someone, then marrying a 35-year-old when you're 18 should mean something different than marrying a 35-year-old when you're 90.

If the action is yelling "Fire!" Performing the action, a crowded theatre should mean something different from performing the action next to a squad of men with rifles. We can't predict an action's outcome without knowing the context.

We map state-action pairs to the values we expect them to produce with the Q function.

The Q function takes as its input an agent's state and action and maps them to probable rewards. Reinforcement learning is the process of running the agent through sequences of state-action pairs, observing the rewards that result, and adapting the predictions of the Q function to those rewards until it accurately predicts the best path for the agent to take. That prediction is known as a policy.

Reinforcement learning is iterative. In its most interesting applications, it doesn't begin by knowing which rewards state-action pairs will produce. It learns those relations by running through states again and again like athletes or musicians iterate through states to improve their performance.

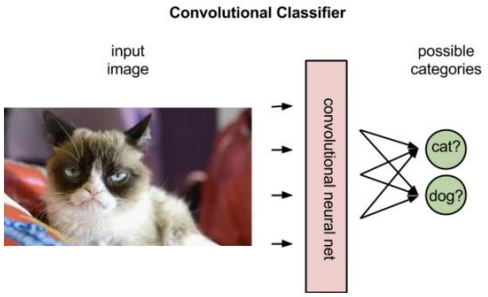
Reinforcement learning is Ground-hog Day for algorithms. And since most humans never experience their Ground-hog Day, that means reinforcement learning gives algorithms the potential to learn more, and better than humans.

In fact, that's the gist of the last several papers published by Deep-mind, since their algorithms now show super-human performance on most of the video games they've trained on.

Neural Networks and Reinforcement Learning

Where do neural networks fit in? Neural networks are the agent that learns to map state-action pairs to rewards. Like all neural networks, they use coefficients to approximate the function relating inputs to outputs, and their learning consists to finding the right coefficients, or weights, by iteratively adjusting those weights along gradients that promise less error.

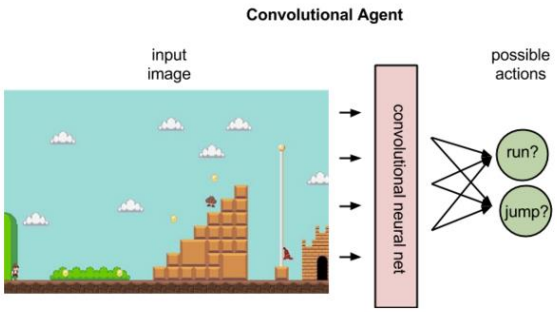
In reinforcement learning, convolutional networks can be used to recognize an agent's state; e.g. the screen that Mario is on, or the terrain before a drone. That is, they perform their typical task of image recognition.



But convolutional networks derive different interpretations from images in reinforcement learning than in supervised learning. In supervised learning, the network applies a label to an image; that is, it matches names to pixels.

In fact, it will rank the labels that best fit the image in terms of their probabilities. Shown an image of a donkey, it might decide the picture is 80% likely to be a donkey, 50% likely to be a horse, and 30% likely to be a dog.

In reinforcement learning, given an image that represents a state, a convolutional net can rank the actions possible to perform in that state; for example, it might predict that running right will return 5 points, jumping 7, and running left none.



Having assigned values to the expected rewards, the Q function simply selects the state-action pair with the highest so-called Q value.

At the beginning of reinforcement learning, the neural network coefficients may be initialized stochastically, or randomly. Using feedback from the environment, the neural net can use the difference between its expected reward and the ground-truth reward to adjust its weights and improve its interpretation of state-action pairs.

This feedback loop is analogous to the back-propagation of error in supervised learning. However, supervised learning begins with knowledge of the ground-truth labels the neural network is trying to predict. Its goal is to create a model that maps different images to their respective names.

Reinforcement learning relies on the environment to send it a scalar number in response to each new action. The rewards returned by the environment can be varied, delayed or affected by unknown variables, introducing noise to the feedback loop.

This leads us to a complete expression of the Q function, which considers not only the immediate rewards produced by an action but also the delayed rewards that may be returned many time steps deeper in the sequence.

Like human beings, the Q function is recursive. Just as calling the wetware method human() contains within it another method human(), of which we are all the fruit, calling the Q function on a given state-action pair requires us to call a nested Q function to predict the value of the next state, which in turn depends on the Q function of the state after that, and so forth.

Deep Learning

Deep learning is a subset of machine learning. Usually, when people use the term deep learning, they are referring to deep artificial neural networks, and somewhat less frequently too deep reinforcement learning

Deep artificial neural networks are a set of algorithms that have set new records in accuracy for many important problems, such as image recognition, sound recognition, recommender systems, etc. For example, deep learning is part of DeepMind's well-known AlphaGo algorithm, which beat the former world champion Lee Sedol at Go in early 2016, and the current world champion Ke Jie in early 2017. A complete explanation of neural works is here.

Deep is a technical term. It refers to the number of layers in a neural network. A shallow network has one so-called hidden layer, and a deep network has more than one. Multiple hidden layers allow deep neural networks to learn features of the data in a so-called feature hierarchy, because simple features (e.g. two pixels) recombine from one layer to the next, to form more complex features (e.g. a line). Nets with many layers pass input data (features) through more mathematical operations than nets with few layers, and are therefore more computationally intensive to train. Computational intensivity is one of the hallmarks of deep learning, and it is one reason why GPUs are in demand to train deep-learning models.

So, you could apply the same definition to deep learning that Arthur Samuel did to machine learning – a "field of study that gives computers the ability to learn without being explicitly programmed" – while adding that it tends to result in higher accuracy, require more hardware or training time, and perform exceptionally well on machine perception tasks that involved unstructured data such as blobs of pixels or text.

Implementation Techniques of AI

Regression Analysis

Capturing the change (or rate of)

In statistical modelling, regression analysis is a set of statistical processes for estimating the relationships among variables. It includes many techniques for modelling and analysing several variables when the focus is on the relationship between a dependent variable and one or more independent variables (or 'predictors'). More specifically,

regression analysis helps one understand how the typical value of the dependent variable (or 'criterion variable') changes when any one of the independent variables is varied, while the other independent variables are held fixed.

Types of Regression

- **Linear Regression:** Univariate, Bivariate & Multivariate Regression.
- **Logistic (Bounded) Regression**
- **Polynomial (Non-Linear) Regression**

Linear Regression

A sub-category of supervised learning used when the value being predicted differs to a "yes or no" label as it falls somewhere on a continuous spectrum. Regression systems could be used, for example, to answer questions of "How much?" or "How many?"

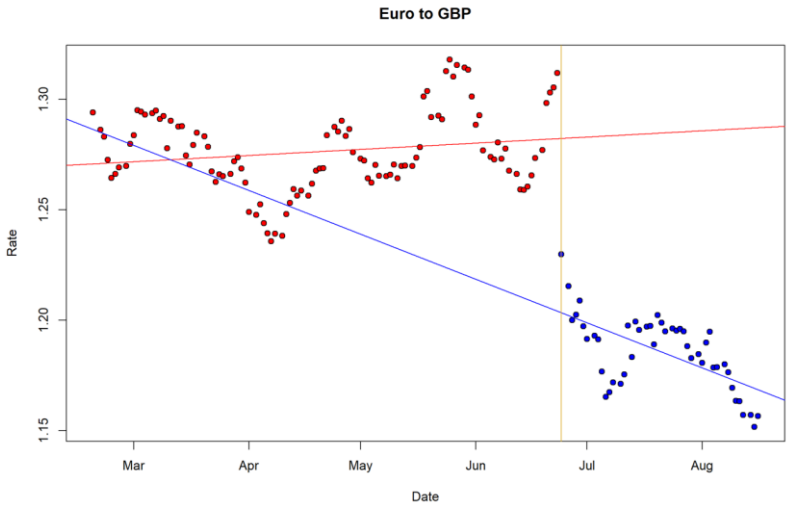
Dividing the data into a straight line or line of best fit

In statistics, linear regression is a linear approach to modelling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X. The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression. This term is distinct from the multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable.

Data mining technique that helps you know more about your data. It does not tell you the cause but relationship.

- A lot of money does not cause having a costlier house
- There is a relationship between having a lot of money and having a costlier house

It is the most popular statistic technique for data analysis to date.

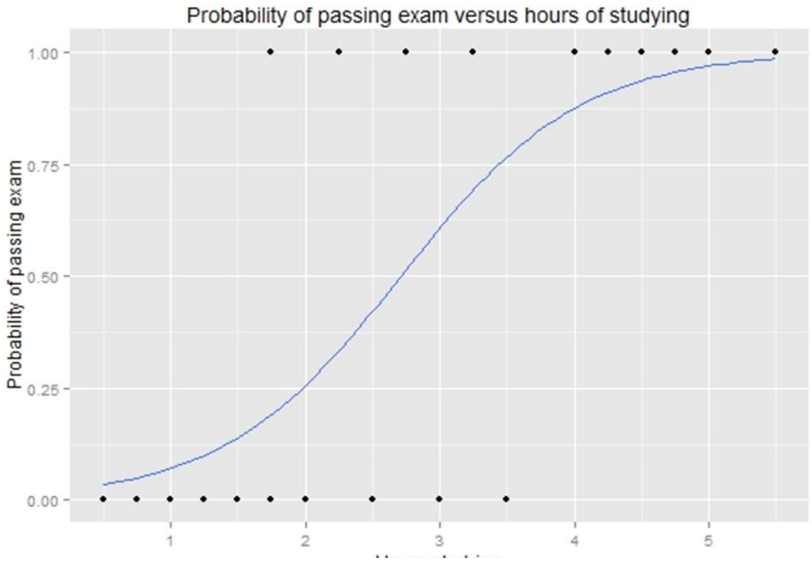


Logistic Regression

Categorical regression, Data is Plotted between 0 and 1 (100%) probability or Discrete with categorical data and uses a logarithmic sum of least squares

In statistics, logistic regression, or logit regression, or logit model is a regression model where the dependent variable (DV) is categorical. This article covers the case of a binary dependent variable—that is, where the output can take only two values, "0" and "1", which represent outcomes such as pass/fail, win/lose, alive/dead or healthy/sick. Cases, where the dependent variable has more than two outcome categories, may be analysed in multinomial logistic regression, or, if the multiple categories are ordered, in ordinal logistic regression. In the terminology of economics, logistic regression is an example of a qualitative response/discrete choice model.

Logistic regression was developed by statistician David Cox in 1958. The binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features). It allows one to say that the presence of a risk factor increases the odds of a given outcome by a specific factor.

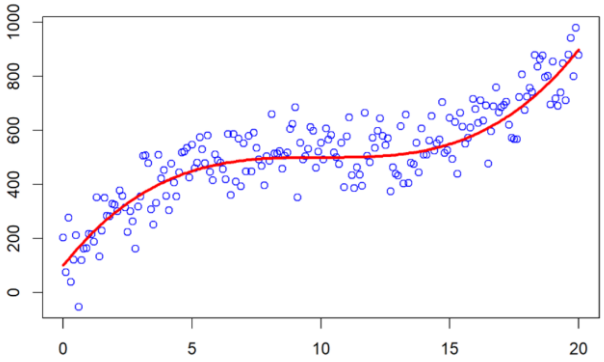


Polynomial Regression

Used to describe non-linear phenomena, e.g. Progression of an epidemic, it fits a higher order degree curve to fit your plotted data in a non-linear fashion, nth degree (high order) curves, parabolic or hyperbolic functions

In statistics, polynomial regression is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modelled as an nth degree polynomial in x. Polynomial regression fits a non-linear relationship between the value of x and the corresponding conditional mean of y, denoted E(y |x), and has been used to describe non-linear phenomena such as the growth rate of tissues, The distribution of carbon isotopes in lake sediments and the progression of disease epidemics. Although polynomial regression fits a non-linear model to the data, as a statistical estimation problem it is linear, in the sense that the regression function E(y | x) is linear in the unknown parameters that are estimated from the data. For this reason, polynomial regression is a special case of multiple linear regression.

The predictors resulting from the polynomial expansion of the "baseline" predictors are known as interactive features. Such predictors/features are also used in classification settings.



Classification

Classification is a general process related to categorization, the process in which ideas and objects are recognized, differentiated and understood. A classification system is an approach to accomplishing classification

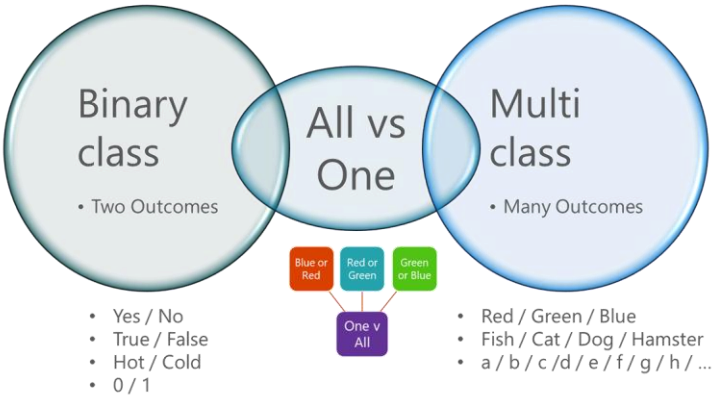
In machine learning and statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, based on a training set of data containing observations (or instances) whose category membership is known. An example would be assigning a given email into "spam" or "non-spam" classes or assigning a diagnosis to a given patient as described by observed characteristics of the patient (gender, blood pressure, presence or absence of certain symptoms, etc.). Classification is an example of pattern recognition.

A sub-category of Supervised Learning, Classification is the process of taking some sort of input and assigning a label to it. Classification systems are usually used when predictions are of a discrete, or “yes or no” nature. Example: Mapping a picture of someone to a male or female classification.

Types of Classification

- Binary-class
- Multiclass

- All vs One



Clustering

Data analysis for identifying similarities and differences among data sets so that similar ones can be clustered together.

Discover structure, for unsupervised learning

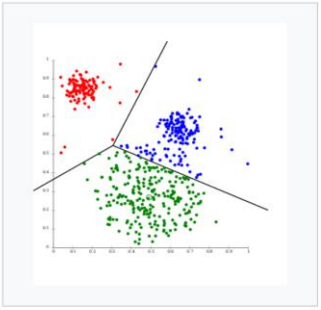
Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is the main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics.

Types of Clustering

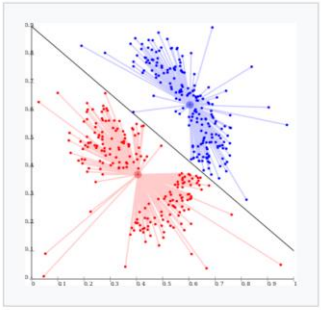
- Centroid-based clustering (K-Means)
- Connectivity-based clustering (Hierarchical Clustering)
- Distribution-based clustering
- Density-based clustering

Centroid-based Clustering

In centroid-based clustering, clusters are represented by a central vector, which may not necessarily be a member of the data set. When the number of clusters is fixed to k, k-means clustering gives a formal definition as an optimization problem: find the k cluster centres and assign the objects to the nearest cluster centre, such that the squared distances from the cluster are minimized.



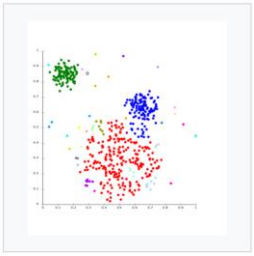
K-means separates data into Voronoi-cells, which assumes equal-sized clusters (not adequate here)



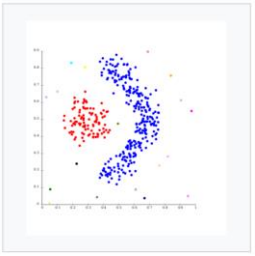
K-means cannot represent density-based clusters

Connectivity-based clustering (Hierarchical Clustering)

Connectivity-based clustering, also known as hierarchical clustering, is based on the core idea of objects being more related to nearby objects than to objects farther away. These algorithms connect "objects" to form "clusters" based on their distance. A cluster can be described largely by the maximum distance needed to connect parts of the cluster. At different distances, different clusters will form, which can be represented using a dendrogram, which explains where the common name "hierarchical clustering" comes from: these algorithms do not provide a single partitioning of the data set, but instead provide an extensive hierarchy of clusters that merge with each other at certain distances. In a dendrogram, the y-axis marks the distance at which the clusters merge, while the objects are placed along the x-axis such that the clusters don't mix.



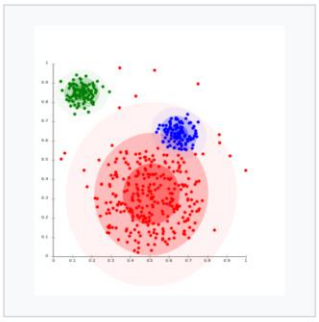
Single-linkage on Gaussian data. At 35 clusters, the biggest cluster starts fragmenting into smaller parts, while before it was still connected to the second largest due to the single-link effect.



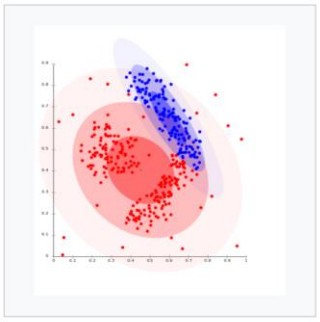
Single-linkage on density-based clusters. 20 clusters extracted, most of which contain single elements, since linkage clustering does not have a notion of "noise".

Distribution-based clustering

The clustering model most closely related to statistics is based on distribution models. Clusters can then easily be defined as objects belonging most likely to the same distribution. A convenient property of this approach is that this closely resembles the way artificial data sets are generated: by sampling random objects from a distribution.



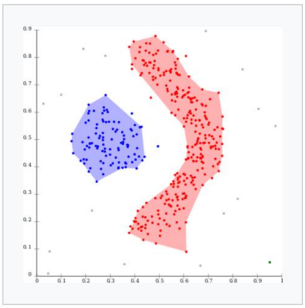
On Gaussian-distributed data, EM works well, since it uses Gaussians for modelling clusters



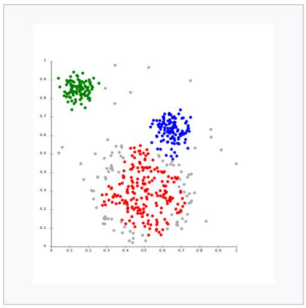
Density-based clusters cannot be modeled using Gaussian distributions

Density-based clustering (DBSCAN)

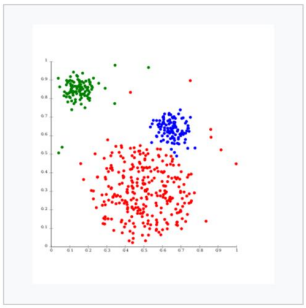
In density-based clustering, clusters are defined as areas of higher density than the remainder of the dataset. Objects in these sparse areas – that are required to separate clusters – are usually considered to be noise and border points.



Density-based clustering with DBSCAN.



DBSCAN assumes clusters of similar density, and may have problems separating nearby clusters



OPTICS is a DBSCAN variant that handles different densities much better

Chapter 04 – Theory of Data

Data or Dataset

The basics of machine learning rely on understanding the data. The data or dataset normally refers to content available in a structured or unstructured format for use in machine learning. Structured datasets have specific formats, and an unstructured dataset is normally in the form of some free-flowing text. Data can be available in various storage types or formats. In structured data, every element is known as an instance or an example or row follows a predefined structure. Data can also be categorized by size: small or medium data have a few hundred to thousands of instances, whereas big data refers to a large volume, mostly in millions or billions, that cannot be stored or accessed using common devices or fit in the memory of such devices.

Working with mean, mode, and median

The mean, median, and mode are basic ways to describe characteristics or summarize information from a dataset. When a new, large dataset is first encountered, it can be helpful to know basic information about it to direct further analysis. These values are often used in the later analysis to generate more complex measurements and conclusions. This can occur when we use the mean of a dataset to calculate the standard deviation, which we will demonstrate in the Standard deviation section of this chapter.

Calculating the mean

The term mean, also called the average, is computed by adding values in a list and then dividing the sum by the number of values. This technique is useful for determining the general trend for a set of numbers. It can also be used to fill in missing data elements.

Calculating the median

The mean can be misleading if the dataset contains many outlying values or is otherwise skewed. When this happens, the mode and median can be useful. The term median is the value in the middle of a range of values. For an odd number of values, this is easy to compute. For an even number of values, the median is calculated as the average of the middle two values.

Calculating the mode

The term mode is used for the most frequently occurring value in a dataset. This can be thought of as the most popular result, or the highest bar in a histogram. It can be a useful piece of information when conducting statistical analysis, but it can be more complicated to calculate than it first appears.

Standard deviation

Standard deviation is a measurement of how values are spread around the mean. A high deviation means that there is a widespread, whereas a low deviation means that the values are more tightly grouped around the mean. This measurement can be misleading if there is not a single focus point or there are numerous outliers.

- Full Population
- Sample Subset

Sample Size Determination

Sample size determination involves identifying the quantity of data required to conduct the accurate statistical analysis. When working with large datasets it is not always necessary to use the entire set. We use sample size determination to ensure we choose a sample small enough to manipulate and analyse easily, but large enough to represent our population of data accurately. It is not uncommon to use a subset of data to train a model and another subset is used to test the model. This can be helpful for verifying accuracy and reliability of data. Some

common consequences for a poorly determined sample size include false-positive results, false-negative results, identifying statistical significance where none exists or suggesting a lack of significance where it is present. Many tools exist online for determining appropriate sample sizes, each with varying levels of complexity. One simple example is available at:

- https://www.surveymonkey.com/mp/sample-size-calculator

Features, attributes, variables, or dimensions

In structured datasets, as mentioned before, there are predefined elements with their own semantics and data type, which are known variously as features, attributes, metrics, indicators, variables, or dimensions.

Big Data

Big data is data sets that are so voluminous and complex that traditional data processing application software is inadequate to deal with them. Big data challenges include capturing data, data storage, data analysis, search, sharing, transfer, visualization, querying, updating and information privacy. There are three dimensions to big data known as **Volume, Variety and Velocity**.

Data types

The features defined earlier need some form of typing in many machine learning algorithms or techniques.

The most commonly used data types are as follows:

- **Categorical or Nominal** This indicates well-defined categories or values present in the dataset. For example, eye colour—black, blue, brown, green, grey; document content type—text, image, video.
- **Continuous or numeric** This indicates a numeric nature of the data field. For example, a person's weight measured by a bathroom scale, the temperature reading from a sensor, or the monthly balance in dollars on a credit card account.
- **Ordinal** This denotes data that can be ordered in some way. For example, garment size—small, medium, large; boxing weight classes: heavyweight, light heavyweight, middleweight, lightweight, and bantamweight.
- **Categorical variables** are also known as discrete or qualitative variables. Categorical variables can be further categorized as either nominal, ordinal or dichotomous.
- **Nominal variables** are variables that have two or more categories, but which do not have an intrinsic order. For example, a real estate agent could classify their types of property into distinct categories such as houses, condos, co-ops or bungalows. So "type of property" is a nominal variable with 4 categories called houses, condos, co-ops and bungalows. Of note, the different categories of a nominal variable can also be referred to as groups or levels of the nominal variable. Another example of a

nominal variable would be classifying where people live in the USA by state. In this case, there will be many more levels of the nominal variable (50 in fact).

- **Dichotomous variables** are nominal variables which have only two categories or levels. For example, if we were looking at gender, we would most probably categorize somebody as either "male" or "female". This is an example of a dichotomous variable (and a nominal variable). Another example might be if we asked a person if they owned a mobile phone. Here, we may categorise mobile phone ownership as either "Yes" or "No". In the real estate agent example, if the type of property had been classified as either residential or commercial then "type of property" would be a dichotomous variable.
- **Ordinal variables** are variables that have two or more categories just like nominal variables only the categories can also be ordered or ranked. So, if you asked someone if they liked the policies of the Democratic Party and they could answer either "Not very much", "They are OK" or "Yes, a lot" then you have an ordinal variable. Why? Because you have 3 categories, namely "Not very much", "They are OK" and "Yes, a lot" and you can rank them from the most positive (Yes, a lot), to the middle response (They are OK), to the least positive (Not very much). However, whilst we can rank the levels, we cannot place a "value" to them; we cannot say that "They are OK" is twice as positive as "Not very much" for example.

Continuous variables are also known as quantitative variables. Continuous variables can be further categorized as either interval or ratio variables.

- **Interval variables** are variables for which their central characteristic is that they can be measured along a continuum and they have a numerical value (for example, the temperature measured in degrees Celsius or Fahrenheit). So, the difference between 20C and 30C is the same as 30C to 40C. However, the temperature measured in degrees Celsius or Fahrenheit is NOT a ratio variable.
- **Ratio variables** are interval variables, but with the added condition that 0 (zero) of the measurement indicates that there is none of that variable. So, the temperature measured in degrees Celsius or Fahrenheit is not a ratio variable because 0C does not mean there is no temperature. However, the temperature measured in Kelvin is a ratio variable as 0 Kelvin (often called absolute zero) indicates that there is no temperature whatsoever. Other examples of ratio variables include height, mass, distance and many more. The name "ratio" reflects the fact that you can use the ratio of measurements. So, for example, ten metres is twice the distance of 5 metres.

Property	Nominal	Ordinal	Interval	Ratio
Frequency of distribution	✓	✓	✓	✓
Mode and median		✓	✓	✓
Order of values is known		✓	✓	✓
Can quantify difference between each value			✓	✓
Can add or subtract values			✓	✓
Can multiply and divide values				✓
Has true zero				✓

Types of Variables

All experiments examine variable(s). A variable is not only something that we measure but also something that we can manipulate and something we can control for. To understand the characteristics of variables and how we use them in research, this guide is divided into three main sections. First, we illustrate the role of dependent and independent variables. Second, we discuss the difference between experimental and non-experimental research. Finally, we explain how variables can be characterised as either categorical or continuous.

Dependent and Independent Variables

An independent variable sometimes called an experimental or predictor variable, is a variable that is being manipulated in an experiment to observe the effect on a dependent variable, sometimes called an outcome variable.

Imagine that a tutor asks 100 students to complete a maths test. The tutor wants to know why some students perform better than others. Whilst the tutor does not know the answer to this, she thinks that it might be because of two reasons: (1) some students spend more time revising for their test; and (2) some students are naturally more intelligent than others. As such, the tutor decides to investigate the effect of revision time and intelligence on the test performance of the 100 students. The dependent and independent variables for the study are:

- **Dependent Variable** Test Mark (measured from 0 to 100)
- **Independent Variables** Revision time (measured in hours) Intelligence (measured using IQ score)

The dependent variable is simply that, a variable that is dependent on an independent variable(s). For example, in our case, the test mark that a student achieves is dependent on revision time and intelligence. Whilst revision time and intelligence (the independent variables) may (or may not) cause a change in the test mark (the dependent variable), the reverse is implausible; in other words, whilst the number of hours a student spends revising and the higher a student's IQ score may (or may not) change the test mark that a student achieves, a change in a student's test mark has no bearing on whether a student revises more or is more intelligent (this simply doesn't make sense).

Therefore, the aim of the tutor's investigation is to examine whether these independent variables – revision time and IQ – result in a change in the dependent variable, the students' test scores. However, it is also worth noting that whilst this is the main aim of the experiment, the tutor may also be interested to know if the independent variables – revision time and IQ – are also connected in some way.

Experimental and Non-Experimental Research

- **Experimental research** in experimental research, the aim is to manipulate an independent variable(s) and then examine the effect that this change has on a dependent variable(s). Since it is possible to manipulate the independent variable(s), experimental research has the advantage of enabling a researcher to identify a cause and effect between variables. For example, take our example of 100 students completing a maths exam where the dependent variable was the exam mark (measured from 0 to 100), and the independent variables were revision time (measured in hours) and intelligence (measured using IQ score). Here, it would be possible to use an experimental design and manipulate the revision time of the students. The tutor could divide the students into two groups, each made up of 50 students. In "group one", the tutor could ask the students not to do any revision. Alternately, "group two" could be asked to do 20 hours of revision in the two weeks prior to the test. The tutor could then compare the marks that the students achieved.
- **Non-experimental research** in non-experimental research, the researcher does not manipulate the independent variable(s). This is not to say that it is impossible to do so, but it will either be impractical or unethical to do so. For example, a researcher may be interested in the effect of illegal, recreational drug use (the independent variable(s)) on certain types of behaviour (the dependent variable(s)). However, whilst possible, it would be unethical to ask individuals to take illegal drugs to study what effect this had on certain behaviours. As such, a researcher could ask both drug and non-drug users to complete a questionnaire that had been constructed to indicate the extent to which they exhibited certain behaviours. Whilst it is not possible to identify the cause and effect between the variables, we can still examine the association or relationship between them. In addition to understanding the difference between dependent and independent variables, and experimental and non-experimental research, it is also important to understand the different characteristics amongst variables.

Ambiguities in classifying a type of variable

In some cases, the measurement scale for data is ordinal, but the variable is treated as continuous. For example, a Likert scale that contains five values – strongly agree, agree, neither agree nor disagree, disagree, and strongly disagree – is ordinal. However, where a Likert scale contains seven or more value – strongly agree, moderately agree, agree, neither agree nor disagree, disagree, moderately disagree, and strongly disagree – the underlying scale is sometimes treated as continuous (although where you should do this is a cause of great dispute).

It is worth noting that how we categorise variables is somewhat of a choice. Whilst we categorised gender as a dichotomous variable (you are either female or male), social scientists may disagree with this, arguing that gender is a more complex variable involving more than two distinctions, but also including measurement levels like genderqueer, intersex and transgender. At the same time, some researchers would argue that a Like scale, even with seven values, should never be treated as a continuous variable.

Types of Data Relationships

A Data Scientist will find relationships, correlations and anomalies (outliers), Data professionals will let the algorithms do the job of the data scientists.

- **Influencer Relationships** Strong (Wage is strongly influenced by education), Weak (Wage is weakly influenced by marriage status), No-Relation

- **Impactor Relationships** Direct, Indirect, Positive, Negative & Neutral

Data Exploration

There are no shortcuts for data exploration. If you are in a state of mind, that machine learning can sail you away from every data storm, trust me, it won't. After some point in time, you'll realize that you are struggling with improving model's accuracy. In such situation, data exploration techniques will come to your rescue.

Steps of Data Exploration and Preparation

Remember the quality of your inputs decides the quality of your output. So, once you have got your business hypothesis ready, it makes sense to spend a lot of time and efforts here. With my personal estimate, data exploration, cleaning and preparation can take up to 70% of your total project time.

Below are the steps involved to understand, clean and prepare your data for building your predictive model:

- Variable Identification
- Univariate Analysis
- Bi-variate Analysis
- Missing values treatment
- Outlier treatment
- Variable transformation
- Variable creation

Finally, we will need to iterate over steps 4 – 7 multiple times before we come up with our refined model.

Let's now study each stage in detail:

Variable Identification

First, identify Predictor (Input) and Target (output) variables. Next, identify the data type and category of the variables.

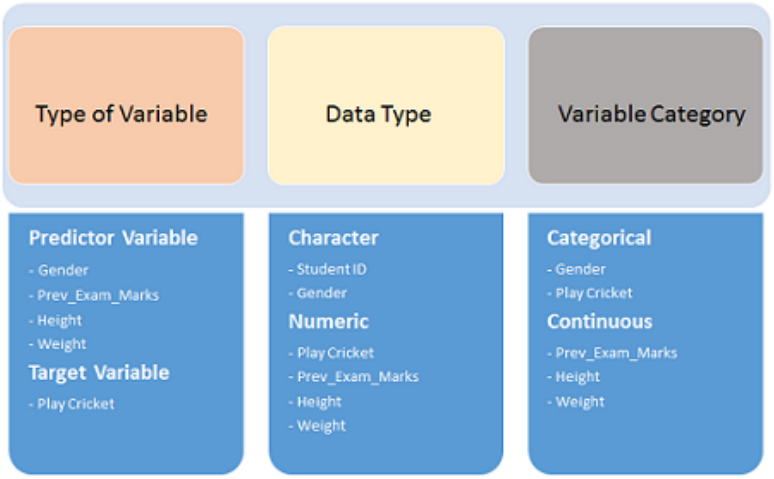
Let's understand this step more clearly by taking an example.

Student_ID	Gender	Prev_Exam_Marks	Height (cm)	Weight Caregory (kgs)	Play Cricket
S001	M	65	178	61	1
S002	F	75	174	56	0
S003	M	45	163	62	1
S004	M	57	175	70	0
S005	F	59	162	67	0

Example Suppose, we want to predict, whether the students will play cricket or not (refer below dataset). Here you need to identify predictor variables, target variable, the data type of variables and category of variables. Business Analytics, Data exploration below, the variables have been defined in a different category:

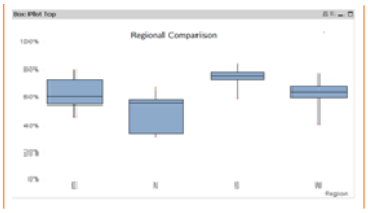
Univariate Analysis

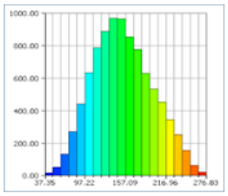
At this stage, we explore variables one by one. Method to perform univariate analysis will depend on whether the variable type is categorical or continuous. Let's look at these methods and statistical measures for categorical and continuous variables individually:



- **Continuous Variables** In case of continuous variables, we need to understand the central tendency and spread of the variable. These are measured using various statistical metrics visualization methods as shown below:

Central Tendency	Measure of Dispersion	Visualization Methods
Mean	Range	Histogram
Median	Quartile	Box Plot
Mode	IQR	
Min	Variance	
Max	Standard Deviation	
	Skewness and Kurtosis	





Note: Univariate analysis is also used to highlight missing and outlier values. In the upcoming part of this series, we will look at methods to handle missing and outlier values.

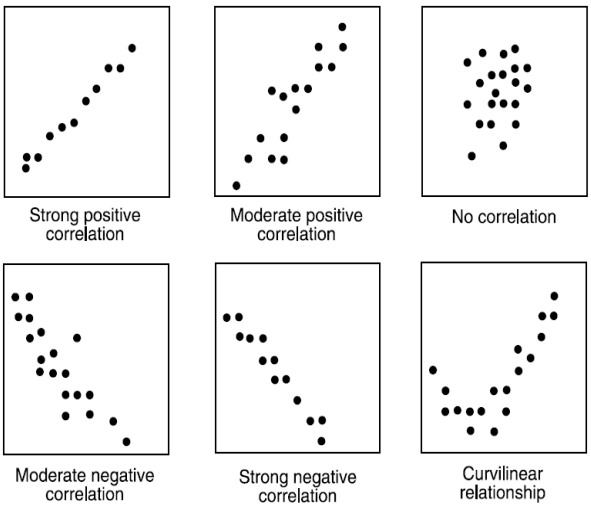
- **Categorical Variables** For categorical variables, we'll use a frequency table to understand the distribution of each category. We can also read as a percentage of values under each category. It can be measured using two metrics, Count and Count% against each category. A bar-chart can be used for visualization.

Bi-variate Analysis

Bi-variate Analysis finds out the relationship between two variables. Here, we look for association and disassociation between variables at a pre-defined significance level. We can perform bi-variate analysis for any combination of categorical and continuous variables. The combination can be Categorical & Categorical, Categorical & Continuous and Continuous & Continuous. Different methods are used to tackle these combinations during the analysis process.

Let's understand the possible combinations in detail:

- **Continuous & Continuous** While doing a bi-variate analysis between two continuous variables, we should look at scatter plot. It is a nifty way to find out the relationship between two variables. The pattern of scatter plot indicates the relationship between variables. The relationship can be linear or non-linear.



Scatter plot shows the relationship between two variables but does not indicates the strength of relationship amongst them. To find the strength of the relationship, we use Correlation. Correlation varies between -1 and +1.

- -1: perfect negative linear correlation
- +1: perfect positive linear correlation and
- 0: No correlation

Correlation can be derived using following formula: **Correlation = Covariance(X,Y) / SQRT(Var(X)* Var(Y))**

Various tools have function or functionality to identify a correlation between variables. In Excel, function CORREL() is used to return the correlation between two variables and SAS uses procedure PROC CORR to identify the correlation. These function returns Pearson Correlation value to identify the relationship between two variables:

X	65	72	78	65	72	70	65	68
Y	72	69	79	69	84	75	60	73

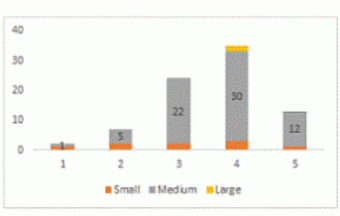
Metrics	Formula	Value
Co-Variance (X,Y)	=COVAR(E6:L6,E7:L7)	18.77
Variance (X)	=VAR.P(E6:L6)	18.48
Variance (Y)	=VAR.P(E7:L7)	45.23
Correlation	=G10/SQRT(G11*G12)	0.65

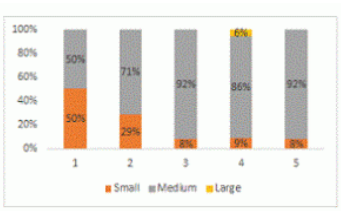
In above example, we have a good positive relationship(0.65) between two variables X and Y.

- **Categorical & Categorical** To find the relationship between two categorical variables, we can use following methods:
- **Two-way table** We can start analysing the relationship by creating a two-way table of the count and count%. The rows represent the category of one variable and the columns represent the categories of the other variable. We show count or count% of observations available in each combination of row and column categories.
- **Stacked Column Chart** This method is more of a visual form of a Two-way table.

Frequency / Row Pct	Product Category 1	2	3	4	5	Total
Small	1 11.11	2 22.22	2 22.22	3 33.33	1 11.11	9
Medium	1 1.43	5 7.14	22 31.43	30 42.86	12 17.14	70
Large	0 0.00	0 0.00	0 0.00	2 100.00	0 0.00	2
Total	2	7	24	35	13	81

Frequency Missing = 77





- **Chi-Square Test** This test is used to derive the statistical significance of the relationship between the variables. Also, it tests whether the evidence in the sample is strong enough to generalize that the relationship for a larger population as well. Chi-square is based on the difference between the expected and observed frequencies in one or more categories in the two-way table. It returns probability for the computed chi-square distribution with the degree of freedom.
    - Probability of 0: It indicates that both categorical variables are dependent
    - The probability of 1: It shows that both variables are independent.
    - Probability less than 0.05: It indicates that the relationship between the variables is significant at 95% confidence. The chi-square test statistic for a test of independence of two categorical variables is found by:

$$X^2 = \sum (O - E)^2 / E$$

    - where *O* represents the observed frequency. *E* is the expected frequency under the null hypothesis and computed by:

$$E = \frac{row\ total \times column\ total}{sample\ size}$$

From the previous two-way table, the expected count for product category 1 to be of small size is 0.22. It is derived by taking the row total for Size (9) times the column total for the Product category (2) then dividing by the sample size (81). This is procedure is conducted for each cell. Statistical Measures used to analyse the power of relationship are:

Cramer's V for Nominal Categorical Variable

Mantel-Haenszed Chi-Square for the ordinal categorical variable.

Different data science language and tools have specific methods to perform chi-square test. In SAS, we can use **Chisq** as an option with **Proc freq** to perform this test.

- **Categorical & Continuous** While exploring the relationship between categorical and continuous variables, we can draw box plots for each level of categorical variables. If levels are small, it will not show the statistical significance. To look at the statistical significance we can perform Z-test, T-test or ANOVA.
- **Z-Test/ T-Test** Either test assesses whether the mean of two groups is statistically different from each other or not.

$$z = \frac{\left|\bar{x}_1 - \bar{x}_2\right|}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

If the probability of Z is small, then the difference between two averages is more significant. The T-test is very similar to Z-test, but it is used when many observations for both categories is less than 30.

- **ANOVA** It assesses whether the average of more than two groups is statistically different.

Example Suppose, we want to test the effect of five different exercises. For this, we recruit 20 men and assign one type of exercise to 4 men (5 groups). Their weights are recorded after a few weeks. We need to find out whether the effect of these exercises on them is significantly different or not. This can be done by comparing the weights of the 5 groups of 4 men each.

Till here, we have understood the first three stages of Data Exploration, Variable Identification, Uni-Variate and Bi-Variate analysis. We also looked at various statistical and visual methods to identify the relationship between variables.

Now, we will look at the methods of Missing values Treatment. More importantly, we will also look at why missing values occur in our data and why treating them is necessary.

Data preparation:

Cleaning up data to the point where you can work with it is a huge amount of work. If you're trying to reconcile a lot of sources of data that you don't control, it can take 80% of your time.

While there are tools to help automate the data cleaning process and reduce the time it takes, the task of automation is made difficult by the fact that the process is as much art as science, and no two data preparation tasks are the same.

"It's an absolute myth that you can send an algorithm over raw data and have insights pop up." *Jeffrey Heer, professor of computer science at the University of Washington*

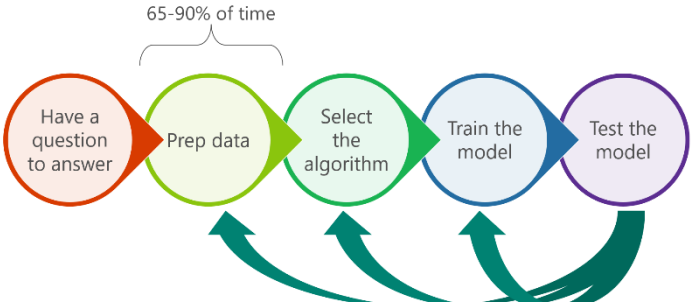
Machine learning isn't Kaggle competitions. A Kaggle competition typically presents a nice, clean, regularized data set to the competitors, but this isn't representative of the real-world process of making predictions from data.

Clean Missing Data

Missing data can be a not so trivial problem when analysing a dataset and accounting for it is usually not so straightforward either.

If the amount of missing data is very small relatively to the size of the dataset, then leaving out the few samples with missing features may be the best strategy in order not to bias the analysis, however leaving out available

datapoints deprives the data of some amount of information and depending on the situation you face, you may want to look for other fixes before wiping out potentially useful datapoints from your dataset.



While some quick fixes such as mean-substitution may be fine in some cases, such simple approaches usually introduce bias into the data, for instance, applying mean substitution leaves the mean unchanged (which is desirable) but decreases variance, which may be undesirable.

"Cleaning data" is dangerous ground and it must be done with a lot of context in mind. While there are tools that can help, I have yet to see an automated process that I would fully trust. In general, this is the part of data science that requires the most expert attention. For example, one scenario is that you find out if the mean of a feature is an outlier, if so, you might consider then to replace the outliers and missing data. The best practice for outliers or missing data is to first account for them, and not blindly erase them. You should try to understand why some data are extreme, and determine, for example, whether these data are the result of a data capture error, or simply occur normally and will recur in new data you will use with your model in the future. What you will do about the extreme data will vary depending on the answers you determine.

The Mice package in R, for example, helps you imputing missing values with plausible data values. These plausible values are drawn from a distribution specifically designed for each missing data point.

For example, before blindly imputing missing value as mean, you could create a script that checks for specific scenarios. Let's illustrate through the following example scenario: If less than 5% of column values are null or missing that it concludes that they are missing completely by random and recommends using mean, if the mean is not an outlier else using Mice, it imputes 5 plausible values and overlays the distribution of predicted value over the distribution of the columns and picks the one that's closest. If more than 5% and less than 25% of column values are missing, then it tries to find the domain the missing values may belong to and imputes values using Mice but within the domains. If more than 25% of column values is missing, then it recommends dropping the feature or review the data ingestion process. And similar assessment for outliers as well. And or multivariate assessment, like if x1, X2 and X3 features are missing across i observations, should the observation be removed?

Missing Value Treatment

Why is missing values treatment required?

Missing data in the training data set can reduce the power/fit of a model or can lead to a biased model because we have not analysed the behaviour and relationship with other variables correctly. It can lead to wrong prediction or classification.

Name	Weight	Gender	Play Cricket/ Not
Mr. Amit	58	M	Y
Mr. Anil	61	M	Y
Miss Swati	58	F	N
Miss Richa	55		Y
Mr. Steve	55	M	N
Miss Reena	64	F	Y
Miss Rashmi	57		Y
Mr. Kunal	57	M	N

Gender	#Students	#Play Cricket	%Play Cricket
F	2	1	50%
M	4	2	50%
Missing	2	2	100%

Name	Weight	Gender	Play Cricket/ Not
Mr. Amit	58	M	Y
Mr. Anil	61	M	Y
Miss Swati	58	F	N
Miss Richa	55	F	Y
Mr. Steve	55	M	N
Miss Reena	64	F	Y
Miss Rashmi	57	F	Y
Mr. Kunal	57	M	N

Gender	#Students	#Play Cricket	%Play Cricket
F	4	3	75%
M	4	2	50%

Notice the missing values in the image shown above: In the left scenario, we have not treated missing values. The inference from this dataset is that the chances of playing cricket by males are higher than females. On the other hand, if you look at the second table, which shows data after treatment of missing values (based on gender), we can see that females have higher chances of playing cricket compared to males.

Why my data has missing values?

We looked at the importance of treatment of missing values in a dataset. Now, let's identify the reasons for the occurrence of these missing values. They may occur at two stages:

- **Data Extraction** It is possible that there are problems with the extraction process. In such cases, we should double-check for correct data with data guardians. Some hashing procedures can also be used to make sure data extraction is correct. Errors at data extraction stage are typically easy to find and can be corrected easily as well.
- **Data collection** These errors occur at the time of data collection and are harder to correct. They can be categorized into four types:
- **Missing completely at random** This is a case when the probability of missing variable is same for all observations. For example, respondents of data collection process decide that they will declare they're earning after tossing a fair coin. If a head occurs, respondent declares his / her earnings & vice versa. Here each observation has an equal chance of missing value.
- **Missing at random** This is a case when the variable is missing at random and missing ratio varies for different values/level of other input variables. For example, we are collecting data for age and female has higher missing value compare to male.
- **Missing that depends on unobserved predictors** This is a case when the missing values are not random and are related to the unobserved input variable. For example: In a medical study, if a diagnostic causes discomfort, then there is a higher chance of drop out from the study. This missing value is not at random unless we have included "discomfort" as an input variable for all patients.
- **Missing that depends on the missing value itself** This is a case when the probability of missing value is directly correlated with missing value itself. For example, People with higher or lower income are likely to provide non-response to their earning.

Which are the methods to treat missing values?

- **Deletion** It is of two types: List Wise Deletion and PairWise Deletion.

In listwise deletion, we delete observations where any of the variables are missing. Simplicity is one of the major advantages of this method, but this method reduces the power of model because it reduces the sample size.

In pairwise deletion, we perform analysis with all cases in which the variables of interest are present. The advantage of this method is, it keeps as many cases available for analysis. One of the disadvantages of this method, it uses different sample size for different variables.

List wise deletion

Gender	Manpower	Sales
M	25	343
F	.	280
M	33	332
M	.	272
F	25	.
M	29	326
	26	259
M	32	297

Pair wise deletion

Gender	Manpower	Sales
M	25	343
F	.	280
M	33	332
M	.	272
F	25	.
M	29	326
	26	259
M	32	297

Deletion methods are used when the nature of missing data is "Missing completely at random" else non-random missing values can bias the model output.

- **Mean/ Mode/ Median Imputation** is a method to fill in the missing values with estimated ones. The objective is to employ known relationships that can be identified in the valid values of the data set to assist in estimating the missing values. Mean / Mode / Median imputation is one of the most frequently used methods. It consists of replacing the missing data for a given attribute by the mean or median (quantitative attribute) or mode (qualitative attribute) of all known values of that variable. It can be of two types:-
- **Generalized Imputation** in this case, we calculate the mean or median for all non-missing values of that variable then replace missing value with mean or median. Like in above table, variable "Manpower" is missing so we take an average of all non-missing values of "Manpower" (28.33) and then replace missing value with it.
- **Similar case Imputation** in this case, we calculate the average for gender "Male" (29.75) and "Female" (25) individually of non-missing values then replace the missing value based on gender. For "Male", we will replace missing values of manpower with 29.75 and for "Female" with 25.
- **The prediction Model** Prediction model is one of the sophisticated methods for handling missing data. Here, we create a predictive model to estimate values that will substitute the missing data. In this case, we divide our dataset into two sets: One set with no missing values for the variable and another one with missing values. First data set become training data set of the model while second data set with missing values are test data set and variable with missing values are treated as the target variable. Next, we create a model to predict target variable based on other attributes of the training data set and populate missing values of test data set. We can use regression, ANOVA, Logistic regression and various modelling technique to perform this. There are two drawbacks to this approach:

  The model estimated values are usually more well-behaved than the true values

  If there are no relationships with attributes in the data set and the attribute with missing values, then the model will not be precise for estimating missing values.
- **KNN Imputation** In this method of imputation, the missing values of an attribute is imputed using the given number of attributes that are most like the attribute whose values are missing. The similarity of two attributes is determined using a distance function. It is also known to have certain advantage & disadvantages.

Advantages

- a k-nearest neighbour can predict both qualitative & quantitative attributes
- Creation of predictive model for each attribute with missing data is not required
- Attributes with multiple missing values can be easily treated
- Correlation structure of the data is taken into consideration

Disadvantages

- KNN algorithm is very time-consuming in analysing large database. It searches through all the dataset looking for the most similar instances.
- Choice of k-value is very critical. A higher value of k would include attributes which are significantly different from what we need whereas a lower value of k implies missing out of significant attributes.
- After dealing with missing values, the next task is to deal with outliers. Often, we tend to neglect outliers while building models. This is a discouraging practice. Outliers tend to make your data skewed and reduces accuracy. Let's learn more about outlier treatment.

Select Columns in Data Set

Creates a view of a dataset that includes or excludes specific columns

Example Delete a column whose data is highly correlated with data in another column

Partition and Sample

Divides or extracts a subset of a data using sampling techniques

Splitting and sampling datasets are both important tasks in machine learning. For example, it is a common practice to divide data into training and testing sets, so that you can evaluate a model on a holdout data set. Sampling is also increasingly important in the era of big data, to ensure that there is a fair distribution of classes in your training data, and that you are not processing more data than is needed, it lets you reduce the size of a dataset while maintaining the same ratio of values.

The Partition and Sample module in ML studio, for example, supports several important machine learning scenarios:

- Dividing your data into multiple subsections of the same size. The goal might be to use the partitions for cross-validation or to assign cases to random groups.
- Separating data into groups and then working with data from a specific group. You might need to randomly assign cases to different groups, and then modify the features that are associated with only one group. You do this in the Partition and Sample module by splitting data into folds and then choosing a fold on which to perform further operations.
- Sampling. You can extract a percentage of the data, apply random sampling, or choose a column to use for balancing the dataset and perform stratified sampling on its values.

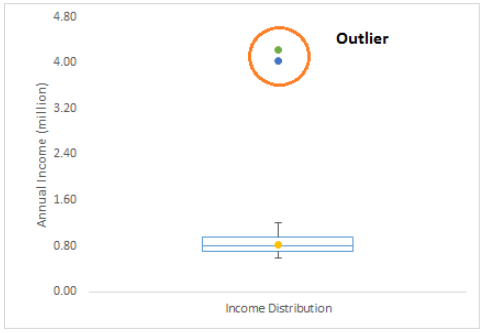
- Creating a smaller dataset for testing. If you a have a lot of data, you might want to use only the first n rows while setting up the experiment, and then switch to using the full dataset when you build your model. You can also use sampling to create s smaller dataset for use in development.

Techniques of Outlier Detection and Treatment

What is an Outlier?

The outlier is a commonly used terminology by analysts and data scientists as it needs close attention else it can result in wildly wrong estimations. Simply speaking, Outlier is an observation that appears far away and diverges from an overall pattern in a sample.

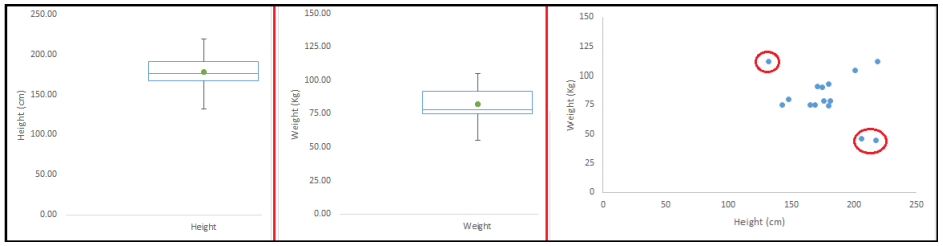
Let's take an example, we do customer profiling and find out that the average annual income of customers is $0.8 million. But, there are two customers having an annual income of $4 and $4.2 million. These two customers annual income is much higher than rest of the population. These two observations will be Outliers.



What are the types of Outliers?

An outlier can be of two types: Univariate and Multivariate. Above, we have discussed the example of a univariate outlier. These outliers can be found when we look at the distribution of a single variable. Multi-variate outliers are outliers in an n-dimensional space. To find them, you must look at distributions in multi-dimensions.

Let us understand this as an example. Let us say we are understanding the relationship between height and weight. Below, we have univariate and bivariate distribution for Height, Weight. Look at the box plot. We do not have any outlier (above and below 1.5*IQR, most common method). Now, look at the scatter plot. Here, we have two values below and one above the average in a specific segment of weight and height.



What causes Outliers?

Whenever we come across outliers, the ideal way to tackle them is to find out the reason for having these outliers. The method to deal with them would then depend on the reason for their occurrence. Causes of outliers can be classified into two broad categories:

- Artificial (Error) / Non-natural
- Natural.

Let's understand various types of outliers in more detail:

- **Data Entry Errors** Human errors such as errors caused during data collection, recording, or entry can cause outliers in data. For example, the Annual income of a customer is $100,000. Accidentally, the data entry operator puts an additional zero in the figure. Now the income becomes $1,000,000 which is 10 times higher. Evidently, this will be the outlier value when compared with rest of the population.
- **Measurement Error** It is the most common source of outliers. This is caused when the measurement instrument used turns out to be faulty. For example, there are 10 weighing machines. 9 of them are correct, 1 is faulty. Weight measured by people on the faulty machine will be higher / lower than the rest of people in the group. The weights measured on the faulty machine can lead to outliers.
- **Experimental Error** Another cause of outliers is experimental error. For example: In a 100m sprint of 7 runners, one runner missed out on concentrating on the 'Go' call which caused him to start late. Hence, this caused the runner's runtime to be more than other runners. His total run time can be an outlier.
- **Intentional Outlier** This is commonly found in self-reported measures that involve sensitive data. For example, Teens would typically under-report the amount of alcohol that they consume. Only a fraction of them would report actual value. Here actual values might look like outliers because rest of the teens are under-reporting the consumption.
- **Data Processing Error** Whenever we perform data mining, we extract data from multiple sources. It is possible that some manipulation or extraction errors may lead to outliers in the dataset.
- **Sampling Error** for instance, we must measure the height of athletes. By mistake, we include a few basketball players in the sample. This inclusion is likely to cause outliers in the dataset.
- **Natural Outlier** When an outlier is not artificial (due to error), it is a natural outlier. For instance: In my last assignment with one of the renowned insurance company, I noticed that the performance of top 50 financial advisors was far higher than rest of the population. Surprisingly, it was not due to any error. Hence, whenever we perform any data mining activity with advisors, we used to treat this segment separately.

What is the impact of Outliers on a dataset?

Outliers can drastically change the results of the data analysis and statistical modelling. There are numerous unfavourable impacts of outliers in the dataset:

- It increases the error variance and reduces the power of statistical tests
- If the outliers are non-randomly distributed, they can decrease normality
- They can bias or influence estimates that may be of substantive interest
- They can also impact the basic assumption of Regression, ANOVA and other statistical model assumptions

To understand the impact deeply, let's take an example to check what happens to a data set with and without outliers in the data set.

Example

Without Outlier	With Outlier
4, 4, 5, 5, 5, 5, 6, 6, 6, 7, 7	4, 4, 5, 5, 5, 5, 6, 6, 6, 7, 7,300
Mean = 5.45	Mean = 30.00
Median = 5.00	Median = 5.50
Mode = 5.00	Mode = 5.00
Standard Deviation = 1.04	Standard Deviation = 85.03

As you can see, a dataset with outliers has significantly different mean and standard deviation. In the first scenario, we will say that average is 5.45. But with the outlier, average soars to 30. This would change the estimate completely.

How to detect Outliers?

Most commonly used the method to detect outliers is visualization. We use various visualization methods, like Box-plot, Histogram, Scatter Plot (above, we have used box plot and scatter plot for visualization). Some analysts also various thumb rules to detect outliers. Some of them are:

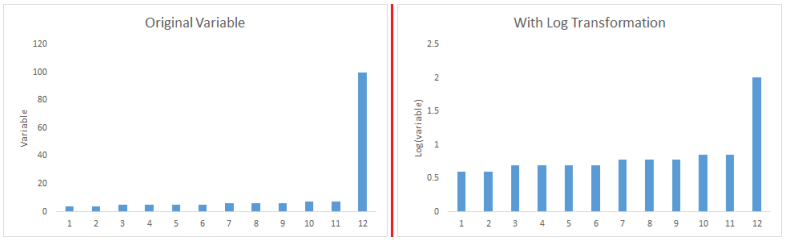
- Any value, which is beyond the range of -1.5 x IQR to 1.5 x IQR
- Use capping methods. Any value which out of range of 5th and 95th percentile can be considered an outlier
- Data points, three or more standard deviation away from mean are considered outlier
- Outlier detection is merely a special case of the examination of data for influential data points and it also depends on the business understanding
- Bivariate and multivariate outliers are typically measured using either an index of influence or leverage or distance. Popular indices such as Mahalanobis' distance and Cook's D are frequently used to detect outliers.
- We can use PROC Univariate, PROC SGPLOT. To identify outliers and influential observation, we also look at statistical measures like STUDENT, COOKD, RSTUDENT and others.

How to remove Outliers?

Most of the ways to deal with outliers are like the methods of missing values like deleting observations, transforming them, binning them, treat them as a separate group, imputing values and other statistical methods. Here, we will discuss the common techniques used to deal with outliers:

- **Deleting observations,** we delete outlier values if it is due to data entry error, data processing error or outlier observations are very small in numbers. We can also use trimming at both ends to remove outliers.

- **Transforming and binning values** Transforming variables can also eliminate outliers. Natural log of a value reduces the variation caused by extreme values. Binning is also a form of variable transformation. Decision Tree algorithm allows dealing with outliers well due to binning of the variable. We can also use the process



of assigning weights to different observations.

- **Imputing** Like imputation of missing values, we can also impute outliers. We can use mean, median, mode imputation methods. Before imputing values, we should analyse if it is a natural outlier or artificial. If it is artificial, we can go with imputing values. We can also use a statistical model to predict values of outlier observation and after that, we can impute it with predicted values.
- **Treat separately** If there are a significant number of outliers, we should treat them separately in the statistical model. One of the approaches is to treat both groups as two different groups and build an individual model for both groups and then combine the output.

Till here, we have learnt about steps of data exploration, missing value treatment and techniques of outlier detection and treatment. These 3 stages will make your raw data better in terms of information availability and accuracy. Let's now proceed to the final stage of data exploration. It is Feature Engineering.

The Art of Feature Engineering

What is Feature Engineering?

This exercising of bringing out information from data is known as feature engineering

Feature engineering is the science (and art) of extracting more information from existing data. You are not adding any new data here, but you are making the data you already have more useful.

For example, let's say you are trying to predict footfall in a shopping mall based on dates. If you try and use the dates directly, you may not be able to extract meaningful insights from the data. This is because the footfall is less affected by the day of the month than it is by the day of the week. Now this information about the day of the week is implicit in your data. You need to bring it out to make your model better.

What is the process of Feature Engineering?

You perform feature engineering once you have completed the first 5 steps in data exploration – Variable Identification, Univariate, Bivariate Analysis, Missing Values Imputation and Outliers Treatment. Feature engineering itself can be divided into 2 steps:

- Variable Transformation.
- Variable / Feature creation.

These two techniques are vital in data exploration and have a remarkable impact on the power of prediction. Let's understand each of this step in more details.

What is Variable Transformation?

In data modelling, transformation refers to the replacement of a variable by a function. For instance, replacing a variable x by the square/cube root or logarithm x is a transformation. In other words, the transformation is a process that changes the distribution or relationship of a variable with others.

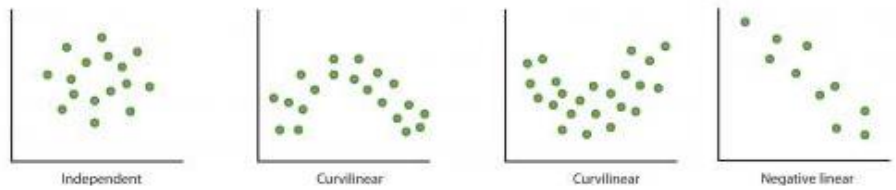
Let's look at the situations when the variable transformation is used.

When should we use Variable Transformation?

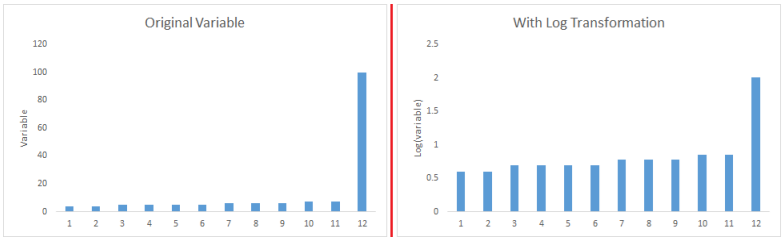
Below are the situations where variable transformation is requisite:

When we want to change the scale of a variable or standardize the values of a variable for better understanding. While this transformation is a must if you have data in different scales, this transformation does not change the shape of the variable distribution

When we can transform complex non-linear relationships into linear relationships. The existence of a linear relationship between variables is easier to comprehend compared to a non-linear or curved relation. Transformation helps us to convert a non-linear relation into a linear relation. Scatter plot can be used to find the relationship between two continuous variables. These transformations also improve the prediction. Log transformation is one of the commonly used transformation technique used in these situations.



Symmetric distribution is preferred over skewed distribution as it is easier to interpret and generate inferences. Some modelling techniques require a normal distribution of variables. So, whenever we have a skewed distribution, we can use transformations which reduce skewness. For right-skewed distribution, we take square/cube root or logarithm of variable and for left skewed, we take square/cube or exponential of variables.



Variable Transformation is also done from an implementation point of view (Human involvement). Let's understand it more clearly. In one of my project on employee performance, I found that age has a direct correlation with the performance of the employee i.e. higher the age, better the performance. From an implementation standpoint, launching age-based programme might present implementation challenge. However, categorizing the sales agents in three age group buckets of <30 years, 30-45 years and >45 and then formulating three different strategies for each group is a judicious approach. This categorization technique is known as Binning of Variables.

What are the common methods of Variable Transformation?

There are various methods used to transform variables. As discussed, some of them include square root, cube root, logarithmic, binning, reciprocal and many others. Let's look at these methods in detail by highlighting the pros and cons of these transformation methods.

- **Logarithm** Log of a variable is a common transformation method used to change the shape of the distribution of the variable on a distribution plot. It is generally used for reducing right skewness of variables. Though, it can't be applied to zero or negative values as well.
- **Square / Cube root** the square and cube root of a variable has a sound effect on variable distribution. However, it is not as significant as logarithmic transformation. Cube root has its own advantage. It can be applied to negative values including zero. The square root can be applied to positive values including zero.
- **Binning** it is used for categorizing variables. It is performed on original values, percentile or frequency. The decision of categorization technique is based on business understanding. For example, we can categorize income in three categories, namely: High, Average and Low. We can also perform co-variate binning which depends on the value of more than one variables.

What is Feature / Variable Creation & its Benefits?

Feature / Variable creation is a process to generate a new variables / features based on existing variable(s). For example, say, we have a date(dd-mm-yy) as an input variable in a data set. We can generate new variables like a day, month, year, week, weekday that may have a better relationship with target variable. This step is used to highlight the hidden relationship in a variable:

Emp_Code	Gender	Date	New_Day	New_Month	New_Year
A001	Male	21-Sep-11	21	9	2011
A002	Female	27-Feb-13	27	2	2013
A003	Female	14-Nov-12	14	11	2012
A004	Male	07-Apr-13	7	4	2013
A005	Female	21-Jan-11	21	1	2011
A006	Male	26-Apr-13	26	4	2013
A007	Male	15-Mar-12	15	3	2012

There are various techniques to create new features. Let's look at the some of the commonly used methods:

- **Creating derived variables** This refers to creating new variables from existing variable(s) using a set of functions or different methods. Let's look at it through "Titanic – Kaggle competition". In this data set, variable age has missing values. To predict missing values, we used the salutation (Master, Mr, Miss, Mrs) of the name as a new variable. How do we decide which variable to create? Honestly, this depends on a business understanding of the analyst, his curiosity and the set of the hypothesis he might have about the problem. Methods such as taking the log of variables, binning variables and other methods of variable transformation can also be used to create new variables.

- **Creating dummy variables** One of the most common applications of the dummy variable is to convert a categorical variable into numerical variables. Dummy variables are also called Indicator Variables. It is useful to take a categorical variable as a predictor in statistical models. A categorical variable can take values 0 and 1. Let's take a variable 'gender'. We can produce two variables, namely, "Var_Male" with values 1 (Male) and 0 (No male) and "Var_Female" with values 1 (Female) and 0 (No Female). We can also create dummy variables for more than two classes of categorical variables with n or n-1 dummy variables.

Emp_Code	Gender	Var_Male	Var_Female
A001	Male	1	0
A002	Female	0	1
A003	Female	0	1
A004	Male	1	0
A005	Female	0	1
A006	Male	1	0
A007	Male	1	0

Chapter 05 – Data Extract, Transformation and Loading (ETL)

Acquiring Data for an Application

Data Acquisition

Data may be stored in a variety of formats. Popular formats for text data include HTML, Comma Separated Values (CSV), JavaScript Object Notation (JSON), and XML. Image and audio data are stored in many formats. However, it is frequently necessary to convert one data format into another format, typically plain text.

Data is acquired using techniques such as processing live streams, downloading compressed files, and through screen scraping, where the information on a web page is extracted. Web crawling is a technique where a program examines a series of web pages, moving from one page to another, acquiring the data that it needs.

The importance and process of Cleaning Data

Data Wrangling, Reshaping, or Munging

Real-world data is frequently dirty and unstructured and must be reworked before it is usable. Data may contain errors, have duplicate entries, exist in the wrong format, or be inconsistent. The process of addressing these types of issues is called data cleaning. Data cleaning is also referred to as **data wrangling, massaging, reshaping, or munging**.

Data merging where data from multiple sources is combined is often considered to be a data cleaning activity. We need to clean data because any analysis based on inaccurate data can produce misleading results. We want to ensure that the data we work with is quality data.

Data imputation refers to the process of identifying and replacing missing data in each dataset. In almost any substantial case of data analysis, missing data will be an issue, and it needs to be addressed before data can be properly analysed. Trying to process data that is missing information is a lot like trying to understand a conversation where every occasion, a word is dropped. Sometimes we can understand what is intended. In other situations, we may be completely lost as to what is trying to be conveyed. Among statistical analysts, there exist differences of opinion as to how missing data should be handled but the most common approaches involve replacing missing data with a reasonable estimate or with an empty or null value. To prevent skewing and misalignment of data, many statisticians advocate for replacing missing data with values representative of the average or expected value for that dataset. The methodology for determining a representative value and assigning it to a location within the data will vary depending upon the data and we cannot illustrate every example in this chapter. However, for example, if a dataset contained a list of temperatures across a range of dates, and one date was missing a temperature, that date can be assigned a temperature that was the average of the temperatures within the dataset.

Data Quality involves:

- **Validity** Ensuring that the data possesses the correct form or structure
- **Accuracy** The values within the data are truly representative of the dataset
- **Completeness** There are no missing elements
- **Consistency** Changes to data are in sync

- **Uniformity** The same units of measurement are used

Data validation is an important part of data science. Before we can analyse and manipulate data, we need to verify that the data is of the type expected. We have organized our code into simple methods designed to accomplish very basic validation tasks. The code within these methods can be adapted into existing applications.

There are several techniques and tools used to clean data. We will examine the following approaches:

- Handling different types of data
- Cleaning and manipulating text data
- Filling in missing data
- Validating data

Visualizing Data to Enhance Understanding

Data Visualisation

The human mind is often good at seeing patterns, trends, and outliers in visual representations. A large amount of data present in many data science problems can be analysed using visualization techniques. Visualization is appropriate for a wide range of audiences, ranging from analysts to upper-level management, to clientele.

Visualization is an important step in data analysis because it allows us to conceive of large datasets in practical and meaningful ways. We can look at small datasets of values and perhaps draw conclusions from the patterns we see, but this is an overwhelming and unreliable process. Using visualization tools helps us identify potential problems or unexpected data results, as well as construct meaningful interpretations of good data.

One example of the usefulness of data visualization comes with the presence of outliers. Visualizing data allows us to quickly see data results significantly outside of our expectations, and we can choose how to modify the data to build a clean and usable dataset. This process allows us to see errors quickly and deal with them before they become a problem later. Additionally, visualization allows us to easily classify information and help analysts organize their inquiries in a manner best suited to their dataset.

Various Visualisation Models are Bar Charts, Pie Charts, Time Series Graphs, Index Charts, Histograms, Scatter Plots, Area Charts, Donut Charts, Bubble Charts.

Visualisation Goals

Each type of visual expression lends itself to different types of data and data analysis purposes. One common purpose of data analysis is data classification. This involves determining which subset within a dataset a data value belongs to. This process may occur early in the data analysis process because breaking data apart into manageable and related pieces simplifies the analysis process. Often, classification is not the end goal but rather an important intermediary step before further analysis can be undertaken.

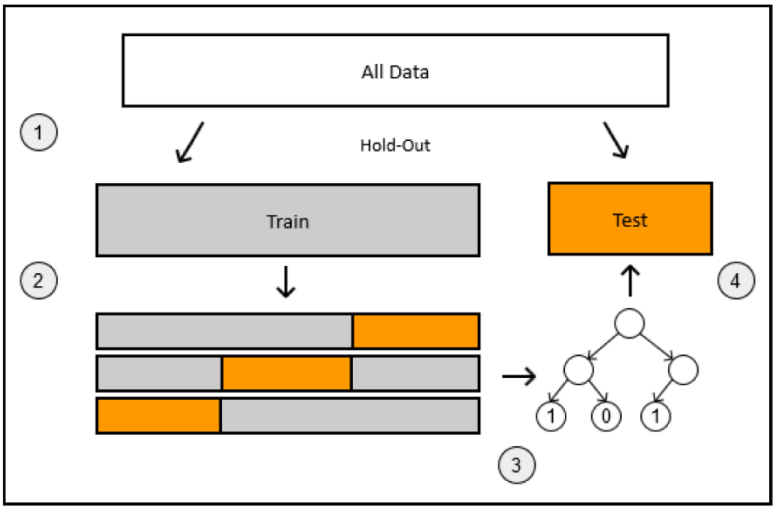
Training, validation, and Testing

When doing cross-validation, there's still a danger of overfitting. Since we try a lot of different experiments on the same validation set, we might accidentally pick the model which just happened to do well on the validation set--but it may, later, fail to generalize to unseen data.

The solution to this problem is to hold out a test set at the very beginning and do not touch it at all until we select what we think is the best model. And we use it only for evaluating the final model on it.

So how do we select the best model? What we can do is to do cross-validation on the remaining train data. It can hold out or k-fold cross-validation. In general, you should prefer doing k-fold cross-validation because it also gives you the spread of performance, and you may use it in for model selection as well.

The following diagram illustrates the process:



According to the diagram, a typical data science workflow should be the following:

- 0: Select some metric for validation, for example, accuracy or AUC
- 1: Split all the data into train and test sets
- 2: Split the training data further and hold out a validation dataset or split it into k folds
- 3: Use the validation data for model selection and parameter optimization
- 4: Select the best model according to the validation set and evaluate it against the holdout test set

Evaluation

As with classification, we also need to evaluate the results of our models. There are some metrics that help to do that and select the best model. Let's go over the two most popular ones: **Mean Squared Error (MSE)** and **Mean Absolute Error (MAE)**.

Chapter 06 – Theory of Deep Learning

Introduction

Deep Learning allows computational models composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection, and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large datasets by using the back-propagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about dramatic improvements in processing images, video, speech and audio, while recurrent nets have shone on sequential data such as text and speech. Representation learning is a set of methods that allow a machine to be fed with raw data and to automatically discover the representations needed for detection or classification. Deep learning methods are representation learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level.

Deep learning is a subset of machine learning. Usually, when people use the term deep learning, they are referring to deep artificial neural networks, and somewhat less frequently towards deep reinforcement learning.

Deep artificial neural networks are a set of algorithms that have set new records in accuracy for many important problems, such as image recognition, sound recognition, recommender systems, etc. For example, deep learning is part of DeepMind's well-known AlphaGo algorithm, which beat the former world champion Lee Sedol at Go in early 2016, and the current world champion Ke Jie in early 2017. A complete explanation of neural works is here.

Deep is a technical term. It refers to the number of layers in a neural network. A shallow network has one so-called hidden layer, and a deep network has more than one. Multiple hidden layers allow deep neural networks to learn features of the data in a so-called feature hierarchy, because simple features (e.g. two pixels) recombine from one layer to the next, to form more complex features (e.g. a line). Nets with many layers pass input data (features) through more mathematical operations than nets with few layers, and are therefore more computationally intensive to train. Computational insensitivity is one of the hallmarks of deep learning, and it is one reason why GPUs are in demand to train deep-learning models.

So, you could apply the same definition to deep learning that Arthur Samuel did to machine learning – a "field of study that gives computers the ability to learn without being explicitly programmed" – while adding that it tends to result in higher accuracy, require more hardware or training time, and perform exceptionally well on machine perception tasks that involved unstructured data such as blobs of pixels or text.

Neural Network Definition

Neural networks are a set of algorithms, modelled loosely after the human brain, that is designed to recognize patterns. They interpret sensory data through a kind of machine perception, labelling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.

Neural networks help us cluster and classify. You can think of them as a clustering and classification layer on top of data you store and manage. They help to group unlabelled data according to similarities among the example inputs, and they classify data when they have a labelled dataset to train on. (To be more precise, neural networks extract features that are fed to other algorithms for clustering and classification; so, you can think of deep neural networks as components of larger machine-learning applications involving algorithms for reinforcement learning, classification and regression.)

What kind of problems does deep learning solve, and more importantly, can it solve yours? To know the answer, you need to ask yourself a few questions: What outcomes do I care about? Those outcomes are labels that could be applied to data: for example, spam or not_spam in an email filter, good_guy or bad_guy in fraud detection, angry_customer or happy_customer in customer relationship management. Then ask: Do I have the data to accompany those labels? That is, can I find labelled data, or can I create a labelled dataset (with a service like Mechanical Turk or Crowd-flower) where spam has been labelled as spam, to teach an algorithm, the correlation between labels and inputs?

Single-Layer Neural Network

A single-layer neural network in deep learning is a net composed of an input layer, which is a visible layer, and a hidden output layer.

The single-layer network's goal, or objective function, is to learn features by minimizing reconstruction entropy.

This allows it to auto-learn features of the input, which leads to finding good correlations and higher accuracy in identifying discriminatory features. From there, a multilayer network leverages this to accurately classify the data. This is the pre-training step.

Each single-layer network has the following attributes:

- **Hidden Bias** The bias for the output
- **Visible Bias** The bias for the input
- **Weight Matrix** The weights for the machine

Types of single-layer neural networks

- Restricted Boltzmann Machines
- Continuous Restricted Boltzmann Machine
- De-noising Auto Encoder

Training a single-layer network

Train a network by joining the input vector to the input layer. Distort the input with some Gaussian noise. This noise function will vary depending on the network. Then minimize reconstruction entropy through pre-training until the network learns the best features for reconstructing the input data.

Learning rate

A typical learning-rate value is between 0.001 and 0.1. The learning rate, or step rate, is the rate at which a function steps within a search space. Smaller learning rates mean higher training times but may lead to more precise results.

Momentum

Momentum is an extra factor in determining how fast an optimization algorithm converges.

L2 regularization constant

L2 is the lambda discussed in the equation here.

Multi-Layer Neural Networks (Creating Deep-Learning)

A multilayer neural network is a stacked representation of a single-layer neural network. The input layer is tacked onto the first-layer neural network and a feed-forward network. Each subsequent layer after the input layer uses the output of the previous layer as its input.

A multilayer network will accept the same kinds of inputs as a single-layer network. The multilayer network parameters are also typically the same as their single-layer network counterparts.

The output layer for a multilayer network is typically a logistic regression classifier, which sorts results into zeros and ones. This is a discriminatory layer used for classification of input features based on the final hidden layer of the deep network.

A multilayer network is composed of the following kinds of layers:

- K single layer networks
- A soft-max regression output layer.

Types of multilayer networks

- Stacked De-Noising Auto-Encoders
- Deep Belief Networks

Parameters

Below are the parameters what you need to think about when training a network.

Learning rate

The learning rate, or step rate, is the rate at which a function steps through the search space. The typical value of the learning rate is between 0.001 and 0.1. Smaller steps mean longer training times but can lead to more precise results.

Momentum

Momentum is an additional factor in determining how fast an optimization algorithm converges on the optimum point.

If you want to speed up the training, increase the momentum. But you should know that higher speeds can lower a model's accuracy.

To dig deeper, momentum is a variable between zero and one that is applied as a factor to the derivative of the rate of change of the matrix. It affects the change rate of the weights over time.

L2 regularization constant

L2 is the lambda discussed in this equation here.

Pre-training step

For pre-training – i.e. learning the features via reconstruction at each layer – a layer is trained and then the output is piped to the next layer.

Fine-tuning step

Finally, the logistic regression output layer is trained, and then backpropagation happens for each layer.

Questions to Ask When Applying Deep Learning

We can't answer these questions for you because the responses will be specific to the problem you seek to solve. But we hope this will serve as a useful checklist to clarify how you initially approach your choice of algorithms and tools:

- **Is my problem supervised or unsupervised?** If supervised, is it a classification or regression problem? Supervised learning has a teacher. That teacher takes the form of a training set that establishes correlations between two types of data, your input and your output. You may want to apply labels to images, for example. In this classification problem, your input is raw pixels, and your output is the name of whatever's in the picture. In a regression example, you might teach a neural net how to predict continuous values such as housing price based on an input like square-footage. Unsupervised learning, on the other hand, can help you detect similarities and anomalies simply by analysing unlabelled data. Unsupervised learning has no teacher; it can be applied to use cases such as image search and fraud detection.
- **If supervised, how many labels am I dealing with?** The more labels you need to apply accurately, the more computationally intensive your problem will be. ImageNet has a training set with about 1000 classes; the Iris dataset has just 3.
- **What's my batch size?** A batch is a bundle of examples, or instances from your dataset, such as a group of images. In training, all the instances of a batch are passed through the net, and the error resulting from the net's guesses is averaged from all instances in the batch and then used to update the weights of the model. Larger batches mean you wait longer between each update, or learning the step. Small batches mean the net learns less about the underlying dataset with each batch. Batch sizes of 1000 can work well on some problems if you have a lot of data and you're looking for a smart default to start with.
- **How many features am I dealing with?** The more features you have, the more memory you'll need. With images, the features of the first layer equal the number of pixels in the image. So MNIST's 28*28-pixel images have 784 features. In medical diagnostics, you may be looking at 14 megapixels.
- **Another way to ask that same question is:** What is my architecture? Resnet, the Microsoft Research net that won the most recent ImageNet competition, had 150 layers. All other things being equal, the more layers you add, the more features you must deal with, the more memory you need. A dense layer in a multilayer perceptron (MLP) is a lot more feature intensive than a convolutional layer. People use convolutional nets with subsampling precisely because they get to aggressively prune the features they're computing.
- **How am I going to tune my neural net?** Tuning neural nets is still something of a dark art for a lot of people. There are a couple of ways to go about it. You can tune empirically, looking at the f1 score of your

net and then adjusting the hyperparameters. You can tune with some degree of automation using tools like hyperparameter optimization example here. And finally, you can rely on heuristics like a GUI, which will show you exactly how quickly your error is decreasing, and what your activation distribution looks like.

- **How much data is sufficient to train my model?**
    - How do I go about finding that data?
- **Hardware: Will I be using GPUs, CPUs or both?** Am I going to rely on a single-system GPU or a distributed system? A lot of research is being conducted on 1-4 GPUs. Enterprise solutions usually require more and must work with large CPU clusters as well.
- **What's my data pipeline?** How do I plan to extract, transform and load the data (ETL)? Is it in a SQL DB? Is it on a Hadoop cluster? Is it local or in the cloud?
- **How will I featurise that data?** Even though deep learning extracts features automatically, you can lighten the computational load and speed training with different forms of feature engineering, especially when the features are sparse.
- **What kind of non-linearity, loss function and weight initialization will I use?** The non-linearity is the activation function tied to each layer of your deep net. It might be sigmoid, rectified linear, or something else. Specific non-linearities often go hand in hand with specific loss functions.
- **What is the simplest architecture I can use for this problem?** Not everyone is willing or able to apply Resnet to image classification.
- **Where will my net be trained and where will the model be deployed?** What does it need to integrate with? Most people don't think about these questions until they have a working prototype, at which point they find themselves forced to rewrite their net with more scalable tools. You should be asking whether you'll eventually need to use Spark, AWS or Hadoop, among other platforms.

A Few Concrete Examples

Deep learning maps inputs to outputs. It finds correlations. It is known as a "universal approximator", because it can learn to approximate the function f(x) = y between any input x and any output y, assuming they are related through correlation or causation at all. In the process of learning, a neural network finds the right f, or the correct manner of transforming x into y, whether that be f(x) = 3x + 12 or f(x) = 9x – 0.1. Here are a few examples of what deep learning can do.

Classification

All classification tasks depend upon labelled datasets; that is, humans must transfer their knowledge to the dataset for a neural to learn the correlation between labels and data. This is known as supervised learning.

- Detect faces, identify people in images, recognize facial expressions (angry, joyful)
- Identify objects in images (stop signs, pedestrians, lane markers…)
- Recognise gestures in video
- Detect voices, identify speakers, transcribe speech to text, recognize sentiments in speech (voice)
- Classify text as spam (in emails), or fraudulent (in insurance claims); recognize sentiment in text (customer feedback)

Any labels that humans can generate, any outcomes you care about and which correlate to data, can be used to train a neural network.

Clustering

Clustering or grouping is the detection of similarities. Deep learning does not require labels to detect similarities. Learning without labels is called unsupervised learning. Unlabelled data is most of the data in the world. One law of machine learning is: the more data an algorithm can train on, the more accurate it will be. Therefore, unsupervised learning has the potential to produce highly accurate models.

- Search: Comparing documents, images or sounds to surface similar items.
- Anomaly detection: The flip-side of detecting similarities is detecting anomalies or unusual behaviour. In many cases, unusual behaviour. correlates highly with things you want to detect and prevent, such as fraud.

Predictive Analytics

With classification, deep learning can establish correlations between, say, pixels in an image and the name of a person. You might call this a static prediction. By the same token, exposed to enough of the right data, deep learning can establish correlations between present events and future events. The future event is like the label in a sense. Deep learning doesn't necessarily care about time or the fact that something hasn't happened yet. Given a time series, deep learning may read a string of number and predict the number most likely to occur next.

- Hardware breakdowns (data centres, manufacturing, transport)
- Health breakdowns (strokes, heart attacks based on vital stats and data from wearables)
- Customer churn (predicting the likelihood that a customer will leave, based on web activity and metadata)
- Employee turnover (ditto, but for employees)

The better we can predict, the better we can prevent and pre-empt. As you can see, with neural networks, we're moving towards a world of fewer surprises. Not zero surprises, just marginally fewer.

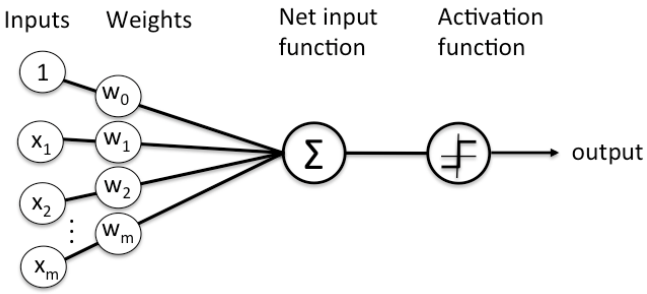
With that brief overview of deep learning use cases, let's look at what neural nets are made of.

Neural Network Elements

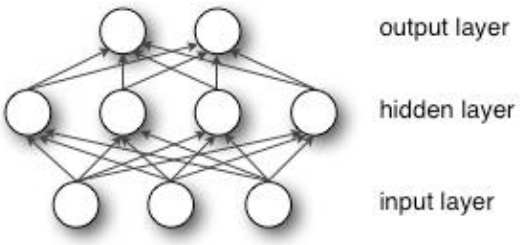
Deep learning is the name we use for "stacked neural networks"; that is, networks composed of several layers.

The layers are made of nodes. A node is just a place where computation happens, loosely patterned on a neuron in the human brain, which fires when it encounters sufficient stimuli. A node combines input from the data with a set of coefficients, or weights, that either amplify or dampen that input, thereby assigning significance to inputs for the task the algorithm is trying to learn. (For example, which input is most helpful is classifying data without error?) These input-weight products are summed, and the sum is passed through a node's so-called activation function, to determine whether and to what extent that signal progresses further through the network to affect the ultimate outcome, say, an act of classification.

Here's a diagram of what one node might look like.



A node layer is a row of those neuron-like switches that turn on or off as the input is fed through the net. Each layer's output is simultaneously the subsequent layer's input, starting from an initial input layer receiving your



data.

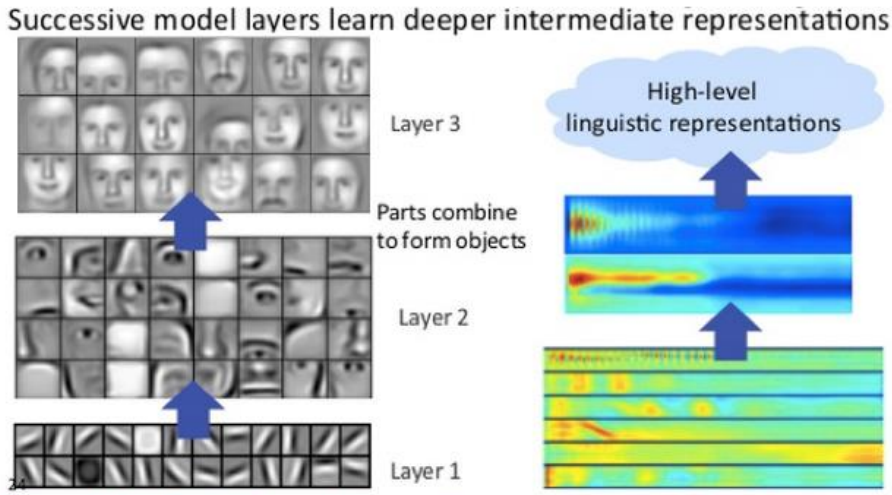
Pairing adjustable weights with input features is how we assign significance to those features about how the network classifies and clusters input.

Key Concepts of Deep Neural Networks

Deep-learning networks are distinguished from the more commonplace single-hidden-layer neural networks by their depth; that is, the number of node layers through which data passes through a multi-step process of pattern recognition.

Traditional machine learning relies on shallow nets, composed of one input and one output layer, and at most one hidden layer in between. More than three layers (including input and output) qualifies as "deep" learning. So deep is a strictly defined, technical term that means more than one hidden layer.

In deep-learning networks, each layer of nodes trains on a distinct set of features based on the previous layer's output. The further you advance into the neural net, the more complex the features your nodes can recognize since they aggregate and recombine features from the previous layer.



This is known as feature hierarchy, and it is a hierarchy of increasing complexity and abstraction. It makes deep-learning networks capable of handling very large, high-dimensional data sets with billions of parameters that pass through non-linear functions.

Above all, these nets can discover latent structures within unlabelled, unstructured data, which is most of the data in the world. Another word for unstructured data is raw media; i.e. pictures, texts, video and audio recordings. Therefore, one of the problems deep learning solves best is in processing and clustering the world's raw, unlabelled media, discerning similarities and anomalies in data that no human has organized in a relational database or ever put a name to.

For example, deep learning can take a million images, and cluster them according to their similarities: cats in one corner, icebreakers in another, and in a third all the photos of your grandmother. This is the basis of so-called smart photo albums.

Now apply that same idea to other data types: Deep learning might cluster raw text such as emails or news articles. Emails full of angry complaints might cluster in one corner of the vector space, while satisfied customers, or spam-bot messages, might cluster in others. This is the basis of various messaging filters and can be used in customer-relationship management (CRM). The same applies to voice messages. With time series, data might cluster around normal/healthy behaviour. and anomalous/dangerous behaviour. If the time series data is being generated by a smartphone, it will provide insight into users' health and habits; if it is being generated by an auto-part, it might be used to prevent catastrophic breakdowns.

Deep-learning networks perform automatic feature extraction without human intervention, unlike most traditional machine-learning algorithms. Given that feature extraction is a task that can take teams of data scientists years to accomplish, deep learning is a way to circumvent the choke-point of limited experts. It augments the powers of small data science teams, which by their nature do not scale.

When training on unlabelled data, each node layer in a deep network learns features automatically by repeatedly trying to reconstruct the input from which it draws its samples, attempting to minimize the difference between the network's guesses and the probability distribution of the input data itself. Restricted Boltzmann machines, for examples, create so-called reconstructions in this manner.

In the process, these networks learn to recognize correlations between certain relevant features and optimal results – they draw connections between feature signals and what those features represent, whether it be a full reconstruction, or with labelled data.

A deep-learning network trained on labelled data can then be applied to unstructured data, giving it access to much more input than machine-learning nets. This is a recipe for higher performance: the more data a net can train on, the more accurate it is likely to be. (Bad algorithms trained on lots of data can outperform good algorithms trained on very little.) Deep learning's ability to process and learn from huge quantities of unlabelled data give it a distinct advantage over previous algorithms.

Deep-learning networks end in an output layer: a logistic, or soft-max, a classifier that assigns a likelihood to an outcome or label. We call that predictive, but it is predictive in a broad sense. Given raw data in the form of an image, a deep-learning network may decide, for example, that the input data is 90 percent likely to represent a person.

Example: Feed-Forward Networks

Our goal in using a neural net is to arrive at the point of least error as fast as possible. We are running a race, and the race is around a track, so we pass the same points repeatedly in a loop. The starting line for the race is the state in which our weights are initialized, and the finish line is the state of those parameters when they can produce accurate classifications and predictions.

The race itself involves many steps, and each of those steps resembles the steps before and after. Just like a runner, we will engage in a repetitive act over and over to arrive at the finish. Each step of a neural network involves a guess, an error measurement and a slight update in its weights, an incremental adjustment to the coefficients.

A collection of weights, whether they are in their start or end state, is also called a model, because it is an attempt to model data's relationship to ground-truth labels, to grasp the data's structure. Models normally start out bad and end up less bad, changing over time as the neural network updates its parameters.

This is because a neural network is born of ignorance. It does not know which weights and biases will translate the input best to make the correct guesses. It must start out with a guess, and then try to make better guesses sequentially as it learns from its mistakes. (You can think of a neural network as a miniature enactment of the scientific method, testing hypotheses and trying again – only it is the scientific method with a blindfold on.)

Here is a simple explanation of what happens during learning with a feed-forward neural network, the simplest architecture to explain.

Input enters the network. The coefficients, or weights, map that input to a set of guesses the network makes at the end.

input * weight = guess

Weighted input results in a guess about what that input is. The neural then takes its guess and compares it to a ground-truth about the data, effectively asking an expert "Did I get this right?".

ground truth – guess = error

The difference between the network's guess and the ground truth is its error. The network measures that error, and walks the error back over its model, adjusting weights to the extent that they contributed to the error.

error * weight's contribution to error = adjustment

The three pseudo-mathematical formulas above account for the three key functions of neural networks: scoring input, calculating loss and applying an update to the model – to begin the three-step process over again. A neural network is a corrective feedback loop, rewarding weights that support its correct guesses, and punishing weights that lead it to error.

Let's linger on the first step above.

Multiple Linear Regression

Despite their biologically inspired name, artificial neural networks are nothing more than math and code, like any other machine-learning algorithm. In fact, anyone who understands linear regression, one of the first methods you learn in statistics, can understand how neural networks. In its simplest form, linear regression is expressed as

$$Y\_hat = bX + a$$

Where Y_hat is the estimated output, X is the input, b is the slope and an is the intercept of a line on the vertical axis of a two-dimensional graph. (To make this more concrete: X could be radiation exposure and Y could be the cancer risk; X could be daily push-ups and Y could be the total weight you can bench-press; X the amount of fertilizer and Y the size of the crop.) You can imagine that every time you add a unit to X, the dependent variable Y increases proportionally, no matter how far along you are on the X-axis. That simple relation between two variables moving up or down together is a starting point.

The next step is to imagine multiple linear regression, where you have many input variables producing an output variable. It's typically expressed like this:

$$Y\_hat = b\_1*X\_1 + b\_2*X\_2 + b\_3*X\_3 + a$$

(To extend the crop example above, you might add the amount of sunlight and rainfall in a growing season to the fertilizer variable, with all three affecting Y_hat.)

Now, that form of multiple linear regression is happening at every node of a neural network. For each node of a single layer, input from each node of the previous layer is recombined with input from every other node. That is, the inputs are mixed in different proportions, according to their coefficients, which are different leading into each node of the subsequent layer. In this way, a net test which combination of input is significant as it tries to reduce error.

Once you sum your node inputs to arrive at Y_hat, it's passed through a non-linear function. Here's why: If every node merely performed multiple linear regression, Y_hat would increase linearly and without limit as the X's increase, but that doesn't suit our purposes.

What we are trying to build at each node is a switch (like a neuron…) that turns on and off, depending on whether it should let the signal of the input pass through to affect the ultimate decisions of the network.

When you have a switch, you have a classification problem. Does the input's signal indicate the node should classify it as enough, or not_enough, on or off? A binary decision can be expressed by 1 and 0, and logistic regression is a non-linear function that squashes input to translate it to a space between 0 and 1.

The non-linear transforms at each node are usually S-shaped functions like logistic regression. They go by the names of sigmoid (the Greek word for "S"), tanh, hard tanh, etc., and they shape the output of each node. The output of all nodes, each squashed into an s-shaped space between 0 and 1, is then passed as input to the next layer in a feed-forward neural network, and so on until the signal reaches the final layer of the net, where decisions are made.

Gradient Descent

The name for one commonly used optimization function that adjusts weights according to the error they caused is called "gradient descent."

The gradient is another word for slope, and slope, in its typical form on an x-y graph, represents how two variables relate to each other: rise over run, the change in money over the change in time, etc. In this case, the slope we care about describes the relationship between the network's error and a single weight; i.e. that is, how does the error vary as the weight is adjusted.

To put a finer point on it, which weight will produce the least error? Which one correctly represents the signals contained in the input data, and translates them to a correct classification? Which one can hear "nose" in an input image, and know that should be labelled as a face and not a frying pan?

As a neural network learns, it slowly adjusts many weights so that they can map signal to meaning correctly. The relationship between network Error and each of those weights is a derivative, dE/dw, that measures the degree to which a slight change in a weight causes a slight change in the error.

Each weight is just one factor in a deep network that involves many transforms; the signal of the weight passes through activations and sums over several layers, so we use the chain rule of calculus to march back through the networks activations and outputs and finally arrive at the weight in question, and its relationship to overall error.

The chain rule in calculus states that

$$\frac{dError}{dweight} = \frac{dError}{dactivation} * \frac{dactivation}{dweight}$$

That is, given two variables, Error and weight, that are mediated by a third variable, activation, through which the weight is passed, you can calculate how a change in weight affects a change in Error by first calculating how a change in activation affects a change in Error, and how a change in weight affects a change in activation.

The essence of learning in deep learning is nothing more than that: adjusting a model's weights in response to the error it produces until you can't reduce the error anymore.

Updaters

Possible Updaters: ADADELTA, ADAGRAD, ADAM, NESTEROVS, NONE, RMSPROP, SGD, CONJUGATE GRADIENT, HESSIAN FREE, LBFGS, LINE GRADIENT DESCENT

Activation Functions

The activation function determines what output a node will generate base upon its input. Sigmoid activation functions had been very popular, ReLU is currently very popular. In Deep Learning the activation function is set at the layer level and applies to all neurons in that layer.

Possible Activation functions: CUBE, ELU, HARDSIGMOID, HARDTANH, IDENTITY, LEAKYRELU, RATIONALTANH, RELU, RRELU, SIGMOID, SOFTMAX, SOFTPLUS, SOFTSIGN, TANH

Logistic Regression

On a deep neural network of many layers, the final layer has a role. When dealing with labelled input, the output layer classifies each example, applying the most likely label. Each node on the output layer represents one label, and that node turns on or off according to the strength of the signal it receives from the previous layer's input and parameters.

Each output node produces two possible outcomes, the binary output values 0 or 1 because an input variable either deserves a label or it does not. After all, there is no such thing as a little pregnant.

While neural networks working with labelled data produce a binary output, the input they receive is often continuous. That is, the signals that the network receives as input will span a range of values and include any number of metrics, depending on the problem it seeks to solve.

For example, a recommendation engine must make a binary decision about whether to serve an ad or not. But the input it bases its decision on could include how much a customer has spent on Amazon in the last week, or how often that customer visits the site.

So, the output layer must condense signals such as $67.59 spent on diapers, and 15 visits to a website, into a range between 0 and 1; i.e. a probability that a given input should be labelled or not.

The mechanism we use to convert continuous signals into binary output is called logistic regression. The name is unfortunate since logistic regression is used for classification rather than regression in the linear sense that most people are familiar with. It calculates the probability that a set of inputs match the label.

$$F(x) = \frac{1}{1 + e^{-x}}$$

Let's examine this little formula.

For continuous inputs to be expressed as probabilities, they must output positive results, since there is no such thing as a negative probability. That's why you see input as the exponent of e in the denominator – because exponents force our results to be greater than zero. Now consider the relationship of e's exponent to the fraction

1/1. One, as we know, is the ceiling of a probability, beyond which our results can't go without being absurd. (We're 120% sure of that.)

As the input x that triggers a label grows, the expression e to the x shrinks toward zero, leaving us with the fraction 1/1, or 100%, which means we approach (without ever quite reaching) absolute certainty that the label applies. Input that correlates negatively with your output will have its value flipped by the negative sign on e's exponent, and as that negative signal grows, the quantity e to the x becomes larger, pushing the entire fraction ever closer to zero.

Now imagine that, rather than having x as the exponent, you have the sum of the products of all the weights and their corresponding inputs – the total signal passing through your net. That's what you're feeding into the logistic regression layer at the output layer of a neural network classifier.

With this layer, we can set a decision threshold above which an example is labelled 1, and below which it is not. You can set different thresholds as you prefer – a low threshold will increase the number of false positives, and a higher one will increase the number of false negatives – depending on which side you would like to err.

Loss Functions

Possible Loss Functions: MSE: Mean Squared Error → Linear Regression, EXPLL: Exponential log likelihood→Poisson Regression, XENT: Cross Entropy→Binary Classification, MCXENT: Multiclass Cross Entropy, RMSE_XENT: RMSE Cross Entropy, SQUARED_LOSS: Squared Loss, NEGATIVELOGLIKELIHOOD: Negative Log Likelihood

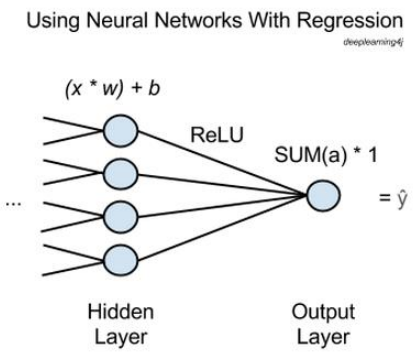
Neural-network with Regression

Broadly speaking, neural networks are used for clustering through unsupervised learning, classification through supervised learning, or regression. That is, they help group unlabelled data, categorize labelled data or predict continuous values.

While classification typically uses a form of logistic regression in the net's final layer to convert continuous data into dummy variables like 0 and 1 – e.g. given someone's height, weight and age you might bucket them as a heart-disease candidate or not – true regression maps one set of continuous inputs to another set of continuous outputs.

For example, given the age and floor space of a house and its distance from a good school, you might predict how much the house would sell for: continuous to continuous. No dummy variables like one find in classification, just mapping independent variables x to a continuous y.

Reasonable people can disagree about whether using neural networks for regression is overkill. The point of this post is just to explain how it can be done (it's easy).



In the diagram above, x stands for input, the features passed forward from the network's previous layer. Many x's will be fed into each node of the last hidden layer, and each x will be multiplied by a corresponding weight, w.

The sum of those products is added to a bias and fed into an activation function. In this case, the activation function is a *rectified linear unit* (ReLU), commonly used and highly useful because it doesn't saturate on shallow gradients as sigmoid activation functions do.

For each hidden node, ReLU outputs an activation, a, and the activations are summed going into the output node, which simply passes the activations' sum through.

That is, a neural network performing regression will have one output node, and that node will just multiply the sum of the previous layer's activations by 1. The result will be ŷ, "y hat", the network's estimate, the dependent variable that all your x's map to.

To perform back-propagation and make the network learn, you simply compare ŷ to the ground-truth value of y and adjust the weights and biases of the network until the error is minimized, much as you would with a classifier. Root-means-squared-error (RMSE) could be the loss function.

In this way, you can use a neural network to get the function relating an arbitrary number of independent variables x to a dependent variable y that you're trying to predict.

Neural Networks & Artificial Intelligence

In some circles, neural networks are thought of as "brute force" AI, because they start with a blank slate and hammer their way through to an accurate model. They are effective but to some eyes inefficient in their approach to modelling, which can't make assumptions about functional dependencies between output and input.

That said, gradient descent is not recombining every weight with every other to find the best match – its method of pathfinding shrinks the relevant weight space, and therefore the number of updates and required computation, by many orders of magnitude.

Enterprise-Scale Deep Learning

To train complex neural networks on very large datasets, a deep learning cluster using multiple chips, distributed over both GPUs and CPUs, is necessary if one is to train the network in a reasonable amount of time. Software engineers training those nets may avail themselves of GPUs in the cloud, or choose to depend on proprietary racks. Deep Learning should scale out equally well on both, using Spark as an access layer to orchestrate multiple host threads over many cores.

DENSER: Deep Evolutionary Network Structured Representation

Deep Evolutionary Network Structured Representation (DENSER) is a novel approach to automatically design Artificial Neural Networks (ANNs) using Evolutionary Computation. The algorithm not only searches for the best network topology but also tunes hyper-parameters (e.g., learning or data augmentation parameters). The automatic design is achieved using a representation with two distinct levels, where the outer level encodes the general structure of the network, and the inner level encodes the parameters associated with each layer. The allowed layers and hyper-parameter value ranges are defined by means of a human-readable Context-Free Grammar. The main contributions of this work are:

- DENSER, a general framework based on evolutionary principles that automatically searches for the adequate structure and parameterisation of large-scale deep networks that can have different layer types and/or goals;
- An automatically generated CNN that without any prior knowledge is effective on the classification of the CIFAR-10 dataset, with an average accuracy of **94.27%**;
- The demonstration that ANNs evolved with DENSER generalises well. In concrete, an average accuracy of **78.75%** on the CIFAR-100 dataset is obtained by a network whose topology was evolved for the CIFAR-10 dataset. To the best of our knowledge, this is the best result reported on the CIFAR-100 dataset by methods that automatically design CNNs.

The best-trained models can be found in https://git.io/vNtYV.

Proposed Approach: DENSER

To promote the evolution of the structure and parameters of ANNs we propose DENSER: Deep Evolutionary Network Structured Representation. DENSER gathers the basic ideas of Genetic Algorithms (GAs) and Dynamic Structured Grammatical Evolution (DSGE).

Representation

Each solution encodes an ANN by means of an ordered sequence of feedforward layers and their respective parameters; the learning and any other hyper-parameters can be encoded with everyone too. The representation of the candidate solution is made at two different levels:

- **GA Level** encodes the macrostructure of the networks and is responsible for representing the sequence of layers that later serves as an indicator of the grammatical starting symbol. It requires the definition of the allowed structure of the networks, i.e., the valid sequence of layers.
- **DSGE Level** encodes the parameters associated with a layer. The parameters and their allowed values or ranges are codified in the grammar that must be defined by the user.

Crossover

Two crossover operators are proposed, which are based on the different genotypic levels. In the context of the current work, a module does not stand for a set of layers that can be replicated multiple times but is rather a set of layers that belong to the same GA structural index.

Considering that all individuals have the same modules (with a possibly different number of layers) the first cross-over operator is a one-point crossover that changes layers between two individuals, within the same module.

The second crossover operators are a uniform crossover, that changes entire modules between two individuals.

Mutation

We develop a set of mutation operators, specifically targeted at promoting the evolution of ANNs. On the GA level, the mutations aim at manipulating the structure of the network:

- **Add layer** a new layer is generated based on the starting symbol possibilities of the module where the layer is to be placed;
- **Replicate layer** a layer is selected at random and copied into another valid position of the module. The copy is made by reference, i.e., if a parameter is changed in the layer all the copies are modified;
- **Remove layer** a layer is selected and deleted from a given module.

The above mutation operators only act on the general structure of the network; to change the parameters of the layers the following DSGE mutations are used:

- **Grammatical mutation** an expansion possibility is replaced by another one;
- **Integer mutation** a new integer value is generated (within the allowed range);
- **Float mutation** a Gaussian perturbation is applied to a given float value.

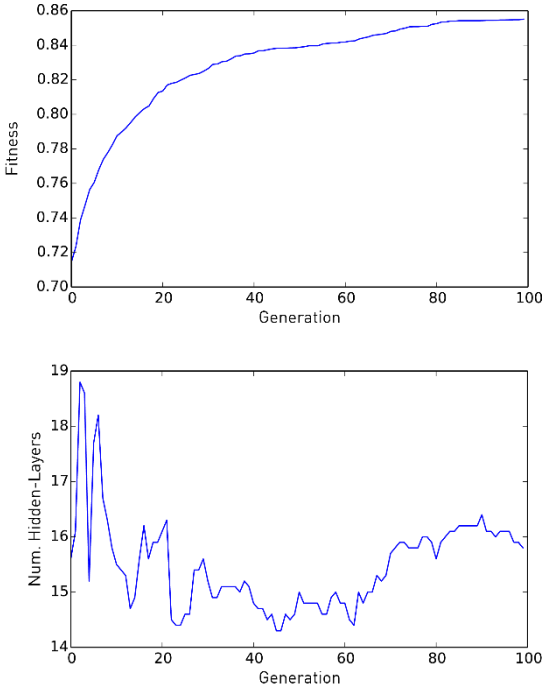
Experimental Results

To test the approach, we perform experiments on the generation of CNNs for the classification of the CIFAR-10 benchmark. The CIFAR-10 is made of 60000 instances, each a 32 x 32 RGB colour image that belongs to one of ten possible classes. The solutions evolved by DENSER are mapped to Keras models so that their performance can be measured. The goal of the task is the maximisation of the accuracy of the object recognition task.

To analyse the generalisation and scalability ability of the evolved topologies we then take the best-found CNN topologies and test them on the classification of the CIFAR-100 benchmark.

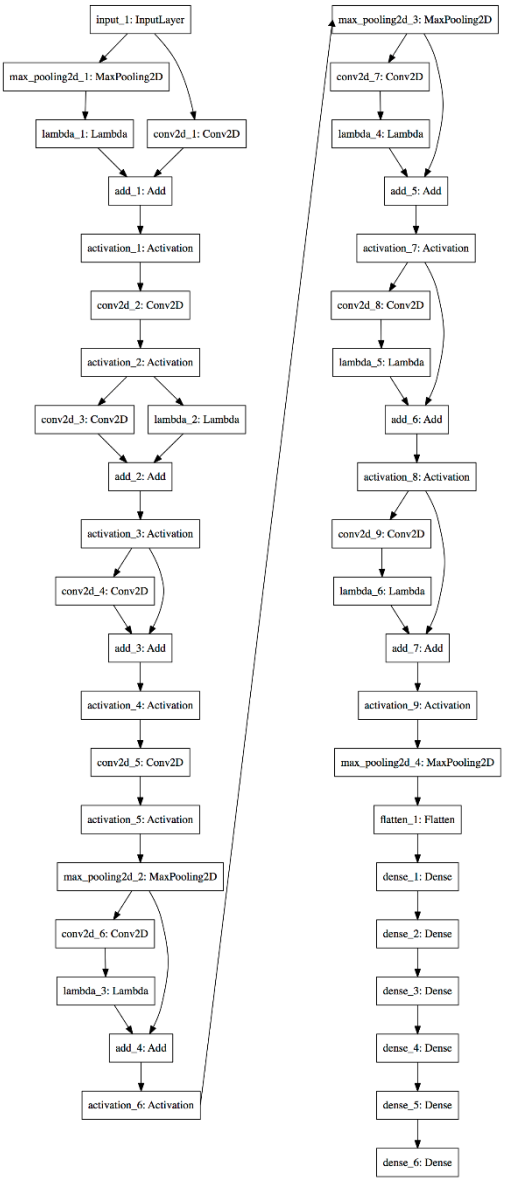
Evolution of CNNs for the CIFAR-10

We conduct 10 evolutionary runs on the generation of CNNs for the classification of the CIFAR-10 dataset. For the generated networks we analyse their fitness (i.e., accuracy on the classification task), and the number of hidden-layers.



Evolution of the fitness (top) and many hidden-layers (bottom) of the best individuals across generations.

Figure depicts the evolution of the average fitness and number of layers of the best CNNs across generations. A brief perusal of the results indicates that evolution is occurring, and solutions tend to converge around the 80th generation. Two different and contradictory behaviours are observable. From the start of evolution and until approximately the 60th generation an increase in performance is accompanied by a decrease in the number of layers; this changes from the 60th generation until the last generation where an increase in performance is followed by an increase in the number of hidden-layers of the best networks. This analysis reveals an apparent contradiction, that is explained by the fact that in the first generation the randomly generated solutions have many layers, whose



parameters are set at random.

The topology of the best performing network found by DENSER.

The fittest network found during evolution (in terms of validation accuracy) is represented in Figure 2. The most puzzling characteristic of the evolved network is the importance and number of dense layers that are used at the

end of the topology. To the best of our knowledge, the sequential use of such large number of dense layers is unprecedented, and it is fair to say that a human would never think of such topology, which makes this evolutionary outcome remarkable.

Once the evolutionary process is completed, the best network found in each run is re-trained 5 times. First, we train the networks with the same learning rate that is used during evolution (lr=0.01) but during 400 epochs (instead of 10). With this setup, we obtain, on average, a classification accuracy of 88.41% on the test set. To further enhance the accuracy of the networks we adopt the strategy described by Snoek et al.: for each instance of the test set we generate 100 augmented images, and the prediction is the maximum value of the average confidence values on the 100 augmented images. Following this validation approach, the average accuracy on the test set of the best-evolved networks increases to 89.93%.

To investigate if it is possible to increase the performance of the best networks we re-train them using the same strategy of CGP-CNN: a varying learning rate that starts at 0.01; on the 5th epoch it is increased to 0.1; by the 250th epoch it is decreased to 0.01; and finally, at the 375th it is reduced to 0.001. With the previous training policy, the average accuracy of the fittest network increases to 93.38%. If performing data augmentation on the test set, the average accuracy is 94.13%, i.e., an average error of 5.87%.

Method	**Accuracy**	
DENSER	94.13%	10.81×10^6
CGP-CNN (ResSet)	94.02%	1.68×10^6
CGP-CNN (ConvSet)	93.25%	1.52×10^6
Snoek et al.	93.63%	–
CoDeepNEAT	92.7%	–

The table of Figure shows a comparison with the best results reported by other methods. An analysis of the results shows that DENSER is the one that reports the highest accuracy. The number of trainable parameters is much higher in our methodology because we allow the placement of fully-connected layers in the evolved CNNs. In addition, during evolution, no prior knowledge is used to bias the search space.

Generalisation to the CIFAR-100

To test the generalisation and scalability of the evolved networks we take the best network generated by DENSER on the CIFAR-10 dataset and apply it to the classification of the CIFAR-100 dataset. For the network to work on the CIFAR-100 dataset we only change the SoftMax layer to have 100 output neurons, instead of 10.

The training of each network is stochastic; the initial conditions are different, and they are trained using different instances of the dataset, because of the data augmentation process. Thus, and to further improve the results, we investigate if the approach proposed by Graham to test the performance of the fractional max-pooling increases the performance reported by our network. In brief words, instead of a single network, an ensemble is used, where each network that is part of the ensemble is the result of an independent training of the network. Using this

method, an ensemble of the same network trained 5 times reports a test accuracy of 77.51%, an ensemble of 10 trains a test accuracy of 77.89%, and an ensemble of 12 trains a test accuracy of 78.14%. These results outperform those reported in the literature for the evolution of CNNs with a standard data augmentation methodology.

Moreover, and instead of forming ensembles with the same network structure we also tested the performance of building an ensemble using by the two best network topologies found by DENSER, similarly to what is done by Real et al. Following this methodology, we obtain were able to increase the accuracy to 78.75%.

Method	Accuracy
DENSER	78.75%
Real et al.	77.00%
Graham	73.61%
Snoek et al.	72.60%

The table of Figure 4 shows a comparison of the results obtained by DENSER on CIFAR-100 with those of other approaches.

Deep Learning Use Cases

Deep learning excels at identifying patterns in unstructured data, which most people know as media such as images, sound, video and text. Below is a list of sample use cases we've run across, paired with the sectors to which they pertain.

General Use Case	Industry
Sound	
Voice Recognition	UX/UI, Automotive, Security, IoT
Voice Search	Handset maker, Telecoms
Sentiment Analysis	CRM
Flaw Detection	Automotive, Aviation, Finance, Retail
Fraud Detection	Finance, Credit Cards
Time Series	

Log Analysis/Risk Detection	Data Centres, Security, Finance
Enterprise Resource Planning	Manufacturing, Auto., Supply Chain
Predictive Analytics using Sensor Data	IoT, Smart Home, Hardware Manufacturing
Business and Economics Analysis	Finance, Accounting, Government
Recommendation Engine	E-Commerce, Media, Social Networks
Text	
Sentiment Analysis	CRM, Social Media, Reputation Management
Augmented searches, Theme Detection	Finance
Threat Detection	Social Media, Government
Fraud Detection	Insurance, Finance
Image	
Facial Recognition	Handset, Laptops, Security
Image Watch	Social Media
Machine Vision	Automotive, Aviation
Photo Clustering	Telecoms, Handset makers
Video	
Motion Detection	Gaming, UX/UI
Real-Time Threat Detection	Security, Airports, Government

Feature Introspection

Traditional machine learning has the advantage of feature introspection – that is, it knows why it is classifying an input in one way or another, which is important for analytics. But that very advantage is what excludes it from working with unlabelled and unstructured data, as well as attaining the record-breaking accuracy of the latest

deep learning models. Feature engineering is one of the major choke-points of traditional machine learning since so few people can do it well and quickly enough to adapt to changing data.

For cases where feature introspection is necessary (e.g. the law requires that you justify a decision to, say, deny a loan due to predicted credit risk), we recommend using a deep net in an ensemble with machine-learning algorithms, allowing each one to vote and relying on each for its strength. Alternatively, one can perform various analyses on the results of deep nets to form hypotheses about their decision-making.

Text

Named-Entity Recognition

One use of deep-learning networks is named-entity recognition, which is a way to extract from unstructured, unlabelled data certain types of information like people, places, companies or things. That information can then be stored in a structured schema to build, say, a list of addresses or serve as a benchmark for an identity validation engine.

Speech-to-Text

With the proper data transforms, a deep network is capable of understanding audio signals. This can be used to identify snippets of sound in larger audio files and transcribe the spoken word as text.

Image

Object Recognition

Object recognition is an algorithm's ability to identify arbitrary objects – such as balls, animals, or even faces – within larger images. This is typically used in engineering applications to identify shapes for modelling purposes. It's also used by social networks for photo tagging. Facebook's Deep Face is a good example of a deep-learning application in this realm.

Machines Vision + Natural-Language Processing

Advances in reality capture and reality computing are making virtual and real worlds converge. One application of deep learning to this newly available data is to recognize and label objects in 3D environments, and in real life.

From there, it's a short step to simulated semantics, in which machines learn the nature and constraints of objects in the world, through their virtual representations, and then bring that understanding to the language they generate and ingest. We believe that is one of many futures in store for neural nets.

Appendix A – Deep Learning Glossary

The intent of this glossary is to provide clear definitions of the technical terms specific to deep artificial neural networks. It is a work in progress.

Activation

An activation, or an activation function, for a neural network, is defined as the mapping of the input to the output via a non-linear transform function at each "node", which is simply a locus of computation within the net. Each layer in a neural net consists of many nodes, and the number of nodes in a layer is known as its width.

Activation algorithms are the gates that determine, at each node in the net, whether and to what extent to transmit the signal the node has received from the previous layer. A combination of weights (coefficients) and biases work on the input data from the previous layer to determine whether that signal surpasses a given threshold and is deemed significant. Those weights and biases are slowly updated as the neural net minimizes its error; i.e. the level of nodes' activation change during learning. Deep Learning includes activation functions such as sigmoid, relu, tanh and ELU. These activation functions allow neural networks to make complex boundary decisions for features at various levels of abstraction.

Adadelta

Adadelta is an updater, or learning algorithm, related to gradient descent. Unlike SGD, which applies the same learning rate to all parameters of the network, Adadelta adapts the learning rate per parameter.

- ADADELTA: An Adaptive Learning Rate Method

Adagrad

Adagrad, short for the adaptive gradient, is an updater or learning algorithm that adjusts the learning rate for each parameter in the net by monitoring the squared gradients during learning. It is a substitute for SGD and can be useful when processing sparse data.

- Adaptive Subgradient Methods for Online Learning and Stochastic Optimization

Adam

Adam (Gibson) co-created Deeplearning4j. Adam is also an updater, like rmsprop, which uses a running average of the gradient's first and second moment plus a bias-correction term.

- Adam: A Method for Stochastic Optimization

Affine Layer

Affine is a fancy word for a fully connected layer in a neural network. "Fully connected" means that all the nodes of one layer connect to all the nodes of the subsequent layer. A restricted Boltzmann machine, for example, is a fully connected layer. Convolutional networks use affine layers interspersed with both their namesake convolutional layers (which create feature maps based on convolutions) and downsampling layers, which throw out a lot of data and only keep the maximum value. "Affine" derives from the Latin affinis, which means bordering or connected with. Each connection, in an affine layer, is a passage whereby input is multiplied by a weight and added

to a bias before it accumulates with all other inputs at a given node, the sum of which is then passed through an activation function: e.g. **output = activation(weight*input+bias)**, or **y = f(w*x+b)**.

AlexNet

AlexNet is a deep convolutional network named after Alex Krizhevsky, a former student of Geoff Hinton's at the University of Toronto, now at Google. AlexNet was used to win ILSVRC 2012 and foretold a wave of deep convolutional networks that would set new records in image recognition. AlexNet is now a standard architecture: it contains five convolutional layers, three of which are followed by max-pooling (downsampling) layers, two fully connected (affine) layers – all of which ends in a soft-max layer.

- ImageNet Classification with Deep Convolutional Neural Networks

Attention Models

Attention models "attend" to specific parts of an image in the sequence, one after another. By relying on a sequence of glances, they capture visual structure, much like the human eye is believed to function with foveation. This visual processing, which relies on a recurrent network to process sequential data, can be contrasted with other machine vision techniques that process a whole image in a single, forward pass.

- DRAW: A Recurrent Neural Network For Image Generation.

Auto-encoder

Auto-encoders are at the heart of representation learning. They encode input, usually by compressing large vectors into smaller vectors that capture their most significant features; that is, they are used for data compression (dimensionality reduction) as well as data reconstruction for unsupervised learning. A restricted Boltzmann machine is a type of autoresponder, and in fact, auto-encoders come in many flavours, including Variational Auto-encoders, De-noising Auto-encoders and Sequence Auto-encoders. Variational auto-encoders have replaced RBMs in many labs because they produce more stable results. De-noising auto-encoders provide a form of regularization by introducing Gaussian noise into the input, which the network learns to ignore in search of the true signal.

- Auto-Encoding Variational Bayes
- Stacked De-noising Auto-encoders: Learning Useful Representations in a Deep Network with a Local De-noising Criterion
- Semi-supervised Sequence Learning

Back-propagation

To calculate the gradient the relate weights to error, we use a technique known as back-propagation, which is also referred to as the backward pass of the network. Back-propagation is a repeated application of chain rule of calculus for partial derivatives. The first step is to calculate the derivatives of the objective function with respect to the output units, then the derivatives of the output of the last hidden layer to the input of the last hidden layer; then the input of the last hidden layer to the weights between it and the penultimate hidden layer, etc. Here's a derivation of back-propagation. And here's Yann LeCun's important paper on the subject.

A special form of back-propagation is called back-propagation through time, or BPTT, which is specifically useful for recurrent networks analysing text and time series. With BPTT, each time step of the RNN is the equivalent of a layer in a feed-forward network. To back-propagate over many time steps, BPTT can be truncated for efficiency. Truncated BPTT limits the time steps over which error is propagated.

- Back-propagation Through Time: What It Does and How to Do It

Batch Normalization

Batch Normalization does what it says: it normalizes mini-batches as they're fed into a neural-net layer. Batch normalization has two potential benefits: it can accelerate learning because it allows you to employ higher learning rates, and regularizes that learning.

- Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
- Overview of mini-batch gradient descent (U. Toronto)

Bayes Theorem

Bayes' Theorem is *a mathematical framework for integrating new evidence with prior beliefs.* For example, suppose you're sitting in your quiet suburban home and you hear something that sounds like a lion roaring. You have some prior beliefs that lions are unlikely to be near your house, so you figure that it's probably not a lion. Probably it's some weird machine of your neighbour's that just happens to sound like a lion, or some kids pranking you by playing lion noises, or something. You end up believing that there's probably no lion nearby, but you do have a slightly higher probability of there being a lion nearby than you had before you heard the roaring noise. Bayes' Theorem is just this kind of reasoning converted to math. source

- Bayes Rules: A Theoretical Guide

Bidirectional Recurrent Neural Networks

A Bidirectional RNN has composed of two RNNs that process data in opposite directions. One reads a given sequence from start to finish; the other reads it from finish to start. Bidirectional RNNs are employed in NLP for translation problems, among other use cases. Deeplearning4j's implementation of bidirectional Graves LSTMs is here.

- Bidirectional Recurrent Neural Networks

Binarization

The process of transforming data into a set of zeros and ones. An example would be grey-scaling an image by transforming a picture from the 0-255 spectrum to a 0-1 spectrum.

Boltzmann Machine

"A Boltzmann machine learns internal (not defined by the user) concepts that help to explain (that can generate) the observed data. These concepts are captured by random variables (called hidden units) that have a joint distribution (statistical dependencies) among themselves and with the data, and that allow the learner to capture highly non-linear and complex interactions between the parts (observed random variables) of any observed example (like the pixels in an image). You can also think of these higher-level factors or hidden units as another, more

abstract, representation of the data. The Boltzmann machine is parametrized through simple two-way interactions between every pair of random variable involved (the observed ones as well as the hidden ones).” - Yoshua Bengio

Channel

Channel is a word used when speaking of convolutional networks. ConvNets treat colour images as volumes; that is, an image has height, width and depth. The depth is the number of channels, which coincide with how you encode colours. RGB images have three channels, for red, green and blue respectively.

Class

Used in classification a Class refers to a label applied to a group of records sharing similar characteristics.

Confusion Matrix

Also known as an error matrix or contingency table. Confusions matrices allow you to see if your algorithm is systematically confusing two labels, by contrasting your net’s predictions against a benchmark.

Contrastive Divergence

“Contrastive divergence is a recipe for training undirected graphical models (a class of probabilistic models used in machine learning). It relies on an approximation of the gradient (a good direction of change for the parameters) of the log-likelihood (the basic criterion that most probabilistic learning algorithms try to optimize) based on a short Markov chain (a way to sample from probabilistic models) started at the last example seen. It has been popularized in the context of Restricted Boltzmann Machines (Hinton & Salakhutdinov, 2006, Science), the latter being the first and most popular building block for deep learning algorithms.” ~*Yoshua Bengio*

Convolutional Network (CNN)

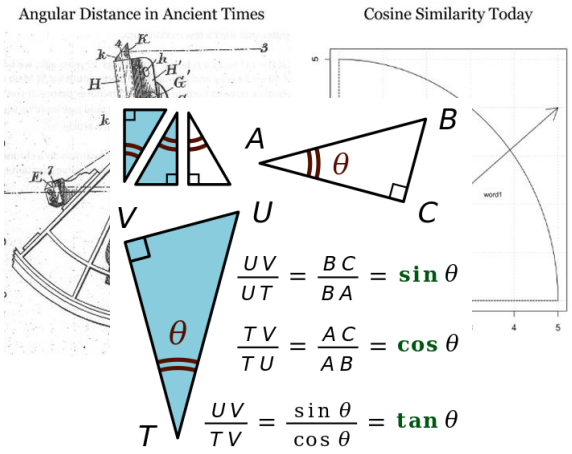
Convolutional networks are a deep neural network that is currently the state-of-the-art in image processing. They are setting new records in accuracy every year on widely accepted benchmark contests like ImageNet.

From the Latin *convolvere*, “to convolve” means to roll together. For mathematical purposes, a convolution is the integral measuring how much two functions overlap as one passes over the other. Think of a convolution as a way of mixing two functions by multiplying them: a fancy form of multiplication.

Imagine a tall, narrow bell curve standing in the middle of a graph. The integral is the area under that curve. Imagine near it a second bell curve that is shorter and wider, drifting slowly from the left side of the graph to the right. The product of those two functions’ overlap at each point along the x-axis is their convolution. So, in a sense, the two functions are being “rolled together.”

Cosine Similarity

It turns out two vectors are just 66% of a triangle, so let’s do a quick trig review.



Trigonometric functions like *sine*, *cosine* and *tangent* are ratios that use the lengths of a side of a right triangle (opposite, adjacent and hypotenuse) to compute the shape's angles. By feeding the sides into ratios like these

We can also know the angles at which those sides intersect. Remember SOH-CAH-TOA?

Differences between word vectors, as they swing around the origin like the arms of a clock, can be thought of as differences in degrees.

And like ancient navigators gauging the stars by a sextant, we will measure the *angular distance* between words using something called *cosine similarity*. You can think of words as points of light in a dark canopy, clustered together in constellations of meaning.

To find that distance knowing only the word vectors, we need the equation for vector dot multiplication (multiplying two vectors to produce a single, scalar value).

Cosine is the angle attached to the origin, which makes it useful here. (We normalize the measurements, so they come out as percentages, where 1 means that two vectors are equal, and 0 means they are perpendicular, bearing no relation to each other.)

Data Parallelism and Model Parallelism

Training a neural network on a very large dataset requires some form of parallelism, of which there are two types: data parallelism and model parallelism.

- **Data parallelism** Let's say you have a very large image dataset of 1,000,000 faces. Those faces can be

$$\vec{a} \cdot \vec{b} = \left\|\vec{a}\right\| \left\|\vec{b}\right\| \cos\theta$$

The dot product of vectors *a* and *b* is equal to the product of those vectors' norms (the absolute values of their respective lengths) times the cosine of the angle that separates them, theta.

Which can be rewritten like this.

$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{\left\|\vec{a}\right\| \left\|\vec{b}\right\|}$$

divided into batches of 10, and then 10 separate batches can be dispatched simultaneously to 10 different

convolutional networks so that 100 instances can be processed at once. The 10 different CNNs would then train on a batch, calculate the error on that batch, and update their parameters based on that error. Then, using parameter averaging, the 10 CNNs would update a central, master CNN that would take the average of their updated parameters. This process would repeat until the entire dataset has been exhausted.

- **Model parallelism** is another way to accelerate neural net training on very large datasets. Here, instead of sending batches of faces two separate neural networks, let's imagine a different kind of image: an enormous map of the earth. Model parallelism would divide that enormous map into regions, and it would park a separate CNN in each region, to train on only that area and no other. Then, as each enormous map was peeled off the dataset to train the neural networks, it would be broken up and different patches of it would be sent to train on separate CNNs. No parameter averaging necessary here.

Data Science

Data science is the discipline of drawing conclusions from data using computation. There are three core aspects of effective data analysis: exploration, prediction, and inference.

Deep-Belief Network (DBN)

A deep-belief network is a stack of restricted Boltzmann machines, which are themselves a feed-forward autoresponder that learns to reconstruct input layer by layer, greedily. Pioneered by Geoff Hinton and crew. Because a DBN is deep, it learns a hierarchical representation of the input. Because DBNs learn to reconstruct that data, they can be useful in unsupervised learning.

- A fast learning algorithm for deep belief nets

Distributed Representations

The Nupic community has a good explanation of distributed representations here. Other good explanations can be found on this Quora page.

Downpour Stochastic Gradient Descent

Downpour stochastic gradient descent is an asynchronous stochastic gradient descent procedure, employed by Google among others, that expands the scale and increases the speed of training deep-learning networks.

Dropout

Dropout is a hyperparameter used for regularization in neural networks. Like all regularization techniques, its purpose is to prevent overfitting. Dropout randomly makes nodes in the neural network "drop out" by setting them to zero, which encourages the network to rely on other features that act as signals. That, in turn, creates more generalizable representations of data.

- Dropout: A Simple Way to Prevent Neural Networks from Overfitting
- Recurrent Neural Network Regularization

DropConnect

DropConnect is a generalization of Dropout for regularizing large fully-connected layers within neural networks. Dropout sets a randomly selected subset of activations to zero at each layer. DropConnect, in contrast, sets a randomly selected subset of weights within the network to zero.

- Regularization of Neural Networks using DropConnect

Embedding

An embedding is a representation of the input or an encoding. For example, a neural word embedding is a vector that represents that word. The word is said to be embedded in vector space. Word2vec and GloVe are two techniques used to train word embeddings to predict a word's context. Because an embedding is a form of representation learning, we can "embed" any data type, including sounds, images and time series.

Epoch vs. Iteration

In the machine-learning parlance, an epoch is a complete pass through a given dataset. That is, by the end of one epoch, your neural network – be it a restricted Boltzmann machine, convolutional net or deep-belief network – will have been exposed to every record to an example within the dataset once. Not to be confused with an iteration, which is simply one update of the neural net model's parameters. Many iterations can occur before an epoch is over. Epoch and iteration are only synonymous if you update your parameters once for each pass through the whole dataset; if you update using mini-batches, they mean different things. Say your data has 2 mini batches: A and B..numIterations(3) performs training like AAABBB, while 3 epochs look like ABABAB.

Epoch

An Epoch is a complete pass through all the training data. A neural network is trained until the error rate is acceptable, and this will often take multiple passes through the complete dataset.

Note An iteration is when parameters are updated and is typically less than a full pass. For example, if BatchSize is 100 and data size is 1,000 an epoch will have 10 iterations. If trained for 30 epochs there will be 300 iterations.

Extract, transform, load (ETL)

Data is loaded from disk or other sources into memory with the proper transforms such as binarization and normalization. Broadly, you can think of a data pipeline as the process over gathering data from disparate sources and locations, putting it into a form that your algorithms can learn from, and then placing it in a data structure that they can iterate through.

F1 Score

An f1 score is a number between zero and one that explains how well the network performed during training. It is analogous to a percentage, with 1 being the best score and zero the worst. f1 is basically the probability that your net's guesses are correct.

F1 = 2 * ((precision * recall) / (precision + recall))

Accuracy measures how often you get the right answer, while f1 scores are a measure of accuracy. For example, if you have 100 fruits – 99 apples and 1 orange – and your model predicts that all 100 items are apples, then it is 99% accurate. But that model failed to identify the difference between apples and oranges. f1 scores help you judge whether a model is doing well as classifying when you have an imbalance in the categories you're trying to tag.

An f1 score is an average of both precision and recall. More specifically, it is a type of average called the harmonic mean, which tends to be less than the arithmetic or geometric means. Recall answers: "Given a positive example,

how likely is the classifier going to detect it?" It is the ratio of true positives to the sum of true positives and false negatives.

Precision answers "Given a positive prediction from the classifier, how likely is it to be correct?" It is the ratio of true positives to the sum of true positives and false positives.

For f1 to be high, both recall and precision of the model must be high.

Feed-Forward Network

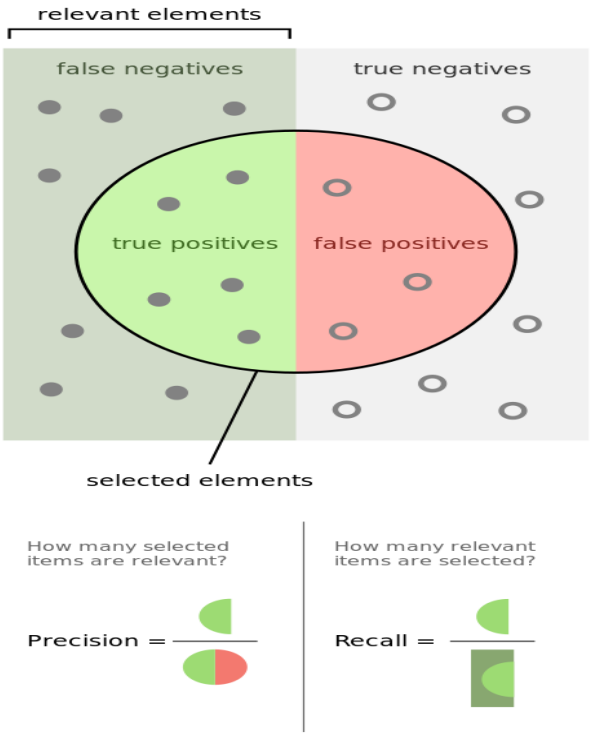
A neural network that takes the initial input and triggers the activation of each layer of the network successively, without circulating. Feed-forward nets contrast with recurrent and recursive nets in that feed-forward nets never let the output of one node circle back to the same or previous nodes.

Gaussian Distribution

A Gaussian, or normal, distribution, is a continuous probability distribution that represents the probability that any given observation will occur at different points of a range. Visually, it resembles what's usually called a Bell curve.

Gaussian Process

Nando de Freitas's lecture on Gaussian Processes.



Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are a tool to conduct unsupervised learning, essentially pitting a generative net against a discriminative net. The first net tries to fool the second by mimicking the probability distribution

of a training dataset to fool the discriminative net into judging that the generated data instance belongs to the training set.

Global Vectors (GloVe)

The glove is a generalization of Tomas Mikolov's word2vec algorithms, a technique for creating neural word embeddings. It was first presented at NIPS by Jeffrey Pennington, Richard Socher and Christopher Manning of Stanford's NLP department.

- GloVe: Global Vectors for Word Representation

Gradient Descent

The gradient is a derivative, which you will know from differential calculus. That is, it's the ratio of the rate of change of a neural net's parameters and the error it produces, as it learns how to reconstruct a dataset or make guesses about labels. The process of minimizing error is called gradient descent. Descending a gradient has two aspects: choosing the direction to step in (momentum) and choosing the size of the step (learning rate).

Since MLPs are, by construction, differentiable operators, they can be trained to minimise any differentiable objective function using gradient descent. The basic idea of gradient descent is to find the derivative of the objective function with respect to each of the network weights, then adjust the weights in the direction of the negative slope. -Graves

Gradient Clipping

Gradient Clipping is one way to solve the problem of exploding gradients. Exploding gradients arise in deep networks when gradients associating weights and the net's error become too large. Exploding gradients are frequently encountered in RNNs dealing with long-term dependencies. One way to clip gradients is to normalize them when the L2 norm of a parameter vector surpasses a given threshold.

Graphical Models

A directed graphical model is another name for a Bayesian net, which represents the probabilistic relationships between the variables represented by its nodes.

Gated Recurrent Unit (GRU)

A GRU is a pared-down LSTM. GRUs rely on gating mechanisms to learn long-range dependencies while sidestepping the vanishing gradient problem. They include reset and update gates to decide when to update the GRUs memory at each time step.

Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation

Highway Networks

Highway networks are an architecture introduced by Schmidhuber et al to let information flow unhindered across several RNN layers on so-called "information highways." The architecture uses gating units that learn to regulate the flow of information through the net. Highway networks with hundreds of layers can be trained directly using SGD, which means they can support very deep architectures.

- Highway Networks

Hyperplane

"A hyperplane in an n-dimensional Euclidean space is a flat, n-1-dimensional subset of that space that divides the space into two disconnected parts. What does that mean intuitively?

First, think of the real line. Now pick a point. That point divides the real line into two parts (the part above that point, and the part below that point). The real line has 1 dimension, while the point has 0 dimensions. So, a point is a hyperplane of the real line.

Now think of the two-dimensional plane. Now pick any line. That line divides the plane into two parts ("left" and "right" or maybe "above" and "below"). The plane has 2 dimensions, but the line has only one. So, a line is a hyperplane of the 2d plane. Notice that if you pick a point, it doesn't divide the 2d plane into two parts. So, one point is not enough.

Now think of a 3d space. Now to divide the space into two parts, you need a plane. Your plane has two dimensions, your space has three. So, a plane is a hyperplane for a 3d space.

OK, now we've run out of visual examples. But suppose you have a space of n dimensions. You can write down an equation describing an n-1-dimensional object that divides the n-dimensional space into two pieces. That's a hyperplane." -Quora

International Conference on Learning Representations

ICLR, pronounced "I-clear". An important conference. See representation learning.

International Conference for Machine Learning

ICML, or the International Conference for Machine Learning, is a well-known and well attended machine-learning conference.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

The ImageNet Large Scale Visual Recognition Challenge is the formal name for ImageNet, a yearly contest held to solicit and evaluate the best techniques in image recognition. Deep convolutional architectures have driven error rates on the ImageNet competition from 30% to less than 5%, which means they now have human-level accuracy.

Iteration

An iteration is an update of weights after analysing a batch of input records. See Epoch for clarification.

LeNet

Google's LeNet architecture is a deep convolutional network. It won ILSVRC in 2014 and introduced techniques for paring the size of a CNN, thus increasing computational efficiency.

- Going Deeper with Convolutions

Long Short-Term Memory Units (LSTM)

LSTMs are a form of recurrent neural network invented in the 1990s by Sepp Hochreiter and Juergen Schmidhuber, and now widely used for image, sound and time series analysis, because they help solve the vanishing gradient

problem by using a memory gates. Alex Graves made significant improvements to the LSTM with what is now known as the Graves LSTM.

- Original Paper: LONG SHORT-TERM MEMORY

Log-Likelihood

Log-likelihood is related to the statistical idea of the likelihood function. The likelihood is a function of the parameters of a statistical model. "The probability of some observed outcomes given a set of parameter values is referred to as the likelihood of the set of parameter values given the observed outcomes."

Logistic Regression

Logistic regression behaves like an on-off switch, and is usually used for classification problems; e.g. does this data instance belong to category X or not? It produces a value between 1 and 0 that corresponds to the probability that the data belongs to a given class.

Maximum Likelihood Estimation

"Say you have a coin and you're not sure it's "fair." So, you want to estimate the "true" probability it will come up heads. Call this probability P, and code the outcome of a coin flip as 1 if it's heads and 0 if it's tails. You flip the coin four times and get 1, 0, 0, 0 (i.e., 1 heads and 3 tails). What is the likelihood that you would get these outcomes, given P? Well, the probability of heads is P, as we defined it above. That means the probability of tails is (1 - P). So, the probability of 1 heads and 3 tails is P * (1 - P)3 [Edit: We call this the "likelihood" of the data]. If we "guess" that the coin is fair, that's saying P = 0.5, so the likelihood of the data is L =.5 * (1 - .5)3 = .0625. What if we guess that P = 0.45? Then L = .45 * (1 - .45)3 = ~.075. So, P = 0.45 is a better estimate than P = 0.5, because the data are "more likely" to have occurred if P = 0.45 than if P = 0.5. At P = 0.4, the likelihood is 0.4 * (1 - 0.4)3 = .0864. At P = 0.35, the likelihood is 0.35 * (1 - 0.35)3 = .096. In this case, it turns out that the value of P that maximizes the likelihood is P = 0.25. So that's our "maximum likelihood" estimate for P. In practice, the max likelihood is harder to estimate than this (with predictors and various assumptions about the distribution of the data and error terms), but that's the basic concept behind it." *–u/jacknbox*

So, in a sense, the probability is treated as an unseen, internal property of the data. A parameter. And the likelihood is a measure of how well the outcomes recorded in the data match our hypothesis about their probability; i.e. our theory about how the data is produced. The better our theory of the data's probability, the higher the likelihood of a given set of outcomes.

Model

In neural networks, the model is the collection of weights and biases that transform input into output. A neural network is a set of algorithms that update models such that the models guess with less error as they learn. A model is a symbolic, logical or mathematical machine whose purpose is to deduce output from input. If a model's assumptions are correct, then one must necessarily believe its conclusions. Neural networks produced trained models that can be deployed to process, classify, cluster and make predictions about data.

MNIST

MNIST is the "hello world" of deep-learning datasets. Everyone uses MNIST to test their neural networks, just to see if the networks at all. MNIST contains 60,000 training examples and 10,000 test examples of the handwritten

numerals 0-9. These images are 28×28 pixels, which means they require 784 nodes on the first input layer of a neural network. MNIST is available for download here.

Model Score

As your model trains, the goal of training is to improve the "score" for the output or the overall error rate. The WebUI will present a graph of the score for each iteration. For text-based console output of the score as the model trains, you would use Score Iteration Listener.

Nesterov's Momentum

Momentum also known as Nesterov's momentum, influences the speed of learning. It causes the model to converge faster to a point of minimal error. Momentum adjusts the size of the next step, the weight update, based on the previous step's gradient. That is, it takes the gradient's history and multiplies it. Before each new step, a provisional gradient is calculated by taking partial derivatives from the model, and the hyperparameters are applied to it to produce a new gradient. Momentum influences the gradient your model uses the next step.

Multilayer Perceptron

MLPs are perhaps the oldest form of deep neural network. They consist of multiple, fully connected feed-forward layers.

Neural Machine Translation

Neural machine translation maps one language to another using neural networks. Typically, recurrent neural networks are used to ingest a sequence from the input language and output a sequence in the target language.

- Sequence of Sequence Learning with Neural Networks

Noise-Contrastive Estimations (NCE)

Noise-contrastive estimation offers a balance of computational and statistical efficiency. It is used to train classifiers with many classes in the output layer. It replaces the soft-max probability density function, an approximation of a maximum likelihood estimator that is cheaper computationally.

- Noise-contrastive estimation: A new estimation principle for unnormalized statistical models
- Learning word embeddings efficiently with noise-contrastive estimation

Non-linear Transform Function

A function that maps input on a non-linear scale such as sigmoid or tanh. A non-linear function's output is not directly proportional to its input.

Normalization

The process of transforming the data to span a range from 0 to 1.

Objective Function

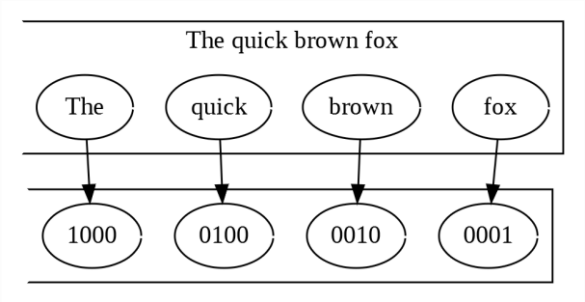
Also called a loss function or a cost function, an objective function defines what success looks like when an algorithm learns. It is a measure of the difference between a neural net's guess and the ground truth; that is, the error. Measuring that error is a precondition to updating the neural net in such a way that its guesses generate less error.

The error resulting from the loss function is fed into back-propagation to update the weights and biases that process input in the neural network.

One-Hot Encoding

Used in classification and bag of words. The label for each example is all 0s, except for a 1 at the index of the actual class to which the example belongs. For BOW, the one represents the word encountered.

Below is an example of one-hot encoding for the phrase "The quick brown fox".



Pooling

Pooling, max pooling and average pooling are terms that refer to downsampling or subsampling within a convolutional network. Downsampling is a way of reducing the amount of data flowing through the network, and therefore decreasing the computational cost of the network. Average pooling takes the average of several values. Max pooling takes the greatest of several values. Max pooling is currently the preferred type of downsampling layer in convolutional networks.

Probability Density

Probability densities are used in unsupervised learning, with algorithms such as auto-encoders, VAEs and GANs.

"A probability density essentially says, "for a given variable (e.g. radius) what, at that particular value, is the likelihood of encountering an event or an object (e.g. an electron)?" So, if I'm at the nucleus of an atom and I move to, say, one Angstrom away, at one Angstrom there is a certain likelihood I will spot an electron. But we like to not just ask for the probability at one point; we'd sometimes like to find the probability for a range of points: What is the probability of finding an electron between the nucleus and one Angstrom, for example. So, we add up ("integrate") the probability from zero to one Angstrom. For the sake of convenience, we sometimes employ "normalization"; that is, we require that adding up all the probabilities over every possible value will give us 1.00000000 (etc)." *–u/beigebox*

Probability Distribution

"A probability distribution is a mathematical function and/or graph that tells us how likely something is to happen.

So, for example, if you're rolling two dice and you want to find the likelihood of each possible number you can get, you could make a chart that looks like this. As you can see, you're most likely to get a 7, then a 6, then an 8, and so on. The numbers on the left are the percent of the time where you'll get that value, and the ones on the right are a fraction (they mean the same thing, just different forms of the same number). The way that if you use the distribution to find the likelihood of each outcome is this:

There are 36 possible ways for the two dice to land. There are 6 combinations that get you 7, 5 that get you 6/8, 4 that get you 5/9, and so on. So, the likelihood of each one happening is the number of possible combinations that get you that number divided by the total number of possible combinations. For 7, it would be 6/36, or 1/6, which you'll notice is the same as what we see in the graph. For 8, it's 5/36, etc. etc.

The key thing to note here is that the sum of all the probabilities will equal 1 (or, 100%). That's important because it's essential that there be a result of rolling the two die every time. If all the percentages added up to 90%, what the heck is happening that last 10% of the time?

So, for more complex probability distributions, the way that the distribution is generated is more involved, but the way you read it is the same. If, for example, you see a distribution that looks like this, you know that you're going to get a value of μ 40% (corresponding to.4 on the left side) of the time whenever you do whatever the experiment or test associated with that distribution.

The percentages in the shaded areas are also important. Just like earlier when I said that the sum of all the probabilities must equal 1 or 100%, the area under the curve of a probability distribution must equal 1, too. You don't need to know why that is (it involves calculus), but it's worth mentioning. You can see that the graph I linked is helpfully labelled; the reason they do that is to show you that you what percentage of the time you're going to end up somewhere in that area.

So, for example, about 68% of the time, you'll end up between -1σ and 1σ." *–u/corpuscle634*

Reconstruction Entropy

After applying Gaussian noise, a kind of statistical white noise, to the data, this objective function punishes the network for any result that is not close to the original input. That signal prompts the network to learn different features to reconstruct the input better and minimize error.

Rectified Linear Units

Rectified linear units, or reLU, are a non-linear activation function widely applied in neural networks because they deal well with the vanishing gradient problem. They can be expressed so: f(x) = max(0, x), where activation is set to zero if the output does not surpass a minimum threshold, and activation increases linearly above that threshold.

- Rectifier Non-linearities Improve Neural Network Acoustic Models
- Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification
- Rectified Linear Units Improve Restricted Boltzmann Machines
- Incorporating Second-Order Functional Knowledge for Better Option Pricing

Recurrent Neural Networks

Recurrent neural networks are just a special form of shared weights. While "a multilayer perceptron (MLP) can only map from input to output vectors, whereas an RNN can in principle map from the entire history of previous inputs to each output. Indeed, the equivalent result to the universal approximation theory for MLPs is that an RNN with enough hidden units can approximate any measurable sequence-to-sequence mapping to arbitrary accuracy (Hammer, 2000). The key point is that the recurrent connections allow a 'memory' of previous inputs to persist in the network's internal state, which can then be used to influence the network output. The forward pass of an RNN

is the same as that of an MLP with a single hidden layer, except that activations arrive at the hidden layer from both the current external input and the hidden layer activations one step back in time. " -Graves

"If you imagine a neural net as a 2D graph, an RNN is a 3D graph where the topology of every 2D slice is a duplicate of the original non-recurrent network. Every slice has connections going to the next slice, these inter-slice connections also have the same topology. The inter-slice connections represent connections in time, going into the future. So, when you are performing a back-propagation step, you might step into the prior layer, and/or you might also step into the prior time step." – link

Recursive Neural Networks

Recursive neural networks learn data with structural hierarchies, such as text arranged grammatically, much like recurrent neural networks learn data structured by its occurrence in time. Their chief use is in natural-language processing, and they are associated with Richard Socher of Stanford's NLP lab.

- Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank

Reinforcement Learning

Reinforcement learning is a branch of machine learning that is goal oriented; that is, reinforcement learning algorithms have as their objective to maximize a reward, often over the course of many decisions. Unlike deep neural networks, reinforcement learning is not differentiable.

Representation Learning

Representation learning is learning the best representation of the input. A vector, for example, can "represent" an image. Training a neural network will adjust the vector's elements to represent the image better, or lead to better guesses when a neural network is fed the image. The neural net might train to guess the image's name, for instance. Deep learning means that several layers of representations are stacked atop one another, and those representations are increasingly abstract; i.e. the initial, low-level representations are granular, and may represent pixels, while the higher representations will stand for combinations of pixels, and then combinations of combinations, and so forth.

Residual Networks (Resnet)

Microsoft Research used deep Residual Networks to win ImageNet in 2015. ResNets create "shortcuts" across several layers (deep resnets have 150 layers), allowing the net to learn so-called residual mappings. ResNets are like nets with Highway Layers, although they're data independent. Microsoft Research created ResNets by generating by different deep networks automatically and relying on hyperparameter optimization.

- Deep Residual Learning for Image Recognition

Restricted Boltzmann Machine (RBM)

Restricted Boltzmann machines are Boltzmann machines that are constrained to feed input forward symmetrically, which means all the nodes of one layer must connect to all the nodes of the subsequent layer. Stacked RBMs are known as a deep-belief network, and are used to learn how to reconstruct data layer by layer. Introduced by Geoff Hinton, RBMs were partially responsible for the renewed interest in deep learning that began circa 2006. In many labs, they have been replaced with more stable layers such as Variational Auto-encoders.

- A Practical Guide to Training Restricted Boltzmann Machines

RMSProp

RMSProp is an optimization algorithm like Adagrad. In contrast to Adagrad, it relies on a decay term to prevent the learning rate from decreasing too rapidly.

- Optimization Algorithms (Stanford)
- An overview of gradient descent optimization algorithms

Score

Measurement of the overall error rate of the model. The score of the model will be displayed graphically in the web UI or it can be displayed at the console.

Skip-gram

The prerequisite to a definition of skip-grams is one of the ngrams. An n-gram is a contiguous sequence of n items from a given sequence of text or speech. A unigram represents one "item," a bigram two, a trigram three and so forth. Skipgrams are ngrams in which the items are not necessarily contiguous. This can be illustrated best with a few examples. Skipping is a form of noise, in the sense of noising and de-noising, which allows neural nets to better generalize their extraction of features.

Soft-max

Soft-max is a function used as the output layer of a neural network that classifies input. It converts vectors into class probabilities. Soft-max normalizes the vector of scores by first exponentiating and then dividing by a constant.

- A Scalable Hierarchical Distributed Language Model

Stochastic Gradient Descent

Stochastic Gradient Descent optimizes gradient descent and minimizes the loss function during network training.

Stochastic is simply a synonym for "random." A stochastic process is a process that involves a random variable, such as randomly initialized weights. Stochastic derives from the Greek word *stochazesthai*, "to guess or aim at". Stochastic processes describe the evolution of, say, a random set of variables, and as such, they involve some indeterminacy – quite the opposite of having precisely predicted processes that are deterministic and have just one outcome.

The stochastic element of a learning process is a form of search. Random weights represent a hypothesis, an attempt, or a guess that one tests. The results of that search are recorded in the form of a weight adjustment, which effectively shrinks the search space as the parameters move toward a position of less error.

Neural-network gradients are calculated using back-propagation. SGD is usually used with mini batches, such that parameters are updated based on the average error generated by the instances of a whole batch.

Support Vector Machine

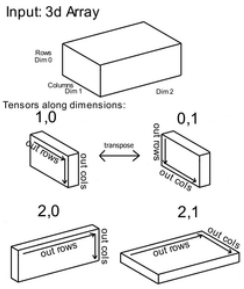
While support-vector machines are not neural networks, they are an important algorithm that deserves explanation:

An SVM is just trying to draw a line through your training points. So, it's just like regular old linear regression except for the following three details: (1) there is an epsilon parameter that means "If the line fits a point to within epsilon then that's good enough; stop trying to fit it and worry about fitting other points." (2) there is a C parameter and the smaller you make it the more you are telling it to find "non-wiggly lines". So, if you run SVR and get some crazy wiggly output that's obviously not right you can often make C smaller and it will stop being crazy. And finally (3) when there are outliers (e.g. bad points that will never fit your line) in your data they will only mess up your result a little bit. This is because SVR only gets upset about outliers in proportion to how far away they are from the line it wants to fit. This is contrasted with normal linear regression which gets upset in proportion to the square of the distance from the line. Regular linear regression worries too much about these bad points. TL;DR: SVR is trying to draw a line that gets within epsilon of all the points. Some points are bad and can't be made to get within epsilon and SVR doesn't get too upset about them whereas other regression methods flip out.

- Reddit

Tensors

Here is an example of tensor along a dimension (TAD):



Vanishing Gradient Problem

The vanishing gradient problem is a challenge the confront back-propagation over many layers. Back-propagation establishes the relationship between a given weight and the error of a neural network. It does so through the chain rule of calculus, calculating how the change in each weight along a gradient affects the change in error. However, in very deep neural networks, the gradient that relates the weight change to the error change can become very small. So small that updates in the net's parameters hardly change the net's guesses and error; so small, in fact, that it is difficult to know in which direction the weight should be adjusted to diminish error. Non-linear activation functions such as sigmoid and tanh make the vanishing gradient problem particularly difficult because the activation function tapers off at both ends. This has led to the widespread adoption of rectified linear units (reLU) for activations in deep nets. It was in seeking to solve the vanishing gradient problem that Sepp Hochreiter and Juergen Schmidhuber invented a form of a recurrent network called an LSTM in the 1990s. The inverse of the vanishing gradient problem, in which the gradient is impossibly small, is the exploding gradient problem, in which the gradient is impossibly large (i.e. changing a weight has too much impact on the error.)

- On the difficulty of training recurrent neural networks

Training

The goal of training is to have your own trained network that you can either use right away or give to other users so that they can use your network completely standalone.

Training is done in steps. Each step takes elements of the teaching dataset in the batch and slowly fits the blocks of each layer with some coefficients so that the overall set of layers can give a result close to the expected output. The activation functions used are in some sense mimicking the human memory process.

After each teaching step, the network is tested for its accuracy using the provided test dataset. At this stage, the network is run with the current internal coefficients and compared with a previous version of itself to know whether it performs better or not. The score of each step gives the effectiveness of the network, and the closer the loss is to zero, the better the network is performing. So, while training your network, one of your goals should be to get that loss value as close to zero as possible.

Transfer Learning

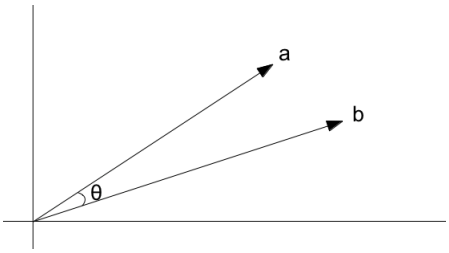
Transfer learning is when a system can recognize and apply knowledge and skills learned in previous domains or tasks to novel domains or tasks. That is, if a model is trained on image data to recognize one set of categories, transfer learning applies if that same model is capable, with minimal additional training, or recognizing a different set of categories. For example, trained on 1,000 celebrity faces, a transfer learning model can be taught to recognize members of your family by swapping in another output layer with the nodes “mom”, “dad”, “elder brother”, “younger sister” and training that output layer on the new classifications.

Vector

A vector is a data structure with at least two components, as opposed to a *scalar*, which has just one. For example, a vector can represent velocity, an idea that combines speed and direction: *wind velocity* = (50mph, 35 degrees North East). A scalar, on the other hand, can represent something with one value like temperature or height: 50 degrees Celsius, 180 centimetres.

Therefore, we can represent two-dimensional vectors as arrows on an x-y graph, with the coordinates x and y each representing one of the vector’s values.

Two vectors can relate to one another mathematically, and similarities between them (and therefore between anything you can vectorize, including words) can be measured with precision.



As you can see, these vectors differ from one another in both their length, or magnitude, and in their angle, or direction. The angle is what concerns us here.

VGG

VGG is a deep convolutional architecture that won the benchmark ImageNet competition in 2014. A VGG architecture is composed of 16–19 weight layers and uses small convolutional filters.

- Very Deep Convolutional Networks for Large-Scale Image Recognition

Xavier Initialization

The Xavier initialization is based on the work of Xavier Glorot and Yoshua Bengio in their paper "Understanding the difficulty of training deep feed-forward neural networks." An explanation can be found here. Weights should be initialized in a way that promotes "learning". The wrong weight initialization will make gradients too large or too small, and make it difficult to update the weights. Small weights lead to small activations, and large weights lead to large ones. Xavier weight initialization considers the distribution of output activations about input activations. Its purpose is to maintain same distribution of activations, so they aren't too small (mean zero but with small variance) or too large (mean zero but with large variance).

Appendix B – Deep Learning Algorithms Cheat Sheet

With new neural network architectures popping up every now and then, it's hard to keep track of them all. Knowing all the abbreviations being thrown around (DCIGN, BiLSTM, DCGAN, anyone?) can be a bit overwhelming at first.

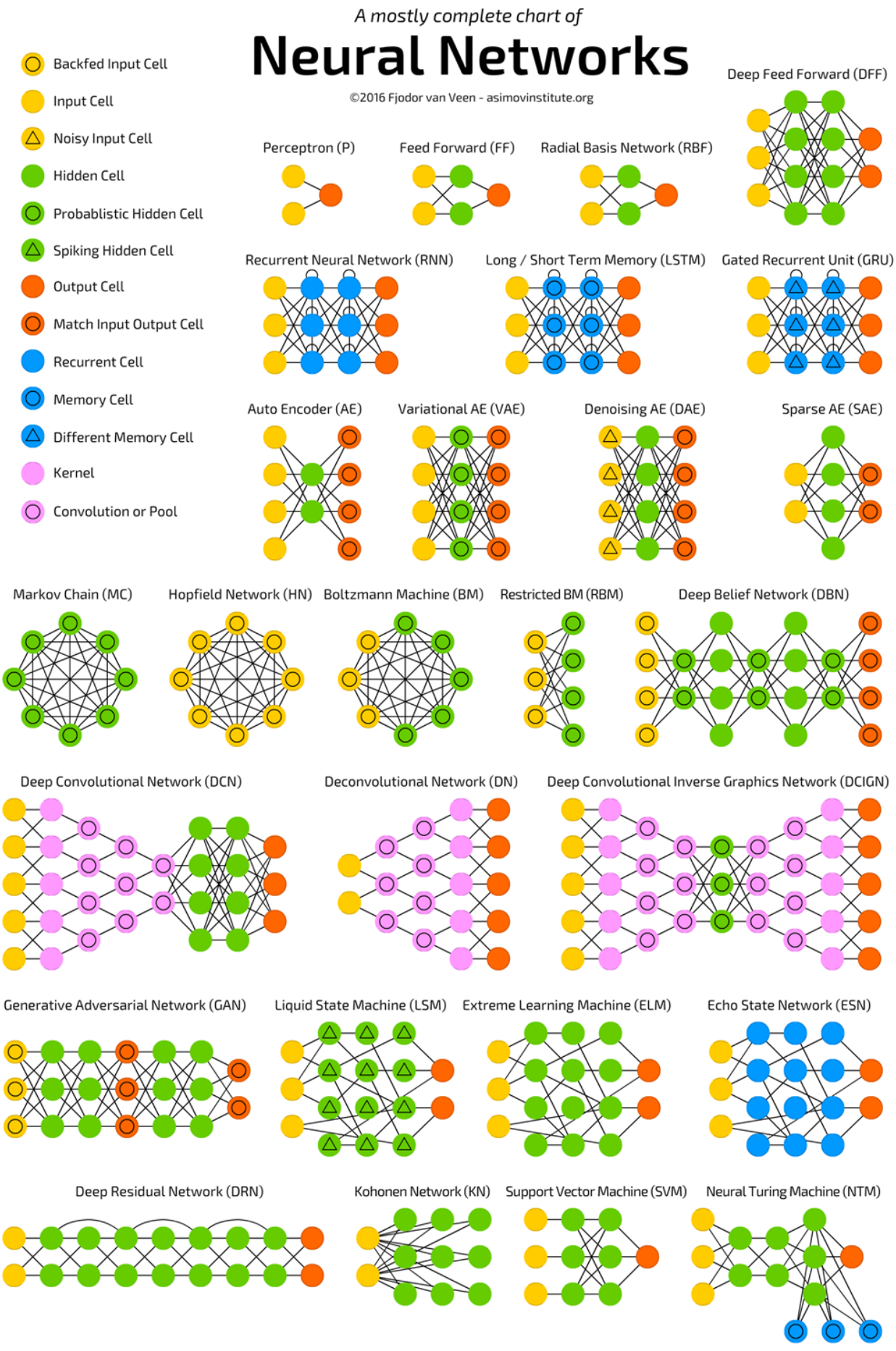
So, I decided to compose a cheat sheet containing many of those architectures. Most of these are neural networks, some are completely different beasts. Though these architectures are presented as novel and unique when I drew the node structures… their underlying relations started to make more sense.

One problem with drawing them as node maps: it doesn't really show how they're used. For example, Variational auto-encoders (VAE) may look just like auto-encoders (AE), but the training process is quite different. The use-cases for trained networks differ even more because VAEs are generators, where you insert noise to get a new sample. AEs, simply map whatever they get as input to the closest training sample they "remember". I should add that this overview is in no way clarifying how each of the different node types works internally (but that's a topic for another day).

It should be noted that while most of the abbreviations used are generally accepted, not all of them are. RNNs sometimes refer to recursive neural networks, but most of the time they refer to recurrent neural networks. That's not the end of it though, in many places you'll find RNN used as a placeholder for any recurrent architecture, including LSTMs, GRUs and even the bidirectional variants. AEs suffer from a similar problem from time to time, where VAEs and DAEs and the like are called simply AEs. Many abbreviations also vary in the amount of "N"s to add at the end, because you could call it a convolutional neural network but also simply a convolutional network (resulting in CNN or CN).

Composing a complete list is practically impossible, as new architectures are invented all the time. Even if published it can still be quite challenging to find them even if you're looking for them, or sometimes you just overlook some. So, while this list may provide you with some insights into the world of AI, please, by no means take this list for being comprehensive; especially if you read this post long after it was written.

For each of the architectures depicted in the picture, I wrote a very, very brief description. You may find some of these to be useful if you're quite familiar with some architectures, but you aren't familiar with a one.


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

Feedforward Neural Networks (FF or FFNN) and Perceptrons (P)

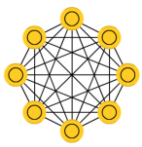
are very straightforward, they feed information from the front to the back (input and output, respectively). Neural networks are often described as having layers, where each layer consists of either input, hidden or output cells in parallel. A layer alone never has connections and in general two adjacent layers are fully connected (every neuron from one layer to every neuron to another layer). The simplest somewhat practical network has two input cells and one output cell, which can be used to model logic gates. One usually trains FFNNs through back-propagation, giving the network paired datasets of "what goes in" and "what we want to have coming out". This is called supervised learning, as opposed to unsupervised learning where we only give it input and let the network fill in the blanks. The error being back-propagated is often some variation of the difference between the input and the output (like MSE or just the linear difference). Given that the network has enough hidden neurons, it can theoretically always model the relationship between the input and output. Practically their use is a lot more limited, but they are popularly combined with other networks to form new networks.

Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological Review 65.6 (1958): 386.
Original Paper PDF

Radial Basis Function (RBF)

Networks are FFNNs with radial basis functions as activation functions. There's nothing more to it. Doesn't mean they don't have their uses, but most FFNNs with other activation functions don't get their own name. This mostly has to do with inventing them at the right time.

Broomhead, David S., and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. No. RSRE-MEMO-4148. ROYAL SIGNALS AND RADAR ESTABLISHMENT MALVERN (UNITED KINGDOM), 1988.
Original Paper PDF

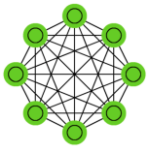


Hopfield Network (HN)

is a network where every neuron is connected to every other neuron; it is a completely entangled plate of spaghetti as even all the nodes function as everything. Each node is input before training, then hidden during training and output afterwards. The networks are trained by setting the value of the neurons to the desired pattern after which the weights can be computed. The weights do not change after this. Once trained for one or more patterns, the network will always converge to one of the learned patterns because the network is only stable in those states. Note that it does not always conform to the desired state (it's not a magic black box sadly). It stabilises in part due to the total "energy" or "temperature" of the network being reduced incrementally during training. Each neuron

has an activation threshold which scales to this temperature, which if surpassed by summing the input causes the neuron to take the form of one of two states (usually -1 or 1, sometimes 0 or 1). Updating the network can be done synchronously or more commonly one by one. If updated one by one, a fair random sequence is created to organise which cells update in what order (fair random being all options (n) occurring exactly once every n items). This is so you can tell when the network is stable (done converging), once every cell has been updated and none of them changed, the network is stable (annealed). These networks are often called associative memory because the converge to the most similar state as the input; if humans see half a table we can image the other half, this network will converge to a table if presented with half noise and half a table.

Hopfield, John J. "Neural networks and physical systems with emergent collective computational abilities." Proceedings of the national academy of sciences 79.8 (1982): 2554-2558. Original Paper PDF



Markov Chains (MC or Discrete-time Markov Chain, DTMC)

are kind of the predecessors to BMs and HNs. They can be understood as follows: from this node where I am now, what are the odds of me going to any of my neighbouring nodes? They are memoryless (i.e. Markov Property) which means that every state you end up in depends completely on the previous state. While not really a neural network, they do resemble neural networks and form the theoretical basis for BMs and HNs. MC isn't always considered neural networks, as goes for BMs, RBMs and HNs. Markov chains aren't always fully connected either.

Hayes, Brian. "First links in the Markov chain." American Scientist 101.2 (2013): 252. Original Paper PDF

Boltzmann Machines (BM)

are a lot like HNs, but: some neurons are marked as input neurons and others remain "hidden". The input neurons become output neurons at the end of a full network update. It starts with random weights and learns through back-propagation, or more recently through contrastive divergence (a Markov chain is used to determine the gradients between two informational gains). Compared to an HN, the neurons mostly have binary activation patterns. As hinted by being trained by MCs, BMs are stochastic networks. The training and running process of a BM is fairly like an HN: one sets the input neurons to certain clamped values after which the network is set free (it doesn't get a sock). While free the cells can get any value and we repetitively go back and forth between the input and hidden neurons. The activation is controlled by a global temperature value, which if lowered lowers the energy of the cells. This lower energy causes their activation patterns to stabilise. The network reaches an equilibrium given the right temperature.

Hinton, Geoffrey E., and Terrence J. Sejnowski. "Learning and relearning in Boltzmann machines." Parallel distributed processing: Explorations in the microstructure of cognition 1 (1986): 282-317.
Original Paper PDF

Restricted Boltzmann Machines (RBM)

are remarkably like BMs (surprise) and therefore also like HNs. The biggest difference between BMs and RBMs is that RBMs are a better usable because they are more restricted. They don't trigger-happily connect every neuron to every other neuron but only connect every different group of neurons to every other group, so no input neurons are directly connected to other input neurons and no hidden to hidden connections are made either. RBMs can be trained like FFNNs with a twist: instead of passing data forward and then back-propagating, you forward pass the data and then backwards pass the data (back to the first layer). After that, you train with forward-and-back-propagation.

Smolensky, Paul. Information processing in dynamical systems: Foundations of harmony theory. No. CU-CS-321-86. COLORADO UNIV AT BOULDER DEPT OF COMPUTER SCIENCE, 1986.
Original Paper PDF

Auto-Encoders (AE)

are somewhat like FFNNs as AEs are more like a different use of FFNNs than a fundamentally different architecture. The basic idea behind auto-encoders is to encode information (as in compress, not encrypt) automatically, hence the name. The entire network always resembles an hourglass-like shape, with smaller hidden layers than the input and output layers. AEs are also always symmetrical around the middle layer(s) (one or two depending on an even or odd number of layers). The smallest layer(s) is are almost always in the middle, the place where the information is most compressed (the chokepoint of the network). Everything up to the middle is called the encoding part, everything after the middle the decoding and the middle (surprise) the code. One can train them using backpropagation by feeding input and setting the error to be the difference between the input and what came out. AEs can be built symmetrically when it comes to weights as well, so the encoding weights are the same as the decoding weights.

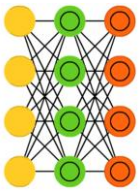
Bourlard, Hervé, and Yves Kamp. "Auto-association by multilayer Perceptrons and singular value decomposition." Biological cybernetics 59.4-5 (1988): 291-294.
Original Paper PDF

Sparse Auto-Encoders (SAE)

are in a way the opposite of AEs. Instead of teaching a network to represent a bunch of information in less "space" or nodes, we try to encode information in more space. So instead of the network converging in the middle and

then expanding back to the input size, we blow up the middle. These types of networks can be used to extract many small features from a dataset. If one were to train an SAE the same way as an AE, you would in almost all cases end up with a pretty useless identity network (as in what comes in is what comes out, without any transformation or decomposition). To prevent this, instead of feeding back the input, we feedback the input plus a sparsity driver. This sparsity driver can take the form of a threshold filter, where only a certain error is passed back and trained, the other error will be "irrelevant" for that pass and set to zero. In a way, this resembles spiking neural networks, where not all neurons fire all the time (and points are scored for biological plausibility).

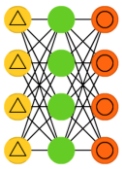
Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann LeCun. "Efficient learning of sparse representations with an energy-based model." Proceedings of NIPS. 2007.
Original Paper PDF



Variational Auto-Encoders (VAE)

have the same architecture as AEs but are "taught" something else: an approximated probability distribution of the input samples. It's a bit back to the roots as they are a bit more closely related to BMs and RBMs. They do however rely on Bayesian mathematics regarding probabilistic inference and independence, as well as a re-parametrisation trick to achieve this different representation. The inference and independence parts make sense intuitively, but they rely on somewhat complex mathematics. The basics come down to this: take influence into account. If one thing happens in one place and something else happens somewhere else, they are not necessarily related. If they are not related, then the error propagation should consider that. This is a useful approach because neural networks are large graphs (in a way), so it helps if you can rule out influence from some nodes to other nodes as you dive into deeper layers.

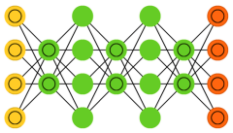
Kingma, Diederik P., and Max Welling. "Auto-encoding Variational Bayes." arXiv preprint arXiv:1312.6114 (2013).
Original Paper PDF



Denoising Auto-Encoders (DAE)

are AEs where we don't feed just the input data, but we feed the input data with noise (like making an image grainier). We compute the error the same way though, so the output of the network is compared to the original input without noise. This encourages the network not to learn details but broader features, as learning smaller features often turns out to be "wrong" due to it constantly changing noise.

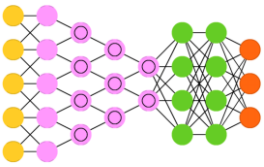
Vincent, Pascal, et al. "Extracting and composing robust features with denoising auto-encoders." Proceedings of the 25th international conference on Machine learning. ACM, 2008.
Original Paper PDF



Deep Belief Networks (DBN)

is the name given to stacked architectures of mostly RBMs or VAEs. These networks have been shown to be effectively trainable stack by stack, where each AE or RBM only must learn to encode the previous network. This technique is also known as greedy training, where greedy means making locally optimal solutions to get to a decent but possibly not optimal answer. DBNs can be trained through contrastive divergence or back-propagation and learn to represent the data as a probabilistic model, just like regular RBMs or VAEs. Once trained or converged to a (more) stable state through unsupervised learning, the model can be used to generate new data. If trained with contrastive divergence, it can even classify existing data because the neurons have been taught to look for different features.

Bengio, Yoshua, et al. "Greedy layer-wise training of deep networks." Advances in neural information processing systems 19 (2007): 153.
Original Paper PDF

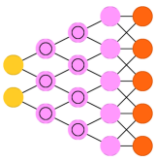


Convolutional Neural Networks (CNN or deep convolutional neural networks, DCNN)

are quite different from most other networks. They are primarily used for image processing but can also be used for other types of input such as for an audio. A typical use case for CNNs is where you feed the network images and the network classifies the data, e.g. it outputs "cat" if you give it a cat picture and "dog" when you give it a dog picture. CNNs tend to start with an input "scanner" which is not intended to parse all the training data at once. For example, to input an image of 200 x 200 pixels, you wouldn't want a layer with 40 000 nodes. Rather, you create a scanning input layer of say 20 x 20 which you feed the first 20 x 20 pixels of the image (usually starting in the upper left corner). Once you passed that input (and possibly use it for training) you feed it the next 20 x 20 pixels: you move the scanner one pixel to the right. Note that one wouldn't move the input 20 pixels (or whatever scanner width) over, you're not dissecting the image into blocks of 20 x 20, but rather you're crawling over it. This input data is then fed through convolutional layers instead of normal layers, where not all nodes are connected to all nodes. Each node only concerns itself with close neighbouring cells (how close depends on the implementation, but usually not more than a few). These convolutional layers also tend to shrink as they become deeper, mostly by easily divisible factors of the input (so 20 would probably go to a layer of 10 followed by a layer of 5). Powers of two are very commonly used here, as they can be divided cleanly and completely by definition: 32, 16, 8, 4, 2, 1. Besides these convolutional layers, they also often feature pooling layers. Pooling is a way to filter out details: a commonly found pooling technique is max pooling, where we take say 2 x 2 pixels and pass on the pixel with the most amount of red. To apply CNNs for audio, you basically feed the input audio waves and inch over the length of the clip, segment by segment. Real-world implementations of CNNs often glue an FFNN to the end to further

process the data, which allows for highly non-linear abstractions. These networks are called DCNNs but the names and abbreviations between these two are often used interchangeably.

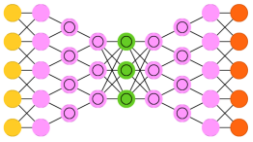
LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.
Original Paper PDF



Deconvolutional Networks (DN)

also called inverse graphics networks (IGNs), are reversed convolutional neural networks. Imagine feeding a network the word "cat" and training it to produce cat-like pictures, by comparing what it generates to real pictures of cats. DNNs can be combined with FFNNs just like regular CNNs, but this is the point where the line is drawn with coming up with new abbreviations. They may be referenced as deep deconvolutional neural networks, but you could argue that when you stick FFNNs to the back and the front of DNNs that you have yet another architecture which deserves a new name. Note that in most applications one wouldn't feed text-like input to the network, more likely a binary classification input vector. Think <0, 1> being cat, <1, 0> being dog and <1, 1> being cat and dog. The pooling layers commonly found in CNNs are often replaced with similar inverse operations, mainly interpolation and extrapolation with biased assumptions (if a pooling layer uses max pooling, you can invent exclusively lower new data when reversing it).

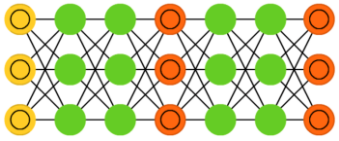
Zeiler, Matthew D., et al. "Deconvolutional networks." Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. IEEE, 2010.
Original Paper PDF



Deep Convolutional Inverse Graphics Networks (DCIGN)

have a somewhat misleading name, as they are VAEs but with CNNs and DNNs for the respective encoders and decoders. These networks attempt to model "features" in the encoding as probabilities, so that it can learn to produce a picture with a cat and a dog together, having only ever seen one of the two in separate pictures. Similarly, you could feed it a picture of a cat with your neighbours' annoying dog on it, and ask it to remove the dog, without ever having done such an operation. Demos have shown that these networks can also learn to model complex transformations on images, such as changing the source of light or the rotation of a 3D object. These networks tend to be trained with back-propagation.

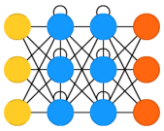
Kulkarni, Tejas D., et al. "Deep convolutional inverse graphics network." Advances in Neural Information Processing Systems. 2015.
Original Paper PDF



Generative Adversarial Networks (GAN)

are a different breed of networks, they are twins: two networks working together, GANs consist of any two networks (although often a combination of FFs and CNNs), with one, tasked to generate content and the other must judge content. The discriminating network receives either training data or generated content from the generative network. How well the discriminating network was able to correctly predict the data source is then used as part of the error for the generating network. This creates a form of competition where the discriminator is getting better at distinguishing real data from generated data and the generator is learning to become less predictable to the discriminator. This works well in part because even quite complex noise-like patterns are eventually predictable but generated content similar in features to the input data is harder to learn to distinguish. GANs can be quite difficult to train, as you don't just have to train two networks (either of which can pose its own problems) but their dynamics need to be balanced as well. If prediction or generation becomes too good compared to the other, a GAN won't converge as there is intrinsic divergence.

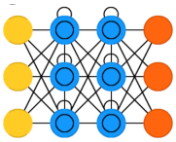
Goodfellow, Ian, et al. "Generative adversarial nets." Advances in Neural Information Processing Systems. 2014. Original Paper PDF



Recurrent Neural Networks (RNN)

are FFNNs with a time twist: they are not stateless; they have connections between passes, connections through time. Neurons are fed information not just from the previous layer but also from themselves from the previous pass. This means that the order in which you feed the input and train the network matters: feeding it "milk" and then "cookies" may yield different results compared to feeding it "cookies" and then "milk". One big problem with RNNs is the vanishing (or exploding) gradient problem where, depending on the activation functions used, information rapidly gets lost over time, just like very deep FFNNs lose information in depth. Intuitively this wouldn't be much of a problem because these are just weights and not neuron states, but the weights through time are where the information from the past is stored; if the weight reaches a value of 0 or 1 000 000, the previous state won't be very informative. RNNs can in principle be used in many fields as most forms of data that don't have a timeline (i.e. unlike sound or video) can be represented as a sequence. A picture or a string of text can be fed one pixel or character at a time, so the time-dependent weights are used for what came before in the sequence, not actually from what happened x seconds before. In general, recurrent networks are a good choice for advancing or completing the information, such as auto-completion.

Elman, Jeffrey L. "Finding structure in time." Cognitive science 14.2 (1990): 179-211. Original Paper PDF

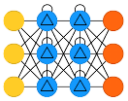


Long / Short Term Memory (LSTM)

networks try to combat the vanishing / exploding gradient problem by introducing gates and an explicitly defined memory cell. These are inspired mostly by circuitry, not so much biology. Each neuron has a memory cell and three gates: input, output and forget. The function of these gates is to safeguard the information by stopping or allowing

the flow of it. The input gate determines how much of the information from the previous layer gets stored in the cell. The output layer takes the job on the other end and determines how much of the next layer gets to know about the state of this cell. The forget gate seems like an odd inclusion at first but sometimes it's good to forget: if it's learning a book and a new chapter begins, it may be necessary for the network to forget some characters from the previous chapter. LSTMs have been shown to be able to learn complex sequences, such as writing like Shakespeare or composing primitive music. Note that each of these gates has a weight to a cell in the previous neuron, so they typically require more resources to run.

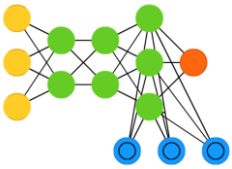
Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural Computation 9.8 (1997): 1735-1780.
Original Paper PDF



Gated Recurrent Units (GRU)

are a slight variation on LSTMs. They have one less gate and are wired slightly differently: instead of an input, output and a forget gate, they have an update gate. This update gate determines both how much information to keep from the last state and how much information to let in from the previous layer. The reset gate functions much like the forget gate of an LSTM but it's located slightly differently. They always send out their full state, they don't have an output gate. In most cases, they function very similarly to LSTMs, with the biggest difference being that GRUs is slightly faster and easier to run (but also slightly less expressive). In practice, these tend to cancel each other out, as you need a bigger network to regain some expressiveness which then, in turn, cancels out the performance benefits. In some cases where the extra expressiveness is not needed, GRUs can outperform LSTMs.

Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modelling." arXiv preprint arXiv:1412.3555 (2014).
Original Paper PDF



Neural Turing Machines (NTM)

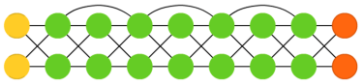
can be understood as an abstraction of LSTMs and an attempt to un-black-box neural networks (and give us some insight into what is going on in there). Instead of coding a memory cell directly into a neuron, the memory is separated. It's an attempt to combine the efficiency and permanency of regular digital storage and the efficiency and expressive power of neural networks. The idea is to have a content-addressable memory bank and a neural network that can read and write from it. The "Turing" in Neural Turing Machines comes from them being Turing complete: the ability to read and write and change state based on what it reads means it can represent anything a Universal Turing Machine can represent.

Graves, Alex, Greg Wayne, and Ivo Danihelka. "Neural Turing machines." arXiv preprint arXiv:1410.5401 (2014).
Original Paper PDF

Bidirectional Recurrent Neural Networks, Bidirectional Long/Short Term Memory Networks And Bidirectional Gated Recurrent Units (BiRNN, BiLSTM and BiGRU Respectively)

are not shown on the chart because they look the same as their unidirectional counterparts. The difference is that these networks are not just connected to the past, but also to the future. As an example, unidirectional LSTMs might be trained to predict the word "fish" by being fed the letters one by one, where the recurrent connections through time remember the last value. A BiLSTM would also be fed the next letter in the sequence on the backward pass, giving it access to future information. This trains the network to fill in gaps instead of advancing information, so instead of expanding an image on the edge, it could fill a hole in the middle of an image.

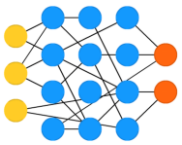
Schuster, Mike, and Kuldip K. Paliwal. "Bidirectional recurrent neural networks." IEEE Transactions on Signal Processing 45.11 (1997): 2673-2681.
Original Paper PDF



Deep Residual Networks (DRN)

are very deep FFNNs with extra connections passing input from one layer to a later layer (often 2 to 5 layers) as well as the next layer. Instead of trying to find a solution for mapping some input to some output across says 5 layers, the network is enforced to learn to map some input to some output + some input. Basically, it adds an identity to the solution, carrying the older input over and serving it freshly to a later layer. It has been shown that these networks are very effective at learning patterns up to 150 layers deep, much more than the regular 2 to 5 layers one could expect to train. However, it has been proven that these networks are just RNNs without the explicit time-based construction and they're often compared to LSTMs without gates.

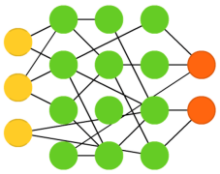
He, Kaiming, et al. "Deep residual learning for image recognition." arXiv preprint arXiv:1512.03385 (2015).
Original Paper PDF



Echo State Networks (ESN)

are yet another different type of (recurrent) network. This one sets itself apart from others by having random connections between the neurons (i.e. not organised into neat sets of layers), and they are trained differently. Instead of feeding input and back-propagating the error, we feed the input, forward it and update the neurons for a while, and observe the output over time. The input and the output layers have a slightly unconventional role as the input layer is used to prime the network and the output layer acts as an observer of the activation patterns that unfold over time. During training, only the connections between the observer and the (soup of) hidden units are changed.

Jaeger, Herbert, and Harald Haas. "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication." science 304.5667 (2004): 78-80.
Original Paper PDF



Extreme Learning Machines (ELM)

are basically FFNNs but with random connections. They look very similar to LSMs and ESNs, but they are not recurrent nor spiking. They also do not use backpropagation. Instead, they start with random weights and train the weights in a single step according to the least-squares fit (lowest error across all functions). This results in a much less expressive network but it's also much faster than backpropagation.

Cambria, Erik, et al. "Extreme learning machines [trends & controversies]." IEEE Intelligent Systems 28.6 (2013): 30-59.
Original Paper PDF

Liquid State Machines (LSM)

are similar soups, looking a lot like ESNs. The real difference is that LSMs are a type of spiking neural networks: sigmoid activations are replaced with threshold functions and each neuron is also an accumulating memory cell. So, when updating a neuron, the value is not set to the sum of the neighbours, but rather added to itself. Once the threshold is reached, it releases its' energy to other neurons. This creates a spiking like a pattern, where nothing happens for a while until a threshold is suddenly reached.

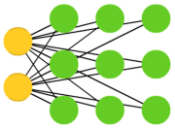
Maass, Wolfgang, Thomas Natschläger, and Henry Markram. "Real-time computing without stable states: A new framework for neural computation based on perturbations." Neural Computation 14.11 (2002): 2531-2560.
Original Paper PDF

Support Vector Machines (SVM)

find optimal solutions for classification problems. Classically they were only capable of categorising linearly separable data; say finding which images are of Garfield and which of Snoopy, with any other outcome not being possible. During training, SVMs can be thought of as plotting all the data (Garfields and Snoopys) on a graph (2D) and figuring out how to draw a line between the data points. This line would separate the data so that all Snoopys are on one side and the Garfields on the other. This line moves to an optimal line in such a way that the margins between the data points and the line are maximised on both sides. Classifying new data would be done by plotting a point on this graph and simply looking on which side of the line it is (Snoopy side or Garfield side). Using the kernel trick, they can be taught to classify n-dimensional data. This entails plotting points in a 3D plot, allowing it

to distinguish between Snoopy, Garfield AND Simon's cat, or even higher dimensions distinguishing even more cartoon characters. SVMs are not always considered neural networks.

Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20.3 (1995): 273-297.
Original Paper PDF



Kohonen Networks (KN, Also Self-Organising (Feature) Map, SOM, SOFM)

KNs utilise competitive learning to classify data without supervision. Input is presented to the network, after which the network assesses which of its neurons most closely match that input. These neurons are then adjusted to match the input even better, dragging along their neighbours in the process. How much the neighbours are moved depends on the distance of the neighbours to the best matching units. KNs are sometimes not considered neural networks either.

Kohonen, Teuvo. "Self-organized formation of topologically correct feature maps." Biological cybernetics 43.1 (1982): 59-69.
Original Paper PDF

Picklist of Algorithms

Insert VISIO

Type	Method	Output variable	Description	Pros	Cons
Linear	Linear regression	Continuous	The "Best Fit" Line through all data points. Predictions are Numerical	○ Easy to understand, you clearly see what the biggest driver of the model are	○ Sometimes too simple to capture complex relationship between variables ○ Overfitting Model Tendency
	Logistic regression	Continuous in range [0,1]	The adaption of linear regression to problems of classification e.g. Yes/No, True/False, Black/White	○ Easy to understand ○ Low variance ○ Provides probability of outcomes ○ Works well with diagonal decision boundaries	○ Sometimes too simple to capture complex relationship between variables ○ Overfitting Model Tendency

Tree-Based	Decision trees	Continuous or discrete	A graph that uses a branching method to match all possible outcomes of a decision	○ Easy to understand ○ Easy to interpret visually ○ Can easily handle categorical features ○ Works well with boundaries parallel to feature axis	○ Not often used on its own for prediction because it's also often too simple and not powerful enough for complex data ○ Prone to overfitting
	Random Forest	Continuous or discrete	Takes the average of many decision trees, each of which is made with a sample of the data. Each tree is weaker than a full decision tree, but by combining them we get a better overall performance	○ A sort of "Wisdom of Crowd". ○ Tends to result in very high-quality models. ○ Fast to Train	○ Can be slow to output predictions relative to other algorithms ○ Not easy to understand the predictions
	Gradient Boosting	Continuous or discrete	Uses even weaker decision trees, that are increasingly focused on "hard" examples	○ High Performing	○ A small change in the feature set or training set can create radical changes in the model ○ Not easy to understand the predictions
	k-means	Discrete	???	○ Works well with large amount of data ○ Easy to implement and interpret	Poor performance for non-hyper-spherical clusters Results depend on selection of K

Deep Learning	Neural Networks	Any	Mimics the behaviour of the brain. NNets are interconnected neurons that pass messages to each other. Deep Learning uses several layers of NNets put one after another	○ Can handle complex tasks – no other algorithms come close to results	○ Very slow to train, because they have so many layers, requires lot of resources ○ Almost impossible to understand predictions

Appendix C – Deep Neural-network Pseudo Configuration

Steps to configure Neural Networks (pseudo-code)

Create DNNs Configuration:

- Multi Layer Configuration conf = new NeuralNet Configuration // Start Configuration
- Seed = Random_Seed // Random seed for same weight Initialization everytime
- Gradient Normalization = GradientNormalization.ClipElementWiseAbsoluteValue // Error / Optimisation Function
- Gradient Normalization Threshold = threshold_value // Threshold value for Normalization / Error / Optimisation function
- Iterations = num_of_iterations // Number of Iteration over every batch of Dataset
- Epochs = num_of_epochs // number of total run of whole data sets
- Momentum = momentum_value_for_convergence // momentum for faster convergence
- Optimization Algorithm = OptimizationAlgorithm.CONJUGATE_GRADIENT // choose optimisation algorithm
- list = number_of_layers // number of layers in DNN
- layer = LAYER_INDEX, LAYER_NAME, LAYER_TYPE = LossFunction.NEGATIVELOGLIKELIHOOD, ACTIVATION_FUNCTION, INPUT, OUPUT // define each layer as chained
- pretrain = TRUE|FALSE // pre-train true on for RBMs…
- backprop = TRUE|FALSE // backward error learning true or false
- build // build the configuration

- MultiLayer Network model = new Multi Layer Network conf // initialize the ANN
- Model Fit on Data // run model on data for learning and verification
- Model Eval data and ouput // evaluate the effectiveness of model over date with epocs

Appendix D – R vs Python for Data Science – Raw (Needs tiding up and verification from Robin)

If you're interested in a career in data, and you're familiar with the set of skills you'll need to master, you know that Python and R are two of the most popular languages for data analysis. When it comes to data analysis, both Python and R are simple (and free) to install and relatively easy to get started with. If you're a new to the world of data science and don't have experience in either language, or with programming in general, it makes sense to be unsure whether to learn R or Python first.

To get a better background on this discussion, we recommend an excellent presentation on recent advances of both languages by Eduardo Ariño de la Rubia, the Chief Data Scientist at Domino Data Lab. It shows how both Python and R have progressed so far. Both languages have become well rounded for data science. Some people point to traditional weaknesses of each language (e.g. data visualization in Python or data wrangling in R), but thanks to recent packages like **Altair** for Python and **dplyr** for R, those weaknesses have been alleviated.

This section is a summary of the modern advances discussed in the video.

The Case for Python

Key quote: *"I have this hope that there is a better way. Higher-level tools that actually let you see the structure of the software more clearly will be of tremendous value."* - Guido van Rossum

Guido van Rossum was the creator of the Python programming language.

Why Python is Great for Data Science

- Python was released in 1989. It has been around for a long time, and it has object-oriented programming baked in.
- IPython / Jupyter's notebook IDE is excellent.
- There's a large ecosystem. For example, Scikit-Learn's page receives 150,000 - 160,000 unique visitors per month.
- There's Anaconda from Continuum Analytics, making package management very easy.
- The Pandas library makes it simple to work with data frames and time series data.

Advances in Modern Python for Data Science

1. Collecting Data

Feather (Fast reading and writing of data to disk)

- Fast, lightweight, easy-to-use binary format for filetypes
- Makes pushing data frames in and out of memory as simply as possible

- Language agnostic (works across Python and R)
- High read and write performance (600 MB/s vs 70 MB/s of CSVs)
- Great for passing data from one language to another in your pipeline

Ibis (Pythonic way of accessing datasets)

- Bridges the gap between local Python environments and remote storages like Hadoop or SQL
- Integrates with the rest of the Python ecosystem

ParaText (Fastest way to get fixed records and delimited data off of disk and into RAM)

- C++ library for reading text files in parallel on multi-core machines
- Integrates with Pandas: paratext.load_csv_to_pandas("data.csv")
- Enables CSV reading of up to 2.5GB a second
- A bit difficult to install

bcolz (Helps you deal with data that's larger than your RAM)

- Compressed columnar storage
- You can define a Pandas-like data structure, compress it, and store it in memory
- Helps get around the performance bottleneck of querying from slower memory

2. Data Visualization

Altair (Like a Matplotlib 2.0 that's much more user friendly)

- You can spend more time understanding your data and its meaning.
- Altair's API is simple, friendly and consistent.
- Create beautiful and effective visualizations with a minimal amount of code.
- Takes a tidy DataFrame as the data source.
- Data is mapped to visual properties using the group-by operation of Pandas and SQL.
- Primarily for creating static plots.

Bokeh (Reusable components for the web)

- Interactive visualization library that targets modern web browsers for presentation.
- Able to embed interactive visualizations.

- D3.js for Python, except better.
- Already has a big gallery that you can borrow steal from.

Geoplotlib (Interactive maps)

- Extremely clean and simple way to create maps.
- Can take a simple list of names, latitudes, and longitudes as input.

3. Cleaning & Transforming Data

Blaze (NumPy for big data)

- Translates a NumPy / Pandas-like syntax to data computing systems.
- The same Python code can query data across a variety of data storage systems.
- Good way to future-proof your data transformations and manipulations.

xarray (Handles n-dimensional data)

- N-dimensional arrays of core pandas data structures (e.g. if the data has a time component as well).
- Multi-dimensional Pandas dataframes.

Dask (Parallel computing)

- Dynamic task scheduling system.
- "Big Data" collections like parallel arrays, dataframes, and lists that extend common interfaces like NumPy, Pandas, or Python iterators to larger-than-memory or distributed environments.

4. Modeling

Keras (Simple deep learning)

- Higher level interface for Theano and Tensorflow
- We wrote a complete Keras tutorial for beginners

PyMC3 (Probabilistic programming)

- Contains the most high end research from labs in academia
- Powerful Bayesian statistical modeling

The Case for R

Key quote: *"There should be an interface to the very best numerical algorithms available."* - John Chambers

John Chambers actually created S, the precursor to R, but the spirit of R is the same.

Why R is Great for Data Science

- R was created in 1992, after Python, and was therefore able to learn from Python's lessons.
- Rcpp makes it very easy to extend R with C++.
- RStudio is a mature and excellent IDE.
- (Our note) CRAN is a candyland filled with machine learning algorithms and statistical tools.
- (Our note) The Caret package makes it easy to use different algorithms from 1 single interface, much like what Scikit-Learn has done for Python

Advances in Modern R for Data Science

1. Collecting Data

Feather (Fast reading and writing of data to disk)

- Same as for Python

Haven (Interacts with SAS, Stata, SPSS data)

- Reads SAS and brings it into a dataframe

Readr (Reimplements read.csv into something better)

- read.csv sucks because it takes strings into factors, it's slow, etc
- Creates a contract for what the data features should be, making it more robust to use in production
- Much faster than read.csv

JsonLite (Handles JSON data)

- Intelligently turns JSON into matrices or dataframes

2. Data Visualization

ggplot2 (ggplot2 was recently massively upgraded)

- Recently had a very significant upgrade (to the point where old code will break)
- You can do faceting and zoom into facets

htmlwidgets (Reusable components)

- Brings of the best of JavaScript visualization to R
- Has a fantastic gallery you can borrow steal from

Leaflet (Interactive maps for the web)

- Nice Javascript maps that you can embed in web applications

Tilegramsr (Proportional maps)

- Create maps that are proportional to the population
- Makes it possible to create more interesting maps than those that only highlight major cities due to population density

3. Cleaning & Transforming Data

Dplyr (Swiss army chainsaw)

- The way R should've been from the first place
- Has a bunch of amazing joins
- Makes data wrangling much more humane

Broom (Tidy your models)

- Fixes model outputs (gets around the weird incantations needed to see model coefficients)
- tidy, augment, glance

Tidy_text (Text as tidy data)

- Text mining using dplyr, ggplot2, and other tidy tools
- Makes natural language processing in R much easier

4. Modeling

MXNet (Simple deep learning)

- Intuitive interface for building deep neural networks in R
- Not quite as nice as Keras

TensorFlow

- Now has an interface in R

Our Recommendation

As you can see, both languages are actively being developed and have an impressive suite of tools already. It sounds cliché to say this, but there's no one-size-fits-all answer.

If you're just starting out, one simple way to choose would be based on your comfort zone. For example, if you come from a C.S./developer background, you'll probably feel more comfortable with Python. On the other hand, if you come from a statistics/analyst background, R will likely be more intuitive.

Illustrate differences through code samples

This section aims to look at the languages more objectively. We'll analyse a dataset side by side in Python and R, and show what code is needed in both languages to achieve the same result. This will let us understand the strengths and weaknesses of each language without the conjecture.

We'll be analysing a dataset of NBA players and their performance in the 2013-2014 season. You can download the file here. For each step in the analysis, we'll show the Python and R code, along with some explanation and discussion of the different approaches. Without further ado, let's get this head to head Python vs R matchup started!

Importing a CSV

R

```
nba <- read.csv("nba_2013.csv")
```

Python

```
import pandas
nba = pandas.read_csv("nba_2013.csv")
```

The above code will load the csv file nba_2013.csv, which contains data on NBA players from the 2013-2014 season, into the variable nba in both languages. The only real difference is that in Python, we need to import the pandas library to get access to Dataframes. Dataframes are available in both R and Python, and are two-dimensional arrays (matrices) where each column can be of a different datatype. At the end of this step, the csv file has been loaded by both languages into a dataframe.

Finding the number of rows

R

```
dim(nba)
 [1] 481  31
```

Python

```
nba.shape
 (481, 31)
```

This prints out the number of players and the number of columns in each. We have 481 rows, or players, and 31 columns containing data on the players.

Looking at the first row of the data

R

```
head(nba, 1)
 player pos age bref_team_id
1 Quincy Acy  SF  23         TOT
[output truncated]
```

Python

```
nba.head(1)
 player pos  age bref_team_id
0  Quincy Acy  SF   23         TOT
[output truncated]
```

This is pretty much identical. Both print out the first row of the data, and the syntax is very similar. Python is more object-oriented here, and head is a method on the dataframe object, and R has a separate head function. This is a common theme you'll see as you start to do analysis with these languages, where Python is more object-oriented, and R is more functional.

Find the average of each statistic

Let's find the average value for each statistic. The columns, as you can see, have names like fg (field goals made), and ast (assists). These are the season statistics for the player. If you want a fuller explanation of all the stats, look here.

R

```
sapply(nba, mean, na.rm=TRUE)
 player NA
pos NA
age 26.5093555093555
bref_team_id NA
[output truncated]
```

Python

```
nba.mean()
 age          26.509356
g            53.253638
gs           25.571726
[output truncated]
```

There are some major differences in approach here. In both, we're applying a function across the dataframe columns. In python, the mean method on dataframes will find the mean of each column by default.

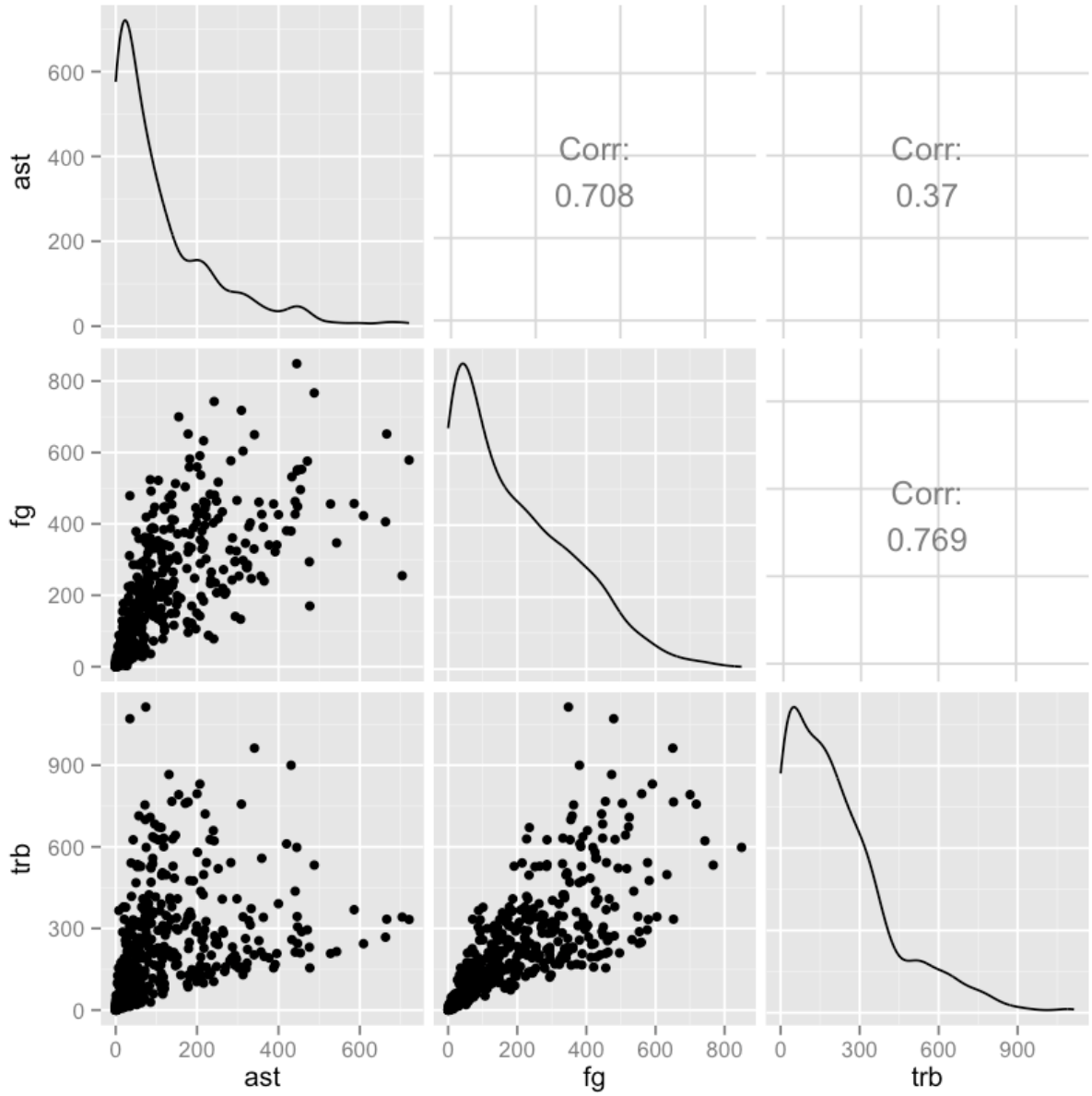
In R, taking the mean of string values will just result in NA — not available. However, we do need to ignore NA values when we take the mean (requiring us to pass na.rm=TRUE into the mean function). If we don't, we end up with NA for the mean of columns like x3p.. This column is three point percentage. Some players didn't take three point shots, so their percentage is missing. If we try the mean function in R, we get NA as a response, unless we specify na.rm=TRUE, which ignores NA values when taking the mean. The .mean() method in Python already ignores these values by default.

Make pairwise scatterplots

One common way to explore a dataset is to see how different columns correlate to others. We'll compare the ast, fg, and trb columns.

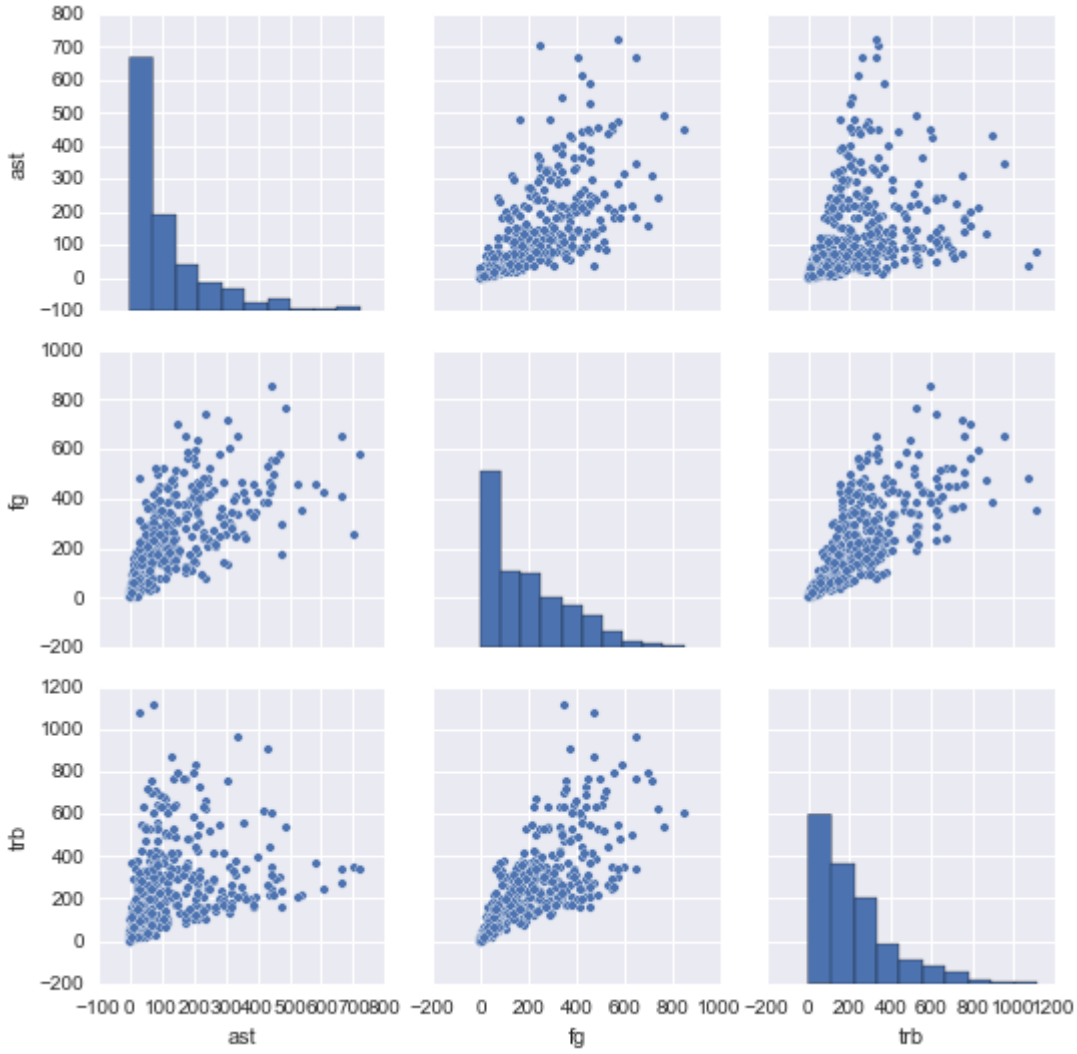
R

```
library(GGally)
ggpairs(nba[,c("ast", "fg", "trb")])
```



Python

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.pairplot(nba[["ast", "fg", "trb"]])
plt.show()
```



We get very similar plots in the end, but this shows how the R data science ecosystem has many smaller packages (GGally is a helper package for ggplot2, the most-used R plotting package), and many more visualization packages in general. In Python, matplotlib is the primary plotting package, and seaborn is a widely used layer over matplotlib. With visualization in Python, there is usually one main way to do something, whereas in R, there are many packages supporting different methods of doing things (there are at least a half dozen packages to make pair plots, for instance).

Make clusters of the players

One good way to explore this kind of data is to generate cluster plots. These will show which players are most similar.

R

```
library(cluster)
set.seed(1)
isGoodCol <- function(col){
```

```
  sum(is.na(col)) == 0 && is.numeric(col)
}
goodCols <- sapply(nba, isGoodCol)
clusters <- kmeans(nba[,goodCols], centers=5)
labels <- clusters$cluster
```

Python

```
from sklearn.cluster import KMeans
kmeans_model = KMeans(n_clusters=5, random_state=1)
good_columns = nba._get_numeric_data().dropna(axis=1)
kmeans_model.fit(good_columns)
labels = kmeans_model.labels_
```

In order to cluster properly, we remove any non-numeric columns, or columns with missing values (NA, Nan, etc). In R, we do this by applying a function across each column, and removing it if it has any missing values or isn't numeric. We then use the cluster package to perform k-means and find 5 clusters in our data. We set a random seed using set.seed to be able to reproduce our results.

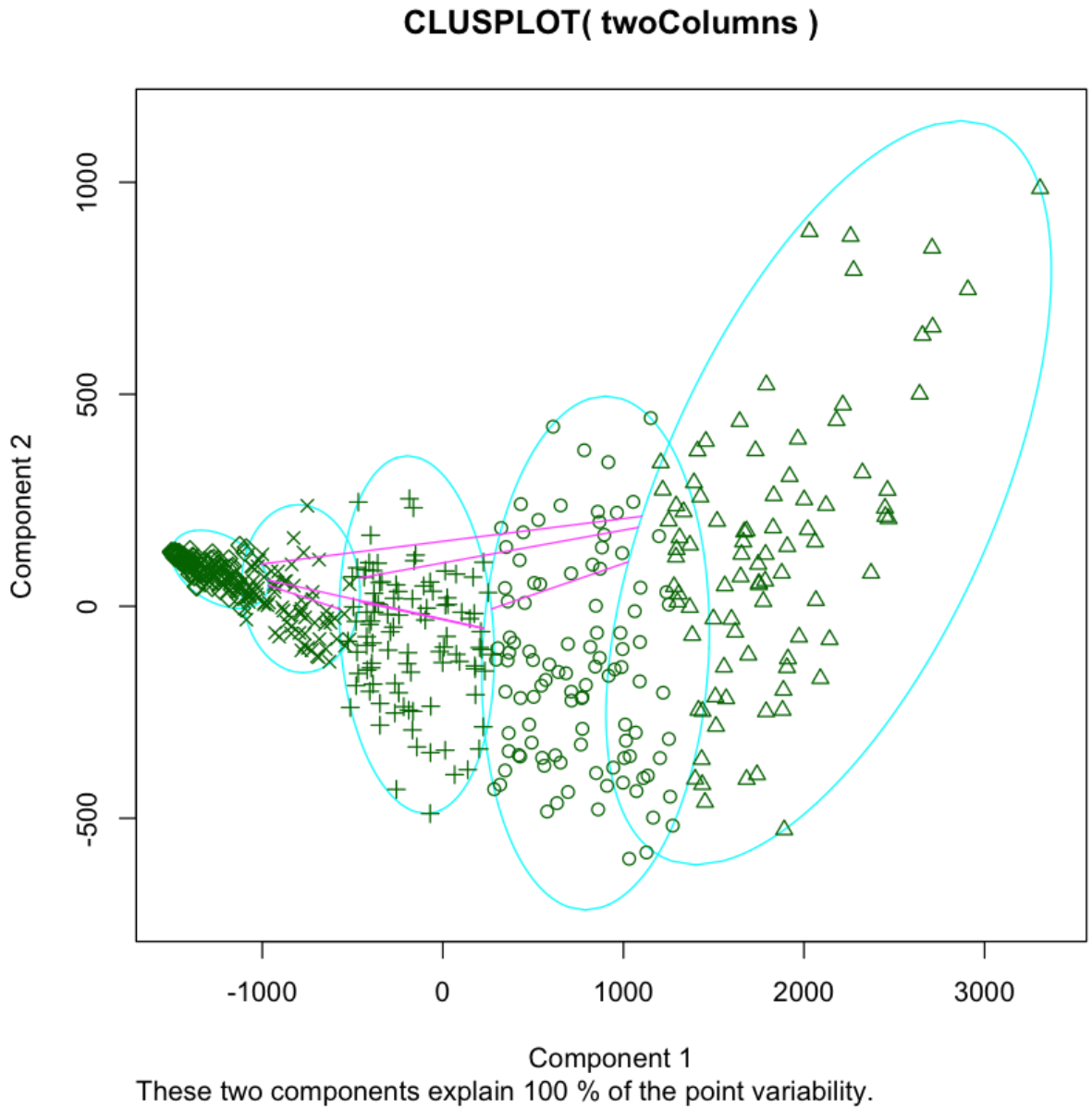
In Python, we use the main Python machine learning package, scikit-learn, to fit a k-means clustering model and get our cluster labels. We perform very similar methods to prepare the data that we used in R, except we use the get_numeric_data and dropna methods to remove non-numeric columns and columns with missing values.

Plot players by cluster

We can now plot out the players by cluster to discover patterns. One way to do this is to first use PCA to make our data 2-dimensional, then plot it, and shade each point according to cluster association.

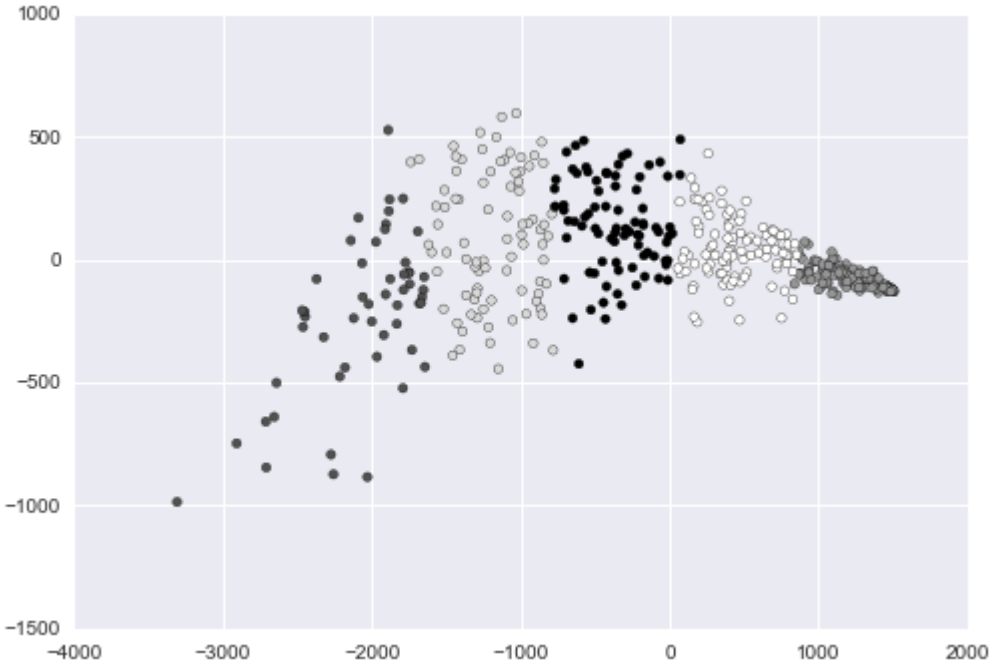
R

```
nba2d <- prcomp(nba[,goodCols], center=TRUE)
twoColumns <- nba2d$x[,1:2]
clusplot(twoColumns, labels)
```



Python

```
from sklearn.decomposition import PCA
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(good_columns)
plt.scatter(x=plot_columns[:,0], y=plot_columns[:,1], c=labels)
plt.show()
```



Made a scatter plot of our data, and shaded or changed the icon of the data according to cluster. In R, the clusplot function was used, which is part of the cluster library. We performed PCA via the pccomp function that is builtin to R.

With Python, we used the PCA class in the scikit-learn library. We used matplotlib to create the plot.

Split into training and testing sets

If we want to do supervised machine learning, it's a good idea to split the data into training and testing sets so we don't overfit.

R

```
trainRowCount <- floor(0.8 * nrow(nba))
set.seed(1)
trainIndex <- sample(1:nrow(nba), trainRowCount)
train <- nba[trainIndex,]
test <- nba[-trainIndex,]
```

Python

```
train = nba.sample(frac=0.8, random_state=1)
test = nba.loc[~nba.index.isin(train.index)]
```

You'll notice that R has many more data-analysis focused builtins, like floor, sample, and set.seed, whereas these are called via packages in Python (math.floor, random.sample, random.seed). In Python, the recent version of

pandas came with a sample method that returns a certain proportion of rows randomly sampled from a source dataframe — this makes the code much more concise. In R, there are packages to make sampling simpler, but aren't much more concise than using the built-in sample function. In both cases, we set a random seed to make the results reproducible.

Univariate linear regression

Let's say we want to predict number of assists per player from field goals made per player.

R

```
fit <- lm(ast ~ fg, data=train)
predictions <- predict(fit, test)
```

Python

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(train[["fg"]], train["ast"])
predictions = lr.predict(test[["fg"]])
```

Scikit-learn has a linear regression model that we can fit and generate predictions from. R relies on the built-in lm and predict functions. predict will behave differently depending on the kind of fitted model that is passed into it — it can be used with a variety of fitted models.

Calculate summary statistics for the model

R

```
summary(fit)
 Call:
lm(formula = ast ~ fg, data = train)

Residuals:
Min      1Q  Median      3Q     Max
-228.26  -35.38  -11.45   11.99  559.61
[output truncated]
```

Python

```
import statsmodels.formula.api as sm
model = sm.ols(formula='ast ~ fga', data=train)
fitted = model.fit()
```

```
fitted.summary()
 OLS Regression Results

Dep. Variable: ast
 R-squared: 0.568
 Model: OLS
 Adj. R-squared: 0.567
 [output truncated]
```

If we want to get summary statistics about the fit, like r-squared value, we'll need to do a bit more in Python than in R. With R, we can use the builtin summary function to get information on the model. With Python, we need to use the statsmodels package, which enables many statistical methods to be used in Python. We get similar results, although generally it's a bit harder to do statistical analysis in Python, and some statistical methods that exist in R don't exist in Python.

Fit a random forest model

Our linear regression worked well in the single variable case, but we suspect there may be nonlinearities in the data. Thus, we want to fit a random forest model.

R

```
library(randomForest)
predictorColumns <- c("age", "mp", "fg", "trb", "stl", "blk")
rf <- randomForest(train[predictorColumns], train$ast, ntree=100)
predictions <- predict(rf, test[predictorColumns])
```

Python

```
from sklearn.ensemble import RandomForestRegressor
predictor_columns = ["age", "mp", "fg", "trb", "stl", "blk"]
rf = RandomForestRegressor(n_estimators=100, min_samples_leaf=3)
rf.fit(train[predictor_columns], train["ast"])
predictions = rf.predict(test[predictor_columns])
```

The main difference here is that we needed to use the randomForest library in R to use the algorithm, whereas it was built in to scikit-learn in Python. scikit-learn has a unified interface for working with many different machine learning algorithms in Python, and there's usually only one main implementation of each algorithm in Python. With R, there are many smaller packages containing individual algorithms, often with inconsistent ways to access them. This results in a greater diversity of algorithms (many have several implementations, and many are fresh out of research labs), but with a bit of a usability hit.

Calculate error

Now that we've fit two models, let's calculate error. We'll use MSE.

R

```
mean((test["ast"] - predictions)^2)
 4573.86778567462
```

Python

```
from sklearn.metrics import mean_squared_error
mean_squared_error(test["ast"], predictions)
 4166.9202475632374
```

In Python, the scikit-learn library has a variety of error metrics that we can use. In R, there are likely some smaller libraries that calculate MSE, but doing it manually is pretty easy in either language. There's a small difference in errors that almost certainly due to parameter tuning, and isn't a big deal.

Download a webpage

Now that we have data on NBA players from 2013-2014, let's scrape some additional data to supplement it. We'll just look at one box score from the NBA Finals here to save time.

R

```
library(RCurl)
url <- "http://www.basketball-reference.com/boxscores/201506140GSW.html"
data <- readLines(url)
```

Python

```
import requests
url = "http://www.basketball-reference.com/boxscores/201506140GSW.html"
data = requests.get(url).content
```

In Python, the requests package makes downloading web pages easy, with a consistent API for all request types. In R, RCurl provides a similarly simple way to make requests. Both download the webpage to a character datatype. Note: this step is unnecessary for the next step in R, but is shown for comparisons's sake.

Extract player box scores

Now that we have the web page, we'll need to parse it to extract scores for players.

R

```
library(rvest)
page <- read_html(url)
table <- html_nodes(page, ".stats_table")[3]
```

```
rows <- html_nodes(table, "tr")
cells <- html_nodes(rows, "td a")
teams <- html_text(cells)

extractRow <- function(rows, i){
   if(i == 1){
      return
   }
   row <- rows[i]
   tag <- "td"
   if(i == 2){
      tag <- "th"
   }
   items <- html_nodes(row, tag)
   html_text(items)
}

scrapeData <- function(team){
   teamData <- html_nodes(page, paste("#",team,"_basic", sep=""))
   rows <- html_nodes(teamData, "tr")
   lapply(seq_along(rows), extractRow, rows=rows)
}

data <- lapply(teams, scrapeData)
```

Python

```
from bs4 import BeautifulSoup
import re
soup = BeautifulSoup(data, 'html.parser')
box_scores = []
for tag in soup.find_all(id=re.compile("[A-Z]{3,}_basic")):
   rows = []
   for i, row in enumerate(tag.find_all("tr")):
      if i == 0:
         continue
      elif i == 1:
         tag = "th"
      else:
         tag = "td"
      row_data = [item.get_text() for item in row.find_all(tag)]
      rows.append(row_data)
```

```
box_scores.append(rows)
```

This will create a list containing two lists, the first with the box score for CLE, and the second with the box score for GSW. Both contain the headers, along with each player and their in-game stats. We won't turn this into more training data now, but it could easily be transformed into a format that could be added to our nba dataframe.

The R code is more complex than the Python code, because there isn't a convenient way to use regular expressions to select items, so we have to do additional parsing to get the team names from the HTML. R also discourages using for loops in favor of applying functions along vectors. We use lapply to do this, but since we need to treat each row different depending on whether it's a header or not, we pass the index of the item we want, and the entire rows list into the function.

We use rvest, a new and widely used R web scraping package to extract the data we need. Note that we can pass a url directly into rvest, so the last step wasn't needed in R.

In Python, we use BeautifulSoup, the most commonly used web scraping package. It enables us to loop through the tags and construct a list of lists in a straightforward way.

Python vs R in Conclusion

We've looked at how to analyse a dataset with R and Python. There are many tasks we didn't dive into, such as persisting the results of our analysis, sharing the results with others, testing and making things production-ready, and making more visualizations. We'll dive into these later, which will let us make some more definitive conclusions. For now, here's what we can say:

R is more functional, Python is more object-oriented

As we saw from functions like lm, predict, and others, R lets functions do most of the work. Contrast this to the LinearRegression class in Python, and the sample method on dataframes.

R has more data analysis built-ins, Python relies on packages

When we looked at summary statistics, we could use the summary built-in function in R, but had to import the statsmodels package in Python. The dataframe is a built-in construct in R, but must be imported via the pandas package in Python.

Python has "main" packages for data analysis tasks, R has a larger ecosystem of small packages

With Python, we can do linear regression, random forests, and more with the scikit-learn package. It offers a consistent API, and is well-maintained. In R, we have a greater diversity of packages, but also greater fragmentation and less consistency (linear regression is a builtin, lm, randomForest is a separate package, etc).

R has more statistical support in general

R was built as a statistical language, and it shows. statsmodels in Python and other packages provide decent coverage for statistical methods, but the R ecosystem is far larger.

It's usually more straightforward to do non-statistical tasks in Python

With well-maintained libraries like BeautifulSoup and requests, web scraping in Python is far easier than in R. This applies to other tasks that we didn't look into closely, like saving to databases, deploying web servers, or running complex workflows.

There are many parallels between the data analysis workflow in both

There are clear points of inspiration between both R and Python (pandas Dataframes were inspired by R data-frames, the *rvest* package was inspired by *BeautifulSoup*), and both ecosystems continue to grow stronger. It's remarkable how similar the syntax and approaches are for many common tasks in both languages.

More comparison using another data set:

Comparative analysis of genome data

To directly compare R and Python, I am following Zhuyi Xue's "A Comprehensive Introduction To Your Genome With the SciPy Stack" (with some minor tweaks here and there). He gives a nice introduction to the data, so I will not repeat it here but focus on the comparison between the code lines.

For R, I am working with RStudio, for Python with Anaconda and Spyder.

***Python*:**

For this analysis, we need the SciPy stack with pandas for data wrangling and matplotlib for visualisation. Anaconda already comes with all these packages that we need. Zhuyi Xue first imports pandas as "pd", so that we can call pandas function by prefixing them with "pd.".

```
import pandas as pd
```

R:

While the code could be replicated with base R, I prefer dplyr for data wrangling and ggplot2 for visualisation.

```
library(dplyr)
library(ggplot2)
```

Reading in data

Reading in data is straight forward in both R and Python. The code we need to read in the file is comparable between R and Python. Zhuyi Xue specified "compression = 'gzip'", but this would not have been necessary as the default is to infer it from the file suffix.

One big difference in the general syntax we can see here too: Boolean true/false values are written in all caps in R (TRUE/FALSE), while Python uses first letter capitalisation (True/False). The same prinicple applies to "none".

Python:

```
df = pd.read_csv('Homo_sapiens.GRCh38.85.gff3.gz',
                 compression = 'gzip',
                 sep = '\t',
                 comment = '#',
                 low_memory = False,
                 header = None,
                 names = ['seqid', 'source', 'type', 'start', 'end', 'score', 'strand', 'phase', 'attributes'])
df.head()
```

R:

```
df <- read.csv("~/Documents/Github/Homo_sapiens.GRCh38.85.gff3.gz",
         header = FALSE,
         sep = "\t",
         col.names = c('seqid', 'source', 'type', 'start', 'end', 'score', 'strand', 'phase', 'attributes'),
         comment.char = "#")
head(df)
```

```
##   seqid source            type start      end   score strand phase
## 1     1 GRCh38      chromosome     1 248956422       .      .      .
## 2     1      . biological_region 10469    11240 1.3e+03      .      .
## 3     1      . biological_region 10650    10657   0.999      +      .
## 4     1      . biological_region 10655    10657   0.999      -      .
## 5     1      . biological_region 10678    10687   0.999      +      .
## 6     1      . biological_region 10681    10688   0.999      -      .
##                                attributes
## 1 ID=chromosome:1;Alias=CM000663.2,chr1,NC_000001.11
## 2         external_name=oe %3D 0.79;logic_name=cpg
## 3                         logic_name=eponine
## 4                         logic_name=eponine
## 5                         logic_name=eponine
## 6                         logic_name=eponine
```

Examining data

- listing unique strings

The first thing we want to know from the data is how many unique entries there are in the “seqid” column.

Here, we can already see the main difference in syntax between R and Python:

Python concatenates the object name (“df) with the column name and the functions that we want to run on this column in a sequential manner, separated by a dot. Base R uses nested functions, where each function is called with “function_name()” and we specify columns with “object_name$column_name”.

However, both R and Python can also call columns in a dataframe with “[ ]” with the difference that Python per default subsets data columns df[“seqid”], while R always needs index specifications for rows and columns, separated by “,”: e.g. df[, “seqid”] would subset every row and only the column named “seqid”.

The sequential calling of functions is indeed very handy, it makes the code easier to read and understand than lots of interwoven functions and brackets. But while this isn’t the concept of base R, dplyr uses the magrittr principle of chaining functions with the pipe symbol “%>%”. Even though this symbol isn’t as easily typed, it’s functionality is often superior to base R, especially if you need to run many functions on a dataframe. However, with just or two functions, I usually keep to base R as it is shorter.

Python:

```
df.seqid.unique() # alternatively: df['seqid'].unique()
```

R:

```
unique(df$seqid)
```

```
##   [1] 1         10        11        12        13        14
##   [7] 15        16        17        18        19        2
##  [13] 20        21        22        3         4         5
##  [19] 6         7         8         9         GL000008.2 GL000009.2
##  [25] GL000194.1 GL000195.1 GL000205.2 GL000208.1 GL000213.1 GL000214.1
##  [31] GL000216.2 GL000218.1 GL000219.1 GL000220.1 GL000221.1 GL000224.1
##  [37] GL000225.1 GL000226.1 KI270302.1 KI270303.1 KI270304.1 KI270305.1
##  [43] KI270310.1 KI270311.1 KI270312.1 KI270315.1 KI270316.1 KI270317.1
##  [49] KI270320.1 KI270322.1 KI270329.1 KI270330.1 KI270333.1 KI270334.1
##  [55] KI270335.1 KI270336.1 KI270337.1 KI270338.1 KI270340.1 KI270362.1
##  [61] KI270363.1 KI270364.1 KI270366.1 KI270371.1 KI270372.1 KI270373.1
##  [67] KI270374.1 KI270375.1 KI270376.1 KI270378.1 KI270379.1 KI270381.1
##  [73] KI270382.1 KI270383.1 KI270384.1 KI270385.1 KI270386.1 KI270387.1
##  [79] KI270388.1 KI270389.1 KI270390.1 KI270391.1 KI270392.1 KI270393.1
##  [85] KI270394.1 KI270395.1 KI270396.1 KI270411.1 KI270412.1 KI270414.1
##  [91] KI270417.1 KI270418.1 KI270419.1 KI270420.1 KI270422.1 KI270423.1
##  [97] KI270424.1 KI270425.1 KI270429.1 KI270435.1 KI270438.1 KI270442.1
## [103] KI270448.1 KI270465.1 KI270466.1 KI270467.1 KI270468.1 KI270507.1
## [109] KI270508.1 KI270509.1 KI270510.1 KI270511.1 KI270512.1 KI270515.1
## [115] KI270516.1 KI270517.1 KI270518.1 KI270519.1 KI270521.1 KI270522.1
## [121] KI270528.1 KI270529.1 KI270530.1 KI270538.1 KI270539.1 KI270544.1
## [127] KI270548.1 KI270579.1 KI270580.1 KI270581.1 KI270582.1 KI270583.1
## [133] KI270584.1 KI270587.1 KI270588.1 KI270589.1 KI270590.1 KI270591.1
## [139] KI270593.1 KI270706.1 KI270707.1 KI270708.1 KI270709.1 KI270710.1
## [145] KI270711.1 KI270712.1 KI270713.1 KI270714.1 KI270715.1 KI270716.1
## [151] KI270717.1 KI270718.1 KI270719.1 KI270720.1 KI270721.1 KI270722.1
## [157] KI270723.1 KI270724.1 KI270725.1 KI270726.1 KI270727.1 KI270728.1
## [163] KI270729.1 KI270730.1 KI270731.1 KI270732.1 KI270733.1 KI270734.1
## [169] KI270735.1 KI270736.1 KI270737.1 KI270738.1 KI270739.1 KI270740.1
## [175] KI270741.1 KI270742.1 KI270743.1 KI270744.1 KI270745.1 KI270746.1
## [181] KI270747.1 KI270748.1 KI270749.1 KI270750.1 KI270751.1 KI270752.1
## [187] KI270753.1 KI270754.1 KI270755.1 KI270756.1 KI270757.1 MT
## [193] X         Y
## 194 Levels: 1 10 11 12 13 14 15 16 17 18 19 2 20 21 22 3 4 5 6 7 8 ... Y
```

```
# with dplyr:
# df %>% select(seqid) %>% unique
```

- how many unique seqids are there?

To get the number of unique entries in the "seqid" column, we simply need to append ".shape" to the above Python code. In R, we can either wrap the above R code with the "length()" function or use dplyr and piping instead. If we use the latter, we need to use the "nrow()" function because base R returns a vector, while dplyr returns a dataframe. Here, we can see that with two functions, using dplyr is still a bit more code but it already looks much tidier.

Python:

```
df.seqid.unique().shape
```

R:

```
length(unique(df$seqid))
```

```
## [1] 194
```

```
# with dplyr:
# df %>% select(seqid) %>% unique %>% nrow
```

- counting occurrences

To count the frequencies of each unique entry in the "source" column, we use the "value_counts()" function in Python and the "table()" function in R. These two functions differ in how they sort the output table: value_counts() sorts by decreasing frequency, while R alphabetically sorts the variables. To order the data as in Python, we need to add the "sort()" function to our R code.

Python:

```
df.source.value_counts()
```

R:

```
# table(df$source) is per default ordered alphabetically, if we want it ordered by decreasing counts, like with Python:
sort(table(df$source), decreasing = TRUE)
```

```
##
##       havana ensembl_havana      ensembl            .      mirbase
##      1441093       745065       228212       182510         4701
##       GRCh38        insdc
##          194           74
```

```
# dplyr:
# df %>% select(source) %>% table %>% sort(decreasing = TRUE)
```

How Much of the Genome Is Incomplete?

- subsetting a dataframe

We are now subsetting our original dataframe and assign it a new object name with " = " or " <- ".

To subset the dataframe to keep only rows which say "GRCh38" in the "source" column, ther are several ways to do this in R: the way that would be directly comparable to how it was done in Python would be to also use the square bracket indexing. However, there are two solutions which are more elegant: 1) base R's "subset()" or dplyr's "filter()" function. But with this short example, there is no big difference between the three.

Python's "shape" gives us the same information as R's "dim()" function: how many rows and columns our data-frame has.

To preview a random subset of 10 rows from our dataframe, we use Python's "sample()" and dplyr's "sam-ple_n()" function.

Python:

```
gdf = df[df.source == 'GRCh38']

gdf.shape
gdf.sample(10)
```

R:

```
gdf <- df[df$source == "GRCh38", ]

# alternatively:
# gdf <- subset(df, source == "GRCh38")

# dplyr:
# gdf <- df %>% filter(source == "GRCh38")

# get number of rows and columns
dim(gdf)

## [1] 194   9

# randomly sample 10 rows for observation
sample_n(gdf, 10)
```

```
##            seqid source       type start       end score strand phase
## 2511484 KI270375.1 GRCh38 supercontig     1      2378     .      .     .
## 2511545 KI270466.1 GRCh38 supercontig     1      1233     .      .     .
## 2511473 KI270337.1 GRCh38 supercontig     1      1121     .      .     .
## 2511504 KI270411.1 GRCh38 supercontig     1      2646     .      .     .
## 2511487 KI270379.1 GRCh38 supercontig     1      1045     .      .     .
## 483371         12 GRCh38  chromosome     1 133275309     .      .     .
## 2511494 KI270387.1 GRCh38 supercontig     1      1537     .      .     .
## 674768         14 GRCh38  chromosome     1 107043718     .      .     .
## 2511493 KI270386.1 GRCh38 supercontig     1      1788     .      .     .
## 2511505 KI270412.1 GRCh38 supercontig     1      1179     .      .     .
##                                       attributes
## 2511484 ID=supercontig:KI270375.1;Alias=chrUn_KI270375v1,NT_187493.1
## 2511545 ID=supercontig:KI270466.1;Alias=chrUn_KI270466v1,NT_187421.1
## 2511473 ID=supercontig:KI270337.1;Alias=chrUn_KI270337v1,NT_187466.1
## 2511504 ID=supercontig:KI270411.1;Alias=chrUn_KI270411v1,NT_187409.1
## 2511487 ID=supercontig:KI270379.1;Alias=chrUn_KI270379v1,NT_187472.1
## 483371         ID=chromosome:12;Alias=CM000674.2,chr12,NC_000012.12
## 2511494 ID=supercontig:KI270387.1;Alias=chrUn_KI270387v1,NT_187475.1
## 674768          ID=chromosome:14;Alias=CM000676.2,chr14,NC_000014.9
## 2511493 ID=supercontig:KI270386.1;Alias=chrUn_KI270386v1,NT_187480.1
## 2511505 ID=supercontig:KI270412.1;Alias=chrUn_KI270412v1,NT_187408.1
```

- creating new columns and performing calculations

Now we want to create a new column called “length”. It should contain the gene lengths, i.e. the distance in base pairs between the start and end point on the chromosomes (“start” and “end” columns). In R we don’t need to copy the dataframe first but the rest of the code is very similar: we define a new column name for the dataframe and assign its value by subtracting the values of the “start” from the values of the “end” column (+1). Here, we can see again that in Python we use a dot to define columns, while R uses the Dollar sign.

The sum of all lengths is calculated with the “sum()” function in both languages.

Python:

```
gdf = gdf.copy()
gdf['length'] = gdf.end - gdf.start + 1

gdf.length.sum()
```

R:

```
gdf$length <- gdf$end - gdf$start + 1

sum(gdf$length)
```

```
## [1] 3096629726
```

Next, we want to calculating the proportion of the genome that is not on main chromosome assemblies. For that, we first define a character string with the main chromosomes: 1 to 23, X, Y and MT (mitochondrial chromosome). Defining this string is a bit easier in R.

We will use this string to calculate the sum of lengths of the subsetted dataframe and divide it by the sum of lengths of the whole dataframe. For subsetting, we use the “isin()” function of Python, which corresponds to R’s “%in%”.

Python:

```
chrs = [str(_) for _ in range(1, 23)] + ['X', 'Y', 'MT']
gdf[-gdf.seqid.isin(chrs)].length.sum() / gdf.length.sum()

# or
gdf[(gdf['type'] == 'supercontig')].length.sum() / gdf.length.sum()
```

R:

```
chrs <- c(1:23, "X", "Y", "MT")
sum(subset(gdf, !seqid %in% chrs)$length) / sum(gdf$length)
```

```
## [1] 0.003702192
```

How Many Genes Are There?

Here, we are again using the same functions as above for subsetting the dataframe, asking for its dimensions, printing 10 random lines and asking for the frequencies of each unique item in the “type” column. For the latter I am using base R over dplyr, because it’s faster to type.

Python:

```
edf = df[df.source.isin(['ensembl', 'havana', 'ensembl_havana'])]
edf.shape

edf.sample(10)
edf.type.value_counts()
```

R:

```
edf <- subset(df, source %in% c("ensembl", "havana", "ensembl_havana"))
dim(edf)
```

```
## [1] 2414370       9
```

```
sample_n(edf, 10)

##        seqid        source          type    start      end score
## 2548897    X     ensembl three_prime_UTR  67723842  67730618     .
## 790670    15 ensembl_havana           CDS  42352030  42352139     .
## 1298369   19        havana           CDS  38942683  38942751     .
## 189728     1 ensembl_havana           CDS 186304108 186304257     .
## 1080599   17        havana          exon  44401337  44401453     .
## 1562968   20 ensembl_havana          exon   2964272   2964350     .
## 1965100    5       ensembl    transcript    144087    144197     .
## 2581554    X        havana          exon 147927670 147928807     .
## 785399    15        havana          exon  40611478  40611511     .
## 1417830    2 ensembl_havana           CDS  71520178  71520208     .
##        strand phase
## 2548897     +     .
## 790670      +     2
## 1298369     -     0
## 189728      +     2
## 1080599     -     .
## 1562968     +     .
## 1965100     -     .
## 2581554     +     .
## 785399      +     .
## 1417830     +     0
##                                                                     attributes
## 2548897                                                         Parent=tran-
script:ENST00000612452
## 790670                                   ID=CDS:ENSP00000326227;Parent=tran-
script:ENST00000318010;protein_id=ENSP00000326227
## 1298369                                  ID=CDS:ENSP00000472465;Parent=tran-
script:ENST00000599996;protein_id=ENSP00000472465
## 189728                                   ID=CDS:ENSP00000356453;Parent=tran-
script:ENST00000367483;protein_id=ENSP00000356453
## 1080599          Parent=transcript:ENST00000585614;Name=ENSE00002917300;constitutive=0;en-
sembl_end_phase=2;ensembl_phase=2;exon_id=ENSE00002917300;rank=7;version=1
## 1562968          Parent=transcript:ENST00000216877;Name=ENSE00003603687;constitutive=1;en-
sembl_end_phase=1;ensembl_phase=-1;exon_id=ENSE00003603687;rank=4;version=1
## 1965100 ID=transcript:ENST00000362670;Parent=gene:ENSG00000199540;Name=Y_RNA.43-201;bio-
type=misc_RNA;tag=basic;transcript_id=ENST00000362670;transcript_support_level=NA;version=1
## 2581554          Parent=transcript:ENST00000620828;Name=ENSE00003724969;constitutive=0;en-
sembl_end_phase=-1;ensembl_phase=-1;exon_id=ENSE00003724969;rank=1;version=1
```

```
## 785399                Parent=transcript:ENST00000399668;Name=ENSE00003659959;constitutive=0;en-
sembl_end_phase=2;ensembl_phase=1;exon_id=ENSE00003659959;rank=7;version=1
## 1417830                                                ID=CDS:ENSP00000398305;Parent=tran-
script:ENST00000429174;protein_id=ENSP00000398305
```

```
sort(table(edf$type), decreasing = TRUE)
```

```
##
##                exon                 CDS
##             1180596                704604
##       five_prime_UTR       three_prime_UTR
##              142387                133938
##          transcript                gene
##               96375                42470
##    processed_transcript aberrant_processed_transcript
##               28228                26944
##   NMD_transcript_variant              lincRNA
##               13761                13247
##    processed_pseudogene         lincRNA_gene
##               10722                 7533
##          pseudogene                  RNA
##                3049                2221
##               snRNA           snRNA_gene
##                1909                1909
##               snoRNA           snoRNA_gene
##                 956                 944
##   pseudogenic_transcript              rRNA
##                 737                 549
##           rRNA_gene               miRNA
##                 549                 302
##        V_gene_segment        J_gene_segment
##                 216                 158
##       VD_gene_segment        C_gene_segment
##                  37                  29
##      biological_region          chromosome
##                   0                   0
##          miRNA_gene               mt_gene
##                   0                   0
##          supercontig
##                   0
```

Now we want to subset the dataframe to rows with the attribute “gene” in the “type” column, look at 10 random lines from the “attributes” column and get the dataframe dimensions.

Python:

```
ndf = edf[edf.type == 'gene']
ndf = ndf.copy()
ndf.sample(10).attributes.values
ndf.shape
```

R:

```
ndf <- subset(edf, type == "gene")
sample_n(ndf, 10)$attributes
```

```
##  [1] ID=gene:ENSG00000170949;Name=ZNF160;biotype=protein_coding;description=zinc finger protein 160
[Source:HGNC Symbol%3BAcc:HGNC:12948];gene_id=ENSG00000170949;havana_gene=OT-
THUMG00000182854;havana_version=3;logic_name=ensembl_havana_gene;version=17
##  [2] ID=gene:ENSG00000215088;Name=RPS5P3;biotype=processed_pseudogene;description=ribosomal pro-
tein S5 pseudogene 3 [Source:HGNC Symbol%3BAcc:HGNC:10427];gene_id=ENSG00000215088;ha-
vana_gene=OTTHUMG00000065532;havana_version=1;logic_name=havana;version=3
##  [3] ID=gene:ENSG00000249077;Name=RP11-478C1.8;biotype=processed_pseudo-
gene;gene_id=ENSG00000249077;havana_gene=OTTHUMG00000160290;havana_version=1;logic_name=ha-
vana;version=1
##  [4] ID=gene:ENSG00000110619;Name=CARS;biotype=protein_coding;description=cysteinyl-tRNA synthetase
[Source:HGNC Symbol%3BAcc:HGNC:1493];gene_id=ENSG00000110619;havana_gene=OT-
THUMG00000010927;havana_version=9;logic_name=ensembl_havana_gene;version=16
##  [5] ID=gene:ENSG00000168010;Name=ATG16L2;biotype=protein_coding;description=autophagy related 16
like 2 [Source:HGNC Symbol%3BAcc:HGNC:25464];gene_id=ENSG00000168010;havana_gene=OT-
THUMG00000167961;havana_version=2;logic_name=ensembl_havana_gene;version=10
##  [6] ID=gene:ENSG00000055957;Name=ITIH1;biotype=protein_coding;description=inter-alpha-trypsin inhibi-
tor heavy chain 1 [Source:HGNC Symbol%3BAcc:HGNC:6166];gene_id=ENSG00000055957;havana_gene=OT-
THUMG00000150312;havana_version=9;logic_name=ensembl_havana_gene;version=10
##  [7] ID=gene:ENSG00000258781;Name=RP11-496I2.4;biotype=unprocessed_pseudo-
gene;gene_id=ENSG00000258781;havana_gene=OTTHUMG00000170502;havana_version=2;logic_name=ha-
vana;version=2
##  [8] ID=gene:ENSG00000157965;Name=SSX8;biotype=unprocessed_pseudogene;description=SSX family
member 8 [Source:HGNC Symbol%3BAcc:HGNC:19654];gene_id=ENSG00000157965;havana_gene=OT-
THUMG00000021573;havana_version=4;logic_name=havana;version=11
##  [9] ID=gene:ENSG00000273489;Name=RP11-180C16.1;biotype=antisense;gene_id=ENSG00000273489;ha-
vana_gene=OTTHUMG00000186336;havana_version=1;logic_name=havana;version=1
## [10] ID=gene:ENSG00000279532;Name=CTB-96E2.6;biotype=TEC;gene_id=ENSG00000279532;ha-
vana_gene=OTTHUMG00000179414;havana_version=1;logic_name=havana;version=1
## 1623077 Levels: external_name=Ala;logic_name=trnascan ...
```

```
dim(ndf)
```

```
## [1] 42470     9
```

- extracting gene information from attributes field

I don't know if there is an easier way in Python but in R we don't need to create big helper functions around it. We can simply use "gsub()" with defining the regular expression for what we want to extract. This makes the R code much shorter and easier to understand! We then drop the original "attributes" column.

To have a look at the dataframe, we use "head()" this time.

Python:

```
import re

RE_GENE_NAME = re.compile(r'Name=(?P<gene_name>.+?);')
def extract_gene_name(attributes_str):
    res = RE_GENE_NAME.search(attributes_str)
    return res.group('gene_name')

ndf['gene_name'] = ndf.attributes.apply(extract_gene_name)

RE_GENE_ID = re.compile(r'gene_id=(?P<gene_id>ENSG.+?);')
def extract_gene_id(attributes_str):
    res = RE_GENE_ID.search(attributes_str)
    return res.group('gene_id')

ndf['gene_id'] = ndf.attributes.apply(extract_gene_id)

RE_DESC = re.compile('description=(?P<desc>.+?);')
def extract_description(attributes_str):
    res = RE_DESC.search(attributes_str)
    if res is None:
        return ''
    else:
        return res.group('desc')

ndf['desc'] = ndf.attributes.apply(extract_description)

ndf.drop('attributes', axis=1, inplace=True)
ndf.head()
```

R:

```
ndf$gene_name <- gsub("(.*Name=)(.*?)(;biotype.*)", "\\2", ndf$attributes)
ndf$gene_id <- gsub("(ID=gene:)(.*?)(;Name.*)", "\\2", ndf$attributes)
ndf$desc <- gsub("(.*description=)(.*?)(;.*)", "\\2", ndf$attributes)
```

```
# some genes don't have a description
ndf$desc <- ifelse(grepl("^ID=.*", ndf$desc), "", ndf$desc)

ndf <- subset(ndf, select = -attributes)
head(ndf)

##     seqid         source type  start    end score strand phase gene_name
## 17      1         havana gene  11869  14409     .      +     .   DDX11L1
## 29      1         havana gene  14404  29570     .      -     .    WASH7P
## 72      1         havana gene  52473  53312     .      +     .    OR4G4P
## 75      1         havana gene  62948  63887     .      +     .   OR4G11P
## 78      1 ensembl_havana gene  69091  70008     .      +     .    OR4F5
## 109     1         havana gene 131025 134836     .      +     .    CICP27
##             gene_id
## 17  ENSG00000223972
## 29  ENSG00000227232
## 72  ENSG00000268020
## 75  ENSG00000240361
## 78  ENSG00000186092
## 109 ENSG00000233750
##                                                                desc
## 17                     DEAD/H-box helicase 11 like 1 [Source:HGNC Symbol%3BAcc:HGNC:37102]
## 29                WAS protein family homolog 7 pseudogene [Source:HGNC Symbol%3BAcc:HGNC:38034]
## 72   olfactory receptor family 4 subfamily G member 4 pseudogene [Source:HGNC Sym-
bol%3BAcc:HGNC:14822]
## 75  olfactory receptor family 4 subfamily G member 11 pseudogene [Source:HGNC Sym-
bol%3BAcc:HGNC:31276]
## 78           olfactory receptor family 4 subfamily F member 5 [Source:HGNC Symbol%3BAcc:HGNC:14825]
## 109           capicua transcriptional repressor pseudogene 27 [Source:HGNC Symbol%3BAcc:HGNC:48835]
```

Next, we want to know how many unique gene names and gene IDs there are. As above we use the "unique()", "shape()" (Python) and "length()" (R) functions.

The count table for gene names can again be obtained with R's "table()" function, even though Zhuyi Xue uses a slightly different approach: he first groups the "gene_name" column, then counts and sorts.

Finally, we can calculate the proportion of genes that have more than appear more than once and we can have a closer look at the SCARNA20 gene.

Python:

```
ndf.shape
ndf.gene_id.unique().shape
ndf.gene_name.unique().shape
```

```
count_df = ndf.groupby('gene_name').count().ix[:, 0].sort_values().ix[::-1]
count_df.head(10)

count_df[count_df > 1].shape
count_df.shape
count_df[count_df > 1].shape[0] / count_df.shape[0]

ndf[ndf.gene_name == 'SCARNA20']
```

R:

```
dim(ndf)

## [1] 42470    11

length(unique(ndf$gene_id))

## [1] 42470

length(unique(ndf$gene_name))

## [1] 42387

count_df <- sort(table(ndf$gene_name), decreasing = TRUE)
head(count_df, n = 10)

##
##      SCARNA20      SCARNA16      SCARNA17      SCARNA11
##             7             6             5             4
##      SCARNA15      SCARNA21 Clostridiales-1       SCARNA4
##             4             4             3             3
##      ACTR3BP2         AGBL1
##             2             2

length(count_df[count_df > 1])

## [1] 63

length(count_df)

## [1] 42387

length(count_df[count_df > 1]) / length(count_df)

## [1] 0.001486305

ndf[ndf$gene_name == "SCARNA20", ]
```

```
##        seqid  source type     start       end score strand phase
## 179400     1 ensembl gene 171768070 171768175     .      +     .
## 201038     1 ensembl gene 204727991 204728106     .      +     .
## 349204    11 ensembl gene   8555016   8555146     .      +     .
## 718521    14 ensembl gene  63479272  63479413     .      +     .
## 837234    15 ensembl gene  75121536  75121666     .      -     .
## 1039875   17 ensembl gene  28018770  28018907     .      +     .
## 1108216   17 ensembl gene  60231516  60231646     .      -     .
##        gene_name         gene_id
## 179400   SCARNA20 ENSG00000253060
## 201038   SCARNA20 ENSG00000251861
## 349204   SCARNA20 ENSG00000252778
## 718521   SCARNA20 ENSG00000252800
## 837234   SCARNA20 ENSG00000252722
## 1039875  SCARNA20 ENSG00000251818
## 1108216  SCARNA20 ENSG00000252577
##                                                    desc
## 179400          Small Cajal body specific RNA 20 [Source:RFAM%3BAcc:RF00601]
## 201038          Small Cajal body specific RNA 20 [Source:RFAM%3BAcc:RF00601]
## 349204          Small Cajal body specific RNA 20 [Source:RFAM%3BAcc:RF00601]
## 718521          Small Cajal body specific RNA 20 [Source:RFAM%3BAcc:RF00601]
## 837234          Small Cajal body specific RNA 20 [Source:RFAM%3BAcc:RF00601]
## 1039875          Small Cajal body specific RNA 20 [Source:RFAM%3BAcc:RF00601]
## 1108216 small Cajal body-specific RNA 20 [Source:HGNC Symbol%3BAcc:HGNC:32578]

# dplyr:
# ndf %>% filter(gene_name == "SCARNA20")
```

How Long Is a Typical Gene?

To calculate gene lengths we use the same code as before. R's summary() is not exactly the same as Python's describe() but it's close enough.

Python:

```
ndf['length'] = ndf.end - ndf.start + 1
ndf.length.describe()
```

R:

```
ndf$length <- ndf$end - ndf$start + 1
summary(ndf$length)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       8     884    5170   35830   30550 2305000
```

Now we produce the first plot, showing a histogram of gene length. In base R you can't really plot a histogram with logarithmic y-axis scales (at least not without manually tweaking the hist() output but it isn't recommended anyway because 0 will become -Inf). But we can do it easily with ggplot2 with "scale_y_log10()". The code we need for ggplot2 is a bit longer than with matplotlib. We could of course further customize our plot but for now, let's keep it simple.

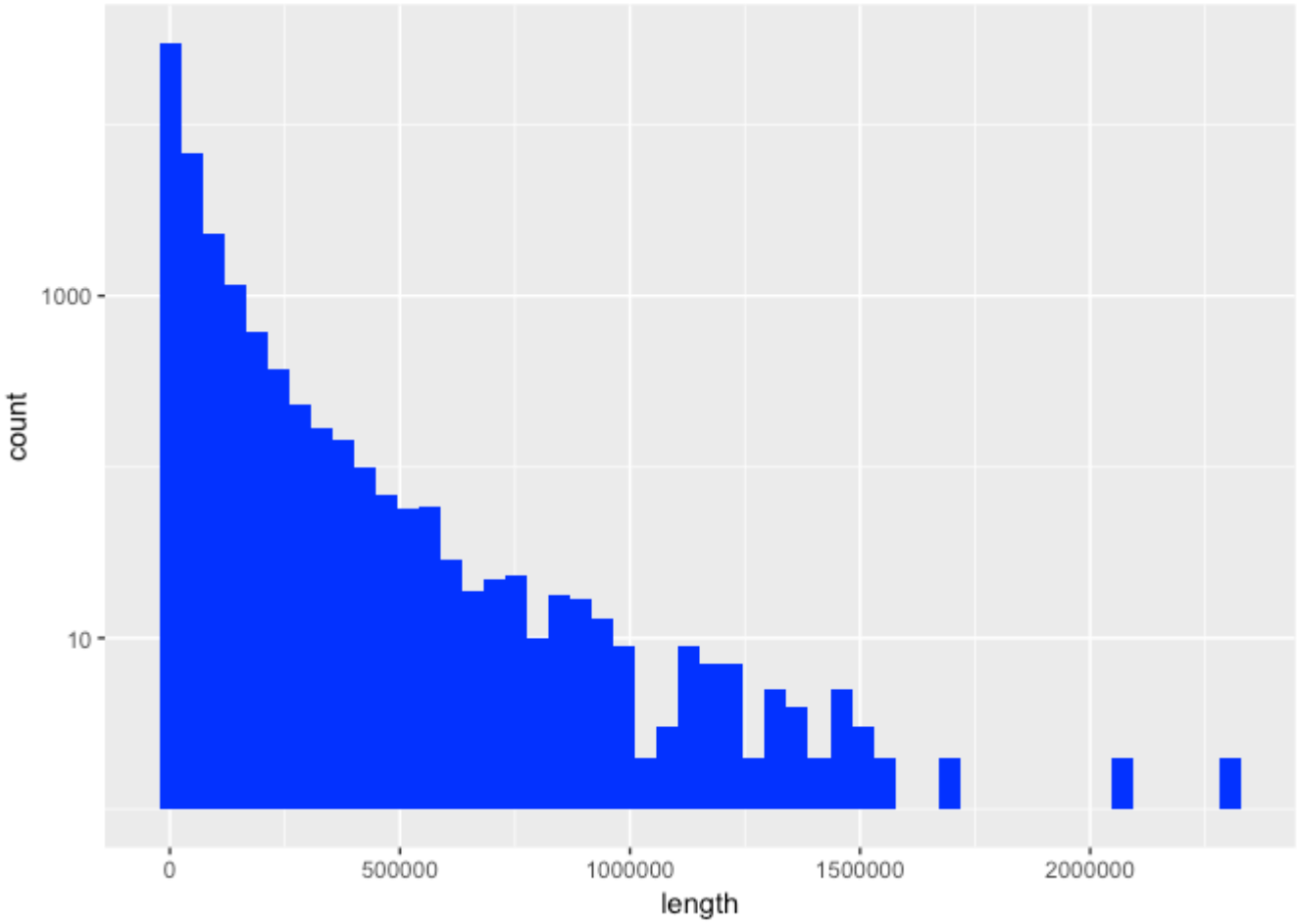
Python:

```
import matplotlib as plt

ndf.length.plot(kind='hist', bins=50, logy=True)
plt.show()
```

R:

```
ndf %>% ggplot(aes(x = length)) +
  geom_histogram(bins = 50, fill = "blue") +
  scale_y_log10()
```



Now, we subset the dataframe to keep only rows where the "length" column contains values bigger than 2 million and order it by descending gene length. To see the shortest genes, we order the original dataframe and view the first 6 rows. Instead of "sort()", we use dplyr's "arrange()" function this time (I didn't use it before because it can only be applied to dataframes).

Python:

```
ndf[ndf.length > 2e6].sort_values('length').ix[::-1]
ndf.sort_values('length').head()
```

R:

```
ndf %>% filter(length > 2e6) %>% arrange(desc(length))
```

```
##   seqid       source type     start       end score strand phase
## 1     7 ensembl_havana gene 146116002 148420998     .      +     .
## 2     9 ensembl_havana gene   8314246  10612723     .      -     .
## 3     X ensembl_havana gene  31097677  33339441     .      -     .
## 4    11 ensembl_havana gene  83455012  85627922     .      -     .
## 5     8 ensembl_havana gene   2935353   4994972     .      -     .
```

```
## 6    20 ensembl_havana gene  13995369  16053197     .     +     .
##   gene_name         gene_id
## 1   CNTNAP2 ENSG00000174469
## 2     PTPRD ENSG00000153707
## 3       DMD ENSG00000198947
## 4      DLG2 ENSG00000150672
## 5     CSMD1 ENSG00000183117
## 6   MACROD2 ENSG00000172264
##                                                  desc
## 1          contactin associated protein-like 2 [Source:HGNC Symbol%3BAcc:HGNC:13830]
## 2 protein tyrosine phosphatase%2C receptor type D [Source:HGNC Symbol%3BAcc:HGNC:9668]
## 3                          dystrophin [Source:HGNC Symbol%3BAcc:HGNC:2928]
## 4          discs large MAGUK scaffold protein 2 [Source:HGNC Symbol%3BAcc:HGNC:2901]
## 5            CUB and Sushi multiple domains 1 [Source:HGNC Symbol%3BAcc:HGNC:14026]
## 6                MACRO domain containing 2 [Source:HGNC Symbol%3BAcc:HGNC:16126]
##    length
## 1 2304997
## 2 2298478
## 3 2241765
## 4 2172911
## 5 2059620
## 6 2057829

head(arrange(ndf, length))

##   seqid source type     start       end score strand phase    gene_name
## 1    14 havana gene  22438547  22438554     .     +     .       TRDD1
## 2    14 havana gene  22439007  22439015     .     +     .       TRDD2
## 3     7 havana gene 142786213 142786224     .     +     .       TRBD1
## 4    14 havana gene  22449113  22449125     .     +     .       TRDD3
## 5     4 havana gene  10238213  10238235     .     -     .   AC006499.9
## 6     3 havana gene 179452395 179452419     .     -     . RP11-145M9.2
##          gene_id
## 1 ENSG00000223997
## 2 ENSG00000237235
## 3 ENSG00000282431
## 4 ENSG00000228985
## 5 ENSG00000271544
## 6 ENSG00000239255
##                                                desc
## 1 T cell receptor delta diversity 1 [Source:HGNC Symbol%3BAcc:HGNC:12254]
## 2 T cell receptor delta diversity 2 [Source:HGNC Symbol%3BAcc:HGNC:12255]
## 3  T cell receptor beta diversity 1 [Source:HGNC Symbol%3BAcc:HGNC:12158]
```

```
## 4 T cell receptor delta diversity 3 [Source:HGNC Symbol%3BAcc:HGNC:12256]
## 5
## 6
##   length
## 1      8
## 2      9
## 3     12
## 4     13
## 5     23
## 6     25
```

Gene Distribution Among Chromosomes

The number of genes per chromosome are counted with the “subset()”, “table()” and “sort()” functions as described earlier.

Python:

```
ndf = ndf[ndf.seqid.isin(chrs)]
chr_gene_counts = ndf.groupby('seqid').count().ix[:, 0].sort_values().ix[::-1]
chr_gene_counts
```

R:

```
ndf$seqid <- as.character(ndf$seqid) # as factors it will subset the dataframe but keep the factor levels
ndf <- subset(ndf, seqid %in% chrs)
chr_gene_counts <- sort(table(ndf$seqid), decreasing = TRUE)
chr_gene_counts
```

```
##
##    1    2   11   19   17    3    6   12    7    5   16    X    4    9    8
## 3902 2806 2561 2412 2280 2204 2154 2140 2106 2002 1881 1852 1751 1659 1628
##   10   15   14   22   20   13   18   21    Y
## 1600 1476 1449  996  965  872  766  541  436
```

To see all genes that are on mitochondrial chromosome, we subset the first dataframe by two conditions. In both, R and Python this is done with the ampersand symbol but in R we don’t need brackets around the individual conditions.

Python:

```
df[(df.type == 'gene') & (df.seqid == 'MT')]
```

R:

```
subset(df, type == "gene" & seqid == "MT")

##        seqid source type start   end score strand phase
## 2514004    MT  insdc gene   648  1601     .      +     .
## 2514010    MT  insdc gene  1671  3229     .      +     .
## 2514017    MT  insdc gene  3307  4262     .      +     .
## 2514030    MT  insdc gene  4470  5511     .      +     .
## 2514049    MT  insdc gene  5904  7445     .      +     .
## 2514059    MT  insdc gene  7586  8269     .      +     .
## 2514066    MT  insdc gene  8366  8572     .      +     .
## 2514070    MT  insdc gene  8527  9207     .      +     .
## 2514074    MT  insdc gene  9207  9990     .      +     .
## 2514081    MT  insdc gene 10059 10404     .      +     .
## 2514088    MT  insdc gene 10470 10766     .      +     .
## 2514092    MT  insdc gene 10760 12137     .      +     .
## 2514105    MT  insdc gene 12337 14148     .      +     .
## 2514109    MT  insdc gene 14149 14673     .      -     .
## 2514116    MT  insdc gene 14747 15887     .      +     .
##
attributes
## 2514004                                  ID=gene:ENSG00000211459;Name=MT-RNR1;biotype=Mt_rRNA;descrip-
tion=mitochondrially encoded 12S RNA [Source:HGNC Sym-
bol%3BAcc:HGNC:7470];gene_id=ENSG00000211459;logic_name=mt_genbank_import;version=2
## 2514010                                  ID=gene:ENSG00000210082;Name=MT-RNR2;biotype=Mt_rRNA;descrip-
tion=mitochondrially encoded 16S RNA [Source:HGNC Sym-
bol%3BAcc:HGNC:7471];gene_id=ENSG00000210082;logic_name=mt_genbank_import;version=2
## 2514017   ID=gene:ENSG00000198888;Name=MT-ND1;biotype=protein_coding;description=mitochondrially
encoded NADH:ubiquinone oxidoreductase core subunit 1 [Source:HGNC Sym-
bol%3BAcc:HGNC:7455];gene_id=ENSG00000198888;logic_name=mt_genbank_import;version=2
## 2514030   ID=gene:ENSG00000198763;Name=MT-ND2;biotype=protein_coding;description=mitochondrially
encoded NADH:ubiquinone oxidoreductase core subunit 2 [Source:HGNC Sym-
bol%3BAcc:HGNC:7456];gene_id=ENSG00000198763;logic_name=mt_genbank_import;version=3
## 2514049                   ID=gene:ENSG00000198804;Name=MT-CO1;biotype=protein_coding;description=mi-
tochondrially encoded cytochrome c oxidase I [Source:HGNC Sym-
bol%3BAcc:HGNC:7419];gene_id=ENSG00000198804;logic_name=mt_genbank_import;version=2
## 2514059                  ID=gene:ENSG00000198712;Name=MT-CO2;biotype=protein_coding;description=mi-
tochondrially encoded cytochrome c oxidase II [Source:HGNC Sym-
bol%3BAcc:HGNC:7421];gene_id=ENSG00000198712;logic_name=mt_genbank_import;version=1
## 2514066                        ID=gene:ENSG00000228253;Name=MT-ATP8;biotype=protein_coding;descrip-
tion=mitochondrially encoded ATP synthase 8 [Source:HGNC Sym-
bol%3BAcc:HGNC:7415];gene_id=ENSG00000228253;logic_name=mt_genbank_import;version=1
```

2514070 ID=gene:ENSG00000198899;Name=MT-ATP6;biotype=protein_coding;description=mitochondrially encoded ATP synthase 6 [Source:HGNC Symbol%3BAcc:HGNC:7414];gene_id=ENSG00000198899;logic_name=mt_genbank_import;version=2
2514074 ID=gene:ENSG00000198938;Name=MT-CO3;biotype=protein_coding;description=mitochondrially encoded cytochrome c oxidase III [Source:HGNC Symbol%3BAcc:HGNC:7422];gene_id=ENSG00000198938;logic_name=mt_genbank_import;version=2
2514081 ID=gene:ENSG00000198840;Name=MT-ND3;biotype=protein_coding;description=mitochondrially encoded NADH:ubiquinone oxidoreductase core subunit 3 [Source:HGNC Symbol%3BAcc:HGNC:7458];gene_id=ENSG00000198840;logic_name=mt_genbank_import;version=2
2514088 ID=gene:ENSG00000212907;Name=MT-ND4L;biotype=protein_coding;description=mitochondrially encoded NADH:ubiquinone oxidoreductase core subunit 4L [Source:HGNC Symbol%3BAcc:HGNC:7460];gene_id=ENSG00000212907;logic_name=mt_genbank_import;version=2
2514092 ID=gene:ENSG00000198886;Name=MT-ND4;biotype=protein_coding;description=mitochondrially encoded NADH:ubiquinone oxidoreductase core subunit 4 [Source:HGNC Symbol%3BAcc:HGNC:7459];gene_id=ENSG00000198886;logic_name=mt_genbank_import;version=2
2514105 ID=gene:ENSG00000198786;Name=MT-ND5;biotype=protein_coding;description=mitochondrially encoded NADH:ubiquinone oxidoreductase core subunit 5 [Source:HGNC Symbol%3BAcc:HGNC:7461];gene_id=ENSG00000198786;logic_name=mt_genbank_import;version=2
2514109 ID=gene:ENSG00000198695;Name=MT-ND6;biotype=protein_coding;description=mitochondrially encoded NADH:ubiquinone oxidoreductase core subunit 6 [Source:HGNC Symbol%3BAcc:HGNC:7462];gene_id=ENSG00000198695;logic_name=mt_genbank_import;version=2
2514116 ID=gene:ENSG00000198727;Name=MT-CYB;biotype=protein_coding;description=mitochondrially encoded cytochrome b [Source:HGNC Symbol%3BAcc:HGNC:7427];gene_id=ENSG00000198727;logic_name=mt_genbank_import;version=2

We can get the chromosome lengths from the dataframe as well. We again subset to only the main chromosomes, then drop unwanted columns and order by length.

Python:

```
gdf = gdf[gdf.seqid.isin(chrs)]
gdf.drop(['start', 'end', 'score', 'strand', 'phase' ,'attributes'], axis=1, inplace=True)
gdf.sort_values('length').ix[::-1]
```

R:

```
gdf$seqid <- as.character(gdf$seqid) # as factors it will subset the dataframe but keep the factor levels
gdf <- subset(gdf, as.character(seqid) %in% chrs) %>%
  select(-(start:attributes))
arrange(gdf, desc(length))
```

```
##   seqid source      type    length
## 1     1 GRCh38 chromosome 248956422
## 2     2 GRCh38 chromosome 242193529
```

```
## 3      3 GRCh38 chromosome 198295559
## 4      4 GRCh38 chromosome 190214555
## 5      5 GRCh38 chromosome 181538259
## 6      6 GRCh38 chromosome 170805979
## 7      7 GRCh38 chromosome 159345973
## 8      X GRCh38 chromosome 156040895
## 9      8 GRCh38 chromosome 145138636
## 10     9 GRCh38 chromosome 138394717
## 11    11 GRCh38 chromosome 135086622
## 12    10 GRCh38 chromosome 133797422
## 13    12 GRCh38 chromosome 133275309
## 14    13 GRCh38 chromosome 114364328
## 15    14 GRCh38 chromosome 107043718
## 16    15 GRCh38 chromosome 101991189
## 17    16 GRCh38 chromosome  90338345
## 18    17 GRCh38 chromosome  83257441
## 19    18 GRCh38 chromosome  80373285
## 20    20 GRCh38 chromosome  64444167
## 21    19 GRCh38 chromosome  58617616
## 22     Y GRCh38 chromosome  54106423
## 23    22 GRCh38 chromosome  50818468
## 24    21 GRCh38 chromosome  46709983
## 25    MT GRCh38 chromosome     16569
```

Now, we merge the dataframe with the number of genes per chromosome with the dataframe of chromosome lengths. Because R's "table()" function produces a vector, we need to convert it to a dataframe first and define the column names. Then, we use the "merge()" function and point to the name of the column we want to merge by.

Python:

```
cdf = chr_gene_counts.to_frame(name='gene_count').reset_index()
cdf.head(2)

merged = gdf.merge(cdf, on='seqid')
```

R:

```
cdf <- as.data.frame(chr_gene_counts)
colnames(cdf) <- c("seqid", "gene_count")
head(cdf, n = 2)
```

```
##   seqid gene_count
## 1     1       3902
## 2     2       2806
```

```
merged <- merge(gdf, cdf, by = "seqid")
merged
```

```
##    seqid source       type    length gene_count
## 1      1 GRCh38 chromosome 248956422       3902
## 2     10 GRCh38 chromosome 133797422       1600
## 3     11 GRCh38 chromosome 135086622       2561
## 4     12 GRCh38 chromosome 133275309       2140
## 5     13 GRCh38 chromosome 114364328        872
## 6     14 GRCh38 chromosome 107043718       1449
## 7     15 GRCh38 chromosome 101991189       1476
## 8     16 GRCh38 chromosome  90338345       1881
## 9     17 GRCh38 chromosome  83257441       2280
## 10    18 GRCh38 chromosome  80373285        766
## 11    19 GRCh38 chromosome  58617616       2412
## 12     2 GRCh38 chromosome 242193529       2806
## 13    20 GRCh38 chromosome  64444167        965
## 14    21 GRCh38 chromosome  46709983        541
## 15    22 GRCh38 chromosome  50818468        996
## 16     3 GRCh38 chromosome 198295559       2204
## 17     4 GRCh38 chromosome 190214555       1751
## 18     5 GRCh38 chromosome 181538259       2002
## 19     6 GRCh38 chromosome 170805979       2154
## 20     7 GRCh38 chromosome 159345973       2106
## 21     8 GRCh38 chromosome 145138636       1628
## 22     9 GRCh38 chromosome 138394717       1659
## 23     X GRCh38 chromosome 156040895       1852
## 24     Y GRCh38 chromosome  54106423        436
```

To calculate the correlation between length and gene count, we subset the merged dataframe to those two columns and use the “corr()” (Python) or “cor()” (R) function.

Python:

```
merged[['length', 'gene_count']].corr()
```

R:

```
cor(merged[, c("length", "gene_count")])
```

```
##              length gene_count
## length     1.0000000  0.7282208
## gene_count 0.7282208  1.0000000
```

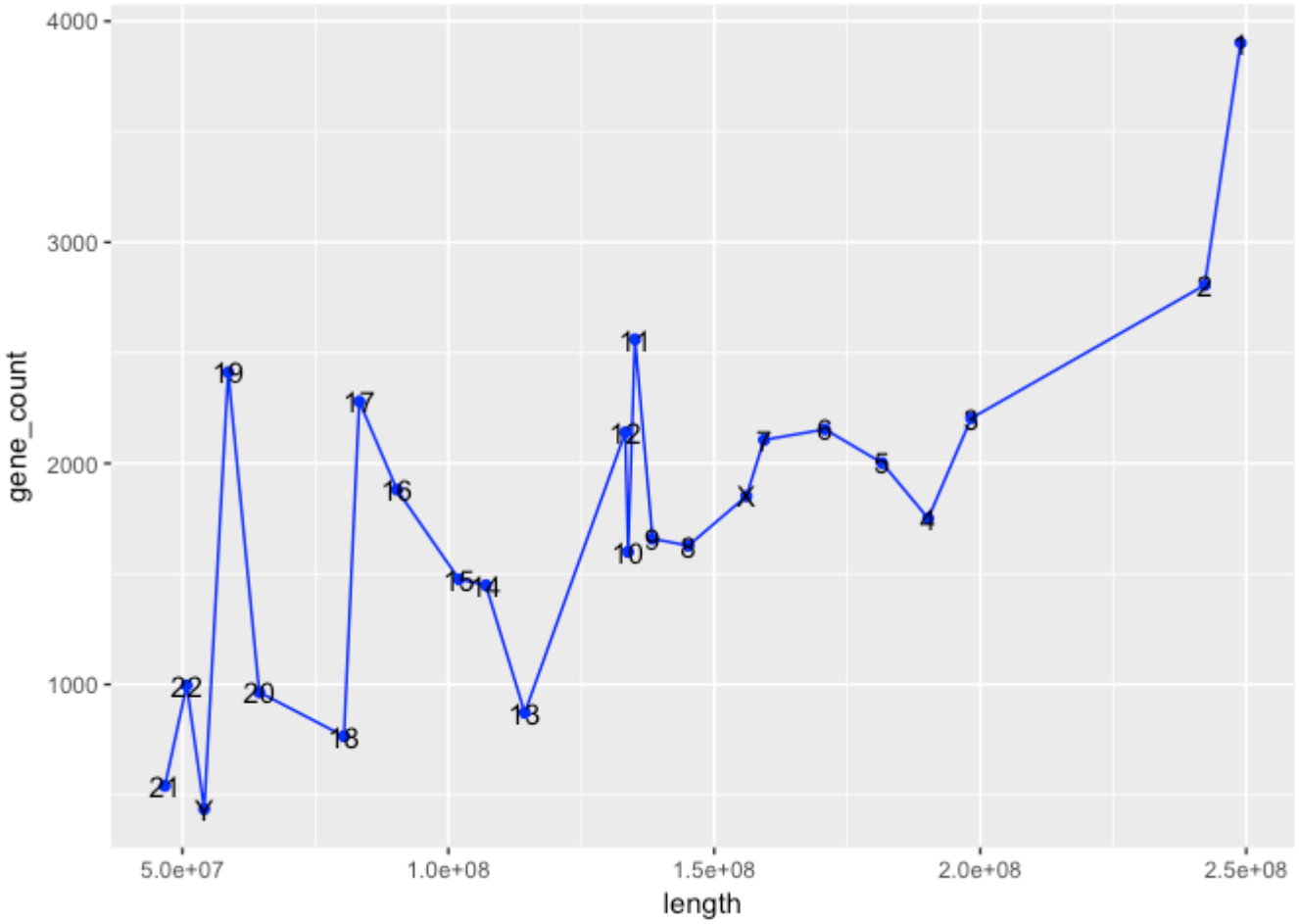
And now we produce the final plot: a line plot of chromosome length by number of genes per chromosome. For Python, we again use the matplotlib and for R the ggplot2 packages. Because Zhuyi Xue creates a new dataframe and tweaks the plot somewhat, our ggplot2 code is simpler and tidier here.

Python:

```
ax = merged[['length', 'gene_count']].sort_values('length').plot(x='length', y='gene_count', style='o-')
# add some margin to both ends of x axis
xlim = ax.get_xlim()
margin = xlim[0] * 0.1
ax.set_xlim([xlim[0] - margin, xlim[1] + margin])
# Label each point on the graph
for (s, x, y) in merged[['seqid', 'length', 'gene_count']].sort_values('length').values:
    ax.text(x, y - 100, str(s))
```

R:

```
merged[, c("seqid", "length", "gene_count")] %>%
  arrange(desc(length)) %>%
  ggplot(aes(x = length, y = gene_count, label = seqid)) +
  geom_point(color = "blue") +
  geom_line(color = "blue") +
  geom_text()
```




```
## R version 3.3.2 (2016-10-31)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: macOS Sierra 10.12.1
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] ggplot2_2.2.1 dplyr_0.5.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.8      codetools_0.2-15 digest_0.6.11    rprojroot_1.1
```

```
##  [5] assertthat_0.1   plyr_1.8.4       grid_3.3.2       R6_2.2.0
##  [9] gtable_0.2.0     DBI_0.5-1        backports_1.0.4  magrittr_1.5
## [13] scales_0.4.1     evaluate_0.10    stringi_1.1.2    lazyeval_0.2.0
## [17] rmarkdown_1.3    labeling_0.3     tools_3.3.2      stringr_1.1.0
## [21] munsell_0.4.3    yaml_2.1.14      colorspace_1.3-2 htmltools_0.3.5
## [25] knitr_1.15.1     tibble_1.2
```

Appendix E – Deep Learning Resouces (Python, Java, R) – CNTK / DL4J

Code Examples for Learning / Understanding

Code will be hosted on GitHub @ https://meetnavpk.github.io/code/ and will have following Tech. Demos for Understanding ANNs/DNNs.

- MNIST Example
- XOR
- Feed-forward
- Convolutional
- NLP
- Recurrent
- Transfer Learning
- Spark Hadoop
- CPU vs GPU
- HyperparameterOptimisation

Labs

Labs will be hosted on GitHub @ https://meetnavpk.github.io/labs/

Industry Use Case Samples

Labs will be hosted on GitHub @ https://meetnavpk.github.io/samples/

Appendix F – Deep Learning Tooling Options

The following table compares some of the most popular software frameworks, libraries and computer programs for deep learning.

Software	Creator	License[a]	Open source	Platform	Written in	Interface	OpenMP support	OpenCL support	CUDA support	Automatic differentiation[1]	Has pre-trained models	Recurrent nets	Convolutional nets	RBM / DBNs	Parallel execution (multi node)
Caffe	Berkeley Vision and Learning Center	BSD license	Yes	Linux, macOS, Windows[2]	C++	Python, MATLAB	Yes	Under development[3]	Yes	Yes	Yes[4]	Yes	Yes	No	?
Caffe2	Facebook	Apache 2.0	Yes	Linux, macOS, Windows[5]	C++, Python	Python, MATLAB	Yes	Under development[6]	Yes	Yes	Yes[7]	Yes	Yes	No	Yes
Deeplearning4j	Skymind engineering team; Deeplearning4j community; originally Adam Gibson	Apache 2.0	Yes	Linux, macOS, Windows, Android (Cross-platform)	C++, Java	Java, Scala, Clojure, Python (Keras), Kotlin	Yes	On roadmap[8]	Yes[9][10]	Computational Graph	Yes[11]	Yes	Yes	Yes	Yes[12]
Dlib	Davis King	Boost Software License	Yes	Cross-Platform	C++	C++	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes
Intel Data Analytics Acceleration Library	Intel	Apache License 2.0	Yes	Linux, macOS, Windows on Intel CPU[13]	C++, Python, Java	C++, Python, Java[13]	Yes	No	No	Yes	No		Yes		Yes
Intel Math Kernel Library	Intel	Proprietary	No	Linux, macOS, Windows on Intel CPU[14]		C[15]	Yes[16]	No	No	Yes	No	Yes[17]	Yes[17]		No
Keras	François Chollet	MIT license	Yes	Linux, macOS, Windows	Python	Python, R	Only if using Theano as backend	Under development for the Theano backend (and on	Yes	Yes	Yes[18]	Yes	Yes	Yes	Yes[19]

								roadmap for the TensorFlow backend)							
MatConvNet	Andrea Vedaldi, Karel Lenc	BSD license	Yes	Windows, Linux[20] (macOS via Docker on roadmap)	C++	MATLAB, C++,	No	No	Yes	Yes	Yes	Yes	Yes	No	Yes
MATLAB + Neural Network Toolbox	MathWorks	Proprietary	No	Linux, macOS, Windows	C, C++, Java, MATLAB	MATLAB	No	No	Train with Parallel Computing Toolbox and generate CUDA code with GPU Coder[21]	No	Yes[22][23]	Yes[22]	Yes[22]	No	With Parallel Computing Toolbox[24]
Microsoft Cognitive Toolkit (CNTK)	Microsoft Research	MIT license[25]	Yes	Windows, Linux[20] (macOS via Docker on roadmap)	C++	Python (Keras), C++, Command line,[26] BrainScript[27] (.NET on roadmap[28])	Yes[29]	No	Yes	Yes	Yes[30]	Yes[31]	Yes[31]	No[32]	Yes[33]
Apache MXNet	Apache Software Foundation	Apache 2.0	Yes	Linux, macOS, Windows,[34][35] AWS, Android,[36] iOS, JavaScript[37]	Small C++ core library	C++, Python, Julia, Matlab, JavaScript, Go, R, Scala, Perl	Yes	On roadmap[38]	Yes	Yes[39]	Yes[40]	Yes	Yes	Yes	Yes[41]
Neural Designer	Artelnics	Proprietary	No	Linux, macOS, Windows	C++	Graphical user interface	Yes	No	No	?	?	No	No	No	?
OpenNN	Artelnics	GNU LGPL	Yes	Cross-platform	C++	C++	Yes	No	Yes	?	?	No	No	No	?

PaddlePaddle	Baidu Paddle-Paddle team	Apache 2.0	Yes	Linux, macOS, Android,[42] Raspberry Pi[43]	C++, Go	C/C++, Python	Yes	No	Yes	Yes	Yes[44]	Yes	Yes	No	Yes
PyTorch	Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan	BSD license	Yes	Linux, macOS, Windows,[45]	Python, C, CUDA	Python									
Apache SINGA	Apache Incubator	Apache 2.0	Yes	Linux, macOS, Windows	C++	Python, C++, Java	No	No	Yes	?	Yes	Yes	Yes	Yes	Yes
TensorFlow	Google Brain team	Apache 2.0	Yes	Linux, macOS, Windows[46]	C++, Python	Python (Keras), C/C++, Java, Go, R[47]	No	On roadmap[48] but already with SYCL[49] support	Yes	Yes[50]	Yes[51]	Yes	Yes	Yes	Yes
Theano	Université de Montréal	BSD license	Yes	Cross-platform	Python	Python (Keras)	Yes	Under development[52]	Yes	Yes[53][54][1]	Through Lasagne's model zoo[55]	Yes	Yes	Yes	Yes[56]
Torch	Ronan Collobert, Koray Kavukcuoglu, Clement Farabet	BSD license	Yes	Linux, macOS, Windows,[57] Android,[58] iOS	C, Lua	Lua, LuaJIT,[59] C, utility library for C++/OpenCL[60]	Yes	Third party implementations[61][62]	Yes[63][64][1]	Through Twitter's Autograd[65]	Yes[66]	Yes	Yes	Yes	Yes[67]
Wolfram Mathematica	Wolfram Research	Proprietary	No	Windows, macOS, Linux, Cloud computing	C++	Wolfram Language	No	No	Yes	Yes	Yes[68]	Yes	Yes	Yes	Yes
LaonSill	Laonbud	Apache 2.0	Yes	Linux, Cloud computing	C++	Python	No	No	Yes	No	Yes[69]	No	Yes	No	Yes

Appendix G – AI Wikipedia Resources

Artificial intelligence – Machine Learning – Deep Learning

Theory

Bias-variance dilemma – Computational learning theory – Empirical risk minimization – Occam learning – PAC learning – Statistical learning – VC theory

Problems

Classification – Clustering – Regression – Anomaly detection – Association rules – Reinforcement learning – Structured prediction – Feature engineering – Feature learning – Online learning – Semi-supervised learning – Unsupervised learning – Learning to rank – Grammar induction

Supervised Learning

Decision trees – Ensembles (Bagging, Boosting, Random forest) - k-NN – Linear regression – Naive Bayes – Neural networks – Logistic regression – Perceptron – Relevance vector machine (RVM) - Support vector machine (SVM)

Clustering

BIRCH – CURE – Hierarchical – k-means – Expectation–maximization (EM) -
DBSCAN – OPTICS – Mean-Shift

Dimensionality Reduction

Factor analysis – CCA – ICA – LDA – NMF – PCA – t-SNE

Structured Prediction

Graphical models (Bayes net, CRF, HMM)

Anomaly Detection

k-NN – Local outlier factor

Artificial Neural Networks

Auto-encoder – Deep learning – Multilayer perceptron – RNN – Restricted Boltzmann machine – SOM – Convolutional neural network

Reinforcement Learning

Q-learning – SARSA – Temporal difference (TD)

Machine Learning Venues

NIPS – ICML – ML – JMLR – ArXiv:cs. LG

Related Articles

List of datasets for machine-learning research – Outline of machine learning

References

Most of the book is a collation effort to have a curated sequenced resource for reading and learning Deep Learning and Artificial Intelligence. However, there are elements added/modified to fit the purpose of the book references are provided in GitHub repo @ https://meetnavpk.github.io/references/Links.txt

Majority Content is gathered from:

- Microsoft Azure Portal Documentation - Microsoft Azure Documentation
- Deep Learning for Java - Eclipse Deep Learning 4 Java (Skymind)
- IBM Learning - IBM Learning on Cognitive Computing
- Wikipedia Articles – Whole section in (Appendix C)
- Etc. - References are in the GitHub link above.