POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Empowering Machine Learning Workflows with AWS: Engineering Algorithms for Scalability and Efficiency

Supervisors

Prof. Tatiana TOMMASI

Dott. Enrico Maria GIRAUDO

Candidate

Simone SONCIN

OCTOBER 2023

Abstract

The aim of this thesis is to the design and implement a Machine Learning Operation (MLOps) model for large enterprise BigData and Machine Learning algorithms management on cloud. MLOps is the process that moves machine learning models into production environments. It unifies data collection, preprocessing, model training, evaluation, deployment, and retraining in a single process that teams can maintain and continuously monitor.
Specifically, this work got started from a customer project in the Energy Sector with the need of using real data and actual machine learning algorithms provided by the customer. The data elaboration and model learning ask for a tailored strategy to exploit cloud services (Amazon Web - AWS) and guarantee security and scalability. The work mainly focused on migrate pre-existing algorithms from an outdated environment to one that is more modern, more efficient, and has more functionality. Another goal of this work is to take advantage of the modularity of this new tool to be able to create multiple flows by rewriting a minimal amount of code.
A thorough comparison with pre-existing approaches on the basis of speed, resilience, ease of use metrics showed the advantage of the proposed strategy, that combines the minimum time reduction of the order to the 50% and the resiliency provided by the Endpoints.
For the future we are planning to expand the workflows by supporting other types of procedures, and to improve the automatic deploying of the models.

We always overestimate the change that
will occur in the next two years and underestimate
the change that will occur in the next ten.
Don't let yourself be lulled into inaction.

Bill Gates

Table of Contents

Acronyms

AI

Artificial intelligence

AWS

Amazon Web Services

CSP

Cloud Service Provider

ML

Machine learning

MLOps

Machine learning operations

DevOps

Development operations

NLP

Natural Language Processing

BERT

Bidirectional Encoder Representations from Transformers

GPU

Graphics Processing Unit

Chapter 1

Introduction

1.1 MLOps and DevOps

DevOps is a software practice that integrates the two worlds of development and operations with automated development, deployment and infrastructure monitoring. It's an organizational shift where instead of distributed silo-like functions cross-functional teams work on continuous operational feature deliveries. This integrative approach helps teams deliver value in a faster and continuous way, reducing problems generated by miscommunication between team members and enhancing a faster resolution of problems. [1]

MLOps (Machine Learning Operations) is a paradigm, including aspects like best practices, sets of concepts, as well as a development culture when it comes to the end-to-end conceptualization, implementation, monitoring, deployment, and scalability of machine learning products. Most of all, it is an engineering practice that leverages three contributing disciplines: machine learning, software engineering (especially DevOps), and data engineering. MLOps is aimed at productionizing machine learning systems by bridging the gap between development (Dev) and operations (Ops).

Essentially, MLOps aims to facilitate the creation of machine learning products by leveraging these principles: CI/CD automation, workflow orchestration, reproducibility; versioning of data, model, and code; collaboration; continuous ML training and evaluation; ML metadata tracking and logging; continuous monitoring;

and feedback loops. [2]
From the previous definitions, we can deduce that MLOps is the application of the concept of DevOps in a Machine Learning and Data Engineering contexts, and contains task like defining pipelines to automating train models when new data are inserted in a storage and automatize model deployment.

1.2 On-premises AI approach

The On-Premises AI approach is the simplest and cheapest way to perform the training and the inference of a Machine Learning model.
This approach consists in owning your own server and run your python notebook, that is a literate programming tool widely used to write data science applications. A notebook environment supports chunks of content, called "cells."
A cell can contain code, output, a table, a plot, formatted "Markdown" text, or other kinds of media. [3]
One of the advantages in the notebook approach is that even if it use a vertical order by default, it is not enforced because the progression of the code is chronological. The results can be different based on the cells order executions.
That is useful because it is possible to execute only the required snippet of code instead of the entire project, saving a lot of time spread across multiple executions.
The main advantages of running your code on your own on-premises device are:

- Simplicity: you can run your application over your device without time limit, without needing an interrupted internet connection and just owning your own device, that could be a simple laptop or an entire data center.

- Costs: this could be both an advantage or a disadvantage, but if you already have you own data center mainly used for other applications and there is the possibility of using it for running additional notebooks the total expenses are just for maintenance and for the electricity supply.
  If there is no data centers, the costs could be an advantage if there are planned to run a very large number of notebooks. In this case the costs are mainly accountable to the construction of the data center, because a lot of GPUs are needed, and their costs are very high, but running a lot of notebooks allows

this expenses to be divided for each execution, resulting in a low cost per execution described as:
$ExpensesPerExecution = \frac{DataCenterCosts}{NotebookExecutions} + EnergySupply + Maintenance$

This solution have also some disadvantages which makes it applicable only under special circumstances:

- Computational Power: as previously mentioned, using your own data center for running your tasks requires a lot of computational power, that means that if there is a need to execute a task that requires more power than the provided one, you will have to upgrade your own hardware to make sure that you have enough resources to perform it.

- Backup Strategy: It's charge to the system analyst to take backups and store them in a safe place. There is no automated service that helps with the backup acquisition and the maintenance. Without a right and strict backup policy there could be a data loss as a result of a system interruption caused by a device malfunction or a loss of electricity.

- Costs: The costs could be a disadvantage as well as an advantage. If there is a need to run few but heavy notebooks, it is not convenient to spend a lot of money to build and maintain an entire data center. In this case it is impossible to use a cheap system to run the required tasks because of the computational power requirements, but considering that the tasks will be executed rarely there is no conditions to invest a large amount of money.

- Downtime: There is a possibility that the data center will suffer of a system failure, that could be both software or hardware. In addiction of costing money to replace the malfunctioning device, an hardware failure can not be easily resolved if there is no immediately availability of a replacement device and that could bring to a loss of hours or days of work.
  In addition to that a on-premises data center, to remains always available, requires an uninterrupted connection to the electricity system. Any kind of interruption results in a task failure, that means that a task that has been running for several hours or even days will have to be started over again.

To make a final analysis, this approach is largely used by very big companies or by universities that can afford a data center and have a lot of tasks to be ran on it. Running a notebook on your own personal computer could be done only if the task to be executed is lightweight, for example a simple classification problem that not need to analyze images or tasks not related with the machine learning world, like a simple application written in Python.

1.3 Cloud AI approach

The Cloud AI approach is a general name to describe a Machine Learning algorithm executed on a machine hosted by a Cloud Service Provider.
In the cloud environment there are different ways to implement a Machine Learning algorithm in contrast to the on-premises approach that offers less options, because in a cloud environment there are many different services hosted by the Cloud Service Provider that could be used to implement different types of solutions that is impossible to do in your own computer.
In a cloud approach, the main advantage is that you does not need to handle the hardware components used to run your Machine Learning models, meaning that you never handle issues related to hardware failures, hardware upgrades or, most of the time, you never handle operative system failure.
To upgrade your hardware system, you only need to select the more powerful virtual machine and you have to pay more than before, without doing nothing else.
There are several types of services that rely on a cloud solution, and each of them works in a different way:

- Online notebook: This solution consists of using an online tools that have the same functionality as a Python notebook.
  The characteristics of this tool is that you does not need to manage the underlying hardware components. You don't even have to deal with issues like scalability of the resources or operative systems issues, because the service provider allows you only to write your Python code in their notebook.
  If there is a need of more computational power, you can upgrade the underlying hardware, but is less customizable in confront of the other solutions and you cannot do distributed calculations, effectively making impossible to scale

horizontally (add more virtual machines to better perform your task) but only vertically (use a powerful machine). An example of a very used tool is Google Colab that has different types of purchase plans:

  - a free tier useful to perform isolated executions, that provides a well performing GPU, but low tier one compared to the other tiers. This purchasing option is subject to frequently timeouts if you are away from your pc or even if the notebook execution lasts a long time.
  - there are different type of paying options, but the main characteristics is that paying more results in having more features, like longer or no timeouts and better hardware to perform your tasks.
    This solution is cheap, so is useful when you need a lot of computer calculation but when you can sacrifice the elasticity of your system.

- cloud native solution: Develop a cloud native Machine Learning application consists of using some Cloud Service Provider's tools that could host the model of the application and the algorithm used to make the train and the inference of the model.
  With this solution is possible to perform inference by exposing the model outside the cloud private subnet, and the user does not need to manage the underlying hardware.
  As opposed to the previous solution, the user must design the cloud architecture and must deploy the infrastructure. This solution does not provide natively a notebook, but it could be installed (Jupyter) or even not used.
  The main advantage is that you can design your architecture any way you want, being free to choose the best fit virtual machine and a customized way to expose the model.
  On the other hand, the costs are generally higher because of using a not specialized tools to run a very particular job.
  It's possible to use different components depending on the computational power needed, for example serverless computations for light tasks, several virtual machines for average workloads and managed clusters for heavy workloads.
  Not having an ad-hoc solution causes that there are no simplifications in using this type of solutions, which mainly provide an improvement on performances

and network elements, but not providing tools that can help data scientists and data engineers in the actual development of the final product.

- hybrid approach: The hybrid approach is a combinations of the previous ones, combining the elasticity of the cloud native solutions and the utilities of the online notebook.
  This solution is implemented as a service that relies on the Cloud Service Provider, in practice one or more notebook is deployed on some cloud resources, that are customizable by the user to scale up or scale down the computational power provided.
  Alongside the notebook, the Cloud Service Provider can provide some services to expand the possible features provided. The advantages of the hybrid approach are:

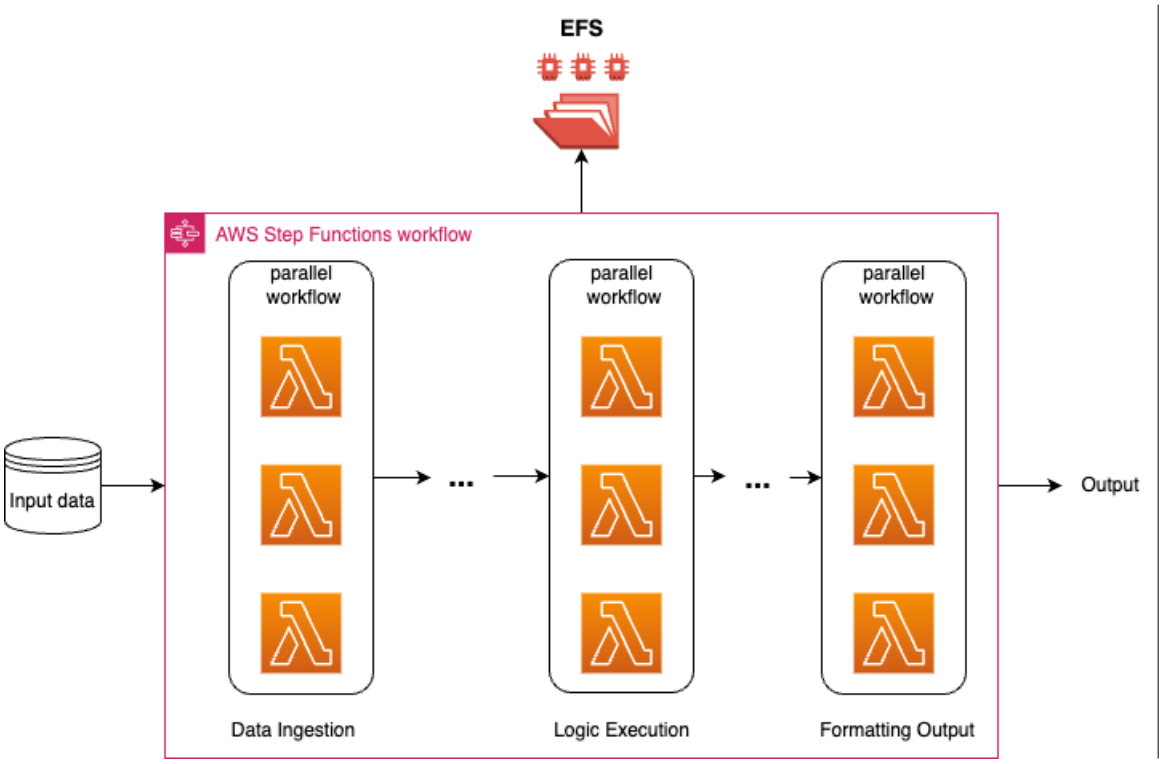
  – It's possible to use in a simple way all the services made available by the CSP, that can expand the pool of the services available with little effort;
  – It's possible to easily scale up or down the numbers of the virtual machine horizontally or vertically;
  – All the CSP services are available and ready to use;
  – The notebook can reduce the code development complexity because of its characteristics;
  – simplicity of the model deploy, train and inference.

  This solution it's the most expensive one. Helping the data scientist and the data engineer in the training and the inference means that there are some dedicated hardware that consume a lot of money and computational power. This solution is the best trade-off, because it's customizable and with a little increase of the expenses it's possible to greatly speed up the work.

1.3.1 Step Functions approach

One example of a cloud approach could use the AWS Stepfunctions, a visual workflow tool useful to build distributed applications, automate processes and create data and machine learning pipelines.

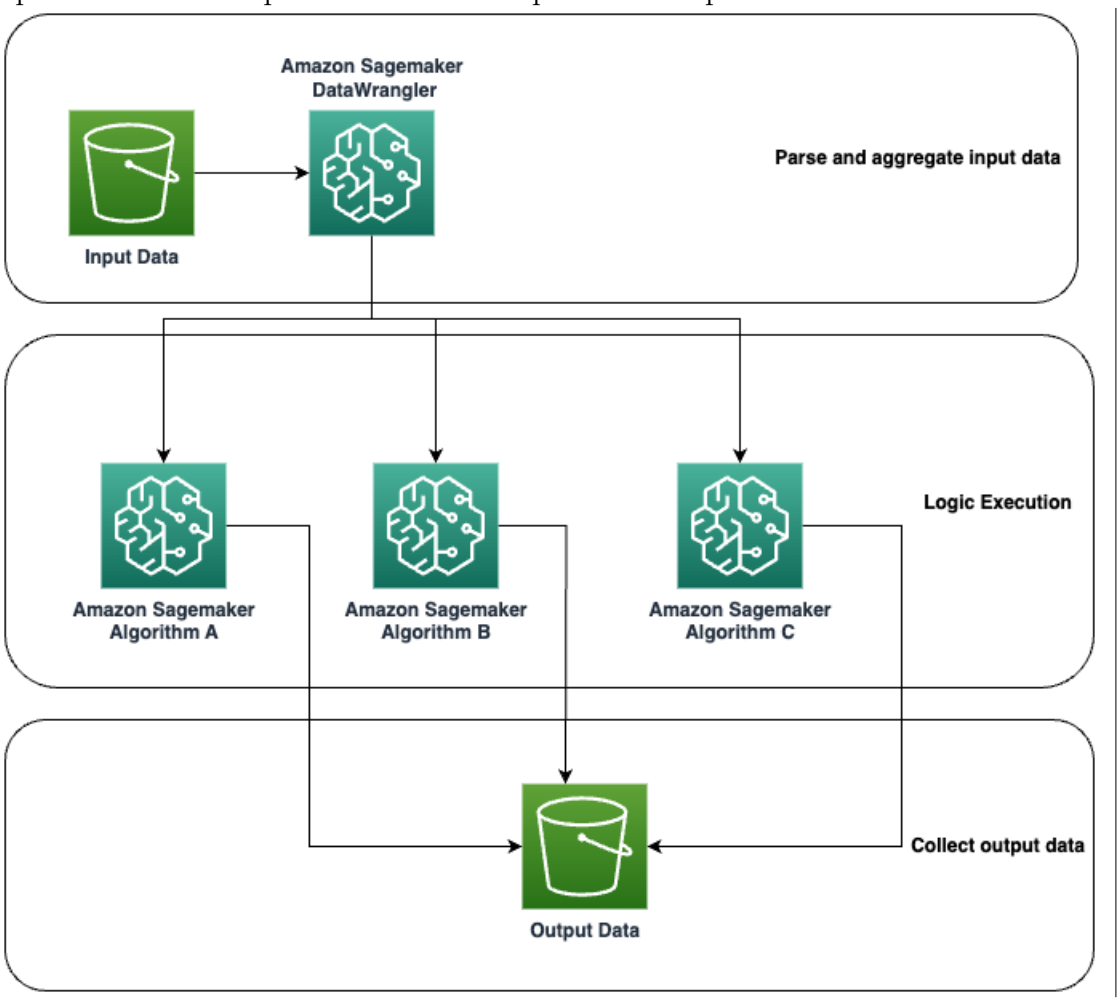
AWS Step Functions is a serverless workflow orchestrator service, that allows to easily orchestrate several Lambda functions that are in charge to perform one single job each.
With the help of AWS Step Functions it is possible to concatenate the execution of these Lambda functions to create multiple parallel workflows that can take different paths based on the model that must be used.
Through AWS Step Functions it is possible to re-utilize Lambda functions in different points in our workflow to avoid to have to rewrite the code.
This approach consists of having a StepFunction that orchestrate several Lambda Function, that are in charge to execute scripts written in Python that are used to perform the inference of the model, the data preparation and the parsing of the output. The StepFunction regulates the execution of the Lambda functions, adjusting the execution flow and limiting the execution parallelism.
Using this approach require that the models are stored in a shared storage service, like Elastic File System (EFS), that is a shared file system accessible in parallel by several Lambda functions at the same time.
This method has the advantages of the cloud approach, which are the infrastructure scalability and the operative system management performed by the Cloud Service Provider.
This solution is not well-designed, because it has a lot of limitations, for example the time limit of the execution of the lambda functions fixed to 15 minutes, not enough to execute the inference of an algorithm to large datasets, or the EFS that reach the throughput limit by providing the model to multiple executions of the Lambda functions.



1.3.2 Sagemaker approach

An example of hybrid approach in AWS is using the managed service "Amazon Sagemaker", that allows you to train, deploy, and perform inference of machine learning models without worrying about maintaining multiple environments and workflows.
It provides the flexibility to use the same models, frameworks, and algorithms you already use today, but with the freedom to focus all of your time on your models rather than the complexities of scaling and application integration.
Sagemaker provides the possibility of using Jupyter to manage Python notebooks that are used to train the models and to create pipelines used to perform inference on it.
Sagemaker provides Endpoints, that corresponds to templates used to know which model have to be exposed and how much computational powers needs. The Endpoints are used to expose a trained machine learning model and make it available to perform inference on it.

The model training is performed one time, so it is done by executing the Python notebook that create the inference pipeline. This pipeline is triggered by uploading a file on a S3 Bucket, so it will be executed several times. It basically consists of create an Endpoint that expose the right model, perform inference with the uploaded data and perform the clean up of the Endpoint.



Chapter 2

Background

This chapter will focus on explaining the technologies used as part of this thesis to help understand why is emerging the need to move complexity from local servers to machines managed by a third-party company.

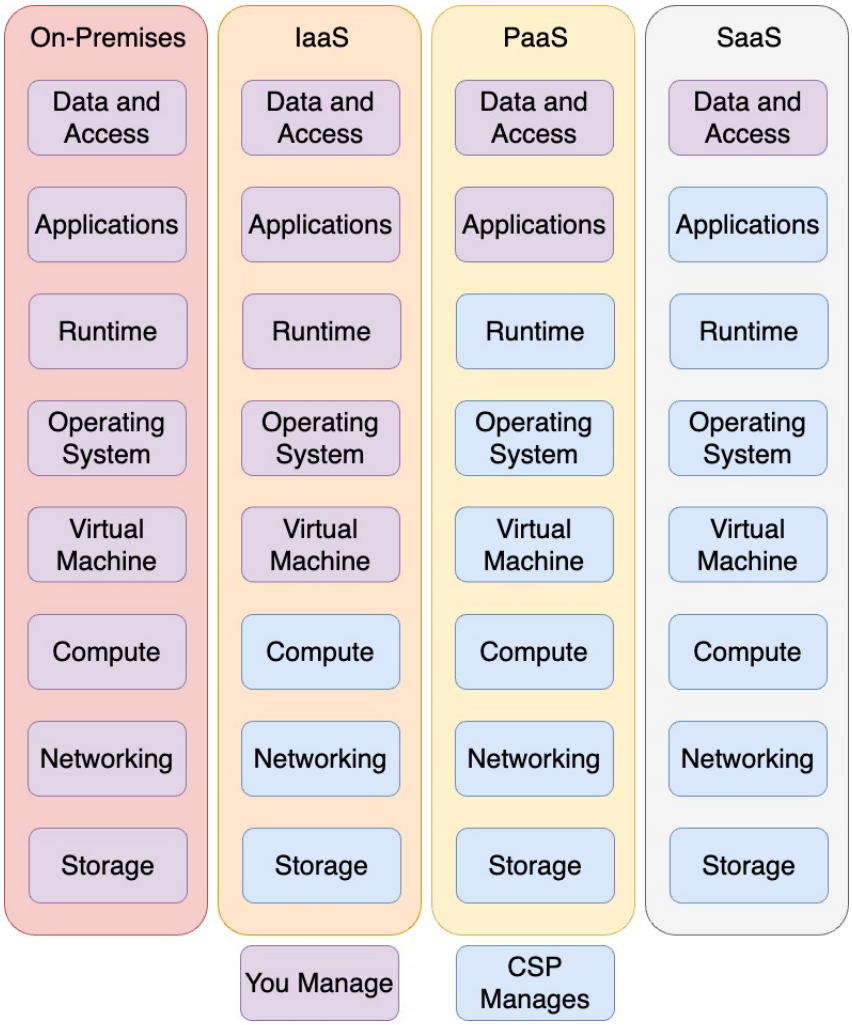
2.1 Introduction to Cloud Computing

Cloud Computing is a discipline that focus mainly in provides a large quantity of computational power, storage, databases and networking elements through the internet, allowing customers to buy virtual machines, disk space or provided services like a managed database and providing an access method from the customer's network to the Cloud Service Provider data center.
The main characteristic of the cloud computing is that the customer pays only for the resources it uses, avoiding paying for the maintenance of old servers.
The biggest Cloud Service Provider is Amazon, with its product "Amazon Web Services", or AWS.

2.1.1 Shared Responsibility

The Shared Responsibility model is meant to illustrate who holds certain responsibilities, whether the user or the Cloud Service Provider, when a decision is made to adopt a particular service type model chosen between, on-premises, IaaS, PaaS, and SaaS.

The image following, shows a diagram that briefly recapitulates who holds the responsibility to manage a particular aspect of an application that has been deployed to the cloud.



IaaS Services

An IaaS (Infrastructure as a Service) service residing within a cloud network can be identified as a virtual machine running within the Cloud Service Provider's data center for the purpose of hosting one or more applications.
The Cloud Service Provider is responsible for creating the virtual machine from an image chosen by the customer, but it is the customer's responsibility to choose the characteristics of the hardware on which it will be deployed, configure the machine,

install the necessary software, and manage the operating system once the machine is delivered.
Applications can be installed in a virtual machine to perform any type of operation, for example Tomcat to expose the machine to the Internet, a back-end of an application to act as an application layer, or a SQL Server to work as a data layer.
The main benefits of an IaaS solution are:

- It is not expensive, because you only have to pay for the virtual machine and not for any particular service. You only pay for the time the machine is operational;
- It is an elastic solution, because you can customize every aspect of the software you use and the operating system installed on the machine.

The disadvantages of this solution, on the other hand, are:

- You have to manage every aspect of the system, including operating system updates, updates of installed software and scaling of the resources delivered;
- It is not particularly integrated with cloud services. Because it is a virtual machine and not a service adapted by the CSP to run efficiently on the cloud, it does not have native integrations with other services and does not have unique features that can be very useful (e.g., automatic backup, integration with data ingestion services, etc...);
- The virtual machine, unless special purchase options are available, runs on a physical server shared with other customers, so there may be some remote security issues.

It is advisable to use an IaaS solution only in cases of special customisation needs that a PaaS service cannot provide, where lower cost is an important requirement, or when no viable PaaS solution exists.

PaaS services

A PaaS (Platform as a Service) service residing within a cloud network can be described as a platform provided by the Cloud Service Provider with the working

environment already configured inside. The user no longer has to worry about updating the operating system or software it is intended to use because it is in charge of the Cloud Service Provider.
The CSP will take care of managing the software and the infrastructure on which it runs, leaving it up to the customer to configure the service, such as choosing how performant the underlying infrastructure should be, or setting parameters that pertain to the software one wants to use.
The advantages and disadvantages of this solution are the opposite of IaaS, namely:

- All the PaaS services are managed, you do not have to take over system maintenance, software or operating system upgrades. One simply has to pay and use the service as it is provided by the CSP.
- Can be highly integrated with other services within the cloud, simplifying the creation of applications that leverage multiple interconnected services
- In addition to paying for the underlying infrastructure you also pay for the service that the Cloud Service Provider is providing and possibly also for the software license, making the cost of the service vary according to the functionality required during deployment. Some PaaS services support "bring your own license," meaning they make it possible to apply a license purchased for the on-premises version of the software to the cloud without having to purchase it again.
- You do not have the system fully customizable; you can use the service as it is provided

SaaS services

The SaaS (Software as a Service) model allows developers to use applications that reside on the cloud without having to install them on a physical machine. An example of a SaaS application might be an e-mail web application like Gmail.
By using a SaaS application, the provider of that application will take care of any issues related to the cloud infrastructure, development, and delivery of the application over the Internet, while the users will have no aspect to manage.
The advantages and disadvantages of this solution are:

- The software is delivered ready to use, you only need to have an internet connection to be able to use the service wherever you are without having to worry about neither the power of the hardware used nor configurations of any kind

- Payment is according to the rules imposed by the application developer, so you could either have a free SaaS service or incur very expensive services

- There are no margins to modify the software, being delivered ready to use you can only use the features provided by the developers, without the possibility of customization

It is recommended to use a SaaS solution in case you find software that is competitively priced while still providing all the functionality you need.

2.1.2 Serverless computation

The serverless paradigm is based on the idea that the Cloud Service Provider will allocate, at the time of the request to execute an operation, the resources necessary to ensure its execution, to deallocate them as soon as the execution of the operation is finished.
The serverless paradigm has the following characteristics:

- A serverless resource is much more elastic and easily scalable than a server-allocated resource because the CSP can easily create new instances of the resource quickly and automatically.

- It is not possible to decide where the resources that will be responsible for executing the operation will be allocated, so there is no guarantee that the operation will be executed on a machine reserved for itself, creating possible privacy and security issues.

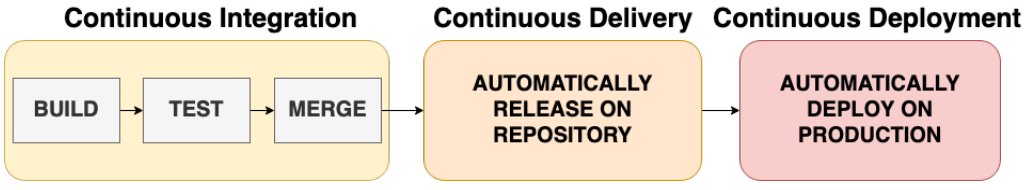
- Generally, using a serverless resource incurs much lower costs, since you only pay when it is instantiated, so only when it is actually used. It is no cost-effective only when the instance is almost always running, making cheaper to opt for a server based solution.

- A serverless resource is fully managed by the Cloud Service Provider, so it requires no special configuration once allocated.
- It is complicated to monitor the resource utilization of a serverless service because of its high volatility.

It is recommended to use a serverless architecture when you need a lightweight, flexible environment that needs to scale or be upgraded in a short time. Through this architecture, resource costs can be greatly reduced, especially when used infrequently but with high peak usage.

2.1.3 Pipelines

The main concept about pipelines is strictly correlated with the acronym CI/CD, that means Continuous Integration, Continuous Delivery and Continuous Deployment. These two concepts are related because pipelines are the main tools to integrate the CI/CD pattern.
Continuous Integration is an automation process for developer, in fact when new code changes to an app are regularly built, tested, and merged to a shared repository. It's a solution to the problem of having too many branches of an app in development at once that might conflict with each other.
Continuous Delivery means a developer's changes to an application are automatically bug tested and uploaded to a repository, where they can then be deployed to a live production environment by the operations team. It's an answer to the problem of poor visibility and communication between development and business teams.
Continuous deployment can refer to automatically releasing a developer's changes from the repository to production, where it is usable by customers. It addresses the problem of overloading operations teams with manual processes that slow down app delivery. [4]



This process is implemented with pipelines, that are the main tools to split different

stages of the software life cycle, defining several actions within each stage and performing status checks between them.

Continuous Integration

Continuous Integration has the goal to easily allow multiple developer to work simultaneously to the same project in a centralized code repository, for example Git. To achieve this goal, the CI stage allow to run automatically processes to create builds and to perform tests on the deployed code. With this automation part is possible to detect possible issues within each build provided by the developers.

Continuous Delivery

Once the code is validated through the Continuous Integration pipeline and no issues were found, the Continuous Delivery stage have the purpose to provide a centralized code base to allow developers to push their code changes, ensuring that each push to the code base will be done without conflicts, using the merge function. The main concept is that each developer can work on its own code version, called "branch", and then it is possible to merge several branches in the release version, automatically unite the code parts that not generate conflicts and notify parts modified by more than one developer. Finally a human operator will decide how to merge the conflicting code snippets into the release version of the project.

Continuous Deployment

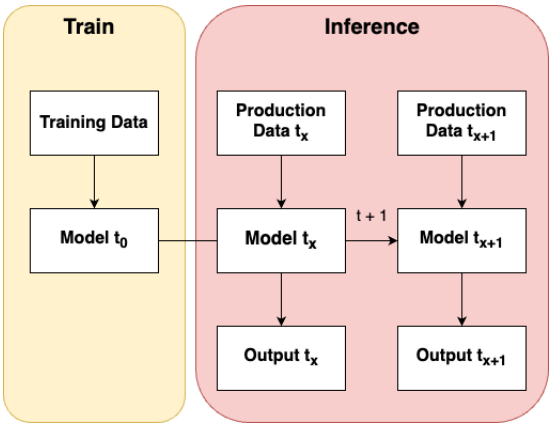
Continuous Deployment is the final stage of our pipeline. Once the code is ready for the production environment, this stage automate the release of the code on the production infrastructure. For example, in a cloud environment, Continuous Deployment stage is used to push the new release code into the cloud infrastructure, managing the switching between the old version to the new one. One strategy to manage the deployment is called "Blue/Green deployment" that consists in deploying the new code version in a mirrored copy of the infrastructure, gradually redirecting traffic into the new one to allow a rapid rollback if some issues were found.

2.2 Introduction to Machine learning

Machine Learning is a branch of artificial intelligence with the purpose to use algorithms to learn and replicate patterns, to later use the learned information to resolve new types of problems that are relatable to the learned patterns.
The aims of Machine Learning is to continuously acquire knowledge from the given data to continuously improve the model ability to solve the problem.

2.2.1 Basic concepts

What is a model?
A machine learning model is a mathematical representation or algorithm that a machine learning system uses to make predictions, decisions, or classifications based on input data. It's essentially the "learned" part of machine learning.
the life cycle of the model is characterized by two phases, train and inference, which correspond to the initialization of the model and its use, respectively.
The model continues to evolve not only during the training phase, because during the inference it can be seen that after producing the output it will proceed to update its internal parameters.



Dataset
A Dataset is a structured set of data that serves as input for training, testing, or evaluating machine learning models.
Datasets are composed by raw data that, in the majority of cases, can not be given

yet in input to the machine learning model.
To proceed and input these data into the model, it is necessary to standardize and normalize them in order to avoid biases due to poor data quality.
There are differences in the utilization of the dataset when is used to Train the models than when it is used to make Inference.
When the data are used to make inference there are no special needs other than the data transformation and the data structuring.
When the data are used during the training phase of the model, there are special cautions to take in order to ensure that it is trained properly, for example dividing into different sets in order to avoid biases during the training phase.
As a first analysis, one must check that the dataset is not unbalanced, that is, with many more samples from one class than from the others. In this case, the predictive model may have bias, leading it to more frequently predict the class with more samples purely because it was more present in the train phase.
If the dataset is huge, it is possible to optimize it removing the less impactful columns, because providing less columns to a model can reduce exponentially the execution time. To detect the less impactful columns, it is necessary to look for those that are more related to the other ones.
A correlation matrix is a table that shows the correlation coefficients between sets of variables. Each random variable (X_i) in the table is correlated with each of the other values in the table (X_j); this allows you to see which pairs have the highest correlation. Correlation refers to any statistical association, but in common usage of the term it indicates how close two variables are to having a linear relationship with each other. If two data are highly correlated with each other, it is possible to remove one as it can be described by the performance of the other, removing data and thus complexity.
This value is calculated as follows: $Corr_{i,j} = \frac{Corr(X,Y)}{\sigma_x \sigma_y}$

How to evaluate a model

To monitor the performances of the model there is a need to create a mathematical function to measure how good is the model to assign classes to the training data. A loss function is a tool in machine learning that calculates how far off a model's predictions are from the actual values. It measures the error between what the

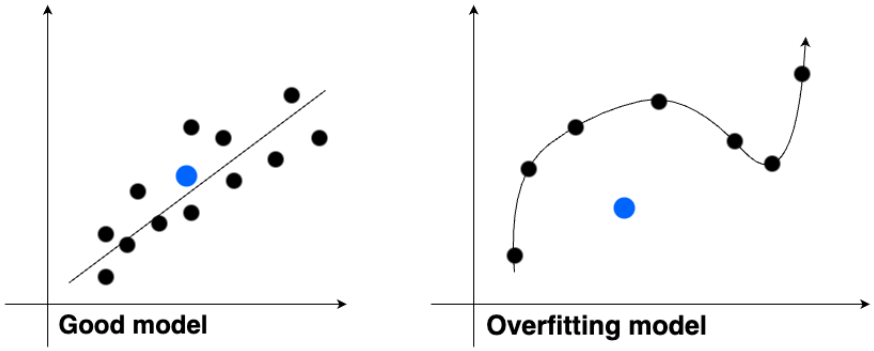
model predicts and what is actually true.
Broadly, loss functions can be classified into two major categories depending upon the type of learning task we are dealing with — Regression losses and Classification losses. In classification, we are trying to predict output from set of finite categorical values i.e Given large data set of images of hand written digits, categorizing them into one of 0–9 digits. Regression, on the other hand, deals with predicting a continuous value for example given floor area, number of rooms, size of rooms, predict the price of the house. [5]
For regression tasks, it is not possible to obtain enough information by only check if the model predict the correct value, because it is working on real values and the model can be wrong either by a negligible value or a huge value. For this reason there is a need to use Loss functions, which consists of mathematical functions that calculate the distance between the predicted value and the actual one.
It is necessary to define the concept of distance, but each model calculates the distance in its own way, but there are some standard formulas to obtain the distance between two points, and hence, the error value.

- **Manhattan distance**: $L_1(P_1, P_2) = |x_1 - x_2| + |y_1 - y_2|$ this type of distance penalize each error in the same way
- **Euclidean distance**: $L_2(P_1, P_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ this distance penalizes larger values more
- **Minkowski Distance**: given two points $P = (x_1, x_2, ..., x_n) Q = (y_1, y_2, ..., y_n)$ we can define $L_m(P, Q) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{1-p}$ can be considered a generalization of the Euclidean distance

Train and Test

The learning phase is composed by two different activities: training the model and testing the model's ability to work with fresh data.
For training and testing purpose, the dataset must be enriched with labels to let the model know about the classes of the data.
The first activity is performed by providing the model the data without their labels and, after the assignment of the class, calculate the performance of the prediction using the Loss function and then tell the model how far it was from the true

value. The model update its parameters, which are responsible for the wrong class assignment, to improve the next predictions.
This activity ends when the model is good to predict the correct value, this measure can be measured with Loss or with the accuracy score, that is useful to calculate the validity of our model in classifications tasks, and can be assume values in the range of 0 to 1, where 1 is the highest.
To perform better tries to generate the best model possible, it is necessary to modify the hyperparameters, consisting of parameters that can be used to affect the training process. These parameters are mostly unique to each model.
When the best set of hyperparameters are found with the training data, there is the need to perform the second activity, that consists in testing the goodness of the model with data that was not involved in the training process, in order to avoid biases based on the data used in the previous step.
During the test phase, the model perform the same predictions done in training without update its parameters. if the model can classify the test data with good accuracy one can keep the chosen hyperparameters, otherwise one will have to run this process again with a new set of hyperparameters.
When the model is particularly good at classifying the training data, but the performance deteriorates when tries to classify another data, for exemple the testing ones, means that the is too specialized withhin a particular subset of data, and this phenomenon is called "overfitting".



In the image above, when the model tries to classify the class of the data corresponding at the blue dot, with a good model it is possible to always retrieve an acceptable average value. With a model in overfitting, it is not possible to classify properly data far from the training ones.

Inference

Inference is the application of the trained machine learning model on new data to create a result. Machine learning model inference is also known as moving the model into the production environment. This is the point that the model is performing the task it was designed to do in the live business environment. Deploying the machine learning model includes moving the model to a live environment where the model starts processing new and unseen data. This is different to the model training phase, which is usually performed in a local or offline environment. [6]

2.2.2 NLP

NLP (Natural Language Processing) is a Machine Learning branch that deals with developing algorithms and models to comprehend and imitate the human natural language.

The main goal of NLP is to create systems that are capable of comprehend the natural language as spoken or written by people and generate text similar as like an human being would do it. To achieve this goal it is necessary to understand the meaning of the used terms, the sentences and corpus semantics and also the ability of generating coherent and understandable text.

NLP uses machine learning techniques and models like neural networks, decision trees, clustering algorithm and other to elaborate linguistic data. Those algorithms are trained on an huge amount of texts to enrich the capability of recognize and modelling language patterns.

The steps used by NLP in order to work properly are:

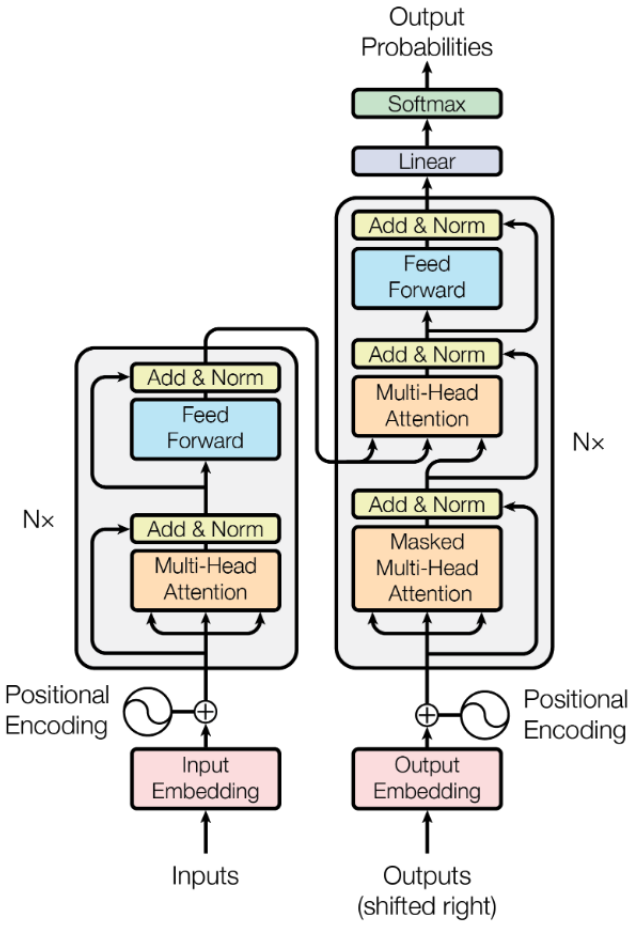
- In order to understand the meaning of a word and to perform mathematical operations, NLP breaks down the text into shorter sentences and tokenize them, especially the model transform each word in the correspondent number. These techniques are used to comprehend the semantic of the sentences and to analyze the grammatical structure.
- After that, NLP uses neural networks to train language models used to recognize language patterns and to generate text.

The trained models are used to perform a wide variety of tasks, for example to perform sentiment analysis, search engines, chatbot, automatic translation, voice

and text recognition and so on.
In short, NLP uses a combination of natural language processing and machine learning techniques to comprehend and manipulate the humans language. These techniques allow computers to analyse, process and generate text like a human being would do it.

2.2.3 Transformer

Transformer is the state of the art technology for sequence modeling problems, and it's largely used for natural language processing.
Transformer is based on Attention mechanism, that makes easier to learn dependencies and similarities between different positions in the same sequence.
Transformers not contain convolutions, so it can be parallelized because there are no dependencies between elements at the beginning of the sequence or at the end of it.
The model architecture is based on encoder-decoder structure, in which the encoder maps an input sequence in an input array $x = (x_1, ..., x_n)$ in a sequence, that is given to the decoder that generate a sequence of symbols $x = (y_1, ..., y_m)$.
At each step, the model consume the generated symbols, using as inputs.

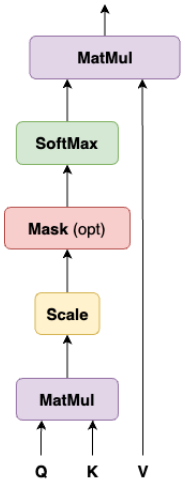


Attention

Transformer is based on attention functions, that can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key. [7]

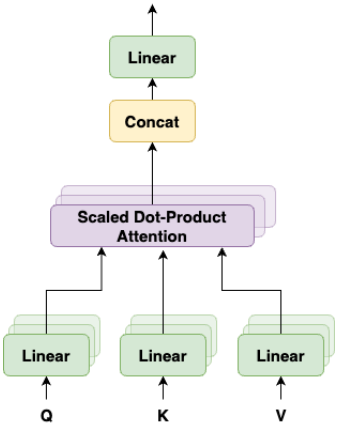
The most famous way to compute attention is the scaled dot-product attention, that consists of computing on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$



To make Transformers parallelizable, the Attention mechanism can be calculated concurrently for a subset of the input values and then concatenated in a single output. This mechanism is called Multi-Head Attention:

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_n)W^O \quad where\, head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$



The Transformer uses multi-head attention in three different ways:

- In "encoder-decoder attention" layers, the queries come from the previous decoder layer and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models.

- The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
- Similarly, self-attention layers in the decoder is implemented by computing attention scores between positions in the input sequence. However, masking is applied to prevent the model from seeing future positions. Queries, keys, and values are used to calculate attention scores, which guide the decoder's focus on relevant input positions. The weighted sum of values, considering positional encodings, helps generate coherent and accurate output tokens while respecting the sequence order.

Chapter 3

Tools

3.1 AWS

AWS (Amazon Web Services) is a cloud computing platform provided by Amazon. Through AWS it is possible to gain the access to a wide range of services, for example from computing services, storage components, security and network infrastructures and a lot of other services.
AWS environment is designed to be flexible and scalable, allowing companies to using only the services needed and to create in few steps new resources if needed. The companies can manage huge workloads without investing in hardware infrastructure and without hiring IT employers to manage it.
AWS is divided in several operative regions in witch Amazon own huge datacenters located in different areas within the region, called Availability Zones. Deploying resources in several Availability Zones allows companies to make them redundant, so they can avoid disruptions of service.
AWS offers a wide range of Machine Learning oriented services. Some services are fully managed, so the customer does not have to manage the underlying structure. A list of Machine Learning services provided by AWS is:

- Amazon Sagemaker is a fully managed service that allows companies to create, train and deploy Machine Learning models in a simple and rapid way, without having to manage the underlying infrastructure composed by computational servers and network infrastructure.

Amazon Sagemaker supports a wide range of algorithms and provides instruments for data visualization, model creation, model training, model inference and model management.

- Amazon Rekognition is a service specialized in video and image analysis. It is base on deep learning and consent companies to detect objects, faces and text in images and videos.
  Amazon Rekognition is used in video surveillance applications, in management of multi medial content and in pattern recognition in videos and images.

- Amazon Comprehend is an text analysis service based on a Machine Learning model which uses an NLP algorithm.
  Amazon Comprehend allows companies to analyze huge quantities of text to identify entities, relations, sentiment and language in it.
  Amazon Comprehend is used in applications that needs to classify texts, to perform semantic finding, analyze social media text and so on.

- Amazon Transcribe is a service used to automatically transcribe speech into text.
  Amazon Transcribe supports a lot of variety of languages and audio formats and it is used in applications that needs to transcribe phone calls, to generate automatic subtitles and to transcribe meetings.

- Amazon Polly is a vocal synthesis service that allows companies to convert text in realistic speech audio in real time. It is the opposite of Amazon Transcribe.
  Amazon Polly supports a lot of variety of languages and voices and it is used in applications as vocal bots, clients assistant and for conversational agents like Amazon

Thanks to these services, it is possible to create Machine Learning applications without the necessity of creating algorithm to train a model and make the inference. These services simplify the interactions with the model to easily deploy a Machine Learning application without having a huge infrastructure behind it.

3.1.1 AWS Lambda

AWS Lambda is a serverless compute service offered by Amazon Web Services that allows to execute some code in Python, Java or nodeJS without the needs to manage the underlying infrastructure like servers, load balancers or network configurations, reducing significantly the infrastructure costs and optimizing the application performances. Summarizing, using AWS Lambda makes you in charge to only write your own code uploaded in a ZIP package that must contains all dependencies required to run the code.
Lambda is an event-driven elaboration service, whose code will be executed in response to certain events received like an image loaded on a S3 bucket, a message received on a message queue like SQS or to respond to requests made by an API Gateway. To respond to events it is necessary to configure triggers, that guarantee that the code is executed only to respond at those events.
AWS Lambda is extremely and easily scalable and it adapt itself to the volume of the requests, creating more computing instances if required. It is also pay-for-use, that means that you only pay for the effective usage time multiplied for the number of instances deployed.
AWS Lambda also offer a logging and monitoring system that allows to logs all the code's requests, errors and results, that will be stored in AWS CloudWatch, a monitoring service that contains logs, alarms and metrics regarding other AWS services.

AWS Step Functions

AWS Step Functions is a serverless workflow orchestrator service offered by Amazon Web Services. It allows to easily create, execute and monitor workflows in a visual way, through a intuitive graphic interface.
AWS Step Functions allow to easily coordinate the execution of a set of activities, allowing to define the workflow as a series of states and transactions. Each state represent an activity or a step inside the workflow, while an activity consent to define the conditions for transition from one state to another.
AWS Step Functions offers a lot of advantages, like:

- Ease of use: workflows are created and managed through an intuitive graphical

interface, without the need to write code.

- Workflow orchestration: AWS Step Functions enables efficient coordination of the execution of a set of tasks by defining a state and transition scheme.
- Flexibility: you can create workflows that include different activities, such as API calls, sending messages, executing code on Lambda, triggering another workflow, and more.
- Monitoring: AWS Step Functions provides a built-in monitoring and logging system to track workflow execution and any errors or exceptions.
- Scalability: workflows defined in AWS Step Functions are highly scalable and can run on large volumes of data, without the need to manage the underlying infrastructure.

Some of the main disadvantages of AWS Step Functions include:

- Costs: AWS Step Functions has costs associated with the use of the service. Although these costs are generally low, they can increase significantly based on the volume of use of the service.
- Complexity: AWS Step Functions has a rather steep learning curve and can take time and effort to become familiar with the GUI and service configuration. In addition, defining workflows can be complicated if the workflow is particularly complex or includes many tasks.
- Dependence on AWS: AWS Step Functions is an Amazon Web Services service, which means that its integration with other non-AWS technologies can be more complicated.
- Performance: AWS Step Functions is a cloud-based service and, as such, may experience interruptions or slowdowns due to connectivity or cloud service performance issues.

In summary, AWS Step Functions is a serverless workflow orchestration service highly flexible and scalable, that consent to coordinate the executions of a set of activities in an efficient way, simplifying the development of applications and improving productivity.

3.1.2 Sagemaker

Amazon SageMaker is a cloud computing service offered by Amazon Web Services (AWS) that enables data scientist to develop, train and deploy machine learning models quickly and scalably. The service consists of a wide range of tools that cover the entire machine learning lifecycle, from data collection to model deployment. Specifically, Amazon SageMaker provides a platform for data preparation, model training, model deployment, and model management.
SageMaker allows the use of custom models trained by the customer's data scientist, but it also provides some pre trained models to make easier to implement a machine learning solution without owning a huge quantity of data useful to train it.
Data preparation can be done with data visualization and data cleaning tools, while model training can be done with predefined or customized machine learning algorithms. In addition, Amazon SageMaker enables tuning of model parameters, i.e., optimization of specific performance metrics.
After training, the model can be put into production through the use of scalable infrastructure such as Amazon EC2, Amazon Elastic Container Service (ECS) or Amazon Elastic Kubernetes Service (EKS), which enable model deployment to be managed in a cloud environment. In addition, SageMaker allows model performance to be monitored and updates to be made automatically to ensure maximum efficiency.
The main advantages of the utilization of Amazon SageMaker are:

- Ease of use: SageMaker greatly simplifies the process of creating, training and deploying machine learning models by eliminating the need to configure and manage the underlying infrastructure.

- Scalability: SageMaker offers distributed processing, which allows models to be trained on large amounts of data and deployed to different platforms quickly and scalably.

- Automation: SageMaker offers automation features, such as automatic hyper-parameter optimization, which allows you to find the optimal configuration for a given model.

- Integration with AWS: SageMaker is fully integrated with the AWS ecosystem

of services, which makes it easy to integrate machine learning models into a wide range of applications and services.

The main disadvantages of using this service are:

- Cost: SageMaker is a paid service, the cost of which depends on the resources used and the duration of use. Although the cost is generally competitive with other machine learning solutions, it can still be a limiting factor for some companies.
- Dependence on AWS: because SageMaker is an AWS service, companies using SageMaker become dependent on AWS and may find it difficult to move to other platforms in the future.
- Customization: although SageMaker offers a wide range of machine learning algorithms and machine learning libraries, some companies may require specific functionality that is not available out-of-the-box with SageMaker. In this case, it may be necessary to develop and customize machine learning models using other libraries or services.

In summary, Amazon SageMaker represents a comprehensive and integrated solution for developing machine learning models that can manage the entire machine learning lifecycle, from data processing to model implementation.

3.1.3 DataWrangler

Amazon DataWrangler is a fully managed data processing service from Amazon Web Services (AWS) that makes it easy to prepare data for analysis and modelling. DataWrangler greatly simplifies the data preparation process by providing an intuitive graphical interface for data exploration, cleaning, and transformation. With DataWrangler, you can work with a wide range of data sources, including CSV files, Excel, JSON and relational databases.
The service also offers a wide range of data processing capabilities, including removing missing values, aggregating data, creating variables, and normalizing data. In addition, DataWrangler offers advanced transformation capabilities, such as text transformation, combining columns, and creating new columns using custom

functions.
Once the data is prepared, it can be exported to a variety of formats, including CSV, JSON and Parquet, and uploaded directly to other AWS services, such as Amazon S3, Amazon Redshift and Amazon Aurora. In addition, DataWrangler makes it easy to create Python scripts for data processing, which can be used in other applications or services.

3.2 NLP Models

An NLP model is an algorithm that uses natural language processing techniques to understand and generate text automatically. NLP deals with the manipulation, comprehension and generation of human language by computers or machines.
NLP models are mostly based on neural networks and are usually structured in several layers.
An NLP model is trained on a large collection of English texts to learn language patterns and semantic relationships between words.

Chapter 4

Pre existing structure

4.1 Architecture Overview

There was an pre-existing structure already deployed on a Cloud Service Provider, more specifically on Amazon Web Services.
The re-engineering work does not consists on bringing the infrastructure on AWS, but consists in deliver a product that use the right tools to perform the operations.
The existing solution was composed by different step functions that manage and coordinate a lot of lambda functions that perform a wide range of different operations, i.e. for data ingestion, for execute algorithms and to provide and transform the output.
Each Lambda function needs access to a shared storage Elastic File System (EFS) to interact with the model.
Lambda functions needs also access to a blob storage on cloud, called Simple Storage Service, or S3.
This solution is not sustainable for intensive uses, because several reasons:

- AWS Lambda functions has an hard limit of 15 minutes of execution, each Lambda function that run for more that this time will be terminated with an error code.
  To keep the execution under 15 minutes there was the need to constantly split the dataset into several smaller data sets and perform the inference of the model by executing the Lambda function for each of the dataset.

As data grow this problem reappeared, forcing the data scientists to split the dataset periodically.
This workaround can not work forever, because there is a limit on how much a Lambda can be parallelized due to the enormous costs that this type of execution can generate and the data scientists can not waste their time to continuously split the dataset.

- Step Functions have a limit on the number of lines that Lambda functions can log in a single execution. After Lambda functions have logged more than 25.000 lines of log, the Step Function will return an error and will not complete the execution. This limit is an hard limit, so it is not possible to contact AWS to increase it.
- With this type of architecture all the phases of the process are managed by Lambda functions.
  They are not the recommended service to provide an endpoint for the Machine Learning model, because there are services able of providing scalable HTTPS endpoint.
- Models are saved on a shared storage device, called AWS EFS. All Lambda Functions have to access to this storage service to download the entire model during the run time.
  The models are bigger than 1 GB, requiring EFS to use a high throughput and require Lambda functions to use a lot of time to download it entirely each time, reducing the useful time to perform the execution of the algorithm.

4.2 Goals

The main goal of this service is performing research and development by investigating public opinion about the company by collecting and analyse comments and posts written on different social media and articles written on newspapers and journals.
To obtain these information this company uses Sprinklr, a software specialized in performing social media and newspaper monitoring that uses AI tools to listen

from several data sources (for example YouTube, Facebook, Twitter, Instagram, Reddit, Wikipedia, ...), looking for a set of keywords posted on these media or newspaper article. These keywords are the name and the acronym of this company. Sprinklr provides an API which once contacted return a Json with the message written, the public data of the user that wrote the message and the permalink to the post (or to the article).
After the data collection, the next step consists in standardize and analyze the data collected. First, the messages written in Italian will be translated in English and the result filtered.
After the translation it will performed an enrichment on the data originating from newspaper articles to check if the original content delivered in English is more substantial then the translated one.
The last operations consists of extracting keywords and sentiments belonging to the message and assign categories that can give an idea of the content written on it.
This operation is performed to generate a report that helps monitor public opinion about the company.

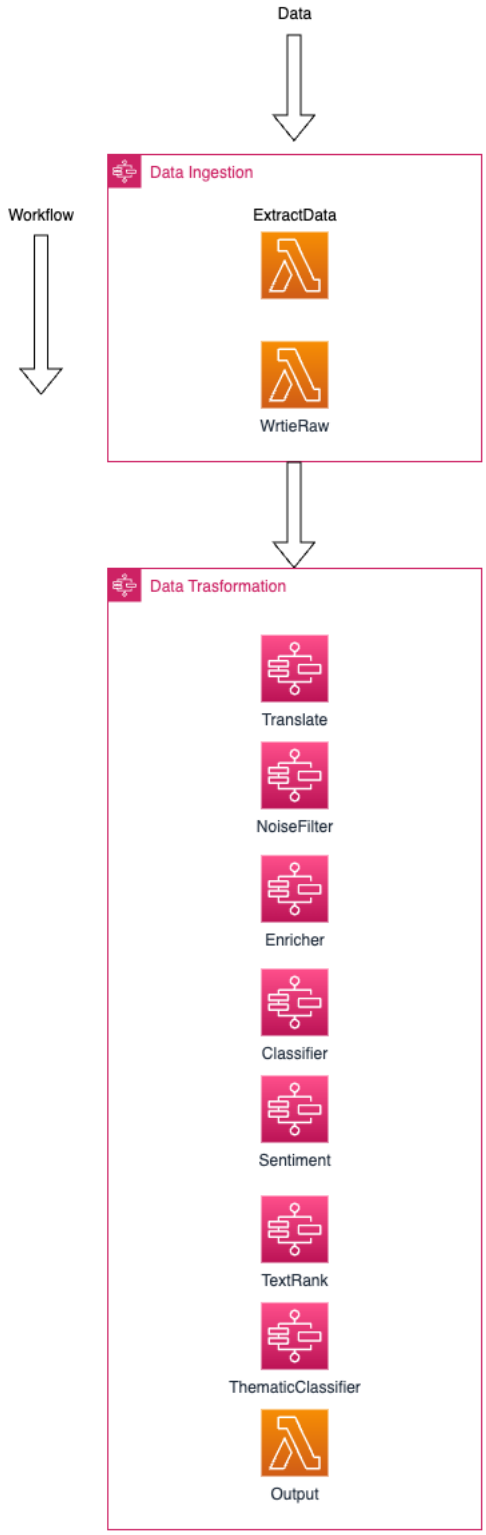
4.3 Data Structure

The input data are structured are received by Sprinklr as a list of Json dictionaries. These data are composed by a lot of useless fields for the algorithms purposes.
During the Data Ingestion step, these data are converted to a CSV, that is more human readable and it is also the standard format used by Pandas.
After the transformations performed by the algorithms, the data is stored in Parquet format, an open-source file format optimized for efficient storage and processing of large amounts of data.

4.4 Step functions

There are two different step functions, each of them used for a different phase of the workflow.
The first step function is composed by two Lambda functions, and it is used to

perform the ingestion of the data that will be provided to the models.
The second step function is composed itself by several step functions, each of them is used to perform a specific action inside the workflow, for example to translate data or to perform data enriching.
This step function is executed after the ingestion of the data, and provide the final output of the models.
There are two S3 buckets called "LandingData" and "RawData". LandingData is used to save the input data awaiting to be processed.
RawData contains the input data, the raw output data and the data enriched by the Step Function.
In Raw Data there are also the curated data parted by different keys, which it is possible to consult by performing queries with Amazon Athena.


Data
Data Ingestion
Workflow
ExtractData
WrtieRaw
Data Trasformation
Translate
NoiseFilter
Enricher
Classifier
Sentiment
TextRank
ThematicClassifier
Output

These Stepfunctions are composed by:

- Ingestion: The ingestion Stepfunction contains two Lambda functions. The first function contacts an API exposed by Sprinklr that deliver a Json list object and extract a single element of this list, each of them correspond to a row of the dataset. After that the Lambda function write this data into the LandingData S3 bucket.
  The second Lambda function read each data from the LandingData S3 bucket and write it on the RawData S3 bucket.

- Data transformation: This Step function is executed after the first one, and is composed by several Stepfunctions, each of them is in charge to perform a single operation on the data produced in output by the precedent step.
  Each Step function that compose DataTransformation are composed in a similar way, except for the first one that execute the translation of the ingested data.

  - DataLoading: This Lambda Function is in charge to perform several preliminary operations to standardize the data. These operations are specifics for each Step Function and will be discussed later.
    This Function is also in charge to partitions data into smaller files. These files will be stored into LandingData S3 Bucket as temporary files and will be used into the next steps.

  - MainStep: If there are unprocessed data, it is executed the Lambda function that is in charge to execute the main activity of the step function, that can be translation, ranking, etc...

  - LastVersion: This step is executed after that all the input data are processed by the main step. This Lambda Function combines the various outputs created by the previous step and writes an output file into the S3 Bucket LandingData.

  - DeleteTmpFiles: This step is the last one to be executed in a Step Function and it is in charge to delete all the temporary files created in the previous steps.

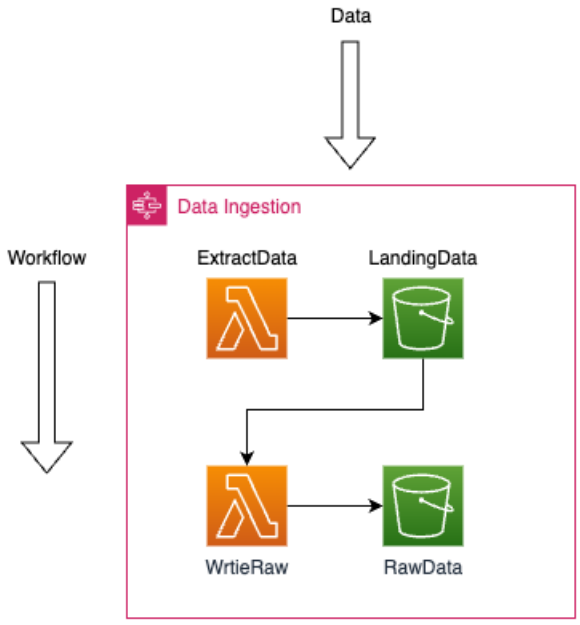
The last operation performed by this Step Function consists in a Lambda Function that write the output data on the RawData S3 bucket in order to be queried using Amazon Athena.

4.4.1 Data ingestion

The ingestion of the data is performed by a Step function scheduled to be executed each hour, and it is used to perform the ingestion of the data originating from the services Sprinklr.
This Step Function is composed by two different Lambda Functions:

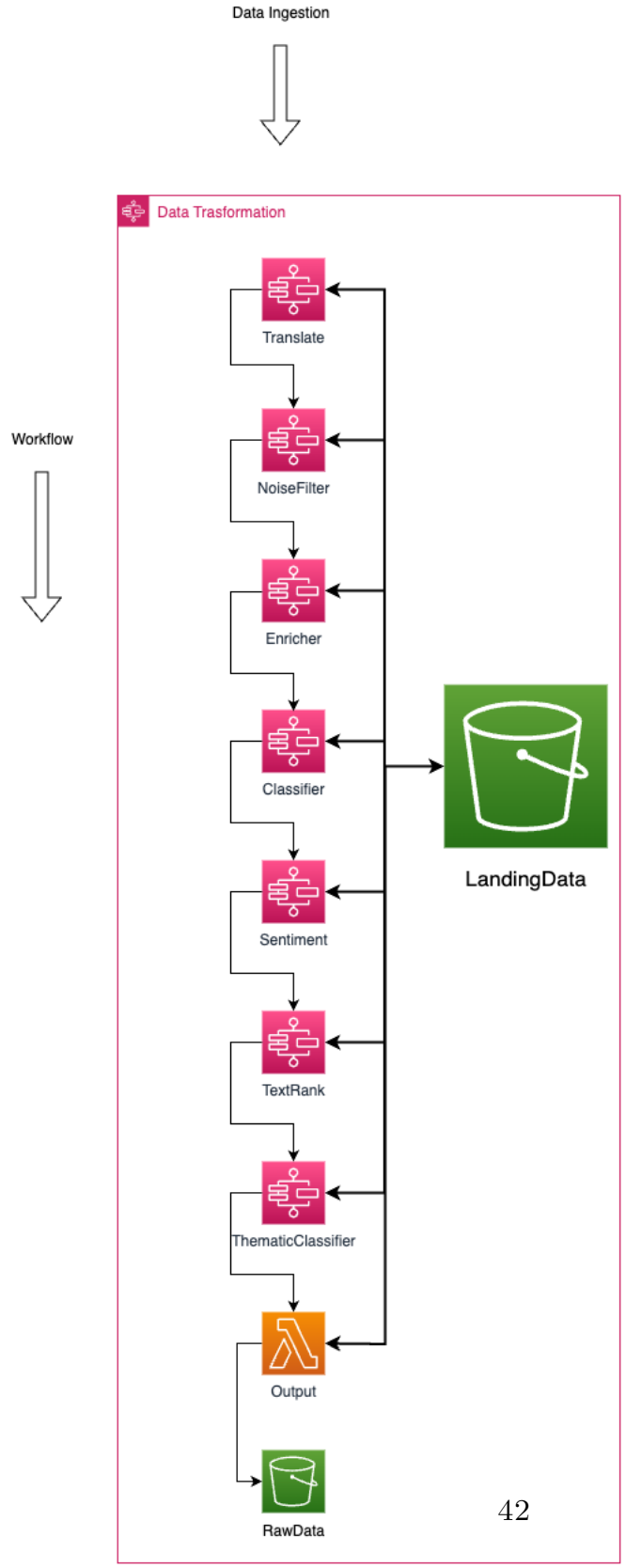
- The first Lambda Function is executed automatically when the Step Function is invoked. It contacts the API exposed by Sprinklr that is used to download a batch of data collected by this service.
  After the downloading of the data, this function read the Json input data and write them in CSV format into the LandingData S3 bucket.
  Finally this Lambda Function invoke the DataManipulation Step Function in order to process the collected data.

- The second Lambda Function is executed immediately following the first one and perform the copy of the data written by the first function from the LandingData S3 Bucket into the RawData S3 Bucket.



4.4.2 Data manipulation

The manipulation of the data is a series of operations performed on the ingested data with the purpose of translate it into English, assign categories and a sentimental tone, extract keywords belonging to the text and finally organize the data to be queried easily.
Because of these Step Functions are very similar between them, The description of the common flows has been already explained. In the following sections only the unique parts will be described.


Data Ingestion
Workflow
Data Trasformation
Translate
NoiseFilter
Enricher
Classifier
Sentiment
TextRank
ThematicClassifier
Output
RawData
LandingData

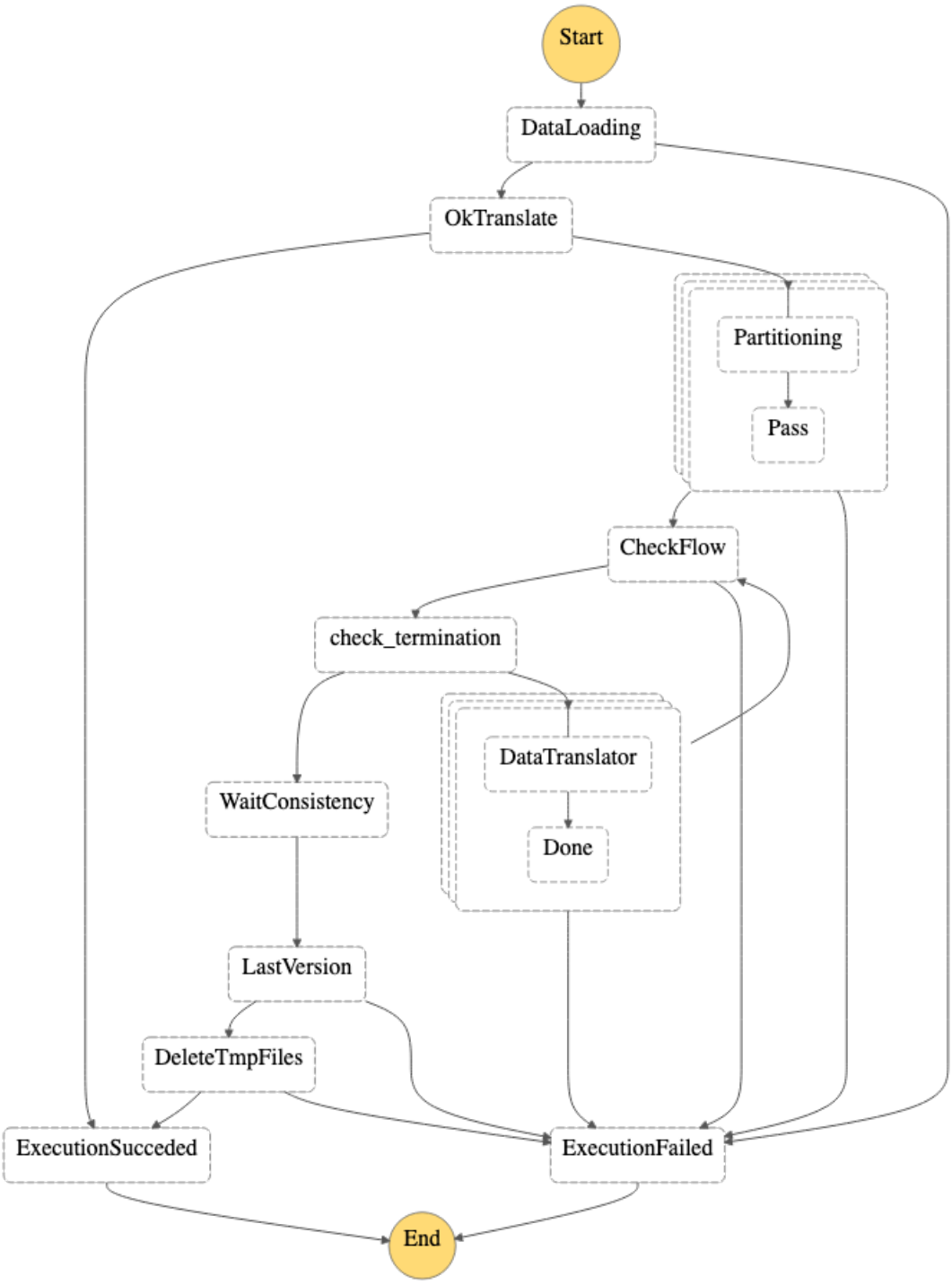
Translate

the first Step Function filter for information obtained from data sources written in Italian or in English. It perform the translation of the data from Italian tongue to English, not modifying the data already written in English.
This Step Function is different between the other ones, because not only the main step is executed concurrently, but even the partitioning step, because each file requires to include only one data row otherwise the translation will fail due to Lambda limitations.
For this reason, it is required to separate this step from the DataLoading and it is required to add one more step called CheckFlow that must check if all the partitions have been created by the Partitioning step and processed by the main Function.
If that is the case it sets a parameter that modify the flows of the Step Function, executing the LastVersion step, otherwise it fetch another partitioned file and trigger the execution of The main step's Lambda Function.
The DataLoading Lambda Function will perform these preliminary actions in order to prepare the data to be elaborated from the DataTranslator step:

- it check the mandatory columns to be presents, for example the Language and the Message ones, otherwise it creates them.
- it filters only for messages written in Italian or English, this information is stored into a column provided by Sprinklr.

The main step of this Step Function is performed by the DataTranslator Lambda Function.
To performing the translation, this Function uses the model MarianMT in conjunction with the tokenizer MarianTokenizer, both provided by Hugging Face.
MarianMT is a model composed by a transformer encoder-decoder with 6 layers specialized in translations developed with the Marian C++ library. [8]
Hugging Face provides for a pre-trained version of the model, but there is a fine-tuned one created by the customer's data scientists and stored in one shared file system which resides on Amazon Web Services.
This model translates texts only starting from Italian to English, because the data in input to this Lambda are already filtered to eliminate other languages, and

during the translations all texts in English are skipped.
The output is composed by the same input file with the value correspondent to the message modified with the translated value.


Start
DataLoading
OkTranslate
Partitioning
Pass
CheckFlow
check_termination
DataTranslator
Done
WaitConsistency
LastVersion
DeleteTmpFiles
ExecutionSucceded
ExecutionFailed
End

NoiseFilter

This Step Function is used to remove the noise produced by the translation step to provide a well formatted text to the next steps.

Referring to NLP, noise is composed by several concepts like information that are irrelevant, incorrect or unsolicited that are generated along with the text, in our case with the translation of the message.

In our use case we have to leverage grammatical errors, syntactical errors and we have to remove abbreviations, acronyms, or colloquial language that come from social media messages.

Noise can impact the performance of NLP models if not handled correctly, making them less accurate and reliable. To avoid this, it is possible to pre-process the input text with one among many techniques.

The DataLoading Lambda Function split the input CSV file in several smaller ones, composed by 30 rows of data.

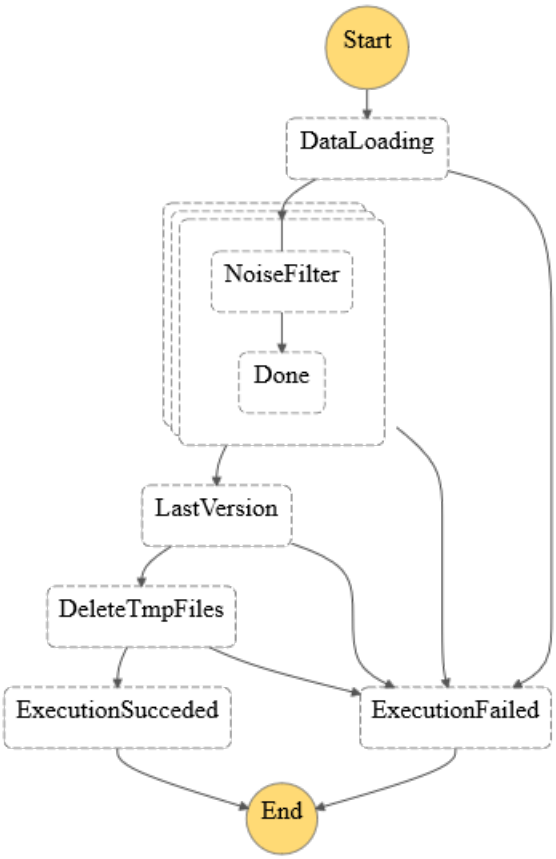
The main step is called NoiseFilter, and use the Tokenization technique to reduce noise performed with a pre-trained version of DistilBERT, whose model is stored in the shared file system (AWS EFS).

DistilBERT is a smaller and faster version of the BERT model. The main idea behind DistilBERT is to compress the original BERT model while preserving much of its language understanding capabilities. It achieves this by using a process called "distillation," where knowledge from a larger teacher model (BERT) is transferred to a smaller student model (DistilBERT). [9]

DistilBERT is used to tokenize sequentially each word of the messages and, for each of them, generate text corresponding to the word tokenized. This technique is useful to filter the word by regenerating after the tokenization performed by DistilBERT.

The main concept is that a token is mapped with a concept, and by generating a word using that token we can use the best fitting word.

Another effect of the tokenization is language normalization, that consists in converting contractions to their expanded forms (e.g., "can't" to "cannot") or replacing abbreviations with their full forms (e.g., "u" to "you").



Enricher

This Step function enriches text extracted from newspaper articles via Sprinklr and it is not applied to messages extracted via social media.

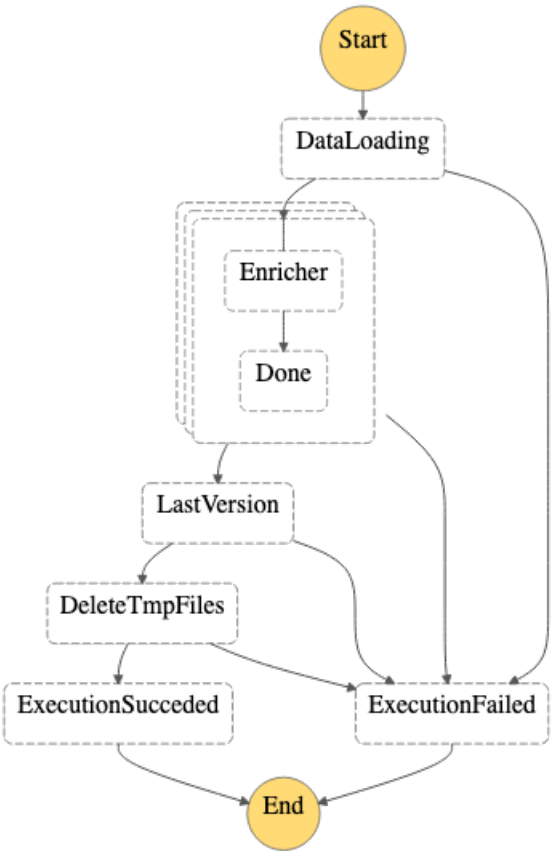
To enrich the text referred to articles, this Step Function performs data scraping using the permalink provided by Sprinklr.

Data scraping is the process to automatically extract data from web sites using scripts to access to the web pages. The data collected through the scraping are processed and analyzed further.

The DataLoading Lambda Function split the input CSV file in several smaller ones, composed by 30 rows of data.

The main step, called Enricher, perform the data scraping using the Python library Newspaper3k using the permalink provided by Sprinklr. The data scraping is performed on the English version of the articles, if available.

Newspaper3k is a library that helps to perform data scraping by providing API to download an entire article composed by its title and its text auto detecting the language in witch it is written.
This Lambda Function is executed on sequentially on each rows belonging to the batch given in input, filtering for articles.
The length of the extracted text is compared with the one of the original message provided by Sprinklr.
If the extracted message is 20% longer, it means that it carry a lot of new information, so this new message will be used instead of the original one, otherwise nothing will change.



Classifier

The task of this Step Function is to manipulate and analyze the input text to extract membership to one or more categories based on keywords.

The DataLoading Lambda Function split the input CSV file in several smaller ones, composed by 30 rows of data and perform several column name changes.
The main phase of this Step Function is composed by a Lambda Function that, firstly, divides the data according to the origin of the message, filtering only the ones from social media.
This Lambda function performs two different operations, the first one prepare the data and the second one perform the classification.

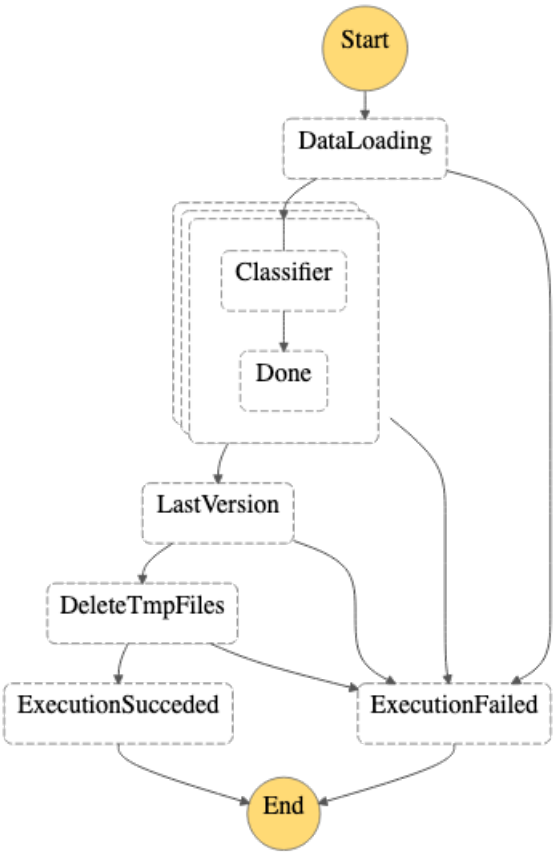
The preparation consists of removing non ASCII characters, for example emojis, special characters like hash signs and punctuation.
One the characters are removed, the text is transformed in lower case and it is used the library nltk to remove the stopwords defined by this tool, in addition to other words defined based on the social media slang.
NLTK, short for "Natural Language Toolkit," is a popular open-source Python library designed for working with human language data in the field of Natural Language Processing. It provides a variety of tools, such as tokenization, stemming, lemmatization, part-of-speech tagging, and syntax parsing, along with access to diverse corpora and lexicons for NLP research and experimentation.[10]
Stopwords are the words in any language which does not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence.
Lastly, the text is lemmatized using nltk. This operation restore the different forms in which a word occurs in a text to their basic form, for example, converting a verb to the present infinitive form, an adjective or a noun to the masculine singular form.

The classification is performed by using different models stored into the EFS and trained locally by the customer's data scientists.
The processed message can be categorized in one or more of these classes:

- Products
- Innovation
- Workplace
- Governance

- Citizenship
- Leadership
- Performance

To classify the input text into these classes, first it is necerray to perform a Term Frequency Vectorization with a pre-trained model, to obtain the word vectorized and the frequency of the words inside the text.
After the Vectorization it is used another model to build a matrix with the most used terms in relations with the Vectorized words.The result is a matrix with words and their related occurrences as they appear in the text.
The last step is to give this matrix to another model that will predict the most suitable labels for the text, chosen between the seven given in input.
The resulting output is the input file with seven more columns, each of them identify if the text belongs to the corresponding class.



Sentiment Analysis

This Step Function is used to analyze the messages to extrapolate information on the emotional tone used when writing them.

This process, called sentiment analysis, utilizes algorithms to determine the emotional tone or sentiment expressed in a piece of text, including whether it's positive, negative, neutral, or even more nuanced emotions.

This technique is used to automatically understand and classify the sentiments conveyed in text data, making it valuable for analyzing opinions, customer feedback, and trends in various contexts like social media.[11]

The DataLoading Lambda Function split the input CSV file in several smaller ones, composed by 1000 rows of data.

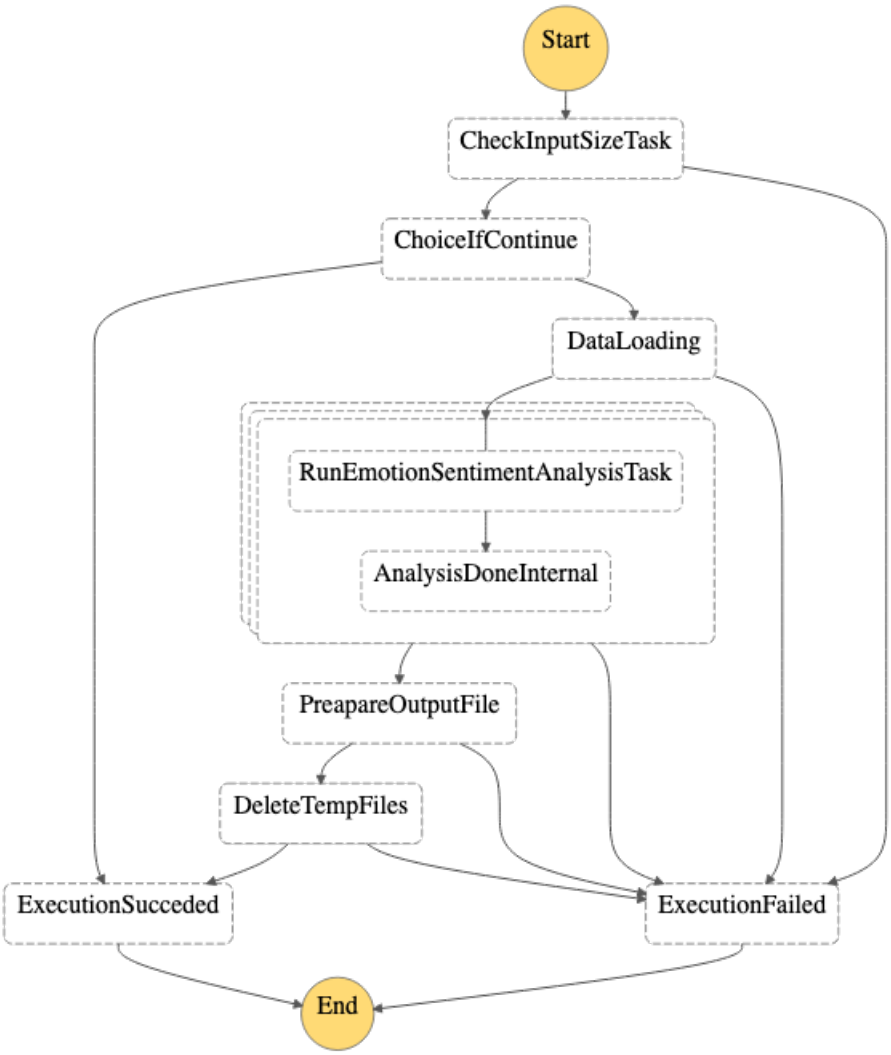
The main step is composed by a Lambda Function that perform tokenization to the input text and filter out the tokenized messages longer than 512 tokens.

After that there is performed sentiment analysis using RoBERTa, a potent language model. RoBERTa can be tuned for sentiment analysis by training it on labeled data with sentiments. During fine-tuning, the model learns to correlate text patterns with distinct sentiments. After fine-tuning, inputting text into the RoBERTa model predicts the sentiment based on its learned patterns. Consequently, RoBERTa automatically detects the sentiment expressed in a given piece of text.

RoBERTa is an advanced variant of the BERT model. It improves upon BERT's pretraining process by using larger batch sizes, more training data, and longer training times.

RoBERTa achieves state-of-the-art performance on various natural language understanding tasks and has become a popular choice for many NLP applications. It is designed to better understand context and nuances in text, making it highly effective for tasks like text classification, question answering, and more.[12]

The output is composed by new columns that indicates the tone of the messages, chosen by anger, joy, optimism or sadness.



Text Rank

This Step Function aims to analyze the message and performing a keyword extraction using TextRank.

TextRank is a graph-based algorithm used for text summarization and keyword extraction. It treats sentences in a text as nodes in a graph, where edges represent relationships such as co-occurrence or similarity. TextRank assigns scores to these nodes based on their connections, similar to how search engines rank web pages.[13] Keyword extraction is an automated process that identifies and pulls out the essential words or phrases from a text document. These keywords reflect the main

themes, subjects, or ideas within the document and aid in swiftly comprehending document content.
The objective is to pinpoint the most pertinent terms that capture the document's core meaning.
To perform TextRank is used "en_core_web_sm", that is a language model provided by the spaCy library.
In spaCy, language models are pre-trained models that have learned to understand and process text in a specific language. The en_core_web_sm model is a small-sized English language model that includes vocabulary, word vectors, syntax, and named entity recognition information.
This model is useful for tasks like tokenization, part-of-speech tagging, named entity recognition, dependency parsing, and sentence segmentation, among others. Since it's a smaller model, it loads faster and requires less memory compared to larger models, making it a good choice for applications with limited computational resources.
The TextRank is performed by executing the following steps:

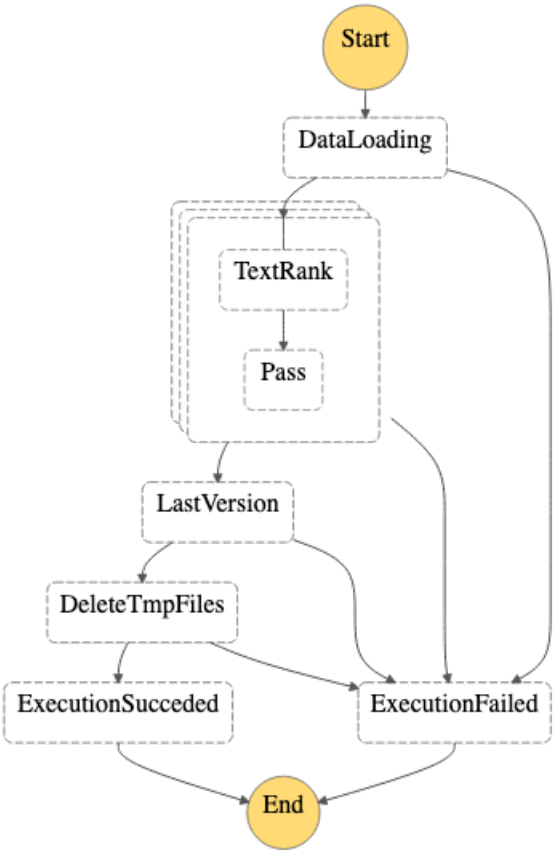
- Text pre-processing: Prepare the text by removing any unnecessary formatting, punctuation, links, special characters and by converting all the text to lowercase. It also perform stemming (remove -ing, -ly, ...), remove stopwords and lemmatize the text.

- Part-of-Speech Tagging: Tokenize the sentences and assign part-of-speech tag (noun, verb, proper noun) to each token. This helps in identifying the grammatical structure of the text.

- Building a Graph: Create a graph where each token is a node. The connections between nodes can be established based on co-occurrence or semantic similarity.

- Selecting Keywords: Apply the TextRank algorithm to calculate scores for each token based on the graph structure. These scores indicate the importance of each token in the context of the entire text. The words with the highest score are chosen as keywords, because represent the most significant and relevant terms in the text.

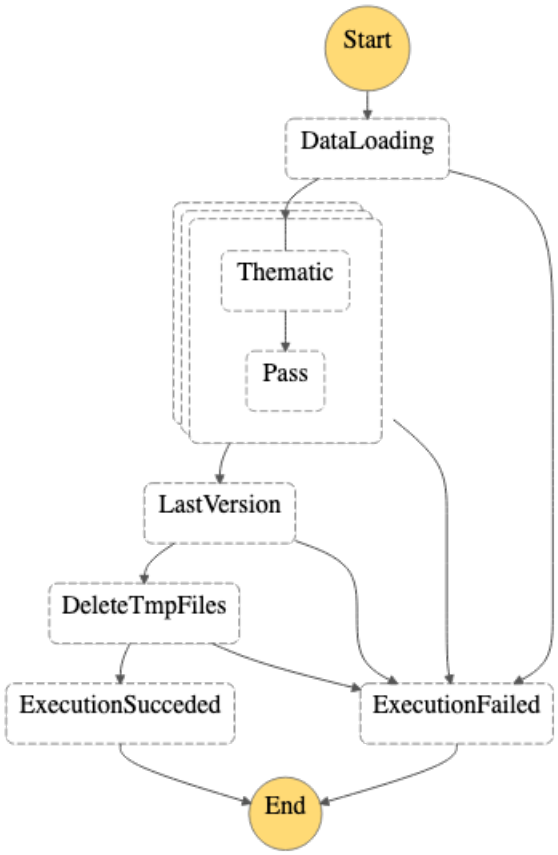
The keywords are written as new columns in the input CSV.



Thematic Classifier

This Step Function is similar to the Classifier one, because performs the same operations on a different set of categories using different models, but always trained by their data scientists.

The categories are: Environment, Politics, Corporate, News, Economy, Foreign news, Legal, Science and Medicine, Sports - Culture - Entertainment and Products.



Output

This Lambda Function is the last one to be executed inside of this Step Function. The purpose of this function is to collect the data produced by the previous steps, clean the output removing the rows without significant data, for instance data written nor in Italian or in English.

The data are written into the S3 Bucket RawData duplicating and dividing them by the tuple (Social,TopicID). In input the TopicID field is a list of values, but the need is to perform queries with Athena, and to achieve that this field must be a string or an integer.

For example the division of the data can be, starting from these two lines, Social = 'Web, TopicID = [1, 2, 3] and Social = Twitter, TopicID = [4,5]:

- Social = Web, TopicID = 1;

- Social = Web, TopicID = 2;
- Social = Web, TopicID = 3;
- Social = Twitter, TopicID = 4;
- Social = Twitter, TopicID = 5.

The data are saved on the S3 Bucket by the following path:
$RawData/OUTPUT/Social = SOCIAL\_NAME/TopicID = ID/Datetime = TIMESTAMP/UniqueID = GENERATED\_ID$

Chapter 5

Implemented Solution

This chapter shows the work performed in order to greatly improve performance and to simplify the tasks of the customer's data scientists.
This project is composed by two phases:

- Lift and Shift: This part of the project consists in transposing the algorithms described in the previous chapter from AWS StepFunction to Amazon SageMaker, creating a pipeline into SageMaker notebooks and executing the algorithms sequentially.

- Refactor: The last part of the project consists in refactor the flows enhancing part of the algorithms and re designing the pipelines, which correspond to creating new flows.

5.1 Architecture Overview

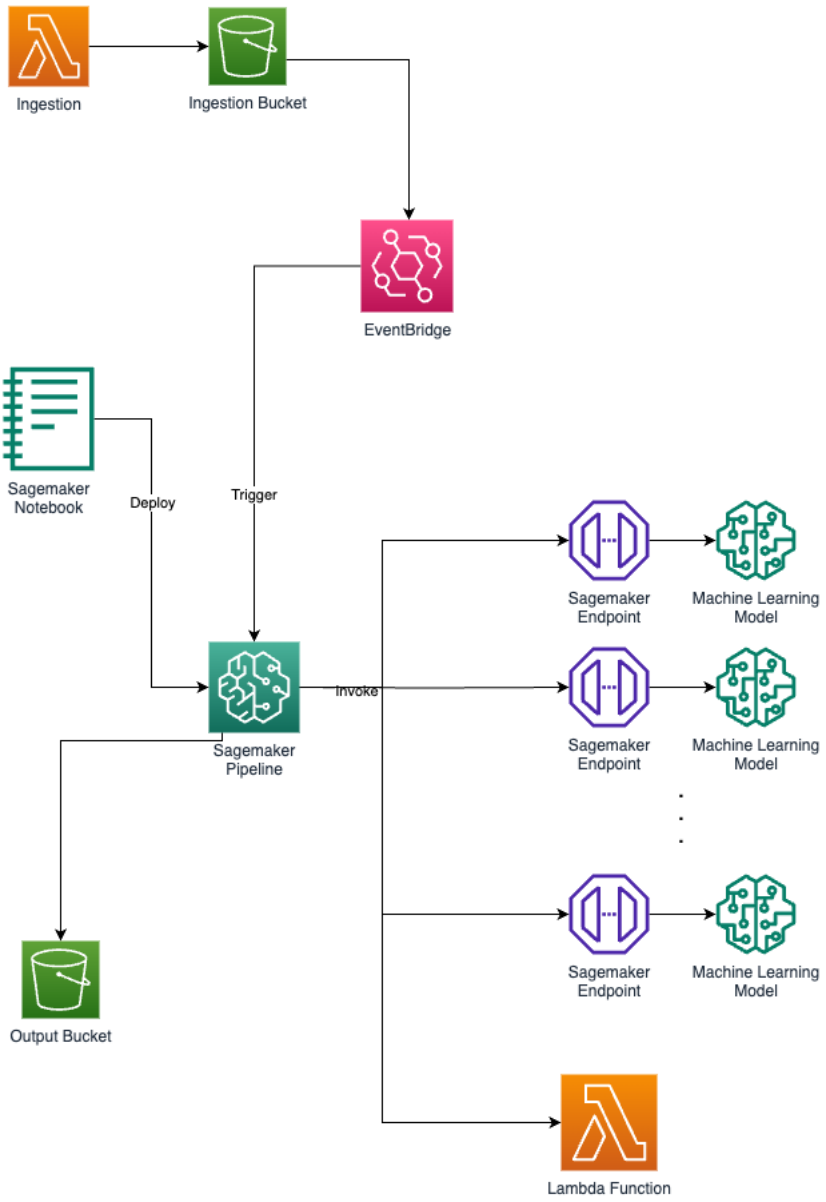
The main idea behind the architecture of this project is that Machine Learning models require more powerful tools to be executed efficiently in a production environment.
The entire Python code has been rewritten to improve it and adapt it to the new tools used. Only the models have been kept unchanged.
The ingestion of the data is performed in a similar way than the original solution, using a Lambda function executed periodically that invoke the Sprinklr API that

return the data collected in the last hour. This Lambda function is also in charge to standardize the data collected and to filter only the useful ones and write the entire dataset obtained into the Landing S3 Bucket, converting it from a Json list to CSV format.
Uploading a file into this S3 Bucket triggers an EventBridge rule, that invoke the pipeline that perform the inference on this dataset. EventBridge is a service that listen for events looking for a match with those defined in the various rules and then execute the corresponding one that can invoke several services like AWS Lambda or Sagemaker notebooks.
Sagemaker Notebook is a service that contain a collection of python notebook organized in folders that are highly integrated with AWS services and with a graphic dashboard that can be consulted in Sagemaker Studio.
The notebook is used to deploy a pipeline triggered by the EventBridge rule in which each step executes one python script.
Some of these python scripts invokes Sagemaker Endpoints, that expose a trained machine learning model providing enough computational power to perform the inference of the model. In a single Endpoint may reside one or more models and they perform a parallel scale in to guarantee better performances.
AWS Sagemaker Endpoints can be of two different types:

- Serverless: used to host smaller models, it provides computational power as required. When a new request comes a new instance is created with dynamic computational power, so it can be adjusted based on the computational complexity. The instance is destroyed 30 minutes after the execution completion, to avoid having to recreate it immediately after the completed inference. With this solution the customer pays only for the actual computational usage time.

- Real-time: This endpoint is used to host larger models that can not be hosted on the Serverless Endpoint. This endpoint require to be manually instantiated and allows to chose the size of the instance used to perform inference. Within the pipeline, this type of endpoint are created and destroyed in two different steps, respectively before and after the inference of the model. providing low latency and a lot of computational power, this solution is convenient when the model is large.

After the execution of the pipeline the results are stored into the Output S3 Bucket.



5.2 Lift and Shift

Data Ingestion

The first part of this project consists in migrate the pre-existing algorithms from

Lambda Functions synchronized with Stepfunctions into Python notebooks hosted on AWS Sagemaker.
The ingestion Stepfunction has been replaced with a Lambda Function that invoke the Sprinklr APIs to obtain the data, parse the input files to remove the not used rows to improve the performance of the model. The majority of the rows are comments written in different languages, so not pertinent to the use case.
Unlike the previous flow, this Lambda function does not split the input file to perform a batch execution, because it is not needed anymore.
When the dataset is ready, the notebook is executed by Event Bridge, a service that is used to trigger a response event when a certain condition is satisfied.

Pipeline Definition

The Python notebooks is firstly manually executed to create a pipeline used to execute all the algorithms sequentially.
The pipeline is defined by several steps, each of them allows to execute a Python script that can contains codes to create an endpoint, delete it or perform the model's inference. The pipeline can invoke also Lambda functions or other AWS services like Glue jobs and so on.

```
pipeline = Pipeline(
    name=pipeline_name,
    parameters=[
        train_instance_param,
        model_approval_status,
        dumpdate,
        input_data,
        s3_output_uri,
        time
    ],
    steps=[
        deploy_step,
        infer_step,
        clean_step,
    ])
```

This snippet is summarized for readability purposes, but the pipeline contains these steps (deploy_step, infer_step, clean_step) for each algorithm described in the previous chapter. The only algorithm that does not have these steps is TextRank, that use a smaller model and can be deployed into a serverless Endpoint, removing the needs to create and destroy a real time Endpoint.
The steps are defined in a similar way between them, because the fields only describe the computational power required, inputs, outputs and the path of the code that must be executed.

```
infer_model_processor = SKLearnProcessor(
    framework_version="0.23-1",
    role=sagemaker_iam_role,
    instance_type='ml.m5.xlarge',
    instance_count=1,
    base_job_name="infer-new-data",
    sagemaker_session=sagemaker_session,
)

infer_step = ProcessingStep(
    name="InferenceModel",
    processor=infer_model_processor,
    code=infer_script_uri,
    inputs=[
        sagemaker.processing.ProcessingInput(
            source = input_data,
            destination=f"{processing_dir}/processed_text",
            input_name='preprocess_tmp_data.csv',
            s3_data_type="S3Prefix",
            s3_input_mode="File",
            s3_data_distribution_type="FullyReplicated"
        )
    ],
    outputs=[
        sagemaker.processing.ProcessingOutput(
            output_name="sentiment_emotion_analysis",
            source=f"{processing_dir}/output/",
            destination = s3_output_uri,
```

```
29            )
30        ],
31        job_arguments=[
32            "--dumpdate",dumpdate  ,
33            "--time",time
34        ],
35        depends_on=[deploy_step.name]
36    )
```

This notebook is in charge to create also:

- the Serverless Endpoint that hosts the TextRank algorithm
- the Endpoint configurations for the Real-time Endpoints.
  An Endpoint configuration is a blueprint that defines how a model should be deployed on a computing resource. It includes details like the type of computing instance, the number of instances, and the model to be used.

Pipeline Execution

The pipeline execution is triggered by an Event Bridge rule that is evaluated when a input file is uploaded into an S3 bucket by the ingestion Lambda function.
First, almost all the algorithms need to create a Real-time Endpoint inside Sagemaker, and it is performed by the step "deploy_algorithm" that use the previous defined Endpoint configurations to create a Real-time Endpoint with a fully trained model.

```
1 create_endpoint_sentiment_response = sagemaker_client.create_endpoint
      (
2     EndpointName=enpoint_name,
3     EndpointConfigName=model_config['EndpointConfigName'])
```

The description of the code can be found here A.1.
The inference is performed by retrieving the input data from the dataset and simply sending it to the model residing in an Endpoint. This operation is similar whether it's a Serverless Endpoint, whether it's a Real-time Endpoint.

```
summarizer =  sagemaker_client.describe_endpoint(EndpointName=
    endpoint_name)

predictor = Predictor(
     endpoint_name=summarizer['EndpointName'],
     sagemaker_session=sagemaker_session,
     serializer = JSONSerializer(),
     deserializer = JSONDeserializer()
)
```

The description of the code can be found here A.2
The cleanup step is important, because prevent the Real-time Endpoint from billing for the whole time they are deployed. The customer does not need for an endpoint immediately available all the time and does not have problems related to the latency produced by the operation of creation and deletion.
The main goal of the customer is to deploy their models in a cheaper and efficient way.

```
sagemaker_client.delete_endpoint(EndpointName=enpoint_name)
```

The description of the code can be found here A.3

5.3 Refactor

This part of the project aims to improve the predictions by using native models provided by SageMaker or the integrations provided by HuggingFace to deploy models directly into endpoints.
The second goal is create new flows by reusing the algorithms created on SageMaker notebook, by using more sources as input data and by supporting more languages.

5.3.1 Models Improvement

It became apparent that using the models created by the customer's data scientists is not the best decision, because they are pre-trained in their data centers and

then uploaded into the Sagemaker Endpoints, adding complexity to the process of generating a new model for integration into a different workflow.
When a new workflow is defined a new Sagemaker notebook have to be created and deployed, because it contains the pipeline definition of the workflow itself. The steps of this pipeline can be associated with existing python code, if the step is associable with an already implemented algorithm or with a new algorithm that can refer to a new model.
Sagemaker Notebook simplify the creation and deploying of a new model, providing tools to train models using a scalable fleet of virtual machine. The training phase can be performed just before the definition of the inference pipeline, and is defined as a pipeline.

```
training_instance_type = ParameterString(
    name="TrainingInstanceType",
    default_value="ml.m5.large"
)

image_uri = sagemaker.image_uris.retrieve(
        framework="xgboost",
        region=region,
        version="1.0-1",
        py_version="py3",
        instance_type=training_instance_type,
)

xgb_train = Estimator(
    image_uri=image_uri,
    instance_type=training_instance_type,
    instance_count=processing_instance_count,
    output_path=model_path,
    base_job_name=training_job_name,
    sagemaker_session=session,
    role=role,
)

xgb_train.set_hyperparameters(
    objective="reg:linear",
```

```
    num_round=50,
    max_depth=5,
    eta=0.2,
    gamma=4,
    min_child_weight=6,
    subsample=0.7,
    silent=0,
)

step_train = TrainingStep(
    name=training_job_name,
    estimator=xgb_train,
    inputs={
        "train": TrainingInput(s3_data=step_process.properties.
    ProcessingOutputConfig.Outputs["train"].S3Output.S3Uri,
    content_type=content_type),
        "validation": TrainingInput(s3_data=step_process.properties.
    ProcessingOutputConfig.Outputs["validation"].S3Output.S3Uri,
    content_type=content_type),
    },
    depends_on=[step_process.name]
)
```

This snippet of code shows that Sagemaker pipelines are not constrained to execute a python script, but can also be used to perform jobs defined by libraries built to be highly integrated with the functionality of Sagemaker.
It is possible to train models stored on a S3 Bucket or downloaded through external libraries, for example HuggingFace.
In order to improve the performances of the models, it has been decided to replace the following models:

- the model created by the customer's data scientists in charge to perform emotion sentiment analysis on the data received by twitter was a custom trained version of DistilBERT.
  It is replaced by "cardiffnlp/twitter-roberta-base-emotion", a model provided by HuggingFace pre-trained to perform the same tasks on data collected by tweets. This model can perform better than DistilBERT, and it is not necessary to perform fine tuning, making easy to use a constantly updated

model.

- the model used to perform the classification task, is replaced with a fine-tuned version of "blazingtext".
  blazingtext is an algorithm provided by SageMaker through a docker image that can be used to instantiate a container that can perform the fine-tuning of the model.
  This model comes with a library that exposes functions to performs the fine-tuning of the model without writing a lot of code. Other than the simplicity of training, this model can scale easily to large datasets.

```
    container = sagemaker.amazon.amazon_estimator.get_image_uri(
region_name, "blazingtext", "latest")

    bt_model = sagemaker.estimator.Estimator(
        container,
        iam_role,
        instance_count=1,
        instance_type="ml.c4.4xlarge",
        volume_size=30,
        max_run=360000,
        input_mode="File",
        output_path=s3_output_location,
        hyperparameters={
            "mode": "supervised",
            "epochs": 30,
            "min_count": 3,
            "learning_rate": 0.02,
            "vector_dim": 100,
            "early_stopping": True,
            "patience": 6,
            "max_count": 10,
            "min_epochs": 20,
            "word_ngrams": 2,
            "buckets": 5000000
        },
    )

```

```
27      train_data = sagemaker.inputs.TrainingInput(
28          s3_train_data,
29          distribution="FullyReplicated",
30          content_type="text/plain",
31          s3_data_type="S3Prefix",
32      )
33      validation_data = sagemaker.inputs.TrainingInput(
34          s3_validation_data,
35          distribution="FullyReplicated",
36          content_type="text/plain",
37          s3_data_type="S3Prefix",
38      )
39      data_channels = {"train": train_data, "validation":
        validation_data}
40
41      bt_model.fit(inputs=data_channels, logs=True)
42
```

- the translator task was performed by a version of Opus_MT, but require to build one model to each language to be translated.
  It was replaced with Facebook M2M100, a model that performs a little poorer than Opus_MT choosing a single language, but that can handle hundreds of them without performing another deployment.

5.3.2 Flows Definition

After migrating the algorithms from Step Functions to SageMaker and upgrading certain models with more efficient ones, the last phase of this project involves additional customization and deployment of the developed algorithms to create various workflows.
This is an important stage of the project, because with the algorithms now residing on SageMaker it is possible to use its features to simplify our operations and unlock new possibilities.
Changing and using these algorithms will let us make different workflows for specific jobs and goals. This means we can adjust and grow how we do things based on what we need and how our business is changing.

In addition to the initial workflow migrated from the Stepfunctions, some new workflows have been implemented, and they will be expanded in the future.
New workflows are required especially when there is the need to process inputs written in different languages or when the data are collected by a new data source.

Swahili

This new workflow is required to process inputs written in Swahili, English or French language, that are the main languages spoken in Rwanda.
This workflow process inputs processed by the Sprinklr Ingestion Stepfunction that create a CSV filled with only messages written in these language and save it in a directory which resides into the Ingestion S3 Bucket, called "Sprinklr-rwanda".
The ingestion is performed by the same Lambda function, that separate the data in input by the language, saving the Swahili, English and French records in a parquet file written into a separate directory inside the Ingestion bucket. The uploading of this file will trigger a EventBridge rule that invoke the correct pipeline.
The pipeline associated with this workflow is composed by the following steps:

- The first step of this pipeline is in charge to perform the translation of the data in input written in Swahili or French in English.
  To perform the translation is used the basic version of the Google translation model (Cloud Translation), the which APIs are exposed by the library api-client.discovery.build.
  The exposed model is the basic version of NMT, a fully pretrained model capable to translate sentences in over 100 languages. That was the easiest way to use a model trained to work with Swahili language.

```
1    from apiclient.discovery import build
2
3    ga_secrets = get_secret()
4    credentials = ServiceAccountCredentials.
    from_json_keyfile_dict(ga_secrets, SCOPES)
5
6    http = httplib2.Http(timeout=900)
7    http = credentials.authorize(http)
```

```
    translator = build('translate', 'v2', http=http)

    response = translator.translations().list(
        target='en',
        q=[text]
    ).execute()

    return response['translations'][0]['translatedText']

```

The entire code can be found here A.4.

- The next step of the pipeline is the summarization of the translated text, that consists in summarize a long text to reduce its size.
  To summarize the input text, is used a model fine-tuned in another SageMaker notebook by a customer's data scientist and deployed in a Real-time endpoint, then reused by me. The base model is sshleifer/distilbart-cnn-12-6.
  The summarization is performed only for sentences longer than 250 tokens, and the results maintained must be lower than 250 tokens and longer than 50. Finally the sentences are joined to compose the summarized message.

```
    summarizer = sagemaker_client.describe_endpoint(
        EndpointName="sshleifer/distilbart-cnn-12-6"
    )

    predictor = Predictor(
        endpoint_name=summarizer['EndpointName'],
        sagemaker_session=sagemaker_session,
        serializer = JSONSerializer(),
        deserializer = JSONDeserializer()
    )

    [...]

    for text in text_compose:
        payload = {
            "inputs": text,
```

```
            "parameters": {
                "max_length" : max_chunk_summary_len,
            }
        }
        result = predictor.predict(payload)
        chunk_summary = clean_result(result)
        prediction.append(chunk_summary)

```

The entire code can be found here A.5.

Google Maps Classification

This workflow introduce a new input source, the reviews written on Google Maps for the customer's selling points.
To perform the ingestion of these data, a new Lambda function has been created. This function has the purpose of contacting a Google Maps API, filter for English messages and write them in a Landing S3 Bucket.
This pipeline must be executed for data coming from two different types of location found on Google Maps, but this value can not be explained for NDA reasons.
The ingestion Lambda function used to obtain the data through the API is the same for each workflow, the only difference is a parameter given in input. The notebook that create the pipeline are almost identical between them, the only difference is the models given in input to the classification task. This two pipelines have to classify for different labels, assigned by the different models. The pipeline executes different classification tasks using several models, but the steps defined are the models deploy, the models inference and the Endpoint cleanup.
Following the snippet of code that

```
deploy_model_processor = SKLearnProcessor(
    framework_version="0.23-1",
    role=sagemaker_role,
    instance_type='ml.m5.xlarge',
    instance_count=1,
    base_job_name="deploy-model",
    sagemaker_session=sagemaker_session,
```

```
)

deploy_step = ProcessingStep(
    name="DeployModel",
    processor=deploy_model_processor,
    code=deploy_model_script_uri,
    job_arguments=[
        "--time",time
        "--place_name", place_name
        "--endpoint_names", ["gmaps-prezzi", "gmaps-accessibilita", "gmaps-gentilezza", "gmaps-tempestivita", "gmaps-competenza"]
    ]
)
```

The inference code can be found here A.6.

Google Maps Emotions

This workflow use the previous defined ingestion Lambda function to trigger both the previous and this pipeline.
This workflow needs to perform only a Emotion and Sentiment analysis, to comprehend if the sentiment of the messages are positive or negative, in particular the label used are "['sent_negative', 'sent_neutral', 'sent_positive']" for the sentiment analysis and "['emo_joy', 'emo_optimism', 'emo_anger','emo_sadness']" for emotion analysis.
The pipeline is structured to create the Real-time Endpoint, perform the inference and then perform the cleanup, removing it.
The models are trained when the Notebook is executed, and the results is the creation of an Endpoint Configuration 5.2.
The model used are HuggingFace Transformers defined through the HuggingFace library written for SageMaker.

```
huggingface_model_EMO = HuggingFaceModel(
    transformers_version='4.17.0',
    pytorch_version='1.10.2',
    py_version='py38',
```

```
    env=emotion_param,
    role=role,
)

huggingface_model_SENT = HuggingFaceModel(
    transformers_version='4.17.0',
    pytorch_version='1.10.2',
    py_version='py38',
    env=sentiment_param,
    role=role,
)
```

The inference is described here A.7.

Chapter 6

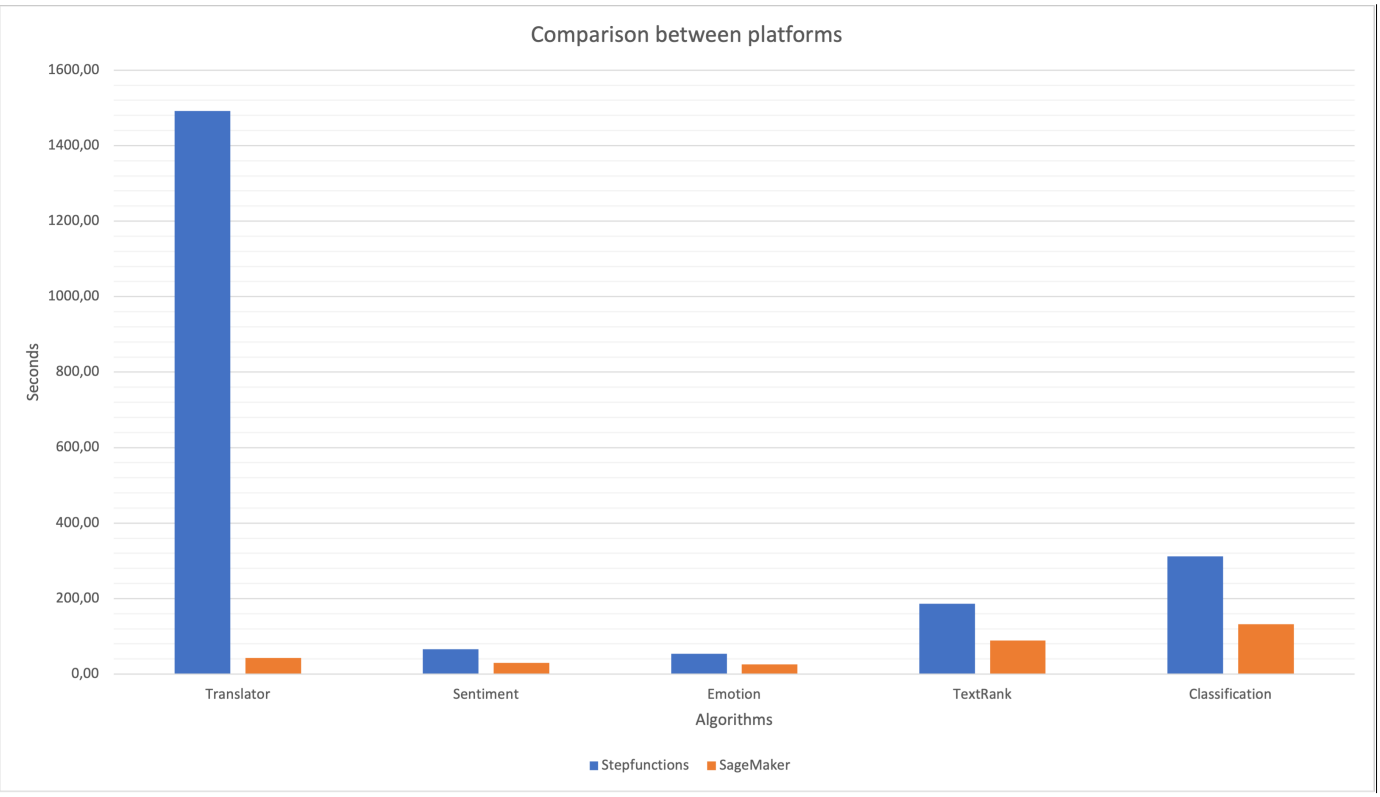
Results

In this section we analyze the results of the implemented solution compared to the previous one.
Through experimentation and analysis, we observed significant improvements in inference speed and resource utilization, showing the efficacy of our proposed approaches. Additionally, our work contributed to set best practices for the customers for taking advantages of AWS services in the deployment of machine learning solutions, with implications for a wide range of applications.

Model	Stepfunction	SageMaker	Reduction
Translator	24:52 min	0:43 min	97,12%
Sentiment	1:06 min	0:30 min	54,55%
Emotion	0:54 min	0:26 min	51,85%
TextRank	3:06 min	1:29 min	52,15%
Classification	5:12 min	2:12 min	57,69%

This huge time execution reduction is caused by the more powerful computing units used and by the parallelization of the execution, that is harder to achieve and manage with a Lambda function that execute the inference for one or two data at time.
This following graph shows the results described in the previous table, highlighting

the gap between the execution time:



Other than the pure time execution reduction, the replacement of the older models created by the customer's data scientists with the ones provided by HuggingFace improved the performances of the algorithms and allows to create new workflows by simply retrain the models with new data and hyperparameters through the pipelines created for this purpose.
Finally, the best result that we obtained is the drastic reduction of the Time to Market, namely the reduction of the amount of time it takes for the algorithms and the models to be developed and deployed in production. This results is achieved through the automations created to automatize the deployment of the infrastructure when an algorithm is executed, deploying the physical machines on Endpoints when needed and exposing them.
This solution provides the possibility of integrating in production new workflows in minutes, without using hours to write the Stepfunction and the related Lambda

functions, allowing the data scientists to focus only on the new algorithms and models.

Chapter 7

Conclusion and Future Work

In this thesis, we analyzed the advantages of using tools developed to execute Machine Learning algorithms in a more efficient way.
We performed two tasks for engineering the pre-existing algorithms.
First, a Lift and Shift approach allowed us to moving the pre-existing algorithms from a inefficient architecture that is difficult to maintain and extend to a modular platform which allows singular steps to be built and then inserted with a minimal effort into a pipeline, allowing it to create several workflows in a simple way.
The next step aims to use more efficient models, some times already provided by SageMaker and sometimes imported externally. This is possible because SageMaker, as opposed to Stepfunctions, is a strong environment which allows to expose models into an Endpoint instead of a shared file system and does not have a timeout during the execution of the algorithm, avoiding having to split and merge continuously the input dataset.
Finally, the last step consists of using these improved algorithms to create new workflows by managing more languages, more input sources and more functionalities, for example the summarization performed for the input coming from Sprinklr written in Swahili language.
The results confirm that the solution allows to perform the training and inference tasks in a significant faster way, paying more than the previous solution but much less than the solutions with equivalent performance, for example using EC2 instances to host the models.

The future works will focus on these topics:

- Improving MLOps: at this moment the CI/CD pattern is not fully integrated, because we have pipelines that are automatically invoked when a new file is uploaded into a S3 Bucket, but there is missing a process to automatically execute the notebooks that train the models and deploy the pipelines when source code is pushed in a repository.
- Generative AI: integrating a generative AI system within the existing algorithms to perform several tasks like: generate images given a input text, create caption to input images, image labeling, image recognition to identify objects in images.
- implementing a data mart, that consists in a smaller subset of a data warehouse in which the data are specific for the use cases.
  Data marts can improve performance and efficiency, as the data scientists only deal with the data directly relevant to their operations. They also allow for more specialized analysis and reporting.

Appendix A

Code Snippets

A.1 Endpoint Deployment

```
import boto3
import time
import subprocess
import sys
import argparse
from botocore.config import Config
subprocess.check_call([sys.executable, "-m", "pip", "install", "
    sagemaker"])

import sagemaker
from sagemaker.predictor import Predictor
from sagemaker.serializers import JSONSerializer
from sagemaker.deserializers import JSONDeserializer


boto_session = boto3.Session(region_name='eu-west-1')
sagemaker_client = boto_session.client("sagemaker", config=Config(
    connect_timeout=5, read_timeout=60, retries={'max_attempts': 20}))

parser = argparse.ArgumentParser()
parser.add_argument('--time', type=str)
```

```
20 args = parser.parse_args()
21
22 time_d = args.time
23
24 model_name =  f'emotion-analisys-huggingface-{time_d}'
25
26
27 model_config = sagemaker_client.describe_endpoint_config(
       EndpointConfigName='emotion-analisys-huggingface')
28
29 existing_endpoint=[]
30
31 existing_endpoint += sagemaker_client.list_endpoints(NameContains=
       model_name, MaxResults=100)["Endpoints"]
32
33 if len(existing_endpoint) ==0:
34     create_endpoint_sentiment_response = sagemaker_client.
       create_endpoint(
35         EndpointName=model_name,
36         EndpointConfigName=model_config['EndpointConfigName'])
37
38
39 endpoint_info = sagemaker_client.describe_endpoint(EndpointName=
       model_name)
40
41 endpoint_status = endpoint_info['EndpointStatus']
42
43 while endpoint_status != 'InService':
44
45     endpoint_info = sagemaker_client.describe_endpoint(EndpointName=
       model_name)
46     endpoint_status = endpoint_info['EndpointStatus']
47
48     if sent_endpoint_status != 'InService':
49         time.sleep(60)
```

A.2 Endpoint Inference

```
import subprocess
import sys, os
import boto3
import datetime as dt
import pandas as pd
import numpy as np
import pathlib
import argparse
import logging

logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)

subprocess.check_call([sys.executable, "-m", "pip", "install", "
    sagemaker"])

import sagemaker
from sagemaker.predictor import Predictor
from sagemaker.serializers import JSONSerializer
from sagemaker.deserializers import JSONDeserializer
from sagemaker.feature_store.feature_group import FeatureGroup


parser = argparse.ArgumentParser()
parser.add_argument('--dumpdate', type=str)
parser.add_argument('--time', type=str)
args = parser.parse_args()

dumpdate = args.dumpdate
time = args.time

bucket = 'summ-ai-prod-curated'
prefix = 'sprinklr'
region = 'eu-west-1'

boto_session = boto3.Session(region_name=region)
sagemaker_client = boto_session.client("sagemaker")
```

```
37 sagemaker_session = sagemaker.session.Session(boto_session=
       boto_session, sagemaker_client=sagemaker_client)
38
39
40 #endpoint retrieval
41 model = sagemaker_client.describe_endpoint(EndpointName=f'emotion-
       analisys-huggingface-{time}')
42
43 predictor = Predictor(
44      endpoint_name=model['EndpointName'],
45      sagemaker_session=sagemaker_session,
46      serializer = JSONSerializer(),
47      deserializer = JSONDeserializer())
48
49 input_data_path = os.path.join("/opt/ml/processing/processed_text", "
       preprocess_tmp_data.csv")
50
51 sprinklr_data = pd.read_csv(input_data_path, dtype = {'dumpdate_p':
       str, 'testo_en':str})
52 sprinklr_data = sprinklr_data.replace(np.nan,'')
53
54 output = pd.DataFrame(columns=['sent_negative', 'sent_neutral', '
       sent_positive'])
55
56 for row in sprinklr_data['testo_en']:
57         result = predictor.predict({
58         'inputs': [row],
59         'parameters': {
60             'return_all_scores':True,
61             'max_length': 512,
62             'truncation': True}
63         })
64         output.loc[len(output)] = list(pd.Series(pd.json_normalize(
       result[0]).set_index('label').T.values[0]))\
65
66
67 sprinklr_data[['sent_negative', 'sent_neutral', 'sent_positive']] =
       np.round(output, 2)
68
69 output_path = pathlib.Path(f"/opt/ml/processing/output/")
```

```
sprinklr_data.to_csv(output_path / 'sentiment_emotion_analysis.csv',
    index=False)
```

A.3 Endpoint Cleanup

```
import boto3
import subprocess
import sys
import argparse
from botocore.config import Config

subprocess.check_call([sys.executable, "-m", "pip", "install", "
    sagemaker"])
import sagemaker

parser = argparse.ArgumentParser()
parser.add_argument('--time', type=str)
args = parser.parse_args()

time = args.time

boto_session = boto3.Session(region_name='eu-west-1')
sagemaker_client = boto_session.client("sagemaker", config=Config(
    connect_timeout=5, read_timeout=60, retries={'max_attempts': 20}))

sagemaker_client.delete_endpoint(EndpointName=f'sentiment-analisys-
    huggingface-{time}')
```

A.4 Translation

```
import subprocess
import sys, os

```

```
import sagemaker
from botocore.exceptions import ClientError
from apiclient.discovery import build
from oauth2client.service_account import ServiceAccountCredentials
import httplib2
import pandas as pd
import argparse
import boto3
import io, json
import pathlib

import time_uuid
import uuid
from datetime import datetime, timedelta

# RECUPERO DEL SEGRETO

secret_name = [SECRET]
SCOPES = ['https://www.googleapis.com/auth/cloud-translation']

parser = argparse.ArgumentParser()
parser.add_argument('--filepath', type=str)
parser.add_argument('--bucket', type=str)
parser.add_argument('--filename', type=str)
args = parser.parse_args()
filepath = args.filepath
bucket = args.bucket
filename = args.filename

boto_session = boto3.Session(region_name = 'eu-west-1')
s3_client= boto_session.client("s3")
sagemaker_client = boto_session.client("sagemaker")
sagemaker_session = sagemaker.session.Session(boto_session=
    boto_session, sagemaker_client=sagemaker_client)



# Read file from S3
def pd_read_s3(key, bucket, s3_client):
    obj = s3_client.get_object(Bucket=bucket, Key=key)
```

```
43      return pd.read_parquet(io.BytesIO(obj['Body'].read()))
44
45
46  def get_secret():
47
48      client = boto_session.client(service_name='secretsmanager')
49      try:
50          get_secret_value_response = client.get_secret_value(SecretId=
        secret_name)
51      except ClientError as e:
52          raise e
53      else:
54          # Decrypts secret using the associated KMS CMK.
55          if 'SecretString' in get_secret_value_response:
56              secret = get_secret_value_response['SecretString']
57          else:
58              secret = base64.b64decode(get_secret_value_response['
        SecretBinary'])
59
60          return json.loads(secret)
61
62
63  # INIZIALIZZAZIONE DELLE CREDENZIALI
64  def initialize_translator():
65
66      ga_secrets = get_secret()
67      credentials = ServiceAccountCredentials.from_json_keyfile_dict(
        ga_secrets, SCOPES)
68      # credentials = ServiceAccountCredentials.from_json_keyfile_name
        ("./IngestionAlgorithm/client_secret.json", SCOPES)
69      http = httplib2.Http(timeout=900)
70      http = credentials.authorize(http)
71      translator = build('translate', 'v2', http=http)
72
73      return translator
74
75
76  def translate_text(translator,text):
77
78      response = translator.translations().list(
```

```
79          target='en',
80          q=[text]
81      ).execute()
82
83      return response['translations'][0]['translatedText']
84
85  s3 = boto3.resource('s3')
86
87  s3_keys = [item.key for item in s3.Bucket(bucket).objects.filter(
        Prefix=filepath) if item.key.endswith('.parquet')]
88
89
90  ts = time_uuid.TimeUUID(bytes=uuid.UUID(filename).bytes).
        get_timestamp()
91  start_date = datetime.fromtimestamp(ts).replace(microsecond=0, second
        =0, minute=0)
92  end_date = start_date +timedelta(hours=1)
93
94  dfs=[]
95  if not s3_keys:
96      print('No parquet found in', bucket, filepath)
97  else:
98      for key in s3_keys:
99          name = key.split('/')[-1].split('.')[0]
100         print(name)
101         time = time_uuid.TimeUUID(bytes=uuid.UUID(name).bytes).
        get_timestamp()
102         print('time = ', datetime.fromtimestamp(time))
103         print('start_date: ', start_date)
104         print('end_date: ', end_date)
105         if start_date < datetime.fromtimestamp(time) and datetime.
        fromtimestamp(time) < end_date:
106             dfs.append(pd_read_s3_parquet(key, bucket=bucket,
        s3_client=s3_client))
107
108 result_text = []
109 result_title = []
110
111 if len(dfs) >= 1:
112     data = pd.concat(dfs, ignore_index=True)
```

```
    print('data len before drop_duplicates: ', len(data))

    data = data.drop_duplicates(subset=['Title'])

    print('data len after drop_duplicates:', len(data))
    tranlsator = initialize_translator()

    for idx, row in data.iterrows():
        if row.LanguageCode in ['fr','sw']:
            result_text.append(translate_text(tranlsator, row.Message))
            result_title.append(translate_text(tranlsator, row.Title))
        elif row.LanguageCode in ['en']:
            result_text.append(row.Message)
            result_title.append(row.Title)
        else:
            result_text.append('')
            result_title.append('')

    print('result: ', len(result_text))
    print('filename: ', len(filename))

else:
    print('no data found in last hour')
    data = pd.DataFrame()

data_report_dict = {
    "df_empty": data.empty
}

output_path = f"/opt/ml/processing/data_report_dict/data_report_dict.json"
with open(output_path, "w") as f:
    f.write(json.dumps(data_report_dict))

data['translated_text'] = result_text
data['translated_title'] = result_title
output_path = pathlib.Path('/opt/ml/processing/translated_text')
data.to_parquet(output_path / f'{filename}.parquet', index=False)
```

A.5 Summarization

```
import pandas as pd

import sys
import subprocess
import uuid
import pathlib
import argparse

from transformers import AutoTokenizer

dir_tokenizer = "./tokenizer"
tokenizer = AutoTokenizer.from_pretrained(
    pretrained_model_name_or_path=dir_tokenizer)

import boto3
import sagemaker
from sagemaker.predictor import Predictor
from sagemaker.serializers import JSONSerializer
from sagemaker.deserializers import JSONDeserializer

parser = argparse.ArgumentParser()
parser.add_argument('--filename', type=str)
args = parser.parse_args()
filename = args.filename


summarization_edp_name = 'prod-summary-edp-'+filename

region_name='eu-west-1'
boto_session = boto3.Session(region_name=region_name)
sagemaker_client = boto_session.client("sagemaker")
sagemaker_session = sagemaker.session.Session(boto_session=
    boto_session, sagemaker_client=sagemaker_client)
```

```
32
33 summarizer = sagemaker_client.describe_endpoint(EndpointName=
       summarization_edp_name)
34
35 predictor = Predictor(
36     endpoint_name=summarizer['EndpointName'],
37     sagemaker_session=sagemaker_session,
38     serializer = JSONSerializer(),
39     deserializer = JSONDeserializer()
40 )
41
42 def compose_gt_250(text):
43     data=''
44     result = []
45     sentence_len = 0
46
47     periods = text.split('.')
48     max_chunk_len = 512
49     for val in periods:
50         string_token_len = len(tokenizer(val, max_length=512,
       truncation=True)['input_ids'])
51         if sentence_len + string_token_len < max_chunk_len:
52             if len(val)>0:
53                 sentence_len += string_token_len
54                 data += str(val) +'.'
55         else:
56             result.append(data)
57             data=str(val) +'.'
58             sentence_len = string_token_len
59
60     result.append(data)
61
62     return result
63
64 def clean_result(res):
65     chunk_summary = res[0]["summary_text"].replace(" . ",". ")
66     if chunk_summary.split(".")[:-1] != []:
67         chunk_summary = ('.'.join(chunk_summary.split(".")[:-1]) + '.
       ').replace(" .", ".")
68     else:
```

```
69          chunk_summary += ". "
70      return chunk_summary
71
72  def summary(msg):
73
74      max_token_length = 250
75      min_token_length = 50
76      summaries = []
77      prediction = []
78
79      sentence_len = len(tokenizer(msg)['input_ids'])
80
81      if sentence_len >= max_token_length:
82
83          max_summary_len = 300
84          text_compose = compose_gt_250(msg)
85          max_chunk_summary_len = int(max_summary_len / len(
    text_compose))
86
87          for text in text_compose:
88              payload = {
89                  "inputs": text,
90                  "parameters": {
91                      "max_length" : max_chunk_summary_len,
92                  }
93              }
94              result = predictor.predict(payload)
95              chunk_summary = clean_result(result)
96              prediction.append(chunk_summary)
97
98      if sentence_len >= min_token_length and sentence_len <=
    max_token_length:
99          payload = {
100             "inputs": msg,
101             "parameters": {
102                 "max_length" : sentence_len,
103             }
104         }
105         res = predictor.predict(payload)
106         chunk_summary = clean_result(res)
```

```
            prediction.append(chunk_summary)

        elif sentence_len < min_token_length:
            prediction.append(msg)

        if len(prediction) > 1:
            prediction_str = ''.join(prediction)
            summaries.append(prediction_str)
        else:
            summaries.append(prediction[0])
        return summaries

df_rwanda = pd.read_parquet(f"/opt/ml/processing/input/{filename}.
    parquet")

result = []
for idx, row in df_rwanda.iterrows():
    msg = row['translated_text'].replace('\n',' ').replace('    ',' ')
    .replace('    ',' ').replace('\"', ' ').replace("[", " ").replace("
    ]", " ")

    if len(msg) > 0:
        result.append(str(summary(msg)[0]))
    else:
        result.append(msg)

df_rwanda['summary'] = result
df_rwanda.to_parquet(f"/opt/ml/processing/processed/summary/{filename
    }.parquet", index=False)
```

A.6 Google Maps Classification

```
import subprocess
import sys, os
import boto3
import datetime as dt
```

```
import pandas as pd
import pathlib
import argparse

subprocess.check_call([sys.executable, "-m", "pip", "install", "
    sagemaker"])
subprocess.check_call([sys.executable, "-m", "pip", "install", "nltk"
    ])

import sagemaker
from sagemaker.predictor import Predictor
from sagemaker.serializers import JSONSerializer
from sagemaker.deserializers import JSONDeserializer
from sagemaker.feature_store.feature_group import FeatureGroup

import nltk
nltk.download('punkt')

def list_of_strings(arg):
    return arg.split(',')

parser = argparse.ArgumentParser()
parser.add_argument('--time', type=str)
parser.add_argument('--place_name', type=str)
parser.add_argument('--endpoint_names', type=list_of_strings)

args = parser.parse_args()

time = args.time
endpoint_names = args.endpoint_names
place_name = args.place_name

boto_session = boto3.Session(region_name = 'eu-west-1')
s3_client= boto_session.client("s3")
sagemaker_client = boto_session.client("sagemaker")
sagemaker_session = sagemaker.session.Session(boto_session=
    boto_session, sagemaker_client=sagemaker_client)

output_path = pathlib.Path('/opt/ml/processing/{place_name}/
    classified_data')
```

```
input_data_path = os.path.join("/opt/ml/processing/{place_name}/processed_text", "processed_text.csv")

gmaps_data = pd.read_csv(input_data_path, dtype = {'dumpdate_p': str, 'testo_en':str})
gmaps_data = gmaps_data.drop(gmaps_data[(gmaps_data['placetype_p'] != place_name)].index)

null_values=[]
for idx,row in gmaps_data.iterrows():
    if pd.isna(row.processed_text) or row.processed_text == '':
        null_values.append(idx)
        gmaps_data.loc[idx, 'processed_text'] = ''

sentences = list(gmaps_data.processed_text)
tokenized_sentences = [" ".join(nltk.word_tokenize(sent)) for sent in sentences]
payload = {"instances": tokenized_sentences}

endpoints = []
results = {}
#endpoint retrieval
for name in endpoint_names:
    endpoint = sagemaker_client.describe_endpoint(EndpointName=f'{name}-{time}')
    predictor = Predictor(
        endpoint_name=endpoint['EndpointName'],
        sagemaker_session=sagemaker_session,
        serializer = JSONSerializer(),
        deserializer = JSONDeserializer())

    result = predictor.predict(payload)

    results[endpoint['EndpointName']] = []

    for i in range(0,len(sentences)):
        if i not in null_values:
            results[endpoint['EndpointName']].append(result[i]['label'][0].split('__')[-1])
        else:
```

```
            results[endpoint['EndpointName']].append('not')

    gmaps_data[endpoint['EndpointName']] = results[endpoint['
    EndpointName']]

gmaps_data.to_csv(output_path / 'output.csv', index=False, sep=';')
```

A.7 Google Maps Sentiment Analysis

```
import subprocess
import sys, os
import boto3
import datetime as dt
import pandas as pd
import numpy as np
import pathlib
import argparse
import logging

logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)

subprocess.check_call([sys.executable, "-m", "pip", "install", "
    sagemaker"])

import sagemaker
from sagemaker.predictor import Predictor
from sagemaker.serializers import JSONSerializer
from sagemaker.deserializers import JSONDeserializer
from sagemaker.feature_store.feature_group import FeatureGroup


parser = argparse.ArgumentParser()
parser.add_argument('--dumpdate', type=str)
parser.add_argument('--time', type=str)
args = parser.parse_args()
```

```
27
28 dumpdate = args.dumpdate
29 time = args.time
30
31 bucket = 'prod-curated'
32 prefix = 'googlemaps'
33 region = 'eu-west-1'
34
35 boto_session = boto3.Session(region_name=region)
36 sagemaker_client = boto_session.client("sagemaker")
37 sagemaker_session = sagemaker.session.Session(boto_session=
       boto_session, sagemaker_client=sagemaker_client)
38
39
40 #endpoint retrieval
41 sentiment = sagemaker_client.describe_endpoint(EndpointName=f'
       sentiment-analisys-huggingface-{time}')
42 emotion = sagemaker_client.describe_endpoint(EndpointName=f'emotion-
       analisys-huggingface-{time}')
43
44 sentiment_predictor = Predictor(
45     endpoint_name=sentiment['EndpointName'],
46     sagemaker_session=sagemaker_session,
47     serializer = JSONSerializer(),
48     deserializer = JSONDeserializer())
49
50 emotion_predictor = Predictor(
51     endpoint_name=emotion['EndpointName'],
52     sagemaker_session=sagemaker_session,
53     serializer = JSONSerializer(),
54     deserializer = JSONDeserializer())
55
56 input_data_path = os.path.join("/opt/ml/processing/processed_text", "
       preprocess_tmp_data.csv")
57
58 gmaps_data = pd.read_csv(input_data_path, dtype = {'dumpdate_p': str,
       'testo_en':str})
59 gmaps_data = gmaps_data.replace(np.nan,'')
60
```

```
sent_output = pd.DataFrame(columns=['sent_negative', 'sent_neutral',
    'sent_positive'])
emo_output = pd.DataFrame(columns=['emo_joy', 'emo_optimism', '
    emo_anger', 'emo_sadness'])

for row in gmaps_data['testo_en']:
        sent_result = sentiment_predictor.predict({
        'inputs': [row],
        'parameters': {
            'return_all_scores':True,
            'max_length': 512,
            'truncation': True}
        })
        emo_result = emotion_predictor.predict({
        'inputs': [row],
        'parameters': {
            'return_all_scores':True,
            'max_length': 512,
            'truncation': True}
        })
        sent_output.loc[len(sent_output)] = list(pd.Series(pd.
    json_normalize(sent_result[0]).set_index('label').T.values[0]))
        emo_output.loc[len(emo_output)] = list(pd.Series(pd.
    json_normalize(emo_result[0]).set_index('label').T.values[0]))


gmaps_data[['sent_negative', 'sent_neutral', 'sent_positive']] = np.
    round(sent_output, 2)
gmaps_data[['emo_joy', 'emo_optimism', 'emo_anger','emo_sadness']] =
    np.round(emo_output, 2)

output_path = pathlib.Path(f"/opt/ml/processing/output/")
gmaps_data.to_csv(output_path / 'sentiment_emotion_analysis.csv',
    index=False)
```

Bibliography

[1] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. «DevOps». In: *IEEE Software* 33.3 (2016), pp. 94–100. DOI: 10.1109/MS.2016.68 (cit. on p. 1).

[2] Dominik Kreuzberger, Niklas Kühl, and Sebastian Hirschl. «Machine Learning Operations (MLOps): Overview, Definition, and Architecture». In: *arXiv preprint arXiv:2205.02302* (2022) (cit. on p. 2).

[3] Mary Beth Kery, Marissa Radensky, Mahima Arya, Bonnie E. John, and Brad A. Myers. «The Story in the Notebook: Exploratory Data Science Using a Literate Programming Tool». In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI '18. Montreal QC, Canada: Association for Computing Machinery, 2018, pp. 1–11. ISBN: 9781450356206. DOI: 10.1145/3173574.3173748. URL: https://doi.org/10.1145/3173574.3173748 (cit. on p. 2).

[4] *What is CI/CD?* Red Hat. 2022. URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd (cit. on p. 16).

[5] Ravindra Parmar. *Common Loss functions in machine learning*. Towards Data Science. Sept. 2018. URL: https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23 (cit. on p. 20).

[6] Diane Castillo. *Machine Learning Model Inference vs Machine Learning Training*. Seldon. Feb. 2022. URL: https://www.seldon.io/machine-learning-model-inference-vs-machine-learning-training (cit. on p. 22).

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL] (cit. on p. 24).

[8] *MarianMT*. Hugging Face. URL: https://huggingface.co/docs/transformers/v4.31.0/en/model_doc/marian#transformers.MarianModel (cit. on p. 43).

[9] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. 2020. arXiv: 1910.01108 [cs.CL] (cit. on p. 46).

[10] *NLTK*. URL: https://www.nltk.org/ (cit. on p. 49).

[11] Bob Pang and Lillian Lee. «Opinion Mining and Sentiment Analysis». In: *Foundations and Trends in Information Retrieval* 2.1-2 (2008), pp. 1–135. DOI: 10.1561/1500000001. URL: http://www.cs.cornell.edu/home/llee/opinion-mining-sentiment-analysis-survey.html (cit. on p. 51).

[12] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL] (cit. on p. 51).

[13] Rada Mihalcea and Paul Tarau. «TextRank: Bringing Order into Text». In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 404–411. URL: https://aclanthology.org/W04-3252 (cit. on p. 52).