SAPIENZA
UNIVERSITÀ DI ROMA

Applications of Deep learning for the field of Anomaly detection

Facoltà Ingegneria dell'informazione, Informatica e Statistica

Corso di Laurea Magistrale in Data Science

Candidate

Vamsi Krishna Varma Gunturi

ID number 1794653

Thesis Advisor

Prof. Filomena Maggino

Co-Advisor

Dr. Francesco Pugliese

Academic Year 2019/2020

Thesis defended on 23 July 2020
in front of a Board of Examiners composed by:

Prof. Tardella Luca (chairman)
Prof. Bressan Marco
Prof. Chatzigiannakis Ioannis
Prof. Cincotti Febo
Prof. Maggino Filomena
Prof. Petti Manuela
Prof. Scardapane Simone

Applications of Deep learning for the field of Anomaly detection
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: gunturi.1794653@studenti.uniroma1.it

To my parents

Abstract

Anomaly detection has been one of the cornerstone requirement for analysis of trends, patterns or unusual behaviors by which we can better understand the uncertain future phenomenon. In this modern fast-paced era of machine learning and GPU's, the identification of anomalies has been a recurring need of any business model, but the applications on this field has been either too basic thereby of limited value or too complex there by introducing additional challenges of technological adaptability and barriers of entry.

This research is aimed to bring together these 2 extremes of ease of usage with a simple easy to use interface and complexity in the implementation using the state of the art machine learning and deep learning methodologies. In other words, the aim of this research is to build a general-purpose automated machine learning (Auto ML) engine for anomaly detection for a given dataset. Through this research, we were able to automate the process of data discovery, data exploration, exploratory data analysis to cater the power of latest advancement of technological sphere around artificial intelligence to a broader audience with zero to limited technological barriers. We created a full fledged machine learning workflow and data pipeline to accomplish the above objectives of automation and availability to a broader audience.

Through this research we created a simple and easy to use interface with API layer integrated with database and machine learning engines to create a loosely coupled distributed architecture. Also, we tried to include best practices like continuous integration and continuous delivery (CI/CD) using Jenkins to scale the engine to cater to frequent changes and adaptation based on host of dataset specific customer needs which is still in integration phase. We made use of Docker, a container solution to improve Machine learning reproducibility which is a crucial aspect of any Machine learning application. We analyzed different cloud providers like Heroku which is a Platform as a Service (PaaS) paradigm, Google cloud platform (GCP) and Microsoft Azure which are Infrastructure as Service (IaaS) paradigms to facilitate easy deployment of machine learning models so that there is less turnaround time between new updates and their availability to different stakeholders, there by providing a fully cloud based solution that can be accessed from anywhere with easy migration process. This is just a proof of concept(PoC) or prototype version of the eventual platform that we are aiming to develop in future. For this proof of concept (PoC) we used a intrusion detection dataset from Kaggle to identify intrusions on new data based on the trained model.

As a future road-map, we plan to adapt this platform to other similar areas like Predictive analytics, Face recognition, Video surveillance, Satellite imagery, Intrusion detection, Malware detection and Spyware detection. We also are looking to scale this application to real time predictions [76] and support for massive data sets through continual learning.

Below is a look-up reference for the thesis structure,

In Chapter 1 we briefly present the fundamentals of what anomalies are, types of anomalies, techniques for detecting anomalies, anomaly detection workflow and some details about bottleneck and use-cases of anomaly detection. Through this chapter we intend to provide some foundational background on anomaly detection which is relevant to our problem statement

In Chapter 2 we briefly present the workflow and/or life-cycle of a machine learning project. In short we tried to emphasize on each step right from data collection to how we can tune our hyper-parameters to improve our model performance

In Chapter 3 we briefly present the fundamental concepts of deep learning i.e.., artificial neural networks, activation functions, optimization algorithms, several deep learning network architectures like CNN (convolutional neural network), RNN (Recurrent neural networks), Auto-encoders to name a few. We also cover the regularization methods that we used to avoid over-fitting in our models.

In Chapter 4 we briefly summarized about what automated machine learning means and how it is different from standard approach. We also specified certain aspects targetted by Auto ML and also avaiable tools for Auto ML. To conclude this chapter, we mentioned some limitations of Automated machine learning.

In Chapter 5 we briefly summarized the Anomal detection platform aka ANOMA we put together in our research. In this chapter we tried to include important aspects about technical ingredients, models that we used and other relevant information about ANOMA platform

In Chapter 6 we briefly put together some screenshots and their corresponding descriptions of the ANOMA platform's interface.

In Chapter 7 we briefly summarized the high points that we achieved during this research and key takeaways on the USP(unique selling proportion) of this product.

In Roadmap we included the high-level details about how this research work can be scaled to multiple use cases of detecting anomalies or machine learning needs in general.

Contents

Chapter 1

Anomaly detection

1.1 What is anomaly detection ?

Anomaly detection [19] refers to the problem of finding patterns in data that do not conform to expected behavior. These non-conforming patterns are often referred to as anomalies, outliers, discordant observations, exceptions, aberrations, surprises, peculiarities or contaminants in different application domains. Of these, anomalies and outliers are two terms used most commonly in the context of anomaly detection sometimes interchangeably. Anomaly detection finds extensive use in a wide variety of applications such as fraud detection for credit cards, insurance or health care, intrusion detection for cyber-security, fault detection in safety critical systems, and military surveillance for enemy activities. The problem of anomaly detection is a very challenging problem often faced in data analysis. Whether it is about clustering, classification or some other machine learning problem, it is of great importance to identify anomalies and handle them in some way, in order to achieve optimal model performances. Furthermore, anomalies could often influence the analysis results, which could be the cause of drawing wrong conclusions, influencing important business decisions. Thus, in every data analysis, it is required to accurately define anomalous behaviour pertaining to a certain domain, apply appropriate anomaly detection model, extract anomalies from the rest of the data and then continue with the analysis, model application and drawing conclusions.

1.2 What are anomalies ?

Anomalies [19] are patterns in data that do not conform to a well defined notion of normal behavior. Although anomalies are often considered to be some kind of irregularities inserting the noise in the data, they can contain more information within, than it is earlier believed. The "interestingness" or real life relevance of anomalies is a key feature of anomaly detection

1.3 Anomaly vs Outlier

. The term "anomaly" is often used as a synonym for the term "outlier" but there is a world of difference on how we can interpret both of them i.e.., an outlier is an instance that differs significantly from the rest of the instances based on its values, or happens randomly and rarely in comparison with the rest of the instances, and thus, it could be considered irrelevant for the analysis, while an anomaly is a

behaviour that is different than expected in relation to some previously recorded behavior and requires a deeper dive and cause analysis.

For example, in some telecommunication network, we can have a cell that is overloaded, and in this case – it is considered to be an outlier among other cells in the network. But if it happens that in the same network for some period of time several different cells have the problem of overload, they as a group can represent an anomaly that occurred, for example, due to some link failure, or congestion on the link that is connecting them.

so in a nutshell, Anomaly represents the type of behaviour in the data that differs from some expected behaviour. While the outlier can be interpreted as an instance that deviates from the rest of the instances, with no meaning, whose behaviour could easily be explained and thus ignored and removed, anomalies represent grouped or correlated outliers, deviations having a deeper cause different from simple human mistake or wrong reading, irregularities that are not so easily detected and explained, since they're usually hidden among normal instances.

1.4 Anomaly detection workflow

Anomaly detection process [23] should include these following steps, in order to be carried out in the right way:

1. Understanding the research domain
   (a) learning the basic concepts of the matter being analysed
   (b) consultations with the domain expert
   (c) determining and defining the term "anomaly" in a given domain
2. Understanding the data
   (a) descriptive analysis – describing the data and getting basic insights into behaviour
   (b) exploratory analysis – detecting the hidden relationships and dependencies among features and getting more detailed insights into data
3. Determining the set of techniques that could be used for the process
   (a) supervised learning techniques
   (b) unsupervised learning techniques
   (c) semi-supervised learning techniques
4. Choosing the model
5. Applying the model
6. Evaluating the model
7. Interpreting the detected anomalies
8. Drawing conclusions

Figure 1.1 shows the key components of a Anomaly detection system [19],

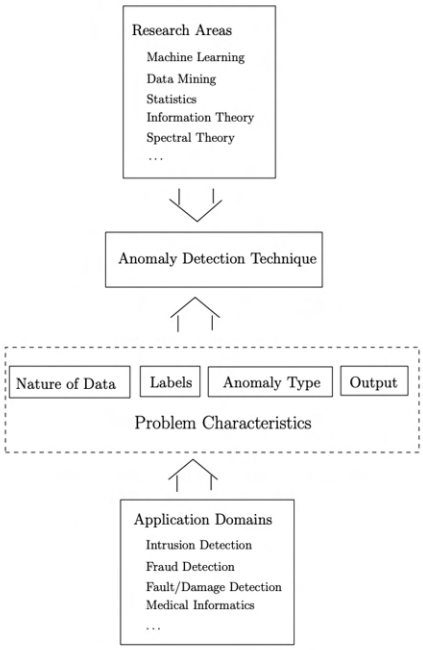


Figure 1.1. Key components associated with an anomaly detection technique

1.5 Types of Anomalies

Anomalies could be grouped into three following classes [23]:

- Point anomalies
- Contextual anomalies
- Collective anomalies

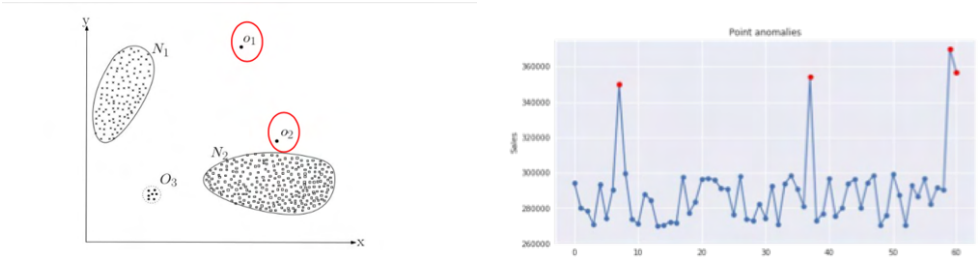
1.5.1 Point anomaly

Point anomaly is an instance that could be considered as anomalous among other instances in the dataset. Point anomalies often represent some extremum, irregularity or deviation that happens randomly and have no particular meaning. On the time series graph below in Figure 1.2, red points represent isolated point anomalies.

1.5.2 Contextual anomaly

Contextual anomaly is an instance that could be considered as anomalous in some specific context. This means that observing the same point through different contexts will not always give us the indication of anomalous behavior. The contextual anomaly is determined by combining contextual and behavioural features. For contextual features, time and space are most frequently used, while the behavioral features depend on the domain that is being analysed – amount of money spent, average temperature, or some other quantitative measure that is being used as a feature.

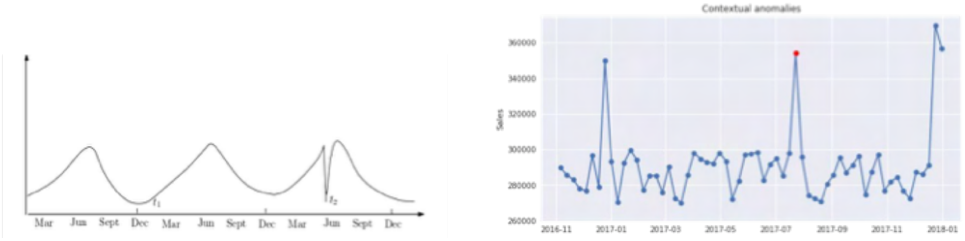
If we add some contextual feature, like time dimension, the same time series will look like following. Similar values are differently flagged for different time periods.



(a) *Example 1* **(b)** *Example 2*

Figure 1.2. Point anomalies [19]

If the time series below shows sales for each month, seasonal peaks in the holidays' period (December) represent a growth in sales due to higher number of purchases, As shown in figure 1.3 while the peak in July, 2017, represent unexpected, anomalous growth, that requires deeper analysis in order to be explained – the cause could be some sale action, music festival, or sport event. There are anomaly detection libraries that have the possibility to receive date ranges that are expected to have extreme values for some feature that is being analysed.



(a) *Example 1* **(b)** *Example 2*

Figure 1.3. Contextual anomalies [19]

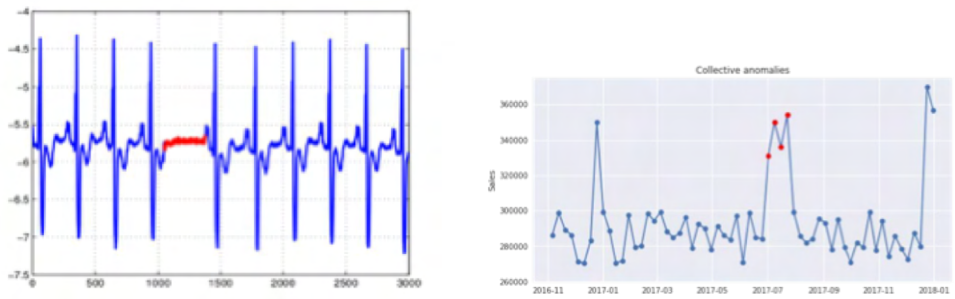
1.5.3 Collective anomaly

Collective anomaly is often represented as a group of correlated, interconnected or sequential instances. While each particular instance of this group doesn't have to be anomalous itself, their collective occurrence is anomalous.

The time series below is pretty similar to the previous one, except that the growth is noted for the whole month of July, 2017 as shown in figure 1.4. Since it is a one time event, there is no doubt that it's about anomalous behaviour. It is very important to stress that contextual and collective anomalies do not require special handling but deeper analysis and root cause identification. Sometimes it is very important to give valuable root cause explanation in order to handle and smooth these kinds of anomalies in a proper way.

1.6 Anomaly detection techniques

Every anomaly detection technique [23] [19] belongs to one of the following basic approaches:



(a) *Example 1* **(b)** *Example 2*

Figure 1.4. Collective anomalies [19]

1.6.1 Supervised anomaly detection

Includes modeling both the normal and anomalous behaviour. It is analogical to supervised approach for classification problem, and it requires labeled data. Model learns on the training data, trying to catch patterns for both types of behaviour, based on the available features. The goal is to obtain a model that could classify each new instance as normal or anomalous, based on the patterns that were caught in the training phase and instance attributes that had been given as an input. This approach includes classification based techniques.

1.6.2 Unsupervised anomaly detection

Involves searching for anomalies with no previous knowledge of the data. It is analogical to unsupervised approach used for clustering, where similar instances are grouped into clusters, based on some similarity measure- whether be it distance, density or the position of the assigned node in a binary tree. Given the assumption that anomalies are well separated from the rest of the data, the goal is to obtain a model that could be able to group instances using the given similarity measure into clusters of normal instances, and anomalies that could as well form more than one cluster. This approach includes techniques based on clustering and nearest neighbour concept.

1.6.3 Semi-supervised anomaly detection

It is a mixture of the previous two types. This approach includes modeling just one type of behaviour, the most frequent – a normal one. It is considered to be semi-supervised since the model learns over the instances belonging to only one class. The advantage is that the model could be incrementally trained, as new instances appear. The goal is to obtain a model that has properly learned patterns of normal behaviour and is able to define a criteria or some kind of limit that will be used to determine whether the instance's behaviour corresponds to learned normal behaviour or not. This approach includes mostly techniques based on statistical methods or novelty detection.

The selection of the right set of techniques [19] depends on the data being available. With labeled data, the decision is pretty simple. A lot could be done when the information of normal and anomalous behavior is given. With partially labeled or unlabeled data, this task is more complex and kind of a state of art. Detailed description of the most famous anomaly detection techniques could be found in [19],

along with assumptions needed to be satisfied for each one of them in order to be used, as well as pros and cons, and required computer complexity.

1.7 Methods for Anomaly detection

Primarily there are 5 main anomaly detection methods [66],

- Distance-based methods
- Ensemble-based methods
- Statistical methods
- Domain-based methods
- Reconstruction-based methods

Figure 1.5 shows different techniques that fall under each of these 5 methods [8],

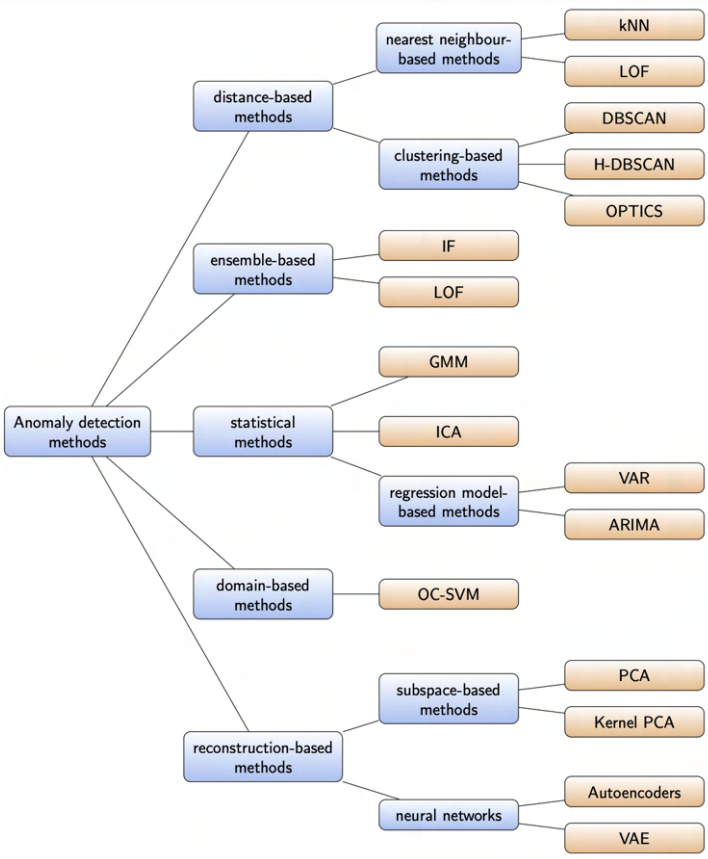


Figure 1.5. Taxonomy of classical anomaly detection methods

Note: In this particular research we mainly concentrated on distance-based methods and reconstruction based methods specifically neural networks.

1.8 Deep Anomaly detection(DAD)

In the broader field of machine learning, the recent years have witnessed a proliferation of deep neural networks, with unprecedented results across various application domains. Deep learning is a subset of machine learning that achieves good performance and flexibility by learning to represent the data as a nested hierarchy of concepts within layers of the neural network. Deep learning outperforms [1] the traditional machine learning as the scale of data increases as illustrated in Figure 1.6. More details on deep learning in Chapter 3

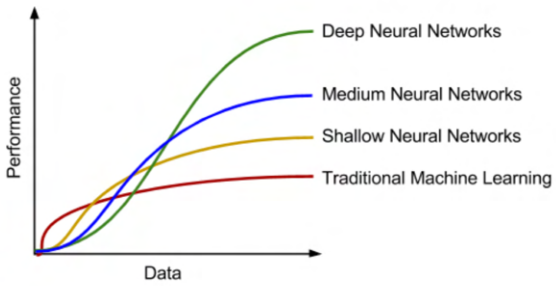


Figure 1.6. Performance Comparison of Deep learning-based algorithms Vs Traditional Algorithms

In recent years, deep learning-based anomaly detection algorithms have become increasingly popular and have been applied for a diverse set of tasks as illustrated in Figure 1.7; studies have shown that deep learning completely surpasses traditional methods.

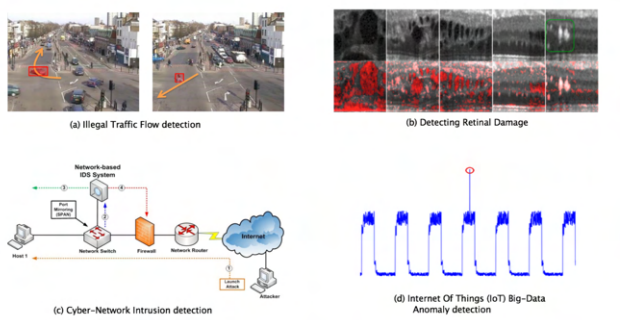


Figure 1.7. Applications Deep learning-based anomaly detection algorithms

1.8.1 Motivation and Challenges

Below are some of the motivations and corresponding challenges of Deep anomaly detection (DAD) techniques [17],

- Performance of traditional algorithms in detecting outliers is sub-optimal on the image (e.g. medical images) and sequence datasets since it fails to capture complex structures in the data.

Type of Data	Examples	DAD model architecture
Sequential	Video,Speech, Protein Sequence,Time Series Text (Natural language)	CNN, RNN, LSTM
Non- Sequential	Image,Sensor Other (data)	CNN, AE and its variants

Table 1.1. Table illustrating nature of input data and corresponding deep anomaly detection model architectures

- Need for large-scale anomaly detection: As the volume of data increases let's say to gigabytes then, it be- comes nearly impossible for the traditional methods to scale to such large scale data to find outliers.
- Deep anomaly detection (DAD) techniques learn hierarchical discriminative features from data. This automatic feature learning capability eliminates the need of developing manual features by domain experts, therefore advocates to solve the problem end-to-end taking raw input data in domains such as text and speech recognition.
- The boundary between normal and anomalous (erroneous) behavior is often not precisely defined in several data domains and is continually evolving. This lack of well-defined representative normal boundary poses challenges for both conventional and deep learning-based algorithms.

1.8.2 Key components of DAD

Figure 1.8 shows the key components associated with Deep Anomaly detection(DAD) [17],

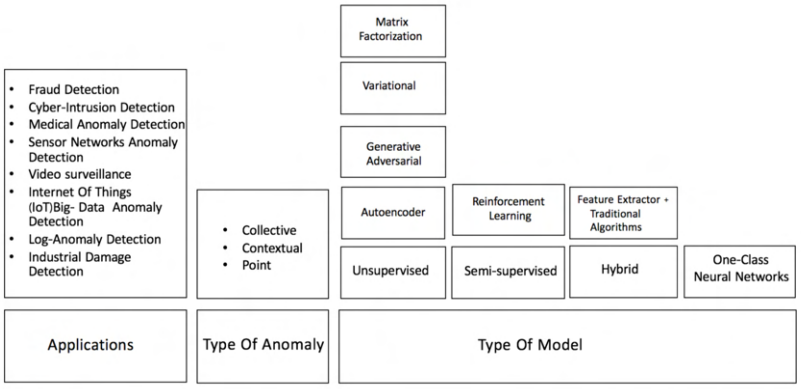


Figure 1.8. Key components associated with deep learning-based anomaly detection technique.

1.8.3 DAD based on nature of data

Table 1.1 illustrates Deep learning architectures applied based on the nature of input data to be analyzed. More details about deep learning model architectures in Chapter 3

1.8.4 Output of DAD Techniques

A critical aspect for anomaly detection methods is the way in which the anomalies are detected [17]. Generally, the outputs produced by anomaly detection methods are either anomaly score or binary labels.

Anomaly Score

Anomaly score describes the level of outlierness for each data point. The data instances may be ranked according to anomalous score, and a domain-specific threshold (commonly known as decision score) will be selected by subject matter expert to identify the anomalies. In general, decision scores reveal more information than binary labels. For instance, in Deep SVDD approach the decision score is the measure of the distance of data point from the center of the sphere, the data points which are farther away from the center are considered anomalous [75]

Labels

Instead of assigning scores, some techniques may assign a category label as normal or anomalous to each data instance. Unsupervised anomaly detection techniques using auto-encoders measure the magnitude of the residual vector (i,e reconstruction error) for obtaining anomaly scores, later on, the reconstruction errors are either ranked or thresholded by domain experts to label data instances.

1.8.5 Applications of DAD

Following are some of the applications of deep anomaly detection[17],

- Intrusion Detection
- Deep learning for Anomaly detection in Social Networks
- Log Anomaly Detection
- Internet of things (IoT) Big Data Anomaly Detection
- Industrial Anomalies Detection
- Anomaly Detection in Time Series
- Video Surveillance

Note: General use-cases of anomaly detection are mentioned in 1.10

1.8.6 Transfer Learning based anomaly detection

Deep learning for long has been criticized for the need to have enough data to produce good results. Both Litjens et al. [55] and Pan et al. [63] present the review of deep transfer learning approaches and illustrate their significance to learn good feature representations. Transfer learning is an essential tool in machine learning to solve the fundamental problem of insufficient training data. It aims to transfer the knowledge from the source domain to the target domain by relaxing the assumption that training and future data must be in the same feature space and have the same distribution. Deep transfer representation-learning has been explored by (Andrews et al. [5], Vercruyssen et al. [17], Li et al. [52], Almajai et al. [3], Kumar and

Vaidehi [47], Liang et al. [53]) are shown to produce very promising results. The open research questions using transfer learning for anomaly detection is, the degree of transfer-ability, that is to define how well features transfer the knowledge and improve the classification performance from one task to another.

1.8.7 Deep Reinforcement Learning based anomaly detection

Deep reinforcement learning (DRL) methods have attracted significant interest due to its ability to learn complex behaviors in high-dimensional data space. Efforts to detect anomalies using deep reinforcement learning have been proposed by de La Bourdonnaye et al. [22], Chengqiang Huang [37]. The DRL based anomaly detector does not consider any assumption about the concept of the anomaly, the detector identifies new anomalies by consistently enhancing its knowledge through reward signals accumulated. DRL based anomaly detection is a very novel concept which requires further investigation and identification of the research gap and its applications.

1.9 Bottlenecks for Anomaly detection

1.9.1 Swamping

Swamping refers to wrongly identifying normal instances as anomalies. This can happen when normal instances are too close to anomalies

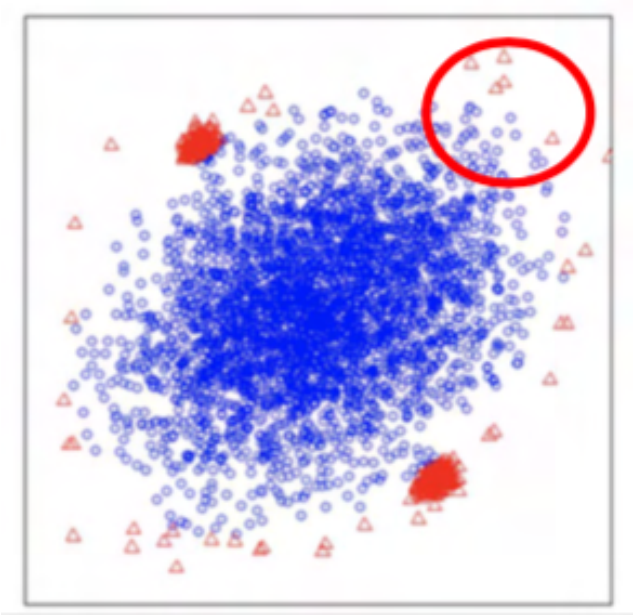


Figure 1.9. Swamping

1.9.2 Masking

Masking is the existence of too many anomalies concealing their own presence. This can happen when an anomaly cluster is large and dense

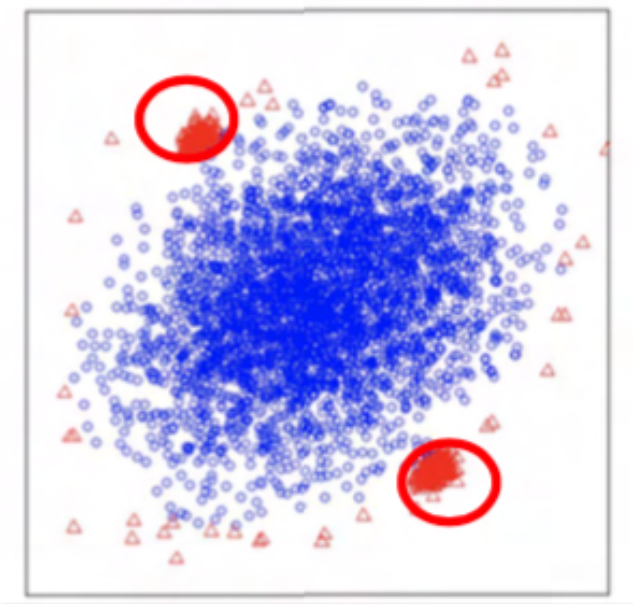


Figure 1.10. Masking

1.10 Use-cases for Anomaly detection

- **Fraud detection** - detecting fraudulent applications for credit cards, state benefits or detecting fraudulent usage of credit cards or mobile phones.
- **Loan application processing** - to detect fraudulent applications or potentially problematic customers.
- **Intrusion detection** - detecting unauthorised access in computer networks.
- **Activity monitoring** - detecting mobile phone fraud by monitoring phone activity or suspicious trades in the equity markets.
- **Network performance** - monitoring the performance of computer networks, for example to detect network bottlenecks.
- **Fault diagnosis** - monitoring processes to detect faults in motors, generators, pipelines or space instruments on space shuttles for example
- **Structural defect detection** - monitoring manufacturing lines to detect faulty production runs for example cracked beams.
- **Satellite image analysis** - identifying novel features or misclassified features.
- **Detecting novelties in images** - for robot neotaxis or surveillance systems.
- **Motion segmentation** - detecting image features moving independently of the background.
- **Time-series monitoring** - monitoring safety critical applications such as drilling or high-speed milling.
- **Medical condition monitoring** - such as heart-rate monitors.

- **Pharmaceutical research** - identifying novel molecular structures.

Figure 1.11. Usecases for Anomaly detection

Chapter 2

Machine Learning workflow

2.1 What is Machine Learning ?

Machine Learning [4] is a subfield of computer science that is concerned with building algorithms which, to be useful, rely on a collection of examples of some phenomenon. The term Machine Learning was coined by Arthur Samuel in 1959, an American pioneer in the field of computer gaming and artificial intelligence and stated that "it gives computers the ability to learn without being explicitly programmed". Machine learning is a tool for turning information into knowledge. The hidden patterns [11] and knowledge about a problem can be used to predict future events and perform all kinds of complex decision making. Machine learning can also be defined as the process of solving a practical problem by Gathering a dataset, Algorithmically building a statistical model based on that dataset.

2.2 Traditional programming vs Machine Learning

Traditionally, software engineering combines human created rules with data to create answers to a problem. Instead, Machine Learning uses data and answers to discover the rules behind a problem. To learn the rules governing a phenomenon, machines have to go through a learning process, trying different rules and learning from how well they perform as shown in Figure 2.2. Hence, why it's known as Machine Learning. [24].

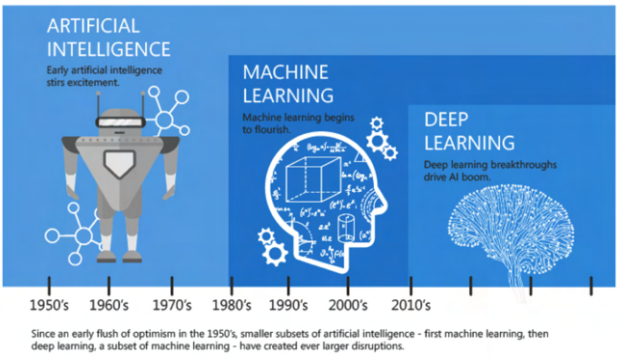
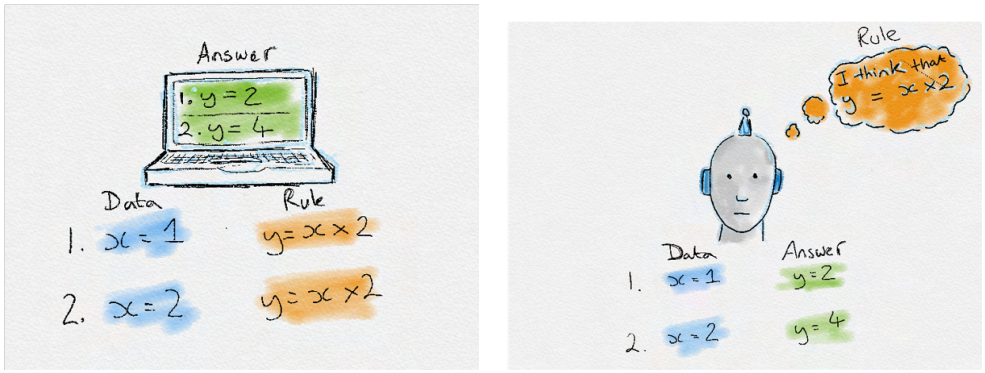


Figure 2.1. Evolution of Artificial Intelligence [62]



(a) *Programming* **(b)** *Machine Learning*

Figure 2.2. Traditional Programming vs Machine Learning

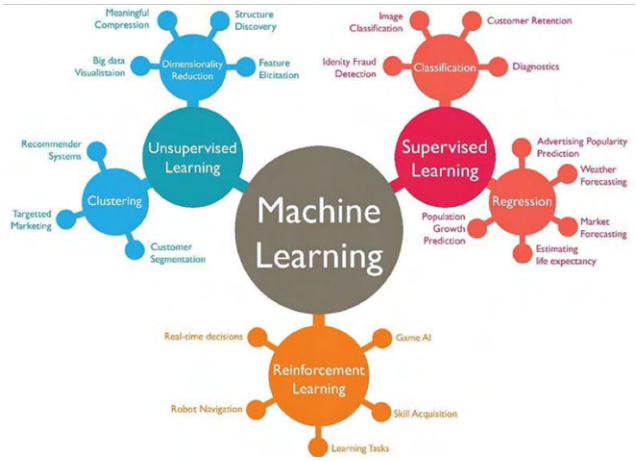


Figure 2.3. Types of Machine Learning [20]

2.3 Types of Machine Learning

There are broadly 4 types of Machine Learning namely,

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Reinforcement learning

Each form of Machine Learning has differing approaches, but they all follow the same underlying process and theory. The No Free Lunch theorem [34] is famous in Machine Learning. It states that there is no single algorithm that will work well for all tasks. Each task that you try to solve has it's own idiosyncrasies. Therefore, there are lots of algorithms and approaches to suit each problems individual quirks.

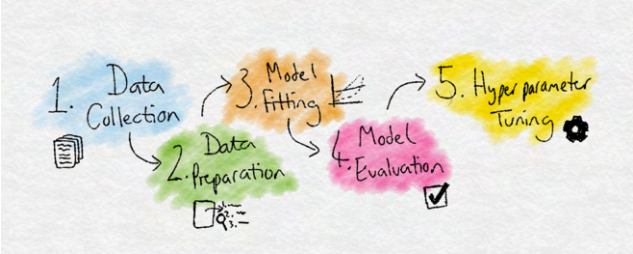


Figure 2.4. Machine Learning workflow [24]

2.4 Machine Learning workflow

In a nutshell, following is a high-level workflow of a Machine Learning project,

- **Data Collection**: Collect the data that the algorithm will learn from.
- **Data Preparation**: Format and engineer the data into aa optimal format, extracting important features and perform dimensionality reduction.
- **Training**: Also known as the fit stage, this is where the Machine Learning algorithm actually learns by passing it the data that has been collected and prepared.
- **Evaluation**: Test the model to see how well it performs.
- **Tuning**: Fine tune the model to maximise it's performance.

We will go in more detail of above steps in the next sections

2.5 Data Pre-processing

Real-world data is often incomplete, inconsistent, and lacking in certain behaviors or trends, and is likely to contain many errors. i.e

- **Incomplete**: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data
- **Noisy**: containing errors or outliers
- **Inconsistent**: containing discrepancies in codes or names

Data pre-processing is a proven method of resolving such issues. It focuses more on Data cleaning, Data integration, Data transformation, Data dimensionality reduction and Data discretization.

2.5.1 Data Wrangling

When we perform data wrangling [58], we are taking our input data from its original state to a format where we can perform meaningful analysis on it. It is also referred to as Data manipulation. There is no set list or order of operations; the only goal is that the data post-wrangling is more useful to us than when we started. There are three common tasks involved in the data wrangling process:

- Data cleaning

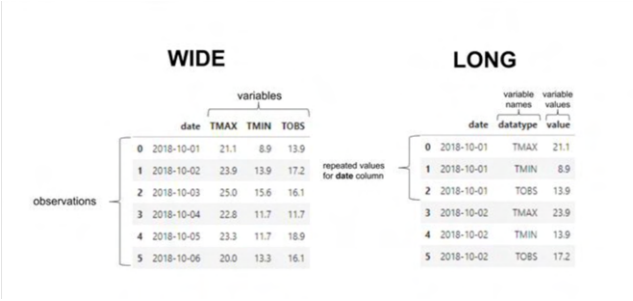


Figure 2.5. Wide vs Long formats

- Data transformation
- Data enrichment

2.5.2 Data Cleaning

An initial round of data cleaning on our data frame will often give us the bare minimum we need to start exploring our data. Some essential data cleaning tasks to master include:

- Columnn renaming
- Sorting and reordering
- Data type conversions
- De-duplicating data
- Addressing missing or invalid data
- Filtering to the desired subset of data

Data cleaning [58] is the perfect starting point for data wrangling since having the data stored as the correct data types and easy-to-reference names will open up many ways for data exploration such as summary statistics, sorting, and filtering.

2.5.3 Data transformation

Frequently, we will reach the data transformation stage after some initial data cleaning, but it is entirely possible that our dataset is unusable in its current shape, and we must restructure it before attempting to do any data cleaning. Here, we mainly focus on changing the structure of data to facilitate our downstream analyses; this usually involves changing which data goes along the rows and which goes down the columns.Most data we will find is either in a wide format or a long format; each of these formats has its merits, and it's important to know which one we will need for our analysis. Often, people will record and present data in the wide format, but there are certain visualizations that require the data to be in the long format. The wide format is preferred for analysis and database design, while the long format is considered poor design because each column should be its own data type and have a singular meaning. When building an API, the long format may be chosen if flexibility is required. Figure 2.5 shows an example for data transformation

2.5.4 Data Enrichment

When we're looking to enrich [58] the data, we can either merge new data with the original data (by appending new rows or columns) or use the original data to create new data.The following are ways to enhance our data using the original data:

- **Adding new columns**: Using functions on the data from existing columns to create new values
- **Binning**: Turning continuous data or discrete data with many distinct values into range buckets, which makes the column discrete while letting us control the number of possible values in the column
- **Aggregating**: Rolling up the data and summarizing it
- **Re-sampling**: Aggregating time series data at specific intervals

2.5.5 Encoding categorical data

There are some algorithms that can work well with categorical data [20], such as decision trees. But most Machine Learning algorithms cannot operate directly with categorical data. These algorithms require the input and output both to be in numerical form. If the output to be predicted is categorical, then after prediction we convert them back to categorical data from numerical data. There are three simple methods of encoding categorical data:

- Replacing
- Label Encoding
- One-Hot Encoding

Replacing

This is a technique in which we replace the categorical data with a number. This is a simple manual replacement and does not involve much logical processing.

Label encoding

This is a technique in which we replace each value in a categorical column with numbers from 0 to N-1. For example, say we've got a list of employee names in a column. After performing label encoding, each employee name will be assigned a numeric label. Label encoding is the best method to use for ordinal data. The scikit-learn library provides **LabelEncoder()**, which helps with label encoding.

One hot encoding

Here, the label-encoded data is further divided into n number of columns. Here, n denotes the total number of unique labels generated while performing label encoding. For example, say that three new labels are generated through label encoding. Then, while performing one-hot encoding, the columns will be divided into three parts. A sample example for one hot encoding is shown in Figure 2.7

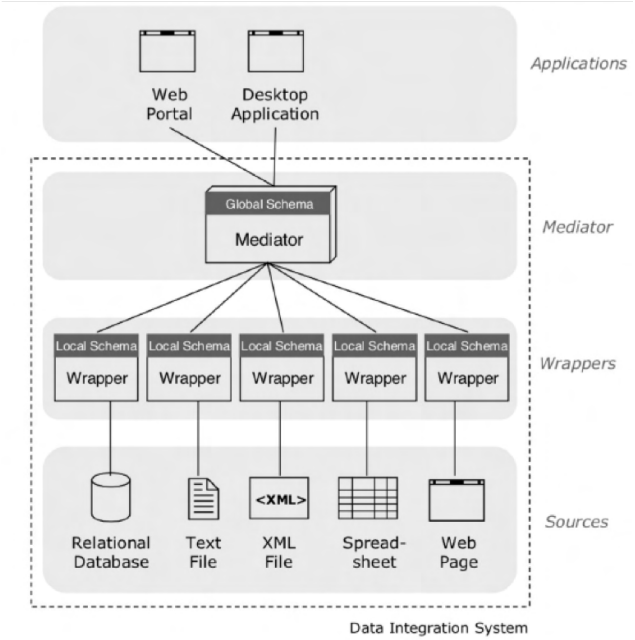


Figure 2.6. View-based data integration system

2.5.6 Data Integration

Data Integration (or Information Integration) [49] is the problem of finding and combining data from different sources. Process of combining data from different sources into a single, unified view. Integration begins with the ingestion process, and includes steps such as cleansing, ETL mapping, and transformation. Data integration ultimately enables analytics tools to produce effective, actionable business intelligence. Abstracting out the differences between individual systems, a typical view-based data integration system (VDIS) conforms to the architecture shown in Figure 2.6 .

- **Sources**: store the data in a variety of formats (relational databases, text files etc..,
- **Wrappers**: solve the heterogeneity in the formats by transforming each source's data model to a common data model used by the integration system. The **wrapped** data sources are usually referred to as local or source databases, the structure of which is described by corresponding local/source schemas. This is in contrast to the unified view exported by the mediator, also called global/target database.
- **Mappings** expressed in a certain mapping language (depicted as lines between the wrapped sources and the mediator) specify the relationship between the wrapped data sources (i.e. the local schemas) and the unified view exported by the mediator (global schema).

2.6 Feature Engineering

The problem of transforming raw data into a data set is called feature engineering. For most practical problems, feature engineering is a labor-intensive process that

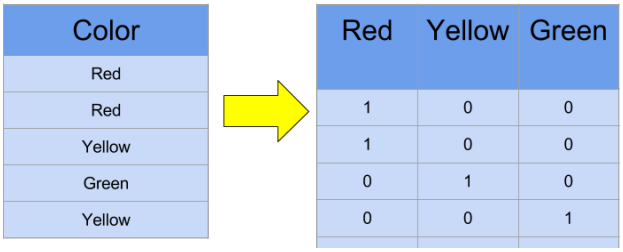


Figure 2.7. One hot encoding

demands from the data analyst a lot of **creativity** and, preferably, domain knowledge. The role of the data analyst is to create informative features: those would allow the learning algorithm to build a model that predicts well labels of the data used for training. Highly informative features are also called features with high predictive power. For example, the average duration of a user's session has high predictive power for the problem of predicting whether the user will keep using the application in the future. We say that a model has a low bias when it predicts the training data well. That is, the model makes few mistakes when we use it to predict labels of the examples used to build the model. Feature engineering involves,

- One hot encoding
- Binning
- Normalization
- Standardization
- Dealing with missing features
- Data imputation techniques

2.6.1 One hot encoding

Some learning algorithms only work with numerical feature vectors. When some feature in your dataset is categorical, like "colors" or "days of the week", you can transform such a categorical feature into several binary ones. If your feature has a categorical feature "colors" and this feature has three possible values: "red", "yellow", "green", you can transform this feature into a vector of three numerical values as shown in Figure 2.7. By doing so, you increase the dimensionality of your feature vectors. It is not efficient to transform red into 1, yellow into 2, and green into 3 to avoid increasing the dimensionality because that would imply that there's an order among the values in this category and this specific order is important for the decision making. If the order of a feature's values is not important, using ordered numbers as values is likely to confuse the learning algorithm because the algorithm will try to find a regularity where there is none, which eventually may lead to over-fitting.

2.6.2 Binning

Binning (also called bucketing) is the process of converting a continuous feature into multiple binary features called bins or buckets, typically based on value range. For example, instead of representing age as a single real-valued feature, the data analyst could create ranges of age into discrete bins: all ages between 0 and 5

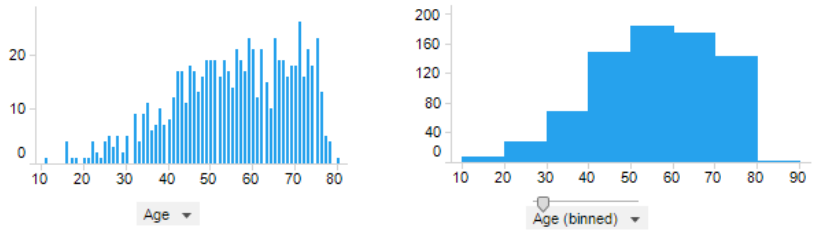


Figure 2.8. Example of binning

years-old could be put into one bin, 6 to 10 years-old could be in the second bin, 11 to 15 years-old could be in the third bin, and so on. In some cases, a carefully designed binning can help the learning algorithm to learn using fewer examples. It happens because we give a "hint" to the learning algorithm that if the value of a feature falls within a specific range, the exact value of the feature does not matter.

2.6.3 Data normalization

Also referred to as Column Normalization. Data Normalization usually means to scale a column or feature to have a value between 0 and 1, and we can achieve that by using the following formula:

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}} [60] \tag{2.1}$$

Example: Let us consider a list of numeric values, x[] = [3.5, 3.0, 3.2, 3.1, 3.6, 3.7, 3.4, 3.4, 2.9, 2.7] and apply data normalization using above equation as follows,
$x_{max} = \max(numAry) = 3.7$
$x_{min} = \min(numAry) = 2.7$
x[] new = [0.7, 0.2, 0.5, 0.3, 0.8, 1.0, 0.6, 0.6, 0.1, 0.0]

2.6.4 Data standardization

Standardization transforms data to have a mean of zero and a standard deviation of 1. Data points can be standardized with the following formula:

$$x_{new} = \frac{x_i - \hat{x}}{S} [60] \tag{2.2}$$

where $\hat{x}$ is the mean and S is the standard deviation

Example: Let us consider a list of numeric values, x[] = [3.5, 3.0, 3.2, 3.1, 3.6, 3.7, 3.4, 3.4, 2.9, 2.7] and apply data standardization using above equation as follows,
$\hat{x}$ = mean(x[]) = 3.25
S = standard deviation(x[]) = 0.324
x[] new = [0.77, -0.77, -0.15, -0.46, 1.08, 1.38, 0.46, 0.46, -1.0, -1.6]

Through data standardization we ensure that the standard deviation of our data points is close enough to 1 and most of the points lie between -1σ and 1σ (as shown below). As 68% of the values lie within one standard deviation of the mean. Here we can say that one standard deviation is ling between -1.5 and 1.5. Column

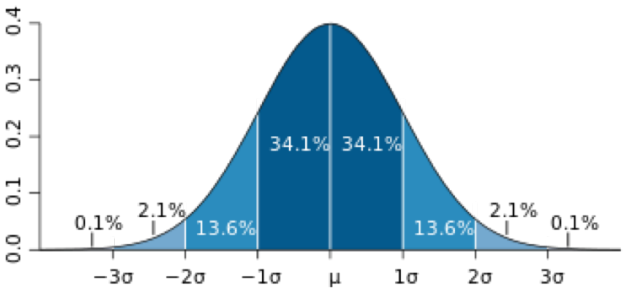


Figure 2.9. Mean centering using Data standardization [60]

Standardization is also called as Mean Centering. Sometimes it's also known as z-score.

2.6.5 Dealing with missing features

In some cases, the data comes to the analyst in the form of a dataset with features already defined. In some examples, values of some features can be missing. That often happens when the dataset was handcrafted, and the person working on it forgot to fill some values or didn't get them measured at all. The typical approaches of dealing with missing values for a feature include:

- Removing the examples with missing features from the dataset (that can be done if your dataset is big enough so you can sacrifice some training examples);
- Using a learning algorithm that can deal with missing feature values (depends on the library and a specific implementation of the algorithm);
- Using a data imputation technique.

2.6.6 Data Imputation techniques

Below are some data imputation techniques,

- One data imputation technique consists in replacing the missing value of a feature by an average value of this feature in the dataset.
- Another technique is to replace the missing value by a value outside the normal range of values. For example, if the normal range is [0, 1], then you can set the missing value to 2 or 1. The idea is that the learning algorithm will learn what is best to do when the feature has a value significantly different from regular values.
- Alternatively, you can replace the missing value by a value in the middle of the range. For example, if the range for a feature is [1, 1], you can set the missing value to be equal to 0. Here, the idea is that the value in the middle of the range will not significantly affect the prediction.
- A more advanced technique is to use the missing value as the target variable for a **regression** problem. You can use all remaining features [x(1), x(2), . . . , x(j1), x(j+1), . . . , x(D)] to form a feature vector $\hat{x}_i$, set $\hat{y}_i \leftarrow x(j)$, where

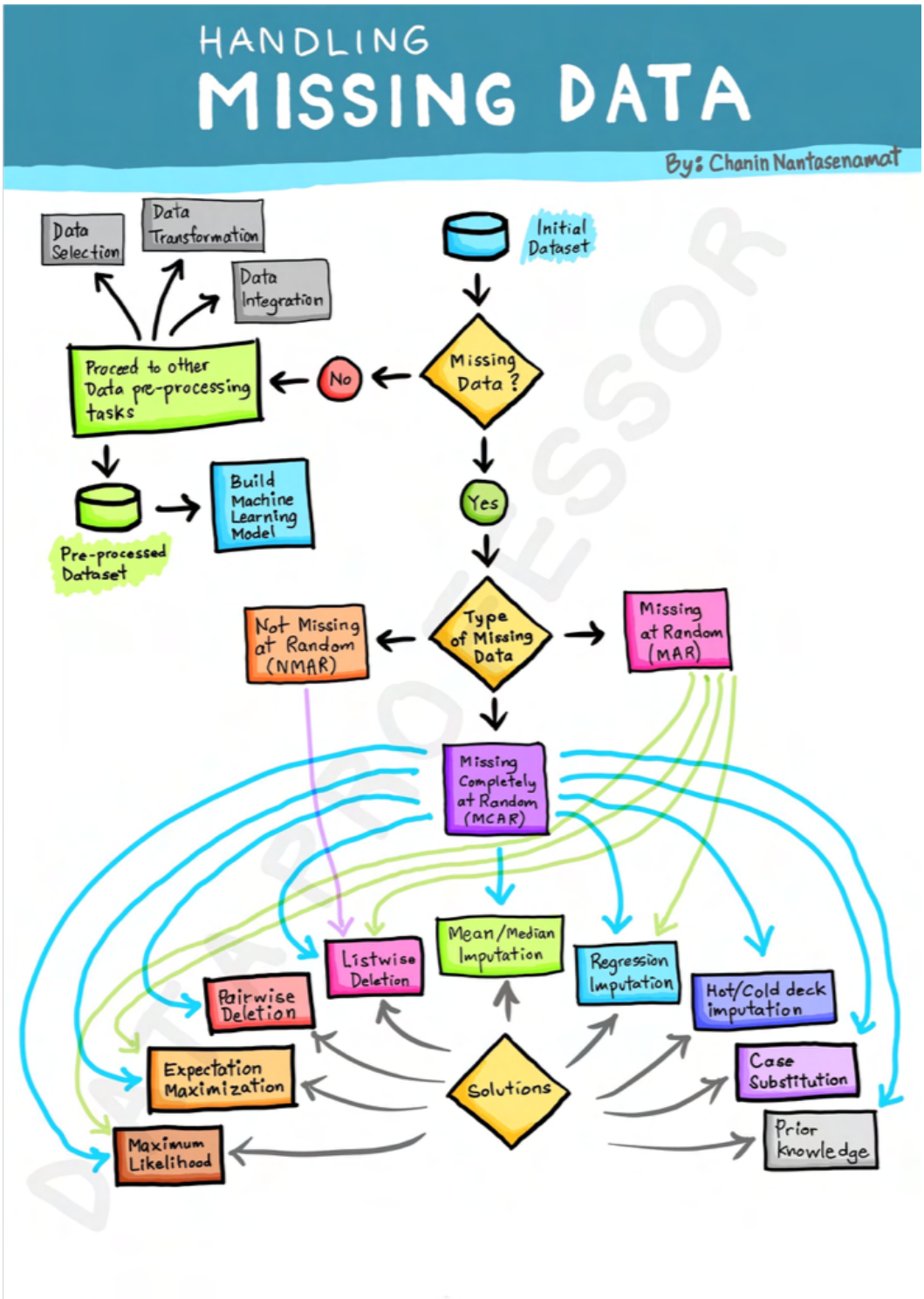


Figure 2.10. Dealing with missing features [67]

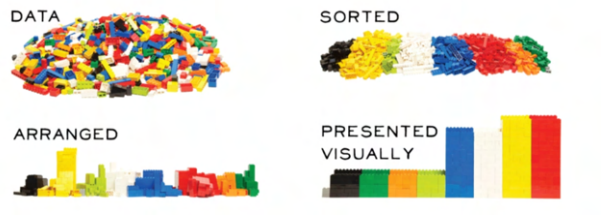


Figure 2.11. EDA in a nutshell [33]

j is the feature with a missing value. Then you build a regression model to predict $\hat{y}$ from $\hat{x}$. Of course, to build training examples ($\hat{x}$, $\hat{y}$), you only use those examples from the original dataset, in which the value of feature j is present.

- Finally, if you have a significantly large dataset and just a few features with missing values, you can increase the dimensionality of your feature vectors by adding a binary indicator feature for each feature with missing values. Let's say feature j = 12 in your D-dimensional dataset has missing values. For each feature vector x, you then add the feature j = D + 1 which is equal to 1 if the value of feature 12 is present in x and 0 otherwise. The missing feature value then can be replaced by 0 or any number of your choice.

Figure 2.10 shows an info-graphic about how we can deal with missing values

Important: At prediction time, if your example is not complete, you should use the same data imputation technique to fill the missing features as the technique you used to complete the training data.

Note:Before you start working on the learning problem, you cannot tell which data imputation technique will work the best. Try several techniques, build several models and select the one that works the best.

2.7 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is the process of visualizing and analyzing data to extract insights from it. In other words, EDA is the process of summarizing important characteristics of data in order to gain better understanding of the dataset.

Methods for EDA:

- Descriptive Statistics
- Grouping of Data
- Handling missing values in dataset
- ANOVA: Analysis of variance Correlation

2.7.1 Descriptive statistics

Descriptive statistics analysis helps to describe the basic features of dataset and obtain a brief summary of the data. The describe() method in Pandas library helps us to have a brief summary of the dataset. It automatically calculates basic statistics for all numerical features. For categorical features, the `value_counts()` method will be useful.

Plots for Descriptive statistics

To analyse the numerical data we can make use of different plots as shown in 2.12 such as,

- Box plot
- Scatter plots
- Histograms

Box-plot: Box plot gives,

- Median of the data, which represents where the middle data point is
- Upper and lower quartiles represent the 75 and 25 percentile of the data respectively
- Upper and lower extremes shows us the extreme ends of the distribution of our data
- Finally, it also represents outliers, which occur outside the upper and lower extremes

Scatter-plot:

- Scatter plots represent each relationship between two continuous variables as individual data point in a 2D graph.

Histogram:

- Shows us the frequency distribution of a variable.
- It partitions the spread of numeric data into parts called as "bins" and then counts the number of data points that fall into each bin.
- So, the vertical axis actually represents the number of data points in each bin.

2.7.2 Grouping of data

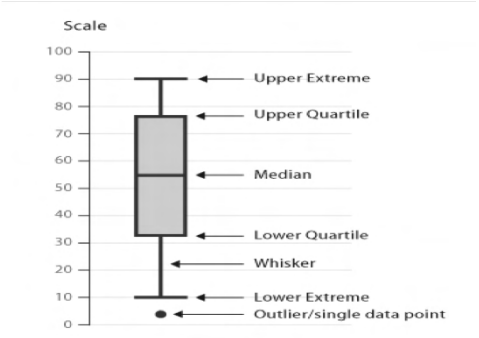
The groupby() method from Pandas library helps us to accomplish this task.

2.7.3 Handling missing values

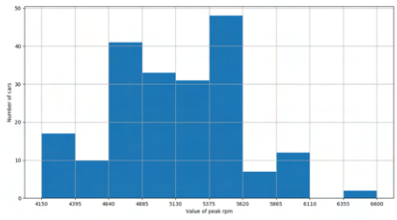
When no data value is stored for a feature in a particular observation, we say this feature has missing values. Examining this is important because when some of your data is missing, it can lead to weak or biased analysis. We can detect missing values by applying isnull() method over the dataframe. The isnull() method returns a rectangular grid of boolean values which tells us if a particular cell in the dataframe has missing value or not.

Using Heatmaps for Missing values

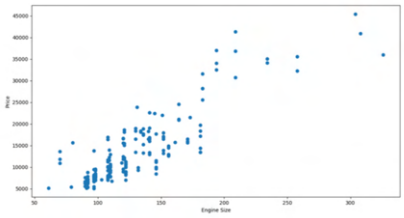
Heatmap takes a rectangular data grid as input and then assigns a color intensity to each data cell based on the data value of the cell. This is a great way to get visual clues about the data. We will generate a heatmap of the output of isnull() in order to detect missing values as shown in 2.14



(a) *Box plot*



(b) *Histogram*



(c) *Scatter*

Figure 2.12. Plots for descriptive statistics [33]

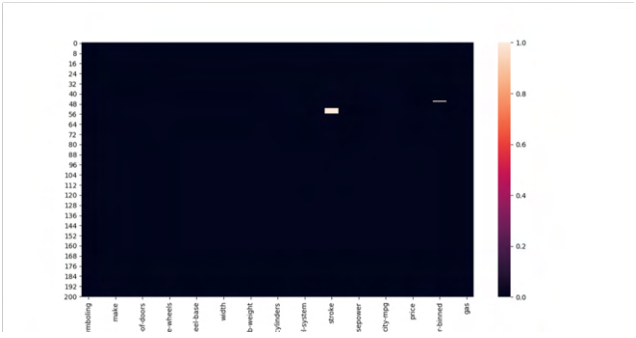


Figure 2.13. Example for heatmap visualization for missing values [33]

What to do with missing values?

We can handle missing values in many ways:

- **Delete**: You can delete the rows with the missing values or delete the whole column which has missing values. The dropna() method from Pandas library can be used to accomplish this task.
- **Impute**: Deleting data might cause huge amount of information loss. So, replacing data might be a better option than deleting. One standard replacement technique is to replace missing values with the average value of the entire column. For example, we can replace the missing values in "stroke" column with the mean value of "stroke" column. The fillna() method from Pandas library can be used to accomplish this task.
- **Predictive filling**: Alternatively, you can choose to fill missing values through predictive filling. The interpolate() method will perform a linear interpolation in order to "guess" the missing values and fill the results in the dataset.
- More details on data imputation techniques can be found in 2.6.6

2.7.4 ANOVA

ANOVA(Analysis of variance) is a statistical method used for figuring out the relation between different groups of categorical data. The ANOVA test, gives us two measures as result:

- **F-test score**: It calculates the variation between sample group means divided by variation within sample group.
- **P value**: It shows us the confidence degree. In other words, it tells us whether the obtained result is statistically significant or not.

Note: The ANOVA test can be performed using the `f_oneway()` method from Scipy library .

2.7.5 Correlation

Correlation is a statistical metric for measuring to what extent different variables are interdependent. In other words, when we look at two variables over time, if one variable changes, how does this effect change in the other variable? For example, smoking is known to be correlated with lung cancer. Since, smoking increases the chances of lung cancer. Another example would be the relationship between the number of hours a student studies and the score obtained by that student. Because, we expect the student who studies more to obtain higher marks in the exam. We can see the correlation between different variables using the **corr()** function. Then we can plot a heatmap over this output to visualize the results as shown in 2.14.

Example: `correlation_matrix` = df.corr()
sns.heatmap(`correlation_matrix`, annot=True)
plt.show()

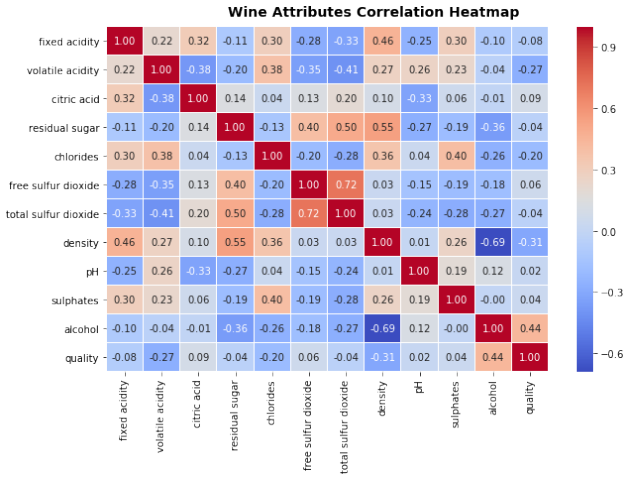


Figure 2.14. Example for heatmaps [54]

2.8 Learning Algorithm Selection

2.8.1 Algorithm selection

Choosing a Machine Learning algorithm [15] can be a difficult task. You can ask yourself several questions before starting to work on the problem. Depending on your answers, you can shortlist some algorithms and try them on your data.

Does your model have to be explainable to a non-technical audience?

- Most very accurate learning algorithms are so-called "black boxes".
- They learn models that make very few errors, but why a model made a specific prediction could be very hard to understand and even harder to explain.
- Examples of such models are neural networks or ensemble models.
- On the other hand, kNN, linear regression, or decision tree learning algorithms produce models that are not always the most accurate, however, the way they make their prediction is very straightforward.

In-memory vs. out-of-memory:

- Can your dataset be fully loaded into the RAM of your server or personal computer? If yes, then you can choose from a wide variety of algorithms.
- Otherwise, you would prefer incremental learning algorithms that can improve the model by adding more data gradually.

Number of features and examples:

- How many training examples do you have in your dataset?
- How many features does each example have?
- Some algorithms, including neural networks and gradient boosting, can handle a huge number of examples and millions of features. Others, like SVM, can be very modest in their capacity.

Categorical vs. numerical features:

- Is your data composed of categorical only, or numerical only features, or a mix of both?
- Depending on your answer, some algorithms cannot handle your dataset directly, and you would need to convert your categorical features into numerical ones.

Training speed:

- How much time is a learning algorithm allowed to use to build a model?
- Neural networks are known to be slow to train. Simple algorithms like logistic and linear regression or decision trees are much faster.
- Specialized libraries contain very efficient implementations of some algorithms; you may prefer to do research online to find such libraries.
- Some algorithms, such as random forests, benefit from the availability of multiple CPU cores, so their model building time can be significantly reduced on a machine with dozens of cores.

Nonlinearity of the data:

- Is your data linearly separable or can it be modelled using a linear model?
- If yes, SVM with the linear kernel, logistic or linear regression can be good choices.
- Otherwise, deep neural networks or ensemble algorithms, might work better.

Prediction speed:

- How fast does the model have to be when generating predictions?
- Will your model be used in production where very high throughput is required?
- Algorithms like SVMs, linear and logistic regression, and (some types of) neural networks, are extremely fast at the prediction time.
- Others, like kNN, ensemble algorithms, and very deep or recurrent neural networks, are slower.

Note: If you don't want to guess the best algorithm for your data, a popular way to choose one is by testing it on the validation set.

2.8.2 3 sets

In practice data analysts work with three distinct sets of labelled examples:

- training set
- validation set
- test set

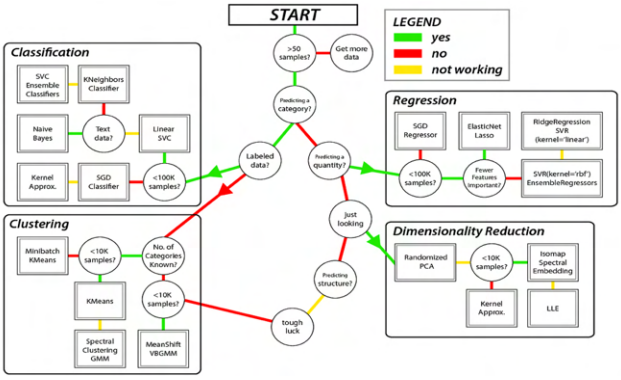


Figure 2.15. Machine Learning Algorithm selection [15]

Once you have got your annotated dataset, the first thing you do is you shuffle the examples and split the dataset into three subsets: training, validation, and test. The training set is usually the biggest one; you use it to build the model. The validation and test sets are roughly the same sizes, much smaller than the size of the training set. The learning algorithm cannot use examples from these two(testing and validation) subsets to build the model. That is why those two sets are often called holdout sets.

There's no optimal proportion to split the dataset into these three subsets. In the past, the rule of thumb was to use 70% of the dataset for training, 15% for validation and 15% for testing. However, in the age of big data, datasets often have millions of examples. In such cases, it could be reasonable to keep 95% for training and 2.5%/2.5% for validation/testing.

Why three sets ?

The answer is simple: when we build a model, what we do not want is for the model to only do well at predicting labels of examples the learning algorithms has already seen. A trivial algorithm that simply memorizes all training examples and then uses the memory to "predict" their labels will make no mistakes when asked to predict the labels of the training examples, but such an algorithm would be useless in practice. What we really want is a model that is good at predicting examples that the learning algorithm didn't see: we want good performance on a holdout set.

Why do we need two holdout sets and not one ?

- We use the validation set to choose the learning algorithm and find the best values of hyper-parameters.
- We use the test set to assess the model before delivering it to the client/customer or putting it in production.

2.8.3 Over-fitting and under-fitting

Under-fitting:

Under-fitting is the inability of the model to predict well the labels of the data it was trained on. There could be several reasons for under-fitting, the most important

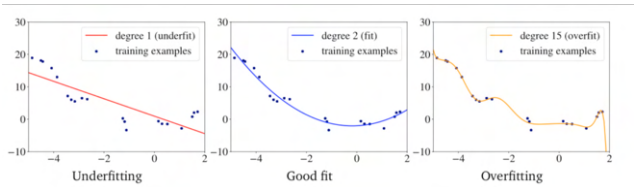


Figure 2.16. Over-fitting and Under-fitting [15]

of which are:

- Your model is too simple for the data (for example a linear model can often under fit) i.e.., the data set can resemble a curved line, but our model is a straight line
- The features you engineered are not informative enough

The second reason(non informative features) can be illustrated like this:

- Let's say you want to predict whether a patient has cancer, and the features you have are height, blood pressure, and heart rate.
- These three features are clearly not good predictors for cancer so our model will not be able to learn a meaningful relationship between these features and the label.

The solution to the problem of under-fitting is to try a more complex model or to engineer features with higher predictive power.

Over-fitting:

The model that overfits predicts very well the training data but poorly the data from at least one of the two holdout sets. Several reasons can lead to over-fitting, the most important of which are:

- Your model is too complex for the data (for example a very tall decision tree or a very deep or wide neural network often over-fit)
- You have too many features but a small number of training examples.

In the literature, you can find another name for the problem of over-fitting: the problem of high variance. This term comes from statistics. The variance is an error of the model due to its sensitivity to small fluctuations in the training set. It means that if your training data was sampled differently, the learning would result in a significantly different model. Which is why the model that over-fits performs poorly on the test data: test and training data are sampled from the data set independently of one another.

Solutions for Overfitting

Several solutions to the problem of overfitting are possible:

- Try a simpler model:

  - Linear instead of polynomial regression or Support vector machine(SVM) with a linear kernel instead of Random Forest, a neural network with fewer layers/units.

- Reduce the dimensionality of examples in the dataset. For example, using one of the dimensionality reduction techniques like PCA
- Add more training data, if possible
- Regularize the model

Note: Regularization is the most widely used approach to prevent overfitting

2.8.4 Regularization

Regularization is the process of introducing additional information in order to prevent over-fitting. In practice, that often leads to slightly higher bias but significantly reduces the variance. This problem is known in the literature as the **bias-variance trade-off**. The two most widely used types of regularization are called L1 and L2 regularization. A regression model that uses L1 regularization technique is called Lasso Regression and model which uses L2 is called Ridge Regression.

The idea is quite simple. To create a regularized model, we modify the objective function by adding a penalizing term whose value is higher when the model is more complex. The key difference between these techniques is that Lasso shrinks the less important feature's coefficient to zero thus, removing some feature altogether. So, this works well for feature selection in case we have a huge number of features.

L1 & L2 regularization

This regularization is more mathematical. It adds a regularization term in order to prevent the coefficients to fit so perfectly on training data. The difference between the two is in how they calculate the weights sum.

To understand this better, let us consider an example of linear regression,

$$\hat{y} = w_1x_1 + w_2x_2 + ... + w_Nx_N + b \tag{2.3}$$

L1 and L2 regularisation owes its name to L1 and L2 norm of a vector w respectively. Here's a primer on norms:

L1 Regularizer

$$||w||_1 = |w_1| + |w_2| + ... + |w_N| \tag{2.4}$$

L2 Regularizer

$$||w||_2 = \sqrt{|w_1^2| + |w_2^2| + ... + |w_N^2|} \tag{2.5}$$

and so on for

Lp Regularizer

$$||w||_p = \sqrt[p]{|w_1^p| + |w_2^p| + ... + |w_N^p|} \tag{2.6}$$

It is the calculation of the loss function that includes these regularisation terms:

Loss function with no regularization

$$Loss = Error(y, \hat{y}) \tag{2.7}$$

Loss function with L_1 regularization

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i| \tag{2.8}$$

Loss function with L_2 regularization

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2 \tag{2.9}$$

The regularisation terms are **constraints** by which an optimisation algorithm must **adhere to** when minimising the loss function, apart from having to minimise the error between the true y and the predicted $\hat{y}$

L1 regularization produces a sparse model, a model that has most of its parameters (in case of linear models, most of w(j)) equal to zero. So L1 makes feature selection by deciding which features are essential for prediction and which are not. That can be useful in case you want to increase model **explainability**. However, if your only goal is to maximize the performance of the model on the holdout data, then L2 usually gives better results. In addition to being widely used with linear models, L1 and L2 regularization are also frequently used with neural networks and many other types of models, which directly minimize an objective function. Neural networks also benefit from two other regularization techniques: dropout and batch-normalization. There are also non-mathematical methods that have a regularization effect: data augmentation and early stopping.

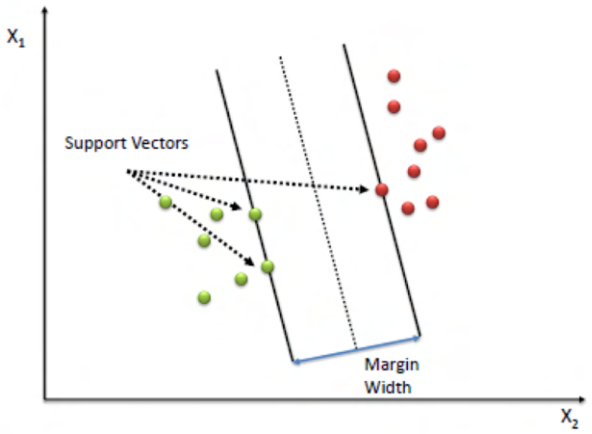
2.9 Machine Learning Algorithms

2.9.1 Support vector machine

SVM creates a line or a hyper-plane which separates the data into multiple classes.Is a supervised Machine Learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. It transforms your data base on it, finds an optimal boundary between the possible outputs. It performs classification by finding the hyper-plane that maximizes the margin between the two classes. The vectors that define the hyper-plane are called the support vectors.

The SVM Algorithm:

- Define an optimal hyperplane with a maximized margin
- Map data to a high dimensional space where it is easier to classify with linear decision surfaces
- Reformulate problem so that data is mapped implicitly into this space



2.9.2 Random Forest

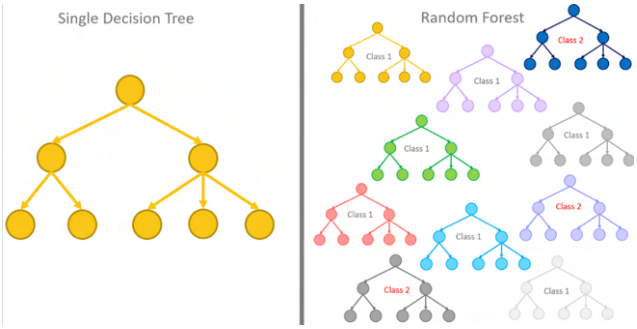
Random forest is an ensemble learning technique about classification, regression and other operations that depend on a multitude of decision trees at the training time. They are fast, flexible, represent a robust approach to mining high-dimensional data and are an extension of classification and regression decision trees we talked about above. Ensemble learning, in general, can be defined as a model that makes predictions by combining individual models. The ensemble model tends to be more flexible with less bias and less variance.Ensemble Learning has two popular methods as:

- **Bagging**: Each individual tree to randomly sample from the dataset and trained by s random subset of data, resulting in different trees
- **Boosting**: Each individual tree /model learns from mistakes made by the previous model and improves

Random forest run times are quite fast.They are pretty efficient in dealing with missing and incorrect data. On the negatives, they cannot predict beyond the defined range in the training data, and that they may over-fit data sets that are particularly noisy. A random forest should have a number of trees between 64–128 trees.

Random Forest vs Decision Tree:

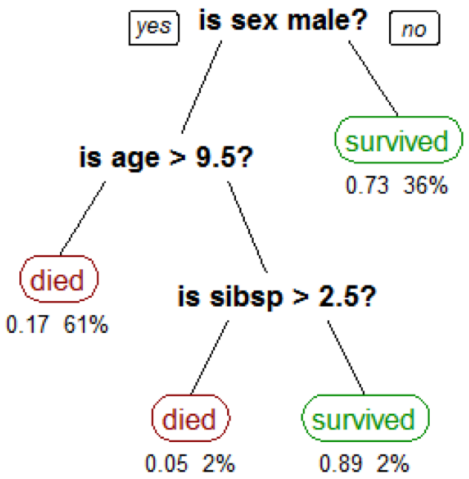
- Random Forest is essentially a collection of Decision Trees.
- A decision tree is built on an entire data set, using all the features/variables of interest, whereas a random forest randomly selects observations/rows and specific features/variables to build multiple decision trees from and then averages the results.



2.9.3 Decision Tree

A decision tree is a decision support tool that uses a tree-like model of decision-making process and the possible consequences. It covers event outcomes, resource costs, and utility of decisions. Resemble an algorithm or a flowchart that contains only conditional control statements.Each decision tree has 3 key parts: a root node, leaf nodes, branches. In a decision tree, each internal node represents a test or an event. Say, a heads or a tail in a coin flip. Each branch represents the outcome of the test and each leaf node represents a class label i.e.., a decision taken after computing all attributes. The paths from root to leaf nodes represent the classification rules. Decision tree can be a powerful Machine Learning algorithm for classification and regression. Classification tree works on the target to classify if it was a heads or a tail. Regression trees are represented in a similar manner, but they predict continuous values like house prices in a neighbourhood. The best part about decision trees:

- Handle both numerical and categorical data
- Handle multi-output problems
- Decision trees require relatively less effort in data preparation
- Nonlinear relationships between parameters do not affect tree performance



2.9.4 Gradient Boosting

GBM is a boosting algorithm used when we deal with plenty of data to make a prediction with high prediction power. Boosting is actually an ensemble of learning algorithms which combines the prediction of several base estimators in order to improve robustness over a single estimator. It combines multiple weak or average predictors to a build strong predictor. These boosting algorithms always work well in data science competitions like Kaggle, AV Hackathon, CrowdAnalytix. Boosting might lead to over fitting if the data sample is very small

Steps of Gradient boosting

- Step 1 : Assume mean is the prediction of all variables.
- Step 2 : Calculate errors of each observation from the mean (latest prediction).
- Step 3 : Find the variable that can split the errors perfectly and find the value for the split. This is assumed to be the latest prediction.
- Step 4 : Calculate errors of each observation from the mean of both the sides of split (latest prediction).
- Step 5 : Repeat the step 3 and 4 till the objective function maximizes/minimizes.
- Step 6 : Take a weighted mean of all the classifiers to come up with the final model.

2.9.5 kNN

Can be used to solve both classification and regression problems. Stores available inputs and classifies new inputs based on a similar measure i.e. the distance function. Has found its major application in statistical estimation and pattern recognition.

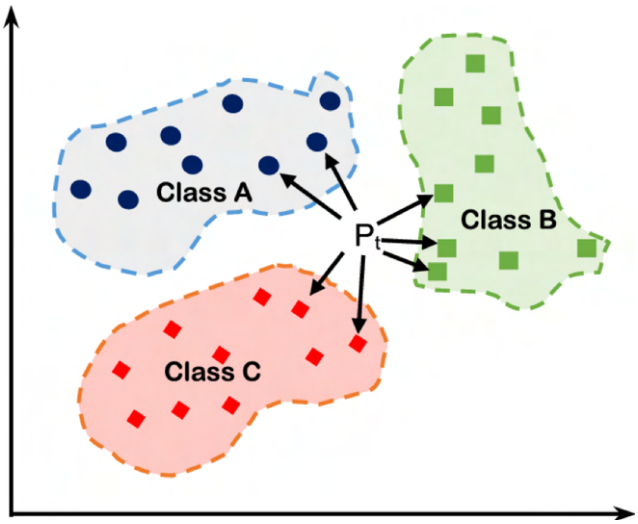
Steps for kNN:

- KNN works by finding the distances between a query and all inputs in the data.
- Next, it selects a specified number of inputs, say K, closest to the query.
- And then it votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression).

kNN Algorithm:

- Load the data
- Initialize k to a chosen number of neighbours in the data
- For each example in the data, calculate the distance between the query example and the current input from the data
- Add that distance to the index of input to make an ordered collection
- Sort the ordered collection of distances and indices in ascending order grouped by distances

- Pick the first K entries from the sorted collection
- Get the labels of the selected K entries
- If regression, return the mean of the K labels; If classification, return the mode of the K labels

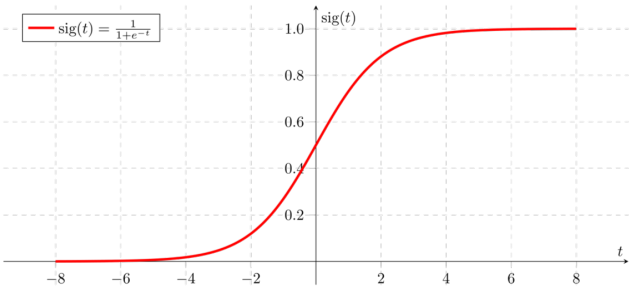


2.9.6 Logistic Regression

Logistic Regression is used when the dependent variable is binary. It is a go-to method for binary classification problems in statistics. First, it is quintessential to understand when to use linear regression and when to use logistic regression.

Linear vs Logistic regression –

Linear regression is used when the dependent variable is continuous and the nature of the regression line is linear. Logistic regression is used when the dependent variable is binary in nature.



Logistic Regression predicts the probability of occurrence of a binary event utilizing a sigmoid function.

2.9.7 Gaussian Naive Bayes

Naive Bayes is super effective, commonly-used Machine Learning classifier. Naive Bayes is in its own a family of algorithms including algorithms for both supervised and unsupervised learning. Naive Bayes classifiers are a collection of classification algorithms based on Bayes theorem. It is not a single algorithm but a family of

algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. In order to understand Naive Bayes, let us recall Bayes rule:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Naive Bayes (NB) is naive because it makes the assumption that attributes of a measurement are independent of each other. We can simply take one attribute as independent quantity and determine proportion of previous measurements that belong to that class having the same value for this attribute only. Naive Bayes is used primarily to predict the probability of different classes based on multiple attributes. It is mostly used in text classification while mining the data.

2.10 Model Performance Assessment

Once you have a model which our learning algorithm has built using the training set, how can you say how good the model is? You use the test set to assess the model. The test set contains the examples that the learning algorithm has never seen before, so if our model performs well on predicting the labels of the examples from the test set, we say that our model generalizes well or, simply, that it's good. To be more rigorous, Machine Learning specialists use various formal metrics and tools to assess the model performance. For regression, we compute the mean squared error (MSE) for the training, and, separately, for the test data. If the MSE of the model on the test data is substantially higher than the MSE obtained on the training data, this is a sign of over-fitting. Regularization or a better hyper-parameter tuning could solve the problem. For classification, things are a little bit more complicated. The most widely used metrics and tools to assess the classification model are:

- Confusion matrix
- Precision & Recall
- Accuracy
- F-measure
- Cost sensitive accuracy
- Area under the ROC curve (AUC)

2.10.1 Confusion matrix

Confusion matrix for binary classification

The confusion matrix is a table that summarizes how successful the classification model is at predicting examples belonging to various classes. One axis of the confusion matrix is the label that the model predicted, and the other axis is the actual label.In a binary classification problem, there are two classes.

Let's say, the model predicts two classes: "spam" and "not_spam":

	Actual Positive Class	Actual Negative Class
Predicted Positive Class	True positive (*tp*)	False negative (*fn*)
Predicted Negative Class	False positive (*fp*)	True negative (*tn*)

Figure 2.17. Confusion Matrix for Binary Classification. [36]

	spam (predicted)	not_spam (predicted)
spam (actual)	23 (TP)	1 (FN)
not_spam (actual)	12 (FP)	556 (TN)

Figure 2.18. Confusion Matrix for spam classification.

Confusion matrix for multi-class classification

The confusion matrix for multi-class classification has as many rows and columns as there are different classes. It can help you to determine mistake patterns. For example, a confusion matrix could reveal that a model trained to recognize different species of animals tends to mistakenly predict "cat" instead of "fish", or "hen" instead of "fish". In this case, you can decide to add more labeled examples of these species to help the learning algorithm to "see" the difference between them. Alternatively, you might add additional features the learning algorithm can use to build a model that would better distinguish between these species. Confusion matrix is used to calculate two other performance metrics: precision and recall. Refer to 2.19 for how we can represent confusion matrix for multiple(more than 2) classes

2.10.2 Precision & recall

"Precision is used to measure the positive patterns that are correctly predicted from the total predicted patterns in a positive class." [36]

$$Precision = \frac{t_p}{t_p + f_p} \tag{2.10}$$

Recall measures "the fraction of positive patterns that are correctly classified" [36]

$$Recall = \frac{t_p}{t_p + t_n} \tag{2.11}$$

For multiple classes:

Even if precision and recall are defined for the binary classification case, you can always use it to assess a multi-class classification model. To do that, first select a

		True/Actual		
		Cat (🐱)	Fish (🐟)	Hen (🐔)
Predicted	Cat (🐱)	4	6	3
	Fish (🐟)	1	2	0
	Hen (🐔)	1	2	6

Figure 2.19. Confusion Matrix for multi-class classification.

class for which you want to assess these metrics. Then you consider all examples of the selected class as positives and all examples of the remaining classes as negatives.

To understand the meaning and importance of precision and recall for the model assessment it is often useful to think about the prediction problem as the problem of research of documents in the database using a query. The precision is the proportion of relevant documents in the list of all returned documents. The recall is the ratio of the relevant documents returned by the search engine to the total number of the relevant documents that could have been returned. In the case of the spam detection problem, we want to have high precision (we want to avoid making mistakes by detecting that a legitimate message is spam) and we are ready to tolerate lower recall (we tolerate some spam messages in our inbox). Almost always, in practice, we have to choose between a high precision or a high recall. It's usually impossible to have both

2.10.3 Accuracy

The accuracy, "the ratio of correct predictions over the total number of instances evaluated" [36], is defined by

$$AccuracyScore = \frac{t_p + t_n}{t_p + t_n + f_p + f_n} \tag{2.12}$$

Accuracy is a useful metric when errors in predicting all classes are equally important. In case of the spam/not spam, this may not be the case. For example, you would tolerate false positives less than false negatives. A false positive in spam detection is the situation in which your friend sends you an email, but the model labels it as spam and doesn't show you. On the other hand, the false negative is less of a problem: if your model doesn't detect a small percentage of spam messages, it's not a big deal.

2.10.4 Cost sensitive accuracy

For dealing with the situation in which different classes have different importance, a useful metric is cost-sensitive accuracy. To compute a cost-sensitive accuracy, you first assign a cost (a positive number) to both types of mistakes: FP and FN. You then compute the counts TP, TN, FP, FN as usual and multiply the counts for FP and FN by the corresponding cost before calculating the accuracy

2.10.5 F-measure

The F-Measure, also called F1-Score, is the harmonic mean between Precision and Recall, calculated by

$$F1 - Score = \frac{2 * Recall * Precision}{Recall + Precision} \tag{2.13}$$

2.10.6 Area under the ROC curve (AUC)

The ROC curve (stands for "receiver operating characteristic", the term comes from radar engineering) is a commonly used method to assess the performance of classification models. ROC curves use a combination of the true positive rate (defined exactly as recall) and false positive rate (the proportion of negative examples predicted incorrectly) to build up a summary picture of the classification performance.

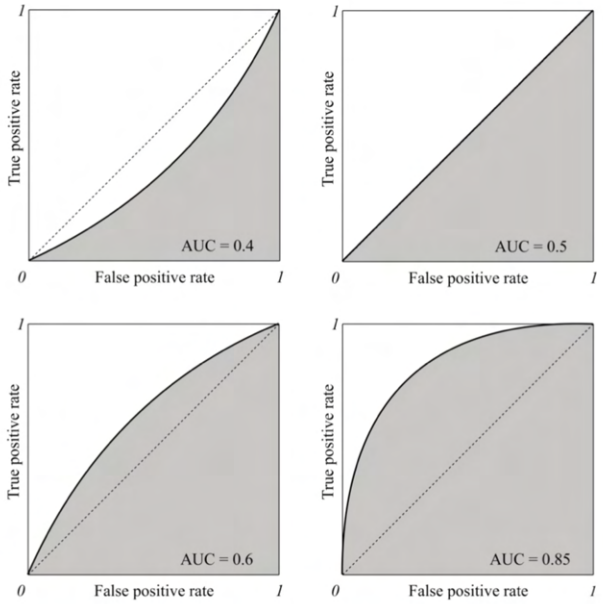


Figure 2.20. Area under the curve(Shown in gray) [15]

The true positive rate (TPR) and the false positive rate (FPR) are respectively defined as,

$$TPR = \frac{t_p}{t_p + f_n} \qquad FPR = \frac{f_p}{f_p + t_n} \tag{2.14}$$

ROC curves can only be used to assess classifiers that return some confidence score (or a probability) of prediction. For example, logistic regression, neural networks, and decision trees (and ensemble models based on decision trees) can be assessed using ROC curves. To draw a ROC curve, you first discretize the range of the confidence score. If this range for a model is [0, 1], then you can discretize it like this: [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]. Then, you use each discrete value as the prediction threshold and predict the labels of examples in your data set using the model and this threshold.

For example, if you want to compute TPR and FPR for the threshold equal to 0.7, you apply the model to each example, get the score, and, if the score if higher than or equal to 0.7, you predict the positive class; otherwise, you predict the negative class. Look at the illustration in Figure 2.20.

It's easy to see that if the threshold is 0, all our predictions will be positive, so both TPR and FPR will be 1 (the upper right corner). On the other hand, if the threshold is 1, then no positive prediction will be made, both TPR and FPR will be 0 which corresponds to the lower left corner. The higher the area under the ROC curve (AUC), the better the classifier. A classifier with an AUC higher than 0.5 is better than a random classifier. If AUC is lower than 0.5, then something is wrong with your model. A perfect classifier would have an AUC of 1.

Usually, if your model behaves well, you obtain a good classifier by selecting the value of the threshold that gives TPR close to 1 while keeping FPR near 0. ROC curves are popular because they are relatively simple to understand, they capture more than one aspect of the classification (by taking both false positives and negatives into account) and allow visually and with low effort comparing the performance of different models.

Note: In case of having a multi class classification problem, the metrics are calculated for each class against all of the other classes. The results are then combined to obtain the overall metric by using one of the average functions.

2.11 Hyper-parameter tuning

Hyper parameters aren't optimized by the learning algorithm itself. The data analyst has to "tune" hyper parameters by experimentally finding the best combination of values, one per hyper parameter. Hyper parameters can be tuned by following approaches,

- Cross validation
- Grid search
- Random search
- Bayesian techniques

2.11.1 Cross validation

When you don't have a decent validation set to tune your hyper parameters on, the common technique that can help you is called cross-validation. When you have few training examples, it could be prohibitive to have both validation and test set. You would prefer to use more data to train the model. In such a case, you only split your data into a training and a test set. Then you use cross-validation on the training set to simulate a validation set. Cross-validation works like follows. First, you fix the values of the hyper parameters you want to evaluate. Then you split your training set into several subsets of the same size. Each subset is called a fold. Typically, five-fold cross-validation is used in practice. With five-fold cross-validation, you randomly split your training data into five folds: F1 , F2 , . . . , F5 . Each Fk , k = 1, . . . , 5 contains 20% of your training data.

Then you train five models as follows,

- To train the first model, f1, you use all examples from folds F2, F3, F4, and F5 as the training set and the examples from F1 as the validation set.
- To train the second model, f2, you use the examples from folds F1, F3, F4, and F5 to train and the examples from F2 as the validation set.
- You continue building models iteratively like this and compute the value of the metric of interest on each validation set, from F1 to F5.
- Then you average the five values of the metric to get the final value.

You can use grid search with cross-validation to find the best values of hyper parameters for your model. Once you have found these values, you use the entire training set to build the model with these best values of hyper parameters you have found via cross-validation. Finally, you assess the model using the test set.

2.11.2 Grid Search

Grid search is the most simple hyper-parameter tuning technique.

Example: Let's say you train an SVM and you have two hyper-parameters to tune: the penalty parameter C (a positive real number) and the kernel (either "linear" or "rbf"). If it's the first time you are working with this particular data-set, you don't know what is the possible range of values for C. The most common trick is to use a logarithmic scale. For example, for C you can try the following values: [0.001, 0.01, 0.1, 1, 10, 100, 1000]. In this case you have 14 combinations of hyper-parameters to try: [(0.001, "linear"), (0.01, "linear"), (0.1, "linear"), (1, "linear"), (10, "linear"), (100, "linear"), (1000, "linear"), (0.001, "rbf"), (0.01, "rbf"), (0.1, "rbf"), (1, "rbf"), (10, "rbf"), (100, "rbf"), (1000, "rbf")].

You use the training set and train 14 models, one for each combination of hyper-parameters. Then you assess the performance of each model on the validation data using one of the metrics we discussed in the previous section (or some other metric that matters to you). Finally, you keep the model that performs the best according to the metric. Once the best pair of hyper-parameters is found, you can try to explore the values close to the best ones in some region around them. Sometimes, this can result in an even better model. Finally, you assess the selected model using the test set. As you could notice, trying all combinations of hyper-parameters, especially if there are more than a couple of them, could be time-consuming, especially for large data-sets. There are more efficient techniques, such as random search and Bayesian hyper-parameter optimization

2.11.3 Other methods

Random search differs from grid search in that you no longer provide a discrete set of values to explore for each hyperparameter; instead, you provide a statistical distribution for each hyperparameter from which values are randomly sampled and set the total number of combinations you want to try.

Bayesian techniques differ from random or grid search in that they use past evaluation results to choose the next values to evaluate. The idea is to limit the number of expensive optimizations of the objective function by choosing the next hyperparameter values based on those that have done well in the past.

There are also gradient-based techniques, evolutionary optimization techniques, and other algorithmic hyperparameter tuning techniques. Most modern Machine Learning libraries implement one or more such techniques. There are also hyperparameter tuning libraries that can help you to tune hyperparameters of virtually any learning algorithm, including ones you programmed yourself.

Figure 2.21 summarizes each step of the Machine Learning workflow,

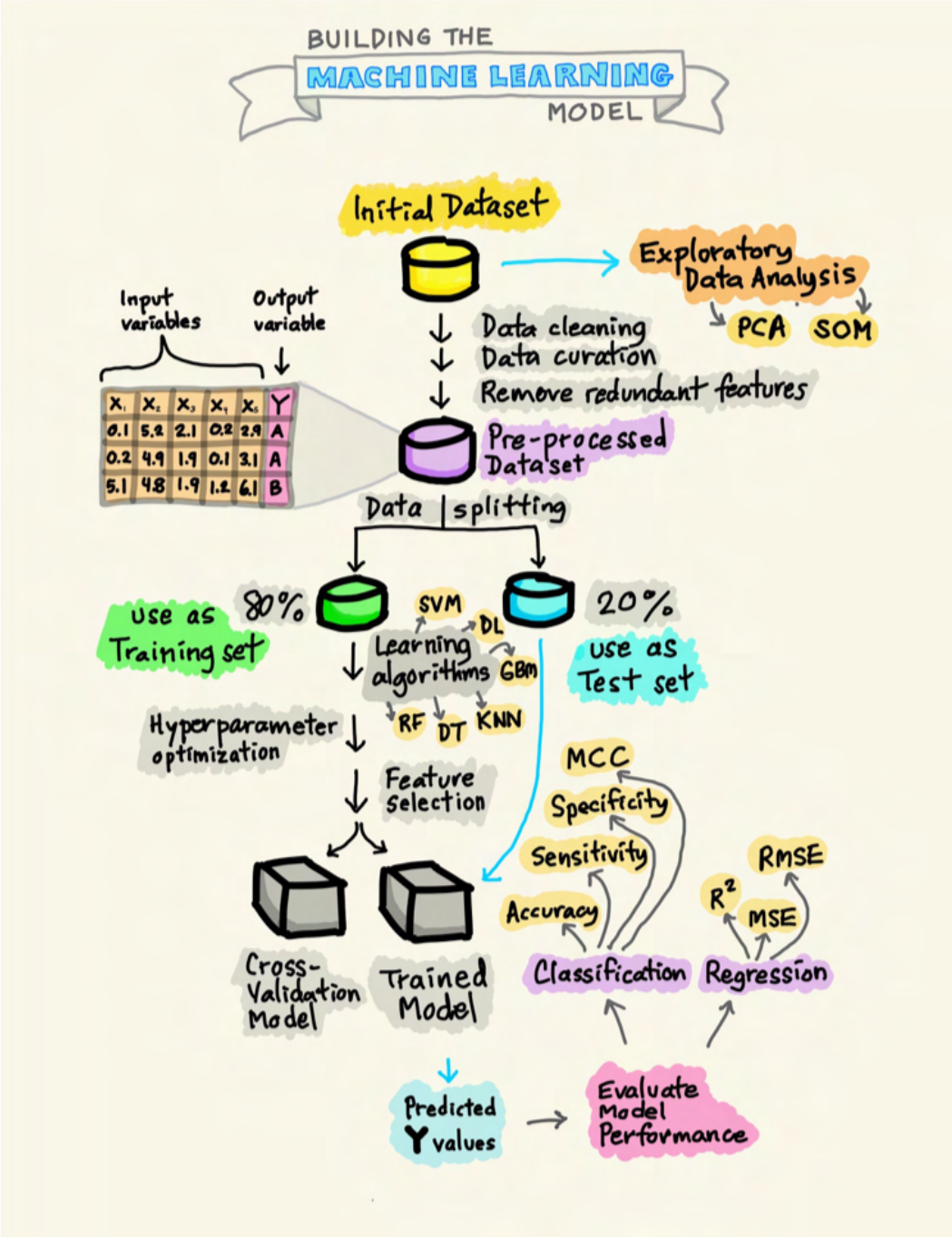


Figure 2.21. ML workflow diagram in a nutshell [67]

Chapter 3

Deep Learning 101

Before starting to talk about Deep Learning techniques for anomaly detection, lets have a look at what Deep Learning(DL) really is, and why it is so popular in the current era of data revolution.

3.1 Introduction to Deep Learning

3.1.1 What is Deep Learning?

Deep learning (also known as deep structured learning) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised.[9]

3.1.2 What is so special about Deep learning?

Even though Artificial intelligence has been in practice from 1950s, since the last decade there has been huge adaptability of Deep Learning based applications in variety of platforms and business cases. This can be attributed to the availability of cheap hardware and advent of GPU's for computation. Deep learning architectures [78] such as deep neural networks, deep belief networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision [46], machine vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance [50].

Fiigure 3.1 shows how Deep Learning is linked with ML and AI. "The subset of machine learning composed of algorithms and complex multi layer neural networks". Substantially, what neural networks can do, is approximate every type of functions. Deep neural networks work adding many layers to make the approximation more precise, but deeper the structure, bigger is the quantity of data needed. Hence the name deep learning. For example, for reinforcement learning tasks. A well-known system of Google DeepMind, "AlphaGo" [28], is a software based on Deep Neural Networks and Monte Carlo Tree Search capable to play "GO" game in an unsupervised way. AlphaGo is the first software able to beat a champion in the Go game. Trained in a semi self-learning way. The input, through a pre-processing module, to extract features, is passed to the neural networks that are trained in a supervised mode, based on how human plays. While in the second part the reinforcement learning is used to train the system to play versus itself over and over so that it gets better

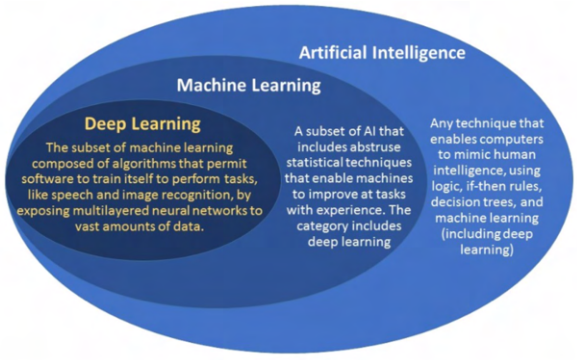


Figure 3.1. Deep Learning vs Machine Learning vs Artificial Intelligence

over time.
Another field of application similar to NLP is computer vision. Deep Learning algorithms were able to reach the state-of-art in terms of accuracy, For example in image recognition.

Primarily, the salient features of Deep Learning are as follows:

1. **Simplicity**: Deep Learning architectures are very simple and allows to extend the structure by adding layers to the network.

2. **Scalability**: Huge datasets are not a problem because for Deep Learning the bigger the data the better

3. **Transfer Learning**: Deep Learning techniques have the concept of transfer learning, a methods that allows to improve the learning on a second task starting from the knowledge learned on the first one. This is a very powerful capibility because it enables to tackle a host of usecases similar to the problem statement.

This was only possible due to the continuous growth of data available (Big Data), and more powerful hardware (GPUs).

3.2 Artificial Neural Networks

The first versions of Neural Network as shown in Figure 3.2 is called "shallow" for its characteristics of the input, output and one hidden layer.

The network is composed by those elements: an activation function, a matrix of weights and a bias. The bias represent a sort of random noise, the weight matrix represents the values of the connections between different layers, and the activation function defines the response given by the neuron according to the different stimuli received. Below is a formula related to a single neuron:

$$A_j = \sum_{i=1}^{N} w_{ij} X_i - b_i \tag{3.1}$$

returns the value which is to be passed as input to the activation function of the jth neuron to decide if is activated or not, as follows:

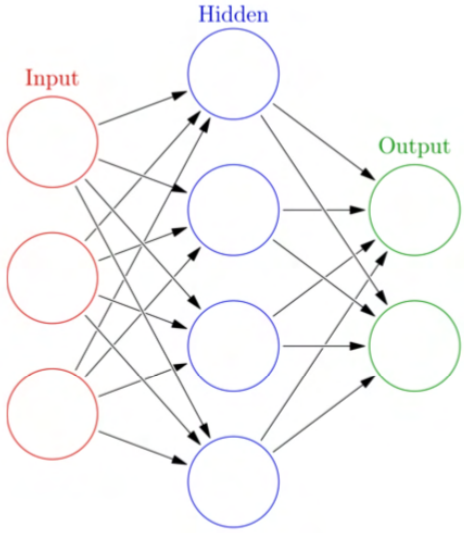


Figure 3.2. Shallow Artificial Neural Network

$$y_j = \varnothing(A_j) = \varnothing(\sum_{i=1}^{N} w_{ij} X_i - b_i) \tag{3.2}$$

where ø indicates the activation function. Usually in the output layer there are different activation functions based on the task. The activation functions will be explored in 3.3.

3.2.1 Multilayer Perceptron

A multilayer perceptron (MLP) [29] is one of the prominent Artificial neural network (ANN), considered as a variant of the original perceptron model proposed by Rosenblatt in 1950 [74]. It has at least one hidden layer between the input and output layer, where the neurons in the same layer are not connected and the edges are unidirectional, see Figure 3.3. The number of the neurons in the input layer is equal to the number of the input vector of the problem and the neurons number in the output layer equals to the number of classes of the problem. The hidden layers process the information of the input layer to output classes, also referred as computational engine of the MLP. The choice of the number of the layers, neurons in each layer and connections is called architecture problem. The goal is to optimize the architecture problem with sufficient parameters [70]. The learning of an MLP is realized by adjusting the weights in order to obtain the minimum prediction error. For this the widely used algorithm is back propagation. In a feed forward network such as MLP there are two primary flows: a constant forward and backward flows. As mentioned before, in the forward flow the signal moves from the input layer through the hidden layers to the output layer and produces an output. The output of the neural network is compared with the ground truth, the difference between them is considered as the error. In the backwards flow, the weights of the neuron connections are changed depending on their influence on the error which can be done with any gradient-based algorithm such as Stochastic Gradient Descent (SGD). This forwards-backwards loop is repeated until the minimum error is reached also called convergence. [70]

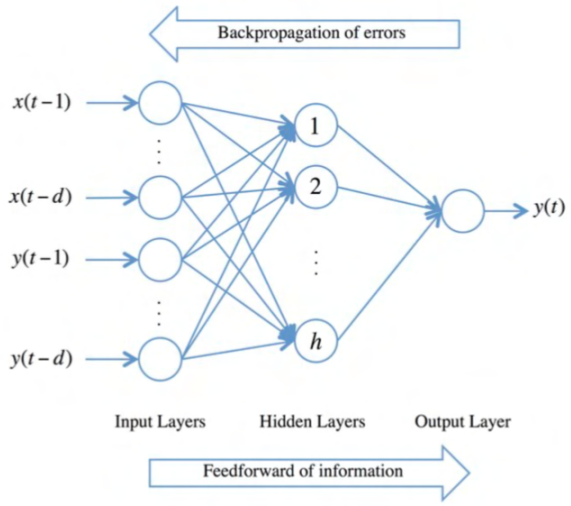


Figure 3.3. Feedforward Backpropagation Neural Network architecture. [18]

3.3 Activation Functions

Activation functions [29] decide whether a neuron is "activated" or not by calculating the weighted sum of its inputs. Furthermore, it computes an output signal from the input signals of a neuron. The output signal is then used as an input signal for a neuron in the next layer. The activation functions can be of both linear and non-linear types. However, the most commonly used activation functions are non-linear activation functions. [43] This allows the network to learn more complex problems a linear function cannot learn. Following are some of the non-linear activation functions.

3.3.1 Sigmoid

The sigmoid function, sometimes also referred as logistic function, is defined by following equation:

$$f(z_i) = \frac{1}{1 + e^{-z_i}} \tag{3.3}$$

It is one of the most popular activation function because the range of the s-shaped curve is between 0 and 1.[85] Therefore, it is very useful in cases where the output is desired to be probabilities. It is also often used in binary classification problem by setting a threshold, usually to 0.5, and assigning an observation to class 1 if the prediction is bigger than the threshold and otherwise to class 0. However the downside is that the Sigmoid activation function suffers major drawbacks which include sharp damp gradients during back propagation from deeper hidden layers to the input layers, gradient saturation, slow convergence and non-zero centred output thereby causing the gradient updates to propagate in different directions."[61]

3.3.2 Softmax

The softmax function, a type of the sigmoid function, is defined by:

$$f(z_i) = \frac{e^{-z_i}}{\sum_{j=1}^{k} e^{-z_j}} \tag{3.4}$$

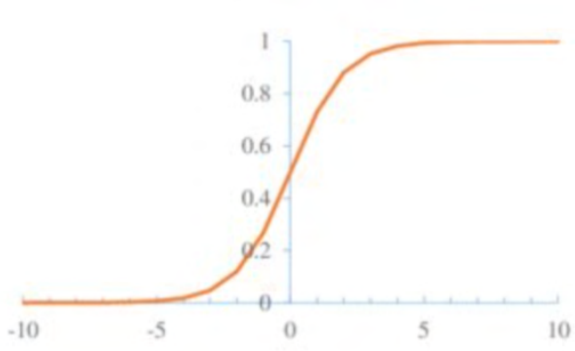


Figure 3.4. Example of a sigmoid function. [85]

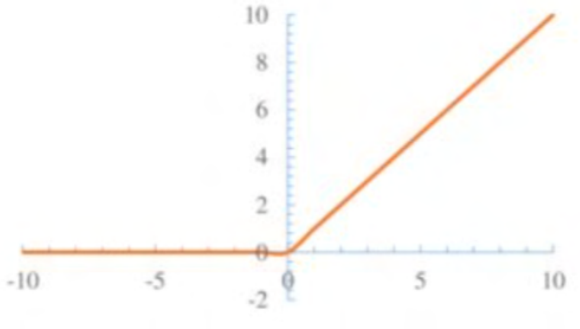


Figure 3.5. Example of a ReLU function. [85]

where k represents the number of events. As the sigmoid function, the softmax function ranges between 0 and 1. The difference is that the sum of the probabilities of an event is equal to 1. Therefore, it is ideally for multi-class classification problems. In case of using softmax as activation function in the output layer, it will return the probability of each class where the highest probability represents the target class. The main difference between sigmoid and softmax activation functions is that sigmoid is used for binary classification and softmax for multi-class problems. [61]

3.3.3 ReLu

ReLU, which stands for Rectified Linear Unit, is an activation function widely used in the hidden layer. It is defined by

$$f(z_i) = max(0, z_i). \tag{3.5}$$

The range of the ReLU is between 0 and infinite. [85] It is 0 for input values smaller than 0 and otherwise, it is equal to the input value. The advantage of ReLU, in comparison to the other activation functions, is the low computational complexity. [85] Another property of the ReLU is that it introduces sparsity in the hidden units as it squishes the values between zero to maximum. [61] However, the main problem of ReLU is that it overfits easily even after adopting dropout techniques to reduce the effect of overfitting. [31]

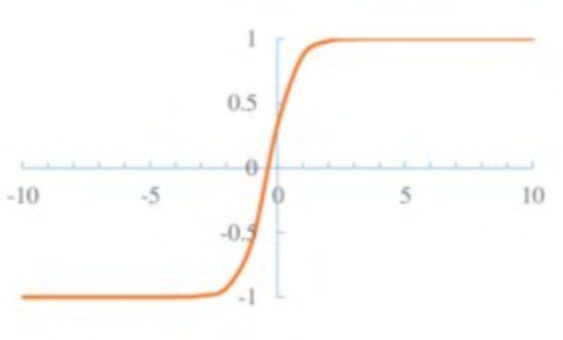


Figure 3.6. Example of a TanH function. [85]

3.3.4 TanH

TanH stands for Tangent Hyperbolic function and is a scaled sigmoid function. It is mathematically defined by

$$f(z_i) = tanh(z_i) = \frac{2}{1 + e^{-2z_i}} - 1 \tag{3.6}$$

where $tanh(z_i)$ can also be defined as

$$tanh(z_i) = 2 * sigmoid(2z_i) - 1 \tag{3.7}$$

The range of a TanH activation function is between -1 and 1.[85] For this reason it is usually used in the hidden layers. The advantage is the centering the mean of the data close to 0, which makes the learning easier for the next layer. However, the TanH function suffers from the problem of not being able to solve the vanishing gradient problem like sigmoid function. [61] Furthermore, in some cases TanH can produce dead neurons during computation, which "is a condition where the activation weight, rarely used as a result of zero gradient." [61] This led researchers develop ReLU as a solution.

3.4 Optimization Algorithms

Optimization algorithms [29] are tools which minimize the value of the loss function, also called error function. They are evaluated on the training set by updating the model parameters. [69] For instance, weights belong to these learnable model parameters. Changing the weights too much or too less can create difficulties in minimizing the loss function. This is regularized by the parameter learning rate , usually = 0.001, that is multiplied with the gradients to scale them. [2]

3.4.1 Stochastic Gradient Descent

In Deep Learning, the objective function is usually is defined the average of the loss functions for each example in the training dataset." [69] The objective function is defined by

$$f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x) \tag{3.8}$$

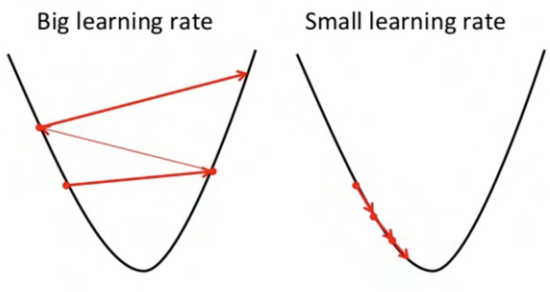


Figure 3.7. Effects of choice of learning rate. [2]

with $f_i(x)$ as loss function of training data instance i with n examples and x as parameter vector. The gradient of the objective function is calculated as

$$\triangle f(x) = \frac{1}{n}\sum_{i=1}^{n} \triangle f_i(x) \tag{3.9}$$

If gradient descent is used, the computation cost of each iteration is very high for a large data set. Stochastic gradient descent (SGD) reduces the cost at each iteration by uniformly sampling an index $i \in 1, ..., n$ for data instances at random and compute (x) to update x by

$$x \leftarrow x - \eta_i(x) \tag{3.10}$$

where η represents the learning rate. In comparison to the gradient descent, the computation cost of each iteration in SGD is lower. Furthermore, the stochastic gradient is on average a good estimate of the gradient:

$$E_i \triangle f_i(x) = \frac{1}{n}\sum_{i=1}^{n} \triangle f_i(x) = \triangle f(x) \tag{3.11}$$

In short, for convex problems, SGD will converge to the optimal solution. Problems can occur by choosing the learning rate, for instance for to small or big learning rates. For non-convex problems the convergence is not guaranteed. [69]

3.4.2 RMS prop

RMSProp stands for Root Mean Square Propagation. RMSProp is an extension of Adagrad, which is an optimization method for adapting the learning rate. Adagrad makes the larger update for infrequent and smaller update for frequent parameters. [84] More details to Adagrad can be read in chapter 11.7 in the book "Dive into Deep Learning" [69]. The main problem of Adagrad is that the learning rate decreases at a predefined schedule. Mathematically, Adagrad accumulates the squares of the gradient g_t into a state vector $s_t = s_{t1} + g_t^2$ which results in s_t growing without bound. RMSProp solves the problem with following equations:

$$S_t \leftarrow \gamma S_{t-1} + (1-\gamma) g_t^2 \tag{3.12}$$

$$x_t \leftarrow x_{t-1} \frac{\eta}{\sqrt{S_t + \in}} g_t \tag{3.13}$$

with parameter $\gamma > 0$, constant $\in > 0$ and learning rate η. The constant $\in$ is typically set to 10^{-6} to avoid division by zero or too large step sizes. [69]

3.4.3 Adaptive Moment Estimation

Adaptive Moment Estimation (Adam) is another optimization method adapting the learning rate. It uses exponential weighted moving averages to estimate the momentum and the momentum of the gradient with state variables

$$V_t \leftarrow \beta_1 V_{t-1} + (1-\beta_1) g_t \tag{3.14}$$

$$S_t \leftarrow \beta_2 S_{t-1} + (1-\beta_2) g_t^2 \tag{3.15}$$

where β_1 and β_2 are non negative weighting parameters with usually $\beta_1 = 0.9$ and $\beta_2 = 0.999$. This results in the variance moving slower than the momentum term and high bias especially during the initial time steps. To overcome the problem, $\sum_{i=0} t\beta^i = \frac{1-\beta_t}{1-\beta}$ is used to re-normalize the terms. The normalized state variables are given by

$$\hat{V}_t = \frac{V_t}{1-\beta_1^t} \tag{3.16}$$

$$\hat{S}_t = \frac{S_t}{1-\beta_2^t} \tag{3.17}$$

Updating the parameter as seen in RMSProp following equation is obtained:

$$g_t^l = \frac{\eta \hat{V}_t}{\sqrt{\hat{S}_t} + \in} \tag{3.18}$$

In comparison to RMSProp, adam uses the momentum $\hat{V}_t$ for updating instead of the gradient itself. The update is defined by

$$x_t \leftarrow x_{t-1} - g_t^l. \text{ [69]} \tag{3.19}$$

3.5 Loss functions

Neural networks are trained using an optimization process that requires a loss function to calculate the model error. Neural networks are trained using stochastic gradient descent and require that you choose a loss function when designing and configuring your model.

3.5.1 What is a Loss Function and Loss?

In the context of an optimization algorithm, the function used to evaluate a candidate solution(i.e. a set of weights) is referred to as the objective function [29]. We may seek to maximize or minimize the objective function, meaning that we are searching for a candidate solution that has the highest or lowest score respectively. Typically, with neural networks, we seek to minimize the error. As such, the objective function is often referred to as a cost function or a loss function and the value calculated by the loss function is referred to as simply loss. The cost or loss function [72] has an important job in that it must faithfully distill all aspects of the model down into a single number in such a way that improvements in that number are a sign of a better model. In calculating the error of the model during the optimization process, a loss function must be chosen. This can be a challenging problem as the function must capture the properties of the problem and be motivated by concerns that are important to the project and stakeholders.

3.5.2 Types of Loss functions

Now that we are familiar with the loss function and loss, we need to know what functions to use.

Maximum Likelihood

There are many functions that could be used to estimate the error of a set of weights in a neural network. We prefer a function where the space of candidate solutions maps onto a smooth(but high-dimensional) landscape that the optimization algorithm can reasonably navigate via iterative updates to the model weights. Maximum likelihood estimation(MLE) is a framework for inference for finding the best statistical estimates of parameters from historical training data which is exactly what we are trying to do with the neural network. [12]. We have a training dataset with one or more input variables and we require a model to estimate model weight parameters that best map examples of the inputs to the output or target variable. Given input, the model is trying to make predictions that match the data distribution of the target variable. Under maximum likelihood, a loss function estimates how closely the distribution of predictions made by a model matches the distribution of target variables in the training data. [29]. A benefit of using maximum likelihood as a framework for estimating the model parameters(weights) for neural networks and in machine learning in general is that as the number of examples in the training dataset is increased, the estimate of the model parameters improves.This is called the property of consistency.

Maximum Likelihood and Cross-Entropy

Under the maximum likelihood framework, the error between two probability distributions is measured using cross-entropy. When modeling a classification problem where we are interested in mapping input variables to a class label, we can model the problem as predicting the probability of an example belonging to each class. In a binary classification problem, there would be two classes, so we may predict the probability of the example belonging to the first class. In the case of multiple-class classification, we can predict a probability for the example belonging to each of the classes. In the training dataset, the probability of an example belonging to a given class would be 1 or 0, as each sample in the training dataset is a known example from the domain. We know the answer. Therefore, under maximum likelihood estimation, we would seek a set of model weights that minimize the difference between the model's predicted probability distribution given the dataset and the distribution of probabilities in the training dataset. This is called the cross-entropy [29].

Technically, cross-entropy [72] comes from the field of information theory and has the unit of bits. It is used to estimate the difference between an estimated and a predicted probability distribution. In the case of regression problems where a quantity is predicted, it is common to use the mean squared error (MSE) loss function instead. Nevertheless, under the framework of maximum likelihood estimation and assuming a Gaussian distribution for the target variable, mean squared error can be considered the cross-entropy between the distribution of the model predictions and the distribution of the target variable. [29]

Therefore, when using the framework of maximum likelihood estimation, we will implement a cross-entropy loss function, which often in practice means a cross-entropy loss function for classification problems and a mean squared error loss

function for regression problems. Almost universally, deep learning neural networks are trained under the framework of maximum likelihood using cross-entropy as the loss function. [29]. The maximum likelihood approach was adopted almost universally not just because of the theoretical framework, but primarily because of the results it produces. Specifically, neural networks for classification that use a sigmoid or softmax activation function in the output layer learn faster and more robustly using a cross-entropy loss function.

3.5.3 What Loss Function to Use?

The choice of loss function is directly related to the activation function used in the output layer of your neural network. These two design elements are connected. Think of the configuration of the output layer as a choice about the framing of your prediction problem, and the choice of the loss function as the way to calculate the error for a given framing of your problem. [29]

Regression Problem

A problem where you predict a real-value quantity.

- **Output Layer Configuration**: One node with a linear activation unit.
- **Loss Function**: Mean Squared Error (MSE).

Binary Classification Problem

A problem where you classify an example as belonging to one of two classes. The problem is framed as predicting the likelihood of an example belonging to class one, e.g. the class that you assign the integer value 1, whereas the other class is assigned the value 0.

- **Output Layer Configuration**: One node with a sigmoid activation unit.
- **Loss Function**: Cross-Entropy, also referred to as Logarithmic loss.

Multi-class Classification Problem

A problem where you classify an example as belonging to one of more than two classes. The problem is framed as predicting the likelihood of an example belonging to each class.

- **Output Layer Configuration**: One node for each class using the softmax activation function.
- **Loss Function**: Cross-Entropy, also referred to as Logarithmic loss.

3.6 Convolutional Neural Network

Convolutional Neural Networks (CNNs), or also called convnet, are ANNs originally invented for computer vision. [45] However in recent years, CNNs showed also good performance on text classification tasks. [7] [41] [83] [88] A CNN uses layers with convolution filters that are applied to model predictions, error analyzes or model improvements. [73] CNNs consist of sets of convolutional and pooling layers as shown in Figure 3.8. The convolutional filters and pooling layers are

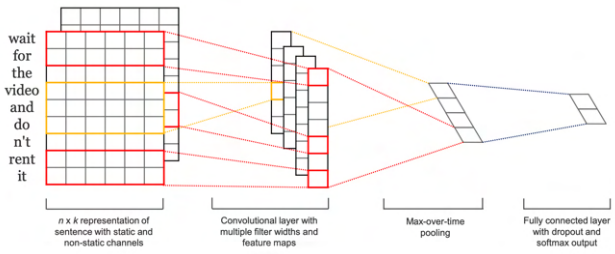


Figure 3.8. A CNN architecture with one layer of convolution and two channels. [45]

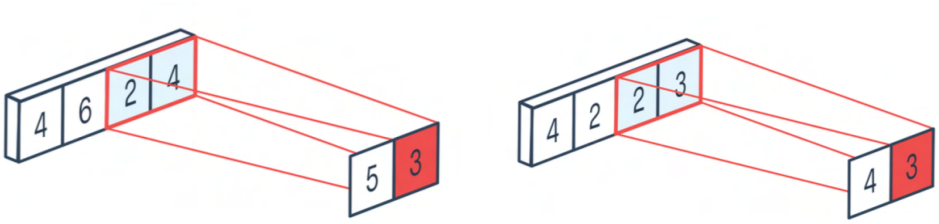
usually interleaved. Each layer applies different filters and combines their results. A convolutional layer followed by a pooling layer create a feature map. Pooling layers are used to reduce the output dimensionality. In CNNs, convolutional and pooling layers are not required to be fully connected like in MLPs. However, fully connected layers also exist in CNNs and are used at the end of the network to perform the classification.

3.6.1 Convolutional Layer

In common classification architectures, each observation or document is represented as a matrix where each row corresponds to a word with x_i as a k-dimensional word vector of the ith word in a document. The convolutional layer applies a 1D convolutional filter with a word window h on the matrix to produce a new feature c_j. The width of the filter is usually equal to the width of the matrix and h is typically between 2 and 5. This filter is applied to each possible window of words in the document to create a feature map c. [45] The shift size of the filter is controlled by a hyperparameter called stride size. At the edges of the matrix, all elements falling outside of the matrix can be padded with zeros, zero-padding. By using zero-padding the number of the outputs of a convolution is equal or bigger than the input, therefore it is also called wide convolution. Narrow convolution, on the other hand, does not use zero-padding and creates less output features. [16] A convolution operation can consist of multiple filters. Then, each convolutional filter creates a channel and each channel represents a different "view" of the input data, for instance different word windows. [16] A convolutional layer with m filters produces an m-dimensional vector for each word window. [39]

3.6.2 Pooling Layer

The vectors produced in a convolutional layer are combined in the pooling layer. It reduces the output dimensionality by keeping the important information. Convolutional filters detect specific features and the positions of them in a document. Pooling operations combine the information of a feature map c by moving a pooling window on it and doing a computation on each pooling window. This results in a smaller output with loosing the exact location of the features in a document by just considering whether a feature appeared in a document or not. Two common pooling methods are average and max pooling. Average pooling calculates the average of the features in a window and max pooling takes the highest value of the features in a window as shown in Figure 3.9 . [16]



(a) *1D Average Pooling* **(b)** *1D Max Pooling*

Figure 3.9. Pooling Operations

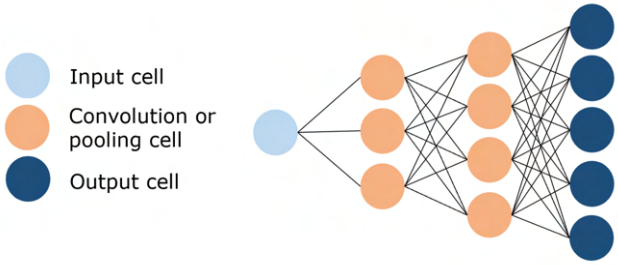


Figure 3.10. Deconvolutional Neural Network (DNN) [86]

3.6.3 Fully Connected Layer

The objective of a fully connected layer, also dense layer, is to make a classification using the results of the convolutional and pooling layers. The output is flattened to a single vector with probabilities for each class. This operation is sometimes replaced by global pooling layers. The global pooling layer is a pooling layer which samples the entire feature map of each channel down to one single value. Often, multiple fully connected layers are combined to make the final classification.

3.6.4 Deconvolutional Neural Network (DNN)

Deconvolutional Neural Networks [86], as its name suggests, performs the opposite of a convolutional neural network as shown in Figure 3.10. Instead of performing convolutions to reduce the dimensionality of an image, a DNN utilizes deconvolutions to create an image, usually from noise. This is an inherently difficult task; consider a CNN's task to write a three-sentence summary of the complete book of Orwell's 1984 while a DNN's task is to write the complete book from a three-sentence structure.

3.6.5 Generative Adversarial Network (GAN)

A Generative Adversarial Network (GAN) [86] is a specialized type of network designed specifically to generate images, and is composed of two networks, a discriminator and a generator as shown in Figure 3.11. The discriminator's task is to discriminate between whether an image is pulled from the dataset or if it has been generated by the generator, and the generator's task is to generate images convincing enough such that the discriminator cannot distinguish whether it is real or not. Over time, with careful regulation, these two adversaries compete with each other, each's drive to succeed improving the other. The end result is a well-trained generator that

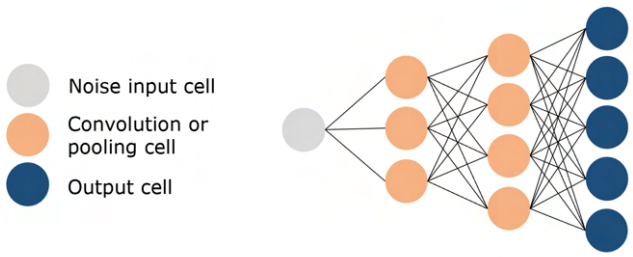


Figure 3.11. Generative Adversarial Network (GAN) [86]

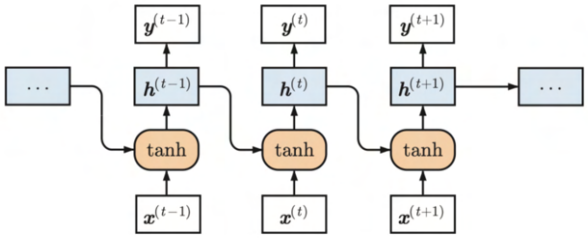


Figure 3.12. Example of a recurrent neural network. [59]

can spit out a realistic-looking image. The discriminator is a convolutional neural network whose goal is to maximize its accuracy in identifying real/fake images, whereas the generator is a deconvolutional neural network whose goal is to minimize the discriminator's performance.

3.7 Recurrent Neural Network

A recurrent neural networks (RNN) is another type of ANN. As traditional neural networks, RNNs consist of input, hidden and output layers. However, the functionality of the hidden layers is very different. Whereas in traditional neural networks all the inputs and outputs are independent, in RNNs each output of a step is fed as input for the next step. For instance, Figure 3.12 illustrates a vanilla RNN. As shown on the image, it takes sequential inputs $x^{(0)}$,...,$x^{(T)}$. The main feature in RNNs is the hidden state $h^{(t)}$. The hidden state is a time-variant, which is maintained with each step t. At step t, the input is equal to $x^{(t)}$ and the hidden state $h^{(t1)}$ is updated to $h^{(t)}$ by

$$h^{(t)} = f(Wh^{(t-1)} + Vx^{(t)} \tag{3.20}$$

with W and V as weight matrices and f as a non-linear activation function. [59]

Depending on the task, at each update, the hidden state $h^{(t)}$ can be used as output or processed further to perform a classification after processing the whole sequence at step T . The most typical input-output schemes are illustrated in Figure 3.13. The input-output scheme is chosen depending on the given task. In Figure 3.13 (a), the sequence-to-one scheme is shown. Here, the RNN takes the whole sequence as input and outputs only in the end of the sequence. This scheme is often used in sentiment analysis or document classification. Figure 3.13 (b) shows the sequence-to-sequence scheme, which is widely used in machine translation. The

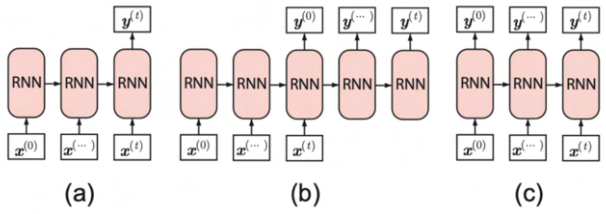


Figure 3.13. Input-output schemes of RNNs [59]

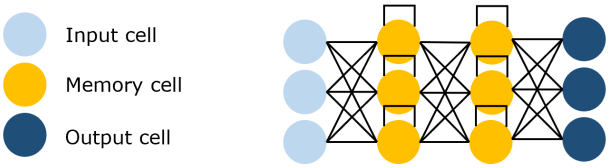


Figure 3.14. LSTM [86]

synced sequence-to-sequence scheme is illustrated in Figure 3.13 (c). This scheme is used for tasks where an output is desired after each input sequence, for instance language modelling or video classification. [59]

3.7.1 Long Short Term Memory Network (LSTM)

RNNs are problematic in that the range of contextual information is, in practice, very limited. The influence (back propagated error) of a given input on an input on the hidden layer (and hence on the network's output) either blows up exponentially or decays into nothing as it is cycled around the network's connections. The solution to this vanishing gradient problem is a Long Short-Term Memory Network, or an LSTM [86]. This RNN architecture is specifically designed to address the vanishing gradient problem, fitting the structure with memory blocks. These blocks can be thought of as memory chips in a computer — each one contains several recurrently connected memory cells and three gates (input, output, and forget, equivalents of write, read, and reset) as shown in Figure 3.14. The network can only interact with cells through each gate, and hence the gates learn to open and close intelligently to prevent exploding or vanishing gradients but also propagate useful information through "constant error carousels", as well as discarding irrelevant memory content.

Where standard RNNs fail to learn the presence of time lags larger than five to ten time steps between input events and target signals, LSTM's are not affected and can learn to connect time lags even 1000 time steps by enforcing a useful constant error flow.

LSTM is an effective solution for combating vanishing gradients by using memory cells [35]. A memory cell as shown in Figure 3.15 is composed of four units: an input gate, an output gate, a forget gate and a self-recurrent neuron. The gates control the interactions between neighboring memory cells and the memory cell itself. Whether the input signal can alter the state of the memory cell is controlled by the input gate. On the other hand, the output gate can control the state of the memory cell on whether it can alter the state of other memory cell. In addition, the forget gate can choose to remember or forget its previous state as shown in Figure 3.15

where,

$$i_t = \sigma(x_t W_{xi} + h_{t-1} W_{hi} + c_{t-1} W_{ci} + b_i) \tag{3.21}$$

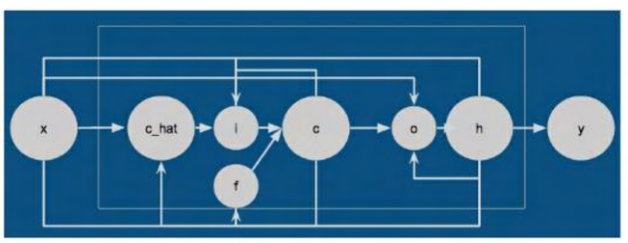


Figure 3.15. Internal working of a LSTM network [35]

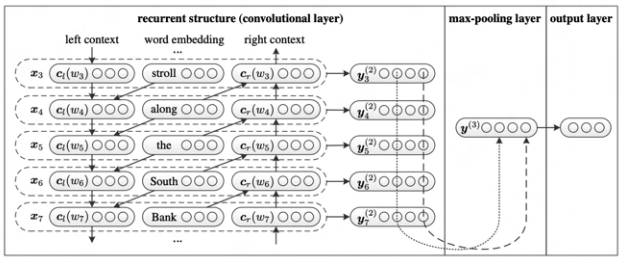


Figure 3.16. Structure of the RCNN proposed by Lai et al. [48]

$$f_t = \sigma(x_t W_{xf} + h_{t-1} W_{hf} + c_{t-1} W_{cf} + b_f) \tag{3.22}$$

$$c_t = f_t c_{t-1} + i_t tanh(x_t W_{xc} + h_{t-1} W_{hc} + b_c) \tag{3.23}$$

$$o_t = \sigma(x_t W_{xo} + h_{t-1} W_{ho} + c_t W_{co} + b_o) \tag{3.24}$$

$$h_t = o_t tanh(c_t) \tag{3.25}$$

3.8 Recurrent Convolutional Neural Network

Recurrent Convolutional Neural Network (RCNNs) is a neural network using advantages of both RNNs and CNNs. It was developed to overcome the limitations of the RNN, where word appearing later in the document are more dominant than earlier words, and the CNN, the challenge of choosing the window size: "small window sizes may result in the loss of some critical information, whereas large windows result in an enormous parameter space." [48] The RCNN, proposed by Lai et al. [48], consist of a bidirectional recurrent structure to capture the contextual information followed by a max-pooling layer as shown in Figure 3.16.

As illustrated in Figure 3.16, the recurrent structure consist of a left context $c_l(w_i)$ and right context $c_r(w_i)$, defined by

$$c_l(w_i) = f(W^{(l)} c_l(w_{i-1}) + W^{(sl)} e(w_{i-1})) \tag{3.26}$$

$$c_r(w_i) = f(W^{(r)} c_r(w_{i+1}) + W^{(sr)} e(w_{i+1})) \tag{3.27}$$

where f is a non linear activation function and e the word embedding vector for word. "The context vector captures the semantics of all left- and right-side contexts." [48]

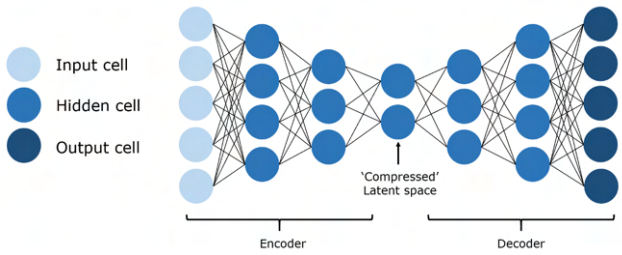


Figure 3.17. Auto Encoder [86]

Then, the left context $c_l(w_i)$, right context $c_r(w_i)$ and word embedding $e(w_i)$ are combined to define a representation for each word w_i by following equation:

$$x_i = [c_l(w_i); e(w_i); c_r(w_i)]. \tag{3.28}$$

Afterwards, the latent semantic vector $y^{(2)}$ is created for each x_i by

$$y_i^{(2)} = tanh(W^{(2)}x_i + b^{(2)}). \tag{3.29}$$

The latent semantic vector $y^{(2)}$ is fed to the pooling layer, to analyze the most important features for representing the text. Finally, the output of the pooling layer is passed the output layer to make the final classification, for instance using the softmax function in case of a multi class classification problem. [48]

3.9 Auto-Encoders

3.9.1 Auto Encoder (AE)

The fundamental idea of an autoencoder [86] is to take in original, high-dimensionality data, 'compress it' into a highly informational and low-dimensional data, and then to project the compressed form into a new space. There are many applications of autoencoders, including dimensionality reduction, image compression, denoising data, feature extraction, image generation, and recommendation systems. It can be applied as both an unsupervised or a supervised method, can be very insightful as to the nature of the data. As show in 3.17, hidden cells can be replaced with convolutional layers to accommodate processing images.

3.9.2 Variational Auto Encoder (VAE)

Whereas an autoencoder learns a compressed representation of an input, which could be images or text sequences, for example, by compressing the input and then decompressing it back to match the original input, a variational autoencoder (VAE) [86] learns the parameters of a probability distribution representing the data. Instead of just learning a function representing the data, it gains a more detailed and nuanced view of the data, sampling from the distribution and generating new input data samples. In this sense, it is more of a purely 'generative' model, like a GAN.

As shown in Figure 3.18, VAE uses a probabilistic hidden cell, which applies a radial basis function to the difference between the test case and the cell's mean.

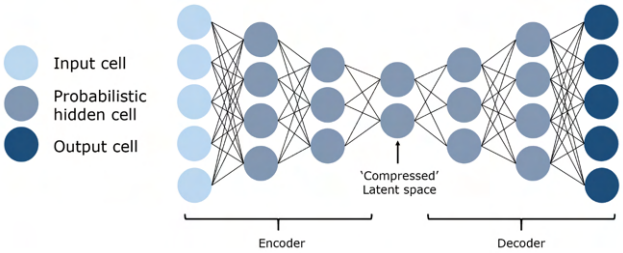


Figure 3.18. Variational Auto Encoder (VAE) [86]

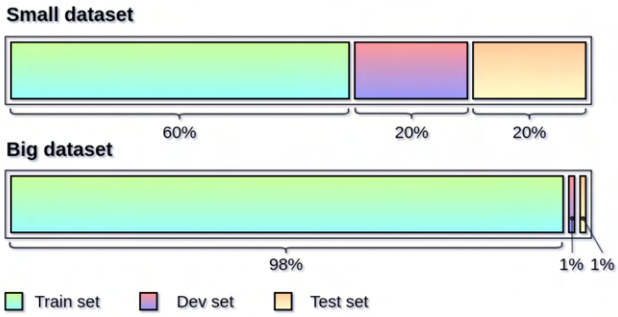


Figure 3.19. Recommended method of dividing the data set. [81]

3.10 Regularisation Methods

In ANNs, the neural network corrects with each iteration (epoch) the weights to reach a better accuracy and fits better to the training data. Driving the error on the training data to a very small value is called overfitting. In this case, the neural network has difficulties to generalize and make predictions for unseen data, which leads to poor performance. Furthermore, it is not an easy challenge to detect overfitting. As mentioned before, it can be only detected when a trained model is tested on an unseen data set. This is the reason of splitting an entire data set in Deep Learning in 3 partitions: training, validation and test set as shown in Figure 3.19. The model is only fitted to the training set. For each epoch, the fitted model does predictions on the observations in the validation set and measures the performance. In this case the model can be evaluated during the tuning of the parameters. The final model is evaluated by using the test set.

3.10.1 Early Stopping

A common challenge in neural networks is the choice of the number of training epochs. As mentioned before, too many epochs lead to overfitting. On the other side, too less epochs lead to underfitting, where a neural network cannot learn the relationships between the features well enough. Early stopping is a regularization method which allows to set a large number of training epochs and interrupt it once the performance does not improve anymore. Figure 3.20 shows a comparison of the training and validation error in the learning phase of an ANN. As seen on the graph, in the first epochs both errors decrease. Stopping the learning here would lead to underfitting. Then, after a specific number of epochs, the validation error starts to increase, whereas the training error continues to decrease. At this point, overfitting starts. Early stopping is used to interrupt the neural network at the minimum of

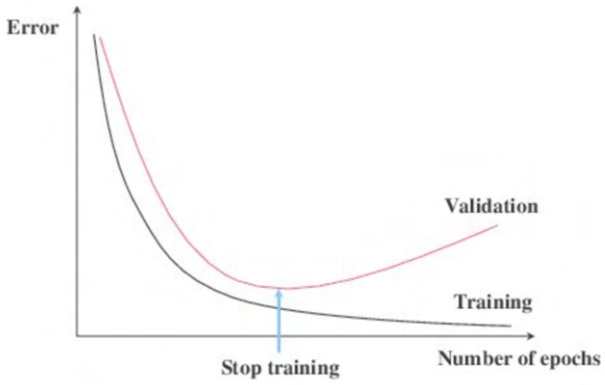


Figure 3.20. Early stopping method. [79]

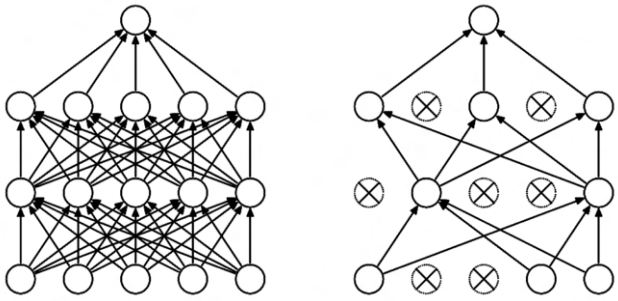


Figure 3.21. Left: ANN without dropout. Right: ANN with dropout. [6]

the validation error to avoid overfitting.

3.10.2 Dropout

Dropout is another regularization method for preventing overfitting. It temporally disables in the training phase randomly some neurons controlled by a hyperparameter, as in Figure 3.21 for instance. Therefore, the neural network is forced to learn different representations of the data, which are independent of each other. In this way, the neural network cannot fit too well on the training data and reduces the probability of overfitting. [6]

3.10.3 Data Augmentation

This method is very used with small training dataset. The aim is the same: make the model more robust and invariant on generalization of data. Taking in example images, the deal is to augment the dataset, with some transformation of the image to make a more accurate model training. The images are subjected to transformations such as:

- Size reorganization
- Sample standardization
- Horizontal and vertical vibration
- Shift

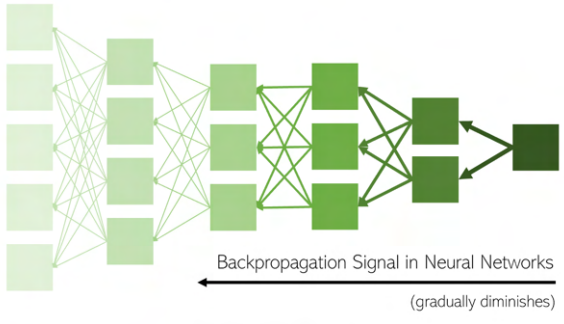


Figure 3.22. Vanishing Gradient

- Zoom
- ZCA whitening
- Random rotation

3.10.4 Batch Normalization

Batch normalization [29] is a technique for training very deep neural networks that standardizes the inputs to a layer for each minibatch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

Vanishing and Exploding gradient problems

The vanishing gradients problem is an example of unstable behavior that is often encountered when training a deep neural network, and occurs when a neural network is so long and complex that the neural network is unable to backpropagate useful gradient information to weights in the beginning of the network. This usually happens because activation functions gradually diminish the signal as it backpropagates throughout the network. As a result, the layers near the input of the model remain relatively unchanged and provide little value, restricting the model from accessing its true predictive power. Hence, the optimizer can only optimize the model so far with the weights it can access, and will, therefore, remain stagnant with no improvement even as the front structures are not being utilized as shown in Figure 3.22. The explosive gradients problem, on the other hand, occurs when large error gradients sequentially accumulate and result in large, unstable, pendulum-like updates on the neural network weights during training. Both the vanishing gradients and explosive gradients provide two huge problems that often plague neural networks. To address these gradient issues, primary methods used to address them include carefully setting the learning rate (slowly decaying the learning rate as the model approaches an optima), designing a better CNN architecture with different activation functions, more careful initialization of weights, and tuning the data distribution. The most effective solution is batch normalization.

What is Batch Normalization?

Deep neural networks are challenging to train, not least because the input from prior layers can change after weight updates. Batch normalization is a technique

to standardize the inputs to a network, applied to either the activation's of a prior layer or inputs directly. Batch normalization [38] accelerates training, in some cases by halving the number of epochs(or better), and provides some regularization effect, reducing generalization error.

How to do Batch Normalization?

Batch normalization, or batchnorm for short, is proposed as a technique to help coordinate the update of multiple layers in the model. It does this by scaling the output of the layer, specifically by standardizing the activation's of each input variable per minibatch, such as the activation's of a node from the previous layer. Recall that standardization in 2.6.4, refers to re-scaling data to have a mean of zero and a standard deviation of one, e.g. a standard Gaussian. This process is also called whitening when applied to images in computer vision.

Standardizing the activation's of the prior layer means that assumptions the subsequent layer makes about the spread and distribution of inputs during the weight update will not change, at least not dramatically. This has the effect of stabilizing and speeding-up the training process of deep neural networks. Normalizing the inputs to the layer has an effect on the training of the model, dramatically reducing the number of epochs required. It can also have a regularizing effect, reducing generalization error much like the use of activation regularization. Although reducing internal co-variate shift was a motivation in the development of the method, there is some suggestion that instead batch normalization is effective because it smooths and, in turn, simplifies the optimization function that is being solved when training the network. [77]

3.11 Models Training & Validation

In the training phase, it is essential to control the evolution of the parameters update, validating the model before to end. Training data is used to find and tuning best hyperparameters (parameters whose value is set a prior to the training phase - number of hidden layers etc..,). Starting from training data, the algorithm learns, instead, the best parameters of the model (derived from the optimization part - weights and bias). Being learned from training data, the parameters must be tested on new data, called validation and test data. The validation and test dataset should follow the probability distribution of the training dataset. Validation data is used to control the parameters updating and to avoid overfitting. Comparing the performance between train and validation datasets is decided the candidate algorithm. Test data, usually, is non-labeled data and are used to obtain performance characteristics such as accuracy, sensitivity, specificity, F-measure etc. The concept of "overfitting" derives from the fact that models, trained on training data, and for this reason, they can learn intrinsic features of training data and could not be able to generalize on new data. Some methods can be used to solve this problems and make the model more stable in terms of predictions. A method to test the generalization of a model prediction is **Cross-Validation**: this methods use to partition the dataset in two parts, train and validation. The splitting is made in a certain percentage of the whole dataset (usually 80% train and 20% validation) to fit the model on the first part and test it on the second. This is repeated iteratively, using each time, different data partitions. The purpose is to make an average on all the measure of fitness in prediction obtained each time, to have a more accurate estimate of the model prediction. More details about how to tune hyper-parameters can be found

in 2.11

3.12 Transfer learning

Transfer learning [68] makes use of the knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars can be used to some extent to recognize trucks.

3.12.1 Pre-Training

When we train the network on a large dataset(for example: ImageNet) , we train all the parameters of the neural network and therefore the model is learned. It may take hours on your GPU.

3.12.2 Fine Tuning

We can give the new dataset to fine tune the pre-trained CNN. Consider that the new dataset is almost similar to the orginal dataset used for pre-training. Since the new dataset is similar, the same weights can be used for extracting the features from the new dataset. If the new dataset is very small, it's better to train only the final layers of the network to avoid overfitting, keeping all other layers fixed. So remove the final layers of the pre-trained network. Add new layers . Retrain only the new layers. If the new dataset is very much large, retrain the whole network with initial weights from the pretrained model.

3.12.3 If the new dataset is different ?

The earlier features of a ConvNet contain more generic features (e.g. edge detectors or color blob detectors), but later layers of the ConvNet becomes progressively more specific to the details of the classes contained in the original dataset. The earlier layers can help to extract the features of the new data. So it will be good if you fix the earlier layers and retrain the rest of the layers, if you got only small amount of data. If you have large amount of data, you can retrain the whole network with weights initialized from the pre-trained network.

3.12.4 Models for Transfer Learning

There are perhaps a dozen or more top-performing models for image recognition [63] that can be downloaded and used as the basis for image recognition and related computer vision tasks. Perhaps three [14] of the more popular models are as follows:

- VGG (e.g. VGG16 or VGG19).
- GoogLeNet (e.g. InceptionV3).
- Residual Network (e.g. ResNet50).

These models are both widely used for transfer learning both because of their performance [87], but also because they were examples that introduced specific architectural innovations, namely consistent and repeating structures (VGG), inception modules (GoogLeNet), and residual modules (ResNet).

Chapter 4

Automated Machine Learning

One of the biggest benefits of Machine Learning [30] is that it can be applied to almost any problem that humanity faces today. However, with that benefit, there are also challenges. Machine Learning algorithms need to be configured and tuned for every different real-world scenario. This makes it very manually intensive and takes a huge amount of time from a human supervising its development. This manual process is also error-prone, not efficient, and difficult to manage. Not to mention the scarcity of expertise out there to be able to configure and tune different types of algorithms. If the configuration, tuning, and model selection is automated, the deployment process will be made more efficient and humans can focus on the more important tasks such as model interpretability, ethics, and business outcomes. So we can agree that automating the Machine Learning model building process is of practical significance. Enter Automated Machine Learning

4.1 What is Auto ML ?

Automated Machine Learning (AutoML) [32] is the process of automating end-to-end process of applying Machine Learning (ML) to create, develop and deploy predictive models so that any enterprise benefits from data. ML relies on patterns and inference to discover insights and make predictions by using algorithms and statistical models.

AutoML covers the complete pipeline from the raw dataset to the deployable machine learning model. AutoML was proposed as an artificial intelligence-based solution to the ever-growing challenge of applying machine learning. The high degree of automation in AutoML allows non-experts to make use of machine learning models and techniques without requiring to become an expert in this field first. Automating the process of applying machine learning end-to-end additionally offers the advantages of producing simpler solutions, faster creation of those solutions, and models that often outperform hand-designed models.

AutoML tools help data scientists to automate the end-to-end life cycle of developing and deploying predictive models — from data preparation to feature engineering & selection, model configuration, training, validation, deploying and managing ML models. Especially as the amount of data and complexity of business problems increase, more machine learning systems need to be deployed and hence AutoML is applied to increase the productivity of data science teams.

4.2 Comparison to the standard approach

In a typical machine learning application [82], practitioners have a set of input data points to train on. The raw data may not be in a form that all algorithms can be applied to it. To make the data amenable for machine learning, an expert may have to apply appropriate data pre-processing, feature engineering, feature extraction, and feature selection methods. After these steps, practitioners must then perform algorithm selection and hyperparameter optimization to maximize the predictive performance of their model. All of these steps induce challenges, accumulating to a significant hurdle to get started with machine learning. AutoML dramatically simplifies these steps for non-experts.

4.3 Targets of automation

Automated machine learning can target various stages of the machine learning process. Steps to automate are:

1. Data preparation and ingestion (from raw data and miscellaneous formats)
   (a) Column type detection; e.g., boolean, discrete numerical, continuous numerical, or text
   (b) Column intent detection; e.g., target/label, stratification field, numerical feature, categorical text feature, or free text feature
   (c) Task detection; e.g., binary classification, regression, clustering, or ranking
2. Feature engineering
   (a) Feature selection
   (b) Feature extraction
   (c) Meta learning and transfer learning
   (d) Detection and handling of skewed data and/or missing values
3. Model selection
4. Hyperparameter optimization of the learning algorithm and featurization
5. Pipeline selection under time, memory, and complexity constraints
6. Selection of evaluation metrics and validation procedures
7. Analysis of results obtained
8. User interfaces and visualizations for automated machine learning

4.4 Auto ML tools

4.4.1 Auto-Sklearn

Auto-Sklearn is a framework built for popular Python machine learning library sklearn. It can perform following:

- Model selection

- Hyper-parameter tuning

It also has feature engineering methods like one hot encoding and uses sklearn estimators for classifiication and regression problems. Once it creates a pipeline it optimizes using Bayesian search. It works pretty well for small and medium sized datasets but cannot be applied to deep learning systems that yield state of the art performance on massive datasets. It does not perform feature engineering in the sense that data set features are combined to create new ones using mathematical primitives like in the case of Featuretools. Auto-sklearn is comparable to Auto-WEKA and Hyperopt-sklearn. The following are some of the classifiers that auto-sklearn can select from Decision Trees, Gaussian Naive Bayes, Gradient Boosting, kNN, LDA, SVM, Random Forest, and Linear Classifier (SGD) to name a few. In terms of preprocessing steps it supports the following: kernel PCA, select percentile, select rates, one-hot encoding, imputation, balancing, scaling, feature agglomeration, to name a few.

4.4.2 TPOT

Tree based pipeline optimization tool(TPOT). It optimizes pipelines using genetic programming which is very unique. It extends sklearn with its own regressor and classification methods. It then explores thousands of pipelines to get the best pipeline for your data. One drawback is it cannot process natural language inputs and also it cannot process categorical strings which have to be encoded to integers before passing to pipeline

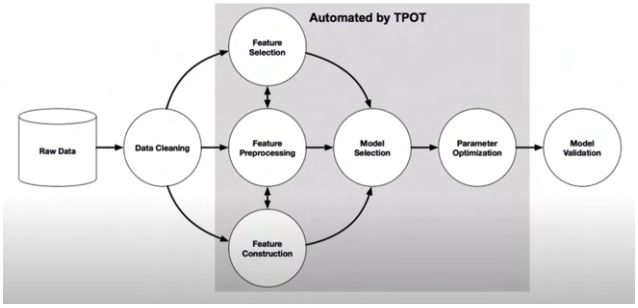


Figure 4.1. Automation by TPOT

The ML pipeline includes Data Cleansing, Feature Selection, Feature Preprocessing, Feature Construction, Model Selection, and Parameter Optimization.

4.4.3 H2O AutoML

H2O AutoML supports both traditional and deep learning model selection. It uses its own algorithm to build pipelines and includes feature engineering as well as hyper-parameter tuning. If you want to do only deep learning model automation then H2O AutoML is a great choice.H2O's Driverless AI is a platform for automatic machine learning. It can be used automate feature engineering, model validation, model tuning, model selection, and model deployment.

Driverless AI supports a whole range of what it calls 'transformers' which can be applied on a data set. The following are some of the transformers available on the platform: Dates Transformer, Cross Validation Categorical to Numeric Encoding , Text Transformer (using TF-IDF or count), One Hot Encoding Transformer, Ewma

Lags Transformer, Text Cluster Distance Transformer and many more. It can also be used to automatically train multiple algorithms at the same time. This is achieved by the h2o.automl package. It can automatically train your data using multiple different algorithms with different parameters such as GLM, Xgboost Random Forest, Deep Learning, Ensemble models etc..,

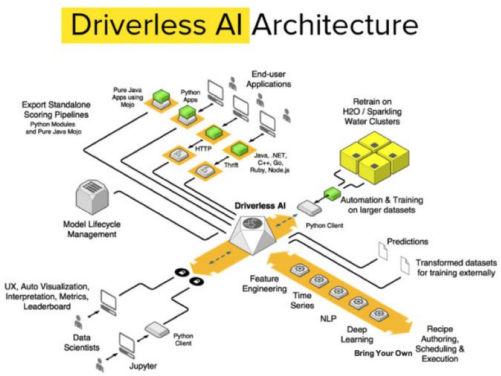


Figure 4.2. Driverless AI Architecture

H2O supports deployment of models by leveraging the concept of Java MOJOs (Model ObJect, Optimized). MOJOs are supported for AutoML, Deep Learning, DRF, GBM, GLM, GLRM, K-Means, Stacked Ensembles, SVM, Word2vec, and XGBoost models. It is highly integrated with Java type environments. For non-Java programmed models (such as R or Python) the model can be saved as serialized objects and loaded upon inferencing.

4.4.4 Auto Keras

Auto Keras [40] follows the design of classic Scikit learn api, so its simple to use and it uses powerful neural architecture search with Keras library under the hood to find the best model for the job. It uses an RNN controller trained in a loop that samples a candidate architecture, i.e. a child model, and then trains it to measure its performance on the task of desire. The controller then uses the performance as a guiding signal to find more promising architectures.

Auto-Keras workflow:

- User calls the API.
- The Searcher generates neural architectures on CPU.
- Graph builds real neural networks with parameters on RAM from the neural architectures.
- The neural network is copied to GPU for training.
- Trained neural networks are saved on storage devices.

Neural Architecture Search [25] is computationally expensive and time consuming, [64] uses 450 GPUs for 40,000 GPU hours. On the other hand, using less resources tends to produce pour results. To address this issue, AutoKeras uses Efficient Neural Architecture Search (ENAS). ENAS [56] applied a similar concept than transfer learning which established that parameters learned for a particular model

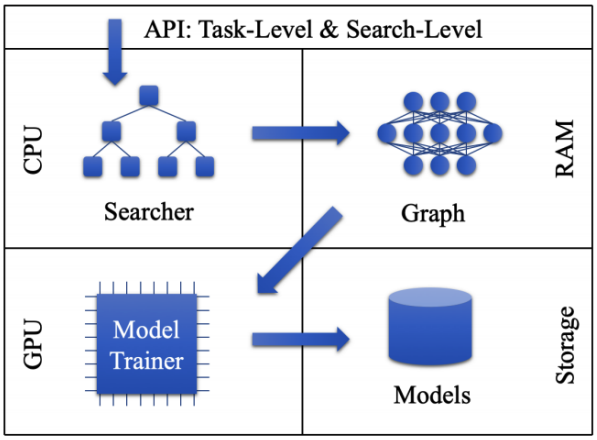


Figure 4.3. Auto-Keras System Overview [40]

on a particular task can be used for other models on other tasks. Hence, ENAS force all child models generated to share weights to deliberately avoid training each child model from scratch. The authors of the paper shows that not only is sharing parameters among child models possible, but it also allows for very strong performance .

4.4.5 Uber Ludwig

Automates the entire deep learning pipeline with minimal to no code. Best suitable for deep learning projects with massive datasets. Ludwig [65] is unique in its ability to help make deep learning easier to understand for non-experts and enable faster model improvement iteration cycles for experienced machine learning developers and researchers alike. By using Ludwig, experts and researchers can simplify the prototyping process and streamline data processing so that they can focus on developing deep learning architectures rather than data wrangling.

Uber's Ludwig is a toolbox build on top of Tensorflow that allows users to create and train models without needing to write any code. Uber's Ludwig was built with 5 core principles in mind:

- **No coding required**: no coding skills are required to train a model and use it for obtaining predictions.
- **Generality**: a new data type-based approach to deep learning model design that makes the tool usable across many different use cases.
- **Flexibility**: experienced users have extensive control over model building and training, while newcomers will find it easy to use.
- **Extensibility**: easy to add new model architecture and new feature data types.
- **Understandability**: standard visualizations to understand the performance of models and compare their predictions.

The main new idea that Ludwig introduces is the notion of data type-specific encoders and decoders, which results in a highly modularized and extensible architecture: each type of data supported (text, images, categories, and so on) has

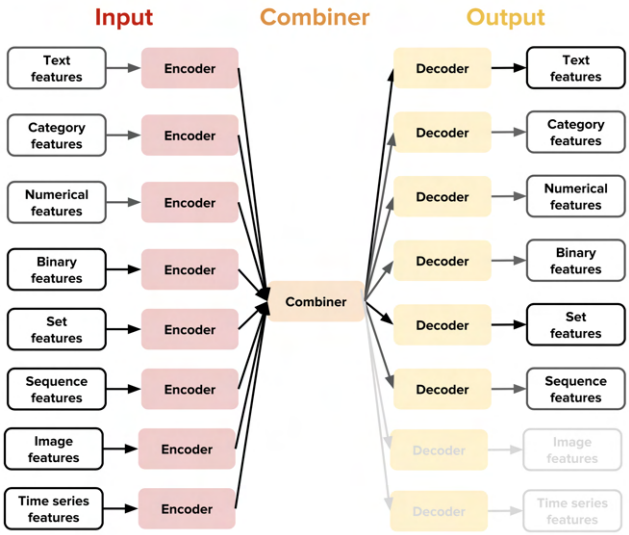


Figure 4.4. Feature data abstraction using Ludwig [65]

a specific preprocessing function. In short, encoders map the raw data to tensors, and decoders map tensors to the raw data. By composing these data type-specific components, users can make Ludwig train models on a wide variety of tasks. For example, by combining a text encoder and a category decoder, the user can obtain a text classifier, while combining an image encoder and a text decoder will enable the user to obtain an image captioning model.

Each data type may have more than one encoder and decoder. For instance, text can be encoded with a convolutional neural network (CNN), a recurrent neural network (RNN), or other encoders. The user can then specify which one to use and its hyperparameters directly in the model definition file without having to write a single line of code.

This versatile and flexible encoder-decoder architecture makes it easy for less experienced deep learning practitioners to train models for diverse machine learning tasks, such as text classification, object classification, image captioning, sequence tagging, regression, language modeling, machine translation, time series forecasting, and question answering. This opens up a variety of use cases that would typically be out of reach for inexperienced practitioners, and allows users experienced in one domain to approach new domains.

4.4.6 Rapid Miner

Rapid Miner [57] is a Data Science Platform for quickly analyzing data. Meant largely for non-programmers and researchers. You have a load of data. You have idea in your mind. And you easily create processes, import data into them, run them over and throw a prediction model. RM supports porting ML models to web apps(flask or nodeJS), android, iOS, etc, which is why it unifies the entire spectrum of the Big Data Analytics Lifecycle.

The Best part about this is, it simplifies the various scattered tasks of data mining and analysis. This is a place to load data (from anywhere, say Hadoop, Cloud, RDBMS, NoSQL, pdf, etc,etc), pre-process and prepare data using standard industrial methods (group items by categories or spawn new child tables or join tables or interpolate missing data, etc), train AI models (even train optimal deep

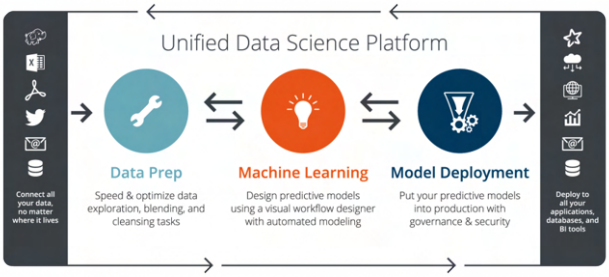


Figure 4.5. Rapid miner platform [71]

learning models: Random Forests, XGBoost, Gradient Boost, etc) or clustering or pruning outliers, to even visualizing outputs.

Finally you easily deploy these models on the cloud or in the production environment. Just need to create user interfaces to collect real time data and run it on the trained model to serve a task. I specialize in android :), and RM's wrapper to android was so easy to grasp than even some of the deep learning frameworks' (like keras or tensorflow).

4.5 Containerization with Docker

4.5.1 What is a container ?

A container is a standard unit of software that packages up code and all its dependencies so that the application runs quickly and reliably from one computing environment to another. In other words, with containers we are performing an operating system level virtualization and what that means is that a container includes everything we need to run our application code.

4.5.2 What is Docker ?

Docker is a tool to make creating, deploying and running containers easy. Realeased in 2013, Docker is open source. A Docker container is a standardized unit of software development, containing everything that your software application needs to run: code, run time, system tools, system libraries, etc. Docker containers are created from a read-only template called an image.

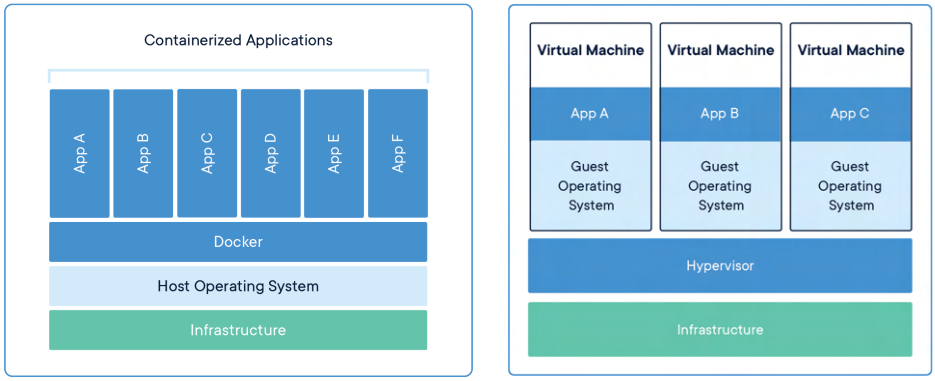
4.5.3 Containers vs Virtual machines

Containers and virtual machines(VM) have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware making Containers more portable and efficient. Figure 4.6 shows the difference in architecture of both Docker containers and Virtual machine

4.5.4 Why use containers ?

Containers are a great choice especially for running machine learning applications in production due to following reasons:

- Reproducibility



(a) *Docker container* **(b)** *Virtual machine*

Figure 4.6. Docker vs Virtual machine [?]

- Isolation
- Simplicity of environment management(Great for making staging/UAT(user acceptance tests match production)
- Ease of continuous integration
- Much faster and more lightweight than a VM
- Container orchestration options (e.g.Kubernetes)

Docker is the most popular tool for creating and running containers

4.6 Continuous Integration and Continuous Deployment

Continuous Integration and Continuous Deployment (CI/CD) automates the states of Machine learning application development. 4.7 shows an overview of those stages where build here refers to preparing everything needed to run your code. So in the case of Python that's installing dependencies in your requirements making sure that the operating system that's going to run your code is set up. The virtual machine or docker container is built and ready to go. It's about automatically testing your code and as soon as you merge in a feature branch automatically releasing that code to either production or a testing environment. User acceptance testing or sandbox and it just ensures that every step along the way is done automatically with no need for a person to run a script Or SSH into a machine and that adds a huge amount of value.

Figure 4.7. CI/CD workflow

What this means is that the system is always in a releasable state and that gives a lot of peace of mind and is extremely valuable and the reason for that is

that you have these faster and more regular release cycles. Code is being in a lot of cases put out into production every day. These small changes mean that the chances of breaking something of doing something that interferes with another system particularly in a micro service environment is reduced. Jenkins, CircleCI, TravisCII, Bamboo, GitlabCI, Teamcity are some popular CI/CD tools.

4.7 Machine Learning Democratization

The prevalence of digital technology is due to initiatives that sought to make them more accessible; for example, graphical interfaces substantially democratized the use of computers. Today, users of various abilities can employ end-user development (EUD) or end-user programming (EUP) tools to build their own computer programs using graphical programming environments, such as Simulink[4], LabView [15], and Scratch [11]. Applications such as Clementine, WEKA [16], RapidMiner [6], and recently Ludwig [8] realised the idea of interactive machine learning; namely, the development of machine learning (ML) models without the user having to write computer code. However, there are several aspects of EUDs that limit their accessibility and use. EUDs use technical nomenclature and the user experience (UX) of their interfaces have arguably not been optimised for lay-users. EUD tools normally run on desktop machines whereas the general public are increasingly becoming more comfortable with web-based tools for email, social media and office applications. Finally, these EUD tools do not automatically deal with data cleansing, wrangling, missing data imputation and therefore require the user to have some deeper understanding of the ML process.

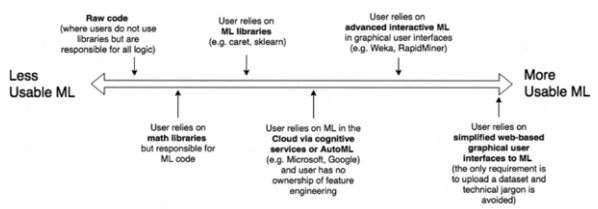


Figure 4.8. Spectrum of usable machine learning, from writing raw code (less usable) to web-based user interfaces (more usable). [13]

More recent advances involve running ML in the cloud, which has been described as 'ML as a Service' (MLaaS or Cloud ML). A new initiative referred to as automated machine learning (AutoML) is evolving (e.g.Google AutoML, Auto-WEKA), where a user provides a dataset (in the case of AutoML, via a web-based interface) and a set of algorithms automatically performs a task that is normally completed by a data scientist, such as feature engineering, model selection, and optimisation. AutoML algorithms tests a large number of feature sets, hyperparameters and permutations of ML techniques allowing for the automatic creation of the 'best' model

4.8 Limitations of Auto ML

Following are some of the limitations of AutoML:

- AutoML while it works well with supervised learning algorithms with labelled data, AutoML still needs to improve for unsupervised learning and reinforcement learning(which are used in time based real-world environment like real

Strengths & Opportunities	Weaknesses & Threats
1. Allowing non-technical users and the public to use and benefit from machine learning	1. Could lower the number of data science jobs and impetus to recruit specialists
2. Facilitates onboarding and machine learning education in schools and amongst adolescents	2. Could result in the deployment of inaccurate machine learning algorithms due to a lack of understanding of statistical concepts such as sampling biases, overfitting, concept drift, data leakage etc.
3. Increases widespread adoption and use of machine learning	3. Could result in the inadvertent deployment of biased and unfair machine learning algorithms due to lack of understanding in ethical practices and use of features such as gender and race for predicting or classifying outcomes
4. Allows industries and small companies to exploit machine learning without the need to recruit a data scientist or pay for data science services	4. Unintended proxies – even without explicit features such as gender, there may be other features which are correlated with gender and thus create the potential for discriminatory rules.
5. Could improve decision making in many industries and companies	5. Could expedite the spread of automation bias where the public 'over' trust machines to make decisions without using their own human reasoning skills and without overreading or second guessing machine decisions
6. Could result in new innovative applications of machine learning	6. Could realise the automation paradox where people lose their skills as a result of sustained dependence on machine learning
7. Opportunity for gradual upskilling where lay users can use educational features to gradually understand the ML process	
8. Potential increased detection of human bias but requires new innovation in automated tools to assist the user and expose bias in datasets of existing human decision making.	

Figure 4.9. SWOT analysis of democratising usable ML [13]

world robot or a game). AlphaGo is one of the best example of AutoML for reinforcement learning. It is a deep learning software for Go game and it automatically improves itself by playing against itself over and over again in that way it learnt it learnt its own model and hyper-parameters to improve itself over time.

- AutoML also cannot handle complex datatypes well.
- As of now feature engineering hasn't been automated that well. That means the process of choosing what are the ideal features to feed to a model for your problem still requires domain knowledge.

Chapter 5

ANOMA Platform

Machine learning in production requires multiple different components in order to work:

- Infrastructure
- Applications
- Data
- Documentation
- Configuration and more.

Together, these parts form the overall system. These vary from simple web applications, to incredibly complex pipelines built by hundreds of people.

5.1 Architecture

5.1.1 What is Architecture ?

[26] defines Architecture as: "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution" In a nutshell software architecture comprises the way software components are arranged and the interactions between them.

5.1.2 Why start with Architecture ?

Developing and deploying machine learning systems is relatively fast and cheap but maintaining them effectively over time is difficult and expensive machine learning systems have a special capacity for incurring technical debt because they have all the maintenance problems of traditional code plus an additional set of machine learning specific issues. Therefore clarity and planning and architecture design is very important to help us mitigate these issues because machine learning systems require close cooperation between data science engineering and dev ops as well as the business itself, a shared understanding of the system architecture and responsibilities is essential for effective cooperation.

5.1.3 ML System Contributors

The architecture of a production machine learning system needs to take into account business requirements, as well as the unique challenges at the intersection of data science, software engineering and devops. When we look at the contributors

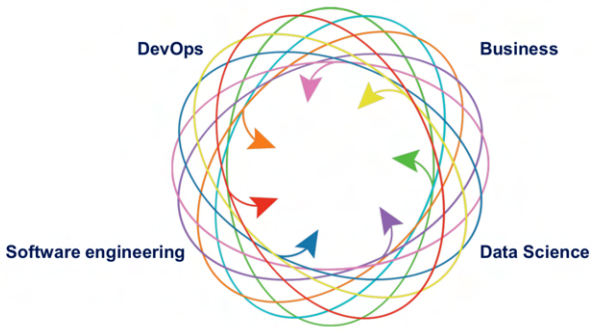


Figure 5.1. Machine learning project contributors

to a machine learning system you can see that we have quite a mix. So you have data scientists developing the models software engineers taking the models and putting them into applications dev ops doing the deployments and the business mix of executives product managersetc. determining what the requirements are. And in this sort of context there's the risk of code just being thrown over the wall from one department to another where no one understands the full process. As you can imagine this can result in errors and wasted time.

5.1.4 Key Principles for ML System Architecture

- **Reproducibility**: Have the ability to replicate a given ML prediction
- **Automation**: Retrain, update and deploy models as part of an automated pipeline
- **Extensibility**: Have the ability to easily add and update models.
- **Modularity**: Preprocessing/feature engineering code used in training should be organized into clear pipelines
- **Scalability**: Ability to serve model predictions to large numbers of customers (within time constraints)
- **Testing**: Test variation between model versions

5.2 Design approaches for ML system architecture

Here are four general machine learning architectures.

- **Rest API**: trained by batch predicts on the fly and serve via a REST API. For example a model trained and persisted offline then loaded into a web application that can give real time predictions about the price of a house. When details about a given house are posted by a client to REST API

- **Shared databases**: Train by batch predict by batch and then serve through a shared database. If we continue the example of housing predictions then for this a user might upload a CSP of houses with the input details and then wait for 30 minutes for an email telling them to check the website for results. This application would perform the batch prediction with an asynchronous task queue such as celery and then those results would be saved to a shared database which could then be accessed by a web application to display the results.

- **Streaming**: Prediction by streaming. So here an application would have access to a conveyor belt of updated data and potentially models. This would be something like a combination of a streaming framework such as spark structured streaming with the data and updated models being read from a dedicated distributed queue such as Apache Kafka or AWS Kansas. Considering housing example, Our application could potentially read a prediction from a distributed queue or it could equally easily receive an updated model from that distributed queue and load and use the updated model seamlessly.

- **Mobile App**: Trained by batch predict on mobile.So here we would have an IOS or Andriod app for example using the core AML framework that would not need to call a back end service to make a prediction, instead the predictions could be made on the device

Note: In this research work for ANOMA platform we used the REST API based design architecture where we can train and test from the same application which is a powerful features since this is a hybrid of above 4 design considerations.

Below is a comparison table highlighting the pros and cons of above 4 design architectures,

	Pattern 1 (REST API)	Pattern 2 (Shared DB)	Pattern 3 (Streaming)	Pattern 4 (Mobile App)
Training	Batch	Batch	Streaming	Streaming
Prediction	On the fly	Batch	Streaming	On the fly
Prediction result delivery	Via REST API	Through the shared DB	Streaming via Message Queue	Via in-process API on mobile
Latency for prediction	So so	High	Very Low	Low
System Management Difficulty	So so	Easy	Very Hard	So so

Figure 5.2. Architecture comparisons

5.3 Architecture of ANOMA platform

We used MongoDB to store and retrieve our datasets. And a dedicated deep learning server for all our machine learning needs. We have a interface based on Bootstrap and Vue.js and a middle ware based on Flask to join all the pieces together. This way we achieve a loosely coupled software architecture which is adaptable and easily scalable to large datasets. Interface is responsive and can be accessible from any device like PC, Mobile or Tablet. We are hosting the engine on GCP with a Docker container with Jenkins for CI/CD. This way it can be easily ported to other environments. Flask middle acts as a API layer and bridge between interface and other layers like Deep learning engine and Mongo server

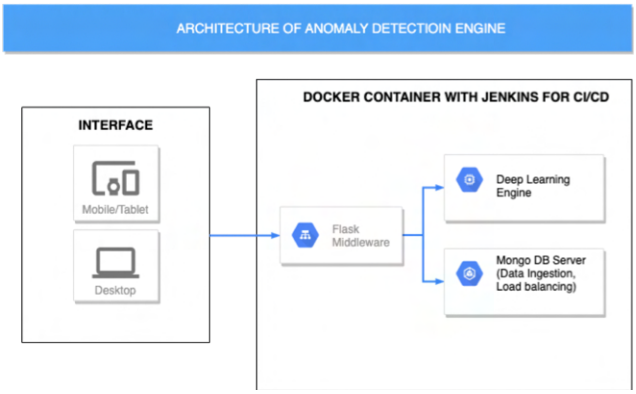


Figure 5.3. Architecture of ANOMA platform

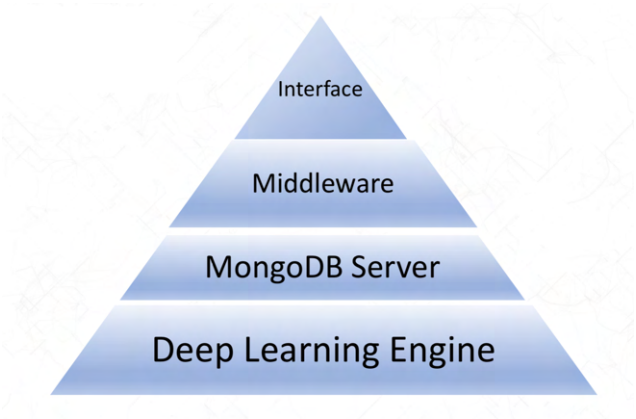


Figure 5.4. Pyramid architecture of ANOMA platform

5.4 Methodology for ANOMA platform

We put more emphasis on following methodology for our general-purpose engine so all the boxes pertaining to the holistic data science research are ticked. Below figure shows different steps of our workflow:

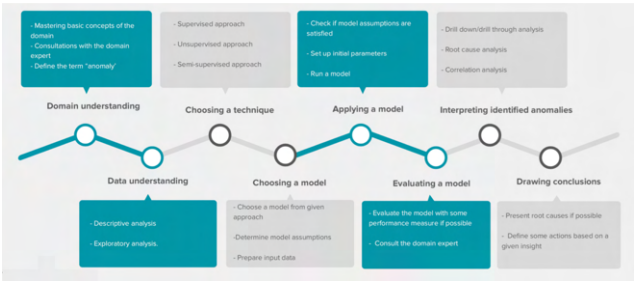


Figure 5.5. Methodology of ANOMA platform

Following are the crucial steps for our analysis:

- Getting required domain knowledge for the dataset(s) that are being analysed
- Understand the data through Exploratory data analysis

- Deciding on a machine learning model
- Properly interpreting the anomalies

5.5 Datasets

The dataset which we used for this particular project is Any nefarious activity that can damage an information system can be broadly classified as an intrusion. Anomaly detection can be effective in both detecting and solving intrusions of any kind. Common data-centric intrusions include cyberattacks, data breaches, or even data defects. The dataset [10] consists of a wide variety of intrusions simulated in a military network environment to acquire raw TCP/IP dump data for a network by simulating a typical US Air Force LAN. The LAN was focused like a real environment and blasted with multiple attacks. A connection is a sequence of TCP packets starting and ending at some time duration between which data flows to and from a source IP address to a target IP address under some well-defined protocol. Also, each connection is labelled as either normal or as an attack with exactly one specific attack type. Each connection record consists of about 100 bytes. For each TCP/IP connection, 41 quantitative and qualitative features are obtained from normal and attack data (3 qualitative and 38 quantitative features) .The class variable has two categories: Normal, Anomalous

5.6 ConvNet's used in ANOMA

5.6.1 Background

A Convolutional Neural Network (CNN, or ConvNet) [44] are a special kind of multi-layer neural networks, designed to recognize visual patterns directly from pixel images with minimal preprocessing.. The ImageNet project is a large visual database designed for use in visual object recognition software research. The ImageNet project runs an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where software programs compete to correctly classify and detect objects and scenes [21]

Following are some of the deep learning models we used for our platform

- Lenet5
- Alexnet
- VGG-Net

For ease of representing complex network architectures below legends are utilized to save some space and workflow analysis,

5.6.2 Lenet5(1998)

LeNet-5 [21], a pioneering 7-level convolutional network by LeCun et al [44] in 1998, that classifies digits, was applied by several banks to recognise hand-written numbers on checks (cheques) digitized in 32x32 pixel greyscale inputimages. The ability to process higher resolution images requires larger and more convolutional layers, so this technique is constrained by the availability of computing resources.

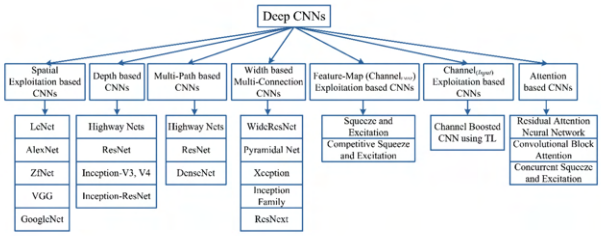


Figure 5.6. Taxonomy of deep CNN architectures showing seven different categories. [44]

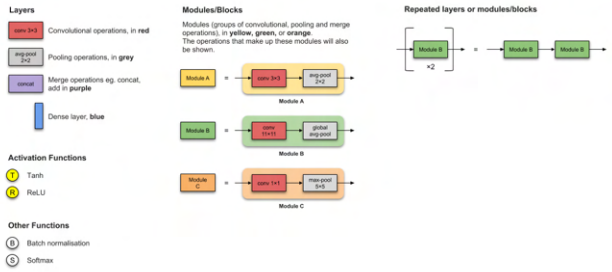


Figure 5.7. Legends for different Conv nets [42]

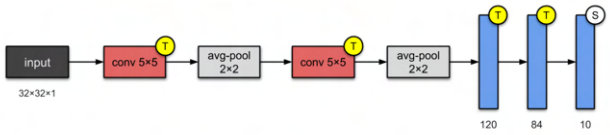


Figure 5.8. LeNet-5 architecture [51] [42]

5.6.3 Alexnet(2012)

In 2012, AlexNet significantly outperformed all the prior competitors and won the challenge by reducing the top-5 error from 26% to 15.3%. The second place top-5 error rate, which was not a CNN variation, was around 26.2%.

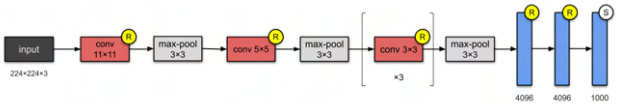


Figure 5.9. Alexnet architecture [42] [46]

The network had a very similar architecture as LeNet by Yann LeCun et al [51] but was deeper, with more filters per layer, and with stacked convolutional layers. It consisted 11x11, 5x5,3x3, convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum. It attached ReLU activations after every convolutional and fully-connected layer. AlexNet was trained for 6 days simultaneously on two Nvidia Geforce GTX 580 GPUs which is the reason for why their network is split into two pipelines. AlexNet was designed by the SuperVision group, consisting of Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever.

5.6.4 VGG-16(2014)

The runner-up at the ILSVRC 2014 competition is dubbed VGGNet by the community and was developed by Simonyan and Zisserman. VGGNet consists of 16 convolutional layers and is very appealing because of its very uniform architecture. Similar to AlexNet, only 3x3 convolutions, but lots of filters. Trained on 4 GPUs for 2–3 weeks. It is currently the most preferred choice in the community for extracting features from images. The weight configuration of the VGGNet is publicly available and has been used in many other applications and challenges as a baseline feature extractor. However, VGGNet consists of 138 million parameters, which can be a bit challenging to handle.

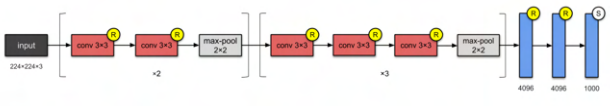


Figure 5.10. VGG-16 architecture [42] [80]

5.7 Machine Learning pipelines

A pipeline is a set of data processing steps connected in series, where typically, the output of one element is the input of the next one. The elements of a pipeline can be executed in parallel or in time-sliced fashion. This is useful when we require use of big data, or high computing power, e.g., for neural networks.

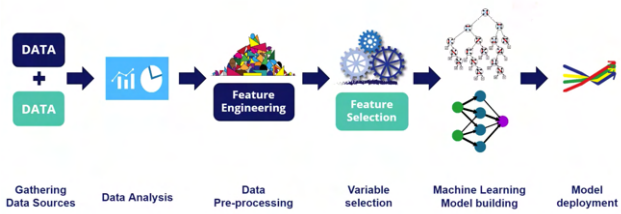


Figure 5.11. Machine Learning pipeline overview

There are 3 approaches -

- Procedural programming
- Custom pipeline
- Third party pipeline

5.7.1 Procedural programming

In the context of Machine Learning pipelines, procedural programming refers to making use of functions for each step of machine learning like feature creation, feature transformation, features selection and model training and model scoring steps. Then we can call these functions in a predefined order one after the other which actually creates a pipeline of task execution. In procedural programming, procedures can also be understood as routines or subroutines or functions.

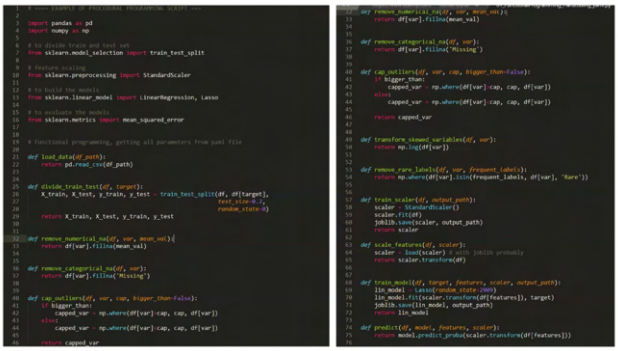


Figure 5.12. Procedural machine learning workflow

As you can see from 5.12 we have a bunch of functions for different steps of machine learning workflow like loading the data, pre-processing the features, splitting the data in to train and validation sets, training and prediction for machine learning model. so we call these functions in a specific order in order to reproduce the same results on production. Figure 5.13 shows how we can combine functions in Figure 5.12 to create a machine learning workflow. Let's see what changes when we deploy

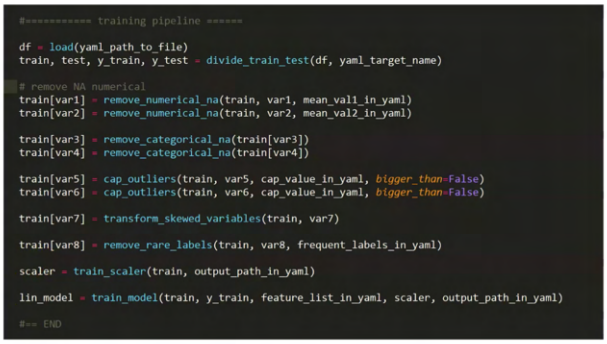


Figure 5.13. Creating workflow using functions

our model on to production environment. Figure 5.14 shows how we load the trained model and its weights to be used on new data in production.

Using YAML files for workflows

Typically machine learning workflow include a YAML file where we have all the bits of information that are critical and are shared across the functions, train and prediction/score scripts. so for example in this YAML file we store the path to the files that we need to save on the path to the files that we need to load. And we also can have code values that we need for example to replace missing values. We can also hard code the values by which we want to keep the outliers we can also hard code the frequent the least of frequent labels on these where we are able to remove rare labels from our data sets

We can also hard code the list of features that we want to use to train our models this way if we want to refresh or retrain our mothers with a different set of values, we only need to change the values here in the YAML file and then we can run the

```
# ========== scoring pipeline ========

data = 'load it from somewhere'

# remove NA numerical
data[var1] = remove_numerical_na(data, var1, mean_val1_in_yaml)
data[var2] = remove_numerical_na(data, var2, mean_val2_in_yaml)

data[var3] = remove_categorical_na(data[var3])
data[var4] = remove_categorical_na(data[var4])

data[var5] = cap_outliers(data, var5, cap_value_in_yaml, bigger_than=False)
data[var6] = cap_outliers(data, var6, cap_value_in_yaml, bigger_than=False)

data[var7] = transform_skewed_variables(data, var7)

data[var8] = remove_rare_labels(data, var8, frequent_labels_in_yaml)

scaler = joblib.load((output_path_in_yaml_to_scaler)
lin_model = joblib.load(output_path_in_yaml_to_model)

score = predict(data, lin_model, feature_list_in_yaml, scaler)

# ===== END
```

Figure 5.14. Using procedural workflows in production

entire train pipeline to retrain our models. Figure 5.15 shows an example structure of an YAML file

```
### example of yaml ####

#paths
path_to_dataset = "path_to_my_dataset"
output_scaler_path = 'path_to_store_scaler'
output_model_path = 'path_to_store_model'

# preproc
var1_mean_val = 1
var2_mean_val = 2

var4_cap_value = 1000
var5_cap_value = 5000

var8_frequent_labels = ['frequent1', 'frequent2', 'frequent3']

# features
features = ['var1', 'var2', 'var3', 'var4', 'etc']


#====== END =======
```

Figure 5.15. YAML file for Machine learning workflow configuration

Below are some advantages and disadvantages of Procedural programming based machine learning workflows:

Advantages

- Straightforward from jupyter notebook, so very good for quick prototyping and rapid development
- No software development skills required
- Easy to manually check if it reproduces the original model

Disadvantages

- Can get buggy
- Difficult to test
- Difficult to build software on top of it

- Need to save a lot of intermediate files to store the transformation parameters

5.7.2 Custom pipeline

For custom pipeline we make use of Object oriented programming paradigm. In Object-oriented programming (OOP) the "objects" can learn and store this parameters, so the parameters get automatically refreshed every time model is re-trained there by avoiding the need for manual hard-coding.

A custom Machine Learning pipeline is therefore a sequence of steps, aimed at loading and transforming the data, to get it ready for training or scoring, where:

- We write the processing steps as objects (OOP)
- We write the sequence, i.e., the pipeline as objects (OOP)

We save one object, the pipeline, as a Python pickle, with all the information needed to transform the raw inputs and get the predictions

Below are some advantages and disadvantages of custom machine learning pipelines:

Advantages:

- Can be tested, versioned, tracked and controlled
- Can build future models on top
- Good software developer practice
- Built to satisfy business needs

Disadvantages:

- Requires team of software developers to build and maintain
- Overhead for DS to familiarise with code for debugging or adding on future models
- Preprocessor not reusable, need to re-write Preprocessor class for each new ML model
- Need to write new pipeline for each new ML model
- Lacks versatility, may constrain DS to what is available with the implemented pipeline

5.7.3 Third party pipeline: Scikit-Learn

Scikit-Learn is a Python library that provides a solid implementation of a range of machine learning algorithms. Following are best parts of the Scikit-Learn package,

- Provides efficient versions of a large number of common algorithms.
- Clean, uniform, and streamlined API.
- Most of its algorithms follow the same functionality.

- Once you understand the basic use and syntax of Scikit-Learn for one type of model, switching to a new model or algorithm is very straightforward.
- Provides useful and complete online documentation that allows you to understand both what the algorithm is about and how to use it from scikit-learn
- Well established in the community to such an extent that new packages are typically designed following scikit-learn functionality to be quickly adopted by end users, e.g, Keras, MLXtend

Scikit-Learn objects

Scikit-learn objects are divided mainly in to 3 categories -

- **Transformers** - class that have fit and transform method, it transforms data
  - Scalers
  - Feature selectors
  - One hot encoders
- **Estimator/Predictor** - class that has fit and predict methods, it fits and predicts.
  - Any ML algorithm like LASSO, Decision trees, SVM, etc
- **Pipeline** - class that allows you to list an run transformers and predictors in sequence. It is just an abstract notion, it's not any existing Machine Learning algorithm.
  - All steps should be transformers except the last one
  - Last step should be a predictor

Figure 5.16 is a good example of pipeline usage [27]. Pipeline gives you a single interface for all 3 steps of transformation and resulting estimator. It encapsulates transformers and predictors inside

```
vect = CountVectorizer()
tfidf = TfidfTransformer()
clf = SGDClassifier()

vX = vect.fit_transform(Xtrain)
tfidfX = tfidf.fit_transform(vX)
predicted = clf.fit_predict(tfidfX)

# Now evaluate all steps on test set
vX = vect.fit_transform(Xtest)
tfidfX = tfidf.fit_transform(vX)
predicted = clf.fit_predict(tfidfX)
```

(a) *Without pipeline*

```
pipeline = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', SGDClassifier()),
])
predicted = pipeline.fit(Xtrain).predict(Xtrain)
# Now evaluate all steps on test set
predicted = pipeline.predict(Xtest)
```

(b) *With pipeline*

Figure 5.16. Scikit-learn objects with and without pipeline

Figure 5.17 shows how we can use a third party pipeline to a regression problem.

Here are some of pros and cons of using Scikit-learn pipelines,

```
price_pipe = Pipeline(
    [
        ('categorical_imputer',
            pp.CategoricalImputer(variables=config.CATEGORICAL_VARS_WITH_NA)),

        ('numerical_inputer',
            pp.NumericalImputer(variables=config.NUMERICAL_VARS_WITH_NA)),

        ('temporal_variable',
            pp.TemporalVariableEstimator(
                variables=config.TEMPORAL_VARS,
                reference_variable=config.DROP_FEATURES)),

        ('rare_label_encoder',
            pp.RareLabelCategoricalEncoder(
                tol=0.01,
                variables=config.CATEGORICAL_VARS)),

        ('categorical_encoder',
            pp.CategoricalEncoder(variables=config.CATEGORICAL_VARS)),

        ('log_transformer',
            pp.LogTransformer(variables=config.NUMERICALS_LOG_VARS)),

        ('drop_features',
            pp.DropUnecessaryFeatures(variables_to_drop=config.DROP_FEATURES)),

        ('scaler', MinMaxScaler()),
        ('Linear_model', Lasso(alpha=0.005, random_state=0))
    ]
)
```

Figure 5.17. Third party pipeline using Scikit-learn objects

Advantages:

- Can be tested, versioned, tracked and controlled
- Can build future models on top
- Good software developer practice
- Leverages the power of acknowledged API
- Data scientists familiar with Pipeline use, reduced over-head
- Engineering steps can be packaged and re-used in future ML models

Disadvantages:

- Requires team of software developers to build and maintain
- Overhead for software developers to familiarise with code for Sklearn API which can result in difficulties during debugging

5.8 Building a reproducible machine learning pipeline

Reproducibility is a real problem that exists both in businesses and in academia and the consequences of the lack of reproducibility can have significant financial costs not to mention the loss of time and the potential loss of reputation. So it is important to try and get this right right from the beginning. So what do we mean by reproducibility in machine learning with a fine reproducibility as the ability to duplicate and model exactly such that given the same raw data as input both mothers return the same output remember that we do not just deploy the machine learning algorithms we rather deploy the entire machine learning pipeline. So we need to make sure that every single step of the pipeline is reproducible.

5.9 Machine Learning system components

Machine learning systems deployed in production have following main components as shown in Figure 5.18. In a nutshell we have the following main components,

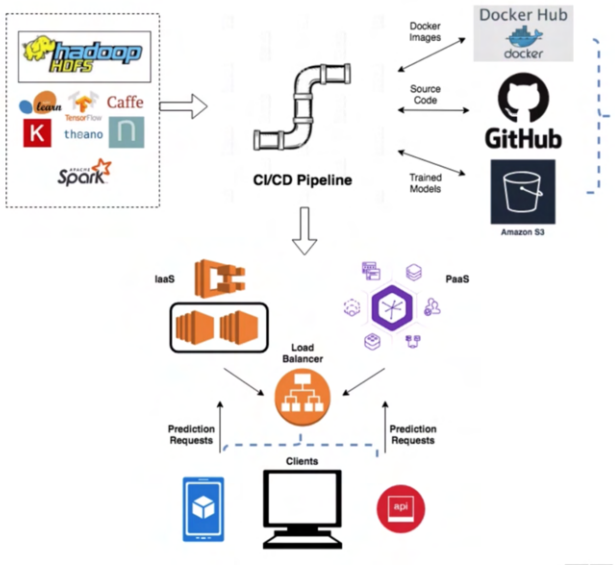


Figure 5.18. Machine Learning system components

- We have a **cloud infrastructure** which can be Platform as a service(PaaS) like for example Heroku or Infrastructure as a service (IaaS) for example AWS, GCP, Azure etc.., where you have complete control of what the actual configuration of your system will be.
- We also have a **container** environment like Docker or Kubernetes to create a container for deploying our application which runs on any configuration seamlessly. See 4.5 for more information regarding this.
- We also have a Continuous integration and continuous deployment(**CI/CD**) pipeline for example using a version control system like Git and CI/CD software like Jenkins to easily automate the stages of application development. See 4.6 for more details regarding this.

5.10 ANOMA pipeline

For ANOMA platform we used a combination of custom and Scikit-learn pipelines. For configuration parameters we used interface as shown in Figure 6.14. To throw some light on our methodology we relied both on .ini files and Mongo DB for configuration but for common configurations we relied on Interface(with Mongo DB) and for critical configurations we relied on .ini files to store something like file paths etc..,
Given a dataset, we take 2 routes for building the machine learning system,

- Data science approach mentioned in the Chapter 2
- Making the model production ready

5.10.1 Data science approach

We initially load the data sets in to Jupyter notebooks and a gist of what the dataset is about using Pandas functions like describe(), info(), $value_counts()$ etc.., to analyse the data set on a higher level. Then apply methodologies mentioned in Chapter 2 and Chapter 3

Common steps

Below are some of the common steps for machine learning base-line and deep learning state of the art. We follow the machine learning workflow in Chapter 2. In other words, we do the following steps to a given dataset,

- Data pre-processing steps mentioned in 2.5
- Feature engineering and feature selection steps as mentioned in 2.6
- Exploratory data analysis (EDA) there by feature selection as mentioned in 2.7

Machine Learning workflow

We follow the machine learning workflow post EDA in Chapter 2,

- Machine learning algorithm selection based on the data analysis and type of problem at hand as per the criterion mentioned in 2.8
- Applying different classification machine learning algorithms mentioned in 2.9 as we are more interested to classify anomalies and non-anomalies in this research. In this research we are interest to apply deep learning state-of-the art. But these algorithms give us the base line up on which we can decide on the deep learning model performance
- Access the model performance using the classification metrics mentioned in 2.10.
- Then based on the identified metrics we do hyper-parameter tuning as per the methodologies mentioned in 2.11

Deep Learning workflow

Once we know what the baseline performance from machine learning algorithms, we proceed to applying deep learning approach. In particular we perform following steps,

- Build the Convolutional neural networks(CNN) using the architectures mentioned in 5.6.
- Make use of different activation functions as in 3.3 , optimization algorithms as in 3.4.
- Handle over-fitting of models by using regularization approaches mentioned in 3.10.
- Access the model performance using the classification metrics mentioned in 2.10.
- Then based on the identified metrics we do hyper-parameter tuning as per the methodologies mentioned in 2.11

5.10.2 Making the model production ready

Once we are satisfied with our model performance, we proceed to building a pipeline for all the workflow steps from both Machine learning and Deep learning workflow so that we can access these steps from interface that we built via API. In short we do the following,

- Building pipelines
- Building interface
- Integrating the pipeline with interface
- Model deployment
- Integrate CI/CD to the pipeline

Building pipelines

We tried to make use of the best parts of custom pipelines as mentioned in 5.7.2 and Scikit-learn pipeline mentioned in 5.7.3. In short, following is how we build pipeline for ANOMA,

- Make use of Object oriented programming(OOP) to separate out the login and different steps like Data-prepossessing, EDA, database integration etc..,
- Read the configuration parameters from MongoDB as well as .ini files. We make use of .ini files only to store sensitive information like file paths, model paths etc..,
- Scikit-learn pipelines to integrate different steps and create a single entry point.
- Create API's in Flask to access from the interface.
- Integrate all the above to make the model ready for production.

Building interface

We have made the process building interfaces loosely coupled to a extent that both these steps can be performed in parallel for the most extent. More details about interface in Chapter 6. Specifically we made use of following tools for building interfaces,

- Bootstrap.js for responsive behavior so that the interface aligns well for all screen resolutions like Desktop, Tablet, Mobile etc..,
- Vue.js as a front end framework so that interface can be scaled well to complex requirements
- Plotly.js and D3.js for displaying charts
- Jquery for Document object model (DOM) manipulation for displaying dynamic content on the interface
- HTML5 and CSS3 for building different interface elements like forms, menus etc..,

- Gulp.js for code minification so that number of REST API calls and network bandwidth is optimized
- Flask as a middle-ware for exposing REST API's to the interface.

We tried to make the interface as simple and light weight as possible for ease of maintenance and development

Integrating the pipeline with interface

Once the above 2 steps of building pipeline and building interface are ready and unit tested, we join these 2 pieces so that we can interact with machine learning workflow through interface. Specifically we are interested in integrating following,

- Data set analysis
- Data visualization
- Model training
- Model prediction
- Updating the model configurations
- Add new data-sets through file upload

More details in Chapter 6
Note: For model training and prediction, we make use of the API's exposed from the pipeline for accessing the pre-saved models and model weights and other intermediate results. For other steps like Data visualization and analysis we use raw data sets with minimal data-preprocessing.

Model deployment

Once the interface and machine learning pipeline are integrated we containerize all the components of the application using Docker. For this we do the following,

- Create a git repository and push all the code
- Create different branches like main and version specific branches so that we can do incremental edits to the code to add new features in the future
- Update all the python packages and other tools that we use in our application in a requirement.txt file
- Define a starting point for the application into a .yaml file so that the target server knows how to start executing the code. Here you specify the python version, target docker image to be used etc..,
- Create an account in any cloud provided like Google cloud platform (GCP), Microsoft Azure, AWS or Platform as a system (PaaS) like Heroku.
- Once you have the account setup, create a new application which is like a target for deploying your code through Docker.
- Integrate application with your Git repository.

- Now when you click on deploy, all your code is pulled from Git to your cloud provider and all the tools like Docker image and other python packages are installed on the fly to run your application on the cloud
- Once the deployment is done, you will see a link where your machine learning application is deployed

Integrate CI/CD to the pipeline

Continuous Integration and Continuous Delivery (CI/CD) automates the states of Machine learning application development. More details in 4.6 Once the docker images are ready and your code is successfully deployed. We use a combination of version control system like Git (where your code is present) and an automation software like Jenkins to automate the deployment of your machine learning pipelines. What this means is, you can easily deploy new features with configuring the CI/CD system. This makes the deployment a hassle-free operation.

5.11 ANOMA technical stack

Technical Stack	
Machine Learning Step	Tool
Data pre-processing	Numpy, Pandas, ML Box
Feature Engineering	Pandas, Scikit-learn
Exploratory data analysis	Matplotlib, Seaborn, Plotly
Machine learning	Scikit-learn, Auto-Scikit-learn
Deep learning	Tensorflow, Keras, Auto-Keras
Middleware	Flask
Database	MongoDB
Model explainability	LIME
CI/CD	Git, Jenkins
Deployment	Docker, Google cloud platform
Front-end	Boostrap, D3.js, Vue.js

We tried to make use of the best tools available for our requirement as well as for different machine learning action items discussed in the above sections. Above is a table with the list of tools used for each step of machine learning workflow

Chapter 6

Results

6.1 Sections

Interface has following sections:

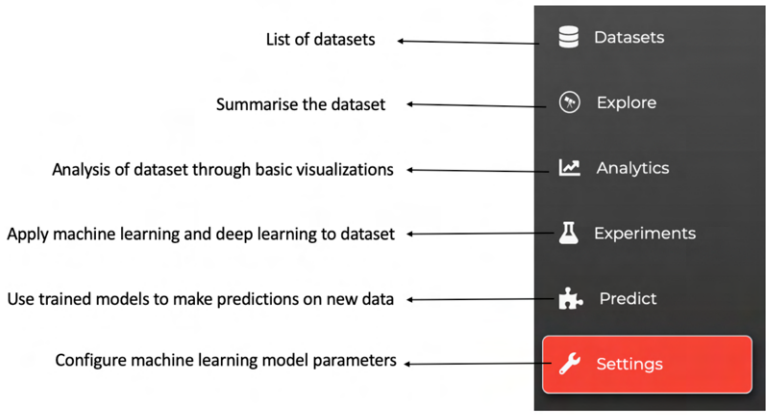


Figure 6.1. Sections of interface

6.2 List of data sets:

Summarise the data set

- As shown in Figure 6.1, we can upload a new data set from the local machine using button on top of list
- Once the data set is uploaded, it is saved in to MongoDB server via Middle ware server.
- Saving in MongoDB gives a lot of flexibility in terms of quick analysis and scaling of large datasets for doing heavy computations.
- We can get high level details about the size, number of rows and columns of a dataset

Note: Currently the platform supports CSV and XLS file for the upload and the upload works seamlessly for dataset up to 150MB

#	Name	Size	Columns	Rows
1	Id_Testset	19.47 MB	41	22544
2	nations	1.27 MB	11	5275
3	accident_reports	1016.41 KB	18	1920
4	titanic	198.7 KB	15	891
5	Id_Trainset	22.2 MB	42	25192

Figure 6.2. List of data sets

6.3 Exploring dataset:

More details about the structure of data set like type of individual fields (Integer, Float, Categorical) as shown in Figure 6.3, summary of each field by type as shown in Figures 6.5 and 6.4. This gives us a gist about the distribution of different fields.

Properties

*Structure of current dataset

Name	Type	NA count
duration	Integer	0
protocol_type	Categorical	0
service	Categorical	0
flag	Categorical	0
src_bytes	Integer	0
dst_bytes	Integer	0
land	Integer	0
wrong_fragment	Integer	0

Figure 6.3. Structure of data set

Numeric

*Summary statistics of all the numeric fields of the data set

	count	mean	std	min	25%	50%	75%	max
duration	5000.0	330.5	2828.7	0.0	0.0	0.0	0.0	41802.0
src_bytes	5000.0	86005.6	5400370.1	0.0	0.0	44.0	280.0	381709090.0
dst_bytes	5000.0	1968.6	24702.8	0.0	0.0	0.0	538.2	1639484.0
land	5000.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
wrong_fragment	5000.0	0.0	0.3	0.0	0.0	0.0	0.0	3.0
urgent	5000.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
hot	5000.0	0.2	2.0	0.0	0.0	0.0	0.0	30.0
num_failed_logins	5000.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0

Figure 6.4. Summary of numeric columns

6.4 Analytics

Understanding the data set through basic visualization to drill-down further in to structure of data set and identify outliers(through box plots), get an glance of

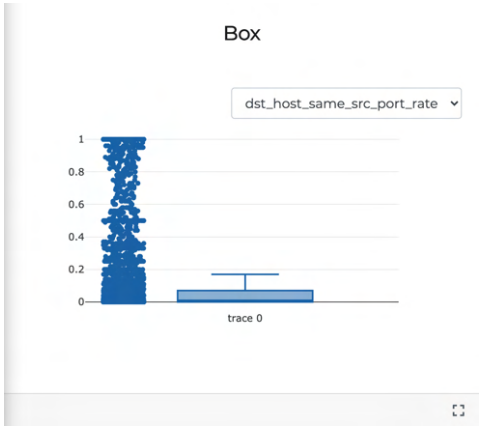
Categorical

*Summary statistics of all the categorical fields of the data set

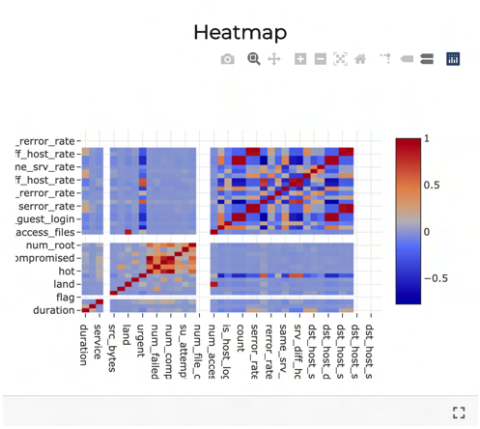
	count	unique	top	freq
protocol_type	5000	3	tcp	4080
service	5000	64	http	1564
flag	5000	11	SF	2964
class	5000	2	normal	2648

Figure 6.5. Summary of categorical columns

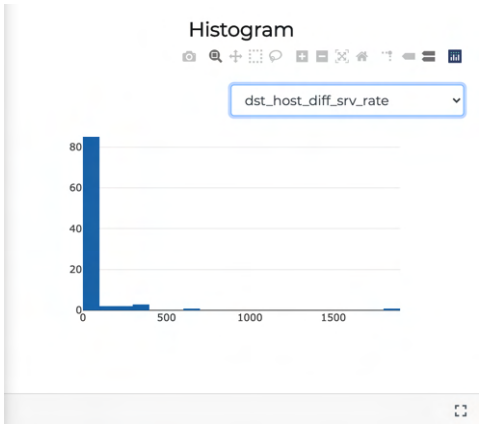
highly correlated variables (via heat map), distribution of categorical and numeric columns (via Pie and Histograms respectively) as shown in 6.6. For every chart there is a dropdown with corresponding fields of that chart type (for example in the dropdown of Box plot only continuous columns are present where as in dropdown of Pie chart only categorical columns are present) so that user of the interface can understand the distribution of individual columns of the dataset.



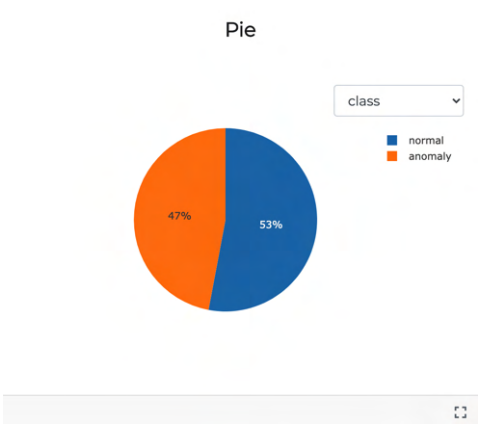
(a) *Box plot*



(b) *Heat-map*



(c) *Histogram*



(d) *Pie*

Figure 6.6. Analytics of data set

6.5 Experiment

Using state of the art Machine learning

- Understand the important features in the dataset as shown in Figure 6.7 and train the deep learning models, all with just couple of button clicks. We use a stacking machine learning algorithms such as Random Forest, Support vector machine (SVM), Decision tree to get the importance score of each feature in the dataset so that user of the interface can identify the important features contributing to the value of the output label.
- Understand how the models are performing with simple data visualizations and metric summaries as shown in Figure 6.11. User has an option to provide the train test split Example: 0.3 (Which means 70% data is for training and 30% data is for validation). Metrics such as Accuracy, Precision, Recall, F1-score and ROC score as mentioned in 2.10 are calculated both on Train and Validation sets.
- Each of these metrics are calculated for different machine learning algorithms such as Random Forest, Decision trees, Gradient boosting, k-Nearest neighbours, Logistic regression, Naive Bayes as mentioned in 2.9.
- These algorithms are displayed in the descending order of their Accuracy.
- We use the machine learning algorithm metrics as a baseline to understand the performance of our deep learning models.

Importance

*Features sorted by their importance

Feature	Trainset score	Testset score
src_bytes	0.239	0.275
dst_host_same_srv_rate	0.018	0.116
dst_bytes	0.036	0.089
dst_host_srv_count	0.062	0.078
flag	0.067	0.070
diff_srv_rate	0.056	0.066
same_srv_rate	0.127	0.064
protocol_type	0.030	0.048

Figure 6.7. Feature importance sorted in ascending order

Metrics

*Metric values sorted by accuracy

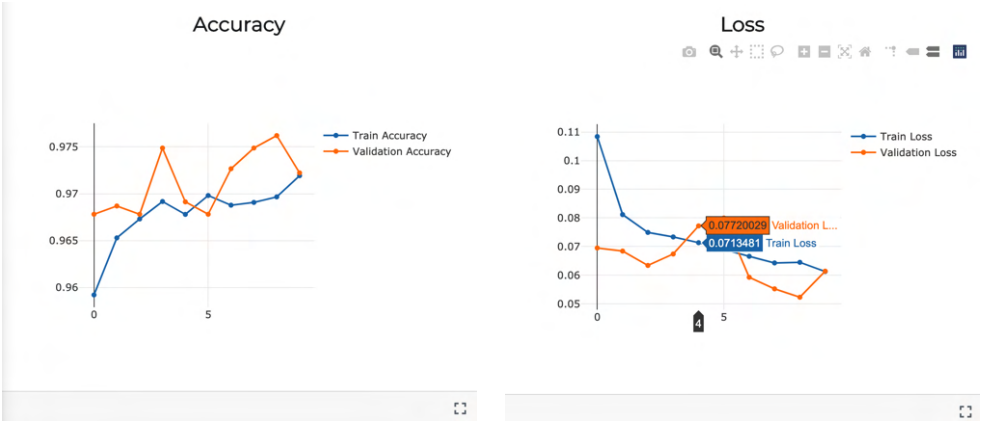
Algoritm	Train Accuracy	Test Accuracy	Train Precision	Test Precision	Train Recall	Test Recall	Train F1 Score	Test F1 Score	Train ROC	Test ROC
Random Forest	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Decision tree	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Gradient Boosting	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
KNN	0.990	0.981	0.990	0.981	0.990	0.981	0.990	0.981	0.990	0.980
Logistic regression	0.961	0.958	0.961	0.958	0.961	0.958	0.961	0.958	0.960	0.958
SGD	0.921	0.804	0.925	0.853	0.921	0.804	0.921	0.799	0.924	0.811
Gaussian Naive Bayes	0.890	0.901	0.890	0.903	0.890	0.901	0.890	0.901	0.889	0.902

Figure 6.8. Metrics of machine learning algorithms for training and validation sets

Using state of the art Deep learning

This is the main area of our research which is making use of state of the art deep learning,

- Once we select our train dataset from the list of datasets shown in Figure 6.2, we can train the model with deep learning model using just a button click
- For this research we mainly used LENET 5 and Alexnet model architectures as mentioned in 5.6
- Once we click on train button, the page loader is displayed for a while, once the training is completed we can see the training history as shown in Figure 6.10 so that we can understand how the accuracy and loss are changing for every epoch.
- User can also see the trend for Accuracy and Loss for both train and validation sets using the line charts shown in 6.9
- Finally, the metrics of our training such as Accuracy, Precision, Recall, F1-score and ROC score for both training and validation sets as shown in 6.11.
- We can change different hyper-parameters like number of ephocs, learning rate, batch size, activation and loss functions etc.., using the settings section shown in Figure 6.14



(a) *Accuracy* **(b)** *Loss*

Figure 6.9. Visualization of Accuracy and Loss

History

*Results of model training

epoch	acc	loss	val_acc	val_loss
0	0.959	0.108	0.968	0.069
1	0.965	0.081	0.969	0.068
2	0.967	0.075	0.968	0.063
3	0.969	0.073	0.975	0.067
4	0.968	0.071	0.969	0.077
5	0.970	0.069	0.968	0.080
6	0.969	0.067	0.973	0.059
7	0.969	0.064	0.975	0.055

Figure 6.10. History of deep learning model training

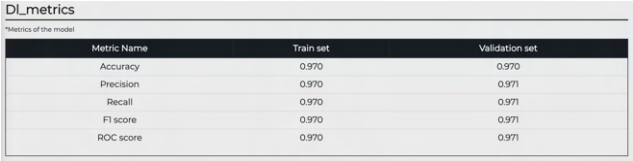


Figure 6.11. Metrics for deep learning model

6.6 Predict

Infer on new data easily using the trained model in previous step i.e.., do predictions on new data with just a button click, we use deep learning models used for model training for prediction. As shown in Table 6.12 we can easily identify the predictions done by our model on the new data and also get a summary of prediction on the new data as shown in Figure 6.13

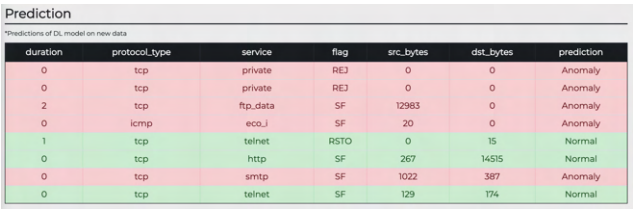


Figure 6.12. Prediction on new data

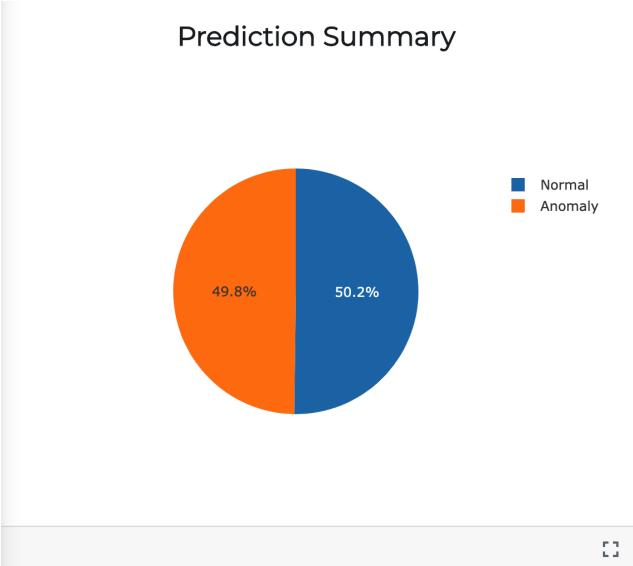


Figure 6.13. Prediction summary

6.7 Settings

Quick and easy machine learning configuration Configuring the host of parameters in the machine learning project is as simple as filling a form. Figure 6.14 shows

different sections we can configure. In particular user of the interface has the flexibility to configure important steps of entire machine learning workflow such as Pre-processing, GPU type, Model training, Metrics, Hyper-parameter tuning to name a few options that can be configured. Once the changes are saved they are retained for future logins depending on the logged in user. We can also change the critical aspects like Deep learning loss functions, optimization algorithms, deep learning architecture to use etc..,

Figure 6.14. Settings for our Machine learning project

6.8 Adding new data sets

To add a new dataset, you just have to upload the file and it will directly get saved in to MongoDB for further analysis as shown in Figure 6.15

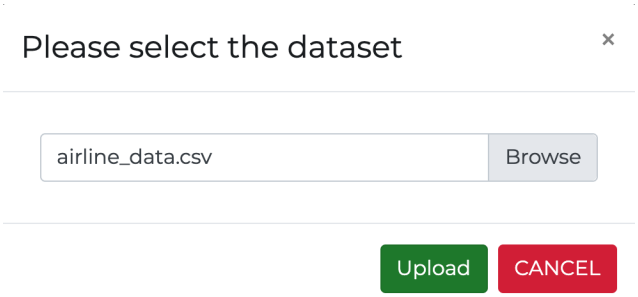


Figure 6.15. Adding new datasets

Chapter 7

Summary

In a nutshell these are some of the highlights of the research carried out so far,

- Simple and easy to use interface accessible from any device.
- Quickly get a glance of your datasets with dedicated sections for analysis
- Dedicated platform for the area of anomaly detection. so you get better in your decision making through the platform as the state of the art matures continuously with new incoming data
- No technological barrier in using the platform as we focussed on the usability aspect more than anything
- Gives you the power of deep learning with just a few button clicks
- Easily interpretable summaries and visualizations
- Fully automated platform with best practices such as Continuous integration and continuous deployment (CI/CD) and docker for seamless migration

Future work

Going forward we intend to customize ANOMA as a flexible, scalable and customizable Anomaly Detection platform so that it can be moulded according to specific customer needs. In a way, making it adaptable to any related business and industrial sector, from cyber security to predictive maintenance to the detection of fraud in the commercial and banking sector, just to mention some areas of future applications we are looking to target. At the moment, ANOMA implements and makes the most innovative Deep Learning algorithms both supervised and non-supervised easily configurable. This allows the user of the platform to process and analyze Big Data whether they are structured or unstructured data (documents, images, videos) from traditional data sources and business systems, from IOT sensors and from social media. ANOMA also provides powerful visualization and data exploration features that make the content of the data sets, the results produced by the artificial intelligence algorithms and the insights contained in their data easy to interpret, even for non-technical users. We also want to add the support for analyzing anomalies in real-time as well add support to multiple data connectors so that diverse data can easily be ingested in to platform.

Below are some key areas that we are targeting to adapt ANOMA in future,

- Data & document insight
- Predictive analytics
- Face recognition
- Video surveillance
- Satellite imagery
- Intrusion detection
- Malware detection
- Spyware detection
- User behaviour analysis

Bibliography

[1] Bahnsen Alejandro. Correa. *Building ai applications using deep learning*, 2016.

[2] Algorithmia. Introduction to Optimizers. https://algorithmia.com/blog/introduction-to-optimizers/, 2018.

[3] Ibrahim Almajai, Fei Yan, Teofilo de Campos, Aftab Khan, William Christmas, David Windridge, and Josef Kittler. Anomaly detection and knowledge transfer in automatic sports video annotation. In *Detection and identification of rare audiovisual cues*, pages 109–117. Springer, 2012.

[4] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[5] J Andrews, Thomas Tanay, Edward J Morton, and Lewis D Griffin. Transfer representation-learning for anomaly detection. JMLR, 2016.

[6] Akm Ashiquzzaman, Abdul Kawsar Tushar, Md Rashedul Islam, Dongkoo Shon, Kichang Im, Jeong-Ho Park, Dong-Sun Lim, and Jongmyon Kim. Reduction of overfitting in diabetes prediction using deep learning neural network. In *IT convergence and security 2017*, pages 35–43. Springer, 2018.

[7] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[8] Luis Basora, Xavier Olive, and Thomas Dubot. Recent advances in anomaly detection methods applied to aviation. *Aerospace*, 6(11):117, 2019.

[9] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[10] Sampada Bhosale. Network Intrusion Detection. https://www.kaggle.com/sampadab17/network-intrusion-detection/, 2018.

[11] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[12] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[13] Raymond Bond, Ansgar Koene, Alan Dix, Jennifer Boger, Maurice D Mulvenna, Mykola Galushka, Bethany Waterhouse Bradley, Fiona Browne, Hui Wang, and Alexander Wong. Democratisation of usable machine learning in computer vision. *arXiv preprint arXiv:1902.06804*, 2019.

[14] Jason Brownlee. Transfer Learning in Keras with Computer Vision Models. https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/, 2019.

[15] Andriy Burkov. The hundred page machine learning book. https://github.com/aburkov/theMLbook/, 2020.

[16] Saurabh Chakravarty, Raja Venkata Satya Phanindra Chava, and Edward A Fox. Dialog acts classification for question-answer corpora. In *ASAIL@ ICAIL*, 2019.

[17] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*, 2019.

[18] Jacinta Chan Phooi M'ng and Mohammadali Mehralizadeh. Forecasting east asian indices futures via a novel hybrid of wavelet-pca denoising and artificial neural network models. *PloS one*, 11(6):e0156338, 2016.

[19] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[20] Jay Chugh. Types of Machine Learning and Top 10 Algorithms Everyone Should Know. https://blogs.oracle.com/datascience/types-of-machine-learning-and-top-10-algorithms-everyone-should-know-v2/, 2018.

[21] Siddharth Das. CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more.... https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5/, 2017.

[22] François de La Bourdonnaye, Céline Teuliere, Thierry Chateau, and Jochen Triesch. Learning of binocular fixations using anomaly detection with deep reinforcement learning. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 760–767. IEEE, 2017.

[23] Valentina Djordjevic. ANOMALY DETECTION. https://thingsolver.com/anomaly-detection/, 2018.

[24] Gavin Edwards. Machine Learning | An Introduction. https://towardsdatascience.com/machine-learning-an-introduction-23b84d51e6d0/, 2018.

[25] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[26] David Emery and Rich Hilliard. Every architecture description needs a framework: Expressing architecture frameworks using iso/iec 42010. In *2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*, pages 31–40. IEEE, 2009.

[27] Ibraim Ganiev. Sklearn-pipelines. https://stackoverflow.com/questions/33091376/python-what-is-exactly-sklearn-pipeline-pipeline/, 2015.

[28] Elizabeth Gibney. Google ai algorithm masters ancient game of go. *Nature News*, 529(7587):445, 2016.

[29] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[30] Think Gradient. Automated Machine Learning — An Overview. https://medium.com/thinkgradient/automated-machine-learning-an-overview-5a3595d5c4b5/, 2019.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[32] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *arXiv preprint arXiv:1908.00709*, 2019.

[33] Code Heroku. Introduction to Exploratory Data Analysis (EDA). https://medium.com/code-heroku/introduction-to-exploratory-data-analysis-eda-c0257f888676/, 2019.

[34] Yu-Chi Ho and David L Pepyne. Simple explanation of the no-free-lunch theorem and its implications. *Journal of optimization theory and applications*, 115(3):549–570, 2002.

[35] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[36] Mohammad Hossin and MN Sulaiman. A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining & Knowledge Management Process*, 5(2):1, 2015.

[37] Chengqiang Huang, Yulei Wu, Yuan Zuo, Ke Pei, and Geyong Min. Towards experienced anomaly detector through reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[38] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[39] Alon Jacovi, Oren Sar Shalom, and Yoav Goldberg. Understanding convolutional neural networks for text classification. *arXiv preprint arXiv:1809.08037*, 2018.

[40] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956, 2019.

[41] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[42] Raimi Karim. Illustrated: 10 CNN Architectures. https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d/, 2019.

[43] Bekir Karlik and A Vehbi Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011.

[44] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, pages 1–62, 2019.

[45] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[47] PM Ashok Kumar and V Vaidehi. A transfer learning framework for traffic video using neuro-fuzzy approach. *Sādhanā*, 42(9):1431–1442, 2017.

[48] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[49] James Le. An Introduction to Big Data: Data Integration. https://medium.com/cracking-the-data-science-interview/an-introduction-to-big-data-data-integration-40715baa7961/, 2019.

[50] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[51] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[52] Kang Li, Nan Du, and Aidong Zhang. Detecting ecg abnormalities via transductive transfer learning. In *Proceedings of the ACM Conference on Bioinformatics, Computational Biology and Biomedicine*, pages 210–217, 2012.

[53] Peng Liang, Hai-Dong Yang, Wen-Si Chen, Si-Yuan Xiao, and Zhao-Ze Lan. Transfer learning for aluminium extrusion electricity consumption anomaly detection via deep neural networks. *International Journal of Computer Integrated Manufacturing*, 31(4-5):396–405, 2018.

[54] Jovian Lin. Data Visualization with Seaborn. https://jovianlin.io/data-visualization-seaborn-part-2/, 2018.

[55] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[56] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[57] Hariharan Manikandan. A Walk-through the Rapid Miner. https://medium.com/@vshshv3/a-walk-through-the-rapid-miner-921dfaf53722/, 2018.

[58] Wes McKinney. *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython.* " O'Reilly Media, Inc.", 2012.

[59] Yao Ming, Shaozu Cao, Ruixiang Zhang, Zhen Li, Yuanzhe Chen, Yangqiu Song, and Huamin Qu. Understanding hidden memories of recurrent neural networks. In *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 13–24. IEEE, 2017.

[60] Shritam Kumar Mund. Normalization vs. Standardization. https://medium.com/analytics-vidhya/normalization-vs-standardization-8937f45b3e20/, 2019.

[61] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[62] Artem Oppermann. Artificial Intelligence vs. Machine Learning vs. Deep Learning. https://towardsdatascience.com/artificial-intelligence-vs-machine-learning-vs-deep-learning-2210ba8cc4ac/, 2019.

[63] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[64] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[65] Yaroslav Dudin Piero Molino and Sai Sumanth Miryala. Introducing Ludwig, a Code-Free Deep Learning Toolbox. https://eng.uber.com/introducing-ludwig/, 2019.

[66] Marco AF Pimentel, David A Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, 2014.

[67] Data Professor. Data Science Infographic. https://github.com/dataprofessor/infographic/, 2020.

[68] Pranoy Radhakrishnan. What is Transfer Learning? https://towardsdatascience.com/what-is-transfer-learning-8b1a0fa42b4/, 2017.

[69] Maithra Raghu and Eric Schmidt. A survey of deep learning for scientific discovery. *arXiv preprint arXiv:2003.11755*, 2020.

[70] Hassan Ramchoun, Mohammed Amine Janati Idrissi, Youssef Ghanou, and Mohamed Ettaouil. Multilayer perceptron: Architecture optimization and training. *IJIMAI*, 4(1):26–30, 2016.

[71] RapidMiner. RapidMiner Auto Model. https://rapidminer.com/products/auto-model/, 2019.

[72] Russell Reed and Robert J MarksII. *Neural smithing: supervised learning in feedforward artificial neural networks.* Mit Press, 1999.

[73] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[74] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[75] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In *International conference on machine learning*, pages 4393–4402, 2018.

[76] Mahsa Salehi and Lida Rashidi. A survey on anomaly detection in evolving data: [with application to forest fire risk prediction]. *ACM SIGKDD Explorations Newsletter*, 20(1):13–23, 2018.

[77] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493, 2018.

[78] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[79] Sangmun Shin, Thanh-Tra Hoang, Tuan-Ho Le, and Moo-Yeon Lee. A new robust design method using neural network. *Journal of Nanoelectronics and Optoelectronics*, 11(1):68–78, 2016.

[80] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[81] Piotr Skalski. Preventing Deep Neural Network from Overfitting. https://towardsdatascience.com/preventing-deep-neural-network-from-overfitting-953458db800a/, 2018.

[82] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 847–855, New York, NY, USA, 2013. Association for Computing Machinery.

[83] Peng Wang, Jiaming Xu, Bo Xu, Chenglin Liu, Heng Zhang, Fangyuan Wang, and Hongwei Hao. Semantic clustering and convolutional neural network for short text categorization. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 352–357, 2015.

[84] Oo Yadanar and Khin Mar Soe. Optimizer comparison with dropout for neural sequence labeling in myanmar stemmer. In *2019 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*, pages 150–155. IEEE, 2019.

[85] Jing Yang and Guanci Yang. Modified convolutional neural network based on dropout and the stochastic gradient descent optimizer. *Algorithms*, 11(3):28, 2018.

[86] Andre Ye. 11 Essential Neural Network Architectures, Visualized Explained. https://towardsdatascience.com/11-essential-neural-network-architectures-visualized-explained-7fc7da3486d8/, 2020.

[87] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[88] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

List of Figures