



Professores: Bianca Pedrosa
Samuel Martins

Projeto Interdisciplinar

1. Especificação

O objetivo do projeto interdisciplinar é a aplicação das técnicas de Ciência de Dados aprendidas nas diferentes disciplinas do semestre, a partir de um problema apoiado por um conjunto de dados (dataset). O projeto requer o desenvolvimento de uma solução de aprendizado de máquina cujo treinamento e processamento deverão ser realizados em ambiente de nuvem (AWS ou Google), com processamento distribuído (cluster de computadores), explorando o uso de paralelismo (spark/mapreduce).

Aluno: Omar Hajime Fidelis

Data: 05/12/2021

2. Regras Gerais

- O tema do projeto é de livre escolha e deve ser levantado pelos os alunos integrantes do grupo.
- O projeto deverá ser feito em grupos de até 3 (três) pessoas.
- Ao final do semestre, os alunos devem elaborar uma apresentação do projeto, mostrando os resultados das respectivas atividades requeridas por cada uma das disciplinas cursadas.
- Todos os grupos devem apresentar o trabalho. A entrega pelo Moodle é só uma formalidade e não garante avaliação. Os alunos que não participarem da apresentação, a princípio, terão nota 0.
- Cada professor exigirá atividades relativas a conteúdos específicos de sua disciplina.
- As notas das disciplinas serão obtidas separadamente.

Orientações para elaboração do Trabalho Interdisciplinar Big Data + ML

2º semestre 2021 – Prof. Bianca

Detalhamento da parte do trabalho referente à Big Data

Processamento de algoritmos de ML em ambiente distribuído, formado por um cluster de máquinas rodando hadoop map reduce/spark. Estes ambientes podem ser encontrados na AWS e Google, entre outros.

1. Organização do conteúdo do trabalho

Os trabalhos devem ter os seguintes tópicos:

- a) Introdução, que deve ser uma breve contextualização.
- b) Descrição dos dados (fonte, período, volume, formato, amostras dos dados)
- c) Workflow/pipeline de arquitetura
- d) Infraestrutura (Configuração do cluster como nro de nós, tipo de máquinas, permissões)
- e) Análise do Processamento (análise do desempenho das operações ETL, segundo critérios como tempo de processamento, formas intermediárias de armazenamento dos dados (partições), comparação do desempenho com diferentes configurações de cluster)
- f) Funções definidas para processamento distribuído (map, reduce, outras)
- g) Conclusões (limitações e vantagens)
- h) Referências

1. Introdução

1.1. Objetivo

Este projeto tem como objetivo a aplicação das técnicas de Ciência de Dados no conjunto de dados (dataset) de "Wine Quality". A escolha do dataset está relacionado a continuidade e comparação do processo de desenvolvimento do projeto do semestre anterior.

O foco principal deste projeto é o desenvolvimento de uma solução de aprendizado de máquina cujo treinamento e processamento deverão ser realizados em ambiente de nuvem (Amazon Web Service - AWS), com processamento distribuído (cluster de computadores), explorando o uso de paralelismo (spark/mapreduce).

Além do ambiente distribuído, aplicou-se técnicas de machine learning (normalização, pipeline, cross validation), e alguns algoritmos de classificação para avaliação de performance e resultados.

1.2. Descrição Dataset

A base de dados consolidada é composta por 1599 amostra de vinho tinto e 4898 amostra de vinho branco, totalizando 6497 amostra e descritas pela coluna "style". O dataset foi criado Paulo Cortez (Univ. Minho), António Cerdeira, Fernando Almeida, Telmo Matos and José Reis em 2009. Cada linha da base de dados é uma amostra de vinho com suas medições físico-químicas, tipo do vinho (vermelho ou branco) e a nota/avaliação de especialistas.

Neste projeto, consideramos a análise para a classificação do vinho, podendo ser "tinto" ou "branco".

1.3. Detalhamento dos atributos

fixed_acidity = Acidez Fixa: ácido não volátil encontrado no vinho, que é tartárico, málico, cítrico e succínico. Todos esses ácidos são originários das uvas, com exceção do ácido succínico, que é produzido pela levedura durante o processo de fermentação

volatile_acidity = Acidez volátil: os elementos ácidos de um vinho que são gasosos, em vez de líquidos e, portanto, podem ser sentidos como um cheiro, mostrando um aroma, em vez de encontrados no palato

citric_acid = Ácido cítrico: um ácido orgânico fraco, frequentemente usado como conservante natural ou aditivo para alimentos ou bebidas para adicionar um sabor azedo e frescor aos alimentos

residual_sugar = Açúcar residual: a quantidade de açúcar remanescente após a parada da fermentação, é raro encontrar vinhos com menos de 1 grama / litro e vinhos com mais de 45 gramas / litro são considerados doces

chlorides = Cloretos: a quantidade de sal no vinho

free_sulfur_dioxide = Dióxido de Enxofre Livre (SO₂): o SO₂ é usado em todas as fases do processo de vinificação para prevenir a oxidação e o crescimento microbiano. Quantidades excessivas de SO₂ podem inibir a fermentação e causar efeitos sensoriais indesejáveis

total_sulfur_dioxide = Dióxido de Enxofre Total: quantidade das formas livre e ligada de SO₂; em baixas concentrações, SO₂ é principalmente indetectável no vinho, mas em concentrações de SO₂ livre acima de 50 ppm, SO₂ torna-se evidente no nariz e no sabor do vinho

density = Densidade: a densidade é próxima à da água, dependendo da porcentagem de álcool e açúcar

pH: produtores de vinho usam o pH como forma de medir a maturação em relação à acidez. Vinhos de pH baixo terão gosto ácido e crocante, enquanto vinhos de pH mais alto são mais suscetíveis ao crescimento bacteriano. A maioria do pH do vinho está em torno de 3 ou 4

sulphates = Sulfatos: aditivo do vinho que pode contribuir para os níveis de gás de dióxido de enxofre (SO₂SO₂), que atua como um antimicrobiano e antioxidante

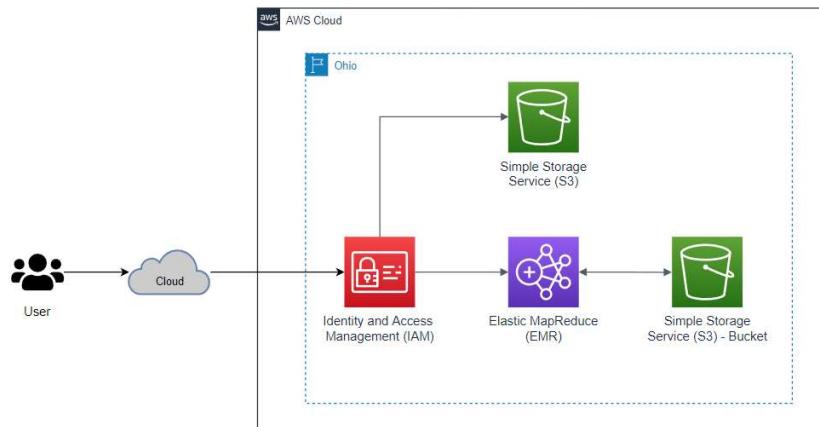
alcohol = Álcool: o teor de álcool do vinho em percentual

quality = Qualidade: notas/avaliações realizadas por no mínimo 3 especialistas em vinhos

style = Tipo: Tipo do Vino (Vermelho ou Branco)

2. Infraestrutura

2.1. Infraestrutura AWS



2.2. AWS Applications

2.2.1. Apache Spark

Apache Spark is a distributed processing framework and programming model that helps you do machine learning, stream processing, or graph analytics using Amazon EMR clusters. Similar to Apache Hadoop, Spark is an open-source, distributed processing system commonly used for big data workloads. However, Spark has several notable differences from Hadoop MapReduce. Spark has an optimized directed acyclic graph (DAG) execution engine and actively caches data in-memory, which can boost performance, especially for certain algorithms and interactive queries.

Spark version information for emr-5.33.0		
Amazon EMR Release Label	Spark Version	Components Installed With Spark
emr-5.33.0	Spark 2.4.7	aws-sagemaker-spark-sdk, emrfs, emr-goodies, emr-ddb, emr-s3-select, hadoop-client, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-https-server, hadoop-kms-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, hadoop-yarn-timeline-server, hudi, hudi-spark, livy-server, nginx, r, spark-client, spark-history-server, spark-on-yarn, spark-yarn-slave

2.2.2. Apache Hive

Hive is an open-source, data warehouse, and analytic package that runs on top of a Hadoop cluster. Hive scripts use an SQL-like language called Hive QL (query language) that abstracts programming models and supports typical data warehouse interactions. Hive enables you to avoid the complexities of writing Tez jobs based on directed acyclic graphs (DAGs) or MapReduce programs in a lower level computer language, such as Java.

Hive version information for emr-6.4.0		
Amazon EMR Release Label	Hive Version	Components Installed With Hive
emr-6.4.0	Hive 3.1.2	emrfs, emr-ddb, emr-goodies, emr-kinesis, emr-s3-dist-cp, emr-s3-select, hadoop-client, hadoop-mapred, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-https-server, hadoop-kms-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, hadoop-yarn-timeline-server, hive-client, hive-hbase, hcatalog-server, hive-server2, hudi, mariadb-server, tez-on-yarn, zookeeper-client, zookeeper-server

2.2.3. Apache Livy

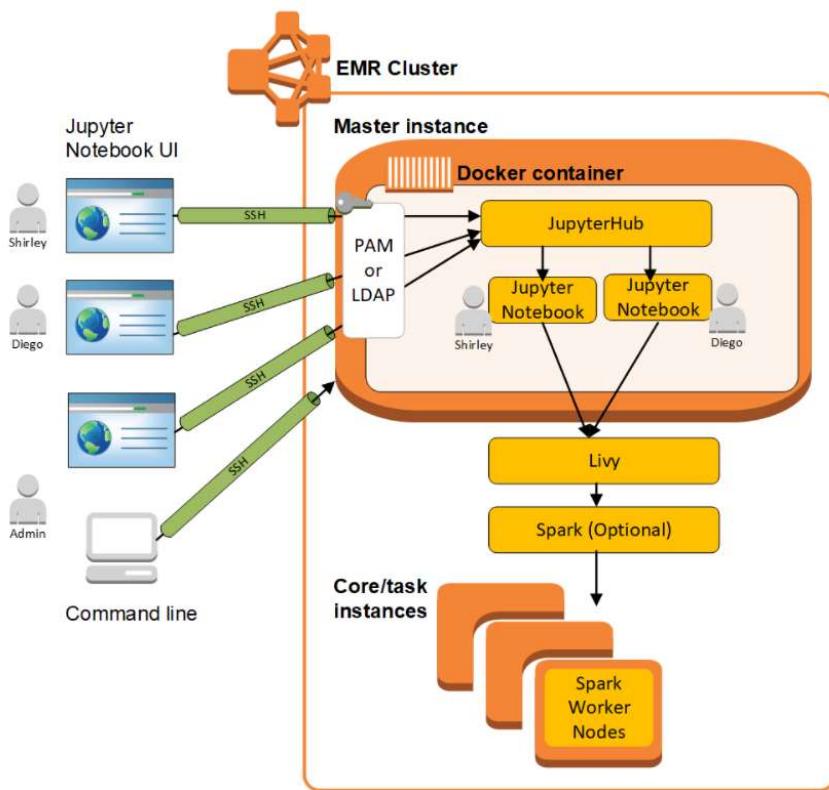
Livy enables interaction over a REST interface with an EMR cluster running Spark. You can use the REST interface or an RPC client library to submit Spark jobs or snippets of Spark code, retrieve results synchronously or asynchronously, and manage Spark Context. For more information, see the Apache Livy website. Livy is included in Amazon EMR release version 5.9.0 and later.

Livy version information for emr-6.4.0		
Amazon EMR Release Label	Livy Version	Components Installed With Livy
emr-6.4.0	Livy 0.7.1	aws-sagemaker-spark-sdk, emrfs, emr-goodies, emr-ddb, hadoop-client, hadoop-hdfs-datanode, hadoop-hdfs-library, hadoop-hdfs-namenode, hadoop-kms-server, hadoop-yarn-nodemanager, hadoop-yarn-resourcemanager, hadoop-yarn-timeline-server, r, spark-client, spark-history-server, spark-on-yarn, spark-yarn-slave, livy-server, nginx

2.2.4. Amazon EMR Notebook based on Jupyter Notebook

EMR Notebooks is a Jupyter Notebook environment built in to the Amazon EMR console that allows you to quickly create Jupyter notebooks, attach them to Spark clusters, and then open the Jupyter Notebook editor in the console to remotely run queries and code. An EMR notebook is saved in Amazon S3 independently from clusters for durable storage, quick access, and flexibility. You can have multiple notebooks open, attach multiple notebooks to a single cluster, and re-use a notebook on different clusters.

2.2.4. Amazon EMR Notebook Arquitetura



2.3. Create EMR Cluster

Lista de aplicações do Cluster

Hardware Configuration

Hardware Description

Cluster Nodes and Instances

Choose the instance type, number of instances, and a purchasing option. [Learn more about instance purchasing options](#)

Node type	Instance type	Instance count	Purchasing option
Master	m5.xlarge	1 Instances	<input checked="" type="radio"/> On-demand <small>1</small> <input type="radio"/> Spot <small>1</small> <small>[Use on-demand as max price]</small>
Master Instance Group	4 vCore, 16 GB memory, EBS only storage EBS Storage: 64 GB Add configuration settings		
Core	m5.xlarge	3 Instances	<input checked="" type="radio"/> On-demand <small>1</small> <input type="radio"/> Spot <small>1</small> <small>[Use on-demand as max price]</small>
Core - 2	4 vCore, 16 GB memory, EBS only storage EBS Storage: 32 GB Add configuration settings		

+ Add task instance group

Total core and task units 3 Total units

Cluster scaling

Adjust the number of Amazon EC2 instances available to an EMR cluster via EMR-managed scaling or a custom automatic scaling policy. [Learn more](#)

Cluster scaling Enable Cluster Scaling
 Use EMR-managed scaling
 Create a custom automatic scaling policy

EMR-managed scaling

EMR will automatically increase and decrease the number of instances in core and task nodes based on workload. Set a minimum and maximum limit of the number of instances for the cluster nodes. Master nodes do not scale.

Core and task units

Minimum: 3
Maximum: 6
On-demand limit: 6
Maximum Core Node: 6

Habilitando Cluster Scaling

Total core and task units 3 Total units

Cluster scaling

Adjust the number of Amazon EC2 instances available to an EMR cluster via EMR-managed scaling or a custom automatic scaling policy. [Learn more](#)

Cluster scaling Enable Cluster Scaling
 Use EMR-managed scaling
 Create a custom automatic scaling policy

EMR-managed scaling

EMR will automatically increase and decrease the number of instances in core and task nodes based on workload. Set a minimum and maximum limit of the number of instances for the cluster nodes. Master nodes do not scale.

Core and task units

Minimum: 3
Maximum: 6
On-demand limit: 6
Maximum Core Node: 6

Auto-termination

Select a time to have the cluster terminate after the cluster becomes idle. Choose a minimum of 1 minute or a max of 24 hours. [Learn more](#)

Auto-termination Enable auto-termination
Terminate cluster when it is idle after (0 hours 10 minutes)

EBS Root Volume

Specify the root device volume size up to 100 GiB. This sizing applies to all instances in the cluster. [Learn more](#)

Root device EBS volume size 20 GiB

Aumento hardware configuration

The screenshot shows the 'Cluster scaling' section with 'Use EMR-managed scaling' selected. It includes fields for 'Core and task units' (Minimum: 2, Maximum: 10, On-demand limit: 10, Maximum Core Node: 10). The 'Auto-termination' section is also visible. The 'EBS Root Volume' section shows a root device volume size of 10 GiB. Navigation buttons 'Cancel', 'Previous', and 'Next' are at the bottom.

Definição de Bucket de LOG

The screenshot shows the 'General Options' section with 'Logging' checked and an S3 folder of 's3://bucket.omar'. The 'Tags' section contains a key-value pair 'creator:NOTEBOOK_CONSOLE'. The 'Additional Options' section includes 'EMRFS consistent view' and a 'Custom AMI ID' dropdown set to 'None'. Navigation buttons 'Cancel', 'Previous', and 'Next' are at the bottom.

Configuração padrão (key, role, profile, etc.)

Step 1: Software and Steps
Step 2: Hardware
Step 3: General Cluster Settings
Step 4: Security

Security Options

EC2 key pair: RSA PEM_Omar
Cluster visible to all IAM users in account:

Permissions

Default: Custom
Select custom roles to tailor permissions for your cluster:
EMR role: EMR_DefaultRole
EC2 Instance profile: EMR_EC2_DefaultRole
Auto Scaling role: Proceed without role

Security Configuration

An EC2 security group acts as a virtual firewall for your cluster nodes to control inbound and outbound traffic. There are two types of security groups you can configure: EMR managed security groups and additional security groups. EMR will automatically update the rules in the EMR managed security groups in order to launch a cluster. [Learn more](#)

Type	EMR managed security groups	Additional security groups
Master	Default: sg-0300ee4848977bda (ElasticMapReduce) <input checked="" type="checkbox"/> EMR will automatically update the selected group	No security groups selected <input type="checkbox"/>
Core & Task	Default: sg-0de18a319cb264079 (ElasticMapReduce) <input checked="" type="checkbox"/> EMR will automatically update the selected group	No security groups selected <input type="checkbox"/>

[Creates a security group](#)

Cancel Previous Create cluster

Feedback English (US) © 2021, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Running server

Cluster: EMR_cluster_Omar_002 Waiting Cluster ready to run steps.

Summary

ID: j-2QQT901GR47H
Creation date: 2021-12-05 15:23 (UTC-3)
Elapsed time: 7 minutes
After last step completes: Cluster waits
Termination protection: Off Change
Tags: creator = NOTEBOOK_CONSOLE [View All / Edit](#)

Master public DNS: ec2-13-59-234-156.us-east-2.compute.amazonaws.com [View](#)
Connect to the Master Node Using SSH

Application user interfaces

Persistent user interfaces: Spark history server, YARN timeline server, Tez UI
On-cluster user: Not Enabled [Enable an SSH Connection](#)

Configuration details

Release label: emr-5.33.1
Hadoop distribution: Amazon 2.10.1
Applications: Spark 2.4.7, Livy 0.7.0, Hive 2.3.7, JupyterEnterpriseGateway 2.1.0
Log URI: s3://aws-logs-00972966644.us-east-2/elastisompreduer/ [View](#)

EMRFS consistent view: Disabled
Custom AMI ID: --

Network and hardware

Availability zone: us-east-2b
Subnet ID: subnet-05798078 [View](#)
Master: Running 1 m5.xlarge
Core: Running 2 m5.xlarge
Task: --
Cluster scaling: EMR-managed scaling
Auto-termination: Terminate if idle for 10 minutes

Feedback English (US) © 2021, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Running Jupyter Notebook

Notebook: PI_Omar Ready Workspace(notebook) is ready to run jobs on cluster j-11RDB040LSPLE.

[Open in JupyterLab](#) [Open in Jupyter](#) [Stop](#) [Delete](#)

Notebook

Notebook ID: e-3UBAAQ80F5GR21GUCMA770VVS
 Description: --
 Last modified: 0 seconds ago
 Last modified by: user/omar.fidelis
 Created on: 2021-11-22 21:27 (UTC-3)
 Created by: user/omar.fidelis
 Service IAM role: EMR_Notebooks_DefaultRole
 Security groups for sg-0feeb1b6e9ee7a01
 master instance:
 Security groups for sg-0111cc053cc68e26b
 notebook instance:
 Notebook tags: creatorUserId=AIDAQEQ7R7JUO2YK2JQ6Q View All / Edit
 Notebook location: s3://bucket.omar/

Cluster

Cluster: EMR_cluster_Omar_002
 Cluster ID: j-11RDB040LSPLE
 Cluster status: Waiting Cluster ready to run steps.
 Cluster tags: creator=NOTEBOOK_CONSOLE View All
 Step logs: s3://aws-logs-009729866344-us-east-2/elastismapreduce/

Git repositories

The repository can be linked to a notebook once the notebook is ready. Make sure your cluster, service role and security groups have the required settings. [Learn more](#)

[Link new repository](#) [Unlink repository](#)

Repository name	URL	Branch	Link status	Failure reason

© 2021, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

2.4. Hardware Configuration

Summary Configuration

Cluster: EMR_cluster_Omar_002 Waiting Cluster ready to run steps.

[Clone](#) [Terminate](#) [AWS CLI export](#)

[Summary](#) [Application user interfaces](#) [Monitoring](#) [Hardware](#) [Configurations](#) [Events](#) [Steps](#) [Bootstrap actions](#)

Summary

ID: j-2QQT901GR417H
 Creation date: 2021-12-05 15:23 (UTC-3)
 Elapsed time: 7 minutes
 After last step completes: Cluster waits
 Termination protection: Off Change
 Tags: creator=NOTEBOOK_CONSOLE View All / Edit
 Master public DNS: ec2-13-59-234-156.us-east-2.compute.amazonaws.com
 Connect to the Master Node Using SSH

Configuration details

Release label: emr-5.33.1
 Hadoop distribution: Amazon 2.10.1
 Applications: Spark 2.4.7, Livy 0.7.0, Hive 2.3.7, JupyterEnterpriseGateway 2.1.0
 Log URI: s3://aws-logs-009729866344-us-east-2/elastismapreduce/

EMRFS consistent view: Disabled
 Custom AMI ID: --

Application user interfaces

Persistent user interfaces: Spark history server, YARN timeline server, Tez UI
 On-cluster user Not Enabled Enable an SSH Connection interfaces: --

Network and hardware

Availability zone: us-east-2b
 Subnet ID: subnet-05798078
 Master: Running 1 m5.xlarge
 Core: Running 2 m5.xlarge
 Task: --
 Cluster scaling: EMR-managed scaling
 Auto-termination: Terminate if idle for 10 minutes

Security and access

Key name: kp_RSA_PEM_Omar
 EC2 instance profile: EMR_EC2_DefaultRole
 EMR role: EMR_DefaultRole
 Visible to all users: All Change
 Security groups for Master: sg-0390ee484bf9777bda (ElasticMapReduce-master)
 Security groups for Core & Task: sg-0de18a319cb264079 (ElasticMapReduce-task: slave)

© 2021, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

Hardware Configuration

Cluster: EMR_cluster_Omar_002 Waiting Cluster ready to run steps.

Summary Application user interfaces Monitoring Hardware Configurations Events Steps Bootstrap actions

Instance groups

ID	Status	Node type & name	Instance type	Instance count	Purchasing option
ig-1BK80S8ADEPZ2	Running	MASTER	m5.xlarge 4 vCore, 16 GiB memory, EBS only storage EBS Storage: 64 GiB	1 Instances	On-demand

Cluster Scaling Policy

No scaling enabled

Auto-termination

Select a time to have the cluster terminate after the cluster becomes idle. Choose a minimum of 1 minute or a max of 24 hours. [Learn more](#)

Auto-termination:

Application Configuration

Cluster: EMR_cluster_Omar_002 Waiting Cluster ready to run steps.

Summary Application user interfaces Monitoring Hardware Configurations Events Steps Bootstrap actions

Persistent application user interfaces

Applications installed on the Amazon EMR cluster publish user interfaces (UI) as web sites to monitor cluster activity. Persistent UI logs are available for 30 days after an application ends. Persistent UI don't require SSH tunneling. They are hosted off of the cluster.

Application user interface

- YARN timeline server
- Spark history server
- Tez UI

On-cluster application user interfaces

On-cluster UI are available only while clusters are running. Because they are hosted on the master node, on-cluster UI require a connection via SSH tunneling. Set up SSH tunneling before accessing these application UI. [Learn more](#)

Application	User interface URL	Status
HDFS Name Node	http://ec2-3-23-94-223.us-east-2.compute.amazonaws.com:50070/	SSH tunnel not enabled
Spark History Server	http://ec2-3-23-94-223.us-east-2.compute.amazonaws.com:18080/	SSH tunnel not enabled
LWY	http://ec2-3-23-94-223.us-east-2.compute.amazonaws.com:8998/	SSH tunnel not enabled
Resource Manager	http://ec2-3-23-94-223.us-east-2.compute.amazonaws.com:8088/	SSH tunnel not enabled

The following table lists web interfaces you can view on the task nodes:

Application	User interface URL
HDFS Data Node	http://ec2-000-000-000-000.compute-1.amazonaws.com:50075/
Node Manager	http://ec2-000-000-000-000.compute-1.amazonaws.com:8042/

High-level application history

Amazon EMR collects information from YARN applications on your cluster and keeps a summary of historical information for seven days after applications have completed. [Learn more](#)

YARN applications (9)

Application ID	Type	Action	Status	Start time (UTC-3)	Duration	Finish time (UTC-3)	User
...

Spark Sessions

The screenshot shows the AWS EMR console. On the left, a sidebar lists services: Amazon EMR, EMR Studio, EMR on EC2, Clusters, Notebooks, Git repositories, Security configurations, Block public access, VPC subnets, Events, EMR on EKS, and Virtual clusters. The 'Clusters' section is expanded, showing 'On-cluster application user interfaces' and 'High-level application history'. The 'On-cluster application user interfaces' section lists various application names and their corresponding SSH tunnel URLs. The 'High-level application history' section shows a table of YARN applications with columns for Application ID, Type, Action, Status, Start time (UTC-3), Duration, Finish time (UTC-3), and User. There are 9 applications listed, all of which succeeded.

3. Preparação do ambiente

In [1]:

```
# Import Library
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
```

▶ Spark Job Progress

Starting Spark application

ID	YARN Application ID	Kind	State
----	---------------------	------	-------

1	application_1638734260039_0002	pyspark	idle	Link
---	--------------------------------	---------	------	--

SparkSession available as 'spark'.

In [2]:

```
# Start Spark Session
spark = SparkSession \
    .builder \
    .getOrCreate()
```

```
In [3]: # Data Structure
schema = StructType([ \
    StructField("Acidez_Fixa", FloatType(), True), \
    StructField("Acidez_Volatil", FloatType(), True), \
    StructField("Acido_Citrico", FloatType(), True), \
    StructField("Acucar_Residual", FloatType(), True), \
    StructField("Cloreto_Sodio", FloatType(), True), \
    StructField("Dioxido_Enxofre_Livre", FloatType(), True), \
    StructField("Dioxido_Enxofre_Total", FloatType(), True), \
    StructField("Densidade", FloatType(), True), \
    StructField("pH", FloatType(), True), \
    StructField("Sulfato", FloatType(), True), \
    StructField("Alcool", FloatType(), True), \
    StructField("Qualidade", FloatType(), True), \
    StructField("Tipo", StringType(), True)])
```

3.1. Preparação dataset

```
In [4]: # Import Dataset from S3 Bucket
df = spark.read.schema(schema).option("header", "true").csv(r"s3n://bucket.omar/da
```

```
In [5]: # Overview
df.show(5)
```

▶ Spark Job Progress

```
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+
---+
|Acidez_Fixa|Acidez_Volatil|Acido_Citrico|Acucar_Residual|Cloreto_Sodio|Dioxido_Enxofre_Livre|Dioxido_Enxofre_Total|Densidade| pH|Sulfato|Alcool|Qualidade|Tipo|
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+
---+
| 11.0 | 7.4 | 0.7 | 0.0 | 1.9 | 0.076 | | |
| 11.0 | 34.0 | 0.9978 | 3.51 | 0.56 | 9.4 | 5.0 | red |
| 25.0 | 7.8 | 0.88 | 0.0 | 2.6 | 0.098 |
| 25.0 | 67.0 | 0.9968 | 3.2 | 0.68 | 9.8 | 5.0 | red |
| 15.0 | 7.8 | 0.76 | 0.04 | 2.3 | 0.092 |
| 15.0 | 54.0 | 0.997 | 3.26 | 0.65 | 9.8 | 5.0 | red |
| 17.0 | 11.2 | 0.28 | 0.56 | 1.9 | 0.075 |
| 17.0 | 60.0 | 0.998 | 3.16 | 0.58 | 9.8 | 6.0 | red |
| 11.0 | 7.4 | 0.7 | 0.0 | 1.9 | 0.076 |
| 11.0 | 34.0 | 0.9978 | 3.51 | 0.56 | 9.4 | 5.0 | red |
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+
---+
only showing top 5 rows
```

In [6]: *# Staset Schema*
df.printSchema()

```
root
|-- Acidez_Fixa: float (nullable = true)
|-- Acidez_Volatil: float (nullable = true)
|-- Acido_Citrico: float (nullable = true)
|-- Acucar_Residual: float (nullable = true)
|-- Cloreto_Sodio: float (nullable = true)
|-- Dioxido_Enxofre_Livre: float (nullable = true)
|-- Dioxido_Enxofre_Total: float (nullable = true)
|-- Densidade: float (nullable = true)
|-- pH: float (nullable = true)
|-- Sulfato: float (nullable = true)
|-- Alcool: float (nullable = true)
|-- Qualidade: float (nullable = true)
|-- Tipo: string (nullable = true)
```

In [7]: `# No missing values`
`df.select(df.columns).describe().show()`

► Spark Job Progress

summary	Acidez_Fixa	Acidez_Volatil	Acido_Citrico	Acucar_Residual
Densidade	Cloreto_Sodio	Dioxido_Enxofre_Livre	Dioxido_Enxofre_Total	pH
Qualidade	Tipo	Sulfato	Alcool	
count	6497	6497	6497	6497
6497	6497	6497	6497	6497
6497	6497	6497	6497	6497
6497	6497	6497	6497	6497
mean	7.2153070684174345	0.3396659997464309	0.31863321590700183	5.443235335688381
35688381	0.05603386190293733	30.525319378174544	115.7445744189626	0.9946966336675106
9946966336675106	3.218500847887659	0.5312682763074607	10.491800813027597	5.818377712790519
5.818377712790519	null			
stddev	1.2964337581432526	0.16463647361833078	0.14531786519512516	4.757803743705875
43705875	0.03503360134013471	17.74939977200251	56.52185452263024	0.002998673927059...
2998673927059...	0.16078720153691367	0.1488058737693289	1.1927117488301993	0.8732552715311255
0.8732552715311255	null			
min	3.8	0.08	0.0	
0.6	0.009	1.0	6.0	
0.98711	2.72	0.22	8.0	
3.0	red			
max	15.9	1.58	1.66	
65.8	0.611	289.0	440.0	
1.03898	4.01	2.0	14.9	
9.0	white			

In [8]: `# Create VIEW for dataset`
`df.createOrReplaceTempView("table_wine")`

In [9]: *# Quantidade de amostrar por tipo de vinho*
df2 = spark.sql('SELECT count(*) as Total, Tipo from table_wine group by Tipo')
df2.collect()

► Spark Job Progress

[Row(Total=4898, Tipo='white'), Row(Total=1599, Tipo='red')]

In [10]: *# Valor máximo de cada atributo*
df2 = spark.sql('SELECT max(Acidez_Fixa),\n max(Acidez_Volatil),\n max(Acido_Citrico),\n max(Acucar_Residual),\n max(Cloreto_Sodio),\n max(Dioxido_Enxofre_Livre),\n max(Dioxido_Enxofre_Total),\n max(Densidade),\n max(pH),\n max(Sulfato),\n max(Alcool),\n max(Qualidade) from table_wine')
df2.collect()

► Spark Job Progress

[Row(max(Acidez_Fixa)=15.899999618530273, max(Acidez_Volatil)=1.5800000429153442, max(Acido_Citrico)=1.659999966621399, max(Acucar_Residual)=65.80000305175781, max(Cloreto_Sodio)=0.6110000014305115, max(Dioxido_Enxofre_Livre)=289.0, max(Dioxido_Enxofre_Total)=440.0, max(Densidade)=1.0389800071716309, max(pH)=4.01000228881836, max(Sulfato)=2.0, max(Alcool)=14.899999618530273, max(Qualidade)=9.0)]

In [11]:

```
# Valor mínimo de cada atributo
df2 = spark.sql('SELECT min(Acidez_Fixa),\
                 min(Acidez_Volatil),\
                 min(Acido_Citrico),\
                 min(Acucar_Residual),\
                 min(Cloreto_Sodio),\
                 min(Dioxido_Enxofre_Livre),\
                 min(Dioxido_Enxofre_Total),\
                 min(Densidade),\
                 min(pH),\
                 min(Sulfato),\
                 min(Alcool),\
                 min(Qualidade) from table_wine')
df2.collect()
```

► Spark Job Progress

```
[Row(min(Acidez_Fixa)=3.799999952316284, min(Acidez_Volatil)=0.07999999821186066, min(Acido_Citrico)=0.0, min(Acucar_Residual)=0.6000000238418579, min(Cloreto_Sodio)=0.008999999612569809, min(Dioxido_Enxofre_Livre)=1.0, min(Dioxido_Enxofre_Total)=6.0, min(Densidade)=0.9871100187301636, min(pH)=2.7200000286102295, min(Sulfato)=0.2199999988079071, min(Alcool)=8.0, min(Qualidade)=3.0)]
```

In [12]:

```
# Valor médio de cada atributo
df2 = spark.sql('SELECT avg(Acidez_Fixa),\
                 avg(Acidez_Volatil),\
                 avg(Acido_Citrico),\
                 avg(Acucar_Residual),\
                 avg(Cloreto_Sodio),\
                 avg(Dioxido_Enxofre_Livre),\
                 avg(Dioxido_Enxofre_Total),\
                 avg(Densidade),\
                 avg(pH),\
                 avg(Sulfato),\
                 avg(Alcool),\
                 avg(Qualidade) from table_wine')
df2.collect()
```

► Spark Job Progress

```
[Row(avg(Acidez_Fixa)=7.2153070684174345, avg(Acidez_Volatil)=0.3396659997464309, avg(Acido_Citrico)=0.31863321590700183, avg(Acucar_Residual)=5.443235335688381, avg(Cloreto_Sodio)=0.05603386190293733, avg(Dioxido_Enxofre_Livre)=30.525319378174544, avg(Dioxido_Enxofre_Total)=115.7445744189626, avg(Densidade)=0.9946966336675106, avg(pH)=3.218500847887659, avg(Sulfato)=0.5312682763074607, avg(Alcool)=10.491800813027597, avg(Qualidade)=5.818377712790519)]
```

In [13]: *# 10 amostras melhor avaliadas*

```
df2 = spark.sql('SELECT Qualidade, Tipo from table_wine order by Qualidade desc ')
df2.collect()
```

► Spark Job Progress

```
[Row(Qualidade=9.0, Tipo='white'), Row(Qualidade=9.0, Tipo='white'), Row(Qualidade=9.0, Tipo='white'), Row(Qualidade=9.0, Tipo='white'), Row(Qualidade=9.0, Tipo='white'), Row(Qualidade=8.0, Tipo='red'), Row(Qualidade=8.0, Tipo='red'), Row(Qualidade=8.0, Tipo='red'), Row(Qualidade=8.0, Tipo='red'), Row(Qualidade=8.0, Tipo='red')]
```

In [14]: *# Quantidade de amostras por avaliação e Label*

```
df2 = spark.sql('SELECT count(*) as Total, Qualidade, Tipo\
                 from table_wine\
                 group by Qualidade, Tipo\
                 order by Qualidade desc, Tipo')
df2.collect()
```

► Spark Job Progress

```
[Row(Total=5, Qualidade=9.0, Tipo='white'), Row(Total=18, Qualidade=8.0, Tipo='red'), Row(Total=175, Qualidade=8.0, Tipo='white'), Row(Total=199, Qualidade=7.0, Tipo='red'), Row(Total=880, Qualidade=7.0, Tipo='white'), Row(Total=638, Qualidade=6.0, Tipo='red'), Row(Total=2198, Qualidade=6.0, Tipo='white'), Row(Total=681, Qualidade=5.0, Tipo='red'), Row(Total=1457, Qualidade=5.0, Tipo='white'), Row(Total=53, Qualidade=4.0, Tipo='red'), Row(Total=163, Qualidade=4.0, Tipo='white'), Row(Total=10, Qualidade=3.0, Tipo='red'), Row(Total=20, Qualidade=3.0, Tipo='white')]
```

```
In [15]: # Transform categorical attribute ("Tipo") to binary attribute
df = df.withColumn('label', col('Tipo') == 'red')
df = df.drop('Tipo')
df.show(5)
```

▶ Spark Job Progress

	Acidez_Fixa	Acidez_Volatil	Acido_Citrico	Acucar_Residual	Cloreto_Sodio	Dioxido_Enxofre_Livre	Dioxido_Enxofre_Total	Densidade	pH	Sulfato	Alcool	Qualidad e	label
11.0	7.4	0.7	0.0	3.51	0.56	9.4	5.0	5.0	0.9978	true	0.076		
25.0	7.8	0.88	0.0	3.2	0.68	9.8	5.0	5.0	0.9968	true	0.098		
15.0	7.8	0.76	0.04	3.26	0.65	9.8	5.0	5.0	0.997	true	0.092		
17.0	11.2	0.28	0.56	3.16	0.58	9.8	6.0	6.0	0.998	true	0.075		
11.0	7.4	0.7	0.0	3.51	0.56	9.4	5.0	5.0	0.9978	true	0.076		

only showing top 5 rows

```
In [16]: # Cast binary attribute to numerical attribute named "label" (standard)
from pyspark.sql.types import IntegerType
df = df.withColumn("label", col("label").cast(IntegerType()))
df.printSchema()
```

```
root
|-- Acidez_Fixa: float (nullable = true)
|-- Acidez_Volatil: float (nullable = true)
|-- Acido_Citrico: float (nullable = true)
|-- Acucar_Residual: float (nullable = true)
|-- Cloreto_Sodio: float (nullable = true)
|-- Dioxido_Enxofre_Livre: float (nullable = true)
|-- Dioxido_Enxofre_Total: float (nullable = true)
|-- Densidade: float (nullable = true)
|-- pH: float (nullable = true)
|-- Sulfato: float (nullable = true)
|-- Alcool: float (nullable = true)
|-- Qualidade: float (nullable = true)
|-- label: integer (nullable = true)
```

4. Processamento (Machine Learning)

```
In [17]: # Import Libraries - Machine Learning
from pyspark.ml.feature import VectorAssembler, Normalizer, StandardScaler
from pyspark.ml.classification import LogisticRegression, NaiveBayes, LinearSVC,
from pyspark.ml.evaluation import MulticlassClassificationEvaluator, BinaryClassi
from pyspark.ml import Pipeline
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
```

```
In [18]: # Split dataset (training 80% and test 20%)
training, test = df.randomSplit([0.8, 0.2], seed=10)
```

```
In [19]: # Create Assembler
assembler = VectorAssembler(inputCols=['Acidez_Fixa', 'Acidez_Volatil', 'Acido_Citr
                                         'Cloreto_Sodio', 'Dioxido_Exnofre_Livre', 'D
                                         'Densidade', 'pH', 'Sulfato', 'Alcool', 'Quali
                                         outputCol="inputFeatures")
```

```
In [20]: # Standard Scaler (Normalize)
#scaler = Normalizer(inputCol = "inputFeatures", outputCol = "features")
scaler = StandardScaler(inputCol = "inputFeatures", outputCol = "features", withS
```

4.1. Logistic Regression

```
In [21]: # Model Creation
lr = LogisticRegression()
```

```
In [22]: # Creating Pipeline
#pipeline_lr = Pipeline(stages = [assembler, lr])
pipeline_lr = Pipeline(stages = [assembler, scaler, lr])
```

```
In [23]: # Parameter
paramgrid_lr = ParamGridBuilder().addGrid(lr.regParam, [0.0, 0.1]).addGrid(lr.ma)
```

```
In [24]: # Evaluator
evaluator_lr = MulticlassClassificationEvaluator(metricName = "accuracy")
```

```
In [25]: # Cross Validation
crossval_lr = CrossValidator(estimator = pipeline_lr, estimatorParamMaps = paramg
                           evaluator = evaluator_lr, numFolds = 3, seed=10)
```

```
In [26]: # Training and evaluation
cvModel_lr = crossval_lr.fit(training)
evaluator_lr.evaluate(cvModel_lr.transform(test))
```

► Spark Job Progress

0.7574568288854003

```
In [27]: # Evaluator
evaluator_lr = MulticlassClassificationEvaluator(metricName = "f1")
```

```
In [28]: # Cross Validation
crossval_lr = CrossValidator(estimator = pipeline_lr, estimatorParamMaps = paramMaps,
                             evaluator = evaluator_lr, numFolds = 3, seed=10)
```

```
In [29]: # Training and evaluation
cvModel_lr = crossval_lr.fit(training)
evaluator_lr.evaluate(cvModel_lr.transform(test))
```

► Spark Job Progress

0.653716541955618

4.2. Naive Bayes

```
In [30]: # Model Creation
nb = NaiveBayes()
```

```
In [31]: # Creating Pipeline
#pipeline_nb = Pipeline(stages = [assembler, nb])
pipeline_nb = Pipeline(stages = [assembler, scaler, nb])
```

```
In [32]: # Parameter
paramgrid_nb = ParamGridBuilder().build()
```

```
In [33]: # Evaluator
evaluator_nb = MulticlassClassificationEvaluator(metricName = "accuracy")
```

```
In [34]: # Cross Validation
crossval_nb = CrossValidator(estimator = pipeline_nb, estimatorParamMaps = paramMaps_nb,
                             evaluator = evaluator_nb, numFolds = 3, seed=10)
```

```
In [35]: # Training and evaluation
cvModel_nb = crossval_nb.fit(training)
evaluator_nb.evaluate(cvModel_nb.transform(test))
```

▶ Spark Job Progress

0.957613814756672

```
In [36]: # Evaluator
evaluator_nb = MulticlassClassificationEvaluator(metricName = "f1")
```

```
In [37]: # Cross Validation
crossval_nb = CrossValidator(estimator = pipeline_nb, estimatorParamMaps = paramMaps_nb,
                             evaluator = evaluator_nb, numFolds = 3, seed=10)
```

```
In [38]: # Training and evaluation
cvModel_nb = crossval_nb.fit(training)
evaluator_nb.evaluate(cvModel_nb.transform(test))
```

▶ Spark Job Progress

0.9567514419209051

4.3. Linear Support Vector Machine

```
In [39]: # Model Creation
lsvc = LinearSVC()
```

```
In [40]: # Creating Pipeline
#pipeline_lsVC = Pipeline(stages = [assembler, lsVC])
pipeline_lsVC = Pipeline(stages = [assembler, scaler, lsvc])
```

```
In [41]: # Parameter
paramgrid_lsVC = ParamGridBuilder().build()
```

```
In [42]: # Evaluator
evaluator_lsVC = MulticlassClassificationEvaluator(metricName = "accuracy")
```

```
In [43]: # Cross Validation
crossval_lsvc = CrossValidator(estimator = pipeline_lsvc, estimatorParamMaps = pa
                                evaluator = evaluator_lsvc, numFolds = 3, seed=10)
```

```
In [44]: # Training and evaluation
cvModel_lsvc = crossval_lsvc.fit(training)
evaluator_lsvc.evaluate(cvModel_lsvc.transform(test))
```

▶ Spark Job Progress

```
Exception in thread cell_monitor-44:
Traceback (most recent call last):
  File "/mnt/notebook-env/lib/python3.7/threading.py", line 926, in _bootstrap_
    inner
    self.run()
  File "/mnt/notebook-env/lib/python3.7/threading.py", line 870, in run
    self._target(*self._args, **self._kwargs)
  File "/mnt/notebook-env/lib/python3.7/site-packages/awseditorssparkmonitoring
widget-1.0-py3.7.egg/awseditorssparkmonitoringwidget/cellmonitor.py", line 178,
  in cell_monitor
    job_binned_stages[job_id][stage_id] = all_stages[stage_id]
KeyError: 1299
```

0.978021978021978

```
In [45]: # Evaluator
evaluator_lsvc = MulticlassClassificationEvaluator(metricName = "f1")
```

```
In [46]: # Cross Validation
crossval_lsvc = CrossValidator(estimator = pipeline_lsvc, estimatorParamMaps = pa
                                evaluator = evaluator_lsvc, numFolds = 3, seed=10)
```

```
In [47]: # Training and evaluation
cvModel_lsvc = crossval_lsvc.fit(training)
evaluator_lsvc.evaluate(cvModel_lsvc.transform(test))
```

▶ Spark Job Progress

```
Exception in thread cell_monitor-47:
Traceback (most recent call last):
  File "/mnt/notebook-env/lib/python3.7/threading.py", line 926, in _bootstrap_inner
    self.run()
  File "/mnt/notebook-env/lib/python3.7/threading.py", line 870, in run
    self._target(*self._args, **self._kwargs)
  File "/mnt/notebook-env/lib/python3.7/site-packages/awseditorssparkmonitoringwidget-1.0-py3.7.egg/awseditorssparkmonitoringwidget/cellmonitor.py", line 178, in cell_monitor
    job_binned_stages[job_id][stage_id] = all_stages[stage_id]
KeyError: 2699
```

0.9778985331439274

4.4. Random Forest Classifier

```
In [48]: # Model Creation
rf = RandomForestClassifier()
```

```
In [49]: # Creating Pipeline
#pipeline_rf = Pipeline(stages = [assembler, rf])
pipeline_rf = Pipeline(stages = [assembler, scaler, rf])
```

```
In [50]: # Parameter
paramgrid_rf = ParamGridBuilder().build()
```

```
In [51]: # Evaluator
evaluator_rf = MulticlassClassificationEvaluator(metricName = "accuracy")
```

```
In [52]: # Cross Validation
crossval_rf = CrossValidator(estimator = pipeline_rf, estimatorParamMaps = paramg
evaluator = evaluator_rf, numFolds = 3, seed=10)
```

```
In [53]: # Training and evaluation
cvModel_rf = crossval_rf.fit(training)
evaluator_rf.evaluate(cvModel_rf.transform(test))
```

▶ Spark Job Progress

0.9945054945054945

```
In [54]: # Evaluator
evaluator_rf = MulticlassClassificationEvaluator(metricName = "f1")
```

```
In [55]: # Cross Validation
crossval_rf = CrossValidator(estimator = pipeline_rf, estimatorParamMaps = paramMaps,
                             evaluator = evaluator_rf, numFolds = 3, seed=10)
```

```
In [56]: # Training and evaluation
cvModel_rf = crossval_rf.fit(training)
evaluator_rf.evaluate(cvModel_rf.transform(test))
```

▶ Spark Job Progress

```
Exception in thread cell_monitor-53:
Traceback (most recent call last):
  File "/mnt/notebook-env/lib/python3.7/threading.py", line 926, in _bootstrap_inner
    self.run()
  File "/mnt/notebook-env/lib/python3.7/threading.py", line 870, in run
    self._target(*self._args, **self._kwargs)
  File "/mnt/notebook-env/lib/python3.7/site-packages/awseditorssparkmonitoring/widget-1.0-py3.7.egg/awseditorssparkmonitoringwidget/cellmonitor.py", line 178, in cell_monitor
    job_binned_stages[job_id][stage_id] = all_stages[stage_id]
KeyError: 3793
```

0.9944902653065121

4.5. Gradient Boosted Tree Classifier

```
In [57]: # Model Creation
gbt = GBTCClassifier()
```

```
In [58]: # Creating Pipeline
#pipeline_gbt = Pipeline(stages = [assembler, gbt])
pipeline_gbt = Pipeline(stages = [assembler, scaler, gbt])
```

```
In [59]: # Parameter
paramgrid_gbt = ParamGridBuilder().build()
```

```
In [60]: # Evaluator
evaluator_gbt = MulticlassClassificationEvaluator(metricName = "accuracy")
```

```
In [61]: # Cross Validation
crossval_gbt = CrossValidator(estimator = pipeline_gbt, estimatorParamMaps = paramgrid_gbt,
                               evaluator = evaluator_gbt, numFolds = 3, seed=10)
```

```
In [62]: # Training and evaluation
cvModel_gbt = crossval_gbt.fit(training)
evaluator_gbt.evaluate(cvModel_gbt.transform(test))
```

▶ Spark Job Progress

```
Exception in thread cell_monitor-56:
Traceback (most recent call last):
  File "/mnt/notebook-env/lib/python3.7/threading.py", line 926, in _bootstrap_inner
    self.run()
  File "/mnt/notebook-env/lib/python3.7/threading.py", line 870, in run
    self._target(*self._args, **self._kwargs)
  File "/mnt/notebook-env/lib/python3.7/site-packages/awseditorssparkmonitoring/widget-1.0-py3.7.egg/awseditorssparkmonitoringwidget/cellmonitor.py", line 178, in cell_monitor
    job_binned_stages[job_id][stage_id] = all_stages[stage_id]
KeyError: 3937
```

0.9905808477237049

```
In [63]: # Evaluator
evaluator_gbt = MulticlassClassificationEvaluator(metricName = "f1")
```

```
In [64]: # Cross Validation
crossval_gbt = CrossValidator(estimator = pipeline_gbt, estimatorParamMaps = paramgrid_gbt,
                               evaluator = evaluator_gbt, numFolds = 3, seed=10)
```

In [65]: *# Training and evaluation*

```
cvModel_gbt = crossval_gbt.fit(training)
evaluator_gbt.evaluate(cvModel_gbt.transform(test))
```

▶ Spark Job Progress

0.9905808477237049

In [66]: *# spark.sparkContext.stop()*

```
spark.stop()
```

▶ Spark Job Progress

4.6. Tempo de Execução

Tempo de execução	Servers		Dif.
	01 node (*)	05 nodes (*)	
Preparation	01:25,2	00:40,9	00:44,3
Logistic Regression	00:45,3	00:37,2	00:08,1
Naive Bayes	00:28,0	00:17,1	00:10,9
Linear Support Vector Machine	02:32,2	02:13,4	00:18,8
Random Forest Classifier	00:38,7	00:35,3	00:03,4
Gradient Boosted Tree Classifier	01:42,8	01:32,7	00:10,1

(*) node = m5.xlarge (4 vCore, 16 GiB memory)

4.7. Consolidado dos Resultados

			Normalizer		Standard Scaler	
	Accuracy	F1	Accuracy	F1	Accuracy	F1
Logistic Regression	0,7575	0,6537	0,9788	0,9787	0,7575	0,6537
Naive Bayes	0,9184	0,9185	0,7763	0,6955	0,9576	0,9568
Linear Support Vector Machine	0,9780	0,9779	0,9804	0,9803	0,9780	0,9779
Random Forest Classifier	0,9945	0,9945	0,9780	0,9780	0,9945	0,9945
Gradient Boosted Tree Classifier	0,9906	0,9906	0,9843	0,9844	0,9906	0,9906

(*) Normalizer is a Transformer which transforms a dataset of Vector rows, normalizing each Vector to have unit norm. It takes parameter p, which specifies the p-norm used for normalization. (p=2 by default.) This normalization can help standardize your input data and improve the behavior of learning algorithms.

(*) StandardScaler transforms a dataset of Vector rows, normalizing each feature to have unit standard deviation and/or zero mean. It takes parameters:

withStd: True by default. Scales the data to unit standard deviation.

withMean: False by default. Centers the data with mean before scaling. It will build a dense output, so take care when applying to sparse input.

5. Referência

Página do Professor Paulo Cortez

<http://www3.dsi.uminho.pt/pcortez/Home.html>

Página de download da base de dados (dataset winequality.zip)

<http://www3.dsi.uminho.pt/pcortez/wine/>

Spark Apache

<https://spark.apache.org/>

Apache Spark

<https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-spark.html>

Create a cluster with Spark

<https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-spark-launch.html>

Apache Hive

<https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hive.html>

Apache Livy

<https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-livy.html>

Amazon EMR Notebook based on Jupyter Notebook

<https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-jupyter-emr-managed-notebooks.html>

Spark 3.2.0 - ML Pipelines

<https://spark.apache.org/docs/latest/ml-guide.html>

Machine Learning with PySpark and Amazon EMR

<https://towardsdatascience.com/machine-learning-with-pyspark-and-amazon-emr-3149dbc847ae>

3. Apresentação

- Cada grupo terá no máximo 15 minutos para apresentar o trabalho.
 - Todos os membros do grupo deverão estar presentes na data de sua apresentação.
 - Todos os grupos deverão estar presentes nos dias de apresentação.
 - A banca de avaliação, formada pelos professores das disciplinas, poderá arguir o grupo ou diretamente qualquer membro do grupo.
-

- A banca é soberana e responsável por resolver os casos previstos nestas orientações e definir os casos omissos.
- As datas das apresentações serão definidas durante o semestre juntamente com professores e alunos de todas as disciplinas.

4. Entregáveis

O que será entregue:

- Relatório (pode ser o notebook ou um github bem detalhado e organizado)
- Apresentação (slides)
- Todos os demais artefatos produzidos (*scripts, architecture workflows, jobs*)

5. Critérios de Avaliação

Por ser um projeto interdisciplinar, a banca definirá, em comum acordo, a nota obedecendo-se a proporção de 40% referente à nota geral do projeto, valorada para cada aluno (20% apresentação oral, 10% slides e 10% arguição) e 60% a cargo de cada professor em sua disciplina.