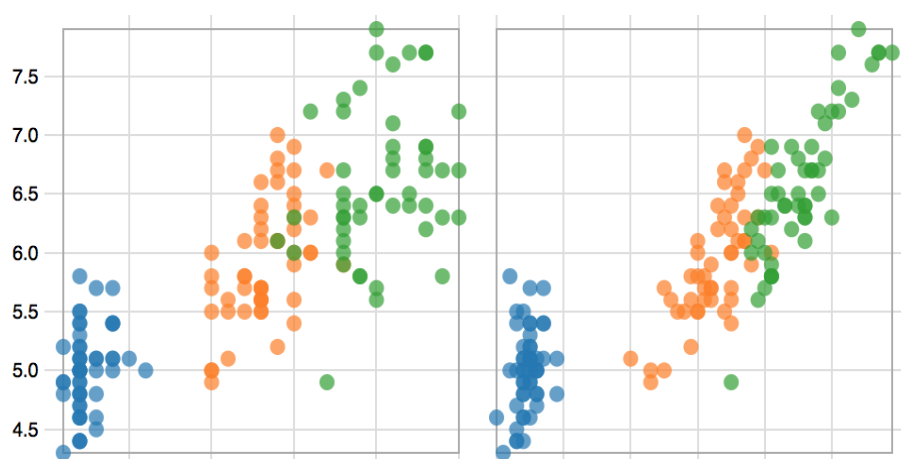




## INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores



### **Análise de soluções para *Big Data Mining***

**JOÃO MIGUEL DA CONCEIÇÃO JUNCEIRA**

(Licenciado)

Dissertação para obtenção do Grau de Mestre  
em Engenharia Informática e de Computadores

Orientadores : Doutor Nuno Miguel Soares Datia  
Doutora Matilde Pós-de-Mina Pato

Júri:

Presidente: Mestre Vitor Jesus Sousa de Almeida

Vogais: Doutor Artur Jorge Ferreira  
Doutora Matilde Pós-de-Mina Pato

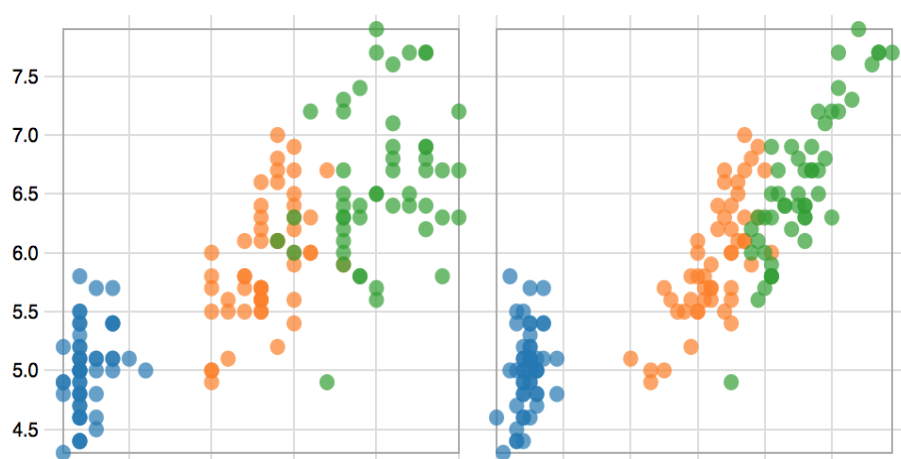
Dezembro, 2017





**INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA**

**Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores**



**Análise de soluções para *Big Data Mining***

**JOÃO MIGUEL DA CONCEIÇÃO JUNCEIRA**

(Licenciado)

Dissertação para obtenção do Grau de Mestre  
em Engenharia Informática e de Computadores

Orientadores : Doutor Nuno Miguel Soares Datia  
Doutora Matilde Pós-de-Mina Pato

Júri:

Presidente: Mestre Vitor Jesus Sousa de Almeida

Vogais: Doutor Artur Jorge Ferreira  
Doutora Matilde Pós-de-Mina Pato

**Dezembro, 2017**



*“A necessidade é a mãe da invenção.”*

*Platão*



# Agradecimentos

Aos meus orientadores por todo o apoio e dedicação ao longo deste trabalho e ao ISEL pela excelência do ensino.

Agradeço à minha família por tudo...





# Resumo

Minerar dados não é uma tarefa trivial. Exige o conhecimento de diversos conceitos e técnicas que não estão acessíveis para a maioria dos utilizadores. O crescente volume de dados e a maior consciencialização do valor que estes podem ter para as organizações levou a um maior número de pessoas a ter de os analisar.

Nos últimos anos, surgiram ferramentas de mineração em larga escala com uma curva de aprendizagem maior face às suas congéneres clássicas. Usam armazenamento e processamento distribuído para lidar com a dimensão dos dados, o que trouxe novos problemas na implementação de soluções de mineração de dados.

Este trabalho aborda o estado da arte das plataformas existentes e a sua evolução histórica. Usando como base uma plataforma de mineração em larga escala (*Apache Spark*) e uma plataforma clássica (R), elaborou-se o desenho de uma solução de classificação e regressão, independentemente das plataformas utilizadas.

Concretizou-se essa solução para cada plataforma para perceber o ponto a partir do qual o desempenho das duas mais se afasta, indicando um possível retorno no investimento na aprendizagem das novas plataformas.

Para o conjunto de dados usado, concluiu-se que o *Apache Spark* é mais vantajoso quando o número de instâncias atinge as 50 mil.

**Palavras-chave:** *Big Data*, Mineração de dados, Classificação, Regressão, *Apache Spark*, R



# Abstract

Mining data is a difficult task that requires knowledge of several concepts and techniques that are not available to the majority of users.

Over the last years, Big Data Mining tools rose to prominence. However, they have a bigger learning curve, as compared with classic tools.

Distributed storage and processing brought new problems in the implementation of Data mining solutions. The state of the art of existing platforms and their historical evolution was carried out. A solution and architecture independent of platforms and a description of the differences of implementation are detailed.

Testing and benchmarking of performance making a comparison between large scale data mining platform Apache Spark and it's classic counterpart R making clear to the user which scenario brings more advantages to the new platforms. Apache Spark has a clear advantage when tested with a dataset over 50 thousand instances.

**Keywords:** Big Data, Data Mining, Classification, Regression, Apache Spark, R



# Índice

<b>Lista de Figuras</b>	<b>xv</b>
<b>Lista de Tabelas</b>	<b>xvii</b>
<b>Lista de Listagens</b>	<b>xix</b>
<b>1 Introdução</b>	<b>1</b>
1.1 Enquadramento . . . . .	1
1.2 Modelos de processo de mineração de dados . . . . .	3
1.3 Objectivos . . . . .	7
1.4 Organização do documento . . . . .	8
<b>2 Ferramentas existentes</b>	<b>9</b>
2.1 Sistemas de armazenamento . . . . .	10
2.1.1 <i>Google BigTable</i> . . . . .	10
2.1.2 <i>Apache Hadoop</i> . . . . .	13
2.1.3 <i>Amazon Dynamo</i> . . . . .	14
2.1.4 <i>Apache Cassandra</i> . . . . .	14
2.2 Bases de dados e <i>datawarehouse</i> . . . . .	15
2.2.1 Bases de dados . . . . .	15
2.2.2 <i>Datawarehouse</i> . . . . .	16

2.3	Processamento e mineração de dados . . . . .	17
2.3.1	Ferramentas de construção de processos de processamento dados . . . . .	20
2.4	Comparação de plataformas . . . . .	21
<b>3</b>	<b>Solução de mineração</b>	<b>25</b>
3.1	Análise dos dados . . . . .	26
3.2	Processo de mineração de dados . . . . .	29
3.3	Implementação de processo de mineração de dados . . . . .	32
3.3.1	Módulo de mineração em <i>Apache Spark</i> . . . . .	34
3.3.2	Módulo de mineração em R . . . . .	37
<b>4</b>	<b>Avaliação experimental</b>	<b>41</b>
4.1	Testes dos modelos de mineração . . . . .	42
4.2	Testes de desempenho . . . . .	43
4.2.1	Resultados de Testes de desempenho . . . . .	45
4.2.2	Comparação de desempenho das plataformas . . . . .	48
<b>5</b>	<b>Conclusões</b>	<b>51</b>
5.1	Síntese e discussão de resultados . . . . .	51
5.2	Principais contribuições . . . . .	52
5.3	Trabalho futuro . . . . .	53
	<b>Referências</b>	<b>55</b>

# Lista de Figuras

1.1	Diagrama de CRISP-DM . . . . .	4
2.1	Processo de <i>MapReduce</i> . . . . .	12
2.2	Esquema do ecossistema Hadoop . . . . .	13
2.3	Esquema da <i>stack</i> do <i>Apache Spark</i> . . . . .	19
3.1	Esquema de processo geral de mineração de dados . . . . .	29
3.2	Esquema de solução de mineração . . . . .	34
3.3	Diagrama de arquitectura em <i>Spark</i> . . . . .	35
4.1	Testes de desempenho de <i>Naive Bayes Classifier</i> . . . . .	46
4.2	Testes de desempenho de <i>Random Forest Classifier</i> . . . . .	46
4.3	Testes de desempenho de Regressão <i>Multinomial</i> . . . . .	47
4.4	Testes de desempenho de <i>One vs Rest</i> . . . . .	47
4.5	Média dos testes de desempenho . . . . .	48





# Lista de Tabelas

2.1	Esquema de tabela da <i>BigTable</i> . . . . .	11
2.2	Algoritmos de classificação e regressão no <i>Spark</i> . . . . .	19
2.3	Síntese de sistemas de armazenamento distribuído . . . . .	23
3.1	Características de dados demográficos (10723 instâncias e 14 características) . . . . .	28
3.2	Características de registo de morte . . . . .	28
3.3	Contagem de instâncias por raça . . . . .	30
4.1	Métricas de qualidade - <i>Random Forest classifier</i> . . . . .	43
4.2	Métricas de qualidade - <i>Naive Bayes Classifier</i> . . . . .	43
4.3	Escala de dados replicados . . . . .	44



## Lista de Listagens

3.1	Pré-processamento em <i>Spark</i> : carregar, adicionar, junção e particionamento paralelo . . . . .	36
3.2	Processamento em <i>Spark</i> : paralelização, indexação, conjuntos de teste e treino . . . . .	37
3.3	Pré-processamento em R: carregar, adicionar e juntar conjunto de dados em R . . . . .	38
3.4	Processamento em R: processos de partição e agregação de conjuntos de dados . . . . .	39
4.1	Teste de desempenho em <i>Spark</i> . . . . .	45
4.2	Teste de desempenho em R . . . . .	45





# Introdução

## 1.1 Enquadramento

A mineração de dados surgiu nos anos 80 da necessidade de extrair conhecimento das bases de dados. Durante muito tempo esteve restringida ao sector académico ou a grandes empresas que possuíam vastas bases de dados [18].

Com o advento da Internet e a necessidade de extracção de conhecimento para pesquisas científicas em áreas como a biologia [23], surgiram alguns problemas que de uma forma sintética podem resumir-se aos seguintes pontos [11][31, p.7]:

- Representação de dados – os conjuntos de dados podem ter heterogeneidade de tipo, de semântica e de granularidade;
- Redundância – os dados disponíveis podem conter redundância, sendo necessário aplicar critérios de selecção e filtragem;
- Tempo de vida – os dados podem ter um tempo de vida, isto é, dados antigos podem ser purgados criando um procedimento que descarta os dados antigos do sistema de armazenamento;
- Expansão e escalabilidade – o sistema de análise de dados deve suportar conjuntos de dados actuais e futuros sendo para isto necessário acomodar o aumento do volume de dados;

- Confidencialidade dos dados – devem ser garantidas medidas para proteger dados sensíveis nos processos de mineração. Um dos exemplos mais comuns é a ofuscação de números de cartões de crédito;
- Cooperação – a cooperação interdisciplinar entre indivíduos de diversas áreas é fundamental para extrair todo o potencial do conjunto de dados. Sem esta cooperação não é correcto entender o que se pretende minerar, ou seja, o domínio do problema.

Os novos desafios foram levantados no início do Século XXI quando o mercado de análise de mineração de dados estava essencialmente concentrado em plataformas clássicas de fraca escalabilidade e maioritariamente proprietárias [29], mas fáceis de manipular sem ter conhecimentos de engenharia de *software* ou de ciências de dados.

Com esta alteração de paradigma, os desafios de minerar grandes volumes de dados que eram um problema essencialmente de investigadores, estão agora a ser colocados à generalidade dos analistas, mostrando as limitações das ferramentas clássicas.

Com a utilização da Internet como infra-estrutura essencial no crescimento económico das empresas, as empresas digitais primeiro, e depois as restantes, começaram a ter acesso a dados em grande volume. Alguns são gerados directamente pelos meios produtivos das empresas; outros são complementares e podem ser obtidos de fontes públicas e/ou privadas através da Internet [26]. Este volume de dados e a necessidade de os explorar para gerar valor deu origem ao conceito *Big Data Mining*. Embora o termo seja abstracto, a sua definição entende-se como grandes quantidades de dados que não podem ser interpretados, adquiridos, geridos e processados por ferramentas de software tradicionais dentro de um tempo tolerável [11, p. 173]. Podemos caracterizar estes problemas de mineração de dados em larga escala através da definição dos V [11, 18]:

- Volume, a dimensão dos dados (e.g. quantidade de dados armazenados);
- Velocidade, rapidez na geração dos dados (e.g. quantidade de dados gerada por dia);
- Variedade, os dados podem vir de diversas origens e em vários formatos;
- Visualização, os gráficos gerados para representar um determinado conjunto de dados devem ser representativos e claros;

- Veracidade, entender se os dados são confiáveis (e.g. os dados podem não ser válidos devido a um erro de introdução na base de dados);
- Valor, que a organização consegue extrair do processo de mineração (e.g. averiguar se o processo de mineração ajuda uma empresa a melhorar as suas margens de lucro face ao custo de implementação da solução).

Para resolver este novo problema, surgiram plataformas de mineração de dados em larga escala, que tiram partido de paralelismo e distribuição [29]. Contudo, a sua manipulação não é uma tarefa trivial, porque exige conceitos e técnicas que não estão acessíveis à maioria dos utilizadores.

## 1.2 Modelos de processo de mineração de dados

Embora o conceito de mineração de dados em larga escala seja novo, os *standards* de processos de mineração já existem desde a década de 90 quando foram tomadas iniciativas de formalização, nomeadamente na identificação de etapas e na relação entre estas. As especificações mais conhecidas são o CRISP-DM (*Cross-industry standard process for data mining*) [9], SEMMA (*Sample, Explore, Modify, Model, Assess*)<sup>1</sup>, KDD (*Knowledge discovery in databases*) [13, 19] e ASUM-DM (*Analytics Solutions Unified Method for Data Mining/Predictive Analytics*) [21, 25].

A especificação CRISP-DM é de todas a mais utilizada<sup>2</sup>, e por isso neste trabalho adoptou-se este procedimento na formalização do processo. A metodologia CRISP-DM foi concebida em 1996 por um consórcio de empresas [9]. O processo de mineração de dados divide-se em 6 fases (ver Figura 1.1):

1. Pesquisa e estudo (*Business Understanding*) – Definir os objectivos do projecto referindo quais os pontos chave que devem ser determinantes para o sucesso, isto é, o conhecimento que se pretende obter. Tendo em conta os riscos inerentes, como por exemplo, determinar se existe informação na fonte de dados para extrair o objectivo pretendido ou se isso está de acordo com as políticas de privacidade da empresa.

---

<sup>1</sup><https://web.archive.org/web/20120308165638/http://www.sas.com/offices/europe/uk/technologies/analytics/datamining/miner/semma.html/> - última consulta a 1 de Novembro de 2017

<sup>2</sup><https://www.kdnuggets.com/2014/10/crisp-dm-top-methodology-analytics-data-mining-data-science-projects.html> - consultado a 1 de Novembro de 2017

Após esta tarefa, deve ser criada uma estratégia preliminar para atingir esses objectivos, que deve ser detalhada em todos os passos. Em último lugar, devem ser estabelecidas métricas de qualidade que permitem avaliar se o modelo criado produziu valor para a empresa (e.g. “Ocorreu aumento das vendas?”).

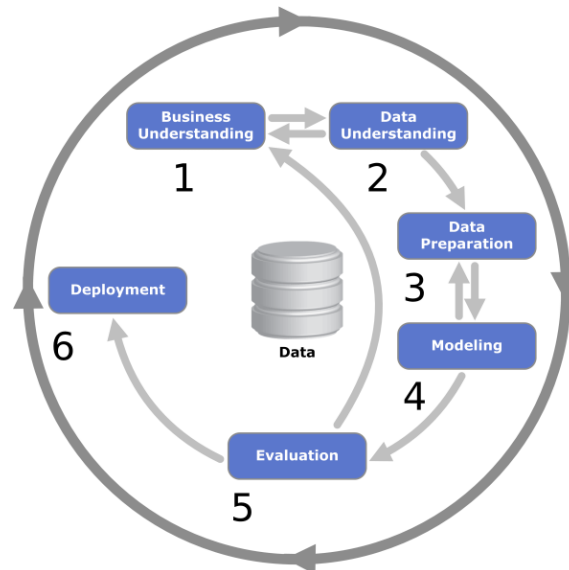


Figura 1.1: Diagrama de CRISP-DM<sup>3</sup>

2. Análise exploratória dos dados (*Data Understanding*) - Numa primeira abordagem deve ser criado um relatório com os locais onde a informação está localizada e as ferramentas necessárias para a sua obtenção, assim como os problemas encontrados. Posteriormente, descrevem-se os dados encontrados como o número de instâncias e características de cada um dos conjuntos de dados. A exploração dos conjuntos de dados feita através de pesquisas e visualização, deve ter em conta se existem subconjuntos mais adequados ou padrões que possam ser identificados como relações entre características, resultados de agregações simples ou até mesmo a aplicação de métodos estatísticos simples (e.g. média) que possam fornecer conhecimento.

Na última fase desta etapa, deve ser feita a verificação de qualidade dos dados para garantir que o resultado final do modelo de processamento não está enviesado por dados incompletos ou erros. A ocorrência destes factores no conjunto de dados deve ser avaliada para determinar a sua frequência.

<sup>3</sup>Fonte: <https://commons.wikimedia.org/w/index.php?curid=24930610> Por Kenneth Jensen CC BY-SA 3.0



3. *Preparação dos dados (Data Preparation)* – Selecção de características (*feature selection*) e redução do tamanho do conjunto de dados para análise, definindo quais os dados que devem ser incluídos e excluídos e justificar a sua decisão. Posteriormente fazer a transformação dos dados, removendo dados irrelevantes ou errados e fazendo a sua substituição utilizando um valor definido através de uma estimativa ou estabelecido no planeamento inicial. Criação de características derivadas, por exemplo, com base em duas características pode ser gerada uma terceira que será representativa de uma intersecção das características e transformação de características para um valor que seja mais adequado. Em último lugar os dados devem ser agregados para facilitar a sua manipulação.
4. *Modelação dos dados (Modeling)* – Seleccionar uma técnica de modelação e registar os pressupostos iniciais do modelo (e.g. o modelo não permite características nulas). Posteriormente, criam-se testes para avaliar o modelo de mineração e inicia-se a construção do modelo com a definição de parâmetros ajustados com base nas características do conjunto de dados e na natureza do algoritmo de modelação. O modelo pode necessitar de ajustes para garantir melhores resultados e pode mesmo ser necessário voltar à fase anterior para pré-processar o conjunto de dados novamente. O desempenho dos vários algoritmos também deve ser comparado. No final, devem ser registados os resultados obtidos para posterior avaliação.
5. *Avaliação (Evaluation)* – Verificar se o resultado obtido pelos processos anteriores está de acordo com os objectivos iniciais e garantir que cumpre métricas de qualidade. Comparar o modelo mais adequado para os objectivos definidos inicialmente. Caso o resultado não seja satisfatório, todo o processo deve ser revisto e os pontos onde existem deficiências devem ser realçados para reajustar e repetir. Determinar os passos seguintes e todas as consequências possíveis que esses passos trazem, assim como as decisões que devem ser tomadas nesse âmbito.
6. *Deployment* – Se na fase de avaliação os resultados estão dentro dos parâmetros desejados, então deve-se proceder ao processo de *deployment*, isto é, disponibilizar o *pipeline* de mineração em ambiente de produção para os utilizadores finais.

Numa primeira etapa desta fase deve ser feito um plano que identifique se o modelo desenvolvido faz sentido para o negócio. Caso faça, devem ser definidos todos os passos para a sua implementação. Posteriormente,

deve ser produzido um relatório com todos os detalhes relativos ao modelo e aos requisitos de implementação para garantir que uma operação de manutenção no futuro seja realizada com sucesso. No final deve ser criado um relatório com as lições aprendidas e com todas as fases do projecto. Corresponde a uma fase muito importante uma vez que garante a reutilização da metodologia implementada quando é utilizada a mesma fonte de dados.

Este modelo não tem que ser implementado de forma rígida e pode ser adequado às necessidades do projecto. Existem várias tentativas para actualizar o modelo, sendo uma das mais conhecidas a ASUM-DM [21, 25] criada pela IBM em 2015. Tem o objectivo de trazer algumas mudanças para a especificação com ideias oriundas das metodologias de desenvolvimento ágil, não é uma mudança radical face ao CRISP-DM mas um refinamento.

**SEMMA** Após a criação do CRISP-DM, o *SAS Institute* criou o SEMMA em 2012. Embora esta metodologia seja específica para as ferramentas SAS, tem algumas similaridades com o CRISP-DM sem a parte de *Business Understanding*. De forma sintética, pode descrever-se o processo nos seguintes pontos:

- *Sample* – selecção dos dados que se pretende minerar obtendo uma amostra representativa do problema ou caso o conjunto de dados inicial seja mais pequeno a totalidade dos dados. A especificação recomenda a divisão em conjunto de Teste, Validação e Treino.
- *Explore* – exploração dos dados para encontrar padrões ou anomalias que possam mostrar um claro entendimento do conjunto de dados. Caso não seja possível encontrar padrões, então aplicam-se algoritmos de mineração para tentar encontrar uma relação.
- *Modify* – criação, transformação e selecção de características para modelar o conjunto de dados, agrupando e eliminando características. Pode-se nesta fase encontrar *outliers* e fazer a sua remoção caso seja interessante. O processo de mineração é iterativo. Esta fase pode ser repetida sempre que os resultados não sejam satisfatórios.
- *Model* – criação de um modelo utilizando um ou mais algoritmos de mineração de dados, incluindo ajustes nos parâmetros de configuração dos algoritmos.

- *Assess* – verificação dos resultados obtidos e teste do modelo para visualização dos resultados e das métricas de teste. Se não for satisfatório, deve-se regressar ao passo *Modify*.

O SEMMA, embora não seja utilizado nas plataformas de larga escala, tem um valor histórico.

## 1.3 Objectivos

As ferramentas de *Big Data Mining* são na sua maioria de código aberto e não-triviais na sua utilização. Sendo uma realidade que a sua adopção em alguns cenários é inevitável, existem outras situações onde a utilização é discutível e como tal a utilização de ferramentas tradicionais pode ter um resultado similar dentro de um tempo similar. Dado o elevado número de ferramentas existentes no mercado, é necessário escolher a ferramenta que melhor se adapta às necessidades do problema que terá que ser resolvido.

A dissertação tem como objectivo comparar entre uma plataforma de mineração de dados clássica (R) e uma plataforma de mineração em larga escala (*Apache Spark*) procurando encontrar o ponto sobre o qual a utilização de uma plataforma de mineração de dados em larga escala se torna mais vantajosa face a uma plataforma clássica. Foram feitos testes com vários algoritmos de mineração de dados para determinar o ponto onde a plataforma de *Big Data* é mais vantajosa. O conjunto de dados foi obtido num repositório de ensaios clínicos (ver Capítulo 3 para mais detalhe).

Realizaram-se os seguintes passos na sua elaboração:

1. Estudo de conceitos e especificações de processos de mineração de dados;
2. Estado da arte de sistemas de armazenamento e processamento de dados em larga escala;
3. Desenho de um processo geral de mineração de dados;
4. Desenho e implementação de processo de mineração de dados em plataformas de mineração de dados em larga escala e numa plataforma clássica;
5. Testes das implementações e análise comparativa dos resultados.

## 1.4 Organização do documento

O documento encontra-se organizado em capítulos que estão divididos da seguinte forma:

- **Ferramentas existentes** – aborda os novos conceitos e plataformas que surgiram nos últimos anos para fazer face ao desafio de armazenar e minerar dados de forma paralela e distribuída – Capítulo 2.
- **Solução de mineração** – descreve o repositório de dados e os conjuntos de dados que foram analisados no trabalho. Posteriormente, define as escolhas tecnológicas e arquiteturas tomadas na criação e implementação da solução de armazenamento, mineração e visualização – Capítulo 3.
- **Avaliação experimental** – comparação das soluções implementadas através da análise do desempenho e testes de validação dos resultados obtidos no processo – Capítulo 4.
- **Conclusões** – discussão dos resultados obtidos, a sua contribuição para o conhecimento do problema e trabalho futuro – Capítulo 5.

## Ferramentas existentes

As novas ferramentas de *Big Data* têm uma curva de aprendizagem acrescida face às ferramentas tradicionais [29][11, p. 174] e o modelo de dados é diferente entre cada uma delas. Portanto, antes de partir para a implementação de uma solução é importante averiguar as suas características e se estas são as melhores para a resolução do problema.

O seu surgimento alterou o paradigma que tinha dominado até ao início do Século XXI onde a mineração de dados era feita na maior parte dos casos sobre bases de dados relacionais e começaram a ser usadas em aplicações tão distintas como a detecção de fraudes de cartões de crédito ou dados de ensaios clínicos. Contudo, não são uma panaceia e não substituem os sistemas tradicionais nem resolvem todos os problemas. São um complemento a necessidades que surgem, resultante dos desafios que foram aparecendo [31, p.6].

O capítulo mostra a análise de algumas plataformas de *Big Data* existentes e a sua evolução histórica. Este trabalho foi realizado para suportar uma escolha coerente da plataforma de larga escala que foi utilizada. A abordagem do tema está dividida em quatro secções:

- Sistemas de armazenamento – apresentação de sistemas de armazenamento distribuído e a sua evolução histórica;
- Bases de dados e *datawarehouse* – resumo descritivo de bases de dados e sistemas de *datawarehouse* construídas sobre plataformas de armazenamento

distribuído;

- Processamento e mineração dos dados – descrição de plataformas para criação de um pipeline de processamento de dados e plataformas de mineração;
- Comparação das plataformas – comparação de plataformas de *Big Data*.

## 2.1 Sistemas de armazenamento

O surgimento do motor de busca da *Google* no início do Século XXI que tem como objectivo indexar todas as páginas *web* existentes na Internet, criou um precedente que não podia ser resolvido com os sistemas de armazenamento que existiam até essa altura [8, 20].

Com o intuito de resolver esses problemas de armazenamento dos índices do motor de busca, a *Google* criou o sistema de ficheiros distribuído *Google File System* [20] e a utilização do conceito de *MapReduce* [14] para obtenção de informação. Os artigos dos projectos da *Google* foram usados como referência na implementação da base de dados não-relacional *Google BigTable* [8], estando agora disponível na *Google Cloud* como *SaaS (Software as a Service)*<sup>1</sup>.

A publicação dos artigos deu origem à criação do *Apache Hadoop (Hadoop)* [5] pela comunidade *open-source* e tornou-se muito popular dando origem a um ecossistema que contém projectos de mineração de dados, *Datawarehouse* ou aprendizagem automática.

A *Amazon* face à sua necessidade de processar encomendas em larga escala criou o *Dynamo* [16], um sistema de armazenamento distribuído, que irá mais tarde inspirar a criação do *Apache Cassandra* [28] pelo *Facebook* para suprimir as suas necessidades de armazenamento de dados gerados em larga escala pelos seus utilizadores.

### 2.1.1 *Google BigTable*

O *BigTable* é um sistema de armazenamento distribuído concebido para armazenar grandes quantidades de dados estruturados espalhados por servidores [8] estando em produção para serviços *Google* como o sistema de busca ou o *Google Maps*, estando também disponível como serviço na *Google Cloud*. A *BigTable* assenta sobre o sistema de ficheiros distribuído *Google File System* [20] e tira partido

<sup>1</sup><https://cloud.google.com/bigtable/> - consultado a 1 de Novembro de 2017

da técnica de *MapReduce* [15] para obtenção de informação. Nesta secção, a análise de cada um destes componentes será feita de forma vertical, ou seja, primeiro será analisado o modelo de dados, seguido pela técnica *MapReduce* e no fim o sistema de ficheiros.

### Modelo de dados

O modelo de dados da *BigTable* é constituído de forma esparsa, distribuída e persistente usando um *sorted map* para armazenar a localização das tabelas. Os dados são organizados e indexados em tripletos constituídos por coluna (*Column*), linha (*Row*) e tempo (*Timestamp*) que constituem uma célula (*Cell*), representado na Tabela 2.1.

Tabela 2.1: Esquema de tabela da *BigTable*

	Família (AD)				Tempo ( <i>Timestamp</i> )
Chave ( <i>Row</i> )	A	B	C	D	
A	1	0	0	0	1511556368
B	0	1	0	0	1509741968
C	0	0	1	0	499898768
D	0	0	0	1	1225745168

A tabela tem uma chave única criada com base em 3 componentes ((*Row:string*, *Column:string*, *Timestamp:int64*) -> *string*) que passo a descrever:

- *Row* – mantém dados ordenados lexicograficamente pela chave que são *strings* arbitrárias com um tamanho até 64 kB.
- *Column* – estão agrupadas por famílias que formam uma unidade de acesso de controlo. Todos os dados são agrupados numa família de colunas do mesmo tipo. A chave da coluna tem a sintaxe família:nome sendo que o controle de acesso ao disco e memória é feito com base na família.
- *Timestamp* – as diferentes células na tabela podem conter múltiplas versões dos mesmos dados que são identificadas pelo *Timestamp* (inteiro de 64-bit).

### MapReduce

A *BigTable* tem como inovação a aplicação do *MapReduce* [15] como modelo de programação na pesquisa nas tabelas, porque o modelo pode ser paralelizado e distribuído facilmente.

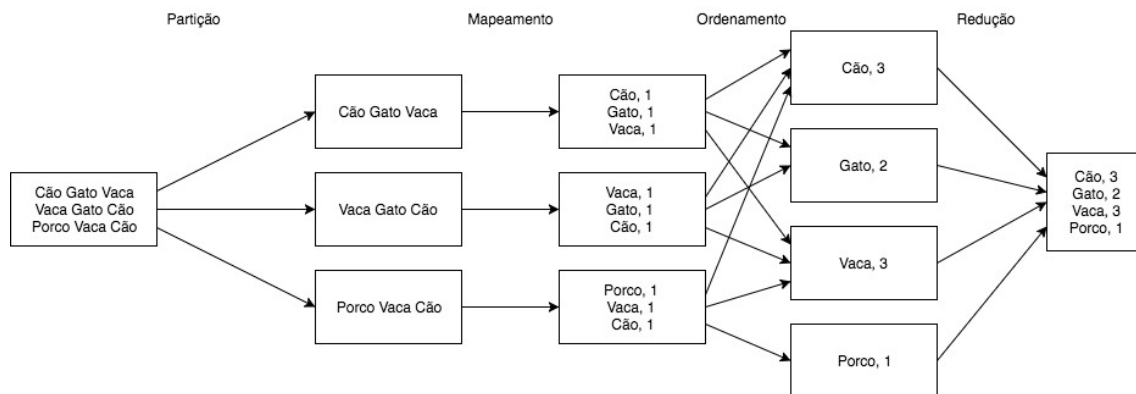


Figura 2.1: Processo de *MapReduce*

A Figura 2.1 mostra os diferentes passos no processo que passo a descrever:

- **Partição dos dados** – o conjunto de dados é particionado em conjuntos de dados mais pequenos que serão processados de forma paralela;
- **Mapeamento** – sobre os conjuntos de dados é aplicada a função de *Map* que faz o processo de mapeamento no qual são seleccionados os valores iguais e criado um peso a cada um;
- **Ordenamento** – neste passo o algoritmo agrupa os elementos similares num único conjunto de dados;
- **Redução** – contabilização dos elementos similares num par chave-valor que contém o elemento e a contagem dos elementos iguais.

As operações podem ser feitas em paralelo em diferentes nós que podem estar distribuídos [43] em máquinas diferentes ou apenas numa única máquina tirando partido das arquitecturas multi-processador.

### Google File System

O armazenamento dos dados está sobre o GFS (*Google File System*) [20] um sistema de ficheiros distribuído criado pela Google para fazer a gestão dos ficheiros armazenados de forma distribuída e tolerante a falhas. As principais características deste sistema são a atomicidade das operações de leitura e escrita assim como a replicação dos dados ao longo dos diversos nós.



### 2.1.2 *Apache Hadoop*

O *Apache Hadoop* nasceu como um projecto inspirado pela publicação dos artigos da Google sobre o *Google File System* e *MapReduce* [37]. O projecto está dividido em três grandes módulos que passo a descrever:

- *MapReduce* – A implementação de uma biblioteca seguindo a descrição do artigo da *Google* (ver secção 2.1.1);
- HDFS (*Hadoop File System*) – sistema de ficheiros distribuído baseado no GFS da Google que providencia tolerância a falhas com recuperação automática e portabilidade através de *hardware* e sistemas operativos diferentes;
- *Hadoop YARN* – ferramenta de criação e gestão de agrupamentos (*clustering*), assim como agendamento de trabalhos, proporcionando uma plataforma capaz de correr em larga escala operações de forma paralela e distribuída.

Para além destes módulos, existe também o *Hadoop Common* contendo um conjunto de bibliotecas comuns a todos os módulos anteriores e por esse motivo não é considerado um módulo separado.

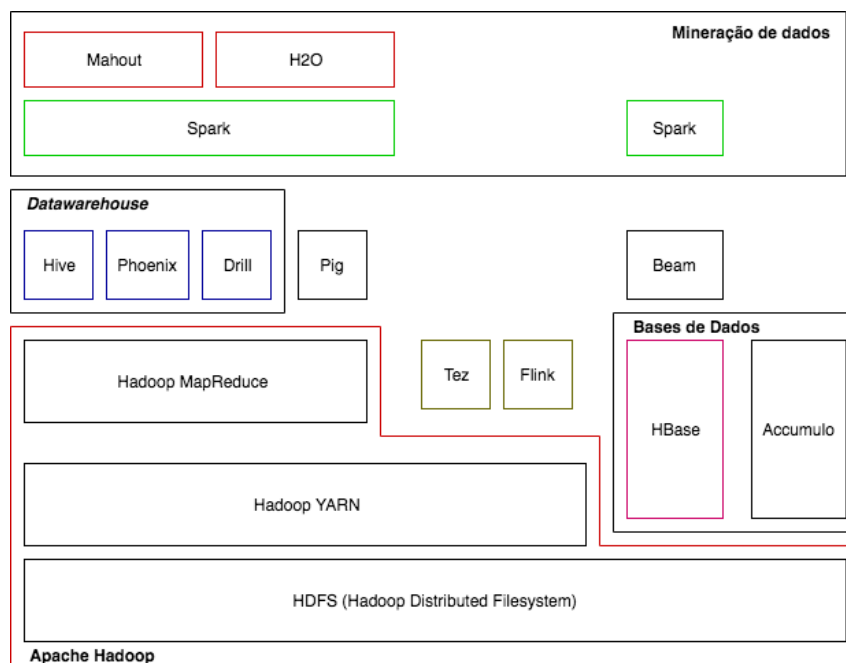


Figura 2.2: Esquema do ecossistema *Hadoop*

O *Hadoop* tornou-se num vasto ecossistema sobre o qual foram construídas várias ferramentas ao longo do tempo. Algumas utilizam todos os módulos enquanto outras apenas tiram partido do HDFS. Na Figura 2.2 [3], apresenta-se uma visão global das ferramentas existentes e dos componentes do *Hadoop* sobre o qual tiram partido. Considera-se a existência de três classes de ferramentas de processamento de dados: *Datawarehouse*, ferramentas de construção de processos de processamento de dados e Bases de dados.

### 2.1.3 Amazon Dynamo

A *Amazon* serve actualmente milhões de clientes e possui uma vasta rede de centros de dados espalhados por todo o mundo. Para fazer face às necessidades de escalabilidade da sua plataforma, construíram o sistema de armazenamento *Dynamo* [16].

O *Dynamo* está construído com base em garantia de partição e replicação consistente utilizando um conjunto de técnicas como o algoritmo *gossip* [17] para detecção distribuída de falhas. O sistema descentralizado permite adicionar e remover nós sem atividade de administração de sistemas como partição e redistribuição. O *Dynamo* é usado em exclusivo dentro da *Amazon* e não possui qualquer funcionalidade de controlo de acessos.

### 2.1.4 Apache Cassandra

O *Apache Cassandra* [28] surgiu no *Facebook* para fazer face às necessidades de armazenamento de conteúdos gerados. Este sistema de armazenamento introduz características que o tornam mais eficiente que o *Hadoop*, porque o CFS (*Cassandra File System*) contrariamente ao HDFS não tem um ponto único de falha graças à sua arquitectura ponto-a-ponto (*peer-to-peer*). Podemos considerar estas características como sendo as mais importantes do sistema de ficheiros:

- Replicação e partição através de vários nós distribuídos em vários locais;
- A arquitectura *peer-to-peer* permite a existência de vários nós a operar de forma independente, conseguindo a ausência de um ponto único de falha. Permitindo a replicação de dados para garantir alta disponibilidade;
- Consistência na escrita pode ser garantida mediante a especificação de um nível de isolamento, isto é, desde garantir que todos os nós estão actualizados com o valor escrito recentemente em detrimento da disponibilidade;

- O nível de isolamento de leitura pode ser especificado no momento da interrogação de uma forma similar ao que é possível fazer nas bases de dados relacionais, ou seja, desde usar um nível similar ao *read uncommitted* que pode gerar *dirty reads* até a um nível de isolamento mais alto.

A linguagem CQL (*Cassandra Query Language*) [10] permite a interrogação de dados presentes na plataforma através de uma linguagem similar ao SQL. Existe também disponível uma API com suporte para várias linguagens como *Java* e *Scala*.

## 2.2 Bases de dados e *datawarehouse*

Esta secção apresenta algumas soluções de bases de dados e *datawarehouse* que foram construídas sobre o *Hadoop*. Cada uma destas soluções tem características diferentes e é importante perceber o que as distingue para fazer a escolha adequada para o problema que se pretende resolver.

### 2.2.1 Bases de dados

O ecossistema *Hadoop* tem bases de dados desenvolvidas com base no artigo da *BigTable* [8]. Foram abordadas as bases de dados *Apache Accumulo* e o *Apache HBase*, embora sejam ambas baseadas em *BigTable* possuem características diferentes.

**Apache HBase** *HBase* é uma base de dados desenvolvida com base no *Google BigTable* e construída sobre o HDFS [3, p. 21]. Embora siga alguns conceitos similares no modelo de dados e arquitectura, tem diferenças na implementação tais como:

- As operações de actualização de dados são feitas em memória e periodicamente gravadas no disco;
- As operações sobre linhas são atómicas com recurso a transacções;
- Contrariamente ao *BigTable* não existe apenas um nó *master*, mas vários para evitar que exista um ponto único de falha;
- As pesquisas são feitas sobre chaves primárias ou com uma pesquisa completa da tabela, porque não existe o conceito de grupo.

**Apache Accumulo** O *Accumulo* [4] é uma base de dados baseada no *Google BigTable* inicialmente criado pela NSA (*National Security Agency*) para um controlo de acesso a dados à linha e coluna (célula), sem que isso seja um problema de desempenho. No desenho original da *BigTable* o controlo de acesso a dados é feito à coluna. Para além dessas funcionalidades de segurança, possui ainda funcionalidades ao nível de desempenho interessantes:

- As linhas (*Rows*) muito grandes não necessitam de ser carregadas inteiramente para memória;
- As chaves são geradas com um algoritmo para permitir a compressão de chaves contíguas;
- Pesquisas demoradas são paralelizadas para aumentar a velocidade;
- Criação de caches para dados lidos recentemente.

### 2.2.2 *Datawarehouse*

As *datawarehouse* tradicionais construídas sobre bases de dados relacionais não estão preparadas para modelar dados provenientes de sistemas de armazenamento novos que possuem dados em larga escala [29].

A Google com a publicação do artigo sobre o sistema *Dremel* [33] proporcionou a criação de novas ferramentas como o *Apache Drill* e o *Google BigQuery* <sup>2</sup> que inicialmente era uma ferramenta interna, mas hoje encontra-se disponível na *Google Cloud*. Pode-se referir que existem algumas plataformas que assumiram um estatuto de referência nas comunidades *open-source*, como:

- *Apache Hive* – Ferramenta de armazenamento de dados de grandes de conjuntos de dados de forma distribuída (*datawarehouse*) sobre *Hadoop* permitindo interação através de SQL [45];
- *Apache Phoenix* – Ferramenta de armazenamento de dados de grandes de conjuntos de dados de forma distribuída (*datawarehouse*) e com suporte para SQL *standard* e JDBC (*Java Database Connectivity*), estando construído sobre OLTP (*Online transaction processing*), ou seja, permitindo a construção desta usando conhecimentos tradicionais [1];

---

<sup>2</sup><https://cloud.google.com/files/BigQueryTechnicalWP.pdf> consultado a 1 de Novembro de 2017

- *Apache Drill* – Disponibiliza uma camada de abstracção que fornece SQL *standard* sobre *Hadoop* e outras bases de dados *NoSQL* sendo baseada nos artigos [22, 33] da *Google*.

## 2.3 Processamento e mineração de dados

A aplicação de algoritmos de aprendizagem automática no *pipeline* de processamento de mineração de dados para processar informações está disponível através de ferramentas como o *Apache Spark*, *Mahout* e *H2O* [3]. Em todas estas ferramentas é também possível estender as suas funcionalidades através das API que possuem e com isso adicionar novos algoritmos para efectuar processamentos. Como foi decidido utilizar o *Spark* como ferramenta na implementação do trabalho a sua descrição tem maior ênfase.

**Ecossistema R** A linguagem R foi criada por John Chambers nos laboratórios *Bell* e é actualmente desenvolvida pelo *R Consortium*<sup>3</sup> sendo amplamente usada em projectos de ciência de dados em indústria e academia. Tem uma especificação muito reduzida em comparação com as linguagem de uso generalista, sendo utilizada por pessoas que não têm conhecimentos de ciências da computação (áreas como a estatística).

O R tem a vantagem de possuir muitas bibliotecas de visualização como o *ggplot2* e o *shiny* para mostrar os dados em páginas HTML (*Hypertext Markup Language*). Existem também bibliotecas para aceder a plataformas de processamento em larga escala como o *Apache Spark*.

Todavia, embora o R tenha os pontos fortes apresentados anteriormente, possui desvantagens que outras linguagens não têm. A gestão de memória do R não é automática como em *Java* e o utilizador está encarregue de remover os objectos em memória, marcando o objecto para remoção. Após esta operação, o *Garbage Collector* tem que ser invocado explicitamente para remover os objectos marcados para remoção<sup>4</sup>. A implementação *standard* é o CRAN implementado como um interpretador da linguagem. Contudo, algumas bibliotecas possuem *bindings* para código implementado em linguagens que compilam para código nativo proporcionando um melhor desempenho do processo de mineração.

<sup>3</sup><https://cran.r-project.org/doc/manuals/r-release/R-lang.pdf> - Especificação da linguagem R - consultado a 1 de Novembro de 2017

<sup>4</sup><http://adv-r.had.co.nz/memory.html> - consultado a 1 de Novembro de 2017

Existem implementações da linguagem R que não são standard, como o *Renjin*<sup>5</sup> e o *FastR*<sup>6</sup> que correm sobre a JVM (*Java Virtual Machine*).

**Apache Spark** O *Apache Spark* [42], vulgarmente designado por *Spark*, é uma ferramenta que oferece uma API com acesso a diversas bibliotecas implementadas de forma distribuída e paralela sobre *Hadoop*.

Contudo, guarda os dados em memória durante o processamento para auxiliar as operações em conjunto com a utilização do HDFS e permitindo uma optimização de desempenho.

A sua implementação foi feita sobre a JVM o que permite tirar partido do JIT (*Just-in-Time Compiler*) [44] que apenas compila o código de linguagem intermédia para código nativo em tempo de execução. Isto garante uma vantagem de desempenho na medida em que o JIT pode fazer optimizações durante a execução do programa.

No centro da *framework* existe o conceito de *DataFrame* que representa um conjunto de dados que pode ser processado de forma implícita utilizando paralelismo e distribuição. Esta garantia de abstracção permite que o utilizador final possa estar concentrado na implementação do modelo de mineração sem necessitar de conhecimentos adicionais.

O conceito de *Pipeline* é tratado no *Spark* através de uma abstracção de alto nível, permitindo criar um fluxo de processamento de dados onde cada uma das fases (*Stages*) efectua uma tarefa de modelação ou pré-processamento. Os conceitos definidos são os seguintes:

- *Parameter* – é uma API abstracta que define o conceito de parâmetro usado posteriormente para configurar cada um dos passos;
- *Estimator* – abstrai o conceito de algoritmo de aprendizagem, treino ou processamento, ou seja, irá ser o responsável pela criação do modelo (*Transformer*);
- *Transformer* – é uma abstracção que contém transformação de características e modelos de aprendizagem, ou seja, faz a transformação de um *DataFrame* noutro *DataFrame* que foi modificado pelo modelo (e.g. Transformar uma coluna aplicando etiquetas ou utilizar um algoritmo de processamento).

<sup>5</sup><http://www.renjin.org> - consultado a 1 de Novembro de 2017

<sup>6</sup><http://www.oracle.com/technetwork/java/jvmls2013vitek-2013524.pdf> - consultado a 1 de Novembro de 2017

Na Figura 2.3 é possível visualizar a estrutura que assenta sobre um sistema de armazenamento e *clustering* e por cima bibliotecas que podem ser subdivididas em quatro categorias:

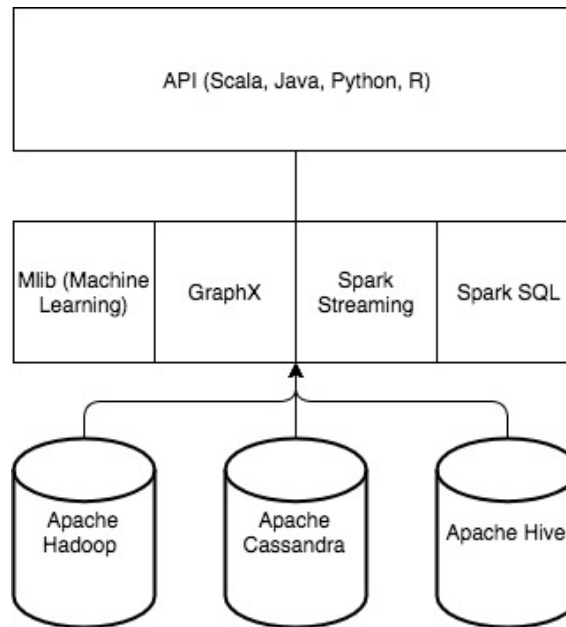


Figura 2.3: Esquema da *stack* do Apache Spark

- *Mlib (Machine Learning library)* – suporta a manipulação do *DataFrame* através da implementação de algoritmos de aprendizagem automática, num *cluster* para agrupamento, filtragem, classificação e regressão (ver Tabela 2.2).

Tabela 2.2: Algoritmos de classificação e regressão no *Spark*

<b>Classificação</b>	<b>Regressão</b>
<i>Binomial logistic regression</i>	<i>Linear regression</i>
<i>Multinomial logistic regression</i>	<i>Generalized linear regression</i>
<i>Decision tree classifier</i>	<i>Decision tree regression</i>
<i>Random forest classifier</i>	<i>Random forest regression</i>
<i>Gradient-boosted tree classifier</i>	<i>Gradient-boosted tree regression</i>
<i>Multilayer perceptron classifier</i>	<i>Survival regression</i>
<i>Linear Support Vector Machine</i>	<i>Isotonic regression</i>
<i>One-vs-Rest classifier</i>	
<i>Naive Bayes</i>	

- *GraphX* – permite a manipulação gráfica e executar operações gráficas em

paralelo. Fornece uma ferramenta uniforme para ETL (*Extract, transform, load*), análise exploratória e cálculos iterativos de grafos além de algoritmos comuns como o *PageRank* [6]. Esta implementação foi baseada num trabalho da *Google* [30].

- *Spark Streaming* – suporta o processamento de transmissão de fluxos de dados em *streaming*, como o *Twitter* [29]. Esta biblioteca recebe os fluxos de dados de entrada e divide-os em lotes (*micro-batching*), que posteriormente os processa pelo mecanismo *Spark* gerando um fluxo final de resultados em lotes.
- *SparkSQL* – contém APIs para manipulação da *DataFrame* utilizando uma sintaxe semelhante ao SQL e pode ser utilizada noutras linguagens como *Scala*, *Java*, *Python* e *R*. Possui suporte para filtros, selecção de dados e agregações.

Para além da integração com o *Hadoop* possui integrações com os sistemas *Apache Cassandra*, *Hive* e *Mesos*. Todas as bibliotecas possuem suporte para *Scala*, *Java*, *Python* e *R*, embora as duas primeiras possuam suporte completo para a API.

**Apache Mahout** *Mahout* é uma biblioteca de algoritmos de mineração de dados e aprendizagem automática assente sobre *Spark*, tem integrações para *H2O* e *Apache Flink* [36].

**H2O** Fornece uma vasta biblioteca de algoritmos de aprendizagem automática que o *Spark* fornece e outros adicionais, assim como uma biblioteca de *deep learning* [12].

Pode-se entender esta ferramenta como uma camada de abstracção sobre o *Apache Spark* fornecendo a aplicação de algoritmos de aprendizagem automática sobre um conjunto de dados ou sobre uma *Datawarehouse* ou dados em *streaming*.

### 2.3.1 Ferramentas de construção de processos de processamento dados

A construção de *pipelines* de processamento directamente sobre *Hadoop* não é trivial, assim como a falta de suporte para *streaming* de dados. No mercado existem



actualmente algumas ferramentas para criar e descrever os fluxos de dados (*pipeline*) como o *Apache Pig*, *Apache Beam*<sup>7</sup>, *Apache Tez* [41] e ferramentas especializadas na criação de *pipelines* em *streaming* como o *Apache Flink* [7].

- *Apache Pig* – Ferramenta que fornece uma linguagem de manipulação de dados (*PigLatin*) sobre o *Apache Hadoop* conferindo abstracção sobre o último [35].
- *Apache Beam* – Ferramenta de construção de *pipelines* de processamento criada pela Google com base no artigo “*The Dataflow Model*” [2]. Permite agregar num só *pipeline* várias ferramentas e linguagens, assim como, usar *MapReduce* do *Hadoop* para processar um conjunto de dados que seguidamente pode ser interrogado com SQL do *Spark* e usar uma ferramenta de aprendizagem automática num único *pipeline* de processamento.
- *Apache Tez* – Ferramenta assenta sobre o *Hadoop YARN* e é uma alternativa ao *MapReduce* para a implementação de processamento em larga escala utilizando uma linguagem para a criação de um fluxo de processamento de dados [41].
- *Apache Flink* – Ferramenta para processamento de dados em *streaming* e *batch* em larga escala e baixa latência permitindo escalar para milhares de nós. Tem manutenção de estado do processamento e tolerância a falhas. As APIs seguem o modelo de programação orientado a eventos para processamento em *streaming* e em *batch*, contém uma biblioteca de manipulação de grafos e aprendizagem automática [7].

## 2.4 Comparação de plataformas

As plataformas de armazenamento de dados em larga escala existentes divergem entre si com tipos de esquemas de dados muito diferentes e com características de segurança, concorrência, persistência e distribuição diferentes. Tendo em conta a diversidade de plataformas é difícil escolher a plataforma para usar num determinado projecto. Consideram-se os seguintes pontos na escolha da plataforma:

- Modelo de dados – terá que ser considerado para escolher algo que se adapte à resolução do problema;

---

<sup>7</sup><https://beam.apache.org/documentation/> - consultado a 1 de Novembro de 2017

- Controlo transaccional – dependendo da implementação da camada aplicacional ou da natureza do negócio, isto pode ser um factor decisivo na escolha da plataforma;
- Segurança – características de segurança como encriptação e controlo de acesso, são decisivos se estiver em causa manipulação de dados confidenciais;
- API – suportar uma linguagem conhecida é uma forma de tornar a curva de aprendizagem mais suave;
- Sistema Operativo – embora exista uma predominância para haver um suporte privilegiado para sistemas baseados em UNIX, também é comum encontrar suporte para *Windows*;
- Licença – existem algumas plataformas que são disponibilizadas como *SaaS*, embora exista uma grande conveniência na medida em que o fornecedor de *cloud* faz a gestão da plataforma, criando uma situação de *vendor lock-in* que pode ser prejudicial para o cliente no futuro.

A Tabela 2.3 faz uma comparação entre algumas plataformas abordadas, neste capítulo onde são comparados os pontos apresentados anteriormente.

Tabela 2.3: Síntese comparativa de sistemas de armazenamento distribuído

	Apache Hadoop	Google BigTable	Apache Cassandra	Amazon Dynamo	Apache Accumulo	Apache Hbase
Armazenamento	HDFS	GFS	CFS	Memória, Disco	HDFS	HDFS
Replicação	Sim	Sim	Sim	Sim	Sim	Sim
Redundância	Sim	Sim	Sim	Sim	Sim	Sim
MapReduce	Sim	Sim	Sim	Sim	Sim	Sim
Controlo transaccional	Não	Não	Sim	Não	Sim	Sim
Controlo de acesso	Não	Não	Não	Não	Sim	Não
Encriptação	Não	Não	Não	Não	Sim	Sim
Persistência	Sim	Sim	Sim	Sim	Sim	Sim
API	Java	REST API, Clientes (Java, Python, etc.)	CQL, Clientes (Java, .NET, etc.)	Clientes (Java, .NET, Node, etc.)	Java	Java
Sistema Operativos suportados	UNIX-based, Windows	SaaS	UNIX-based, Windows	SaaS	UNIX-based, Windows	UNIX-based, Windows
Licença	Apache License 2.0	Google Cloud (Proprietário)	Apache License 2.0	Amazon AWS (Proprietário)	Apache License 2.0	Apache License 2.0



## Solução de mineração

A solução de mineração de dados foi implementada em duas plataformas diferentes para fazer uma comparação de desempenho entre ambas.

Decidiu-se que as plataformas escolhidas para a implementação da solução e comparação são o *Apache Spark* (ver Secção 2.3) construída sobre *Hadoop* [42], com suporte para processamento paralelo e distribuído e o R (ver secção 2.3) por ser uma plataforma clássica sem suporte para processamento paralelo. Por sugestão do orientador escolheu-se uma base de dados de ensaios clínicos para teste deste trabalho. Este capítulo descreve o processo de mineração de dados no qual ficou decidido que iria ser usada a especificação CRISP-DM (ver secção 1.2), tal como a arquitectura da solução e os detalhes de implementação. Está dividido pelos seguintes pontos:

- Análise de dados – Descreve os repositórios e conjunto de dados utilizados no trabalho.
- Processo de mineração de dados – Descrição do processo adoptado no trabalho.
- Implementação de processo de mineração de dados – Descrição dos detalhes gerais de implementação e especificidades de cada plataforma.

## 3.1 Análise dos dados

No sector da saúde graças a iniciativas de recolha de dados de diversas fontes permite obter mais conhecimento sobre as doenças [23, 27]. O conjunto de dados de dados utilizado é fornecido pela plataforma PRO-ACT [39] que contém informações relativas a ensaios clínicos de ELA (Esclerose Lateral Amiotrófica ou ALS na sigla inglesa) [46] realizados por várias farmacêuticas. Neste, estão contidos dados<sup>1</sup> de 10700 registos de pacientes oriundos de 23 ensaios clínicos e mais de 10 milhões de pontos de dados longitudinais. Neste conjunto de dados podemos encontrar informações demográficas dos doentes, historial médico e familiar, e dados recolhidos em laboratório no âmbito de ensaios clínicos com novas drogas e placebo. Entre os doentes, existem registos de diferentes zonas demográficas, podendo haver relações entre os diferentes historiais médicos. Todos estes dados foram fornecidos por farmacêuticas diferentes e recolheram estas informações de forma autónoma sem colaboração entre si.

A plataforma PRO-ACT realizou um processo de fusão entre estas fontes autónomas. No processo de mineração será necessário realizar uma correcta extracção e transformação dos dados, pois existem algumas informações que podem não ser totalmente coerentes entre si.

Neste capítulo existe uma descrição do repositório de dados e dos conjuntos de dados que irão ser sujeitos a análise.

Os dados sofreram uma transformação prévia que não podem ser interpretados ou que não possuíam coerência nas unidades de medida (e.g. *Band Neutrophils* valor 2% e unidade de medida 10E9/L depois de processado valor 2 e unidade de medida %). No repositório encontram-se vários tipos de informações:

- ALSFRS(R) (*Amyotrophic Lateral Sclerosis Functional Rating Scale*) – Medida de avaliação do estado funcional dos pacientes ao longo do tempo, tal como a sua capacidade respiratória e motora.
- Registo de morte – Indica se o paciente faleceu durante o processo de acompanhamento e quanto tempo viveu desde o diagnóstico da doença. O paciente de ELA tipicamente vive entre 3 a 5 anos após o diagnóstico e só 25% sobrevive mais do que 5 anos [32, 46].
- Demografia – Informação demográfica sobre o paciente, tal como a idade no momento do diagnóstico, género, raça, etnia e data de nascimento.

---

<sup>1</sup>informação relativa à data de Dezembro de 2016

- Historial familiar – A ELA afecta cerca de 5 em cada 100.000 pessoas em todo o mundo. Em cerca de 5% dos casos existe um antecedente familiar causado por um defeito genético. No repositório existe um conjunto de dados sobre os familiares afectados e o seu nível de parentesco (eg. pai, mãe, tio).
- Grupo de tratamento – Grupo onde está inserido nos ensaios de ELA.
- Historial de ELA do paciente – Os sintomas manifestados tais como fraqueza muscular, paralisia, movimentos involuntários e problemas de fala.

Existem também outros dados sobre informações laboratoriais relacionadas com testes de drogas como o *Riluzole* ou resultados de análises clínicas e sinais vitais. Este conjunto de dados permite testar os desafios típicos de criação de uma aplicação de mineração em grande escala. Porque tem um cenário de elevada variabilidade, fontes autónomas de dados, relações complexas entre os dados e para entender o domínio do problema é necessário cooperação interdisciplinar.

A documentação relativa ao repositório de dados foi analisada para entender os dados que se encontravam disponíveis e com base nisso foram definidas interrogações que poderiam ser feitas sobre os conjuntos de dados.

**Interrogações** No estudo de doenças como a ELA um dos pontos de pesquisa é a criação de um modelo de previsão da evolução da função neuromuscular após o diagnóstico da doença. Portanto, este trabalho procura saber a esperança média de vida do paciente após o diagnóstico da doença, criando para isso um modelo de classificação. Para gerar o modelo de previsão foram tidos em conta o conjunto de dados de Demografia, Registo de morte e Historial familiar. Existe conhecimento *à priori* de que a doença afecta maioritariamente indivíduos caucasianos [40] do sexo masculino [32].

A Demografia contém informações importantes como a idade, data de nascimento, raça, etnia e género é constituído por 14 características e 10723 instâncias. A Tabela 3.1 faz uma descrição das características e qual a sua relevância. Com a excepção de *Demographics Delta* e *Ethnicity* que não contém informação que seja interessante para a interrogação e *Date\_of\_Birth*, que contém maioritariamente valores em branco.

Tabela 3.1: Características de dados demográficos (10723 instâncias e 14 características)

	<b>Característica</b>	<b>Tipo</b>	<b>Relevante</b>
1	<i>subject_id</i>	Numérico	Sim
2	<i>Demographics_Delta</i>	Numérico	Não
3	<i>Age</i>	Numérico	Sim
4	<i>Date_of_Birth</i>	Numérico	Não
5	<i>Ethnicity</i>	<i>String</i>	Não
6	<i>Race_Americ_Indian_Alaska_Native</i>	Binário	Sim
7	<i>Race_Asian</i>	Binário	Sim
8	<i>Race_Black_African_American</i>	Binário	Sim
9	<i>Race_Hawaiian_Pacific_Islander</i>	Binário	Sim
10	<i>Race_Unknown</i>	Binário	Sim
11	<i>Race_Caucasian</i>	Binário	Sim
12	<i>Race_Other</i>	Binário	Sim
13	<i>Race_Other_Specify</i>	<i>String</i>	Sim
14	<i>Sex</i>	<i>String</i>	Sim

Cada uma das colunas correspondentes à raça possui um valor binário e são exclusivas, ou seja, não é possível ter um paciente de duas raças em simultâneo. A idade (*Age*) está indicada em anos e corresponde à idade no momento em que o paciente foi diagnosticado. O Historial familiar assume especial importância pelo facto da ALS ter origem em alguns casos num defeito genético. Este conjunto de dados contém 1071 instâncias e 39 características. As características contêm o grau de parentesco de familiares portadores de ELA, assim como a indicação se existem doenças neuro-degenerativas na família. Caso o paciente tenha falecido durante o ensaio clínico fica registada a causa da morte.

Tabela 3.2: Características de registo de morte

<b>Característica</b>	<b>Tipo</b>	<b>Relevante</b>
<i>subject_id</i>	Numérico	Sim
<i>Subject_Died</i>	<i>String</i>	Sim
<i>Death_Days</i>	Numérico	Sim

O Registo de morte do paciente é um conjunto de dados que regista as mortes ocorridas em pacientes durante os ensaios clínicos e o número total de dias que



os pacientes viveram desde o diagnóstico da doença até à sua morte. O conjunto de dados tem 3 características (ver Tabela 3.2) e 4634 instâncias.

## 3.2 Processo de mineração de dados

O processo de mineração de dados foi implementado tendo em conta a especificação CRISP-DM (ver secção 1.2), o objectivo do trabalho é testar um modelo similar em duas plataformas diferentes e não existe qualquer intenção de fazer *deployment* da solução em ambiente de produção. Portanto, considera-se que a documentação neste relatório corresponde à fase de *deployment*.



Figura 3.1: Esquema de processo geral de mineração de dados

Na Figura 3.1 observa-se o diagrama do processo geral de mineração adoptado na execução do trabalho e maioritariamente baseado no CRISP-DM (ver secção 1.2). Podemos dividir todos estes procedimentos em 5 etapas principais (Pesquisa e estudo, Análise exploratória, Preparação dos dados, Modelação de dados e Avaliação).

**Pesquisa e estudo** Nesta fase similar ao *Business Understanding* essencialmente o foco foi a análise do repositório de dados e o estudo do domínio do problema. Foram seguidos os seguintes passos:

1. Entender a doença e o significado dos termos técnicos desta área de estudos.
2. Consultar as informações disponíveis no repositório PRO-ACT e descrição dos conjuntos de dados.
3. Pesquisa de artigos sobre estudos feitos anteriormente sobre ELA com este repositório de dados.

**Análise exploratória** Foco nas tarefas seguintes:

1. Perceber as características disponíveis nos conjuntos de dados do repositório e a forma como se relacionam entre si.
2. Interrogações com o objectivo de entender quantos elementos diferentes existem em cada característica e se estas podem ser agrupadas.
3. Entender quais as informações que podem ser extraídas a partir das tarefas anteriores.

Tabela 3.3: Contagem de instâncias por raça

Raças	Contagem
<i>Race_Americ_Indian_Alaska_Native</i>	6
<i>Race_Caucasian</i>	6716
<i>Race_Asian</i>	62
<i>Race_Black_African_American</i>	115
<i>Race_Hawaiian_Pacific_Islander</i>	1
<i>Race_Unknown</i>	14
<i>Race_Other</i>	141

Durante esta fase, extraíram-se as seguintes conclusões. No conjunto de dados Demográficos concluiu-se que existe uma distribuição irregular de instâncias por raça (ver Tabela 3.3). Os pacientes do sexo masculino são 6464 instâncias, feminino são 4253 instâncias e 6 com o valor vazio. No conjunto de dados de Registo de morte sabe-se que existem 3645 pacientes que morreram durante o ensaio (no

atributo *Death\_Days* está indicado o número de dias que sobreviveu) e 989 estavam vivos até ao fim do ensaio clínico. Não existe informação relativa aos restantes pacientes. O objectivo será criar um modelo que faça esta previsão.

No conjunto de dados de Historial familiar é possível ver que existem 518 pacientes com antecedente familiar e 553 sem antecedente familiar.

**Preparação dos dados** Actividade de pré-processamento como junção de conjuntos de dados e selecção de características. Neste ponto pode-se tirar partido de paralelismo para otimizar o processamento. Numa primeira parte fez-se uma visualização dos conjuntos de dados e fez-se uma projecção das colunas fazendo uma selecção das características relevantes para o problema. Após a análise preliminar do conjunto de dados optou-se pela criação de uma coluna que agrupa *Race\_Americ\_Indian\_Alaska\_Native*, *Race\_Hawaiian\_Pacific\_Islander*, *Race\_Unknown* e *Race\_Other* na coluna *Race\_Other\_Unknown* que agrupa todas estas instâncias que estão em reduzido número.

Tendo em conta os dados de antecedentes familiares procedeu-se à criação de uma coluna (*hasFamilyWithAls*) que define se o paciente tem antecedentes familiares, tendo valor binário. Nesta, o conjunto de dados antes de ser modelado é particionado em conjunto de Treino (70%) e Testes (30%).

**Modelação de dados** Os dados são preparados para serem inseridos em diferentes algoritmos de mineração. As características seleccionadas são a Raça, Idade no momento do diagnóstico, Sexo e antecedente familiar de ELA para prever o tempo médio de vida após o diagnóstico da doença. Durante este processo foram encontradas dificuldades, no início houve a ideia de fazer uma regressão para estimar o tempo de vida. Todavia, os resultados não foram satisfatórios, um membro da comunidade médica foi contactado através do orientador, tendo sugerido que a estimativa fosse feita em meses ou trimestres. Portanto a coluna dos dias de vida desde o momento de diagnóstico da doença foi transformado em meses e trimestres. Tendo em conta essa alteração, a tarefa passou a ser uma classificação. Neste processo utilizaram-se os seguintes:

- Classificador *Naive Bayes* – O classificador permite criar um modelo probabilístico, recorrendo a uma técnica de construção de classificadores que cria baseado no princípio de que o valor de uma determinada característica é independente do valor das restantes características. [34].

- Classificador *Random Forest* – As árvores de decisão têm a desvantagem de perda de generalização (*overfitting*). Esta limitação conduz ao enviesamento dos modelos. O *Random Forest* utiliza os princípios de modelação estocástica. Resulta na criação de várias árvores de decisão [24].
- *One vs Rest* – O algoritmo é uma estratégia de classificação que foca o treino num único classificador por classe, as amostras daquela classe serão classificadas como positivas e as outras amostra que não pertencem a essa classe serão classificadas como negativas (classificador binário).
- *Multinomial Regression* – Método de classificação que utiliza a regressão logística para problemas de classificação com múltiplas classes. Este parte do princípio de que cada característica tem um único valor para cada caso.

**Avaliação** Realização de testes de desempenho e avaliação das métricas de qualidade dos modelos desenvolvidos (ver Capítulo 4).

### 3.3 Implementação de processo de mineração de dados

A arquitectura da implementação do processo de mineração foi desenhada para tirar partido do paralelismo durante as fases de preparação e modelação dos dados. Contudo, isso não significa que todos os componentes da arquitectura sejam paralelizáveis, pois existem determinadas tarefas que não podem ser feitas dessa forma como é o caso da aplicação do algoritmo de *Naive Bayes* durante a fase de criação do Modelo. A descrição deste processo de implementação tenta ser genérico em ambas as plataformas para obter uma comparação concreta, embora existam detalhes que foram implementados de forma diferente devido à especificidade de cada plataforma.

Na Figura 3.2 estão enumeradas as diferentes fases de processamento, sendo que alguns processamentos podem ser feitos em paralelo se a plataforma suportar, realiza-se uma comparação com cada uma das fases do processo CRISP-DM. Seguidamente, enumeram-se os processamentos efectuados em cada uma das fases:

1. **Junção de conjunto de dados e selecção de colunas** Os conjuntos de dados (*DeathData*, *demographics* e *FamilyHistory*) foram fundidos fazendo um *left*

*outer join*. Neste processo seleccionam-se as colunas com as características estritamente necessárias para processar (características e etiqueta);

2. **Indexar e Transformar** A visualização do conjunto de dados e observação dos valores diferentes em cada coluna levou a diversas conclusões que permitiram preencher os dados vazios. Dentro da *Race\_Other* existiam outras raças, mas cada uma delas com menos de 10 elementos. Para simplificar o processo de mineração agruparam-se as raças *Race\_Other*, *Race\_Unknown* e *Race\_Hawaiian\_Pacific\_Islander* numa única raça *Race\_Other\_Unknown*.

A coluna *Sex* contém dois valores (*Male*, *Female*) sendo maioritariamente *Male* e nesta fase optou-se por preencher os dados vazios com *Male*, pois era a moda e tal como foi referido anteriormente a doença afecta maioritariamente pacientes do sexo masculino. A coluna *Subject\_died* contém a indicação se o paciente morreu durante os ensaios clínicos, como tal na ausência de valor assume-se que o paciente está vivo.

3. **Particionar por raça - Indexar e transformar** Os resultados foram abaixo do esperado e como tal optou-se por fazer duas transformações:
  - Tendo em conta o excessivo peso da raça caucasiana no conjunto de dados optou-se por fazer a separação em conjuntos de dados diferentes por raça. Este processo de separação permite fazer o processamento em paralelo de cada um dos conjuntos de dados.
  - A coluna *Death\_Days* que contém o número de dias que o paciente viveu desde o diagnóstico da doença, foi transformada em semanas e adicionada ao conjunto de dados, adicionada à coluna do tempo que viveu em meses e também uma coluna do tempo que viveu em trimestres.
4. **Aplicar o Algoritmo** Nesta fase aplica-se o algoritmo com um pré-processamento adicional (e.g. transformar *strings* em valor numérico). Existem algumas alterações que são feitas especificamente para as bibliotecas de cada plataforma.
5. **Aplicar validadores** Detecção de erros e métricas de qualidade do modelo de dados. Os validadores aplicados, estão detalhados no Capítulo 4 (secção 4.1).

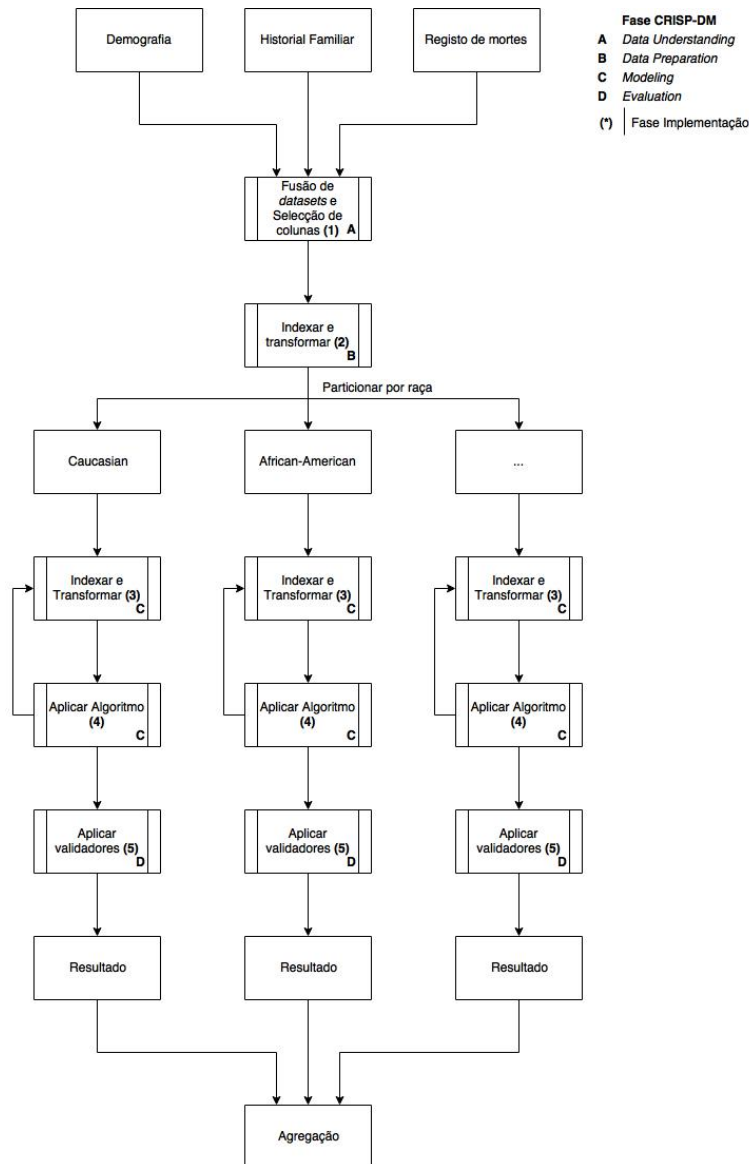


Figura 3.2: Esquema de solução de mineração

### 3.3.1 Módulo de mineração em *Apache Spark*

O Módulo de mineração implementado em *Spark* foi criado em modo *single-node*. A linguagem *Scala* foi escolhida por permitir o acesso à totalidade das APIs do *Spark* e foram utilizadas as bibliotecas *Mllib*, *Spark SQL* e *Spark Core*. O *Hadoop* que está a ser usado é a versão pré-compilada que vem no pacote do *Spark*. Os ficheiros estão a ser guardados em formato *Apache Parquet (Parquet)* porque a dimensão é menor.

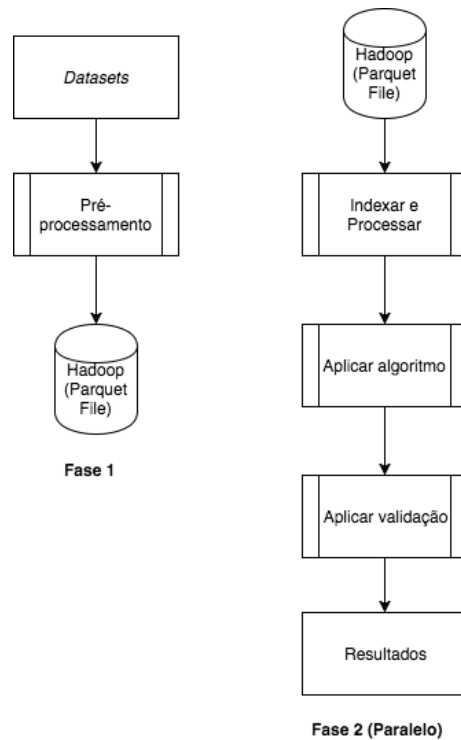


Figura 3.3: Diagrama de arquitectura em *Spark*

**Pré-processamento** Esta fase é comum a todos os modelos. Trata-se de pré-processamento inicial (ver Listagem 3.1). Fizeram-se os seguintes passos:

1. Ler das fontes de dados que contêm as informações relativas a dados demográficos, informações relativas ao tempo de sobrevivência após o diagnóstico da doença e dados sobre a família (Guardar em ficheiros de formato *Parquet*).
2. Extração, substituição e transformação de dados das colunas e preenchimento dos dados em falta nas colunas relativas à raça e sexo (colocando a Moda), número de dias sobrevividos é transformado em semana, mês e trimestre. A idade é transformada em escala de 5 anos.
3. Junção dos 3 conjuntos de dados seleccionando as colunas relativas a Idade, Sexo, Raça, Indicador de antecedente familiar com ELA, informações sobre a morte do paciente.
4. Particionar o conjunto de dados por raça e processar as fases seguintes em paralelo.

```
1 //Carregar Conjuntos de dados para o Spark
2 val demografia = load(demografia_dados)
3 val dados_morte = load(dados_morte_dados)
4 val historial_familiar = load(historial_familiar_dados)
5 //Projectar colunas e transformar colunas de Raça
6 val demografia_transformed = select_demografia(demografia).
7                               transform_columns_race()
8 //Adição da coluna de tempo de vida em meses e de trimestres
9 val dados_morte_transformed = transform_monthly_quarter(dados_morte)
10 //Adicionar coluna de antecedente
11 val historial_familiar_withAncestry = add_column_hasAncestry(
12     historial_familiar)
13 //Join usando o subject_id
14 val merged = join(demografia_transformed, dados_morte_transformed,
15     historial_familiar_withAncestry)
16 //Particionar por raça
17 val particionado_raca = split_race(merged)
18 //Processar paralelamente e obter resultados
19 val aggregated_result = parallel_processing(particionado_raca)
```

Listagem 3.1: Pré-processamento em *Spark*: carregar, adicionar, junção e particionamento paralelo

**Processamento** Foram aplicados vários modelos em separado, isto é, existem *pipelines* distintos para cada um dos algoritmos. Nesta fase ocorre também a substituição de valores vazios de tempo de sobrevivência e idade pela moda e o conjunto de dados é partido em treino e teste numa proporção de 70% e 30% respectivamente. A aplicação do modelo exige ajustes específicos do conjunto de dados para cada algoritmo que passo a descrever:

- **Classificador Naive Bayes** - Indexar as características num único vector e aplicar o classificador;
- **Classificador Random Forest** - Aplicado com 30 árvores (valor ajustado após testes com 10, 20, 25 e 30);
- **One vs Rest** - Usando como parâmetros de configuração 10 iterações;
- **Multinomial Regression** - Usando como parâmetros de configuração 10 iterações, 0.8 *elastic net* e 0.3 parâmetro de regularização.



O código na Listagem 3.2 apresenta o processamento paralelo feito antes da aplicação do algoritmo. Os detalhes da implementação como a Indexação e o preenchimento de dados apenas aplicado para o *subset* correspondente à raça.

```
1  def parallel_processing(datasets) {  
2    parallel_foreach(datasets) {  
3      val (training, test) = split(ds)  
4      val pipeline = Pipeline() {  
5        //Preencher dados vazios  
6        fill_missing_sex(dataset)  
7        fill_missing_family_history(dataset)  
8        //Indexar colunas de características e criar coluna de índice  
9        index_columns(dataset)  
10       //Aplicar algoritmos  
11       ...  
12     }  
13     //Treinar modelo  
14     val model = pipeline.fit(training)  
15     //Aplicar modelo  
16     val predictions = model.transform(test)  
17     //Testar modelo  
18     val result_tests = testing(predictions)  
19   }  
20 }
```

Listagem 3.2: Processamento em *Spark*: paralelização, indexação, conjuntos de teste e treino

### 3.3.2 Módulo de mineração em R

A implementação do processo em R recorreu a um conjunto de bibliotecas disponíveis nos repositórios da implementação CRAN tal como *tidyr*, *dplyr*, *stringr*, *data.table*, *ggplot2*, *e1071* e *randomForest*. Pode-se descrever a implementação em duas partes principais: o pré-processamento e o processamento.

#### Pré-processamento

1. Ler para memória os conjuntos de dados relativos a informação demográfica, tempo de sobrevivência e dados sobre a família;
2. Transformação da história familiar numa única coluna indicativa se possui antecedentes familiares;

3. Transformação da Raça com a adição de uma coluna que indica se pertence às 3 raças com maior população ou se cai na categoria de Outros (raças <20 indivíduos);
4. Preenchimento de dados vazios relativos ao Sexo e a Raça;
5. Fusão dos conjuntos de dados num só e escrita num ficheiro CSV (*Comma-separated values*) em disco.

```
1  #Carregar conjunto de dados para memória
2  death_data <- data.table(read_csv(path_death))
3  demographic_data <- data.table(read_csv(path_demographic))
4  family_history_data <- data.table(read_csv(path_family_history))
5
6  #Adicionar coluna que indica se tem antecedente familiar
7  family_history_data <- add_has_family_with_als(family_history_data)
8
9  #Criar coluna de raça detalhada
10 demographic_data <- add_race_detailed(demographic_data)
11
12 #Adicionar colunas de meses e trimestre
13 death_data <- add_survival_months(death_data)
14 death_data <- add_survival_quarter(death_data)
15
16 #Adicionar valores de sexo as instâncias vazias
17 demographic_data <- fill_sex(demographic_data)
18
19 #Junção de conjuntos de dados
20 merged <- join(demographic_data, family_history_data, death_data)
```

Listagem 3.3: Pré-processamento em R: carregar, adicionar e juntar conjunto de dados em R

A implementação em R apresentada na Listagem 3.3 tem a descrição do código de pré-processamento comum a todos os modelos.

**Processamento** O processamento para cada um destes modelos ocorreu em separado, isto é, não fazem parte do mesmo *pipeline* de processamento. Aplicaram-se os mesmos algoritmos e parâmetros da implementação em *Spark*. A Listagem 3.4 apresenta o processo de particionamento dos conjuntos de dados por raça e posterior processamento. Contrariamente ao *Spark*, não é possível fazer este processo de forma paralela, obrigando a um processo sequencial.

```
1  #Particionar por Raça
2  racas <- split(merged)
3  foreach(i=1:length(races)) %do% {
4      ds <- racas[i]
5      #Indexar as características
6      ds <- index(ds)
7      #Partir em conjunto de dados de teste e treino
8      #Aplicar modelo
9      #Testar
10 }
11 #Agregar resultados
12 resultados_agregados <- aggregated()
```

Listagem 3.4: Processamento em R: processos de partição e agregação de conjuntos de dados



## Avaliação experimental

Como medidas de avaliação de desempenho entre as plataformas utilizadas: *Apache Spark* e R, utilizamos uma replicação do conjunto de dados de tamanho crescente: 2x, 3x e mais do conjunto inicial. Relembramos que a plataforma *Apache Spark* explora intrinsecamente funcionalidades de processamento distribuído e paralelo. A plataforma R por seu lado, não possui um suporte para processamento paralelo e distribuído, excepto se estiver a ser utilizada uma biblioteca de integração com ferramentas de processamento paralelo e distribuído. Neste trabalho, apenas serão utilizadas as bibliotecas que não integram com ferramentas de processamento distribuído.

Neste trabalho apresenta-se um conjunto de métricas obtidas através da realização de testes, neste caso duas classes de teste *CPU Time* e tempo decorrido que pretendem suscitar uma reflexão sobre as vantagens de ambas as plataformas. Posteriormente, verifica-se o ponto no qual uma das plataformas se mostra mais vantajosa em desempenho.

O capítulo encontra-se dividido em duas grandes secções que passo a descrever:

1. Testes dos modelos de mineração – Testes para avaliar a qualidade do modelo de mineração concebido.
2. Testes de desempenho – Pretende conceder uma visão sobre qual o ponto de saturação de cada uma das plataformas.

## 4.1 Testes dos modelos de mineração

Os resultados dos modelos de mineração foram avaliados usando parâmetros de avaliação [38] que passo a descrever:

1. *Accuracy* – corresponde ao números de verdadeiros positivos a dividir pelo total de elementos da amostra;

$$accuracy = \frac{P}{P + N} \quad (4.1)$$

2. *Precision* – Positivos verdadeiros a dividir pelo número de falsos positivos e positivos verdadeiros;

$$precision = \frac{PV}{PV + FP} \quad (4.2)$$

3. *Recall* – Total de positivos verdadeiros a dividir pelo total de positivos verdadeiros somados aos total de falsos negativos;

$$recall = \frac{PV}{PV + FN} \quad (4.3)$$

4. *F1-measure* – Média harmónica entre Precisão e Recall;

$$F1 = \frac{2 * precision * recall}{precision + recall} \quad (4.4)$$

Estes parâmetros foram escolhidos porque são os mais utilizados para averiguar a qualidade dos modelos de classificação [38]. Os testes foram realizados usando a API *MulticlassClassificationEvaluator*<sup>1</sup> do *Spark* e o pacote *MLmetrics*<sup>2</sup> do R para obter as métricas. Os parâmetros foram aplicados sobre o modelo desenvolvido em *Random Forest Classifier* e *Naive Bayes Classifier* (ver secção 3.3).

Os resultados dos testes com *Random Forest* (Tabela 4.1) e *Naive Bayes* (Tabela 4.2) usando o *Apache Spark*, o primeiro classificador (Tabela 4.1) obteve melhores resultados para a característica *Race Black African American*. Em ambos os casos, a implementação mostra que ambos os modelos são bastante similares em resultados e que não existem vantagens na escolha de um modelo face ao outro. Estes resultados foram obtidos através da implementação em *Apache Spark*.

<sup>1</sup><https://spark.apache.org/docs/2.1.2/api/java/index.html?org/apache/spark/ml/evaluation/MulticlassClassificationEvaluator.html> - consultado a 1 de Novembro de 2017

<sup>2</sup><https://cran.r-project.org/web/packages/MLmetrics/index.html> - consultado a 1 de Novembro de 2017

Tabela 4.1: Métricas de qualidade - *Random Forest classifier*

Conjunto de dados	<i>Accuracy</i>	<i>F1-measure</i>	<i>Precision</i>	<i>Recall</i>
Race Asian	0.7917	0.6996	0.6267	0.7916
Race Black African American	0.8611	0.7968	0.7415	0.8611
Race Caucasian	0.7190	0.6015	0.5169	0.7190
Other Unknown	0.7154	0.5967	0.5118	0.7154

Tabela 4.2: Métricas de qualidade - *Naive Bayes Classifier*

Conjunto de dados	<i>Accuracy</i>	<i>F1-measure</i>	<i>Precision</i>	<i>Recall</i>
Race Asian	0.7917	0.6996	0.6267	0.7917
Race Black African American	0.8611	0.7968	0.7415	0.8611
Race Caucasian	0.7190	0.6014	0.5169	0.7190
Other Unknown	0.7145	0.5963	0.5116	0.7145

## 4.2 Testes de desempenho

Os testes de desempenho pretendem aferir o tempo decorrido e *CPU Time* na mesma máquina para cada uma das plataformas e seu impacto. Para despistar eventuais situações em que o sistema operativo possa estar a correr um processo de sistema que consuma muito CPU efectuaram-se 10 iterações. Para analisar se qual dos processamentos teve mais impacto de desempenho vamos ter em conta dois parâmetros de avaliação:

- Tempo decorrido – Tempo decorrido entre o início e o fim do processo.
- *CPU Time* – Entende-se como sendo o tempo gasto pelo núcleo de processamento (*Core*) em segundos a executar as instruções, excluindo assim tempos de espera em operações de I/O. Este parâmetro é particularmente útil para estimar gastos em plataformas de *cloud*, que efectuam cobranças com base no *CPU Time*.

A máquina usada para testes tem as seguintes características:

- Memória RAM – 16GB 1867 MHz DDR3
- CPU – 2,7 GHz Intel Core i5

- Disco – *SSD 251GB*
- Sistema Operativo – *macOS Sierra 10.12.5*

Tabela 4.3: Escala de dados replicados

Escala	Tamanho em instâncias ( <i>Row</i> )
Base	10788
2x	21576
3x	32364
5x	53940
10x	107880
50x	539400
100x	1078800

Tendo em vista a realização de um teste de escalabilidade que verifique qual a plataforma que apresenta maior desempenho fez-se replicação de instâncias, acrescentando dados que já existiam tal como a Tabela 4.3 mostra. Os dados foram replicados, mas a sua distribuição e dimensionalidade foi mantida. Com esta técnica é possível ter uma análise da escalabilidade da plataforma, porque se for utilizado outro conjunto de dados pode-se estar perante uma situação onde o tipo de dados sendo diferente pode ter resultados diferentes embora tenha mais instâncias. Nas secções seguintes do relatório sempre que houver uma referência ao conjunto de dados usado será referido pela sua escala (e.g. 2x, 3x, etc.).

**Implementação dos Testes** Os testes foram implementados em *Spark* e *R* seguindo os seguintes passos:

1. Criação de um fluxo de processamento para cada um dos algoritmos.
2. Iterar 10 vezes sobre esse fluxo de processamento registando o tempo decorrido e *CPU Time* no início e término de cada iteração.
3. Gravar o tempos obtidos num ficheiro CSV.

No *Spark* tal como a Listagem 4.1 mostra, criaram-se testes unitários com *ScalaTest*<sup>3</sup> para medir o tempo decorrido e o *CPU Time*. Contudo, neste teste o tempo de arranque da máquina virtual de *Java* não é contabilizado.

<sup>3</sup><http://www.scalatest.org> - consultado a 1 de Novembro de 2017



```

1  def test() = {
2    for (i <- 1 to 10) {
3      val pBenchMark = ProcessBenchmark.start()
4      //Código de Processamento e Pré-processamento
5      pBenchMark.stop()
6    }
7  }

```

Listagem 4.1: Teste de desempenho em *Spark*

No R os testes contabilizaram o *CPU Time* e Tempo decorrido tal como a Listagem 4.2 mostra. Obtendo o *Timestamp* através da invocação do *proc.time()*<sup>4</sup> no início do processo e no fim, fazendo a subtracção obtém-se o tempo de processo.

```

1  for(i in 1:10){
2    begin <- proc.time()
3    #Código do pipeline de Processamento
4    end <- proc.time()
5    result <- end-begin
6  }

```

Listagem 4.2: Teste de desempenho em R

### 4.2.1 Resultados de Testes de desempenho

Os resultados obtidos pelas medições foram posteriormente transformados para aplicar a média aritmética sobre as 10 iterações. Este passo adicional permite normalizar os resultados diminuindo a influência de eventuais *outliers*.

Os tempos de *CPU Time* foram obtidos a partir da média aritmética das várias iterações feitas. Representando em segundos o tempo que foi gasto em todos os núcleos de CPU para processar o *pipeline*.

**Naive Bayes Classifier** A Figura 4.1(a) mostra que a partir do teste com 10x do tamanho Base o tempo de execução em R torna-se mais acentuado do que em *Spark* demonstrando que a utilização da última ferramenta é mais vantajosa. Embora seja possível verificar o em *Spark* também tem uma subida acentuada.

Na Figura 4.1(b) podemos ver que o tempo de CPU despendido em R ou *Spark* é aproximado. Mas, tal como foi referido o tempo decorrido é menor no *Spark*, isto

<sup>4</sup><https://www.rdocumentation.org/packages/base/versions/3.4.3/topics/proc.time> - consultado a 1 de Novembro de 2017

acontece porque o R não está a explorar o paralelismo e como tal está a demorar mais tempo, mas a gastar o mesmo tempo de CPU.

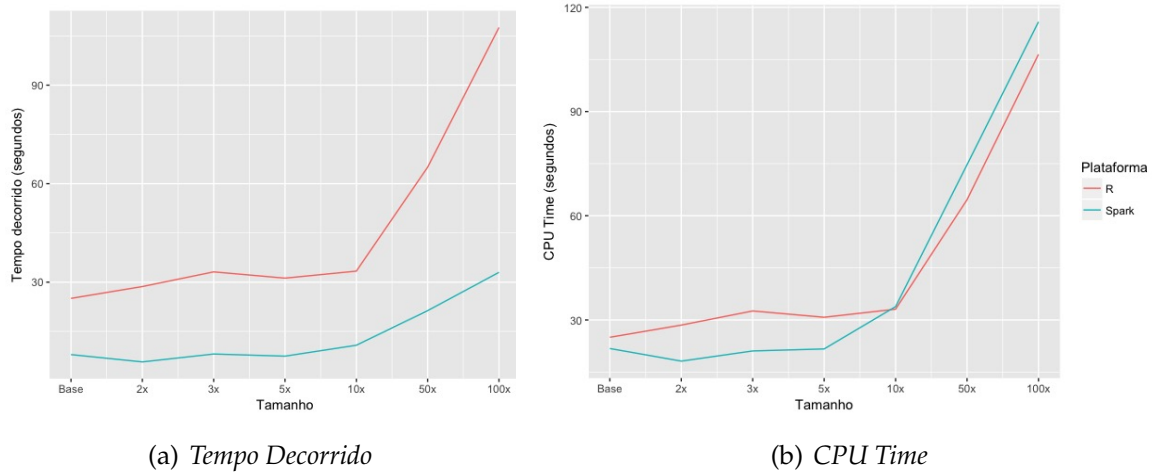


Figura 4.1: Testes de desempenho de *Naive Bayes Classifier*

**Random Forest Classifier** A Figura 4.2(a) mostra um tempo de execução em *Spark* que assume um tempo constante face ao R que assume uma curva acentuada a partir de um conjunto de dados com 10x.

Na Figura 4.2(b) existe clara vantagem na implementação que o *Spark* possui do algoritmo, conseguindo gastar menos tempo de CPU na execução do *pipeline*.

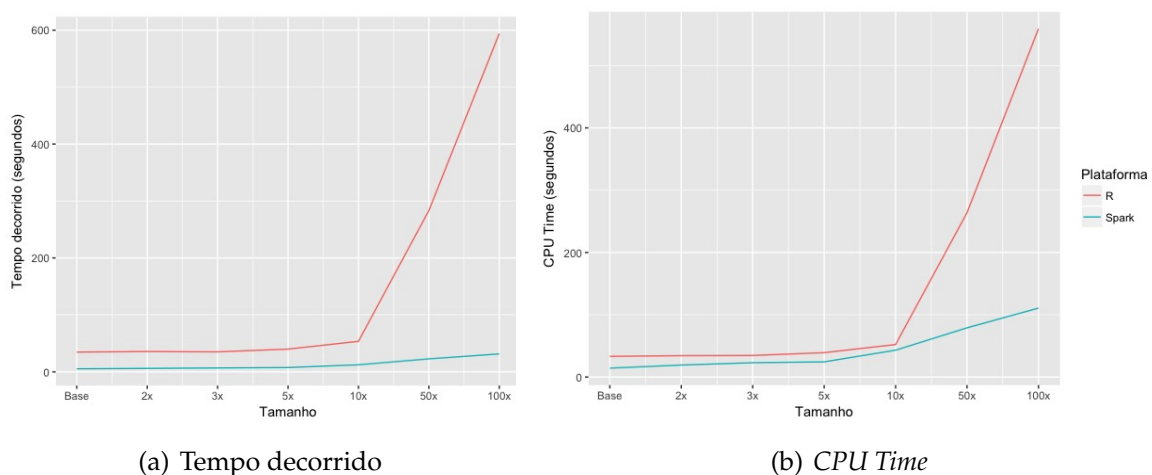


Figura 4.2: Testes de desempenho de *Random Forest Classifier*

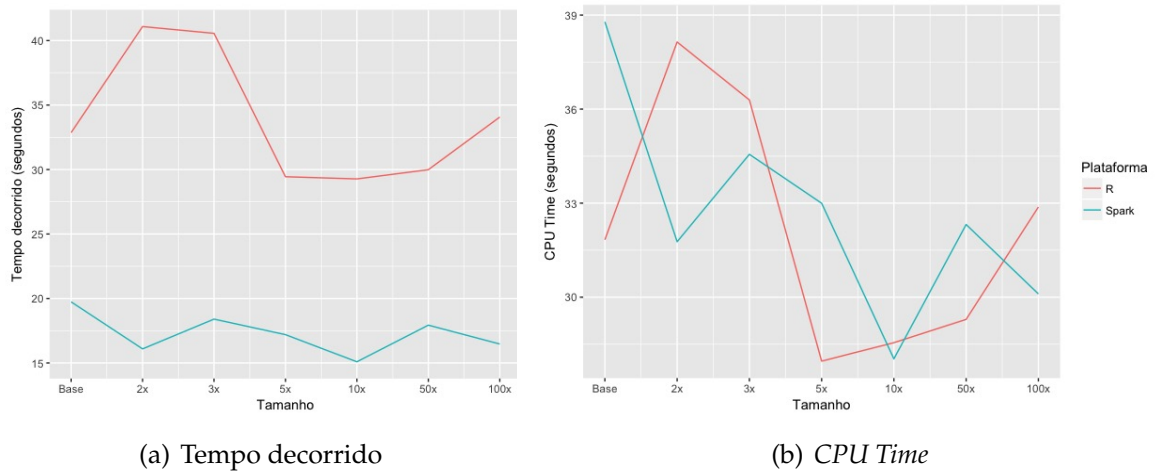


Figura 4.3: Testes de desempenho de Regressão *Multinomial*

**Regressão *Multinomial*** A Figura 4.3(a) mostra-se inconclusiva, pois existem suaves oscilações ao longo do teste em *Spark*. Contudo, em todos os cenários de teste o *Spark* é mais vantajoso do que o R.

Tal como na análise de tempo decorrido, podemos concluir que a Figura 4.3(b) é irregular. O teste deste modelo embora demonstre claramente que o *Spark* é mais rápido a processar o *pipeline* com os diferentes conjuntos de dados, tendo em conta a irregularidade do gráfico pode-se considerar inconclusivo.

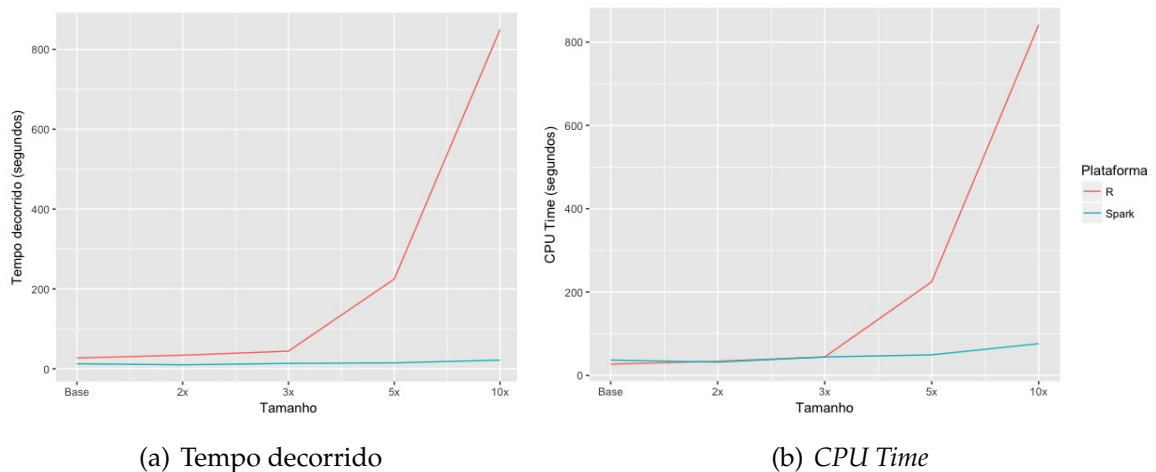


Figura 4.4: Testes de desempenho de *One vs Rest*

***One vs Rest*** A Figura 4.4(a) contrariamente aos testes anteriores mostra que a partir do conjunto de dados 3x do tamanho Base torna-se claramente vantajoso

usar *Spark*. Não foram feitos testes com um conjunto de dados 50x e 100x maior, porque o R deixa de responder em tempo útil (superior a 1 hora).

A Figura 4.4(b) tem uma curva similar ao tempo decorrido. Como tal, tem-se que a implementação em *Spark* consegue processar o mesmo *pipeline* com menos instruções de CPU.

Em todas as figuras anteriores podemos notar que existe um momento no qual o *Spark* se torna obviamente mais vantajoso e que para conjuntos de dados mais pequenos a diferença de desempenho entre ambos é desprezável.

### 4.2.2 Comparação de desempenho das plataformas

Na secção anterior pode-se notar que cada um dos *pipelines* de mineração de dados obteve resultados similares para cada um dos testes e apenas um dos casos se revela inconclusivo.

Portanto, tendo em vista a obtenção de uma possível generalização através da análise de desempenho global das plataformas, fez-se a média aritmética dos tempos obtidos em cada um dos testes.

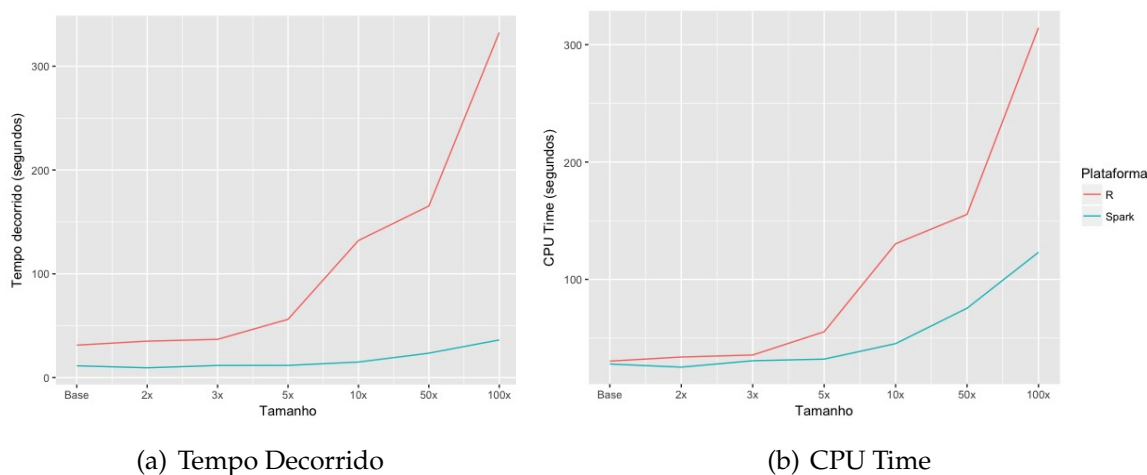


Figura 4.5: Média dos testes de desempenho

Na Figura 4.5(a) podemos ver que o R tem uma curva de crescimento acentuada, enquanto que o *Spark* mantém-se aproximadamente constante. O paralelismo explica a vantagem do *Spark*, mas para além desse factor devemos destacar o facto da linguagem R ser interpretada em oposição ao *Spark* que foi escrito em *Scala*

correndo sobre a plataforma JVM que possui um JIT<sup>5</sup>, onde o código é compilado. Contudo, algumas bibliotecas do R estão escritas em FORTRAN ou C e posteriormente são compiladas para código nativo permitindo algumas vantagens de desempenho mas isso não é relevante, porque não foram construídas para tirar partido do paralelismo. Pese embora o facto de a biblioteca *data.table* suportar processamento paralelo através da API *OpenMP*<sup>6</sup>.

Na Figura 4.5(b) coincide com o cenário anterior onde existiu mais actividade de CPU na plataforma R, enquanto o *Spark* funciona num misto de dados em disco sobre *Hadoop* e em memória. Pode-se considerar que o R é mais vantajoso, por ser mais *user-friendly*, até 5x (53940 instâncias) maior do que o conjunto de dados inicial (10788 instâncias).

---

<sup>5</sup><http://www.oracle.com/technetwork/java/whitepaper-135217.html> - consultado a 1 de Outubro de 2017

<sup>6</sup><http://www.openmp.org/specifications/> - consultado a 1 de Outubro de 2017



## Conclusões

O capítulo faz uma síntese do trabalho, uma reflexão sobre os resultados obtidos e as contribuições da dissertação. Na última parte faz uma descrição do trabalho futuro.

### 5.1 Síntese e discussão de resultados

As plataformas de mineração de dados em larga escala são utilizadas com grande frequência na actualidade, pois os desafios anteriormente colocados a investigadores estão agora a ser colocados à esmagadora maioria dos analistas. Nos últimos anos surgiram diversas plataformas entre as quais o *Google BigTable* que deu origem a um conjunto de artigos no qual os projectos *open-source Apache Hadoop* e *Apache HBase* foram inspirados. Outras empresas como a *Amazon* ou o *Facebook* também criaram plataformas para resolver os problemas internos. As plataformas têm em comum o suporte para processamento paralelo e distribuído que as plataformas clássicas não possuem. As plataformas proliferam e actualmente torna-se difícil escolher a plataforma adequada para o desafio.

Torna-se pertinente para um analista decidir se deve utilizar uma plataforma clássica no qual já possui conhecimento ou pelo contrário optar por uma plataforma de mineração de dados em larga escala, introduzindo maior complexidade no processo. O trabalho comparou a plataforma de mineração em larga escala – *Apache Spark* e uma plataforma clássica – R.

Os conjuntos de dados de teste estão disponíveis num repositório criado no âmbito de ensaios clínicos de fármacos para tratar a ELA. Os dados foram posteriormente analisados e criou-se um processo de mineração que permite saber o tempo que um paciente irá viver desde o momento em que é diagnosticada a doença. Para responder à interrogação anterior foram criados modelos utilizando os algoritmos como *Naive Bayes*, *Random Forest*, *One vs Rest* e *Multinomial Regression*. Os testes de desempenho procuram avaliar quando se torna mais vantajoso utilizar o *Spark*. Para fazer essa avaliação usaram-se duas métricas: o tempo decorrido e o *CPU Time*. Mantendo a dimensionalidade e distribuição dos dados, efectuou-se uma repetição de instâncias de um conjunto de dados Base, usando os múltiplos 2x, 3x, 5x, 10x, 50x e 100x.

Os testes demonstram que a plataforma *Spark* demonstra ser mais vantajosa com qualquer conjunto de dados a partir de 5x o tamanho Base (50 mil instâncias). A análise é dificilmente generalizável, pois o desempenho de um algoritmo varia consoante o conjunto de dados. Contudo, será justo afirmar que para um conjunto de dados similar, isto é com os mesmos tipos de dados, dimensão e algoritmos de mineração os resultados poderão ser similares.

Contudo, olhando para o trabalho será possível concluir que a plataforma *Spark* é claramente mais vantajosa, embora possua uma curva de aprendizagem maior. Possui integrações com a linguagem R e neste caso torna-se possível utilizar os dois em simultâneo.

## 5.2 Principais contribuições

O trabalho contribui para uma melhor compreensão da temática de *Big Data Mining*. Transmitindo uma análise clara do momento a partir do qual se torna mais vantajoso utilizar o *Apache Spark* face ao R. O estado da arte das plataformas de mineração de dados está em constante evolução e conseguir fazer uma escolha de uma plataforma de mineração de dados não é trivial, este trabalho contribui para um melhor entendimento deste processo.

O conjunto de dados foi usado para teste de desempenho de vários algoritmos e análise desses resultados. Confere uma visão das potencialidades das duas plataformas abordadas para os algoritmos analisados e para um conjunto de dados que seja similar ao que foi analisado.



### 5.3 Trabalho futuro

O trabalho procurou cobrir diversos tópicos suscitados pelo objectivo inicial. Contudo, esta é uma área que é muito diversificada e está em constante evolução. Existe trabalho que poderia ser desenvolvido futuramente como:

- Repetir os testes em plataformas de *cloud computing* tais como a *Amazon AWS*, *Microsoft Azure* ou *Google Cloud*;
- Fazer testes no R com a implementação da *Microsoft* ou com o *Renjin*<sup>1</sup> um interpretador de R que funciona sobre JVM e afirma trazer ganhos de desempenho;
- Fazer testes com algoritmos diferentes dos que foram testados ampliando assim o *micro-benchmarking* realizado anteriormente.
- Testar outras plataformas de mineração em larga escala como o H2O ou *Apache Mahout* e outras que foram referidas no capítulo de estado da arte (Capítulo 2).
- Fazer testes com outros conjuntos de dados.

---

<sup>1</sup><http://www.renjin.org/>



## Referências

- [1] Shakil Akhtar and Ravi Magham. Using Phoenix. In *Pro Apache Phoenix*, pages 15–35. Springer, 2017. (p. 16)
- [2] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, et al. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, 8(12):1792–1803, 2015. (p. 21)
- [3] Sherif Sakr Albert Y. Zomaya. *Handbook of Big Data Technologies*. Springer, 1 edition, 2017. ISBN 978-3-319-49340-4. (pp. 14, 15, e 17)
- [4] Apache Accumulo, 2017. URL [https://accumulo.apache.org/1.8/accumulo\\_user\\_manual.html](https://accumulo.apache.org/1.8/accumulo_user_manual.html). (p. 16)
- [5] Apache Hadoop Project. What is Apache Hadoop, 2017. URL <http://hadoop.apache.org>. (p. 10)
- [6] Catherine Benincasa, Adena Calden, Emily Hanlon, Matthew Kindzerske, Kody Law, Eddery Lam, John Rhoades, Ishani Roy, Michael Satz, Eric Valentine, et al. Page Rank algorithm. *Department of Mathematics and Statics, University of Massachusetts, Amherst, Research*, 2006. (p. 20)
- [7] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache Flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015. (p. 21)

- [8] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008. (pp. 10 e 15)
- [9] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rüdiger Wirth. CRISP-DM 1.0. *CRISP-DM Consortium*, 76, 2000. (p. 3)
- [10] Artem Chebotko, Andrey Kashlev, and Shiyong Lu. A Big Data modeling methodology for Apache Cassandra. In *Big Data (BigData Congress), 2015 IEEE International Congress on*, pages 238–245. IEEE, 2015. (p. 15)
- [11] Min Chen, Shiwen Mao, and Yunhao Liu. Big Data: A survey. *Mobile Networks and Applications*, 19(2):171–209, 2014. ISSN 1572-8153. doi: 10.1007/s11036-013-0489-0. URL <http://dx.doi.org/10.1007/s11036-013-0489-0>. (pp. 1, 2, e 9)
- [12] D. Cook. *Practical Machine Learning with H2O: Powerful, Scalable Techniques for Deep Learning and AI*. O'Reilly Media, 2016. ISBN 9781491964552. URL <https://books.google.pt/books?id=C5amDQAAQBAJ>. (p. 20)
- [13] Chantel D. Larose Daniel T. Larose. *Discovering Knowledge in data An Introduction to Data Mining*. Wiley Series on Methods and Applications in Data Mining. Wiley, 2 edition, 2014. ISBN 978-0-470-90874-7. (p. 3)
- [14] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. (p. 10)
- [15] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: a flexible data processing tool. *Communications of the ACM*, 53(1):72–77, 2010. (p. 11)
- [16] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Voss, and Werner Vogels. Dynamo: Amazon’s highly available key-value store. *ACM SIGOPS operating systems review*, 41(6):205–220, 2007. (pp. 10 e 14)
- [17] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth Annual*

- ACM Symposium on Principles of Distributed Computing, PODC '87*, pages 1–12, New York, NY, USA, 1987. ACM. ISBN 0-89791-239-X. doi: 10.1145/41840.41841. URL <http://doi.acm.org/10.1145/41840.41841>. (p. 14)
- [18] Wei Fan and Albert Bifet. Mining Big Data: current status, and forecast to the future. *ACM SIGKDD Explorations Newsletter*, 14(2):1–5, 2013. (pp. 1 e 2)
- [19] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From Data Mining to Knowledge Discovery in databases. *AI magazine*, 17(3):37, 1996. (p. 3)
- [20] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003. (pp. 10 e 12)
- [21] Jason Haffar. Have you seen ASUM-DM?, Oct 2015. URL <https://developer.ibm.com/predictiveanalytics/2015/10/16/have-you-seen-asum-dm/>. (pp. 3 e 6)
- [22] Michael Hausenblas and Jacques Nadeau. Apache Drill: interactive ad-hoc analysis at scale. *Big Data*, 1(2):100–104, 2013. (p. 17)
- [23] Matthew Herland, Taghi M Khoshgoftaar, and Randall Wald. A review of Data Mining using Big Data in health informatics. *Journal of Big Data*, 2014. (pp. 1 e 26)
- [24] Tin Kam Ho. Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition*, volume 1, pages 278–282. IEEE, 1995. (p. 32)
- [25] IBM, 2016. URL <ftp://ftp.software.ibm.com/software/data/sw-library/services/ASUM.pdf>. (pp. 3 e 6)
- [26] Xindong Wu Fellow IEEE, Xingquan Zhu, Gong-Qing Wu, and Wei Ding. Data Mining with Big Data. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):97–107, 2014. (p. 2)
- [27] Indu Khatri and Virendra Kumar Shrivastava. *A Survey of Big Data in Healthcare Industry*, pages 245–257. Springer Singapore, Singapore, 2016. ISBN 978-981-10-1023-1. doi: 10.1007/978-981-10-1023-1\_25. URL [http://dx.doi.org/10.1007/978-981-10-1023-1\\_25](http://dx.doi.org/10.1007/978-981-10-1023-1_25). (p. 26)

- [28] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010. (pp. 10 e 14)
- [29] Jimmy Lin and Dmitriy Ryaboy. Scaling Big Data mining infrastructure: The Twitter experience. *SIGKDD Explorations*, 14(2):6–19, 2014. (pp. 2, 3, 9, 16, e 20)
- [30] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010. (p. 20)
- [31] Nathan Marz and James Warren. *Big Data: Principles and best practices of scalable realtime data systems*. Manning, 2015. (pp. 1 e 9)
- [32] Pamela A McCombe and Robert D Henderson. Effects of gender in amyotrophic lateral sclerosis. *Gender medicine*, 7(6):557–570, 2010. (pp. 26 e 27)
- [33] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: Interactive analysis of web-scale datasets. In *Proc. of the 36th Int’l Conf on Very Large Data Bases*, pages 330–339, 2010. URL <http://www.vldb2010.org/accept.htm>. (pp. 16 e 17)
- [34] Kevin P Murphy. Naive Bayes classifiers. *University of British Columbia*, 2006. (p. 31)
- [35] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008. (p. 21)
- [36] Sean Owen and Sean Owen. Mahout in action. 2012. (p. 20)
- [37] Aditya B Patel, Manashvi Birla, and Ushma Nair. Addressing Big Data problem using Hadoop and Map Reduce. In *Engineering (NUICONE), 2012 Nirma University International Conference on*, pages 1–5. IEEE, 2012. (p. 13)
- [38] David Martin Powers. Evaluation: from Precision, Recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011. ISSN 2229-3981. (p. 42)

- [39] PRO-ACT. Pooled resource open-access ALS clinical trials, 2017. URL <https://nctu.partners.org/ProACT/Document/DisplayLatest/5>. (p. 26)
- [40] Lindsay Rechtman, Heather Jordan, Laurie Wagner, D Kevin Horton, and Wendy Kaye. Racial and ethnic differences among amyotrophic lateral sclerosis cases in the United States. *Amyotrophic Lateral Sclerosis and Frontotemporal Degeneration*, 16(1-2):65–71, 2015. (p. 27)
- [41] Bikas Saha, Hitesh Shah, Siddharth Seth, Gopal Vijayaraghavan, Arun Murthy, and Carlo Curino. Apache Tez: A unifying framework for modeling and building data processing applications. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1357–1369. ACM, 2015. (p. 21)
- [42] Abdul Ghaffar Shoro and Tariq Rahim Soomro. Big Data analysis: Apache Spark perspective. *Global Journal of Computer Science and Technology*, 15(1), 2015. (pp. 18 e 25)
- [43] Michael Stonebraker, Daniel Abadi, David J DeWitt, Sam Madden, Erik Paulson, Andrew Pavlo, and Alexander Rasin. Mapreduce and parallel DBMSs: friends or foes? *Communications of the ACM*, 53(1):64–71, 2010. (p. 12)
- [44] Toshio Suganuma, Takeshi Ogasawara, Mikio Takeuchi, Toshiaki Yasue, Motohiro Kawahito, Kazuaki Ishizaki, Hideaki Komatsu, and Toshio Nakatani. Overview of the IBM Java just-in-time compiler. *IBM systems Journal*, 39(1): 175–193, 2000. (p. 18)
- [45] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu, and Raghotham Murthy. Hive-a petabyte scale data warehouse using Hadoop. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 996–1005. IEEE, 2010. (p. 16)
- [46] Sara Zarei, Karen Carr, Luz Reiley, Kelvin Diaz, Orleiquis Guerra, Pablo Fernandez Altamirano, Wilfredo Pagani, Daud Lodin, Gloria Orozco, and Angel Chinaea. A comprehensive review of amyotrophic lateral sclerosis, 2015. URL <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4653353/>. (p. 26)

