

TESI DI LAUREA MAGISTRALE



Proactive customer care solution for telecommunication companies by exploiting Amazon Web Services

Relatore:

Prof.ssa Tania Cerquitelli

Correlatore:

Prof. Marco Brambilla

Tutor Aziendale:

Dott.ssa Cristina Bartoloni

Candidato:

Valerio Volpe

Matricola 251313

Summary

This work represents a project developed in collaboration with Accenture S.p.A. The activity, started with a curricular internship in February 2019, continued as an extracurricular experience until August 2019.

The idea is to combine machine learning techniques and Amazon Web Services, the cloud computing platform provided by Amazon, to tackle the proactive customer care problem for telecommunication companies.

The contributions of this thesis regard both the tool and the solution proposed. On one side it confirms that using Amazon cloud computing platform guarantees flexibility and scalability to the project and allows companies to save on operating costs, run the infrastructure more efficiently and resize resources based on business needs. Therefore, it could assist businesses in facing modern challenges.

On the other side it confirms machine learning capability to assist telecommunication companies in tackling new challenges. Indeed, current adopted solutions consist in a simple KPIs monitoring to identify and address near real time devices issues with algorithms able to trigger specific alarms when a fixed threshold is exceeded. This work wants to carry out activities in background, using analytic tools and machine learning techniques to identify factors which are causing current problems or which are likely to cause future problems. Thanks to very powerful classification methods, the idea is to create algorithms able to drive insights, understand correlations and recognize symptoms potentially leading to customer calls. Solution could bring benefits to Telco companies such as prevent trouble tickets opening, reduce the churn rate and improve the overall customer experience.

Solution is organized in few steps. In a first phase, data are extracted, cleaned and explored in order to extract only meaningful information. In a second phase, a feature selection is performed in order to improve learning performance and to guarantee a better model interpretability. Then a binary classification model is considered to predict if a problem will occur on a specific line, in terms of probability, and then a multiclass classification algorithm is trained to identify the specific type of issue. Moreover, the latter

is applied to an observation even if the former did not suggest any problem, since the scope is to always give an explanation of a trouble. The developed pipeline is able to detect issue event with a precision of more than 80% and to satisfactorily classify the different types of problem with a weighted precision of 80%. The model is kept up-to-date through a retraining phase and two different solutions are explored: the first one involves Apache Airflow platform, the other one is focus on the drift concept and it represents a cheaper but a bit more difficult approach, since it considers data distribution over the time.

Chapter 9 provides basis for a possible deployment into production, by identifying actors who are and will be involved in the future.

Acknowledgements

Il lavoro di tesi svolto rappresenta la conclusione di un'esperienza iniziata 5 anni fa, che mi ha portato a crescere e maturare moltissimo. Quando arrivai nella città di Torino a 19 anni e cominciai una vita indipendente dai miei genitori in una città a me allora sconosciuta, non poche erano le mie paure e i miei dubbi. Con gli anni si sono trasformati in forza e consapevolezza, anche se so che ho ancora molto da imparare.

In primo luogo vorrei ringraziare la professoressa Cerquitelli, relatrice universitaria, che è stata la persona che mi ha fatto appassionare per la prima volta a tematiche di data analytics, e il professor Brambilla Marco, correlatore del Politecnico di Milano, che ha accettato di supportarmi nel mio percorso di tesi. I loro consigli e suggerimenti sono stati essenziali per la realizzazione di questo lavoro.

Riconoscenza va poi ad Accenture e in particolare al team che mi ha accolto e mi ha offerto la possibilità di imparare molto e di affacciarmi per la prima volta al mondo del lavoro. Ringrazio tutti i membri del team che nel massimo delle loro possibilità mi hanno dedicato parte del loro tempo che so essere molto prezioso. Non si sono mai rifiutati di aiutarmi, suggerirmi e spiegarmi e in questo devo ringraziare, in particolar modo, Alberto e Alessandro, i due "massimi esperti di tutto", come mi piaceva chiamarli.

Il mio percorso accademico non si è limitato a seguire 5 anni di corsi, ma mi ha portato a fare moltissime esperienze diverse, anche all'estero. Devo ringraziare la mia famiglia che in tutti questi spostamenti ha sempre rappresentato il mio punto di forza nonostante tutte le difficoltà. Mi hanno sempre incoraggiato e spinto a non fermarmi mai e a cogliere tutte le opportunità. É anche grazie al loro costante supporto che ho scelto di iniziare a studiare al Politecnico di Torino. Vorrei anche ringraziare mio nonno per i suoi semplici ma importantissimi consigli.

Un altro grande ringraziamento va a Carola che in questi 5 anni mi ha sempre assistito in ogni mia scelta e mi ha spinto a non perdere nessuna occasione. É anche grazie a lei che ho intrapreso alcuni percorsi che si sono rivelati poi molto utili per la mia crescita personale. Anche la sua famiglia mi ha sempre

accolto quasi come un figlio fin dall'inizio rappresentando una sicurezza nei miei primi periodi a Torino.

Poi ringrazio i miei amici storici, Gian e Lollo, che ci sono sempre stati fin dall'asilo, e quelli un po' meno storici Ale, Suren, Matte, Teo, Raffa che nonostante la lontananza so che ci saranno sempre. Anche se ormai ci vediamo non più di una volta al mese il nostro legame rimane molto forte.

Inoltre ringrazio i ragazzi del Collegio Einaudi che mi hanno regalato un quarto anno splendido, fatto di risate, cene, pranzi e feste tutti insieme. Sono stati momenti molto belli e unici sotto molti aspetti. Siete tantissimi e anche se non vi nomino singolarmente sappiate che passare solamente un anno con voi è stato un grande dispiacere.

Ringrazio gli amici di Torino con cui ho esplorato la città, in particolare Andre, Angy e Julien, e la compagnia della Svizzera, Diego, Ste, Giorgia, Giulio e Leo con i quali ho condiviso le mie due esperienze a Losanna. Mi ricordo le super carbonare organizzate all'ultimo e i pranzi in caffetteria che hanno decisamente alleggerito la pesantezza del semestre.

Ringrazio il team "Abajour" dell'ASP con cui ho condiviso le interminabili ore di lavoro nelle 4 settimane organizzate, ma soprattutto le super mangiate con la mitica sfida del double lunch.

Infine ringrazio il Politecnico per tutte le opportunità che mi ha dato e tutte le persone che mi ha permesso di conoscere. Confesso che all'inizio non ero sicuro fosse la scelta giusta, ma tornando indietro ripeterei l'esperienza 1000 volte.

Ultima, ma non ultima, ringrazio la città di Torino di cui mi sono innamorato e che ha rappresentato la mia seconda casa. Torino mi ha colpito perchè sa essere grande città senza fartelo pesare troppo con le corse al parco del Valentino, la Mole Antonelliana che si erge sulla città, Superga che la guarda dall'alto come a volerla proteggere e molti altri scorci unici che questa città è in grado di regalare. Occuperà sempre uno spazio nei miei ricordi più belli.

Contents

Summary	1
Acknowledgements	3
List of Figures	7
List of Tables	9
1 A new industrial revolution	10
1.1 Industry 4.0	10
1.2 Big data	13
1.3 Cloud computing	16
2 Amazon Web Services: a new cloud computing frontier	20
2.1 Discovering the tool	20
2.2 The competitors	22
2.3 The infrastructure	24
2.4 AWS Glue	27
2.5 AWS SageMaker	28
3 Introduction to proactive care	34
3.1 Digital scenario	34
3.2 Business case	36
3.3 Use cases and architecture	39
4 Algorithms background	43
4.1 Decision trees	43
4.2 Random Forest	48
4.3 Extreme Gradient Boosting	50

5	Implementation: dataset presentation and exploration	57
5.1	Data extraction	57
5.2	Data preparation	58
5.3	Model pipeline	62
5.4	Features selection	65
6	Implementation: algorithms development	70
6.1	First step: binary classification	70
6.2	Second step: multiclass classification	81
7	Experiments	83
7.1	Binary classification	83
7.2	Multiclass classification	89
7.3	Results	91
8	Model retraining	95
8.1	Apache Airflow	95
8.2	Drift evaluation	102
8.3	Retraining based on drift	104
9	Model architecture and deployment	107
10	Conclusions	112

List of Figures

1.1	Big data: example of data available on the internet.	14
2.1	Magic quadrant for cloud infrastructure as a service.	21
2.2	Amazon SageMaker basic architecture.	32
3.1	Proactive care and homes' devices interaction.	38
3.2	WAN issue detection.	39
3.3	WLAN issue detection.	40
3.4	CPE status.	40
4.1	Decision Tree: default stopping criterion (top) and more additional conditions to stop (bottom).	46
4.2	Bias and Variance concepts graphical representation	47
4.3	Underfitting and overfitting concepts graphical representation	48
4.4	Bias and Variance trade-off	48
4.5	Example of tree ensemble model. Final output is obtained by summing the score in each tree	51
4.6	Tree structure with default directions.	55
5.1	Creation of the target variable.	59
5.2	Plot of an example of aggregate KPI distribution.	59
5.3	Pie chart representing a ticket arguments distribution over 3 months.	60
5.4	Stacked bar chart representing ticket arguments distribution over 3 months.	61
5.5	Comparison between train and test size.	61
5.6	Compute mean function to the original data distribution. . . .	62
5.7	Concept of irrelevant, redundant, and noisy features. (a) Relevant feature. (b) Irrelevant feature. (c) Redundant feature. (d) Noisy feature.	66
5.8	Random forest features importance for the binary model. . . .	68
5.9	Correlation between target variable and some numerical features.	69

5.10	Correlation between target variable and some categorical features.	69
6.1	AWS Glue Preprocessing: example of running script	72
6.2	AWS Glue Preprocessing: example of failed script	72
6.3	AWS Glue Preprocessing: log inspection	74
6.4	AWS Glue Preprocessing: how to monitor the status	74
6.5	SageMaker: visualize hyperparameter tuning job results	77
6.6	SageMaker: inspection pipeline model configuration	80
6.7	SageMaker: visualize endpoint configuration and performances	80
7.1	Binary classification: extraction of decision tree tuning job. . .	85
7.2	Binary classification: extraction of random forest tuning job. .	87
7.3	Binary classification: extraction of XGBoost tuning job. . . .	88
7.4	Binary classification: trade off between positive class precision and recall in the general call recognition model.	90
7.5	Multiclass classification: extraction of random forest tuning job.	91
7.6	Multiclass classification: extraction of XGBoost tuning job. . .	91
7.7	Multiclass classification: confusion matrix, on the left normalized by class total predictions, on the right not normalized. . .	92
8.1	Airflow list of DAGs	98
8.2	Airflow tree visualization	98
8.3	Airflow graph visualization	99
8.4	Airflow variable view	99
8.5	Airflow Gantt chart	100
8.6	Airflow task duration	100
8.7	Airflow code view	101
8.8	Airflow task instance menu	101
8.9	Airflow email alert	102
8.10	Covariate drift with values computed hourly for the drift of 7 hours before the current time.	105
9.1	Architecture of the final binary and multiclass classification solution	107
9.2	New observation prediction	111

List of Tables

5.1	Call/ No Call binary prediction model.	64
5.2	Issues multiclass prediction model.	65
6.1	Spark configuration.	71
7.1	AWS Glue Preprocessing and Data Preparation: summarize the computational parameters.	83
7.2	Binary classification: example of decision tree tuning job. . . .	86
7.3	Binary classification: example of random forest tuning job. . .	87
7.4	Binary classification: example of XGBoost tuning job.	89
7.5	Binary classification: summarize the training details.	90
7.6	Multiclass classification: summarize the training details. . . .	92
7.7	AWS Model Creation: summarize the computational param- eters to create the pipeline model from the trained algorithms.	93
7.8	AWS Prediction: summarize the endpoint creation and batch transform job computational parameters.	93
7.9	Cost analysis: summarize the fixed cost details.	93
7.10	Cost analysis: summarize the costs to keep the endpoint ready for real time inference.	94
8.1	Cost analysis: summarize the costs to keep the periodic re- training active.	102
8.2	Cost analysis: summary of the model retraining costs.	102
8.3	Drift analysis: comparison between solution with and without drift consideration for 1 month time window.	106

Chapter 1

A new industrial revolution

In recent years information and communication technology (**ICT**) sector has emerged as one of the fastest growing area on the world stage. It had undoubted positive effect on the industrial productivity: the production costs were drastically reduced, efficient client-focused solutions were adopted to serve customers with quality and speed. All these changes need a new industrial model which integrates, in addition to technological development, also the increasing demand for customized solutions: this new model is called **Industry 4.0**.

1.1 Industry 4.0

This term was used for the first time in Germany in 2011, more specifically during the Hannover Messe, one of the world's largest trade fairs. During the event a working group announced a project for the development of the German manufacturing sector, the "Zukunftsprojekt Industrie 4.0", which should have brought the country's industry back to a leading role in the world, and the German government inserted it in the wider High-Tech Strategy 2020 Action Plan [44]. Subsequently, this model inspired numerous European initiatives and the term Industry 4.0 spread internationally [29].

Impacting on a lot of different production sectors, it combines state-of-the-art technologies with the web to obtain more adaptable and cooperative solutions. Santos, Charna-Santos and Lima (2018) defines the interaction between production structures and intelligent devices (able to connect to the network), as the future *intelligent factories* which will have flexibility to tackle current companies challenges with customized solutions and reduced *product life cycles* [18].

Term 4.0 refers to a further step in improving production techniques after the three equally significant past revolutions (leading to the current state), each linked to the introduction of a specific technology [9] :

- Industry 1.0 in the 18th century, characterized by the introduction of production machinery able to leverage the water and steam power; machines took away the dirty, dangerous work with a positive effect on the production, which becomes faster.
- Industry 2.0 in the 19th century, characterized by the introduction of mass production and new work techniques such as labour division; in parallel electrical energy and oil were introduced in the industry. Again, some dangerous parts of the work were moved from human to machine.
- Industry 3.0 in the 20th century, characterized by the further automate production thanks to the introduction of electronic and IT systems. Machines took away the dull work with an industrialization of services and of clerical work.
- Industry 4.0 is the one in progress, characterized by interconnected products and processes through the use of cyber-physical systems, such as advanced software, cloud, robotics etc. Machines take away decisions since they are able to make faster and more reliable choices than humans.

This new revolution shows some innovative high-tech proposals, such as [15]:

- **internet of things**, which allows each product to exchange data with internet network. Information could be captured and channelled where more value could be produced, while smart connected products enable new industrial solutions thanks to a strictly collaboration with the physical world around them. All these steps require new challenges in designing products and services, in order to make possible to collect large quantity of knowledge from different sources and near real-time data;
- **cloud computing**, which provides a more flexible and accessible way for companies and people to exploit storage capacity, processing power, software applications and scalability;
- **big data**, which represent the large amounts of data available today and the basis for a new learning frontier which helps decision-makers and operations optimization. The key point is the large quantity of application domains, from the predictive maintenance to customer profiling to image recognition;

- **artificial intelligence**, which represents the intelligence demonstrated by machines. Indeed, based on the analysis of collected data, automatic algorithms could generate, by themselves, rules which programmers cannot specify;
- **automation robotics**, since collaborative robots could emulate humans behaviours in factories, working alongside them. They play an important role in a lot of applications such as assembly, healthcare, logistics and monitoring;
- **additive manufacturing**, by exploiting the technology's exceptional skill to be not sensitive to quantity and complexity, it gives company an advantage in terms of production volume, time and costs.

Industry 4.0 represents the mass customization, the increased speed, the better quality, and the improved productivity in manufacturing which society needs to be prepared to meet the challenges of globalization. In this way there could be the possibility to elaborate individual projects giving targeted solutions to the acquired customers and allowing companies to be competitive on the global market. World's industrial economy to face this exceptional challenge requires a new production model, which for instance, means a continuous monitoring of activities (thanks to intelligent devices) to find out immediately a possible failure.

Santos, Charna-Santos and Lima (2018) [18] report some outcomes of Industry 4.0's revolution, summarizing its main aspects:

- **mass customization**, defines in [50] as the process of *delivering wide-market goods and services* created as close as possible to the needs of the customers. It is a *marketing and manufacturing technique* which exploits the capability of being extremely flexible, ad hoc designed for clients and associated with *low unit costs*;
- **greater flexibility**, making easier for companies to react to changes in the global market more rapidly;
- **quality control**, defines in [51] as the process used by a business in order to improve product quality and to reduce possible troubles;
- new innovative business models and services significantly change the **value chain interaction** which describes the full range of activities needed to create a product;

- **optimized decision making**, thanks to “smart” products and devices able to connect to the network and to update their status, it is possible to find out almost in real time what the sensors reveal, and analyse the data to monitor the situation.

Finally, in this innovative context the strict collaboration between industries and service providers initiated a new type of sale: sell the solution rather than the product. The difference in some cases could be very difficult to find but a simple example is reported here to help reader understand which could be a possible explanation. Consider a client who provides truck rental services to industrial and construction customers; in this case it is not selling just a “rental truck”, but a solution supporting companies to face their problems related to goods transport [8].

1.2 Big data

The vast majority of people use smartphone every day to surf the internet and spend time on social networks. And every time it happens, a large amount of data is generated. Large companies such as Google, Amazon or Netflix, for example, use data collected to satisfy customers’ needs, sell products or make suggestions. Every minute a large quantity of information is produced, at a rate which was unthinkable only few years ago. All these data taken individually are meaningless and generate only confusion, but if they are organized and divided according to specific rules, they show an incredible power. In this context people start talking about **Big Data** [31].

Taylor-Sakyi (2016) [23] argues that the foundation of big data is based on three interacting concepts:

- **technology**, combining optimal usage of computational resources and really accurate algorithms to analyse and combine large data sets;
- **analysis**, the opportunity to use network and the Internet of Things for Big Data generation, noticing patterns in the data and focusing on the most important ones;
- **mythology**, the idea that a large quantity of data generates a knowledge which helps companies to draw conclusions.

Big Data refers not only to a collection of data which is huge in size and grows exponentially with time, but as Baralis (2015) correctly underlines, it is much more: it refers to data with a different *scale and complexity* which

lead to develop new *architectures, techniques, algorithms and analytics* to integrate them and extract their hidden value [1]. Therefore, traditional data management tools are not suitable for an efficient storage and process of these structured, **semi-structured** and **unstructured** data, which oblige to develop new data analysis techniques and algorithms in order to exploit their potentialities in the most effective manner. Indeed, big data purpose is to analyse the huge amount of data available to extract information in a reasonable time and with limited resources.

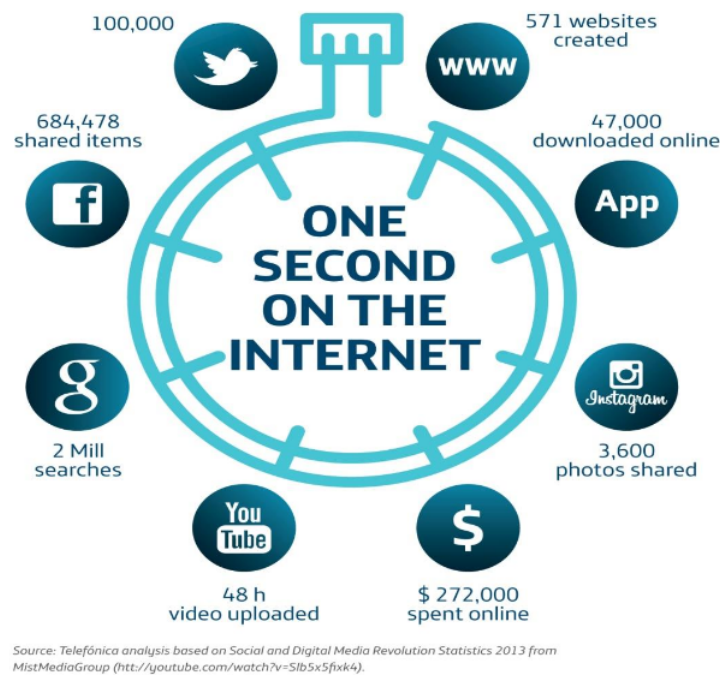


Figure 1.1: Big data: example of data available on the internet.

More specifically, big data could be characterized using 5Vs [13]:

- **volume**: in the world the amount of data generated doubles every 12-18 months, moving probably from petabytes (10^{15} byte) to zettabytes (10^{21} byte) in the coming years and making the volume a critical factor in big data analytics (Figure 1.1).
- **velocity**: data must be managed, processed and analysed faster. A huge amount of information is obtained, even in real time due to the increasing internet speed, and it needs to be quickly analysed.

- **variety**: there are different forms of data, such as texts, numbers, maps, audio, video, e-mails and so on. The majority are unstructured, opposed to the traditional databases organized in diagrams and rigid tables.
- **value**: the main purpose of the data analysis is to derive business value.
- **veracity**: refers to the quality of the data intended as correctness and reliability. For instance, not all online contents are reliable or data could contain noise and abnormalities.

The big data analysis and its subsequent extrapolation of hidden information are called **big data analytics**. This is carried out in a lot of sectors, such as business one, where companies change their operational approach showing an interest in analysing the purchasing behaviour of consumers, in studying marketing campaigns, in improving customer satisfaction and so on: they are all systems which use knowledge obtained from big data to increase companies' profit margins. On the other side, all these positive aspects required users to tackle the 5Vs challenges explained before. For this reason businesses and researchers who increase their attention on big data, abandon the old analytical techniques.

A common traditional model to store sets of data which could be queried for other applications is the **relational database management systems** (RDBMS) [45], a collection of data, containing meaningful information, which is managed by a database management system (DBMS), i.e. a software system able to work with collection of large, persistent and structured data which guarantees integrity and security [46]. RDBMS is a type of DBMS based on a table structure with a unique identifier and with functions able to keep data privacy, accuracy, reliability and consistency [17]. In this case information returned to decision makers are obtained combining two or more database tables, for instance with a join condition. However, this approach does not suit the industry 4.0 necessity as big data analytics does. The incredible amount of unstructured data coming from mobile devices (social media), Web and Internet of Things combined with the necessity of making information almost instantly available obliges companies to change analytical processes moving to big data analytics, suitable for current business requirements [23].

In order to help reader better understand which could be the big data potentiality, some relevant examples are presented below.

United Parcel Service (UPS) [23], the American multinational package delivery and supply chain management company, is one of the pioneer in exploiting big data analytics power. In 2009 they started a project of monitoring their trucks technical information such as speed, location and direction, by installing 46 000 sensors able to gather data and transmit them to a data center where they are analysed each night. A combination of fuel-efficiency sensors and GPS data led company to a significant reduction in fuel consumption and routes duration.

American retailer Target Corporation [6], one of the largest retailers in the United States, wanted to figure out if a specific customer was a pregnant woman. Indeed, the big problem they faced was that the majority of consumers usually bought different items in specific stores and they went to Target retailers only when they were looking for generic products (new socks or six-month supply of toilet paper). However, Target offered also many other items from milk to electronics and for this reason one of the marketers' aims was to convince shoppers to buy also other Target's products. Since it resulted to be hard change consumers' shopping habits, the marketing department idea was to focus on one of the life period when routines fall apart: *around the birth of a child*. The main problem was related to time: since birth records were public, families almost instantaneously received a lot of offers and advertisements from a lot of companies. To beat the competition and gain new customers, the idea was to reach women with specially designed ads in their second trimester, when most future mothers start acquiring whatever they need, like prenatal vitamins and maternity clothing. For this reason Andrew Pole [23], a Target data analyst, created a *pregnancy-predictive* model, where each customer is labelled with a *pregnancy prediction* score. In particular, Pole and his team assign a unique id number to each shopper and, collecting its purchased products, previous payments methods (such as credit cards, cash etc.) and interaction (such as clicking links in sent e-mails etc.), they managed to perform a proper set of analysis to find a pattern useful for the company. The proposed model allowed marketers to come up with appropriate proposals (for instance sending SMS messages with links taking customers to a mobile coupon), which represented an important competitive advantage for the first years of model application to customers.

1.3 Cloud computing

Cloud computing is one of the today's world emerging technologies, which becomes a crucial element in a lot of companies' current innovation strategies to optimize technology usage and to improve business effectiveness. Adopt-

tion of this new tool is motivated by market requirements, where a level of flexibility is required in order to be able to react quickly to changes in business demands, and it allows companies to abandon traditional methods and to be more competitive. Cloud computing has registered an increased and always positive interest on the market, motivated by the some unique properties it could guarantee: **flexibility**, **costs reduction**, **agility** and **scalability** [24].

As Ting Si Xue and Tiong Wee Xin (2016) [24] underline, there is no an unique cloud computing concept definition, but in their study they report two possible descriptions: the first one defines cloud as *virtualized computer resources*, while the second one defines it as a pool to store computational resources.

Reader should imagine cloud as four layers infrastructure, which represents an end to end service, from cloud computing providers until the final user who exploits its advantages for the specific task. In particular these layers are:

- **fabric layer**, which is composed by provider's physical resources;
- **unified resource** layer, where physical devices are been combined together to operate as *virtualised resources* for the users;
- **platform layer** to reduce the burden of deploying applications directly into virtual machine container;
- **application layer**, which represents the environment where all the applications live and can be executed to complete a task. The great advantage is that no particular operational ability or device are required to interact with the applications, whose availability is always guaranteed (unless technical issues). Users have also the possibility to deploy directly to the cloud without having to tackle resources or system issues, since providers automatically manage and control these crucial technical elements.

Cloud computing interacts with final user by providing him three main types of services [19]:

- **Software as a Service (SaaS)**: it represents cloud computing service offering the software owned and promoted by providers which individuals or companies could rent and access via Internet. Differently from the past, where people needed to install software on their devices with high costs, SaaS providers run it on their data centres using their resources and then they rent it or sell a type of subscription. The main

point is that SaaS license is less expensive than a permanent software license and furthermore SaaS allows users to have a constantly updated version of the software. For the reasons before, customers are increasingly convinced to adopt this solution which guarantees less costs and better information migration. IT department of a company could really appreciate the possibility to satisfy organization's business goals without having troubles with resources maintenance or updates. Some examples of Software as a Service are Google Docs and Microsoft Office 365.

- **Platform as a Service (PaaS):** it represents cloud computing service offering a development environment to individuals or companies to create and keep their applications. For instance, users could rent cloud computing infrastructure or buy a type of subscription for their applications. On the other side PaaS providers could propose a low cost and customized computing based solutions, avoiding individuals to face with problems such as number of processing units or quantity of memory required for storage. Some examples of Platform as a Service are Amazon Web Services, Google App Engine. The difference between SaaS and PaaS is that on the former users could only host the completed applications, while on the latter users could also develop step by step the applications.
- **Infrastructure as a Service (IaaS):** it represents cloud computing service offering the infrastructure to run the applications, for instance a virtual machine or an operating system. The great advantage comes in the capability of the providers to allocate resources quickly according to customer's requirements, guaranteeing an on-demand scalability of the infrastructure. Basically, the users task is to configure and use the provided infrastructure, while the vendors task is to focus on its availability and performance. An examples of Infrastructure as a Service is Amazon *EC2*.

The increasing adoption of the cloud solution is strongly motivated by its unique benefits, which allowed it to gain a relevant role in keeping companies competitive on the market [47]:

- **flexibility:** employees both in and out of office could access files simply connecting to it with a web-enabled device such as smartphones or notebooks;
- **scalability:** users could easily upscale or downscale resources on request;

- **cost reduction:** companies could purchase only needed resources, paying only the ones exploited;
- **automatic software/hardware upgrades:** cloud vendors manage required software and hardware maintenance, such as security updates, allowing to keep employees, IT staff, and resources free for other tasks.

On the other side cloud computing brings also some challenges which must be taken into account [24]:

- **data stealing:** in parallel with the increasing number of Internet users, also number of *probing attacks* increases. International Data Corporation (IDC) in one of its surveys underlines that security is the most difficult and important challenge in cloud;
- **data privacy:** it is crucial to understand how to protect the privacy of personal and clients' data.

Moreover, Ting Si Xue and Tiong Wee Xin (2016) [24] in their article also classify cloud in four possible categories, each one offering a different solution to users. Each type has unique properties in terms of adaptability to customers' needs:

- **public cloud service:** with a medium level of efficiency and costs it gives access to anyone provided with an Internet connection and, for this reason, some of the crucial problems of this type of cloud are related to security and privacy. Photo, email and storage services are examples of public cloud;
- **private cloud service:** less risky and more secure and reliable type of cloud, thanks to a check on trusted identity. It is used by companies to store and manage sensitive data;
- **community cloud service:** similar to a private cloud but with lower setup costs since resources and also costs are shared between members of the organization. Clearly, each user of the community could access to the information stored. An examples of this type of cloud is the university cloud shared with all the students for research intent;
- **hybrid cloud service:** combination of two or more previous type of cloud services. The incredible advantage is that it keeps the benefits of both private and public cloud, such as a good security level, since it is possible to store sensitive data in a private area. Hybrid clouds are for instance the ones used for backup purposes.

Chapter 2

Amazon Web Services: a new cloud computing frontier

Amazon Web Services, also called **AWS**, [14] is a platform offering a class of cloud computing services, launched for the first time in 2006 by Amazon.com to manage its online retail operations. It could be considered as a pioneer in the *pay-as-you-go* cloud computing field, built and designed to satisfy the main needs of companies which show an interest in innovating and reinforcing their IT department and equipment.

2.1 Discovering the tool

AWS [14] represents an offer able to include each type of cloud solution (IaaS, PaaS and SaaS) and to solve the most challenging issues. AWS environment seems to have no boundaries, with new functionalities and products added continuously and highly scalable, fast, reliable and accessible services through a common interface: the same Amazon company included those services in its infrastructure to better manage the website. Jeff Bar, Amazon president and CEO, said: “*one thing that I love is the customer-driven innovation cycle*” [42] and this philosophy is probably the main AWS characteristic since its origin.

Surely one of the best web services on the market, analyst firm Gartner in the 2019 edition of the annual *Infrastructure-as-a-service magic quadrant* confirms its superiority with respect to competitors. In particular, the choice is based on the execution ability and completeness of vision and AWS results to be the highest in both criteria [25]. Results are reported in Figure 2.1.

Since the main Amazon goal is to become even more customer-oriented, inside

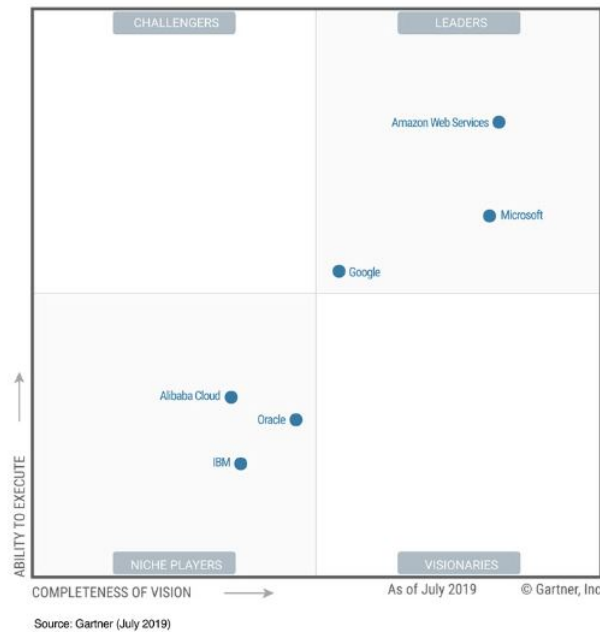


Figure 2.1: Magic quadrant for cloud infrastructure as a service.

its cloud platform users can build any type of infrastructure without worrying about technical and economic constraints and feeling safety about data loss. Its intuitive usage, with the possibility to manage all services through a user-friendly interface, also accessible via mobile, results to be very helpful for developers. The really immediate console allows users to easily monitor all the resources and it simplifies the solution design in the cloud by quickly identifying the required resources to realize it. Online tutorials and courses are freely available on the web platform to make users more confident about Amazon services.

As explained before AWS is a pay-as-you-go service, in the sense that it provides on-demand resources and user pays only for ones he uses. However, a full cost analysis results to be quite complex due to the fact costs strongly depend on the characteristics of the resources used and they are generally weighted on the hourly usage of physical resources. In order to give reader a general idea, as an example, one hour of Amazon **EC2** service (better explained later) costs in dollars approximately 2.5 cents for “Linux small” *EC2* instance and 3.4 cents for Windows “small” equivalents. Price doubles for Linux “medium” *EC2* instance, moves to 5 cents per hour, and increases for Windows “medium”, to 6.8 cents per hour. It quadruples for “large” (10 and 12 cents), until reaching Linux \$5.136 and Windows \$9.552 per hour for the

most powerful *EC2* instance [20].

The model-types defined before (small, medium, large and so on) refer to a well-defined range of available resources, whose distinction is based on:

- number of **virtual CPUs**, where a virtual CPU represents a portion or share of a physical CPU that is assigned to a virtual machine (VM). The CPU, central processing unit, is the operating heart of any device (PC, laptop, smartphone); also called as “processor” or “microprocessor”, it is the part of the device which coordinates the activity of the other processing units;
- amount of available **RAM**, where the random-access memory (RAM) is a memory space where the device can store and retrieve data in a very short time;
- available **disk space** for storage, i.e. the amount of disk space inside the physical host dedicated to the specific instance to run.

The base price option (the account default one) includes a fixed number of free hours of usage and bandwidth occupation. Any additional use, above the established threshold is paid as an extra. For this reason it is good to immediately frame project’s needs, in order to understand required resources and to avoid extra-costs. Due to the wide range of AWS products, it is impossible to analyse all costs; however, the idea is quite the same for all the others: there exists one or more usage models to be chosen carefully according to the fundamental needs since any memory, cpu or disk excess is paid extra based on a well-defined price list.

2.2 The competitors

Talking about cloud computing and Amazon Web Services the comparison with the two other big competitors, **Microsoft Azure** and **Google Cloud Platform** (GCP), comes naturally. In 2006 Amazon launched AWS, the first cloud computing platform in the market. Ten years later, it was able to offer 70 different types of services in 14 distinct regions [32]. Four years later (2010) Microsoft launched Windows Azure (the name was changed in Microsoft Azure). The Redmond Corporation’s platform in 2016 had a rich offer of 67 services in 30 distinct regions [33]. Finally, in 2011, Google decided to launch Cloud Platform (GCP), an infrastructure designed to originally support YouTube and the well-known search engine. GCP in 2016 offered 50 services and 6 data centers globally [34]. Nowadays, AWS is the company which provides the broader set of services, reaching 140 different

solutions. Regarding the Machine Learning field, which is the most relevant in this study, AWS offers solutions for database, machine learning, robotics, storage, cost management and so on [35]. According to the official documentation, machine learning available services are **SageMaker**, to build, train and deploy models at scale, *Comprehend* to investigate insights and relationships in texts, *Forecast* to increase forecast accuracy using machine learning, *Rekognition* to analyze image and video, *Polly* to turn text into lifelike speech, a lot of *Deep Learning* services and so on.

On the other side, Google offers its Cloud Platform, which allows users to create machine learning models considering different areas. For instance, **Google Cloud Machine Learning Engine** helps users to simplify the creation of models and it is completely integrated with other Google platform products. Thanks to the tools provided, including the **Google Cloud APIs**, it is possible to investigate different solutions such as *Video Analysis* to analyze videos, *Image Analysis*, to classify images into different categories and to detect objects or faces, *Voice Recognition* to convert audio into text, and so on [16].

Finally Microsoft Azure proposes **Azure Machine Learning Studio** (or Azure ML) to orchestrate models testing and training and their execution on Azure. Equipped with a set of algorithms which cover the main domains of machine learning (classification, clustering, regression), Azure ML can also host third-party modules [30].

Clearly, it is not so easy to choose between the three competitors because choice depends mainly on company priorities. For this reason, only a brief overview of positive and negative aspects is reported:

- The huge amount of services made available by Amazon and the extremely enterprise-friendly visualization it offers to users represent probably company strongest advantages. Carey, S. (2019) [4] underlines also its *openness and flexibility* with an incredible partner ecosystem. For instance, the main public transport company in British capital has exploited AWS potentialities to manage the peak of users for its online official website. On the other side, a negative aspect could be its enormous list of offers. Carey also argues that even if it is true that it could represent an attractive aspect for companies, it results to be hard to have a full view of the entire set of services, making AWS out to be a difficult vendor to manage.
- Microsoft Azure has the great advantage that Microsoft services are already strongly integrated in a lot of companies infrastructures, making easier to help them in adopting cloud computing. On the other

side, in recent years he had to face some failures making the service unavailable for users to the point that Leong L., Gartner analyst, [7] recommends assuming *disaster recovery capabilities away from Azure*, especially regarding the crucial applications (even if also AWS had similar problem in 2017 with *S3* service). Moreover, clients usually complain about Azure technical support and the available documentation. Finally, from partners' point of view Microsoft platform results to be more restrictive than the Amazon one.

- Google has a good level of experience with cloud-native industries and it covers a key role in the open-source world, but it had a lot of problems in emerging due to the presence of leader companies Microsoft Azure and Amazon. It adopted a go-to-market strategy, taking control of small and strategic projects but without ever representing a cloud partner. It still needs to increase its partnerships, supporting for instance new activities and processes, if it wants to become more competitive with the other two giants. This seems to be the direction taken by new Google Cloud CEO Thomas Kurian which wants to spend a lot of efforts in its machine learning tools, combined both company's experience in artificial intelligent and the popular framework TensorFlow [4].

To summarize, Carey concludes that AWS is still at the top regarding number of functionalities offered and level of the service. The incredible number of tools, combined with the user-friendly approach adopted, makes AWS the favourite cloud solution for the big companies. Moreover, its infrastructure grows continuously providing economies of scale which manage to reduce costs. Meanwhile, Microsoft tries to exploit companies which already adopted its solutions in terms of technologies, to reduce the gap with Amazon: for this reason it is investing a lot in the Azure service. Finally, Google, which does not actually represent a valid alternative, thanks to the Kurian's policy in the future could be the number one AWS competitor.

2.3 The infrastructure

As already mentioned before, Amazon AWS is a container of heterogeneous products. The incredible number of services made available by Amazon and the innovation that this multinational company is able to offer every month, make impossible to treat all tools in a properly detailed way. Thus, in the following, it was decided to present an intelligent overview of the services, with a brief description for each of them, which tries to give a logical structure in presenting them. A block diagram representation can be very useful

for understanding how different services are structured and how they can interact each other.

The AWS **basic infrastructure** [36], which represents the first diagram block, is divided into regions, availability zones and edge locations. It is important for the reader to understand the difference, because this information allows reader to realize how Amazon manages to provide reliability against servers or data centers failures, without interrupting the offer. **Regions** are independent sets of resources, geographically isolated one from the others and spread around the world and they guarantee the correct level of privacy and compliance (user must notice that not all services are always available in all regions). The number of available regions is 8 plus 1 dedicated only to government agencies, and user must choose one of them, usually the closest to him (for instance Ireland in case of an Italian user). Furthermore, resources price could change from one place to another.

Inside a region, there are the *Availability Zones*, which are independent areas not sharing any failure points, connected each other through a low latency network. They can be considered as backup systems, able to manage faults. Indeed, users could develop their applications on different Zones and in this way if one of them has a failure, applications remain online.

Finally, *Edge Locations* are supporting services to distribute content and applications to national users with reduced latency.

Above the basic infrastructure, as second block, there are **networking services** [14], such as **Amazon Virtual Private Cloud (VPC)**, which helps company to expand the IT infrastructure without worrying about computational and application issues. Amazon VPC can easily run services on Amazon's virtual servers, which are accessible only by network and company accounts.

Above the networking services, there are three building blocks, namely the ones related to computation, database and storage.

Computation includes: [37]

- **Amazon Elastic Compute Cloud (EC2)**, considered as AWS's heart, it is the Web service designed to provide computing and data processing capabilities as virtual servers called instances, in a scalable cloud environment. It is suitable for developers or companies which do not or cannot get the physical infrastructure required to their own application or cannot be able to manage sudden computational loads. In this way, they do not have to buy physical resources with large costs, but they

only have to interact with an user-friendly interface, allowing them to quickly increase or decrease the required distributed computational capacity with new cloud servers. It is an elastic service, in the sense that it guarantees an instantaneous scalability and a pay-as-you-go model where only used resources are paid. The elasticity allows users to select more than one hundred virtual servers simultaneously and to configure also the most suitable hardware, selecting the number of virtual CPU, RAM capacity, storage size.

- **Amazon Auto Scaling**, to automatically configure the scalability of EC2 solution. Monitor specific metrics published on Cloud Watch AWS service it is possible to change number of cloud servers, in order to keep high performances. There exists also the possibility to define a series of alarms which trigger automatic resources scaling.
- **Lambda**, to run code without dealing with servers managing. Payment depends only on the amount of time code runs and no extra costs are charged for its maintenance. User has only to upload the code and Lambda takes care of recovering required resources. It is also possible to automatically trigger this function directly from a web or mobile application or by other services such as Amazon S3. In particular, some characteristics relevant for this study are:
 - the possibility to trigger a Lambda when a file is uploaded to one Amazon container or modified.
 - the possibility to use Lambda to transform data and save the result in a data repository.
 - the possibility to configure particular expressions to run a Lambda function at a certain time or at regular intervals.

The base price option (the default one) includes 1 million free requests per month and the price for the next 1 million is \$0.20.

Storage includes: [38]

- **Amazon Simple Storage Service (S3)**, the AWS storage service designed to guarantee reliability and durability. Data are organized in an unlimited number of objects, each one no more than 5 TB. Those ones are organized in larger containers, called **buckets**. The information contained in a bucket can be accessed wherever and whenever

user wants, and each element could be encrypted with either *Amazon S3-managed keys* or *AWS KMS-managed keys*. Accesses could be monitored by setting a series of permissions and each operation on objects or buckets is recorded in specific logs and notifications are sent for the most important events.

Database [39] includes a set of solutions which could be defined as examples of DaaS, Database as a Service. Using this approach, users can avoid problems linked to databases configuration and systems management:

- **Amazon DynamoDB**, born from Amazon e-commerce experience in working with NoSQL databases. It is able to overcome classical relational limitations and to avoid dimensional obstacles. To ensure data reliability, replication on different Availability Zones is performed to manage possible faults.

Above the presented services, as a fourth block, there are all the other products. For this study the most relevant are analytics (in particular AWS Glue Service) and machine learning (in particular AWS Amazon SageMaker)

2.4 AWS Glue

AWS Glue is a serverless, fully managed and cloud optimized service to extract, transform and load data (or ETL). It allows user to organize, locate, move and transform all datasets across the business to put them to use. One of the hardest parts in an analytics or data warehousing project is setting up and maintaining a reliable ETL process. Glue helps users to better understand the data, it suggests a possible transformation and it generates an ETL code, avoiding any waste of time in hand coding the solution. It is provided with a flexible scheduler which can run the job on a scale out smart platform, automatically provisioning resources required to complete it. Glue includes a data catalog, which is basically a central metadata repository, an ETL engine, which can auto generate Python code and a flexible scheduler, which handles job monitoring and retries. Another important point is the ability to automate heavy lifting involved with discovering, categorizing, cleaning, enriching and moving data: less heavy lifting means more time for analysis. Glue makes this step in a simple, cost-effective, but also reliable way.

Glue is different from other ETL products in three important ways. First, Glue automatically discovers data and files, determines the schema and builds centralized metadata catalog for later querying and analysis. It provides an

automatic schema inference for semi-structured and structured data and an out-of-the-box integration with a lot of other AWS services. Second, Glue generates an ETL code, to perform data extraction, transformation and loading, which is just Python language and it is entirely customizable, reusable and portable. Indeed, it can be edit using any integrated development environment or notebook and share with other users using Github. Third, it is serverless, in the sense that there are no resources to manage and user only pays for the ones the jobs consume to run. User does not have to configure servers or manage their life cycle.

Focusing on how Glue suggests transformations and generates ETL code to convert data, after logging into the Glue management console, user can automatically generate a Scala or Python script based on the typology of the source and target data. Glue proposes a starting script (default option) but gives also the possibility to import an existing pyspark script. To allow the service to create a default solution, user must pick data source, stored in a S3 bucket, and then configure data target. In order to set up it, he can select an existing table in the Glue data catalog or he can ask to the service to create a new one by specifying S3 target location and output format. Then, it is possible to specify column mapping from source to target (default is copy). In this way the job is created and a script is proposed with the corresponding diagram to help visualize it. After its creation, user can run it, by optionally pass runtime parameters and the service automatically provides resources required to complete it. Finally, user can inspect logs or schema table and, once the job is completed, also the statistics on rows could be read and written. AWS Glue makes also easy to attach triggers to start job on a schedule or on completion of other activities or to invoke it on-demand from other services like AWS Lambda.

2.5 AWS SageMaker

In the 4.0 industry context, a large volume of discussion is around artificial intelligence and machine learning. Main themes are related to:

- image recognition, to identify the subject in a photo;
- object detection, tracking and navigation for autonomous vehicles;
- speech recognition, which is developed for solutions like Amazon Alexa;
- algorithmic trading strategy performance improvement;

- sentiment analysis, useful for targeted advertisements.

Amazon **SageMaker** is an AWS managed machine learning solution which allows users to define, train and deploy machine learning and deep learning models to tackle previous presented challenges, removing most of the heavy lifting on which for years data scientists spent a lot of time. Its main console is divided into four parts which represent SageMaker features: the first consists in managing **Jupyter Notebooks** which are open-source web applications to write Python or other programming languages code and providing a lot of examples to look at. The second one consists in setting up and managing training clusters, in the sense that user can spin up and run instances to **train** machine learning algorithms with one click or one line of code (inside a notebook), by simply calling the SageMaker API and its fit method. This process spins up a number of servers (decided by the user), trains the machine learning model and stores the results. At the end it shuts the server down and user only pays per second the execution time. Then, once the job is ended the corresponding **model** can be created and **host** in SageMaker (third step) and in the fourth step it can be deployed behind an **endpoint** to make future predictions. The real-time endpoint is auto scalable and it allows monitoring and debugging using other AWS services such as CloudWatch. Indeed, it produces both metrics and logs, making users able to monitor situation inside the instances that host the API endpoint. It could be either privately accessible within VPC, Amazon Virtual Private Cloud, or publicly hosted, giving the flexibility in terms of how API endpoint has to be deployed but still keeping the benefits of auto-scaling and monitoring.

All these four steps should be combined and linked together to obtain an end to end solution go from beginning to production, but they can also be used independently. For instance, user could only use the notebook part if the scope is just data discovering or he could use the deployment part if he comes with a pre-trained model and the only need is to host and deploy it. Indeed, SageMaker gives the option of using or not using any of its features and for instance if user is not interesting in using SageMaker notebooks he can still take advantage of the model training and endpoint deployment. The possibility to pick and choose the favourite solution for the specific work-flow comes from the fact that this service is not a work-flow for machine learning but only a set of tools which allow users to create their pipeline for the specific job.

Focusing on notebook instances, to create a new element it is enough to give it a name and select one of the available instance types (associated to a

number of virtual CPU, amount of RAM and available disk space for storage). Some examples are the *t2 medium* for small datasets and *p2* which is equipped with a GPU and it guarantees more power for larger and more demanding datasets. Once created, familiar Jupyter environment and plenty of examples written by AWS experts are available. It is also essential to remember that data must be always stored in a *S3* bucket, so before using a notebook user must be sure to have a *S3* bucket which lives in the same region as the notebook instance.

As explained before working with SageMaker means not building everything and for instance user may benefit from built-in SageMaker algorithms. Indeed, AWS data scientists provide an implementation of them, leaving user only the task of bringing its own data and use them to train and deploy the model. 17 different algorithms are provided, which include:

- K-means clustering, unsupervised algorithm to identify a fixed number of clusters of different dimension within data;
- PCA or principal component analysis, unsupervised algorithm to reduce the dimensionality (number of features) within a dataset, but still keeping relevant information;
- Amazon SageMaker NTM (Neural Topic Model), unsupervised learning algorithm to organize a corpus of documents into topics;
- Linear Learner, supervised learning algorithms to solve either classification or regression problems performing both linear regression and logistic regression;
- XGBoost (open-source implementation of the gradient boosted trees), supervised learning algorithm to accurately predict a target variable by combining simpler and weaker models;
- Image classification, supervised learning algorithm to detect subjects or to identify objects inside an image;
- DeepAR, forecasting algorithm for one-dimensional time series which exploits neural networks. Predictions are based on combination of several time series, because the idea is that when they move together algorithm can learn across them, trying to better predict next hours, days or weeks situation;
- K-nearest neighbors, index-based algorithm, to identify similarities between items. It classifies observations potentially very similar, tagging them with the same identifier.

If none of the previous solutions works for the considered use case, one option is that user implements its own algorithm and it can still be supported inside SageMaker as long as specific conventions are followed. It consists in creating a docker file and a source code, and insert all into a docker repository provided with SageMaker access. In this way, the AWS service can execute docker container, fire the code and train the model without user having to manage all the heavy lifting.

Third option is to use SageMaker frameworks, including TensorFlow or PyTorch or Scikit Learn. This option helps users in training algorithms proposed in the previous libraries and not already built-in. In other words even if it is possible to use a Scikit Learn algorithm as an “own algorithm”, SageMaker framework simplifies the training, since user does not have to write its own docker file, but he has to simply specify the scikit file and follow a specific convention for how the file will be retrieved by SageMaker.

There is also the possibility to integrate spark workloads into the pipeline as a part of an overall ETL process; in this way user can gain the benefits of an existing spark pipeline (for instance the one generated by AWS Glue) and SageMaker at the same time.

Next aspect is the one-click training mentioned earlier. It is one click or one API call to the SageMaker interface that executes the training for a particular machine learning model. The very important point of this step is SageMaker’s ability to perform an automatic hyper-parameter tuning.

Final step consists in deploying the model into production with a fully managed hosting provided with the auto scaling option. User has only to specify minimum and maximum number of instances and their types and then service, based on those instructions, hosts the model and gives metrics and logs. An important SageMaker feature is the possibility to build multiple models in the same endpoint, allowing to shift traffic from one model to another or to compare their performances.

Giving a more general context, as explained before SageMaker notebook offers the possibility to analyse raw data and to create prepared data sets that can be used for the initial training. However, this is not the best solution, because after having identified a proper preparation routine, user should go back and start an ETL process using other tools like AWS Glue. On the other side, during the exploration phases where the main point consists in trying to identify what is relevant to the business case or to the machine learning problem, usually a lot of iterations are performed, looking maybe at smaller subsets of data: in Figure 2.2 it is possible to see this raw data pre-analysis; indeed, SageMaker is more designed for ad hoc analysis and data preparation for an initial test to investigate if the training is going to

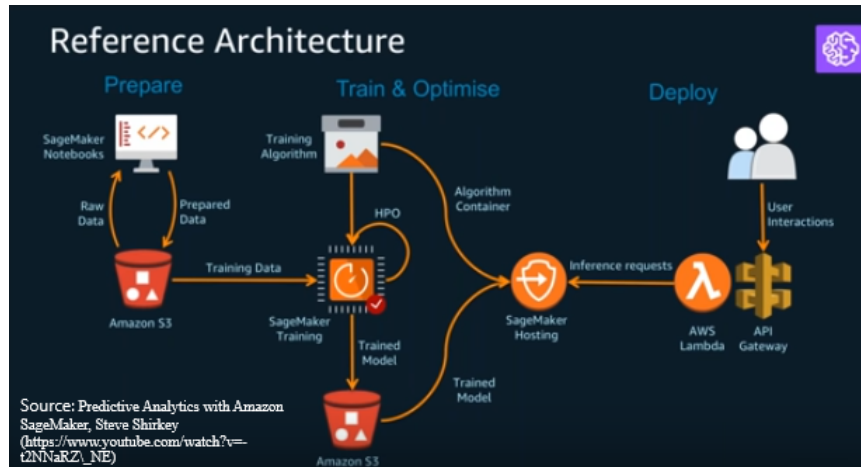


Figure 2.2: Amazon SageMaker basic architecture.

be successful or not. Once user identified a production state which is going to use the same features for millions of rows, using other tools such as AWS Glue is better.

Another relevant point which needs a deeper presentation comes from the fact that the training phase does not happen only once but the necessity of a parameters optimization makes it an iterative process. The hyperparameters optimization, HPO in Figure 2.2, represents an essential part in every machine learning workload. SageMaker offers different possibilities to tune the algorithm parameters. The first approach SageMaker gives user the possibility to follow is a **full grid search** which means that each possible combination is inspected, becoming very computational expensive. With only two different parameters there is a loop inside of a loop, and if number of parameters increases a lot this approach will become not feasible. For this reason it turns out that rather than doing a strict grid search according to different documents and articles [3] a better solution could be a **random search**, where the system randomly guesses the following parameters combination. Finally there is also a third process where each choice is learned from the past: the idea is that at each time and at each step the training system can learn from the parameters chosen and try to make a better guess based on previous experience. Inside SageMaker there exists an implementation of the **Bayesian optimization** strategy which allows to realize the process. To perform the hyperparameters tuning user has to specify a range of values he wants to test and the chosen method to decide the next parameters configuration (such as Bayesian optimization strategy) and the system at each iteration chooses values within those ranges. User can also

customize the regular expression used to scan the logs and to identify the metrics used to compare the solutions obtained. Finally it is possible to set also the number of total and parallel jobs being run. Moreover, from the hyperparameter console it is possible to see the best training job and the corresponding parameters value. For data scientist this is a relief because it provides an automated mechanism which takes away a lot of the heavy lifting that comes from the optimal parameters research.

The final relevant point is about endpoints. When the training is run user only pays per second for the training job execution but when an endpoint is running user pays for the possibility to make real-time predictions, i.e. there is a maintenance cost. For this reason a good choice if user does not want to make prediction on a regular minute-by-minute basis, is a batch transformation job with the same model. By choosing number and type of instances and giving to it a set of input data, the system will run predictions and store the output in *S3*.

Chapter 3

Introduction to proactive care

The context of **predictive maintenance** becomes a priority for a lot of technological companies. This is the case of phones service providers, which need to constantly monitor fixed networks. Proactive care allows firms to anticipate possible issues and to reduce maintenance costs and churn rate.

3.1 Digital scenario

For years, the approach adopted by telecommunications companies (Telco) to address and solve network devices problems encountered in their customers' homes has been a **reactive care** one. An effort to solve a certain issue took place only after it occurred, i.e. when a consumer explained the problem to the customer care service, which took care of capturing the message and activating the resolution actions. The analytics contribution was already incorporated into a reactive care solution, but its adoption was restricted to detail more the analysis of both problems and related responses, trying to provide a better user experience in case of a future similar contact. Basically, the enormous amount of data generated was used only when there was an active interaction between the customer and the telecommunication company. However, with the possibility of collecting more and more technical data both from the network and from the devices in the customers' homes, many Telco companies are shifting their interests towards an approach where data analytics plays a major role in solving a problem. As well as a more data driven concept, new technology trends should completely transform the way Telco design, size and operate moving to a better **customer centrity** approach (customer segmentation with differentiated value propositions and quality of service perceived) moving from a traditional bottom-up service monitoring approach to a digital view of the service. Accenture Global Consumer Pulse

Research (2017) reports some critical points about customers loyalty, showing the extremely crucial role of the “quality” concept. Basing on the tested population:

- 64% of surveyed globally switched providers (referred to all industries) due to poor customer service.
- up to +80% switching could be avoided through a better service.
- 2 out of 3 of consumers extremely frustrated when a company delivers something different than they promise up front.

Accenture Global Consumer Pulse Research confirms quality of the offered services is more significant than price, indeed:

- more than 60% customers are not able to sacrifice quality for low price. In fact, some of them are even willing to pay for them.
- more than 50% customers consider quality as one of the most important criteria when conducting business with companies.

Machine learning and **artificial intelligence** could help developing a digital transformation, addressing a 100% of the customer lifetime value, by providing both a methodology for dealing with the complexity of digital business scenarios and working with large and constantly updated datasets due to the introduction of new market factors.

This research could explain why Telco’s interests move from a reactive care approach to a **proactive care** one, making possible to use information gained to identify and address near real time devices issues. The customer could be alerted that a certain problem has been fixed, or the latter may be fixed through background processes without involving him. This task could be performed by monitoring the status using deterministic KPIs and algorithms able to trigger specific alarms when a fixed threshold is exceeded (**digital monitoring**). However, the aim of this collaboration is to make one step more reaching a state of **digital prediction**: activities are carried out in background, using analytic tools and machine learning algorithms to identify factors which are causing current problems or which are likely to cause future problems. Thanks to very powerful regression and classification methods, the idea is to create algorithms able to drive insights, understand correlations and recognize symptoms (trends) potentially leading to customer calls. The difference between prediction and deterministic monitoring is that the former based its approach on finding path leading to a status deterioration, while the latter hypothesizes that if some monitored KPIs exceeded a fixed threshold

a deterioration is happening; however, in both cases the customer should not perceive the problem. Finally, preventive actions to mitigate impacts on the end user could be performed, i.e. before problem occurs. Basically, a series of checks is performed in background, using analytics to anticipate potential problems and to prevent consumers to contact customer care service.

3.2 Business case

The first step aims to identify the project's stakeholders inside the client's company:

1. **Customer care support:** business dictionary defines the *customer support* as set of services help *customers in making cost effective*, but also *correct use of a product* [48]. Therefore, customer care service focuses his activity on consumer's experience, without any technical knowledge, by using a reactive care approach to support him. Try to understand customer's observations and give him an instantaneous feedback is the basis of a valuable customer care assistance and a good service ensures a customer comfort feeling, in the sense that capital and time spent in a product or in a service have been fully repaid. The developing tool could be helpful for this figure, both reducing the number of consumer complaints by dealing with a problem before it arose and providing the operator with a tool able to give a possible issue explanation (reactive approach), making customer feel more satisfied. Moreover, the average handling time and the repeated call rate could be reduce. To increase end user's feeling of comfort, it appears to be essential to provide him a sense of global understanding where the company he chose is able to understand problems occurring.
2. **Technical customer support:** business dictionary defines the *technical support* as *user-friendly assistance* for customers which have *technical problems with electronic devices* [49]. Therefore, the technical support service focuses his activity on solving a technical problem or incident in the quickest and most convenient way possible. It is essential for the technical support operator to determine what did not (will not) work properly and to restore the initial situation as soon as possible: if the customer does not (or will not) contact customer support again, the technical activity is considered to be successfully completed. The developing tool could help the technical figure in identifying the potential future problem by allowing him to provide a quick solution to the problem (proactive approach).

3. **Assurance team:** client's assurance engineering team could be considered as the main driver of this approach, since they show an interest in supervising the customer care service, trying to improve user experience: they commissioned this tool, they are the ones who will check solution's performance and they could be considered as promoters of this new type of solution (proactive one). The developing tool could be helpful for this figure to better understand the customer care world as a means of maximizing user's satisfaction.

Before presenting the proposed solution, it could be useful to focus deeply on how data are integrated: a proactive monitoring of the line implies retrieving a large set of parameters from **customers' devices** and homes (customers' CPEs) and enriched it with other source of data coming from network systems. In telecommunications field, a customer-premises equipment or customer-provided equipment, **CPE**, refers to *devices, such as telephones, routers* and in general to any *terminal equipment located in user's premises*, which allow customers to *access communications service providers' services* [52]. Data are elaborated to obtain quality performance indicators, **QPI**, and key performance indicators, **KPI**, which could be compared with meaningful thresholds, allowing users to constantly monitor network performances and proactively identify troubles on the line (related to CPE health, WAN issue, WLAN issue). When an indicator exceeds a cut-off level, an alarm is captured and the customer care operator can proactively perform a network optimization action to avoid customer complaint and do not let him notice the situation. Moving from a proactive monitor to a predictive monitor means that the large set of parameters retrieved before is used to train predictive models. Once the algorithm is trained, it can be run over a new sample of preprocessed data to label each customer's line with a **risk score**, reporting the probability of observing in the considered line a particular type of issue (e.g. unstable line, slow connection): this approach makes easier to identify customer at highest risk of issue (for a specific issue category). The interaction is reported in Figure 3.1.

The first crucial step consists in defining datasets by collecting and analyzing data through analytics and machine learning from various sources: those information should make data analyst capable to identify the possible issue that caused the customer claim. Some macro issue categories are identified to classify processed data according to the use cases which will be defined in the next section and then create predictive models to identify customers at highest risk. In the future it could be interesting mapping each identified macro issue category with a list of **actions**, based on past similar experiences and customer care service's capabilities, which could be performed to

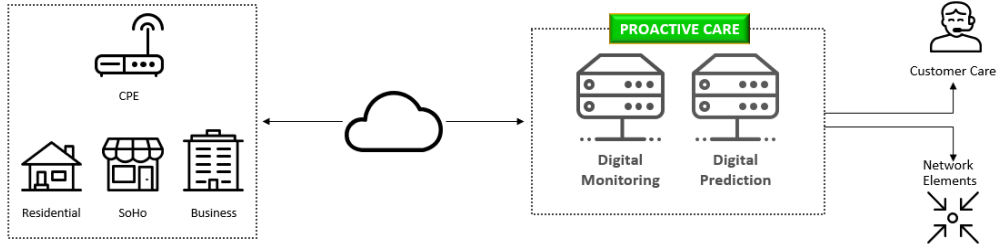


Figure 3.1: Proactive care and homes' devices interaction.

mitigate or fix the issue. Each use cases will be characterized by different and specific performance and quality indicators, allowing to optimize algorithm results (risk scores) by considering a target group and a control group. Starting from exploiting the large quantity of data available and discovering huge data potential, to monitor service quality as perceived by end user and to support customer care operators moving from manual troubleshooting to an automatic recovery process, the idea is to identify customers at highest risk of issue in near real time and to prevent future calls and trouble ticket openings. The highlighted approach could hold some benefits for Telco companies:

- face proactively customer claims;
- prevent trouble tickets opening;
- reduce **operational expenses** or OpEx and the **average handle time** or AHT, which represents the average duration of a single transaction, generally computed from the customer's call to complain until the end of the tasks performed to solve that issue;
- reduce the **churn rate**, i.e. the number of individuals who abandon the service in a given period;
- build long-term relationships with customers;
- improve first call resolution and the overall customer experience.

A cooperative bottom up collaboration is adopted with iterative cycles, strictly linked to the client's needs, to give more flexibility to project changes. For this reason the project team performs a series of cycles where each iteration is evaluated and required changes are determined to better adapt solution to customer's expectations. Effectiveness of the system will be evaluated by

picking a representative customers sample, splitting it in two subgroups and comparing a chosen indicator (e.g. tickets per customer) between them over a predefined period of time: in particular, some of them will compose the control group, where they will receive no suggestions, the others will compose the target group, where they will be exposed to ticket opening probability. It is required that both control and target group should be and remain statistically coherent (achieved through a stratified segmentation) where the only difference between the two groups should be the actions applied, avoiding other external interactions.

3.3 Use cases and architecture

Focusing deeply on the use cases implemented, the figures below show schematically the interaction between Telco and customer domains. On the left there is a telecommunication tower sending a signal to customers' homes: a direct collaboration with the client allows to define three possible use cases:

- **WAN issue detection** (Figure 3.2); the wide area network (WAN) consists of a set of connected computer over a large area [54]. Combining data and parameters from customers' CPEs and network systems with the related customer calls and trouble tickets history, the aim is to proactively identify issues on the WAN. Indeed, thanks to a continuous evaluation of connection performances, the idea is to predict some common issues on the network area, such as **unstable line** or **slow connection**. When a specific problem is identified, or likely to occur, based on predictive algorithms, recovery actions could be performed to avoid customers complains.

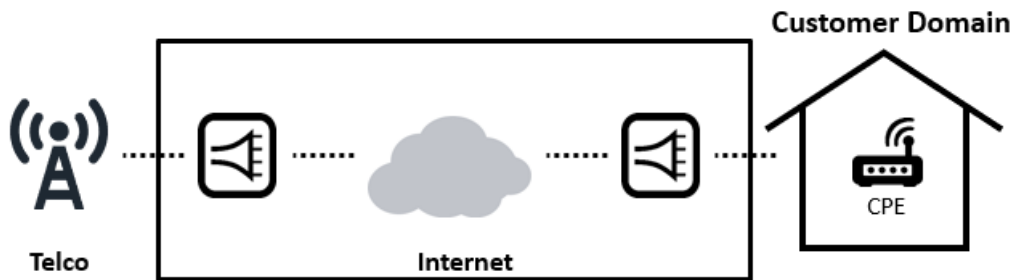


Figure 3.2: WAN issue detection.

- **WLAN issue detection** (Figure 3.3); the wireless LAN (WLAN) allows to connect some devices to create a *local area network (LAN)*

inside a small zone, such as a home or a school [55]. Parameters are retrieved from the customers' CPEs, gathering information on customers wireless local network and service configurations to identify possible issues, such as problems with **Wi-Fi connection**. Once a specific problem is identified or likely to occur, based on predictive algorithms, recovery actions could be performed to avoid customers complains and to optimize local network performances.

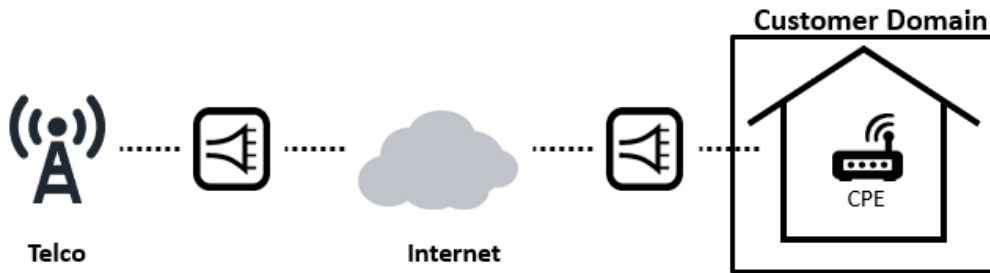


Figure 3.3: WLAN issue detection.

- **CPE status** (Figure 3.4); more technical parameters are retrieved from the CPE to analyse the device health status. Again, once a specific problem is identified or likely to occur, based on predictive algorithms, recovery actions could be performed to avoid customers complains.

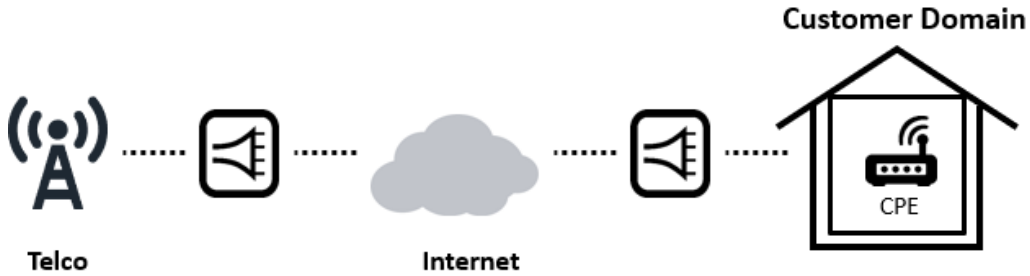


Figure 3.4: CPE status.

Therefore, while digital monitoring means to detect one of the previous faults in **near real time** to avoid customers dissatisfaction, digital prediction means to detect the problem **before** it happens to prevent effects on the customers (which implies no customers complains). In digital monitoring alarm is raised in near real time as soon as the KPIs register the issue, in digital prediction alarm is raised as soon as the probability, computed using predictive algorithms, of a particular issue to happen in the next chosen

hours, exceeds a predefined threshold.

Before moving to the analytics part it could be interesting to focus on the proactive care architecture; gathering data directly from the customers' homes is at the same time very powerful, offering insights and information on how the quality of the service is perceived from the customers, but also complex and difficult to obtain. For this reason, this project required a strict collaboration with another internal team, the operations support system team. They developed a tool, CEM Platform (Customer Experience Management), already tested and adopted by the client, which is able to collect data and integrate information from heterogeneous data to monitor customers' devices parameters. In particular they collect some parameters such as:

- **reason** for the complaint: which contains the reason motivated customer calls to the operator;
- **ticket troubles** : which contains the information about a ticket opened by a customer care agent;
- **CPE data** : which registers the technical parameters obtained from the customers' home devices and representing the largest amount of data;
- **disconnections** : which contains the number of daily CPE disconnections. This information became available only later on, so it is retrieved separately from the others;

Data collected can be divided into two different parts, depending on the different original source: front-end cpe data , coming for instance from modems and wireless routers and back end data, such as network element data, network inventory data, customer calls and ticket troubles. In telecommunication, the front can be considered as a device or service, i.e. the user visible part allows interaction (user interface), while the back is the infrastructure that supports provision of service, i.e. the part which allows the actual functioning of these interactions. Data are collected in two ways: via real-time data flows (for data coming from the CPEs and network) and via batch ingestions (for more static data, such as ticketing information) and stored in a Hadoop file system. A service performs an initial mapping and valid and invalid raw data and mapped data are stored in a data lake. Finally, trouble ticket data and network status at the time of ticket opening are collected and stored for model training; after that machine learning models are developed in Python using machine learning libraries provided by Spark ML and other sources (this part will be examined in detail in next sections).

Furthermore, the models are maintained by an automated pipeline, which is used to retrieve and transform fresh data. Models are able to produce a score for each input sample, meaning that when a customer calls to highlight that a problem arises, the operator has the possibility to visualize all his details: indeed, the Accenture operation support system already developed a dashboard where data retrieved from consumers' devices and network are graphically visualized. The model scoring is added to the previous dashboard, in the sense that when an operator receives a call, he could see the current situation combined with the model prediction and he could make an inference on the occurred problem. Results are also stored in a repository, where a csv file contains all the model scores, allowing technical support system to operate some background corrective actions.

Chapter 4

Algorithms background

Before moving to the training step, this chapter proposes a brief introduction on the algorithms which will be used to generate the predictive model. Thanks to this presentation, reader will be able to understand the design choice made in this project: after some initial tests, only tree algorithms appeared to have enough good results.

4.1 Decision trees

The main idea behind these approaches consists in dividing the *predictor space* (which is the set of distinct values the features can assume) into distinct sub-regions and once a new observation is retrieved, it is assigned to a specific region and the predictive value is simply the mean or the median of the training samples belonging to the considered zone. Regions are created by defining splitting rules applied to the original *predictor space* and they can be easily summarized in a tree, for this reason those algorithms are called **decision tree**. They are very interpretable and simple, but unfortunately they do not show competitive results, for this reason usually more trees are combined together to produce one single prediction, making the solution less interpretable but more accurate: different aggregation type generates different models, in particular two will be inspected in next steps **random forest** (maybe the most popular one) and **gradient boosting**. Both regression and classification tasks could be solved by trees, but since the main project's goal is the predictive maintenance, this section focuses only on **classification trees** where the prediction is a qualitative variable. Two are the main steps to build a tree [11]:

- divide the predictor space into K non-overlapping and distinct sub-regions called R_1, R_2, \dots, R_K ;

- once a specific sample is assigned to a certain region, the most common training set class, contained in it, is considered the new observation's label;

To better understand the second step move to a practical example: suppose after the splitting phase two regions are created, R_1 and R_2 , and that the most common class in R_1 is *call*, while the most common class in R_2 is *no – call* event. This means that if a new sample x is assigned to R_1 *call* event will be predicted, if x is assigned to R_2 *no – call* event will be predicted. Focusing now on the regions creation, to divide the initial space only *high-dimensional rectangles* are used, since they are extremely easy to interpret. Basically, during the training phase, a set of partition is considered: each one selects a specific feature and moves samples with that feature less than a threshold to the left branch and the others to the right branch. Then for each branch a new split could be considered and other two branches could be created and so on until no more steps are required (it depends on the algorithm configuration) [11]. Regions obtained once a branch is no more separated are called *terminal nodes* or *leaves* of the tree; clearly trees are created in a *upside down* approach, putting leaves at the bottom of the tree and configuring a structure of internal points, called *internal nodes*, and lines connecting them, called *branches*. The high-dimensional rectangles for a classification task are created to minimize the *classification error rate*, which is the fraction of the training set does not belong to the most popular class:

$$E = 1 - \max_k(\hat{p}_{mk}) \quad (4.1)$$

where \hat{p}_{mk} is the ratio of training set assigned to the m -th region and belonging to the k -th class. This metric is not able to manage over-fitting related to tree growing. For this reason two other metrics are also defined: the first one is the **Gini index**,

$$G = \sum_{k=1}^K \hat{p}_{mk}(1 - \hat{p}_{mk}) \quad (4.2)$$

which measures the *total variance across classes*. It represents the *node purity* and it takes small values when the considered node contains majority of samples belonging to a *single class*, i.e. when \hat{p}_{mk} is close to 1 or 0. The other considered metric is **cross-entropy**,

$$D = - \sum_{k=1}^K \hat{p}_{mk} \log(\hat{p}_{mk}) \quad (4.3)$$

Again this metric is related to node purity, in the sense that it takes small values when \hat{p}_{mk} is close to 1 or 0, and it is *numerically similar* to gini index. Obviously, since it is impossible to consider each single partition to find the best possible tree due to the limited computational resources, a *top-down greedy approach* is adopted where *recursive binary splits* are performed. It is a top-down approach since the algorithm starts from the tree root and ends when leaf nodes are created and it is a greedy approach since a specific division is made to minimize a metric but without looking at the previous partition, i.e. without computing a global minimization. Only binary splits are considered, each time a single feature X_j and a cut-point s are chosen such that the two subsequent regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ will be associated with the minimum possible value of the considered metric (error rate or gini index or cross-entropy). This process is repeated but considering only one of the two resulting sub-regions and not the entire features space and so on until a stopping criterion is reached (an example of stopping criterion could be the minimum number of samples in each region): at the end of the fitting tree phase, in each created region R_1, R_2, \dots, R_K the predicted value corresponds to the most common class. Obviously, tree algorithms could produce accurate results during the training phase, but they can overfit on the test set. For this reason, it is better to create a less complex model with a limited number of splits, leading to lower variance and better interpretation at the cost of a little bias: the idea is to continue branching as long as a stopping criterion is met [11]. Examples are provided in Figure 4.1.

Decision trees show some advantages and disadvantages, in particular they could be graphically displayed and easily explained to people; they are more similar to human behavior (divide the space into boxes) and they can handle qualitative features, but on the other side their performances are not as well as other predictive models such as logistic regression. For this reason more advanced techniques have been developed, such as **bagging** and **random forest**.

To understand reasons behind the approach used by these new classification methods, reader must be made aware of **bias-variance** trade-off. It is strictly linked with prediction errors and it allows to create a more accurate model able to avoid **overfitting** and **underfitting** issues. Bias could be defined as the gap between model expected prediction and the corresponding true value [22]. Biased models do not adapt themselves to the training data and produce an error in both training and testing phase. Variance could be defined as *the variability of model prediction* when only a single point is considered [22]. Models with an high variance extremely adapt themselves to

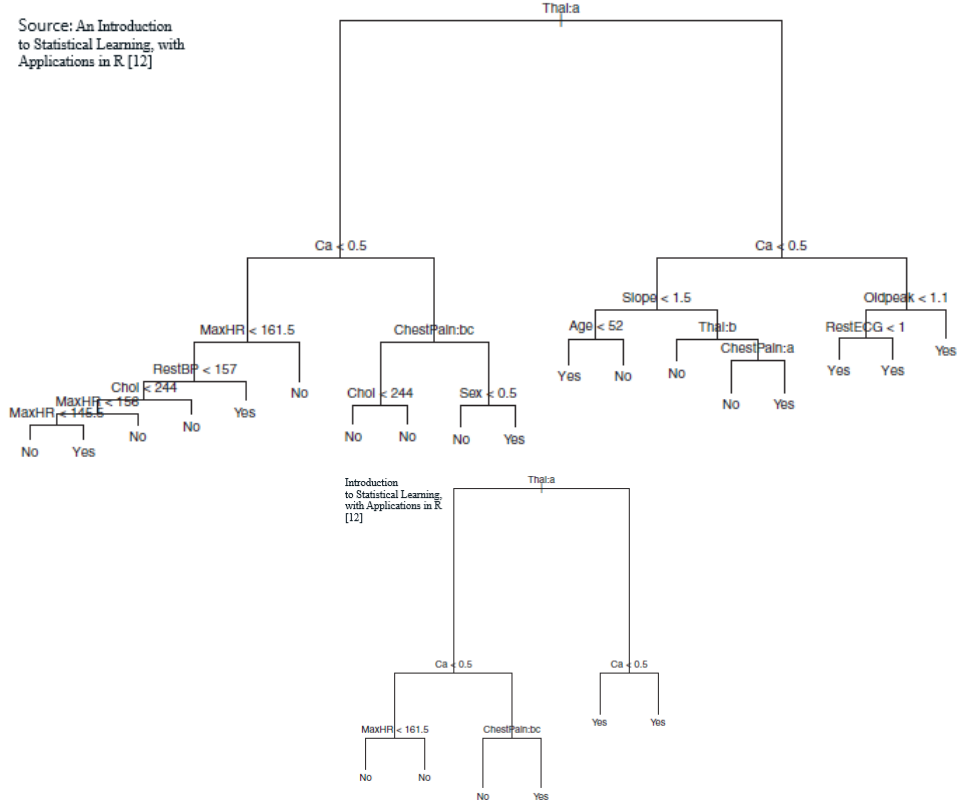


Figure 4.1: Decision Tree: default stopping criterion (top) and more additional conditions to stop (bottom).

the training set, producing a small error in training phase but a large one in the testing phase. To consolidate the concept, a regression practical example is required. Setting Y the quantity to predict and X the set of covariates, assume there exists a relationship such as:

$$Y = f(X) + e \quad (4.4)$$

where e represents the error term, 0-mean normally distributed. A model $\hat{f}(X)$ of $f(X)$ is built, producing in any point x the expected squared error:

$$Err(x) = \mathbb{E}[(Y - \hat{f}(x))^2] \quad (4.5)$$

which can be decomposed as:

$$Err(x) = (\mathbb{E}[\hat{f}(x)] - f(x))^2 + \mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2] + \sigma_e^2 \quad (4.6)$$

and representing as:

$$Err(x) = Bias^2 + Variance + Irreducible\ error \quad (4.7)$$

where the first term is the bias error, the second one is the variance error component and the last one is the irreducible error, which takes into account the data noise and it cannot be removed no matter how good the created model is. In the Figure 4.2, top left model is an example of low variance and

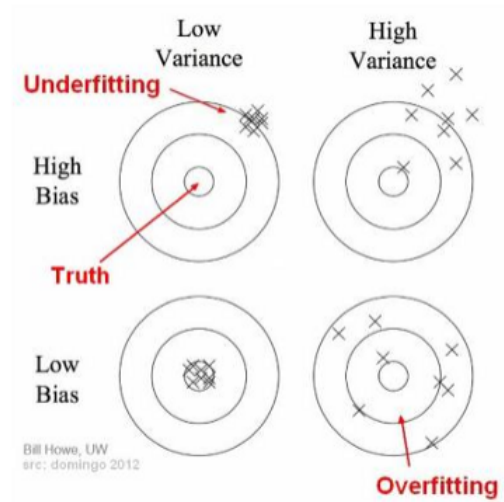


Figure 4.2: Bias and Variance concepts graphical representation

high bias since predictions are not spread in the domain but shifted from the target variable mean, while the bottom right is an example of high variance and low bias since predictions mean is equal to the target mean, but they are too much spread in the domain. The former represents the underfitting episode, the latter the overfitting one. Finally, the top right represents a wrong model, i.e. a model not suitable for the considered data, while the top left the ideal one, quite impossible to reach.

In supervised learning, **underfitting** happens when a model is unable to capture the underlying pattern of the data, for instance due to the lacking amount of observations to build it or a too simple algorithm, while the **overfitting** happens when a model captures the data noise pattern, for instance due to a lot of noisy datasets or very complex algorithms [22] (Figure 4.3). To conclude, a too simple model with few parameters probably is not able to well understand data pattern and it tends to have high bias and low variance, while a too complex model with a lot of parameters probably is extremely adapted to training data, producing high variance and low bias. The difficult step consists in finding the correct trade-off between underfitting and

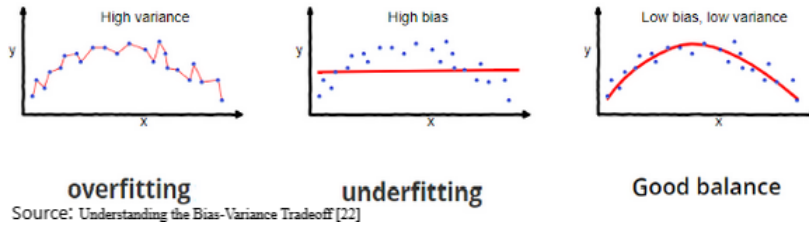


Figure 4.3: Underfitting and overfitting concepts graphical representation

overfitting to minimize the total error in Equation 4.7, finding an optimal balance of bias and variance to avoid the two negative options (Figure 4.4).

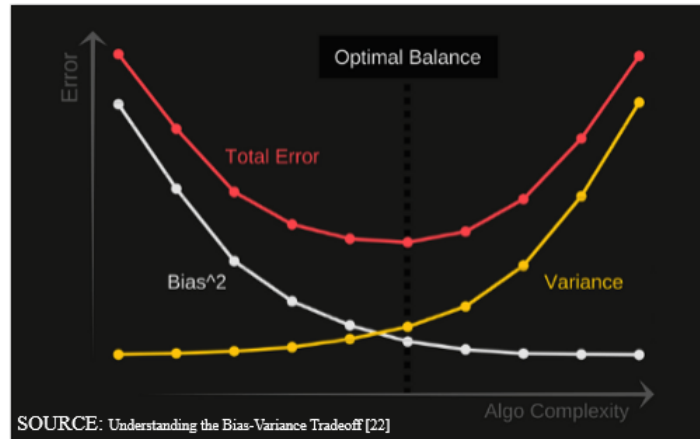


Figure 4.4: Bias and Variance trade-off

4.2 Random Forest

Back to the original presentation, one of the most critical part of decision trees algorithms is their high variance, in the sense that when applied to two input datasets, the output could be very different. On the other side a procedure with low variance caused the model's output to be quite the same, not strictly dependent from the input dataset. Since user wants to avoid both situations and, unfortunately, the previous algorithms belong to the first class, **bootstrap aggregation** or **bagging** technique is explored. It helps user to reduce the statistical learning method variance and it is frequently used for those types of solutions. Starting from a set of N independent samples

X_1, \dots, X_N , characterized by a σ^2 variance, samples' mean \bar{X} has a variance of σ^2/N [12]. The bagging approach consists in choosing many training sets $1, 2, \dots, K$ and used each one to build a model $\hat{f}_1(x), \hat{f}_2(x), \dots, \hat{f}_K(x)$. Each $\hat{f}_k(x)$ gives an output and the average (quantitative target) or the majority vote approach (qualitative target) is adopted to obtain the final prediction and a single low-variance statistical learning model is obtained:

$$\hat{f}_{avg}(x) = \frac{1}{K} \sum_{k=1}^K \hat{f}_k(x) \quad (\text{quantitative target}) \quad (4.8)$$

In the categorical case the final prediction consists in the most common class among the K outputs. However, having access to multiple training sets is not always feasible and for this reason the idea is bootstrap, namely consider multiple samples of the (single) initial dataset. Using the k -th subsample obtained, the $\hat{f}_k^*(x)$ is trained and it produces an output which is combined with all the others:

$$\hat{f}_{bag}(x) = \frac{1}{K} \sum_{k=1}^K \hat{f}_k^*(x) \quad (\text{quantitative target}) \quad (4.9)$$

This is the **bagging** approach [12].

Random forests represent an improvement over bagged trees, able to decorrelate the created trees. To understand why this could be important, suppose there exists a feature whose importance is way higher than the other predictors; most of the bagged trees will involve in the first split this most important feature. Therefore, the $\hat{f}_k^*(x)$ s are quite similar one to each other (correlated) causing the variance to not decrease as much as the trees would be uncorrelated: bagging approach shows a small improvement over a single tree.

For this reason, at each step only a random sub-sample of M features from the entire set of predictors P is considered and the feature involved in the split is searched inside the sub-sample. Each time M different random predictors are considered, where usually $M \simeq \sqrt{P}$, i.e. the total considered variables at each step is roughly the square root of the initial set of features. Basically, in each split the random forest algorithm cannot consider the majority of the variables and, coming back to the practical example before, on average the $(P - M)/P$ of the total separations do not take into account the most relevant feature, decorrelating the trees.

To conclude, bagging and random forest only differ from the number of features considered at each split, only a sub-set versus the entire set: indeed, a

random forest with $M = P$ is a bagging. Small values of M could be helpful when the set of initial features contain a lot of correlated variables [12].

4.3 Extreme Gradient Boosting

Extreme Gradient Boosting or **XGBoost** [5] could be considered another alternative to decision trees. It is a *scalable machine learning system for tree boosting* and it showed incredible results in a lot of machine learning competitions such as the Kaggle ones. The great advantages of this recent algorithm is the scalability, which makes it able to run faster than other classical solutions if built on a single machine and also able to scale up to billions of parallel resources in case of distributed or memory-limited conditions. In particular, new innovative approaches are examined, such as:

- the capability to handle *sparse data*;
- the *weighted quantile sketch* approach making the split finding less computational intensive;
- an innovative *sparsity-aware* approach to deal with null and missing values.

First step consists in presenting the supervised learning algorithm **gradient tree boosting**: assuming that the input data are composed by n examples, each defined in \mathbb{R}^d , i.e. d is the number of features. Defining $D = \{(x_i, y_i)\}$ where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$, a tree *ensemble model* combines K (trees) predictions to obtain the final value:

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^K f_k(x_i), \quad f_k \in \mathcal{F} \quad (4.10)$$

where $\mathcal{F} = \{f(X) = w_{q(x)}\}$ is the *CART*, the set of classification and regression trees. Then, defining T as the number of leaves in the tree, a function mapping each observation to one leaf index could be created $q : \mathbb{R}^d \rightarrow T, w \in \mathbb{R}^T$. So f_k is associated to a single tree structure, independent from the others, and with leaf weights w . An example is provided in Figure 4.5 where the aim is to identify whether someone could appreciate a new computer game. Each family member is assigned to a leaf and it is labelled with the corresponding leaf score. The reader could be a bit confused by this approach, expecting a categorical value instead of a continuous

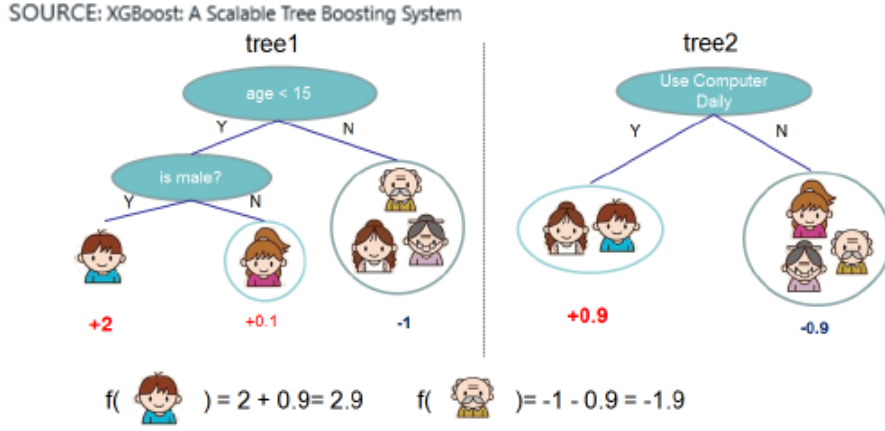


Figure 4.5: Example of tree ensemble model. Final output is obtained by summing the score in each tree

one, but the CART is different from decision trees where leaves only contain categorical prediction; a score is assigned to each leaf, giving user a deeper information going beyond the classification. Reader could ask which is the difference between random forest model and boosted trees. Actually, they are quite similar since both based their own approach on tree ensembles. The difference comes from the training phase: in the xgboost algorithm the set of f_k s are obtained by minimizing the *regularized* objective function:

$$\mathcal{L}(\phi) = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k) = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K (\gamma T + \frac{1}{2} \lambda ||w||^2) \quad (4.11)$$

where l is a *differentiable convex loss* function, measuring the difference between the true values y_i and the predicted ones \hat{y}_i , and Ω is a penalization term depends on the model complexity (useful to avoid over-fitting issue).

To train the model a required step consists in identifying functions f_k containing the tree structure and leaf scores. Clearly, it is much more difficult than traditional optimization methods in an Euclidean space, which simply consider the gradient and set it to zero. Instead of learning all the parameters in one single step, an iterative approach is adopted: adding at each step f_t which most improves the model according to Equation (4.11). Letting $\hat{y}_i^{(t)}$ the prediction at the iteration t -th of the i -th observation, a typical schema

could be showed:

$$\begin{aligned}
 \hat{y}_i^{(0)} &= 0 \\
 \hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
 \hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
 &\dots \\
 \hat{y}_i^{(t)} &= \sum_{k=1}^t f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
 \end{aligned} \tag{4.12}$$

and the objective function becomes:

$$\begin{aligned}
 \mathcal{L}^{(t)} &= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \sum_{k=1}^t \Omega(f_k) \\
 &= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \text{const.}
 \end{aligned} \tag{4.13}$$

In order to solve the previous equation, a second order approximation could be performed; recalling that for a generic two variables function $f(x, y)$ whose first and second partials exist at the point (x_0, y_0) , the second order approximation around (x_0, y_0) is:

$$\begin{aligned}
 f(x, y) &\simeq f(x_0, y_0) + f_x(x_0, y_0)(x - x_0) + f_y(x_0, y_0)(y - y_0) + \\
 &+ \frac{f_{xx}(x_0, y_0)}{2}(x - x_0)^2 + \frac{f_{yy}(x_0, y_0)}{2}(y - y_0)^2 + \\
 &+ f_{xy}(x_0, y_0)(x - x_0)(y - y_0)
 \end{aligned} \tag{4.14}$$

The obtained relationship around $(y_i, \hat{y}_i^{(t-1)})$ is:

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^n [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{4.15}$$

where $g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ and $h_i = \partial_{\hat{y}_i^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)})$ represent the first and second order loss function gradient (all other partial derivatives are equal to 0). Removing constant terms the equation to be minimized becomes:

$$\bar{\mathcal{L}}^{(t)} = \sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{4.16}$$

Then, after defining I_k as the set of observations contained in the leaf k , i.e. $I_k = \{j|q(x_j) = k\}$, Equation (4.16) could be written as

$$\begin{aligned}
 \bar{\mathcal{L}}^{(t)} &= \sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \gamma T + \frac{1}{2} \lambda \sum_{k=1}^T w_k^2 \\
 &= \sum_{k=1}^T \left(\sum_{i \in I_k} [g_i w_k + \frac{1}{2} h_i w_k^2] \right) + \gamma T + \frac{1}{2} \lambda \sum_{k=1}^T w_k^2 \\
 &= \sum_{k=1}^T \left[\left(\sum_{i \in I_k} g_i \right) w_k + \frac{1}{2} w_k^2 \left(\sum_{i \in I_k} h_i \right) + \frac{1}{2} \lambda w_k^2 \right] + \gamma T \\
 &= \sum_{k=1}^T \left[\left(\sum_{i \in I_k} g_i \right) w_k + \frac{1}{2} \left(\sum_{i \in I_k} h_i + \lambda \right) w_k^2 \right] + \gamma T
 \end{aligned} \tag{4.17}$$

and fixing the algorithm structure q , solving for w_k^* to find the function's minimum (convex function), the point obtained is:

$$w_k^* = - \frac{\sum_{i \in I_k} g_i}{\sum_{i \in I_k} h_i + \lambda} \tag{4.18}$$

and the corresponding value is:

$$\bar{\mathcal{L}}^{(t)}(q) = - \frac{1}{2} \sum_{k=1}^T \frac{(\sum_{i \in I_k} g_i)^2}{\sum_{i \in I_k} h_i + \lambda} + \gamma T \tag{4.19}$$

This $\bar{\mathcal{L}}^{(t)}$'s role is similar to the impurity random forest's score and it can be used to select the best possible split: basically it tries to capture the goodness of a tree structure q . Considering I_L and I_R as the set of observations in the left and right nodes created after the split and considering $I = I_L \cup I_R$ the loss reduction is:

$$\mathcal{L}_{split} = \frac{1}{2} \left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma \tag{4.20}$$

and it can be used to choose the best split among all the possibilities.

In addition to the regularization term, there are also two other possibilities to avoid overfitting: the first one is the **shrinkage**, which similarly to the learning rate in stochastic optimization it decreases single trees' weight in order to leave the possibility for future trees to improve the model, and the other one is the **column sub-sampling**, which selects only a sub-sample

of columns to fit the model [5].

One of the tree-based algorithms most critical problem is to find which is the best split to be performed in a specific step. Since an **exact greedy algorithm** where each possible solution is evaluated seems to be *inefficient*, especially if *data do not fit entirely into memory*, a significantly lighter solution is explored, called **weighted quantile sketch**. It is an approximate split finding algorithm, which could be divided into two step:

- choose only a subset of the possible splitting points;
- consider each features split, aggregate the statistics and according to them find the best solution.

More in detail, the first uses **quantiles** to determine the candidate points. For instance assume there exists only 1000 possible split points, called $\{x_1, \dots, x_{1000}\}$, a proper approximate approach could be consider only the 100-quantiles split points $\{x_{100}, x_{200}, \dots, x_{900}\}$ which reduces the computational cost. Theoretically, XGBoost approach is the following: suppose $X_k = \{x_{1,k}, x_{2,k}, x_{3,k}, \dots, x_{n_k}\}$ is the set of all observations k-th feature, define a *rank function* $r_k : \mathbb{R} \rightarrow [0, +\infty)$ as

$$r_k(z) = \frac{1}{\sum_{x \in X_k} h} \sum_{x \in X_k, x < z} h \quad (4.21)$$

which expresses the percentage of observation whose k-feature value is smaller than z .

Starting from the defined function, the set of splitting points $\{s_{k,1}, s_{k,2}, s_{k,3}, \dots, s_{k,l}\}$ could be found such that

$$|r_k(s_{k,j}) - r_k(s_{k,j+1})| < \epsilon, \quad s_{k,1} = \min_i x_{i,k}, \quad s_{k,l} = \max_i x_{i,k} \quad (4.22)$$

The role of ϵ parameter is allowing the algorithm to consider only quantiles. Basically, there will be $1/\epsilon$ points, i.e. if $\epsilon = 0.1$ more or less 10 splits will be consider. Then, since having a lot of continuous points already well classified makes useless to divide them, the main point is to consider very difficult to learn dataset parts. In order to reach the goal weighted quantiles are used, where h_i is the weight of i-th observation. Equation (4.16) can written as:

$$\sum_{i=1}^n \frac{1}{2} h_i [f_t(x_i) - (-g_i/h_i)]^2 + \Omega(f_t) + constant \quad (4.23)$$

where $-g_i/h_i$ is the label and h_i is the weight. Indeed

$$\begin{aligned}
 & \sum_{i=1}^n \frac{1}{2} h_i [f_t(x_i) - (-g_i/h_i)]^2 + \Omega(f_t) + \text{constant} \\
 &= \sum_{i=1}^n \frac{1}{2} h_i [f_t^2(x_i) + 2 \frac{f_t(x_i) g_i}{h_i} + (g_i/h_i)^2] + \Omega(f_t) + \text{constant} \quad (4.24) \\
 &= \sum_{i=1}^n [\frac{1}{2} h_i f_t^2(x_i) + f_t(x_i) g_i + \frac{g_i^2}{2 h_i}] + \Omega(f_t) + \text{constant}
 \end{aligned}$$

and the third term in the square brackets is constant since both g_i and h_i are determined by the previous iteration $t - 1$.

The second step consists in selecting the best solution according to the statistics obtained; there are two algorithm variants depending on when the $s_{k,j}$ s are chosen:

- **global** variant, where all splitting points are proposed during the initial phase of tree building and they are used for each tree level;
- **local** variant, where the splitting points are changed after each split.

The former requires less proposal steps but more points to obtain a satisfactory result, the latter refines the choice after each new separation and it is more suitable for deep trees [5].

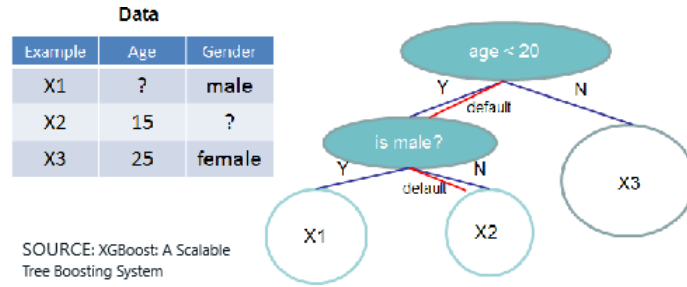


Figure 4.6: Tree structure with default directions.

Another important algorithm component is called **sparsity-aware** split finding. Since in a lot of classification/regression problems the data matrix X is *sparse*, XGBoost developers configure for each of the created node a *default direction* (Figure 4.6). The sparsity-aware ability is essential since sparse matrices are not so rare and they could be generated by:

- missing values;
- output of a specific preprocessing step, such as one-hot encoding;

This approach allows to classify an observation even if the value of splitting feature is missing, by configuring the best default branch determined from the data. To be more clear consider a node with 100 observations and a feature which registers if the user has already rented a car. The idea is to create two new branches, one for the positive answer (Yes) and one for the negative (No). Assume input data could be classified into three distinct group: group A composed by 35 users who have already rented a car, group B composed by 55 users who have never rented a car and the group C where this information is not available and it is only composed by 10 users. The algorithm divides easily the two groups A , B and to understand how to classify the group C , it tries the combination $A+C$ (Yes), B (No) and A (Yes), $B+C$ (No), then it computes the value to be assigned to the generated leaves using all the training data. Next, the loss is computed in both cases and the solution chosen is the one able to minimize the error [5].

Finally XGBoost shows also other interesting characteristics more related to the system design, such as column block for parallel learning, cache-aware access and blocks for out-of-core computation. However, this part is outside the scope of this thesis and it is not deepened, but further information is available on the official algorithm presentation.

Chapter 5

Implementation: dataset presentation and exploration

The previous introduction has been useful to describe the project's background, the ideas that inspired it, the concepts and the design that drove it since first meetings. The reader had the possibility to understand the importance of a strict collaboration between two internal teams and to have a general idea of how data are collected from private homes. Now it could be interesting to focus more on the analytics part, by considering the entire artificial intelligence flow until the model results. First of all a periodic data extraction is performed, bringing up-to-date information from CEM Platform to analytics cloud, then data are processed, algorithms tested and models trained. The final step consists in deploying and running the model by distributing machine learning models and computing real time scoring. More efforts will be spent presenting each single step, allowing reader to better understand the entire pipeline.

5.1 Data extraction

Periodic data extraction is composed by:

1. Data extraction , from *CEM - events pipeline* to *CEM – post processors file systems* : when a new set of data is retrieved, it is exported as a zipped CSV file in a directory on the file system of each post processor node.
2. Data push, from *CEM – post processors file systems* to *AWS analytics cloud – data loader server* : on each post processor node a scheduled

python script will scan periodically the output directory and push the files on the data loader server into AWS analytics cloud.

3. Data Extraction/Push, from *CEM - MongoDB* to *AWS analytics cloud* – *data loader server* : a scheduled python script, will export metadata (static) into files from MongoDB database, then it will push these files into AWS analytics cloud (data loader server).
4. Data Push, from *AWS analytics cloud* – *data loader server* to *AWS analytics cloud* – *S3 storage* : a dedicated script will move the files loaded on the data loader server into the S3 storage to make them available for future analysis.

Once data are stored in a S3 bucket, they can be easily examined by downloading them. Briefly the input data consists in:

- parquet files containing **reasons** for a specific complaint in a timestamp for a single CPE;
- parquet files containing **ticket** technical information after been opened by an operator;
- parquet files containing **anagraphic** information about a single CPE and a single line;
- csv files containing **raw measures**, i.e. all the technical parameters retrieved from customers' CPEs each 15 minutes.

5.2 Data preparation

All previous datasets are joined in a consistent way, combining raw measures with ticket troubles and anagraphic information. While it is quite easy to combine raw measures with anagraphic information, it appears to be not so easy to understand how to link a ticket opening to a raw measure related to same CPE technical parameters. Knowing that a certain consumer calls to report a problem, when does that problem truly appear or begin? This has been one of the most difficult part during the project planning, since it requires to take into consideration **customers behaviour**: some of them can contact immediately the customer service, others could wait hoping that the issue will be solved in few hours. It could happen that a problem occurring in the morning is perceived from the customer only when he will be back home in the evening after working day or an issue in the weekend maybe it

will not be captured until Sunday night. Those are only some of the possible scenarios, but make the readers aware of the complexity of this phase. At the end, always in a strict collaboration with the client, the final project choice has been to consider a time window of **36** hours before. A graphical representation of the choice is reported in Figure 5.1

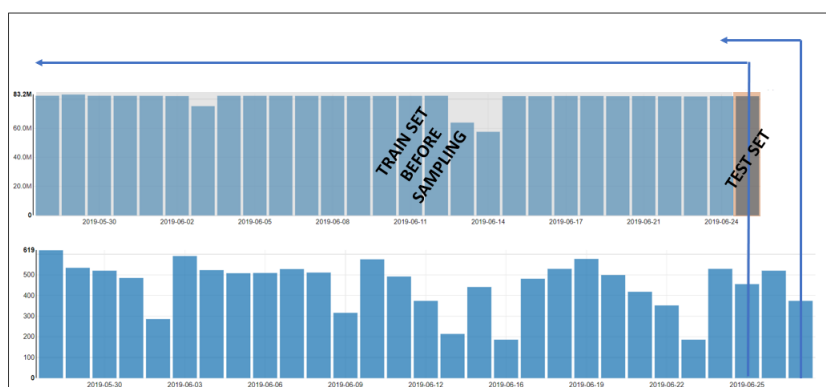


Figure 5.1: Creation of the target variable.

Then, a graphical examination of the data to find unusual observations far from the mass of data is performed (Figure 5.2). It appeared that some observations, called outliers, showed some unusual KPIs values: to avoid leading to wrong conclusions they are removed.

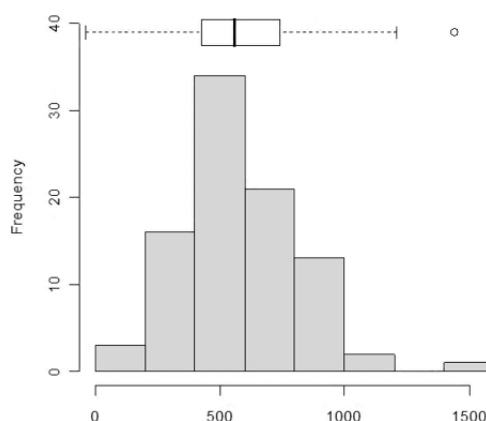


Figure 5.2: Plot of an example of aggregate KPI distribution.

Then, efforts moved to a special type of features : the **derived** ones. In particular, there are some data reporting the result of an aggregation made

with a specific time window. In order to standardise out dataset and to keep more control on the aggregation time window, those features have been eliminated to be computed manually from the initial raw measures in the next step. After that, new key indicators are introduced and others are customized for our task (for instance the line of the age is changed in new and old customers basing on a predefined threshold). Then, a more detailed focus on the target variable is needed: **ticket argument**, which represents for each single customer's call the reason behind it. Obviously, since we perform a left joined between the raw measures, whose granularity is 15 minutes, and the ticket opening on the unique CPE's id, the field related to ticket argument is often a null value since no call occurs. Then, according to the client's needs we decide to restrict our analysis only to a subset of the entire ticket arguments; the considered use cases are:

- **call received;**
 - **wi-fi issue;**
 - **unstable line;**
 - **slow line;**
- **no call received;**

Figure 5.3 and 5.4 display how ticket reasons are distributed over a predefined period. While the former is an aggregate view, the latter shows more in detail the daily distribution.

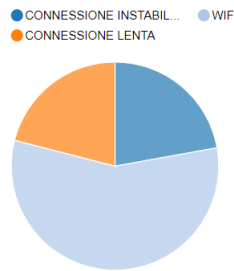


Figure 5.3: Pie chart representing a ticket arguments distribution over 3 months.

Clearly, the dataset is unbalanced since the number of customers that do not call is higher than the customers that call; for this reason the idea is to perform a stratified sampling creating a dataset by keeping 10% of the not null target variable (with respect to the null target variable) and a constant

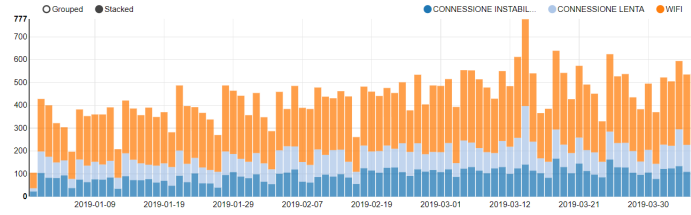


Figure 5.4: Stacked bar chart representing ticket arguments distribution over 3 months.

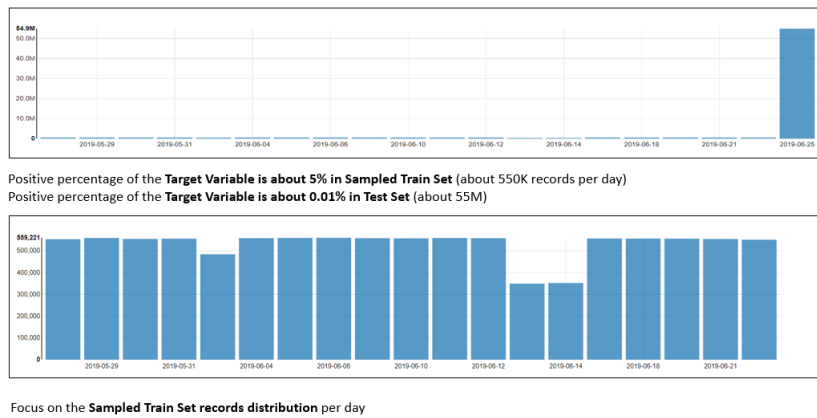


Figure 5.5: Comparison between train and test size.

percentage of some predefined features (this step is completed in a strict collaboration with the client).

Next step consists in creating the derived features by aggregating them considering different time windows (e.g. $2h$, $12h$, $24h$, 1 week) and computing the average, standard deviation, the maximum and the minimum value. This approach is motivated by two different reasons:

- **features augmentation** : increasing the number of available features in order to maximize the performances;
- **reduce data variability** : related mainly to mean computation, making new feature distribution a more gaussian-like one.

Finally we manage missing values in data, distinguishing between three types:

- **missing rows**: an entire row measure is populated with *null* values (this means that a station is not sending information) and it is removed from dataset;

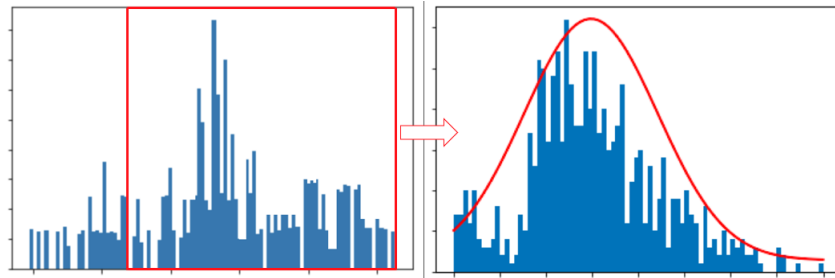


Figure 5.6: Compute mean function to the original data distribution.

- missing values: only some features are populated with *null* (this implies a temporary local problem) and they are imputed using the mean value for numerical features and the most common one for the categorical features;
- missing derived: only derived features are populated with *null* (this can happen after a router restart) and they are imputed using the mean value.

At the end of this part we obtain a large dataset, ready to be trained, with more than 700 columns and 1 million rows : for this reason Apache Spark unified analytics engine for large-scale data processing is used, exploiting the machine learning algorithms already developed in its libraries.

5.3 Model pipeline

Before presenting the proposed solution it could be important to underline the idea which guides it. The client showed an interest in anticipating the line troubles before they happen and he requested a model which has as main goal to be very precise. In particular it would like to be sure that if the trained model predicts a customer call, there will be truly a problem on the line. In other words, he is interesting in a model with high level of call precision. In a classification binary task, it is possible to identify two distinct classes: the **positive** class (in this case it is the **call** event) and the **negative** class (in this case it is the **no call** event). Then, a **true positive** (TP) is an outcome where the model correctly predicts the positive class and, similarly, a **true negative** (TN) is an outcome where the model correctly predicts the negative class. On the other side, a **false positive** (FP) is an outcome where the model incorrectly predicts the positive class while a **false negative** (FN) is an outcome where the model incorrectly predicts the negative class. Using

these 4 different values it is possible to compute some metrics to evaluate classification performances. In particular [56]:

- **accuracy**, which represents the fraction of correctly predicted samples with respect to the entire dataset. This is useful when the dataset is balanced, i.e. number of positive and negative labels are quite similar and when a false positive is as serious as a false negative one.

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN} \quad (5.1)$$

- **precision**, which represents the fraction of correctly predicted positive samples with respect to the entire positive predictions made. This is useful when incorrectly predict the positive class must be avoided.

$$precision = \frac{TP}{TP + FP} \quad (5.2)$$

- **recall**, which represents the fraction of correctly predicted positive samples with respect to the number of positive samples. This is useful when incorrectly predict the negative class must be avoided.

$$recall = \frac{TP}{TP + FN} \quad (5.3)$$

- **F1 score**, which represents the *weighted average of precision and recall* able to consider both false positives and negatives. It results to be usually more useful than accuracy, especially when the two classes are unbalanced.

$$F1_score = 2 * \frac{recall * precision}{recall + precision} \quad (5.4)$$

Another important element in the analysis will be the **confusion matrix**, an NxN table that summarizes the quality of a classification model. One axis of confusion matrix represents the true label while the other one the model predictions and N is the number of classes.

The most important metric for this project is the precision of the positive class. However, also another metric which represents how well the model recalls the class, is taking into account. Basically, there is also an interest in knowing how many customers calls the model is able to capture.

To perform a solution which fully satisfies the client, the pipeline is splitting in two main steps by considering different target variable values. First of all we consider a **binary classification** model, where we want to predict if a call will occur or not, by creating the following objective variable:


```
data = data.withColumn('target',
                        when((col('tick_arg').isNull()),0).otherwise(1))
```

In this way we are able to give an instantaneous feedback to the client, making him aware whether there will be a customer call in the future: at the end of this step we have a binary classification trained algorithm. Since the client is also interested in knowing where the problem will occur, a second step is performed. After filtering the initial dataset to consider only not null ticket arguments, a multiclass target variable is created using the following objective variable:

```
data = data.where(col('tick_arg').isNotNull())
            .withColumn('target',when(col('tick_arg') == 'WIFI', 0)
            .when(col('tick_arg') == 'CONNESSIONE LENTA', 1)
            .when(col('tick_arg') == 'CONNESSIONE INSTABILE', 2))
```

At the end of this second step we have a **multiclass classification** trained algorithm, able to identify a specific issue. Therefore, thanks to the two considered steps we are now able to predict both if a problem/call will occur and give an explanation of the reason behind it. Then the model output changes moving from a simple prediction (call/no call and type of issue) to a **probability** that a certain event will happen. In particular, when a new sample becomes available, it is sent to the model pipeline producing the following output:

- the first step produces as output the probability of a future call event, with the schema reported in Table 5.1:

Sample	Probability of call	Call/No Call prediction
Sample1	0.8	Call

Table 5.1: Call/ No Call binary prediction model.

- the second step produces as output the probability of each considered issue (wifi, unstable line, slow line), with the schema reported in Table 5.2:

To conclude, the last design choice made consists in providing the second step probability also for the sample whose prediction is no call: in this way the client has always the possibility to completely monitor the situation, giving him a large quantity of information.

Sample	Probability of call	Wifi	Unstable Line	Slow Line
Sample1	0.8	0.5	0.3	0.2

Table 5.2: Issues multiclass prediction model.

5.4 Features selection

From the big data theoretical chapter, reader has already understood that during the last few years data dimensionality grew exponentially. This represents a tricky aspect for machine learning algorithms which have to face problems such as the **curse of dimensionality**, the large amount of **storage** space required and an high **computational cost**. Therefore, the interest in techniques able to select only relevant information increases continually, making these approaches quite general and suitable for a lot of fields. Dimensionality reduction represents feature selection main goal, allowing to improve learning performance, i.e. an higher learning accuracy, a lower computational cost and a better model interpretability; the key concept consists in removing **irrelevant**, **redundant** and **noisy** information [26]:

- **irrelevant** features : they do not help distinguish observations belonging to different clusters, in case of unsupervised learning, or different classes, in case of supervised learning. It is important to remove them, because they cannot improve algorithm ability and they can confuse the training model and cause computation inefficiency. Figure 5.7a and 5.7b show the difference between a relevant feature and an irrelevant one: in the first $f1$ helps to correctly identify and classify the two classes, while in the second $f2$ is irrelevant to distinguish class 1 from class 2;
- **redundant** features : individually they could provide useful information, but due to the presence of another feature, they become redundant, i.e. after removing them the learning performance does not change. Figure 5.7c helps to explain this concept, indeed $f6$ is useful to classify the two classes, but when $f1$ is considered before, the previous one does not provide new information and it becomes redundant;
- **noisy** features : generally they provide relevant information, but since they introduce also a noise in data processing they are rarely considered relevant for analysis. Figure 5.7d underlines how feature $f4$ can distinguish only some data points, introducing a confusion in learning model by putting together elements belonging to different classes.

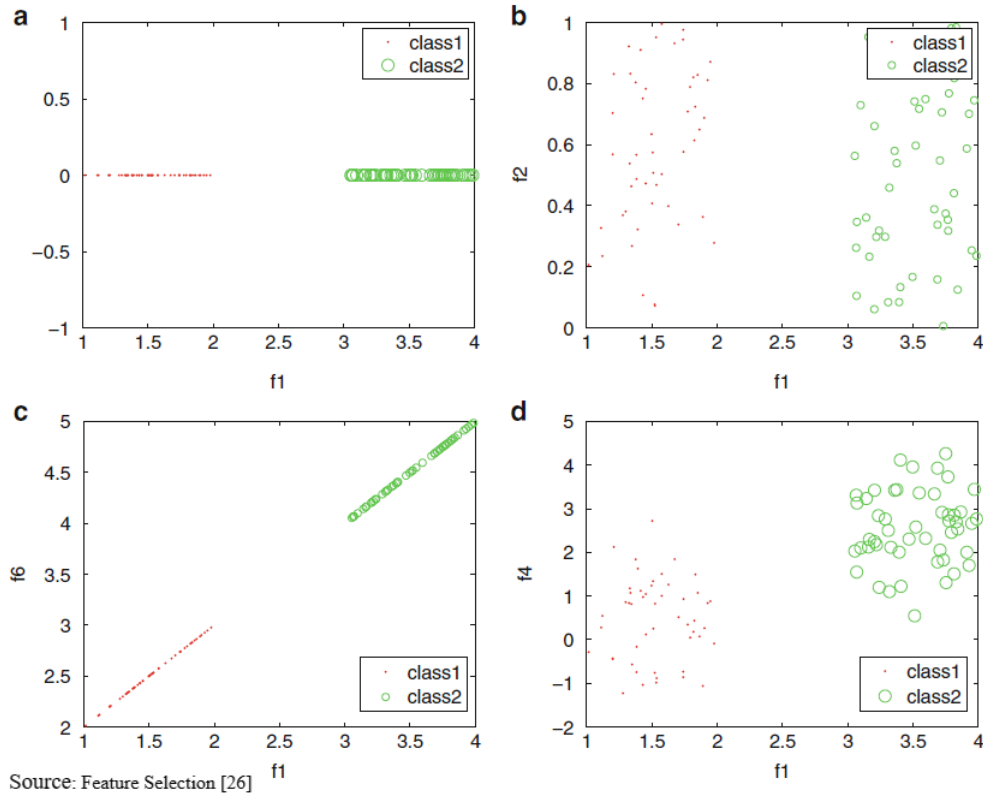


Figure 5.7: Concept of irrelevant, redundant, and noisy features. (a) Relevant feature. (b) Irrelevant feature. (c) Redundant feature. (d) Noisy feature.

However, it is essential to remember that sometimes two noisy features, once combined, could improve learning performances.

Dealing with high dimensional data is become a common situation in a lot of fields, such as data mining and machine learning, where the amount of information uncontrollably grew during the last few years. The increasing amount of data dimensionality obliged data scientist to deal with new complex problems. Some examples are **overfitting** due to the limited number of training samples, the curse of dimensionality due to the elevate number of dimensions, the existence of irrelevant, redundant and noisy features and finally the large amount of time and memory required. Techniques to reduce data dimensionality could be divided into **feature extraction** and **feature selection**, both able to improve model building and to decrease storage space needed to process data. Feature extraction consists in mapping the original space into a low-dimensional one through a combination of starting features, but the new generated variables loose their physical meaning, making the

solution less understandable. On the other side feature selection consists in considering only some of the starting features and it keeps the original dimension meaning, making this second approach better for readability and interpretability [26].

Back to the project, after removing some irrelevant features from the analysis (such as the ones with too much missing values), the project decision is to perform two different features selection, one for each classification task. The next lines will expose the general approach used in both cases, since it is almost the same (except for the target variable type).

Working on the project case, the initial set of features is divided in **numerical** and **categorical** ones using the following python code:

```
def divide_features(sub_sample):
    categorical_features = []
    numerical_features = []
    for i in range(len(sub_sample.columns)):
        type_ = sub_sample.dtypes[i][0]
        if sub_sample.dtypes[i][1] == 'string':
            categorical_features.append(type_)
        else:
            if sub_sample.dtypes[i][0] != 'target':
                numerical_features.append(type_)
    return numerical_features, categorical_features
```

Then two different types of features selection are performed: supervised and unsupervised ones. For the numerical features:

1. **supervised** features selection based its approach on the supervised learning algorithms: in particular, **random forest** and **logistic regression** algorithms are implemented. By keeping the default parameters, two models are trained using the initial dataset producing a list of the models most important features, where the features importance is retrieved from the *features importances* method of random forest or *coefficients* method of logistic regression. Between the two algorithms only one is selected, the most performing one (according to the precision metric, as explained before), use the following python code:

```
def best_values(accuracy_list,coef_list):
    best_accuracy = 0
    for i in range(len(accuracy_list)):
        if accuracy_list[i] >= best_accuracy :
```

```

        best_importance = coef_list[i]
        best_accuracy = accuracy_list[i]
    return best_importance, best_accuracy

```

Only the best features are selected (level of importance > specific threshold): this selection is motivated by a graphical inspection: it is clear that there exists a sort of elbow in the plot, pointing out that below a fixed threshold all the other features could be considered as useless to the analysis (Figure 5.8).

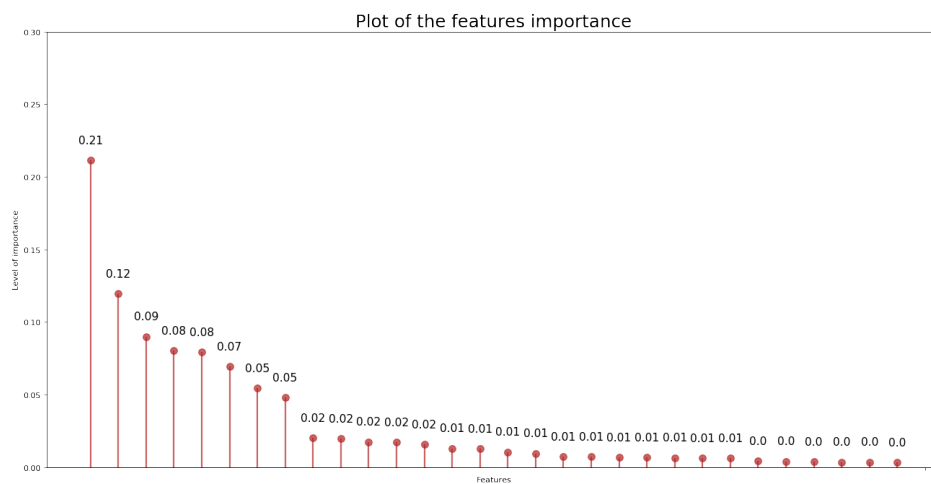


Figure 5.8: Random forest features importance for the binary model.

2. **unsupervised** features selection based its approach on the **Spearman correlation**: in the first step the most correlated features are removed in order to avoid redundant information (correlation > 0.9), then correlation between target variable and all other features is computed and only the most correlated ones are kept. Correlation is inspected in the Figure 5.9.

For the categorical features:

1. **unsupervised** features selection based its approach on the **Spearman correlation**: in the first step the most correlated features are removed in order to avoid redundant information (correlation > 0.9), then correlation between target variable and all other features is computed. We can see the correlation in the Figure 5.10. However since the number of categorical features is not so high, the final decision was to ignore the Spearman correlation output;

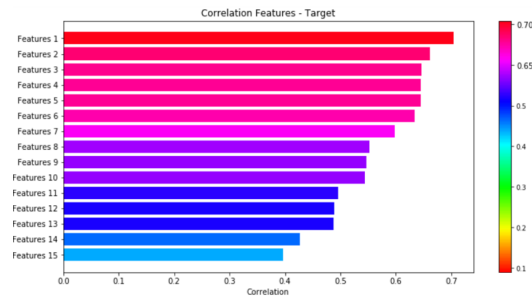


Figure 5.9: Correlation between target variable and some numerical features.

2. checks if the categorical feature contains at least **two distinct** values, otherwise it is useless for the analysis.

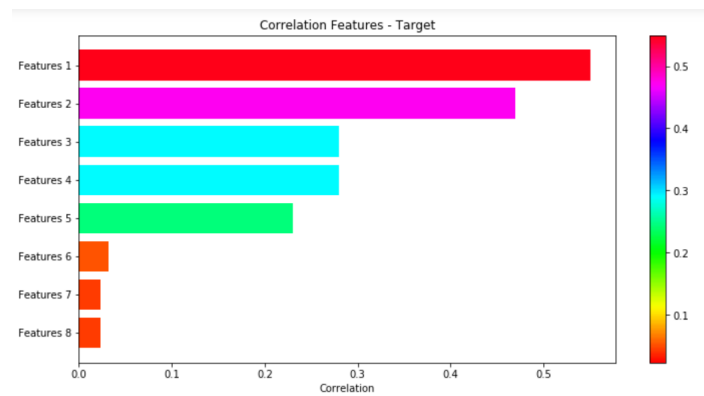


Figure 5.10: Correlation between target variable and some categorical features.

At the end the script combines the two lists and store the results on *S3* to be used for future analysis.

Chapter 6

Implementation: algorithms development

6.1 First step: binary classification

The choice of using Apache Spark defines the solution's starting point, consists in running a **Spark Session**. In particular *spark – 2.4.1 – bin – hadoop2.7.tar* is downloaded and uploaded to a specific folder and then the environmental variable *SPARK_HOME* is set to the folder's path: this allows user to keep control on the Spark version. Used Spark configuration is created in the code below and reported in the Table 6.1.

```
import os
import boto3

os.environ["SPARK_HOME"] = "spark_home"

from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession

import sagemaker
from sagemaker import get_execution_role
import sagemaker_pyspark

# Configure Spark to use the SageMaker Spark dependency jars
jars = sagemaker_pyspark.classpath_jars()

classpath = ":".join(sagemaker_pyspark.classpath_jars() +
```

```

        ['other packages'])

spark = SparkSession.builder\
    .config("spark.driver.extraClassPath", classpath)\
    .master("local[*]")\
    .getOrCreate()

region='considered_region'
spark._jsc.hadoopConfiguration()
    .set("fs.s3.awsAccessKeyId", access_key)
spark._jsc.hadoopConfiguration()
    .set("fs.s3.awsSecretAccessKey", secret_key)
spark._jsc.hadoopConfiguration()
    .set('fs.s3.endpoint', 's3.{}.amazonaws.com'.format(region))

```

Spark Context		
Spark Version	Master	AppName
v2.4.1	local[*]	pyspark-shell

Table 6.1: Spark configuration.

The dataset is retrieved from *S3* and after selecting only the features to be kept (this information is registered in *S3*) an **AWS Glue process** is run: the first step consists in calling **Glue APIs** using the role linked to the AWS user account. This is possible from the AWS dashboard, searching for the configured roles and adding AWS Glue as an additional trusted entity. Data preprocessing applied is composed by the following steps [53]:

- reading from *S3* the selected features;
- **StringIndexer**, which transforms a string format column of labels to a integer format column of indices, whose value is in $[0, \text{numLabels})$;
- **OneHotEncoderEstimator**, which transforms, according to the official documentation, *a categorical feature to a binary vector* which testifies the *presence of a specific feature value* between all the existing values. **OneHotEncoderEstimator** could be applied also to multiple columns, applying the previous approach to each single input column and it gives also user the possibility to set *handleInvalid* parameter to

decide how to tackle invalid input: for instance “keep” to assign to them an extra categorical index or “error” to force an error;

- **VectorAssembler**, which collapses a set of specified columns into one single vector column. VectorAssembler works for numerical, vector and boolean type columns, concatenating them using the defined order for each row.
- **StandardScaler**, which normalizes each feature to have zero mean and/or unit standard deviation. According to the parameters *withMean* and *withStd*’s value, it subtracts the mean from each feature (zero mean) or it scales data dividing each feature by its standard deviation (unit Std).

The screenshot shows the AWS Glue console interface. On the left is a navigation menu with options like Data catalog, Databases, Tables, Connections, Crawlers, Classifiers, Settings, ETL, Workflows, **Jobs**, Triggers, Dev endpoints, Notebooks, Security, and Security configurations. The main panel is titled 'Jobs' and contains a description: 'A job is your business logic required to perform extract, transform and load (ETL) work. Job runs are initiated by triggers which can be scheduled or driven by events.' Below this is a table of jobs. The table has columns: Name, Type, ETL language, Script location, Last modified, and Job bookmark. Three jobs are listed, all named 'sparkml-preprocess-2...'. The third job is selected. Below the jobs table is a section titled 'View run metrics' showing details for a specific run. The run metrics table has columns: Run ID, Retry attempt, Run status, Error, Logs, Error logs, Maximum execution capacity, Execution time, Timeout, Delay, Triggered by, Start time, and End time. The run ID is 'jr_a1a97c8...', the status is 'Running', and the execution time is '0 secs'.

Name	Type	ETL language	Script location	Last modified	Job bookmark
sparkml-preprocess-2...	Spark	python		14 July 2019 12:00 P...	
sparkml-preprocess-2...	Spark	python		15 July 2019 10:13 A...	
sparkml-preprocess-2...	Spark	python		15 July 2019 10:21 A...	

Run ID	Retry attempt	Run status	Error	Logs	Error logs	Maximum execution capacity	Execution time	Timeout	Delay	Triggered by	Start time	End time
jr_a1a97c8...	-	Running		Logs	Error logs	2	0 secs	60 mins			15 Jul...	

Figure 6.1: AWS Glue Preprocessing: example of running script

The screenshot shows the AWS Glue console interface, similar to Figure 6.1. The 'Jobs' section is active. The table of jobs shows the same three jobs. The first job is selected. Below the jobs table is a section titled 'View run metrics' showing details for a specific run. The run metrics table has the same columns as in Figure 6.1. The run ID is 'jr_d834a26...', the status is 'Failed', and the execution time is '0 secs'.

Name	Type	ETL language	Script location	Last modified	Job bookmark
sparkml-preprocess-2...	Spark	python		14 July 2019 12:00 P...	

Run ID	Retry attempt	Run status	Error	Logs	Error logs	Maximum execution capacity	Execution time	Timeout	Delay	Triggered by	Start time	End time
jr_d834a26...	-	Failed		Logs	Error logs	2	0 secs	60 mins			14 Jul...	14 Jul...

Figure 6.2: AWS Glue Preprocessing: example of failed script

In this step we used the AWS Glue possibility to work with a customized pyspark script to perform an ETL process. Situation can be easily monitored by Glue dashboard (as reported from Figure 6.1 to Figure 6.4). Finally post-processed dataset is split in 80 – 20 train and test set and on S3 the following objects are uploaded:

- csv files containing the training set;
- csv files containing the test set;
- the serialized model so that it can be used with SageMaker for inference later;

Apache Spark is best suited batch processing workloads, but in order to use the Spark ML preprocessing trained for future inference, the **MLeap** library is required to serialize it, by using the *SerializeToBundle()* method: in this way it is possible to exploit SageMaker SparkML service to make future **real-time** or **batch** inference. In order to make this functionality available in the preprocessing phase, it needs to pass MLeap dependencies to Glue. As the majority of Spark packages, MLeap is implemented as a Scala package with a front-end wrapper written in Python and for this reason it could be used from PySpark [57].

Next step consists in setting both the transformed dataset output location and the MLeap serialized model location: *getResolvedOptions* method of AWS Glue library helps to reach this goal. Then, Glue client is configured using Boto3 python library and the *create_job* Glue API is invoked to initialize a Glue job. After passing the code location and the dependencies location, the API creates an immutable execution based on job parameters set before. Once the Glue job has succeeded, data into S3 are transformed in csv format and they can be used for training [57].

Three algorithms are adopted during the training phase, two use the scikit-learn customized library and the third one is a SageMaker built-in algorithm. With Scikit-learn Estimators, it is possible to train and deploy Scikit-learn models on SageMaker in a two-step process [58]:

- prepare a Scikit-learn script to run on SageMaker;
- run this script on SageMaker via a SKLearn Estimator.

The first should be a file uploaded in a different source file than the notebook used to train the algorithm via SKLearn Estimator. First step is to create a *.py* Scikit-learn training script composed by a *main* function, where data

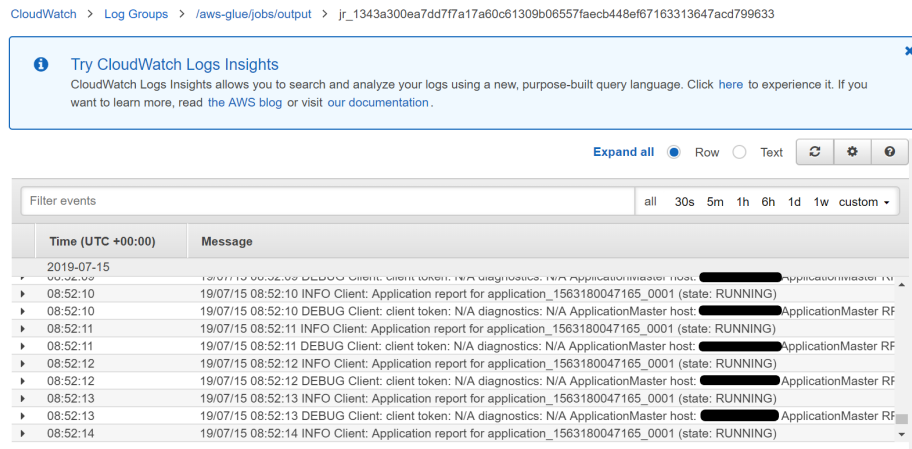


Figure 6.3: AWS Glue Preprocessing: log inspection

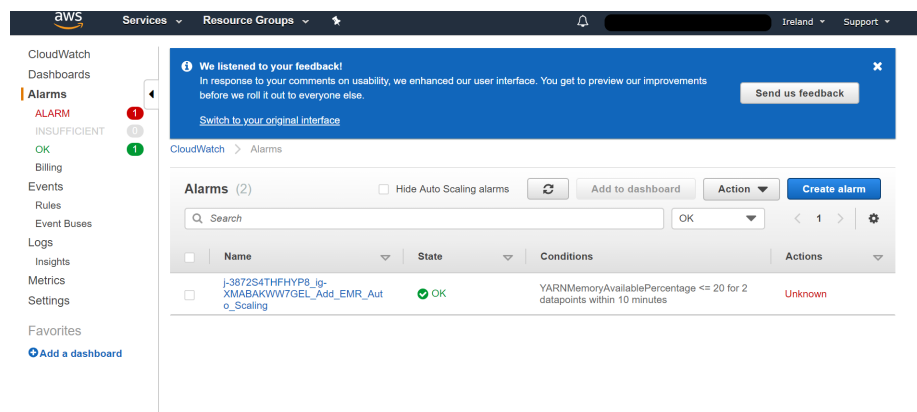


Figure 6.4: AWS Glue Preprocessing: how to monitor the status

are retrieved from *S3* and algorithm hyperparameters are set. Then, the fit method is invoked and the relative model is trained: finally a score is computed on the test set. In addition to the main function which must contain the previous defined steps, also others must be defined inside the *.py* script. One is the *model_fn* function which is required to load the model. The *model_fn* schema is [58]:

```
def model_fn(model_dir):
```

and it returns an object.

Then, SageMaker could serve the model. Model serving represents the set of steps which return answer to an inference request, *received by SageMaker InvokeEndpoint API calls*. The SageMaker Scikit-learn model server manages input requests into three steps: input processing, prediction and output

processing, by defining those functions inside the *.py* script. Each step invokes a python function, whose input and output values are explained in the following chain [58]:

```
# Deserialize the Invoke request body into an object we can
# perform prediction on
input_object = input_fn(request_body, request_content_type)

# Perform prediction on the deserialized object, with the
# loaded model
prediction = predict_fn(input_object, model)

# Serialize the prediction result into the desired response
# content type
output = output_fn(prediction, response_content_type)
```

Next step consists in running the created **Scikit-learn** training script on SageMaker by configuring a *SKLearn Estimator* and calling the *.py* file using the fit method of the SKLearn Estimator. The following code represents an example of training phase, with some input hyperparameters and “train” and “test” channels. According to the official documentation SKLearn Estimator runs the *.py* script in a customized *execution environment*, as a part of a SageMaker Training Job. This environment is basically an *Amazon built-in Docker container* which runs the *.py* file specified in the *entry_point* SKLearn Estimator parameter:

```
from sagemaker.sklearn.estimator import SKLearn
algorithm = 'algorithm chosen'
output_mod = 'output location of the model'
train_input = 'input location of the training data'
validation_input = 'input location of the test data'
hyperparameters= {
    'n_estimators': 10,
    'max_depth': 5
},
script_path = 'path location of the py file'
sklearn = SKLearn(
    base_job_name = 'name of the job',
    entry_point=script_path,
    train_instance_type="ml.m5.large",
    train_instance_count= 1,
    train_max_run= 86400,
```

```

train_volume_size= 30,
role=role,
sagemaker_session=sagemaker_session,
hyperparameters=hyperparameters,
output_path= output_mod)

```

This code can be used for each Scikit-learn algorithms.

It is well-known that an algorithm has a lot of hyperparameters and using only predefined values of them will make the model less powerful. For this reason it is essential to consider an **hyperparameters tuning** and Amazon SageMaker gives users a class to perform this task and also to deploy the originated model. The operator should only create the *HyperparameterTuner* object by setting the *objective_metric_name*, the hyperparameter ranges and the metric definitions. *Hyperparameter ranges* could be a continuous, integer or categorical one; it should be passed as a dictionary with keys the hyperparameters names and values the parameters range. The *metric definitions* is a list of dictionary, each representing a metric and containing *Name* for the name of the metric, and *Regex* for the regular expression used to extract the value of the metric. This parameter must be set only for hyperparameter tuning jobs not considering an Amazon built-in algorithm [58].

```

from sagemaker.tuner import HyperparameterTuner,
                             IntegerParameter,
                             CategoricalParameter

# Configure HyperparameterTuner
my_tuner = HyperparameterTuner(
    # previously-configured Estimator object
    estimator= sklearn,
    objective_metric_name='test-score',
    hyperparameter_ranges=
        {'n_estimators': IntegerParameter(6, 15),
         'max_depth': IntegerParameter(3, 7)},
    metric_definitions=[
        {'Name': 'test-score',
         'Regex': 'Test score: (.*)?;'}
    ],
    max_jobs=12, # REMOVE
    max_parallel_jobs=2,
    tags=tags_list)

```

Then, the fit method is invoked to start the tuning job and once terminated

it is possible to create a model from the trained algorithm, using as hyper-parameters the best configuration found (according to the chosen metric).

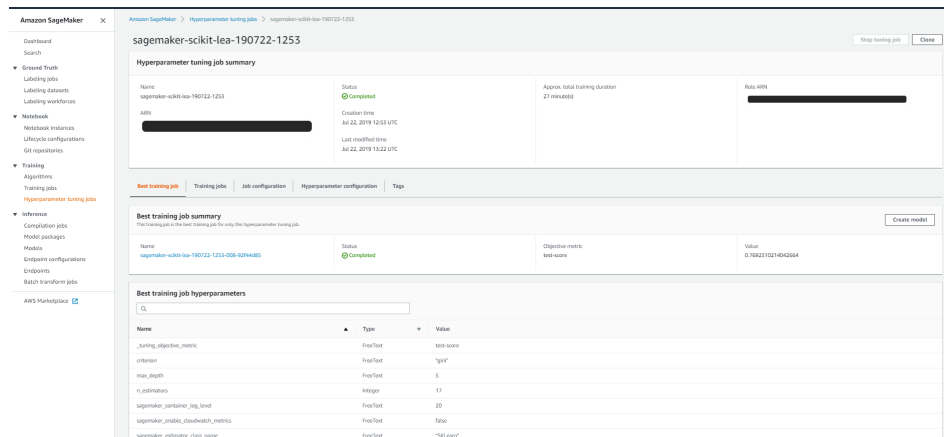


Figure 6.5: SageMaker: visualize hyperparameter tuning job results

```
from sagemaker.estimator import Estimator

my_tuner.fit({
    'train': train_input,
    'test': validation_input
}, logs=True)
my_tuner.wait()

tuner_estimator = Estimator.attach(my_tuner.best_training_job())
training_algorithm = tuner_estimator.create_model()
```

Since the project's scope consists also in finding and testing AWS SageMaker's built-in algorithm, also the **xboost algorithm** is considered. The approach is quite similar, however there is no need to create an external *py* script: it is enough to retrieve the XGBoost built-in algorithm image where the algorithm is developed.

```
from sagemaker.amazon.amazon_estimator import get_image_uri

training_image = get_image_uri(sagemaker_session.boto_region_name,
                               'XGBoost', repo_version="latest")
```

Again an *Estimator* object is created; this time there is no need to have a SKLearn Estimator but a standard one, with the following python code:

```

algorithm = 'XGBoost'
output_mod = 'output location of the model'
train_input = 'input location of the training data'
validation_input = 'input location of the test data'
xgb_model = sagemaker.estimator.Estimator(training_image,
                                           role,
                                           train_instance_count=1,
                                           train_instance_type='ml.m5.large',
                                           train_volume_size = 20,
                                           train_max_run = 3600,
                                           input_mode= 'File',
                                           output_path=output_mod,
                                           sagemaker_session=sagemaker_session)

```

Finally, as before an hyperparameters tuning job is created and the best parameters configuration is used to create the model.

Once the two models are created (preprocessing + training job), they can be easily deployed in SageMaker to create an **inference pipeline**. Deploying a model in SageMaker requires two components [57]:

- **docker image**, i.e. a file to execute code in a Docker container;
- **model artifacts** residing in S3.

For the AWS Glue preprocessing phase, SageMaker team already provides a docker image. In order to correctly configure the container it is essential to pass the schema of input data whose prediction is requested. Instead of repeating the same action over and over, SageMaker allows to pass the schema during the model definition via an environment variable. Later it will show how it is possible to overwrite it for a single request.

```

import json

schema = {
    'input' :
        [{ 'name':column, "type": "double"}
          for column in numerical_features
        ]+
        [{ 'name':column, "type": "string"}
          for column in categorical_features
        ],
    'output':

```

```

        {
            'name': 'predictors',
            'type': 'double',
            'struct': 'vector'
        }
    }
    schema_json = json.dumps(schema)

```

Next step consists in creating a SageMaker *PipelineModel* with SparkML and the Training Algorithm. Both containers are deployed behind a single **API endpoint** in the specified order for real time predictions, but the same pipeline could be used also for a batch transform job.

```

from sagemaker.model import Model
from sagemaker.pipeline import PipelineModel
from sagemaker.sparkml.model import SparkMLModel

sparkml_data = 'location of the preprocessing model'
# passing the schema defined above by using an environment
# variable that sagemaker-sparkml-serving understands
sparkml_model = SparkMLModel(
    model_data=sparkml_data,
    env={'SAGEMAKER_SPARKML_SCHEMA' : schema_json})

timestamp_prefix = strftime("%Y-%m-%d-%H-%M-%S", gmtime())
model_name = 'prefix-' + timestamp_prefix

sm_model = PipelineModel(
    name=model_name,
    role=role,
    models=[sparkml_model, training_algorithm])
containers = sm_model.pipeline_container_def(
    instance_type = 'ml.m5.large')
sagemaker_session.create_model(model_name, role, containers)

```

In particular, the created model is deployed with the *deploy()* method to start an inference endpoint to make **real time** predictions: after that some requests could be sent to the endpoint to obtain predictions.

```

endpoint_name = 'prefix' + timestamp_prefix
sm_model.deploy(
    initial_instance_count=1,

```

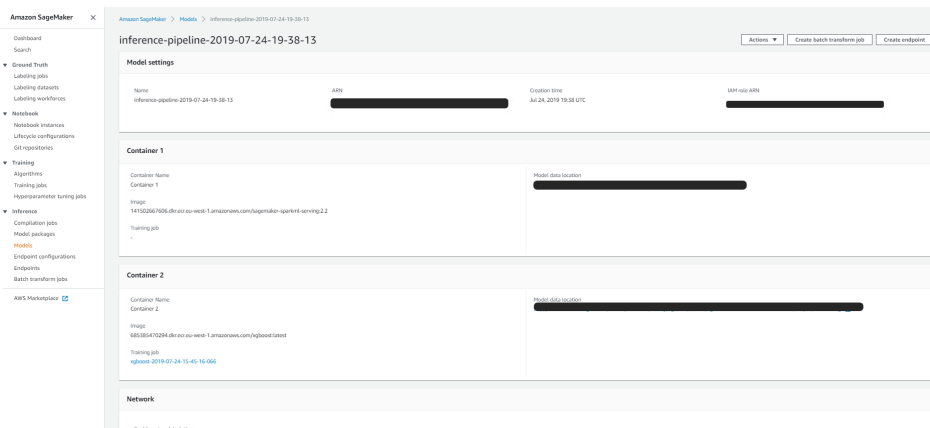



Figure 6.6: SageMaker: inspection pipeline model configuration

```

instance_type='ml.m5.large',
endpoint_name=endpoint_name)
#sm_client = boto_session.client('sagemaker')
#sm_client.delete_endpoint(EndpointName=endpoint_name)

```

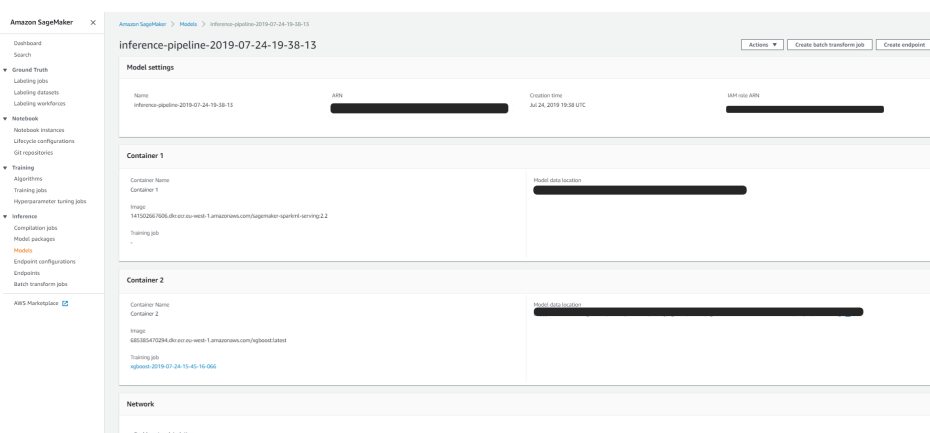


Figure 6.7: SageMaker: visualize endpoint configuration and performances

It is possible to invoke the created endpoint with a valid sample that SageMaker could recognize. Input sample could be passed in three different ways [57]:

- pass it as a valid CSV string, using as input schema the one passed before via environmental variable. Since it is in a CSV format, each input data column must be a basic datatype (such as string, int or double).

- pass it as a valid JSON string, using again as input schema the one passed before via environmental variable. Since it is in a JSON format, each input data column could be a basic datatype but also a Spark Vector or Array, assuming that the corresponding entry in the schema corresponds to the correct value in the input sample.
- pass the request in JSON format, composed by the schema and the data, to overwrite the possible existing schema passed before via environmental variable.

As explained before also SageMaker Batch Transform supports Pipeline model to perform a **batch transform job** to make predictions on a new set of data (batch prediction is similar to the real time one with the only difference that the former is performed on a set, the latter on one observation at a time). Batch Transform requires data in CSV or JSON format. In the first case for instance it is a CSV file similar to the training one, of course without the target field.

```
from sagemaker.content_types import CONTENT_TYPE_CSV

transformer = sm_model.transformer(
    # This was the model created using PipelineModel
    # and it contains feature processing and Decision tree
    instance_count = 1,
    instance_type = 'ml.m5.large',
    output_path = output_batch,
    accept = CONTENT_TYPE_CSV
)
transformer.transform(data = batch_input_loc,
                     job_name = job_name,
                     content_type = CONTENT_TYPE_CSV,
                     split_type = 'Line')

transformer.wait()
```

After transform job has completed, it is possible to inspect the results by downloading the output data from S3. For each file f in the input data, we have a corresponding file $f.out$ containing the predicted labels from each input row. Results are similar to one reported in Table 5.1.

6.2 Second step: multiclass classification

The second step of the pipeline consists in a multiclass classification algorithm to identify which could be the specific issue. In this regard, firstly, I would

like to make a brief comment: the process is roughly the same of the binary classification and only few small differences can be highlighted during the target generation and algorithm training. In particular, the new variable can assume three distinct values, according to our use cases:

```
sub_sample_wifi = raw_data_stratified
                    .filter(raw_data_stratified.target == 0)
sub_sample_slow = raw_data_stratified
                    .filter(raw_data_stratified.target == 1)
sub_sample_unstable = raw_data_stratified
                    .filter(raw_data_stratified.target == 2)
```

Regarding the training phase the algorithm is quite the same, however the metrics computation is slightly different; in fact the concept of precision explained before needed a multiclass adaptation.

First of all, the previous inspection of issues distribution showed that the considered target is unbalanced. Since there is a multilabel target, the precision is computed for each label finding its average weighted by support (the number of true instances for each label). This allows user to take into account label imbalance. The remaining part of the script is almost the same, so it would be redundant to present it again. Example of output is reported in Table 5.2.

Chapter 7

Experiments

Once the entire pipeline has been developed, performances could be summarized. Clearly, two different sets of results will be showed, one for the binary classification and one for the multiclass classification algorithm.

7.1 Binary classification

The powerful of the SageMaker automatic hyperparameter tuning job is adopted, in the sense that a range (grid) of values and a maximum number of jobs are specified and SageMaker tries to understand which could be the most promising hyperparameter combination; in this way not all values are tested but only the most relevant ones [40]. Due to the limited amount of time and resources, the entire pipeline has been tested on a small subset of 20 000 observations and splitting it in 80 – 20 train & test. The Jupyter Notebook used to retrieve the following results is provided with a *ml.t2.medium* instance type, i.e. with 2 vCPU and 4 Mem (GB).

Type	Data Preparation	Features Selection	Glue Preprocessing	
-	Time	Time	Time	DPU's used
Binary	40 Min	120 Min	8 Min	2
Multiclass	45 Min	120 Min	9 Min	2

Table 7.1: AWS Glue Preprocessing and Data Preparation: summarize the computational parameters.

After the data preparation phase, the AWS Glue Preprocessing is run to transform original data to the train and test datasets. More technical details about the job are provided in Table 7.1. In particular, DPU is defined as a data processing unit required to the ETL job and equipped with 4 vCPU and 16 GB of memory.

For the binary classification, as already explained before, the most interesting metric is the precision of the positive class (CALL). The optimization process is composed by two steps:

- algorithm parameters tuning, maximizing the number of true positive over the entire positive predictions. As threshold to decide if a specific test sample belongs to the positive or negative class the used value is 0.5, i.e. it is assign to the class it is more likely to belong (probability $>$ threshold = 0.5);
- since the model output consists in a probability, it is possible to change dynamically threshold's value to reach the required trade-off between positive class precision and recall (probability $>$ threshold).

This approach comes from the design choice of keeping more control on the threshold, allowing to change its value on-the-fly if model performances, constantly monitored, fall under a specific value. For this reason the reader must interpret the first step only as a discovery one to become more confident about algorithm behaviour and metric possible values range. The second step is the crucial one, since it is possible to balance model's positive class precision and recall by considering different threshold's values. As already explained before, three are the considered algorithms [59]:

- **decision tree**. The parameters considered for tuning jobs are:
 - the **criterion** used to measure the quality of a split, where considered values are the Gini index and cross-entropy.
 - the **maximum depth** of the tree. If user does not specify any value, new splits are considered until leaves contain observations with the same label or a number of observations less than a fixed threshold (next parameter);
 - the **minimum** number of samples which must be contained in each **leaf node**. A new partition is performed if and only if the two branches generated contain at least this minimum number of observations; the default value is 1.

From what concerns Figure 7.1, on the top Gini split criterion is used, on the left the minimum number of observations in each leaf is **not**

set (*default* = 1), while on the right the maximum depth is **not** set (*default* = *None*). A similar approach is used on the bottom, but the split criterion considered is entropy. From this figure and Table 7.2 (where the token “-” means that the considered parameter is not set.) it is possible to inspect the algorithm behaviour, with respect to the precision metric, when a specific parameter varies. In particular, the maximum tree depth seems to have no effect on the precision metric (not considering extreme values); clearly while the minimum number of samples contained in leaf nodes increases the model precision goes down since positive class is less predicted. Finally, changing the split criterion from Gini to entropy, the precision metric does not change so much.

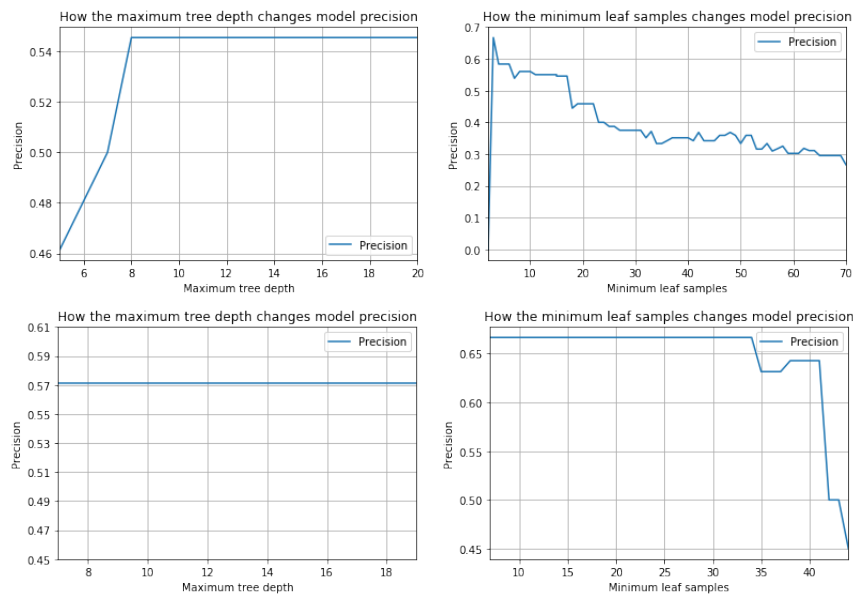


Figure 7.1: Binary classification: extraction of decision tree tuning job.

- **random forest.** The parameters considered for tuning jobs are:
 - the **number** of **trees** in the forest;
 - the **criterion** used to measure the quality of a split in each tree, where considered values are the Gini index and cross-entropy.
 - the **maximum depth** of each tree is **not** set and the **minimum** number of samples in each **leaf** is set to the previous optimal value;

Attempt	Criterion	Max depth	Minimum leaf	Precision
1	gini	-	4	0.58
2	gini	-	23	0.40
3	gini	-	56	0.31
4	gini	5	-	0.46
5	gini	14	-	0.545
6	gini	19	-	0.545
7	entropy	-	10	0.67
8	entropy	-	30	0.67
9	entropy	-	45	0.45
10	entropy	7	-	0.57
11	entropy	14	-	0.57
12	entropy	19	-	0.57

Table 7.2: Binary classification: example of decision tree tuning job.

For what concerns Figure 7.2, by keeping the optimal decision tree configuration, the number of estimators is changed to inspect precision metric reaction using Gini index (left) and entropy (right) split criterion. For what concerns Table 7.3, the other previous single tree parameters not considered in the table are set to the best configuration found before.

From the previous Figure and Table it is possible to inspect the algorithm behaviour, with respect to the precision metric, when the number of trees in the forest changes. Random forest shows better results than decision tree, which is a consistent result according to the literature presented before. The interesting part consists now in comparing the obtained results with the XGBoost algorithm, which could be considered as an alternative solution having the great advantage of being an Amazon built-in algorithm and for this reason easier to insert inside the pipeline.

- **XGBoost.** According to the official documentation the hyperparameters having the most relevant effect on the XGBoost objective metrics are [41]:

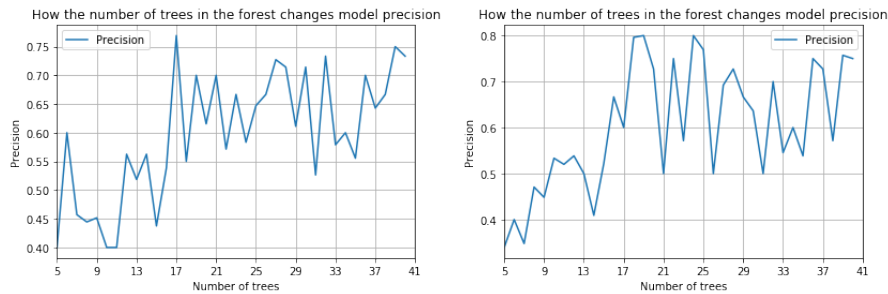


Figure 7.2: Binary classification: extraction of random forest tuning job.

Attempt	Criterion	Number estimators	Precision
1	gini	6	0.60
2	gini	17	0.76
3	gini	27	0.73
4	entropy	6	0.40
5	entropy	18	0.79
6	entropy	28	0.73

Table 7.3: Binary classification: example of random forest tuning job.

- the weights **$L1$ regularization** term, which makes the algorithm more or less conservative;
- the **minimum sum of instance weight** needed in a child. If tree partition creates a leaf where the *sum of instance weight* becomes less than the threshold, no other partitions are performed. If this value increases, algorithm becomes more conservative;
- the **subsample ratio** of the training instance, useful to avoid overfitting. The default value is 0.5 in the sense that only half of the total observations is randomly considered to build trees.
- the **step size** shrinkage used in each boosting step. This parameter shrinks the feature weights inducing the boosting process to be more conservative and trying to avoid overfitting;
- the **number of rounds** (iterations) in the training phase.
- **colsample_bylevel** and **colsample_bytree**, which oblige the algorithm to consider only a subsample of columns for a single split

or an entire tree, are **not set** since a features selection has already performed.

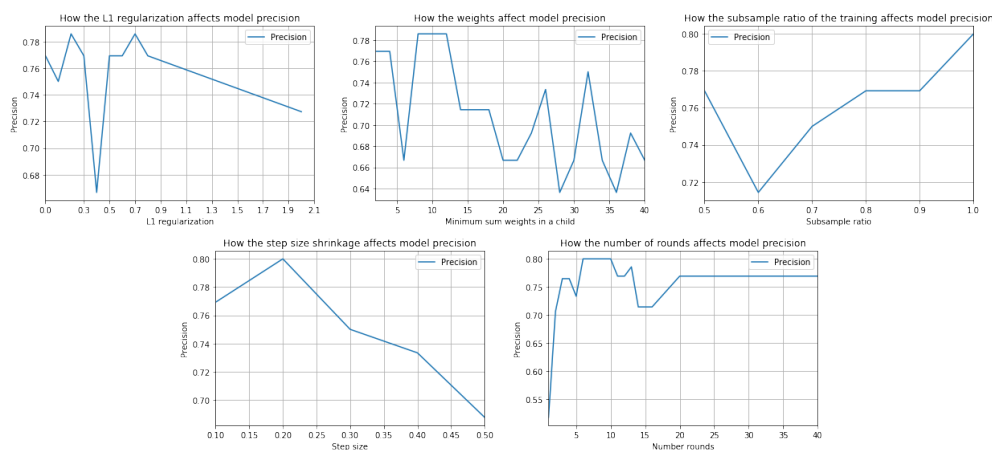


Figure 7.3: Binary classification: extraction of XGBoost tuning job.

For what concerns Figure 7.3, from the top left to the bottom right the parameters inspected to test the model precision are the L1 regularization, the minimum sum of instance weight, the subsample fraction, the step size and the number of rounds. From this Figure and Table 7.4 it is possible to inspect the algorithm behavior, with respect to the precision metric, when a specific parameter varies. Some parameters show a monotone behavior, such as the ratio of the training set used to fit the model and the step size shrinkage, while others are characterized by a less controllable trend. The performances are quite similar to the random forest algorithm, but since it is a SageMaker built-in solution, at the end, **XGBoost** algorithm is adopted for the binary first classification.

Table 7.5 helps to summarize the technical details about the training phase. XGBoost required more powerful instances than Random Forest and Decision Tree, but differently from the others the former can distribute the training on more than one single instance.

Once completed this preliminary choice, it comes the solution's crucial step where the required trade-off between precision and accuracy has to be found. After selecting XGBoost model with the best configuration found before, *L1 regularization = 0*, *minimum sum of instance weight = 10*, *subsample ratio = 1*, *step size = 0.2*, *number of rounds = 10*, the threshold's value is varied to analyze precision and recall behavior. The inspection of Figure 7.4 confirms the conventional output, according to the literature. When the threshold is

Attempt	L1	Child weight	Subsample	Step	Rounds	Precision
1	0.0	8	0.7	0.2	10	0.79
2	0.4	8	0.7	0.2	10	0.67
3	0.8	8	0.7	0.2	10	0.77
4	0	6	0.7	0.2	10	0.67
5	0	10	0.7	0.2	10	0.79
6	0	18	0.7	0.2	10	0.71
7	0	10	0.6	0.2	10	0.71
8	0	10	0.8	0.2	10	0.77
9	0	10	1	0.2	10	0.8
10	0	10	1	0.2	10	0.8
11	0	10	1	0.3	10	0.75
12	0	10	1	0.5	10	0.69
13	0	10	1	0.2	4	0.76
14	0	10	1	0.2	10	0.8
15	0	10	1	0.2	20	0.77

Table 7.4: Binary classification: example of XGBoost tuning job.

small, near 0, the positive class is quite always predicted, which implies a small precision (due to the high number of false positives) and an high recall (due to the absence or small number of false negatives). While threshold increases, the recall falls quickly and the precision grows, since the positive class is less predicted; the extremely rapid process of change of those metrics depends on the unbalanced dataset, which causing the positive class to be harder to be expected: the final choice is to set **0.55** as threshold's value.

7.2 Multiclass classification

For the multiclass classification step the most interest metric is the weighted precision, based on the number of true instances for each label. The optimization process is composed by the same two sub-steps, this time considering a multiclass classification problem instead of a binary one. Only random for-

Algorithm	Time Tr	vCPU	Mem (GB)	Num Inst	Distr Train
Decision Tree	19 Min	2	8	1	✗
Random Forest	30 Min	2	8	1	✗
XGBoost	19 Min	4	16	2	✓

Table 7.5: Binary classification: summarize the training details.

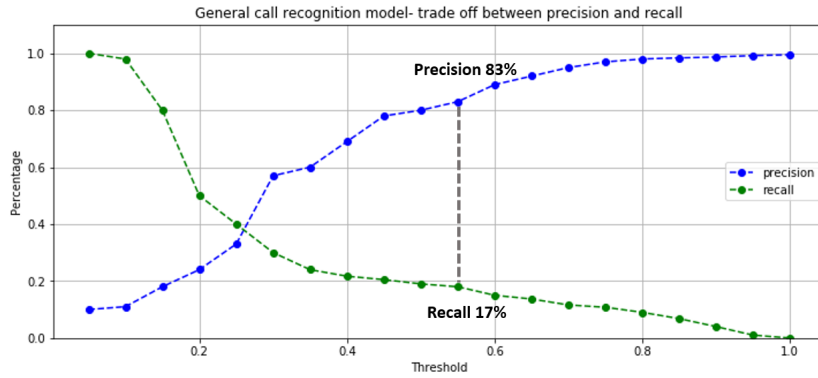


Figure 7.4: Binary classification: trade off between positive class precision and recall in the general call recognition model.

est and XGBoost have been analyzed since they showed better performances with respect to the decision tree.

Figure 7.5 (the number of estimator is changed to inspect precision metric reaction using Gini index (left) and entropy (right) split criterion) shows how the former makes random forest performances more stable than entropy, Figure 7.6 (from the top left to the bottom the parameters inspected to test the model precision are the L1 regularization, the minimum sum of instance weight, the subsample fraction, the step size and the number of rounds) instead shows how the XGBoost parameters do not affect model's capability to predict the single type of issue, confirmed by the precision whose value is always around 63%. For the multiclass classification step, the random forest performances seem to be superior with respect to the XGBoost algorithm and, by manually inspected the single class predictions, the problem seems to be caused by the algorithm inability to recognize the slow line use-case. Therefore, once the **random forest** model is selected with the configuration *Number of trees = 30* and *Criterion = gini*, the confusion matrix is reported in Figure 7.7 to better understand model performances using the

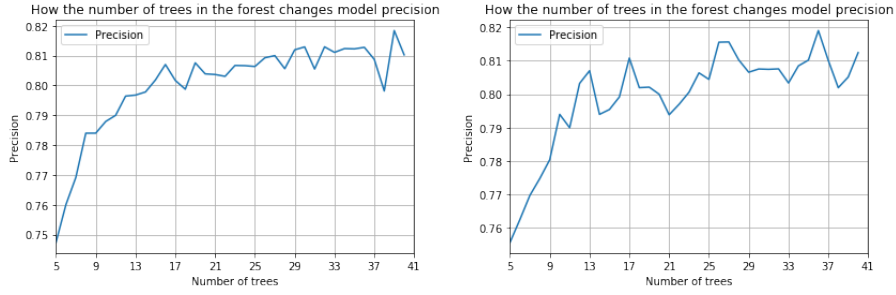


Figure 7.5: Multiclass classification: extraction of random forest tuning job.

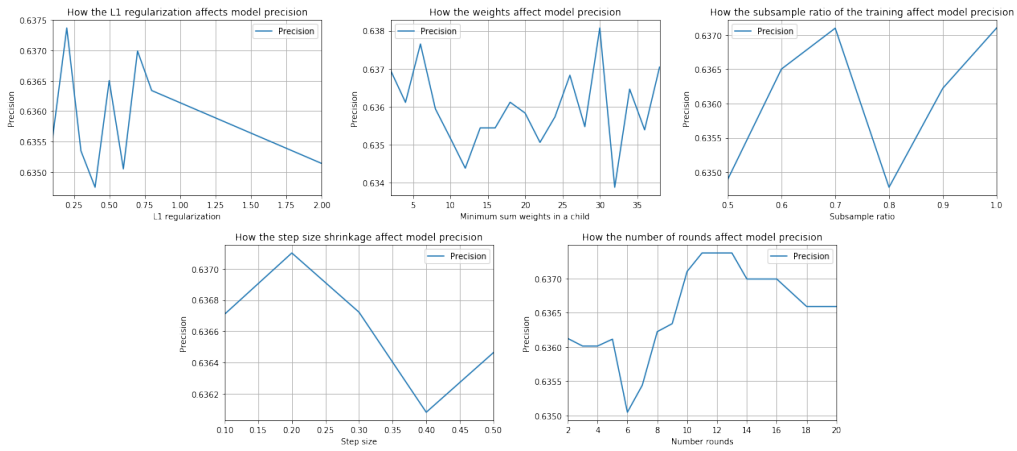


Figure 7.6: Multiclass classification: extraction of XGBoost tuning job.

default threshold (each observation is assigned to the class it is more likely to belong): all the three different types of issue result to be well classified, even if the *Wi-Fi issue* use-case is the most common one, ten times more frequent than the others. For this reason, since probability calibration for 3-class classification is not so intuitive, the **default** threshold is kept.

Table 7.6 helps to summarize the technical details about the multiclass training phase.

7.3 Results

To summarize, at the end the developed pipeline is able to detect **call** event with a precision of more than 80% and, even if the recall metric is not so high, it allows the client to anticipate a small subset of problems before happening. The model is also able to satisfactorily classify the different types of problem (Wifi, Unstable line and Slow line) with a weighted precision of

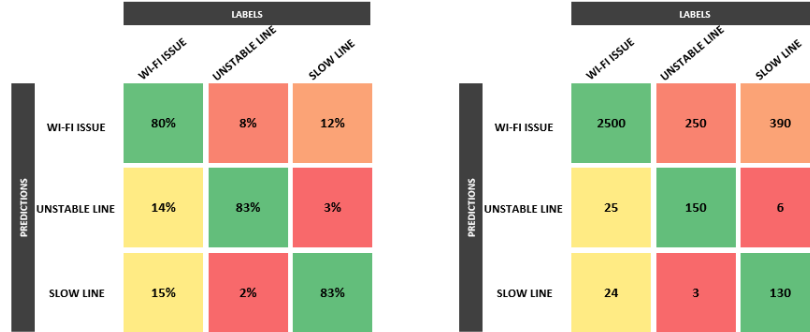


Figure 7.7: Multiclass classification: confusion matrix, on the left normalized by class total predictions, on the right not normalized.

Algorithm	Time Tr	vCPU	Mem (GB)	Num Inst	Distr Train
Random Forest	33 Min	2	8	1	✗
XGBoost	21 Min	4	16	2	✓

Table 7.6: Multiclass classification: summarize the training details.

80%, convincing client to be able to detect line problems. The results obtained confirm that the solution has been correctly realized and it could be submitted to the production environment.

Therefore, the pipeline model is created (Table 7.7 for more details) and it could be deployed to an inference endpoint for real time predictions or it could be used for a batch transform job to make inferences on an entire set of new observations (Table 7.8 where for both real time and batch predictions the same type and number of instances are used, but user could specify different parameters configurations): for the transform job the fresh set of data considered in this case is composed by 100 observations (the table does not take into account endpoint keeping efforts).

Finally, a brief comment about cost analysis must be done: remembering that Amazon Web Services are on-demand services, the cost is restricted to the time where resources are required. Table 7.9 summarizes the fixed cost, from the initialization, considering the pre-processing step, until the deploying phase but not considering the one to keep the endpoint ready to real time inference. It is important to remember that cost related on tuning

Type	Model creation
-	Time
Binary	< 1 Min
Multiclass	< 1 Min

Table 7.7: AWS Model Creation: summarize the computational parameters to create the pipeline model from the trained algorithms.

Type	Prediction				
-	Parameters			Endpoint Creat	Batch Pred
-	vCPU	Mem (GB)	Num Inst	Time	Time
Binary	4	16	2	12 Min	28 Min
Multiclass	4	16	2	15 Min	32 Min

Table 7.8: AWS Prediction: summarize the endpoint creation and batch transform job computational parameters.

job depends on the maximum number of jobs the user decides to execute to optimize model performances. Table 7.10, instead, shows cost to keep the endpoint running which depends on the resources assigned to it during the creation phase (in this case 4 *vCPU* and 16 *GB* of memory)

Type	Glue Prep.	Tuning	Model	Batch	Notebook
-	Cost	Cost	Cost	Cost	Cost
Binary	\$ 0.6	\$ 7	\$ 0.3	\$ 0.4	\$ 47
Multiclass	\$ 0.67	\$ 8.6	\$ 0.4	\$ 0.5	\$ 49

Table 7.9: Cost analysis: summarize the fixed cost details.

Type	Endpoint
-	Cost/Time
Binary	0.60 \$/h
Multiclass	0.60 \$/h

Table 7.10: Cost analysis: summarize the costs to keep the endpoint ready for real time inference.

Chapter 8

Model retraining

Remembering that after having trained a model, it is possible to obtain accurate predictions only if the test dataset has a distribution similar to the set used to train the considered model. However, it can be expected that data distribution changes over time, drifting from one configuration to another: for this reason model deployment is not a static approach but a continuous process. It is essential to continuously check most recent data and considering a model retraining if distribution changes a lot from the initial one [21]. Two different approaches are explored: the first one is the simpler and current adopted strategy of training the model periodically, daily, weekly or monthly depending on the task, the second one consists in a deeper drift evaluation to optimize model retraining and reduce costs. The first approach is reported in section 8.1 where Apache Airflow is adopted, while the other sections present the second approach.

Obviously, the model retraining has some limitations, such as the impossibility to inspect manually the trade-off between precision and recall to set the ideal threshold in the binary case. Theoretically, user could inspect re-trained model prediction on a test set to avoid this issue. Anyway, even if the threshold is not set to the optimal value, the model is able to guarantee good performance both in the binary and multiclass classification step.

8.1 Apache Airflow

Basically, the idea is to retrain the model in Amazon ML based on the new training data: to perform this task **Apache Airflow** is adopted. From the official documentation the tool is defined as a *platform to schedule and monitor workflows* [43]. Airflow's basic concept is the directed acyclic graph (DAG), i.e. a graph without any cycles and composed by a finite number of

vertices and edges, where each edge connects two vertices in a single direction and it does not exist a path to come back to the first vertex (otherwise a cycle is created).

DAG vertices are called *operators*: an operator represents a particular task which needs to be carried out. Different operators are suitable for different tasks, in fact according to the official documentation there are some different predefined types available [21]:

- **BashOperator** allows to execute a bash command;
- **PythonOperator** allows to call a python function;
- **SimpleHttpOperator** to send an HTTP request;
- A list of **SqlOperators** allow to execute some of the most common SQL commands.

On the website it is possible to inspect which are the key capabilities of Apache Airflow:

- **dynamic**: Airflow operations can be written in python language, allowing for a more dynamic pipeline creation;
- **extensible**: it is possible to create customized and ad-hoc operators and import new libraries to generate the required level of abstraction for the developing solution;
- **elegant**: Airflow pipelines are *lean*, *explicit* and easily to understand. Thanks to user interface it is easy to visualize pipelines running and workflows monitoring;
- **scalable**: Airflow has provided with an architecture based on modules and with an efficient message queue helping system in managing an elevate number of operators, without any upper bounds.

Airflow pipeline is simply a Python script aims to create an Airflow DAG object. After defining a dictionary of default parameters values which can be shared between all the operators, a DAG object is created to nest the tasks into. A *dag_id* is defined representing the unique identifier for the DAG: the previous dictionary defines the default arguments. Generate project's tasks implies create operators by calling the relative constructor method: the *task_id* attribute represents the task identifier and it must be unique. To associate each task to the previous configured DAG, it is enough to set the task argument *dag* to the DAG's id. Notice it is possible to assign as

parameters to the operator's constructor a mix of specific arguments and arguments common to all operators, inherited from `BaseOperator` [43].

Before presenting the set of task developed for the predictive maintenance project, it is important to understand that the tasks run in a different context from the context of the main script. A specific operator runs totally independent from each other, which implies that it is impossible for them to exchange information by default: to perform this task there exists a more advanced feature called *XCom*. Airflow needs to know how to connect to the environment: the Amazon Web Services connection type enables the AWS integration. The authentication may be performed using any of the boto3 python library options.

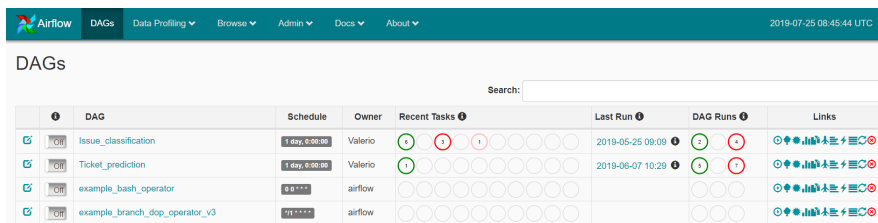
The tasks created are the following ones:

- **PythonOperator** which starts the pipeline printing a message of starting;
- **PythonOperator** which deletes all the objects related to the old solution to be retrained;
- **PythonOperator** which deletes the old endpoint to avoid useless costs;
- **BashOperator** which performs data preprocessing, such that missing data imputation;
- **AWSGlueJobOperator**, a customized operator created to perform an AWS Glue job;
- **BranchPythonOperator**, a customized operator to branch the process based on a parameter value, retrieving from a configuration file; two different paths are available: training job or hyperparameter tuning job;
- **PythonOperator** after connecting via boto3 to AWS SageMaker, it starts a training job;
- **PythonOperator** after connecting via boto3 to AWS SageMaker, it starts a hyperparameter tuning job;
- **PythonOperator** after connecting via boto3 to AWS SageMaker, it creates the entire pipeline and deploys it;

- **PythonOperator** which ends the pipeline printing a message of ending;

The Airflow user interface could be used to easily control the data pipeline: a brief overview of some Airflow UI available tools are shown below.

The home page (Figure 8.1) shows the list of available DAGs in the environment, allowing to quickly check exactly tasks status: succeeded, failed, running, skipped, up for retry, queued, etc. Right panels help user to interact with the pipeline by triggering a DAG, showing details and finally refreshing and deleting a DAG.



DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
Issue_classification	1 day, 0:00:00	Valerio	1 failed, 1 skipped, 1 success	2019-05-25 09:09	1 failed, 1 success	Trigger DAG, Details, Code, Refresh, Delete
Ticket_prediction	1 day, 0:00:00	Valerio	1 success	2019-06-07 10:29	1 success	Trigger DAG, Details, Code, Refresh, Delete
example_bash_operator	1 day, 0:00:00	airflow	1 success			Trigger DAG, Details, Code, Refresh, Delete
example_branch_dop_operator_v3	1 day, 0:00:00	airflow	1 success			Trigger DAG, Details, Code, Refresh, Delete

Figure 8.1: Airflow list of DAGs

Focusing on a single DAG some different graphical views are available in Airflow: a tree representation (Figure 8.2) which spans across time. If the process is late or an error occurs, tree view allows to rapidly control tasks status and identify the blocking or failed ones. However if a DAG with a lot of tasks is considered, maybe the tree could appear a bit cryptic.

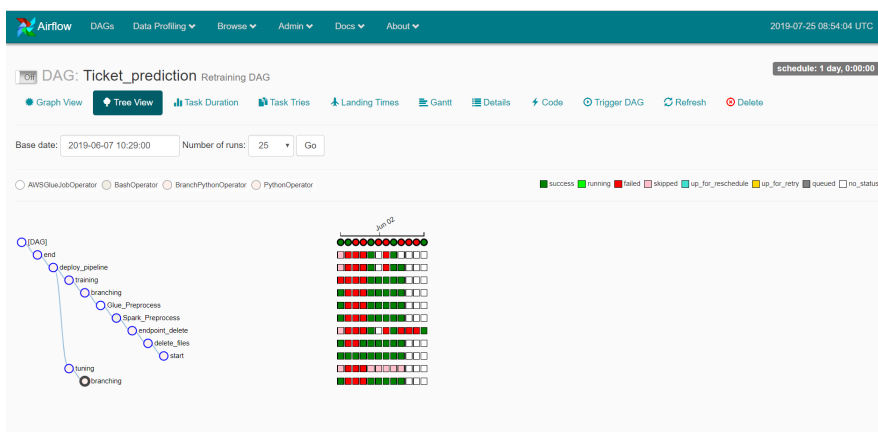


Figure 8.2: Airflow tree visualization

Another possible graphical representation is the graph view (Figure 8.3), which is probably the most intuitive one. It allows user to easily visualize DAG's path and tasks current status.

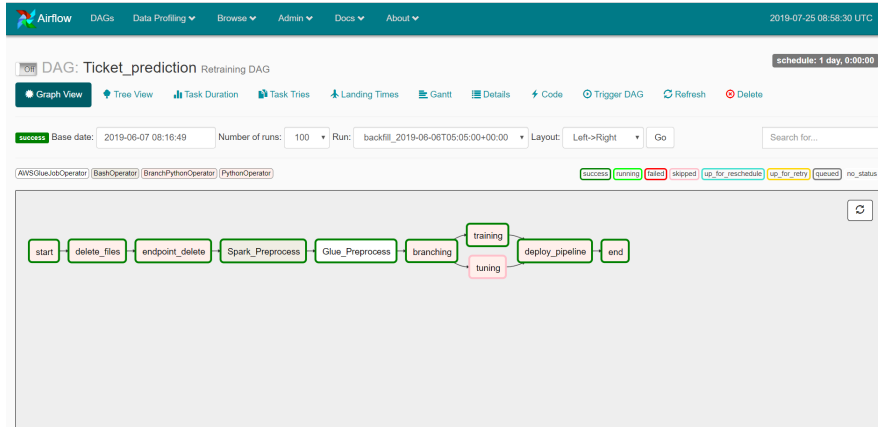


Figure 8.3: Airflow graph visualization

In the variable section (Figure 8.4) user has the possibility to edit the key-value pair of a variable which will be used in the pipeline. There is also the possibility to hidden a variable value if it is confidential.

The screenshot shows the 'Variables' section of the Apache Airflow web interface. It has a table with columns 'Key' and 'Val'. The table contains several rows of variables, each with a checkbox, an edit icon, and a delete icon. The 'Val' column shows the values of the variables, some of which are masked with '*****'.

	Key	Val
<input type="checkbox"/>	secret_password	*****
<input type="checkbox"/>	not_so_hidden	test value
<input type="checkbox"/>	secret	*****
<input type="checkbox"/>	password	*****
<input type="checkbox"/>	passwd	*****
<input type="checkbox"/>	api_key	*****
<input type="checkbox"/>	apikey	*****
<input type="checkbox"/>	authorization	*****
<input type="checkbox"/>	access_token	*****

Figure 8.4: Airflow variable view

Airflow provides also user with the possibility to monitor tasks duration and overlap, thank to a Gantt chart (Figure 8.5). Using this chart the user could identify bottlenecks, i.e. where the large amount of time is spent for each DAG run.

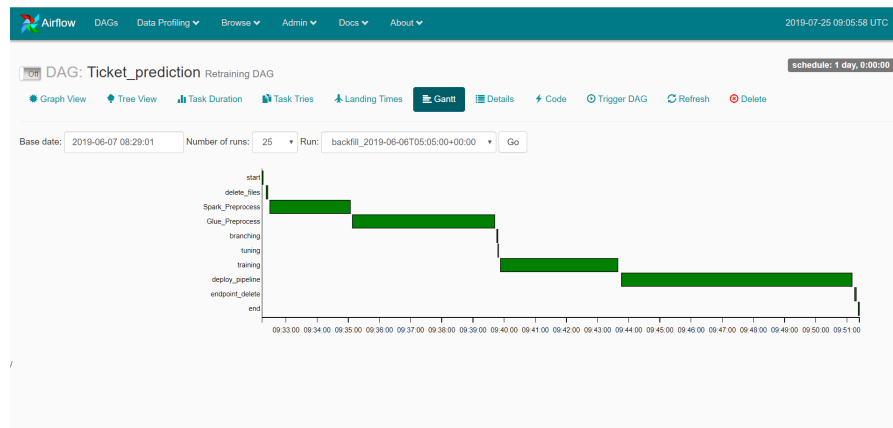


Figure 8.5: Airflow Gantt chart

There is also the chance to consider tasks duration in the previous N runs (Figure 8.6). This view allows to discover outliers and check where and how much time the specific DAG required during latest runs.

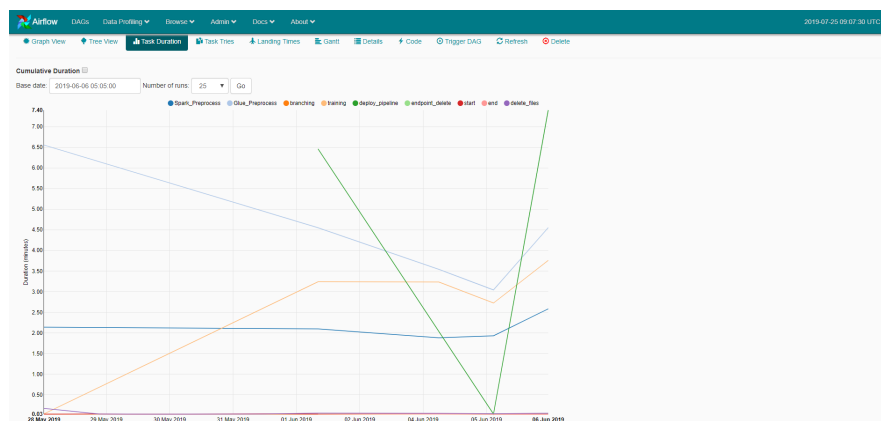


Figure 8.6: Airflow task duration

A code view is available (Figure 8.7), avoiding user to have to check manually on the source code if some unusual behaviours happen.

From all the previous pages (tree page, graph page, gantt chart, ...), there is the possibility to explore the task instance, clicking on it, to access to the more detailed context menu (Figure 8.8) useful to execute some operations.

Finally Airflow gives the possibility to send email alerts when some jobs

The screenshot shows the Apache Airflow web interface with the 'Ticket_prediction' DAG selected. The 'Code' tab is active, displaying the DAG's Python code. The code includes imports for various Airflow operators and functions, and defines a task named 'Ticket_prediction'.

```

1 # airflow operators
2 from airflow.models import DAG
3 from airflow.contrib.operators.trigger_dagrun_operator import TriggerDagRunOperator
4 from airflow.operators.python_operator import BranchPythonOperator
5 from airflow.operators.python_operator import PythonOperator
6 from airflow.operators.bash_operator import BashOperator
7 # airflow executor operators
8 from airflow.contrib.operators.aws_batch import AwsBatchOperator
9
10 # sagemaker utils
11 import boto3
12 import sagemaker
13 from sagemaker.sklearn.estimator import SKLearn
14 from sagemaker.kubernetes import KubernetesOperator
15 from sagemaker.model import Model
16 from sagemaker.pipeline import PipelineModel
17 from sagemaker.sagemaker_model import SageMakerModel
18 from sagemaker.sklearn.model import SKLearnModel
19 from sagemaker.estimator import Estimator
20
21 # other libraries
22 import pandas as pd
23 from time import time, strftime
24 from datetime import datetime, timedelta
25 import sys
26 import os
27 sys.path.insert(0, os.path.join(os.getcwd(), 'utils'))
28 from airflow.utils.decorators import apply_defaults
29
30 # functions
31
32 def is_hpo_enabled():
33     """Check if hyper-parameter optimization is enabled in the config"""
34     hpo_enabled = False
35     return hpo_enabled
36
37 if __name__ == '__main__':
38     """Main function"""
39
40     # DAG definition
41     dag = DAG(
42         'Ticket_prediction',
43         default_args={
44             'owner': 'airflow',
45             'depends_on_past': False,
46             'email': ['airflow@example.com'],
47             'email_on_failure': True,
48             'email_on_retry': False,
49             'retries': 1,
50         },
51         schedule_interval='1 day',
52         start_date=datetime(2019, 7, 25),
53         catchup=False,
54     )
55
56     # Task definition
57     task = Ticket_prediction(
58         task_id='Ticket_prediction',
59         python_callable=Ticket_prediction,
60         dag=dag,
61     )
62
63     task.run()
64
65     # DAG execution
66     dag.run()
67
68     # DAG completion
69     dag.complete()
70
71     # DAG deletion
72     dag.delete()
73
74     # DAG creation
75     dag.create()
76
77     # DAG update
78     dag.update()
79
80     # DAG list
81     dag.list()
82
83     # DAG info
84     dag.info()
85
86     # DAG status
87     dag.status()
88
89     # DAG logs
90     dag.logs()
91
92     # DAG metrics
93     dag.metrics()
94
95     # DAG alerts
96     dag.alerts()
97
98     # DAG notifications
99     dag.notifications()
100
101     # DAG settings
102     dag.settings()
103
104     # DAG configuration
105     dag.configuration()
106
107     # DAG documentation
108     dag.documentation()
109
110     # DAG help
111     dag.help()
112
113     # DAG about
114     dag.about()
115
116     # DAG version
117     dag.version()
118
119     # DAG license
120     dag.license()
121
122     # DAG terms
123     dag.terms()
124
125     # DAG privacy
126     dag.privacy()
127
128     # DAG security
129     dag.security()
130
131     # DAG compliance
132     dag.compliance()
133
134     # DAG governance
135     dag.governance()
136
137     # DAG audit
138     dag.audit()
139
140     # DAG monitoring
141     dag.monitoring()
142
143     # DAG logging
144     dag.logging()
145
146     # DAG debugging
147     dag.debugging()
148
149     # DAG testing
150     dag.testing()
151
152     # DAG deployment
153     dag.deployment()
154
155     # DAG migration
156     dag.migration()
157
158     # DAG backup
159     dag.backup()
160
161     # DAG restore
162     dag.restore()
163
164     # DAG backup and restore
165     dag.backup_and_restore()
166
167     # DAG backup and restore
168     dag.backup_and_restore()
169
170     # DAG backup and restore
171     dag.backup_and_restore()
172
173     # DAG backup and restore
174     dag.backup_and_restore()
175
176     # DAG backup and restore
177     dag.backup_and_restore()
178
179     # DAG backup and restore
180     dag.backup_and_restore()
181
182     # DAG backup and restore
183     dag.backup_and_restore()
184
185     # DAG backup and restore
186     dag.backup_and_restore()
187
188     # DAG backup and restore
189     dag.backup_and_restore()
190
191     # DAG backup and restore
192     dag.backup_and_restore()
193
194     # DAG backup and restore
195     dag.backup_and_restore()
196
197     # DAG backup and restore
198     dag.backup_and_restore()
199
200     # DAG backup and restore
201     dag.backup_and_restore()
202
203     # DAG backup and restore
204     dag.backup_and_restore()
205
206     # DAG backup and restore
207     dag.backup_and_restore()
208
209     # DAG backup and restore
210     dag.backup_and_restore()
211
212     # DAG backup and restore
213     dag.backup_and_restore()
214
215     # DAG backup and restore
216     dag.backup_and_restore()
217
218     # DAG backup and restore
219     dag.backup_and_restore()
220
221     # DAG backup and restore
222     dag.backup_and_restore()
223
224     # DAG backup and restore
225     dag.backup_and_restore()
226
227     # DAG backup and restore
228     dag.backup_and_restore()
229
230     # DAG backup and restore
231     dag.backup_and_restore()
232
233     # DAG backup and restore
234     dag.backup_and_restore()
235
236     # DAG backup and restore
237     dag.backup_and_restore()
238
239     # DAG backup and restore
240     dag.backup_and_restore()
241
242     # DAG backup and restore
243     dag.backup_and_restore()
244
245     # DAG backup and restore
246     dag.backup_and_restore()
247
248     # DAG backup and restore
249     dag.backup_and_restore()
250
251     # DAG backup and restore
252     dag.backup_and_restore()
253
254     # DAG backup and restore
255     dag.backup_and_restore()
256
257     # DAG backup and restore
258     dag.backup_and_restore()
259
260     # DAG backup and restore
261     dag.backup_and_restore()
262
263     # DAG backup and restore
264     dag.backup_and_restore()
265
266     # DAG backup and restore
267     dag.backup_and_restore()
268
269     # DAG backup and restore
270     dag.backup_and_restore()
271
272     # DAG backup and restore
273     dag.backup_and_restore()
274
275     # DAG backup and restore
276     dag.backup_and_restore()
277
278     # DAG backup and restore
279     dag.backup_and_restore()
280
281     # DAG backup and restore
282     dag.backup_and_restore()
283
284     # DAG backup and restore
285     dag.backup_and_restore()
286
287     # DAG backup and restore
288     dag.backup_and_restore()
289
290     # DAG backup and restore
291     dag.backup_and_restore()
292
293     # DAG backup and restore
294     dag.backup_and_restore()
295
296     # DAG backup and restore
297     dag.backup_and_restore()
298
299     # DAG backup and restore
300     dag.backup_and_restore()
301
302     # DAG backup and restore
303     dag.backup_and_restore()
304
305     # DAG backup and restore
306     dag.backup_and_restore()
307
308     # DAG backup and restore
309     dag.backup_and_restore()
310
311     # DAG backup and restore
312     dag.backup_and_restore()
313
314     # DAG backup and restore
315     dag.backup_and_restore()
316
317     # DAG backup and restore
318     dag.backup_and_restore()
319
320     # DAG backup and restore
321     dag.backup_and_restore()
322
323     # DAG backup and restore
324     dag.backup_and_restore()
325
326     # DAG backup and restore
327     dag.backup_and_restore()
328
329     # DAG backup and restore
330     dag.backup_and_restore()
331
332     # DAG backup and restore
333     dag.backup_and_restore()
334
335     # DAG backup and restore
336     dag.backup_and_restore()
337
338     # DAG backup and restore
339     dag.backup_and_restore()
340
341     # DAG backup and restore
342     dag.backup_and_restore()
343
344     # DAG backup and restore
345     dag.backup_and_restore()
346
347     # DAG backup and restore
348     dag.backup_and_restore()
349
350     # DAG backup and restore
351     dag.backup_and_restore()
352
353     # DAG backup and restore
354     dag.backup_and_restore()
355
356     # DAG backup and restore
357     dag.backup_and_restore()
358
359     # DAG backup and restore
360     dag.backup_and_restore()
361
362     # DAG backup and restore
363     dag.backup_and_restore()
364
365     # DAG backup and restore
366     dag.backup_and_restore()
367
368     # DAG backup and restore
369     dag.backup_and_restore()
370
371     # DAG backup and restore
372     dag.backup_and_restore()
373
374     # DAG backup and restore
375     dag.backup_and_restore()
376
377     # DAG backup and restore
378     dag.backup_and_restore()
379
380     # DAG backup and restore
381     dag.backup_and_restore()
382
383     # DAG backup and restore
384     dag.backup_and_restore()
385
386     # DAG backup and restore
387     dag.backup_and_restore()
388
389     # DAG backup and restore
390     dag.backup_and_restore()
391
392     # DAG backup and restore
393     dag.backup_and_restore()
394
395     # DAG backup and restore
396     dag.backup_and_restore()
397
398     # DAG backup and restore
399     dag.backup_and_restore()
400
401     # DAG backup and restore
402     dag.backup_and_restore()
403
404     # DAG backup and restore
405     dag.backup_and_restore()
406
407     # DAG backup and restore
408     dag.backup_and_restore()
409
410     # DAG backup and restore
411     dag.backup_and_restore()
412
413     # DAG backup and restore
414     dag.backup_and_restore()
415
416     # DAG backup and restore
417     dag.backup_and_restore()
418
419     # DAG backup and restore
420     dag.backup_and_restore()
421
422     # DAG backup and restore
423     dag.backup_and_restore()
424
425     # DAG backup and restore
426     dag.backup_and_restore()
427
428     # DAG backup and restore
429     dag.backup_and_restore()
430
431     # DAG backup and restore
432     dag.backup_and_restore()
433
434     # DAG backup and restore
435     dag.backup_and_restore()
436
437     # DAG backup and restore
438     dag.backup_and_restore()
439
440     # DAG backup and restore
441     dag.backup_and_restore()
442
443     # DAG backup and restore
444     dag.backup_and_restore()
445
446     # DAG backup and restore
447     dag.backup_and_restore()
448
449     # DAG backup and restore
450     dag.backup_and_restore()
451
452     # DAG backup and restore
453     dag.backup_and_restore()
454
455     # DAG backup and restore
456     dag.backup_and_restore()
457
458     # DAG backup and restore
459     dag.backup_and_restore()
460
461     # DAG backup and restore
462     dag.backup_and_restore()
463
464     # DAG backup and restore
465     dag.backup_and_restore()
466
467     # DAG backup and restore
468     dag.backup_and_restore()
469
470     # DAG backup and restore
471     dag.backup_and_restore()
472
473     # DAG backup and restore
474     dag.backup_and_restore()
475
476     # DAG backup and restore
477     dag.backup_and_restore()
478
479     # DAG backup and restore
480     dag.backup_and_restore()
481
482     # DAG backup and restore
483     dag.backup_and_restore()
484
485     # DAG backup and restore
486     dag.backup_and_restore()
487
488     # DAG backup and restore
489     dag.backup_and_restore()
490
491     # DAG backup and restore
492     dag.backup_and_restore()
493
494     # DAG backup and restore
495     dag.backup_and_restore()
496
497     # DAG backup and restore
498     dag.backup_and_restore()
499
500     # DAG backup and restore
501     dag.backup_and_restore()
502
503     # DAG backup and restore
504     dag.backup_and_restore()
505
506     # DAG backup and restore
507     dag.backup_and_restore()
508
509     # DAG backup and restore
510     dag.backup_and_restore()
511
512     # DAG backup and restore
513     dag.backup_and_restore()
514
515     # DAG backup and restore
516     dag.backup_and_restore()
517
518     # DAG backup and restore
519     dag.backup_and_restore()
520
521     # DAG backup and restore
522     dag.backup_and_restore()
523
524     # DAG backup and restore
525     dag.backup_and_restore()
526
527     # DAG backup and restore
528     dag.backup_and_restore()
529
530     # DAG backup and restore
531     dag.backup_and_restore()
532
533     # DAG backup and restore
534     dag.backup_and_restore()
535
536     # DAG backup and restore
537     dag.backup_and_restore()
538
539     # DAG backup and restore
540     dag.backup_and_restore()
541
542     # DAG backup and restore
543     dag.backup_and_restore()
544
545     # DAG backup and restore
546     dag.backup_and_restore()
547
548     # DAG backup and restore
549     dag.backup_and_restore()
550
551     # DAG backup and restore
552     dag.backup_and_restore()
553
554     # DAG backup and restore
555     dag.backup_and_restore()
556
557     # DAG backup and restore
558     dag.backup_and_restore()
559
560     # DAG backup and restore
561     dag.backup_and_restore()
562
563     # DAG backup and restore
564     dag.backup_and_restore()
565
566     # DAG backup and restore
567     dag.backup_and_restore()
568
569     # DAG backup and restore
570     dag.backup_and_restore()
571
572     # DAG backup and restore
573     dag.backup_and_restore()
574
575     # DAG backup and restore
576     dag.backup_and_restore()
577
578     # DAG backup and restore
579     dag.backup_and_restore()
580
581     # DAG backup and restore
582     dag.backup_and_restore()
583
584     # DAG backup and restore
585     dag.backup_and_restore()
586
587     # DAG backup and restore
588     dag.backup_and_restore()
589
590     # DAG backup and restore
591     dag.backup_and_restore()
592
593     # DAG backup and restore
594     dag.backup_and_restore()
595
596     # DAG backup and restore
597     dag.backup_and_restore()
598
599     # DAG backup and restore
600     dag.backup_and_restore()
601
602     # DAG backup and restore
603     dag.backup_and_restore()
604
605     # DAG backup and restore
606     dag.backup_and_restore()
607
608     # DAG backup and restore
609     dag.backup_and_restore()
610
611     # DAG backup and restore
612     dag.backup_and_restore()
613
614     # DAG backup and restore
615     dag.backup_and_restore()
616
617     # DAG backup and restore
618     dag.backup_and_restore()
619
620     # DAG backup and restore
621     dag.backup_and_restore()
622
623     # DAG backup and restore
624     dag.backup_and_restore()
625
626     # DAG backup and restore
627     dag.backup_and_restore()
628
629     # DAG backup and restore
630     dag.backup_and_restore()
631
632     # DAG backup and restore
633     dag.backup_and_restore()
634
635     # DAG backup and restore
636     dag.backup_and_restore()
637
638     # DAG backup and restore
639     dag.backup_and_restore()
640
641     # DAG backup and restore
642     dag.backup_and_restore()
643
644     # DAG backup and restore
645     dag.backup_and_restore()
646
647     # DAG backup and restore
648     dag.backup_and_restore()
649
650     # DAG backup and restore
651     dag.backup_and_restore()
652
653     # DAG backup and restore
654     dag.backup_and_restore()
655
656     # DAG backup and restore
657     dag.backup_and_restore()
658
659     # DAG backup and restore
660     dag.backup_and_restore()
661
662     # DAG backup and restore
663     dag.backup_and_restore()
664
665     # DAG backup and restore
666     dag.backup_and_restore()
667
668     # DAG backup and restore
669     dag.backup_and_restore()
670
671     # DAG backup and restore
672     dag.backup_and_restore()
673
674     # DAG backup and restore
675     dag.backup_and_restore()
676
677     # DAG backup and restore
678     dag.backup_and_restore()
679
680     # DAG backup and restore
681     dag.backup_and_restore()
682
683     # DAG backup and restore
684     dag.backup_and_restore()
685
686     # DAG backup and restore
687     dag.backup_and_restore()
688
689     # DAG backup and restore
690     dag.backup_and_restore()
691
692     # DAG backup and restore
693     dag.backup_and_restore()
694
695     # DAG backup and restore
696     dag.backup_and_restore()
697
698     # DAG backup and restore
699     dag.backup_and_restore()
700
701     # DAG backup and restore
702     dag.backup_and_restore()
703
704     # DAG backup and restore
705     dag.backup_and_restore()
706
707     # DAG backup and restore
708     dag.backup_and_restore()
709
710     # DAG backup and restore
711     dag.backup_and_restore()
712
713     # DAG backup and restore
714     dag.backup_and_restore()
715
716     # DAG backup and restore
717     dag.backup_and_restore()
718
719     # DAG backup and restore
720     dag.backup_and_restore()
721
722     # DAG backup and restore
723     dag.backup_and_restore()
724
725     # DAG backup and restore
726     dag.backup_and_restore()
727
728     # DAG backup and restore
729     dag.backup_and_restore()
730
731     # DAG backup and restore
732     dag.backup_and_restore()
733
734     # DAG backup and restore
735     dag.backup_and_restore()
736
737     # DAG backup and restore
738     dag.backup_and_restore()
739
740     # DAG backup and restore
741     dag.backup_and_restore()
742
743     # DAG backup and restore
744     dag.backup_and_restore()
745
746     # DAG backup and restore
747     dag.backup_and_restore()
748
749     # DAG backup and restore
750     dag.backup_and_restore()
751
752     # DAG backup and restore
753     dag.backup_and_restore()
754
755     # DAG backup and restore
756     dag.backup_and_restore()
757
758     # DAG backup and restore
759     dag.backup_and_restore()
760
761     # DAG backup and restore
762     dag.backup_and_restore()
763
764     # DAG backup and restore
765     dag.backup_and_restore()
766
767     # DAG backup and restore
768     dag.backup_and_restore()
769
770     # DAG backup and restore
771     dag.backup_and_restore()
772
773     # DAG backup and restore
774     dag.backup_and_restore()
775
776     # DAG backup and restore
777     dag.backup_and_restore()
778
779     # DAG backup and restore
780     dag.backup_and_restore()
781
782     # DAG backup and restore
783     dag.backup_and_restore()
784
785     # DAG backup and restore
786     dag.backup_and_restore()
787
788     # DAG backup and restore
789     dag.backup_and_restore()
790
791     # DAG backup and restore
792     dag.backup_and_restore()
793
794     # DAG backup and restore
795     dag.backup_and_restore()
796
797     # DAG backup and restore
798     dag.backup_and_restore()
799
800     # DAG backup and restore
801     dag.backup_and_restore()
802
803     # DAG backup and restore
804     dag.backup_and_restore()
805
806     # DAG backup and restore
807     dag.backup_and_restore()
808
809     # DAG backup and restore
810     dag.backup_and_restore()
811
812     # DAG backup and restore
813     dag.backup_and_restore()
814
815     # DAG backup and restore
816     dag.backup_and_restore()
817
818     # DAG backup and restore
819     dag.backup_and_restore()
820
821     # DAG backup and restore
822     dag.backup_and_restore()
823
824     # DAG backup and restore
825     dag.backup_and_restore()
826
827     # DAG backup and restore
828     dag.backup_and_restore()
829
830     # DAG backup and restore
831     dag.backup_and_restore()
832
833     # DAG backup and restore
834     dag.backup_and_restore()
835
836     # DAG backup and restore
837     dag.backup_and_restore()
838
839     # DAG backup and restore
840     dag.backup_and_restore()
841
842     # DAG backup and restore
843     dag.backup_and_restore()
844
845     # DAG backup and restore
846     dag.backup_and_restore()
847
848     # DAG backup and restore
849     dag.backup_and_restore()
850
851     # DAG backup and restore
852     dag.backup_and_restore()
853
854     # DAG backup and restore
855     dag.backup_and_restore()
856
857     # DAG backup and restore
858     dag.backup_and_restore()
859
860     # DAG backup and restore
861     dag.backup_and_restore()
862
863     # DAG backup and restore
864     dag.backup_and_restore()
865
866     # DAG backup and restore
867     dag.backup_and_restore()
868
869     # DAG backup and restore
870     dag.backup_and_restore()
871
872     # DAG backup and restore
873     dag.backup_and_restore()
874
875     # DAG backup and restore
876     dag.backup_and_restore()
877
878     # DAG backup and restore
879     dag.backup_and_restore()
880
881     # DAG backup and restore
882     dag.backup_and_restore()
883
884     # DAG backup and restore
885     dag.backup_and_restore()
886
887     # DAG backup and restore
888     dag.backup_and_restore()
889
890     # DAG backup and restore
891     dag.backup_and_restore()
892
893     # DAG backup and restore
894     dag.backup_and_restore()
895
896     # DAG backup and restore
897     dag.backup_and_restore()
898
899     # DAG backup and restore
900     dag.backup_and_restore()
901
902     # DAG backup and restore
903     dag.backup_and_restore()
904
905     # DAG backup and restore
906     dag.backup_and_restore()
907
908     # DAG backup and restore
909     dag.backup_and_restore()
910
911     # DAG backup and restore
912     dag.backup_and_restore()
913
914     # DAG backup and restore
915     dag.backup_and_restore()
916
917     # DAG backup and restore
918     dag.backup_and_restore()
919
920     # DAG backup and restore
921     dag.backup_and_restore()
922
923     # DAG backup and restore
924     dag.backup_and_restore()
925
926     # DAG backup and restore
927     dag.backup_and_restore()
928
929     # DAG backup and restore
930     dag.backup_and_restore()
931
932     # DAG backup and restore
933     dag.backup_and_restore()
934
935     # DAG backup and restore
936     dag.backup_and_restore()
937
938     # DAG backup and restore
939     dag.backup_and_restore()
940
941     # DAG backup and restore
942     dag.backup_and_restore()
943
944     # DAG backup and restore
945     dag.backup_and_restore()
946
947     # DAG backup and restore
948     dag.backup_and_restore()
949
950     # DAG backup and restore
951     dag.backup_and_restore()
952
953     # DAG backup and restore
954     dag.backup_and_restore()
955
956     # DAG backup and restore
957     dag.backup_and_restore()
958
959     # DAG backup and restore
960     dag.backup_and_restore()
961
962     # DAG backup and restore
963     dag.backup_and_restore()
964
965     # DAG backup and restore
966     dag.backup_and_restore()
967
968     # DAG backup and restore
969     dag.backup_and_restore()
970
971     # DAG backup and restore
972     dag.backup_and_restore()
973
974     # DAG backup and restore
975     dag.backup_and_restore()
976
977     # DAG backup and restore
978     dag.backup_and_restore()
979
980     # DAG backup and restore
981     dag.backup_and_restore()
982
983     # DAG backup and restore
984     dag.backup_and_restore()
985
986     # DAG backup and restore
987     dag.backup_and_restore()
988
989     # DAG backup and restore
990     dag.backup_and_restore()
991
992     # DAG backup and restore
993     dag.backup_and_restore()
994
995     # DAG backup and restore
996     dag.backup_and_restore()
997
998     # DAG backup and restore
999     dag.backup_and_restore()
1000

```

Figure 8.7: Airflow code view

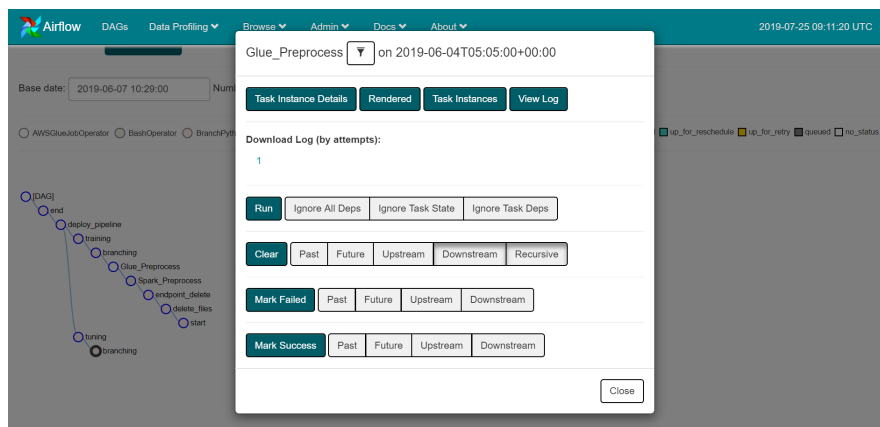


Figure 8.8: Airflow task instance menu

fail (Figure 8.9). Typically, user can request emails by setting parameter *email on failure* to *True* in the operators definition or in the *default arguments* dictionary. It is also possible to create a customized email to include additional information in the content by importing the *send_email* function from Airflow: to use the customized one, it is enough to invoke it *on failure callback* in the operator parameter. An example is reported in Figure 8.9. A brief comment about cost analysis: comparing with the previous chapter no large modifications occur; the only difference consists in an EC2 instance configured to provide the required compute capacity in the cloud instead of a Notebook instance (also Notebook Instance uses an EC2 instance, but its cost is masked into the Notebook cost.). The type used to generate the re-training algorithm is the *t2.small* provided with 1 *vCPU* and 2 *GB*. Table

From:
Sent:
To:

Subject: [External] Airflow alert: <TaskInstance: Workflow_N3.google_api 2019-07-18T04:00:00+00:00 [failed]>

Try 3 out of 3
Exception:
({'invalid_grant': 'Invalid JWT Signature.', '\n "error": "invalid_grant",\n "error_description": "Invalid JWT Signature."\n'})
Log: [Link](#)
Host:
Log file: 2019-07-18T04:00:00+00:00.log
Mark success: [Link](#)

Figure 8.9: Airflow email alert

8.1 summarizes the costs to keep the retraining active. Table 8.2 summarizes

vCPU	GB	Cost/Time
1	2	0.03 \$/h

Table 8.1: Cost analysis: summarize the costs to keep the periodic retraining active.

Task	Frequency	Total Time	Total Cost
Feature selection	Monthly	120 Min	\$ 5
Binary endpoint	Daily	60 Min	\$ 8.6
Multiclass endpoint	Daily	60 Min	\$ 10.4

Table 8.2: Cost analysis: summary of the model retraining costs.

the resulting costs.

8.2 Drift evaluation

Drift is a crucial element in machine learning models, since there is no guarantee that accurate predictions will be obtained if distribution is non-stationary, i.e. it changes over the time. This change is defined as **concept drift**. It concerns data streams, namely sets of data characterized by a time stamp

which introduces an order between samples.

It is possible to characterize the stream with a *joint distribution over random variables* $X = \{X_1, X_2, \dots, X_n\}$ and Y . In particular $y \in Y$ represents a class label, while $x_i \in X_i$ represents an attribute value. The probability distribution at fixed time t can be expressed as $P_t(X, Y)$, but since it is quite hard to identify an instantaneous probability, a distribution over a time interval is more suitable for the task: probability over the interval $[a, b]$ can be expressed as $P_{[a,b]}(X, Y)$. The concept of drift comes when in two different intervals distribution changes, in the sense that

$$P_{[a,b]}(X, Y) \neq P_{[c,d]}(X, Y) \quad (8.1)$$

Literature [28] introduces a probabilistic interpretation of **concept** using the prior class probabilities, i.e. $P(Y)$, combined with the class conditional probabilities, i.e. $P(X|Y)$. However, since $P(Y)$ and $P(X|Y)$ are uniquely associated with the joint distribution, i.e. $P(X, Y)$, it is possible to define *concept* as:

$$\text{Concept} = P(X, Y) \quad (8.2)$$

At time t it is expressed as:

$$\text{Concept}_t = P_t(X, Y) \quad (8.3)$$

and at time period $[a, b]$ as:

$$\text{Concept}_{[a,b]} = P_{[a,b]}(X, Y) \quad (8.4)$$

Starting from these concepts it is possible to define a function, called **concept drift mapping task**, which maps input sample data from two or more distributions into a value describing the drift in data generation process.

One of the most difficult task consists in quantifying the concept drift and giving a concrete expression to the mapping function. Bartlett et al. (2000) [2] introduce the drift rate, but according to Webb et al. (2015) [27] they reduce this complex concept to a too simple function $f : X \rightarrow Y$. For this reason they provide different approaches to quantify drift which consider *probabilistic relationships between X and Y values*. They introduce a way to measure the time difference, called **drift magnitude**: however, it is impossible to find an unique function which correctly predicts it since its expression changes according to the domain it is applied to. For this reason a generic distance function $d(t, t + u)$ is considered, which quantifies the concepts gap between times $t + u$ and t .

Some examples of distance measures which could be used in this case are *Kullback-Leibler Divergence*, *Hellinger Distance* and **Total Variation Distance**. In this work the latter is used, whose general expression is [28]:

$$\sigma_{t,u}(Z) = \frac{1}{2} \sum_{z \in Z} |P_t(z) - P_u(z)| \quad (8.5)$$

where Z is a generic *vector of random variables*. The choice comes from the fact that this measure is less complex and more efficient to compute than the others. In any case, the considered approach could be easily applied to other metrics.

Another crucial step consists in estimating the probability expressed before and the necessity of dealing with variance obliges user to select enough large sets of observations. For this reason, it is not often possible to obtain estimates in a instant of time, but is essential to consider an interval, such as one day or one week. Using the maximum likelihood estimation it is possible to estimate the joint probability $P(X, Y)$, by computing the class $P(Y)$ and covariate $P(X)$ distribution and also the two conditional probabilities $P(X|Y)$ and $P(Y|X)$, which could highlight different drift aspects. The necessity of analysing all those quantities comes from the fact that there could be a change in a class frequency $P(Y)$, or in a covariate frequency $P(X)$ or in the relationships between them, respectively $P(X|Y)$ and $P(Y|X)$, and the scope of the drift evaluation is to capture all of them. The first two cases which represent the **covariate drift** and the **class drift** simply just come down to the Equation 8.5, while for the conditional drift a weighted average of each single dimension is performed: in particular the **conditional marginal covariate drift** ($P(X|Y)$) is expressed as [28]:

$$\sigma_{t,u}^{X|Y} = \sum_{y \in Y} \left[\frac{P_t(y) + P_u(y)}{2} \frac{1}{2} \sum_{x \in X} |P_t(y|x) - P_u(y|x)| \right] \quad (8.6)$$

while the **conditional class drift** ($P(Y|X)$) as:

$$\sigma_{t,u}^{Y|X} = \sum_{x \in X} \left[\frac{P_t(x) + P_u(x)}{2} \frac{1}{2} \sum_{y \in Y} |P_t(x|y) - P_u(x|y)| \right] \quad (8.7)$$

8.3 Retraining based on drift

Since the previous technique is designed for discrete variables, it is not possible to apply it directly to available features: indeed, as presented in chapters

5 and 6, some of them contain continuous values. It is true that solutions to evaluate the total variation distance in case of continuous variables exist, but these approaches require very strong assumptions about the initial distribution, such as be normally distributed, which are not satisfied by the collected data. First of all, only the most 5 relevant features are selected according to the selection performed in section 5.4 and graphically represented in Figures 5.8, 5.9 and 5.10: in particular 3 are numerical and 2 are categorical. Each feature belonging to the first case is discretized according to its value in 10 large categories, which are simply intervals of two values.

Another crucial point comes when drift magnitude has to be quantified. Some authors, instead of considering the conditional, covariate or class drift differently one from the others, propose an unique combined expression. However, in this study this approach is avoided since Webb et al. (2015) [27] prove that when dimensionality increases this representative number becomes close to 1 due to the single small differences accumulation, making trickier to understand the reasons behind the drift. Moreover, using only one value does not allow to describe in detail the type of change and this results to be essential in a world where drift is not uniform (for instance technological evolution could affect only some features but not the others).

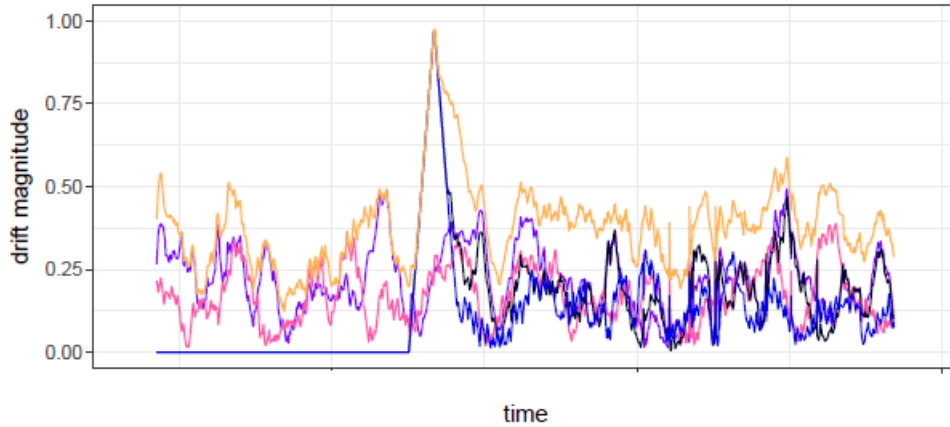


Figure 8.10: Covariate drift with values computed hourly for the drift of 7 hours before the current time.

In order to test the proposed solution, the binary model is again considered. Data are grouped in hourly sets and a time window of 1 month is considered. Then, probabilities in Equations 8.5, 8.7 and 8.6 are computed in order to estimate the covariate, class and conditional drift. An example, taken from literature [28], is reported in Figure 8.10, where each point rep-

resents an hour and shows the drift from the 7 hours period prior to the considered time: it is an example of covariate drift, where each colour represents a different attribute.

Figure 8.10, which is similar to the real one obtained, shows that there is only one sudden increase in covariate drift in the considered month. This encourages the idea that retraining the model daily seems to be not so useful, since in the considered month a drift in the covariate appears only once. However, this is not the only type of change expressed before, so user should also consider class and conditional drift.

In order to decide if and when a model retraining is required, a summarized formula is considered. In particular this operation will be performed if and only if

$$\max \left(\sigma_{t,u}(X), \sigma_{t,u}(Y), \sigma_{t,u}^{X|Y}, \sigma_{t,u}^{Y|X} \right) > 0.6 \quad (8.8)$$

Table 8.3 compares the results between current approach consisting in a daily model retraining versus the presented proposal of basing the decision on data drift. In particular it shows that if user is willing to slightly decrease the monthly average accuracy, costs are dramatically reduced: this means that the new methodology could lead to relevant practical advantages.

Type	Monthly Accuracy	Number of retraining	Cost
Daily retraining	78%	30	\$ 258
Drift evaluation	72%	4	\$ 34.4

Table 8.3: Drift analysis: comparison between solution with and without drift consideration for 1 month time window.

Chapter 9

Model architecture and deployment

Although a continuous maintenance and monitoring is necessary, once pipeline has been tested it is time to proceed with the ending phase. Although this chapter seems to be marginal compared to all the others, it assumes a crucial role since without a clear overview the entire project becomes unprofitable. Therefore, the final step before releasing the solution in a production environment consists in identifying which are the **actors** involved in the project and which is their **role**. Before that, figure 9.1 summarizes the developed program and helps in understanding the final solution. The figure shows

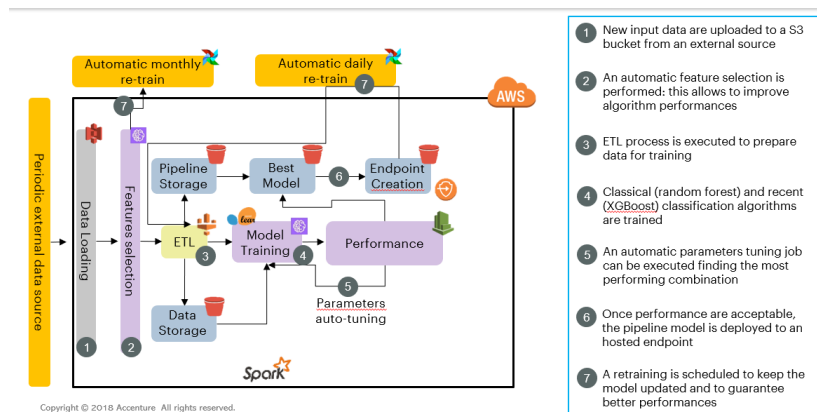


Figure 9.1: Architecture of the final binary and multiclass classification solution

the solution implemented for both binary and multiclass problem, since two distinct endpoints will be created at the end.

Back to the actors, three are the figures engaged: the **data scientist developer**, the **operation employee** and the **client**.

The **data scientist** has to:

- create a jupyter notebook web application;
- run a preprocessing pipeline on the input data;
- train or tune a model, evaluating performances on a test set provided during training phase;
- create a pipeline model combining both preprocessing and trained algorithm, and then save it.

Moreover, he develops also the entire workflow on Apache Airflow platform. SageMaker allows to create notebook instances in a very intuitive way. Developer needs to set only few parameters:

- name and type of notebook instance (CPU,GPU);
- existing or new IAM role with the necessary permissions to access Amazon SageMaker and other resources.

Jupyter notebook server is already configured and SageMaker examples are already available and ready being run.

AWS Glue asks only to:

- create a *.py* file composed by a data loading from an input *S3* location and a pipeline preprocessing which is applied to original data. Data are transformed and uploaded together with the pipeline preprocessing model;
- upload the *.py* and, if needed, other external dependencies on *S3*;
- create a glue job by setting: name of the job, *.py* file and other dependencies location;
- start a glue job by setting: input and output data location and model location.

SageMaker allows, also, to easily create a training or tuning job in few steps and monitor it. An estimator is created based on the type of algorithm chosen:

- built-in algorithm needs the built-in algorithm image;

- pre-built framework containers need a custom training script;
- own algorithm needs a custom container image.

Data scientist can create an Hyperparameter job by defining the algorithm, the hyperparameter ranges and an evaluation metric receives as output the best training job found. Jobs performances could be retrieved from the logs if a test set is provided during the training phase: data scientist can specify a customized metric or use the default one.

SageMaker can create and save a model from the previous training or tuning job for a future deploy if performances are acceptable.

Then, the latter can be saved by setting the default EC2 instance type to deploy the model to.

Finally Airflow has extensive support for Amazon Web Services to single model train, tune, deploy and transform, allowing to monitor easily failed tasks, tasks duration and providing a simple graphical interface.

Operation employee has to:

- access to AWS platform as global admin or platform operator;
- check if exists a new tuning job and the relative performance;
- if performances are not acceptable, he could run a new training or tuning job with different hyperparameters using Airflow rich user interface;
- deploy the model, once the performances become acceptable.

Thanks to the workflow developed on Apache Airflow platform, operation employee could easily:

- monitor the entire process;
- schedule an automatic model retraining;
- run a new hyperparameter tuning job or a new training job.

More in detail, operation user needs to log in to the platform using:

- *Global_Admin*
- *Platform_Operator* (if AWS managed policies AmazonSagemakerFullAccess is attached)

SageMaker graphical user interface allows to check previous training and tuning jobs and then deploy the previous model created:

- on the Amazon SageMaker console, there is a direct link to all hyperparameter tuning jobs;
- check if there exists a new hyperparameter tuning job (*name_tuning_job+timestamp*);
- select the best training job obtained and check the test score.

If performances are not acceptable, Airflow gives the chance to run a new training or tuning job, by triggering the entire workflow:

- on the Airflow main page, select the DAG related to the task and trigger it using the link on the right;
- the rich user interface makes easy to monitor progress (to change hyperparameter values or run a training job instead of a tuning job a configuration file must be changed);
- check new job performances as before.

When performances become acceptable, SageMaker allows to deploy the model into an hosted environment:

- Amazon SageMaker shows all the models and mainly the last one created from which a new endpoint can be configured;
- create a new endpoint configuration by choosing the name and the model (operation user can change also instance number and type);
- create the final endpoint.

At the end of this process an endpoint is available, ready to be invoked to make any predictions.

Client has only to make batch predictions by streaming data from and back to *S3* or real time predictions by invoking the created endpoint. The figure below shows the process which a new observation undergoes to (the only difference comes in the input: one single element for real time prediction versus a sample file for batch prediction).

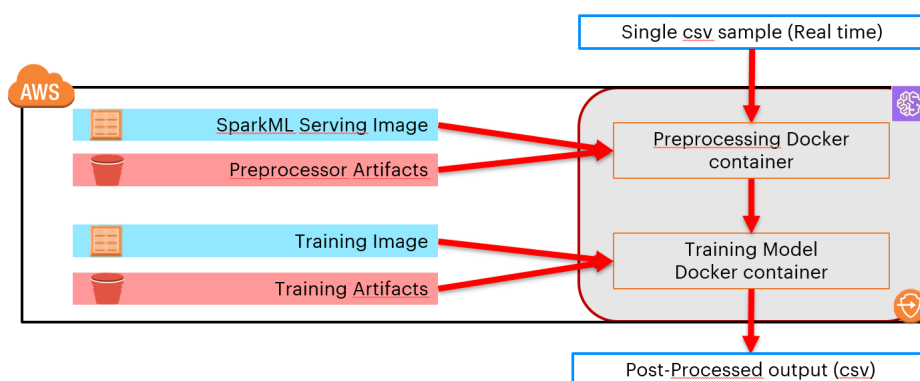


Figure 9.2: New observation prediction

Chapter 10

Conclusions

The first part of this work consisted of exploring the web services provided by Amazon.com, which gives users an extremely fully-functional, flexible technology infrastructure platform. It provides a pay-as-you-go service, where low cost IT resources are available on request avoiding large investments in hardware and in efforts to manage it. Therefore, people pay only for computational power which actually use. The agility of spinning up resources almost instantly, the elasticity of scaling up or down resources without any constraint, the speed of deploying applications in different physical places with few clicks and the cost saving make AWS the first choice for a lot of businesses.

In particular Amazon SageMaker helps data scientist in building, training and deploying machine learning models. It provides pre-built notebooks for common problems and built-in, high-performance algorithms to start working on. A managed environment is available for one-click model training and tuning and another environment for one-click model deployment in production, with the possibility to scale resources automatically in order to reduce costs as far as possible.

The advantages of the cloud solution have been applied to change the way telecommunication companies interact with client issues. Indeed, instead of simply monitoring devices status using deterministic KPIs and algorithms, for the first time companies tried to move to a digital prediction scenario where analytic tools and machine learning algorithms are used to identify factors which are causing current problems or which are likely to cause future problems. Three different use cases have been analysed, which are chosen from the client as the most relevant ones: unstable line, slow line and Wi-Fi connection issues. The main scope was to identify customers at high risk of issue in near real time in order both to prevent future calls and when this is not possible to improve first call resolution.

Some discussions concerned the solution to be proposed to the client, since input data presented a lot of problems, as reported in Section 5.2. Indeed, the preparation phase has required the majority of the efforts, also due to the necessity of considering customers behaviours and the willingness to add new derived features to the original ones. The solution proposed was split into two steps: a binary classification model to identify if an issue will occur on the specific line and a multi-class classification model to understand the type of problems. The idea of providing a probability of line failures in a first phase and then in a second phase try to give an explanation of possible reasons behind them, was highly appreciated by the client. Moreover, having an issue explanation thorough the multi-class model even if the binary model did not suggest a problem the line, was very useful for the client to improve first call resolution. Three different algorithms have been tested for the considered use cases: one is the Extreme Gradient Boosting, which is an Amazon built-in algorithm for tree boosting. Since it is already implemented inside Amazon SageMaker, it is quite easy to use but it was treated like a black-box model. This is the reason why also decision tree and random forest were considered, thanks to ad-hoc structure available within SageMaker environment to integrate Scikit-learn library. In the first phase the main goal was to maximize model capability to be extremely precise in predicting a call event, reducing as much as possible the false positive rate, while in the second phase a weighted precision, based on the number of true instances for each label (Wi-Fi issue is much more frequent than the others), was considered. The pipeline presented to the client is able to detect the presence of an issue with a precision of more than 80% and also to satisfactorily classify different types of problem with a weighted precision of 80%.

As reported in chapter 8, in order to keep predictions update a periodic re-training was required. This work presented two different solutions, the first one using a tool called Apache Airflow and the second one by evaluating drift in the data. The former provided an automatic daily retraining of the entire pipeline (with the exception of features selection due to the large amount of time required): this is an intuitive and easy solution but it is characterized by high cost. This is the reason why this work introduced a technique based on drift, which showed that retrain the model each day appeared unnecessary. The second solution, even if it means reduce precision of the predictions, can guarantee lower costs.

This work was proposed to the patron and it received a positive feedback to the point that a deployment phase was demanded. Chapter 9 tried to meet this requirement by identifying which actors were and will be involved in a future possible release into production. Apart from the data scientist who develops the solution, there is the operational employee who has to mon-

itor and keep the model functioning and the client who has to invoke the endpoint created to obtain real time predictions. Clearly, it is not possible to ask to client to directly interact with the endpoint, but an user-friendly platform or dashboard should be implemented. An idea could be to integrate this prediction inside the CEM platform already developed and deployed by the operations support system team. A section could be reserved to display customer's characteristics, probability of having a line issue and possible explanation for it.

Results obtained confirmed that using Amazon Web Services platform for businesses represents an incredible advantage in terms of resources management and cost investment. It could help companies in developing their solutions to face modern challenges as the one explored in this thesis. On the other side this work confirmed also machine learning potentialities to tackle new telecommunication companies initiatives as moving from digital monitoring to digital prediction. For this reason according to this study, it can be exploited for other future challenges which will receive attention from companies.

Bibliography

- [1] Baralis, Elena (2015) *Big Data: Hype or Hallelujah?*, <http://dbdmg.polito.it>
- [2] Bartlett, P., Ben-David, S., Kulkarni S. (2000) *Learning changing concepts by exploiting the structure of change*. Machine Learning, 41(2): 153-174, 2000.
- [3] Bergstra, J., Bengio, Y. (2012) *Random Search for Hyper-Parameter Optimization*, Département d'Informatique et de recherche opérationnelle, Université de Montreal, Montréal, Canada
- [4] Carey, Scott (2019) *AWS vs Azure vs Google: What's the best cloud platform for enterprise?*, UK Group Editor, Computerworld
- [5] Chen, T., Guestrin, C. (2016) *XGBoost: A Scalable Tree Boosting System*
- [6] Duhigg, Charles (2012) *How Companies Learn Your Secrets*, The New York Times Magazine, <https://www.nytimes.com>
- [7] CloudSoft (2018) *Comparing AWS and Azure: 12 reasons to run Microsoft Workloads on AWS*, a Cloudsoft Whitepaper, Cloudsoft Corporation
- [8] Gruber, Steve (2016) *5 steps to selling the solution not the product*, Venture Accelerator Partners
- [9] Howard, Eric (2018) *The Evolution of the Industrial Ages: Industry 1.0 to 4.0*.
- [10] Jackson, Brian (2019) *Google Cloud vs AWS in 2019 (Comparing the Giants)*, Kinsta, Premium Managed WordPress Hosting
- [11] James, G., Witten, D., Hastie, T., Tibshirani, R. (2015) *An Introduction to Statistical Learning, with Applications in R*, 8:303-314, Springer, New York

- [12] James, G., Witten, D., Hastie, T., Tibshirani, R. (2015) *An Introduction to Statistical Learning, with Applications in R*, 8:316-321, Springer, New York
- [13] Maheshwari, Anil *Big Data*, McGraw-Hill Education, 2017
- [14] Marko, K., Carty, D. (2017) *Amazon Web Services (AWS)*, TechTarget
- [15] Neirotti, Paolo (2019) *The Fourth Industrial Revolution: Introduction to the School*, Politecnico di Torino, Department of Management and Production Engineering, Alta Scuola Politecnica.
- [16] Palevani, Zara (2017) *Data that Knows Humans: Google Cloud APIs for Face, Voice, and Video Detection*, E-Nor - Google Marketing Platform Consulting and Training, Google Certified Service & Sales Partners
- [17] Rouse, Margaret (2018) *RDBMS (relational database management system)*, TechTarget
- [18] Santos, B., Charrua-Santos, F., Lima, T.M. (2018) *Industry 4.0: an overview*.
- [19] Sether, Ayob (2016) *Cloud Computing Benefits*.
- [20] Sharma, Hemant (2019) *AWS Pricing – An Introduction to AWS Pricing*, Edureka
- [21] Siddiqi, Adnan (2018) *Getting started with Apache Airflow*, Towards Data Science
- [22] Singh, S. (2018) *Understanding the Bias-Variance Tradeoff*, Towards Data Science
- [23] Taylor-Sakyi, Kevin (2016) *Big Data: Understanding Big Data*.
- [24] Ting Si Xue, Colin & Tiong Wee Xin, Felicia (2016) *Benefits and Challenges of the Adoption of Cloud Computing in Business*, International Journal on Cloud Computing: Services and Architecture.
- [25] Tsidulko, Joseph (2019) *AWS, Azure and Google top Gartner's IaaS magic quadrant*, CRN Australia, Connecting the Australian Channel
- [26] Wang, S., Tang, J. and Liu, H. (2016) *Feature Selection*, ResearchGate.

-
- [27] Webb, G., Hyde, R., Cao, H., Nguyen, H., François P. (2015) *Characterizing Concept Drift*. Data Mining and Knowledge Discovery. 30. 10.1007/s10618-015-0448-4.
- [28] Webb, G., Lee, L., Goethals, B., Petitjean, F. (2018) *Analyzing concept drift and shift from sample data*. Data Mining and Knowledge Discovery. 10.1007/s10618-018-0554-1.

Sitography

- [29] Hannover Messe, <https://www.hannovermesse.de/home>,
last visit 05/03/2019
- [30] Microsoft Azure, <https://azure.microsoft.com>,
last visit 05/03/2019
- [31] OpenSource, <https://opensource.com/resources/big-data>,
last visit 17/04/2019
- [32] Wikipedia https://en.wikipedia.org/wiki/Amazon_Web_Services,
last visit 21/05/2019
- [33] Wikipedia https://en.wikipedia.org/wiki/Microsoft_Azure,
last visit 21/05/2019
- [34] Wikipedia https://en.wikipedia.org/wiki/Google_Cloud_Platform,
last visit 21/05/2019
- [35] Amazon Web Services, <https://aws.amazon.com/products>,
last visit 23/05/2019
- [36] The Linux Juggernaut, [https://www.linuxnix.com/
amazon-aws-regions-vs-availability-zones-vs-edge-locations-vs-data-centers](https://www.linuxnix.com/amazon-aws-regions-vs-availability-zones-vs-edge-locations-vs-data-centers),
last visit 28/05/2019
- [37] AWS Documentation, [https://docs.aws.amazon.com/en_us/
whitepapers/latest/aws-overview/compute-services.html](https://docs.aws.amazon.com/en_us/whitepapers/latest/aws-overview/compute-services.html),
last visit 02/06/2019
- [38] AWS Documentation, [https://docs.aws.amazon.com/en_us/
whitepapers/latest/aws-overview/storage-services.html](https://docs.aws.amazon.com/en_us/whitepapers/latest/aws-overview/storage-services.html),
last visit 02/06/2019

- [39] AWS Documentation, https://docs.aws.amazon.com/en_us/whitepapers/latest/aws-overview/database.html,
last visit 02/06/2019
- [40] AWS Documentation, https://docs.aws.amazon.com/en_us/sagemaker/latest/dg/automatic-model-tuning-how-it-works.html,
last visit 03/06/2019
- [41] AWS Documentation,
https://docs.aws.amazon.com/en_us/sagemaker/latest/dg/xgboost.html,
last visit 03/06/2019
- [42] AWS News Blog, <https://aws.amazon.com/en/blogs/aws/enhanced-cloudfront-logs-now-with-query-strings>,
last visit 14/06/2019
- [43] Airflow, <https://airflow.apache.org>,
last visit 18/06/2019
- [44] Wikipedia, https://it.wikipedia.org/wiki/Industria_4.0,
last visit 13/07/2019
- [45] Wikipedia, https://en.wikipedia.org/wiki/Relational_database#RDBMS,
last visit 13/07/2019
- [46] Wikipedia,
https://en.wikipedia.org/wiki/Database#Database_management_system,
last visit 13/07/2019
- [47] Radius Technology
<https://radius.ie/10-advantages-of-cloud-computing-from-10-experts-2>,
last visit 15/07/2019
- [48] Business Dictionary,
<http://www.businessdictionary.com/definition/customer-support.html>,
last visit 28/07/2019
- [49] Business Dictionary,
<http://www.businessdictionary.com/definition/technical-support.html>,
last visit 28/07/2019
- [50] Investopedia, <https://www.investopedia.com/terms/m/masscustomization.asp>,
last visit 28/07/2019

-
- [51] Investopedia, <https://www.investopedia.com/terms/q/quality-control.asp>,
last visit 28/07/2019
 - [52] Wikipedia,
https://en.wikipedia.org/wiki/Customer-premises_equipment,
last visit 03/08/2019
 - [53] Apache Spark,
<https://spark.apache.org/docs/latest/ml-features.html>,
last visit 05/08/2019
 - [54] Wikipedia,
https://en.wikipedia.org/wiki/Wide_area_network,
last visit 11/08/2019
 - [55] Wikipedia,
https://en.wikipedia.org/wiki/Wireless_LAN,
last visit 11/08/2019
 - [56] Exilio, <https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures>,
last visit 29/08/2019
 - [57] Github, <https://github.com/aws-labs/amazon-sagemaker-examples>,
last visit 29/08/2019
 - [58] SageMaker, <https://sagemaker.readthedocs.io>,
last visit 03/09/2019
 - [59] Scikit Learn, <https://scikit-learn.org>,
last visit 08/09/2019