Trabalho prático de CDLE

Instituto Superior de Engenharia de Lisboa
Computação de Dados em Larga Escala
Gonçalo Fonseca - A50185
Rúben Santos - A49063
Sofia Condesso- A50309

***Abstract* - Este projeto utiliza o modelo MapReduce, implementado com a biblioteca mrjob em Python, para realizar análises em larga escala de conjuntos de dados textuais no contexto do Hadoop. Três tarefas principais foram desenvolvidas: contagem de palavras, identificação e contagem de bigramas, e cálculo de TF-IDF para destacar a relevância de termos específicos nas revisões. O código modular e o programa principal (main.py) oferecem flexibilidade na execução das tarefas no ambiente Hadoop ou local. O trabalho aborda a problemática do processamento em tempo real, discute vantagens e desvantagens do Hadoop e apresenta trabalhos relacionados na área. O projeto visa proporcionar uma abordagem prática para lidar com desafios associados ao processamento distribuído de grandes conjuntos de dados textuais.**

I. Introdução

Nos dias de hoje, existe um crescimento exponencial nos dados que existem e, com isso, o processamento destes é cada vez mais demorado e complexo, em especial no processamento de dados em tempo real. Para tal, este projeto tem dois grandes objetivos, sendo um deles a análise de grandes conjuntos de dados textuais, explorando a contagem de palavras, estatísticas relevantes e análise de sentimentos, adaptando o uso do *Hadoop* para processamento paralelo em larga escala com grandes *datasets*. Outro dos desafios será integrar o *Python* com o *Hadoop* como linguagem de programação que oferece várias bibliotecas de processamento de texto, explicando a sua implementação e quais as vantagens e desvantagens em relação à aplicação do *Hadoop* realizada durante as aulas da Unidade Curricular.

A. Hadoop

O Hadoop é uma *framework* de software em Java, que se destaca pois utiliza o processamento distribuído de grandes conjuntos de dados em *clusters* de computadores [1]. Sendo uma *framework* robusta, contém uma arquitetura flexível e escalável que oferece uma abordagem eficaz para lidar com os desafios impostos por conjuntos massivos de informações, proporcionando uma base sólida para análise e extração de *insights* [2].

A essência do *Hadoop* reside na integração de três componentes principais: o *Hadoop Distributed File System* (HDFS), o modelo de programação *MapReduce* e o gestor *Yarn*:

- **Hadoop Distributed File System (HDFS)**: Um sistema de arquivos distribuído que divide os dados em blocos e os distribui em diferentes nós do **cluster**. Isso permite o armazenamento eficiente de grandes quantidades de dados em vários servidores;

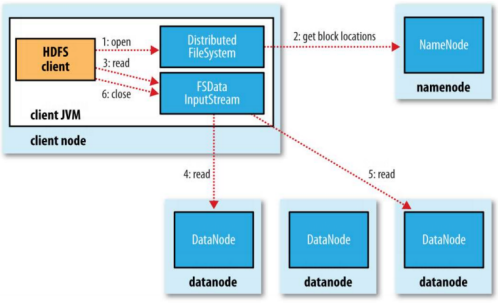

Figura 1: Hadoop - Sistema HDFS [17]

O HDFS funciona num modelo **master-slave** , onde o *NameNode* faz a gestão dos meta-dados, como a localização de blocos, enquanto os *DataNodes* armazenam os próprios dados. Os ficheiros são divididos em blocos replicados em diferentes *DataNodes* para garantir tolerância a falhas. [23]
O HDFS é "**rack-aware**", considerando a topologia física do *cluster*, e utiliza uma arquitetura robusta para leitura e gravação de dados eficientes, uma vez que a sua conceção tem alta disponibilidade e tolerância a falhas, tornando-o ideal para lidar com o processamento distribuído de grandes volumes de dados no contexto de **big data**. [24]

- **MapReduce**: Um modelo de programação e processamento paralelo que divide as tarefas em duas fases principais - a fase de mapeamento (**Map**) e a fase de redução (**Reduce**). Estas fases são executadas em paralelo em vários nós do *cluster*, permitindo o processamento eficiente de grandes conjuntos de dados. [19]
O objetivo do **MapReduce** passa por mapear cada um dos **Jobs** e reduzi-los a tarefas equivalentes para fornecer menos sobrecarga na rede do *cluster* e reduzir o poder de processamento, como podemos ver na arquitetura do **MapReduce** da figura 4:

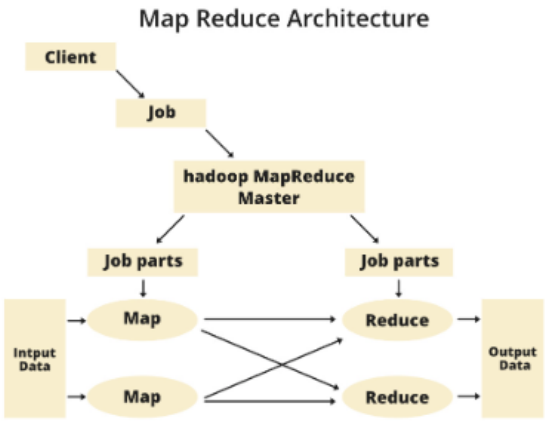

Figura 2: Arquitetura do MapReduce

No modelo **MapReduce** do Hadoop, é submetido um **Job** que é dividido em tarefas pelo **MapReduce Master**, onde o **Job** fica disponível para mapear e reduzir. Os dados de entrada que são utilizados, são então alimentados para a tarefa **Map**, onde este irá construir um par de valores-chave (***key-value***) intermediários para a sua saída, que posteriormente são introduzidos nos *reducers* para executarem operações de redução (ex. consolidar, agregar e agrupar) nos dados intermediários recebidos pela tarefa **Map**. [18]

- **Yarn**: também conhecido como "*Yet Another Resource Negotiator*", é um elemento essencial do Hadoop. [19] Ele atua como uma camada de gestão de recursos, permitindo que diversos mecanismos de processamento de dados, como o **MapReduce**, operem simultaneamente no mesmo *cluster* Hadoop e compartilhem recursos. [20] A capacidade do **YARN** de gerir tarefas e recursos resulta em maior escalabilidade e flexibilidade, permitindo a execução eficiente de uma variedade de tarefas. Desta forma, a camada do **YARN** está localizada entre a camada de armazenamento de dados (**HDFS**) e a camada de processamento de dados (como o **MapReduce**) no ecossistema do *Hadoop* para fazer a gestão de recursos, como podemos ver na figura seguinte:

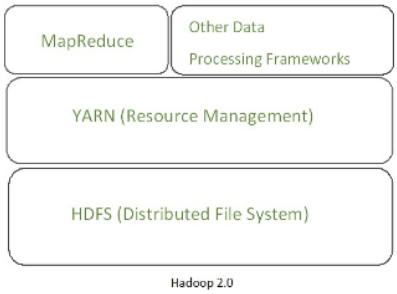

Figura 3: Arquitetura do Yarn [21]

O fluxo de trabalho do **YARN** começa com a submissão de uma aplicação, onde o gestor dos recursos aloca um recipiente para iniciar o gestor da aplicação, que se regista no gestor de recursos e "negocia"recipientes para executar tarefas. O gestor da aplicação notifica o gestor de Nó para lançar os recipientes, onde o código da aplicação é executado. [21] Durante a execução, é possível monitorizar o estado da aplicação. Após a conclusão, o gestor da aplicação faz um "logout"do gestor de recursos. [22]

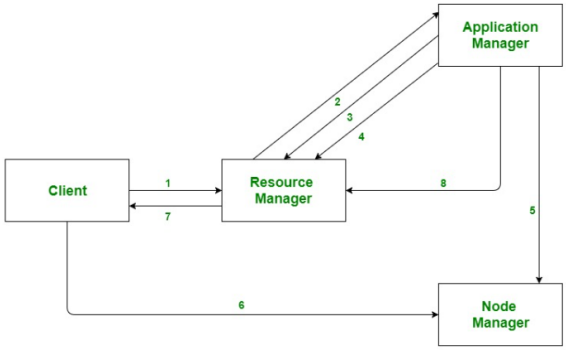

Figura 4: Workflow do Yarn [21]

B. Vantagens e Desvantagens

Ao fragmentar e distribuir os dados, e ao processá-los simultaneamente, o Hadoop supera obstáculos, resultando nas vantagens distintas de escalabilidade horizontal, processamento distribuído, e armazenamento distribuído com tolerância a falhas. Essa abordagem permite que a plataforma atenda eficientemente ao crescimento dos volumes de dados, acelere o tempo de processamento através da execução paralela, e assegure a confiabilidade e a redundância no armazenamento dos dados, mesmo diante de falhas nos sistemas. [3]

Apesar das suas vantagens, o Hadoop ainda apresenta desafios a serem ponderados. A latência elevada para processamento em tempo real é notável devido ao seu modelo de processamento em conjunto, podendo gerar atrasos inadequados. A complexidade da programação em MapReduce pode também ser um entrave para os programadores menos familiarizados. Adicionalmente, as limitações em atualizações em tempo real podem restringir uma aplicação em cenários que requerem informações constantemente atualizadas. [4]

C. Alternativas ao Hadoop

Ao analisar o Hadoop em comparação com tecnologias anteriores, observamos uma evolução marcante no paradigma de processamento e armazenamento de dados. Antes, as tecnologias tradicionais, como os bancos de dados relacionais, eram dominantes. Agora, ao analisar essa evolução, fica evidente que o Hadoop trouxe uma abordagem mais ágil e adaptável para lidar com as demandas crescentes do processamento de grandes conjuntos de dados. [5]

II. Trabalhos Relacionados

Como referido anteriormente, no nosso trabalho usamos *Hadoop* como a tecnologia principal para processamento de texto e análise de sentimentos. Nesta secção, irão ser referidos alguns trabalhos relacionados que usam *Hadoop* e tenham problemas relacionados.

A referência "*Application of HADOOP Technologies on Text Processing*"fornece uma perspetiva prática sobre a aplicação do Hadoop no processamento de texto. Este estudo visa contextualizar a implementação prática do *Hadoop* em cenários semelhantes aos nossos objetivos. [11]

O artigo "*Data-intensive text processing with MapReduce*"discute o processamento intensivo de texto usando o *MapReduce*. Este estudo oferecer abordagens e práticas relevantes, que podem ser usadas em projetos como o nosso, que se baseia no modelo *MapReduce* no *Hadoop* para análise de dados de texto. Ao explorarmos técnicas intensivas de processamento de texto, podendo adaptar e aplicar os modelos desse trabalho no nosso próprio contexto. [10]

O artigo "*H-TFIDF: A scalable implementation of TF-IDF on Hadoop*"destaca uma implementação escalável da técnica *TF-IDF* no *Hadoop*. Dado o nosso interesse em calcular *TF-IDF* para avaliar a importância relativa de palavras, este trabalho pode fornecer *insights* valiosos sobre como implementar eficientemente essa técnica em larga escala. [12] Para além desse artigo, o artigo "*Scalable sentiment analysis on Twitter data using MapReduce on Hadoop*"também dá insights valiosos sobre a análise de sentimentos, neste caso sobre dados do Twitter, através do *MapReduce* no *Hadoop*. [13]

Também, o artigo "*Effective processing of unstructured data using Python in Hadoop MapReduce*"destaca a eficácia da manipulação de dados não estruturados com *Python* no contexto do *Hadoop MapReduce*. Esta abordagem, escolhida pelo estudo, reconhece a flexibilidade do *Python* e aproveita a escalabilidade proporcionada pelo *Hadoop* para lidar eficientemente com grandes volumes de dados não estruturados [9]. Esta metodologia é similar com a nossa própria pesquisa, onde procuramos utilizar o *Hadoop* e *Python* para processar texto e fazer a análise de sentimentos. Para além disso,o trabalho "**Text Processing using Hadoop MapReduce**"de Anunay Rao que, tal como o anterior, demonstra a utilização do Python em conjunto com o Hadoop MapReduce, evidenciando a flexibilidade desta linguagem para análise de dados não estruturados. Este trabalho é relevante para o nosso projeto, que compartilha objetivos semelhantes de processamento eficiente de dados de texto com Hadoop. [14]

III. Definição do problema

O problema que se está a tentar resolver neste projeto é uma análise de um conjunto de documentos para extrair informações estatísticas e sentimentais relevantes. Isto é feito através da contagem de palavras de dimensão n (n-gramas), que são sequências contíguas de n itens de uma amostra de texto. Através desta análise, é possível produzir uma tabela de frequências dos n-gramas, calcular a percentagem de n-gramas que ocorrem uma única vez (também conhecidos como singletons), e aplicar a medida estatística TF-IDF (frequência do termo–inverso da frequência nos documentos) para determinar a importância de cada termo em relação ao conjunto de documentos.

Além disso, o problema também envolve a avaliação da polaridade (positiva, negativa, neutra) do texto nos documentos, um processo conhecido como Análise de Sentimentos. Este processo pode ser realizado através de várias técnicas, como análise léxica, Machine Learning, ou modelos pré-treinados, que são usados para classificar o tom emocional de cada documento. A solução para este problema permitirá uma compreensão mais profunda do conteúdo dos documentos, bem como a identificação de tendências e padrões dentro do conjunto de documentos.

IV. Implementação

Para a implementação deste trabalho, tal como referido inicialmente, irá ser realizada através da linguagem de programação Python, pois é uma linguagem que todos os autores se sentem mais confortáveis, e também é uma implementação diferente daquela apresentada das aulas práticas realizadas em Java.

Inicialmente, a exploração centrou-se na procura de métodos para integrar o Hadoop com Python. Durante essa pesquisa, duas bibliotecas destacaram-se: **pydoop** e **mrjob**. A preferência recaiu sobre o mrjob devido a questões de instalação que surgiram com o pydoop, além de beneficiarmos de uma documentação mais abrangente e acessível [15]. Esse conjunto de escolhas proporcionou uma base sólida para a implementação das análises de texto propostas neste projeto.

A. Teoria

O módulo **mrjob** é uma biblioteca em Python projetada para simplificar o desenvolvimento de trabalhos MapReduce, podendo ser executada em clusters Hadoop. Desenvolvido e mantido pela Yelp, o mrjob abstrai grande parte da complexidade associada ao Hadoop MapReduce, permitindo que os desenvolvedores foquem na lógica de processamento de dados. [16]

O **mrjob** oferece uma interface amigável, permitindo que se definam funções Map e Reduce, usando a sintaxe familiar de Python, tornando-a acessível a um público mais amplo, facilitando a criação de trabalhos MapReduce de forma acessível e eficiente. Além disso, o módulo integra-se sem esforço ao Hadoop, garantindo compatibilidade com diversas distribuições [15].

1) Uso da API: A API do *mrjob* baseia-se na criação de uma classe Python que herda da classe MRJob. Esta classe define as etapas de Map e Reduce, juntamente com qualquer configuração adicional necessária para o programa [15]. O mrjob lida automaticamente com os formatos de entrada e saída, simplificando o processo de desenvolvimento. A função **mapper** processa pares chave-valor como entrada e emite pares chave-valor intermediários. Da mesma forma, a função **reducer** processa esses pares intermediários e produz a saída final [16]. Abaixo um exemplo geral de código.

```
from mrjob.job import MRJob

class MyMRJob(MRJob):
    def mapper(self, _, linha):
        # Lógica da função Map
        pass

    def reducer(self, chave, valores):
        # Lógica da função Reduce
        pass

if __name__ == '__main__':
    MyMRJob.run()
```

Em suma, o módulo *mrjob* proporciona uma abordagem facilitada para o desenvolvimento de trabalhos MapReduce em Python, garantindo compatibilidade com o ecossistema Hadoop.

B. Desenvolvimento

Em relação ao desenvolvimento do código, decidiu-se começar por compreender a lógica do funcionamento do **mrjob** para implementar um **MapReduce** especifico para cada tópico que queremos analisar. No contexto deste projeto, os dados provenientes do conjunto "Amazon Music Reviews"disponíveis em https://www.kaggle.com/datasets/eswarchandt/amazon-music-reviews serão utilizados. Esses dados consistem em avaliações de produtos de música na plataforma Amazon, proporcionando um conjunto diversificado de revisões que abrangem diferentes aspetos dos produtos musicais. Essa fonte de dados rica e variada será fundamental para realizar análises abrangentes de processamento de texto, como contagem de palavras, análise de sentimentos e cálculo da medida estatística TF-IDF.

Para se ter um programa genérico, que permita a escolha do MapReduce pretendido, criou-se um programa ***main.py*** que, como se pode observar nos Anexos na secção VII-A, obtém 3 argumentos iniciais: se a execução é local ou no **Hadoop**, o caminho do ficheiro input e o caminho da pasta de output. De seguida, questiona ao utilizador qual das tarefas é que pretende realizar. O código então verifica se a execução será local ou no **Hadoop**, executando o *job* **MapReduce** correspondente e copiando os resultados de volta para o sistema local ou *HDFS*, garantido a existência de todas as pastas.

Para correr este ficheiro **main.py** apresentado na secção VII-A, é necessário na linha de comandos correr o seguinte comando (assumindo que está na pasta do ficheiro):

```
python3 main.py hadoop_local \
"/path/to/inputfile" "/path/to/outputfolder" \
nrofclusters codec
```

O comando acima é um comando genérico, onde é necessário alterar a variável "hadoop_local"para "hadoop"ou "local", e os caminhos tanto do ficheiro input como na pasta de output. Um exemplo de como correr poderia ser a seguinte:

```
python3 main.py local reviews.json  \
"/home/usermr/examples/output/textanalysis/" \
2 "GzipCodec"
```

No momento da entrega, assume-se que o codec é sempre "GzipCodec"para facilitar a construção, bastando apagar a linha que que coloca sempre esse codec.

Para facilitar também a gestão, assume-se que as pastas incluidas no zip estão dentro da pasta "/home/usermr/examples/Projects/Project", com todas as subpastas e ficheiros main.py e reviews.json nessa mesma pasta. No entanto, este ficheiro json poderá estar noutra localização. Um ficheiro README encontra-se também anexado.

De seguida, iremos falar sobre as várias tarefas desenvolvidas no projeto.

*1) **Word Count e Word Count Json**:* A primeira tarefa do nosso projeto, o **word_count**, exemplifica a aplicação do modelo MapReduce na análise de conjuntos de dados de texto. Inicia-se dividindo o texto em palavras e, em seguida, realiza-se a contagem da frequência de cada palavra. O processo compreende dois passos principais: o mapeamento, que cria pares chave-valor para cada palavra identificada, e a redução, que consolida as contagens associadas a cada palavra. Adicionalmente, o programa incorpora uma etapa de ordenação para apresentar os resultados finais de forma organizada, destacando as palavras mais frequentes no conjunto de dados.

Para completar o compreensão sobre as variantes do mapeamento utilizadas no projeto, é importante verificar que as funções **mapper** nas tarefas **Word Count Json** e **Word Count** apresentam abordagens ligeiramente diferentes. Ambas têm o objetivo de contar a frequência de palavras, mas variam na forma como processam os dados de entrada.

Na tarefa **Word Count** (com o código apresentado nos Anexos da secção VII-B), a função **mapper** recebe um par chave-valor em que o valor é interpretado diretamente como uma string. Aqui, o código utiliza a biblioteca **re** (regex) para encontrar todas as palavras no texto e emite pares chave-valor para cada palavra encontrada.

```
def mapper(self, key, value):
    tokens = re.findall(r"\b\w+\b", value.lower())
    for token in tokens:
        yield token, 1
```

Já na tarefa **Word Count Json** (com o código apresentado nos Anexos da secção VII-C), a função mapper recebe um par chave-valor em que o valor é interpretado como um objeto **JSON**. O código usa a biblioteca json para carregar o valor, obtendo o texto da revisão (review_text). Em seguida, aplica expressões regulares (regex) para *tokenizar* o texto, encontrar todas as palavras e, finalmente, emite pares chave-valor para cada palavra encontrada.

```
def mapper(self, key, value):
    review = json.loads(value)
    review_text = review['reviewText']
    tokens=re.findall(r"\b\w+\b",review_text.lower())
    for token in tokens:
        yield token, 1
```

Na figura 5 abaixo é apresentado o resultado usando o ficheiro "reviews.json".

Figura 5: Output com resultados

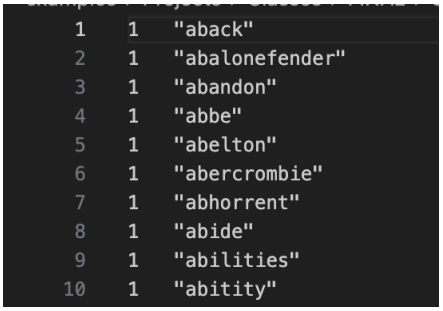

Figura 6: Output com resultados com regex adicional

Para além disso, temos ainda uma contagem similar à anterior, mas que contêm uma expressão regular (regex) que, mesmo antes de aplicar a expressão apresentada anteriormente, remove todos os números e pontuações existentes, para termos um resultado com muito menos palavras aleatórias. O código, apresentado nos Anexos da secção VII-D, usado foi o seguinte.

```
def mapper(self, _, value):
        """Uses regex pattern to find all words in
        the json datatset field 'reviewText'.
          It cleans the text from digits and
          punctuation.
        """
        # loads instead of load to load just
        # a string
        review = json.loads(value)

        # get only the review text
        review_text = review['reviewText']
        # remove digits and punctuation
        review_text = re.sub(r'\d+', '', review_text)
        rgp=r"[!\"#$%&()*+-./:;<=>?@\[\\\]^_`{|}~\n]"
        review_text = re.sub(rgp , '', review_text)
        # get words
        tokens = re.findall(re.compile(r"\b\w+\b"), \\
        review_text.lower())

        for token in tokens:
            yield token, 1
```

Como evidenciado na figura 6 abaixo, em comparação com a figura 5, observamos que neste resultado são apresentadas palavras de forma mais coerente. Por outro lado, no outro output, as palavras são notavelmente mais aleatórias. No entanto, é importante destacar que essa abordagem pode resultar na perda de informações potencialmente cruciais. Este cenário ocorre devido ao facto de que, no exemplo sem a utilização do *regex* adicional, a intenção era separar números de palavras, adicionando espaços entre eles. No entanto, essa operação não estava a ser efetiva dentro do ambiente do mrjob.

*2) **Frequency_Table_Bigrams**:* A segunda tarefa do nosso projeto, denominada **BigramFrequencyTable**, extende a aplicação do modelo MapReduce para analisar grandes conjuntos de dados textuais de uma maneira mais elaborada. Nessa etapa, o foco é na identificação e contagem de frequência de bigramas, que são pares consecutivos de palavras num texto. O processo é dividido em dois passos principais: O mapeamento, onde o texto da revisão é pré-processado, removendo dígitos e pontuações, e os bigramas são identificados a partir das palavras resultantes, resultando assim num conjunto de pares chave-valor para cada bigrama encontrado. As seguintes linhas de código exemplificam a criação efetiva dos pares chave-valor para cada bigrama no conjunto de dados:

```
def mapper(self, _, value):
    # Load
    review = json.loads(value)
    review_text = review['reviewText']

    # Pre-processamento
    review_text = re.sub(r'\d+', '', review_text)
    review_text = re.sub(r"[!\"#$%&()*+-./:;<=>?@\
    [\\\]^_`{|}~\n]" , '', review_text)
    tokens = re.findall(WORD_RE, review_text.lower())

    # Bigrams- Criação da tabela de frequencias
    for i in range(len(tokens)-1):
        yield f"{tokens[i]} {tokens[i+1]}", 1
```

O passo de redução, por sua vez, consolida as contagens associadas a cada bigrama. Adicionalmente, uma etapa de ordenação é incluída para apresentar os resultados finais de forma organizada, destacando os bigramas mais frequentes. Esse processo oferece informações mais detalhados sobre a estrutura linguística e padrões associados aos bigramas presentes nas revisões do conjunto de dados.
A figura 7 apresentada abaixo mostra um resultado possível, neste caso usando o ficheiro reviews.json.

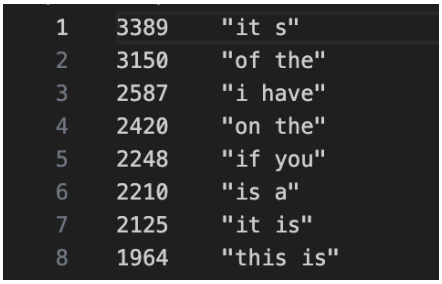

Figura 7: Output com resultados

Todo o código desta tarefa está apresentada nos Anexos da secção VII-E.

*3) **TF-IDF (term frequency–inverse document frequency)**:* A terceira tarefa, denominada **WordTFIDF**, expande a análise do conjunto de dados por meio do modelo MapReduce, focando o cálculo do **Term Frequency-Inverse Document Frequency** (TF-IDF) para cada palavra em cada revisão. O objetivo é destacar termos mais relevantes, proporcionando informações mais profundos sobre a importância de cada termo nas revisões. Durante o passo de mapeamento, o texto de cada revisão é pré-processado, removendo dígitos e pontuações. Em seguida, são gerados pares chave-valor representando o **Term Frequency** (TF) de cada palavra em relação à revisão específica. Essa etapa fundamental, implementada nas seguintes linhas de código seguintes.

```
def mapper_tf(self, _, value):
    review = json.loads(value)
    review_text = review['reviewText']
    review_id = review['reviewerID']

    review_text = re.sub(r'\d+', '', review_text)
    review_text = re.sub(r"[!\"#$%&()*+-./:;<=>?@\
    [\\\]^_`{|}~\n]", '', review_text)
    tokens= re.findall(WORD_RE, review_text.lower())

    for token in tokens:
        yield f"{review_id} {token}", 1/len(tokens)
```

O passo subsequente de redução é dividido em duas fases. A primeira fase prepara os dados para o cálculo do **Inverse Document Frequency** (IDF), enquanto que a segunda fase calcula o **TF-IDF** para cada termo, resultando numa ordenação decrescente de relevância. As linhas a seguir demonstram essa etapa crucial do código:

```
for k, v in sorted(self.terms_tf_idf.items(),
                   key=lambda item: item[1],
                   reverse=True):
    yield k, v
```

Essa abordagem oferece uma visão detalhada sobre a importância relativa de cada termo nas revisões do conjunto de dados, combinando informações sobre a frequência nas revisões individuais (**TF**) e a raridade global (**IDF**).
A figura 8 apresentada abaixo mostra um resultado possível, neste caso usando o ficheiro reviews.json.

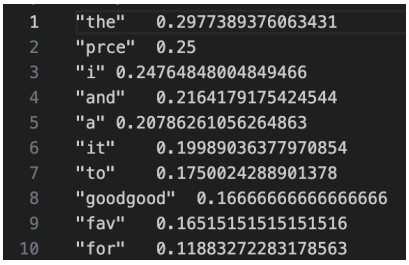

Figura 8: Output com resultados

Todo o código desta tarefa está apresentada nos Anexos da secção VII-H.

*4) **Singletons)**:* A tarefa de identificação de ”singletons”em processamento de texto refere-se à contagem de palavras ou termos que ocorrem apenas uma vez num conjunto de documentos ou textos. **Singletons** são elementos únicos que não se repetem, revelando informações exclusivas e contextuais. Essa análise é importante para compreender a raridade e diversidade vocabulário, proporcionando *insights* sobre a singularidade de certos termos no contexto do estudo. [25]

Esta tarefa consiste em três etapas principais: na primeira etapa, contam-se as ocorrências de bigramas no **mapper**, numa segunda etapa são agregadas as contagens localmente em cada nó para reduzir a quantidade de dados transferidos pela rede. Por fim nos **reducers** é inicialmente consolidada as contagens dos bigramas, criando uma lista para armazenar bigramas únicos, em que depois é realizada a análise final, calculando a percentagem de bigramas únicos, apresentando os resultados.

Foram, no entanto, feitas duas versões, em que numa, uma palavra é considerada um *singleton* (com o código apresentado nos Anexos da secção VII-F), e na outra uma frase com 2 palavras é considerada o *singleton* (com o código apresentado nos Anexos da secção VII-G). Nas figuras 9 e 10 os resultados de cada uma das tarefas, usando o ficheiro reviews.json.

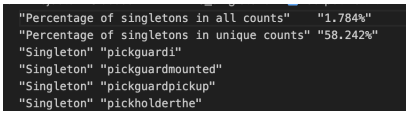

Figura 9: Output com resultados - Single Singletons

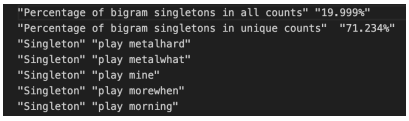

Figura 10: Output com resultados - Double Singletons

*5) **Analise de Sentimentos)**:* A análise de sentimentos é o processo de análise de texto digital para determinar se o tom emocional de uma mensagem é positivo, negativo ou neutro. As ferramentas de análise de sentimentos podem analisar este texto para determinar automaticamente a atitude do autor em relação a um tópico. As empresas utilizam as informações da análise de sentimentos para melhorar o serviço ao cliente e aumentar a reputação da marca. [26]

Para esta tarefa geral, foi pensado fazer dois tipos de análise de sentimentos.

O primeiro foi a análise de sentimentos, usando um modelo, que neste caso veio da biblioteca nltk que se irá referir mais à frente. O processo de analise de sentimentos, com o **mapreduce**, implementado no código como **SentimentAnalysis**, em vários passos. O código está presente nos Anexos na secção VII-I

A etapa **mapper_init** (que podemos observar no código seguinte) é a primeira fase do processo MapReduce e é responsável por preparar o ambiente para a execução do mapeamento. No contexto da análise de sentimentos, essa etapa é usada para inicializar o **SentimentIntensityAnalyzer** da biblioteca **NLTK**. O **SentimentIntensityAnalyzer** é uma ferramenta que usa um conjunto de regras gramaticais e léxicais para calcular a intensidade do sentimento numa frase. Ele retorna um *score* de sentimento composto que varia de -1 (muito negativo) a +1 (muito positivo).

```
def mapper_init(self):
    nltk.download('vader_lexicon')
    self.sid = SentimentIntensityAnalyzer()
    if self.options.nltk_archive:
        # Operações de configuração, se necessário
```

Neste código, **nltk.download('vader_lexicon')** é usado para fazer download o léxico **Vader**, que é um conjunto de palavras pré-computadas com valores de intensidade de sentimento associados. Esses valores são usados pelo **SentimentIntensityAnalyzer** para calcular os *score* de sentimento.

A etapa **mapper** seguinte processa cada revisão individualmente. A análise de sentimentos é realizada nesta etapa, usando o **SentimentIntensityAnalyzer** e de resultado obtém-se um *score* de sentimento composto (**sent**), sendo este usado para determinar a polaridade como positiva ou negativa.

```
def mapper(self, key, value):
    sent= self.sid.polarity_scores(value)['compound']
    if sent > 0.5:
        pol = 'positive'
    else:
        pol = 'negative'

    yield pol, 1
```

A função **combiner** é opcional e ocorre após o mapeamento. Neste caso, esta função soma as contagens parciais para cada polaridade, otimizando a eficiência na transmissão de dados para o passo de redução seguinte.

No passo **reducer**, os dados são agrupados por chave (polaridade) e a função **reducer** calcula a soma total de ocorrências para cada polaridade, como podemos ver no código seguinte:

```
def reducer(self, key, value):
    yield None, (sum(value), key)
```

A etapa final do processo de redução (**reducer_final**) é responsável por calcular a percentagem de cada polaridade em relação ao total de revisões analisadas e emitir os resultados finais

```
def reducer_final(self, _, values):
    total_counts = 0
    for count, key in values:
        total_counts += count
        yield key, count
    yield 'total', total_counts
    yield f"Percentage of {key}",
                            f"{count/total_counts}%"
```

Após esta etapa obtemos os resultados finais que representam a distribuição de sentimentos nas revisões analisadas. Estes resultados são apresentados na forma de pares chave-valor, onde a chave representa a polaridade do sentimento (positivo, negativo ou neutro) e o valor representa a contagem total ou a percentagem de revisões que expressam essa polaridade.

Um dos resultados possível (neste caso usando o ficheiro reviews.json) é o observado na figura 11.

Figura 11: Output com resultados - Análise de Sentimento

Isso indica que, do total de 10.261 revisões analisadas, 8.260 foram classificadas como positivas e 2.001 como neutras. Além disso, a percentagem de revisões positivas é de aproximadamente 80,5%. Estes resultados fornecem uma visão quantitativa do sentimento expresso nas revisões. Eles podem ser usados para diversas avaliações no mundo real (ex.identificar tendências gerais, avaliar a reação dos utilizadores a um produto ou serviço, ou informar decisões de negócios).

Contudo, ao realizar esta tarefa, apareceu um erro que devido ao tempo não nos foi possível encontrar uma solução, erro esse presente na figura 12, que só ocorre ao usar o ficheiro **reviews.json** através do **Hadoop** (sistema hdfs). Apesar de vários esforços, não foi possível encontrar uma solução.

No entanto, durante a execução desta tarefa, deparámo-nos com um problema para o qual, devido a restrições temporais, não conseguimos encontrar uma solução. O erro em questão está ilustrado na figura 12 e manifesta-se exclusivamente quando utilizamos o ficheiro **reviews.json** através do **Hadoop** (sistema HDFS). Apesar dos esforços feitos, não conseguimos identificar uma solução para esta situação.

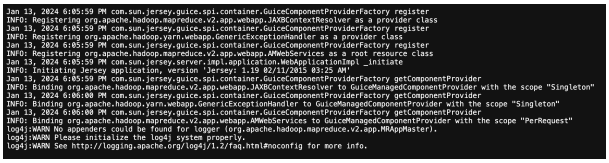

Figura 12: Erro na análise de sentimentos usado o Hadoop e HDFS

Na segunda fase da análise de sentimentos, utilizámos um ficheiro preexistente anexado a este relatório com o nome "words_sentiment.json", obtido a partir de uma versão JSON dos dados originais [27]. O objetivo era agregar todos os resultados presentes nesse JSON e apresentar um resultado geral já mapeado e reduzido. No entanto, deparamo-nos com vários erros ao executar esta tarefa no **Hadoop**, e, devido a restrições de tempo, não conseguimos concluir esta parte. Deixamos esta etapa para trabalhos futuros, para além de outros tópicos referidos no tópico seguinte. O código está presente nos Anexos na secção VII-J

V. Trabalhos futuros

Para futuras direções deste projeto, identificamos dois grandes objetivos que visam ampliar a flexibilidade e a capacidade do sistema implementado.

A. Generalização do Sistema

Uma dos objetivos principais consiste em tornar o sistema mais generalista, permitindo que o utilizador forneça apenas o ficheiro que deseja analisar, e o programa seja capaz de inferir automaticamente a tarefa a realizar. Atualmente, o utilizador precisa especificar a operação desejada, o que pode ser otimizado para tornar o processo mais intuitivo e simplificado. Este aprimoramento facilitaria a aplicação do sistema em diferentes cenários, tornando-o mais acessível e adaptável a diversas necessidades.

B. Processamento de Vídeos no MrJob

Outro ponto crucial para desenvolvimentos futuros está relacionado com a capacidade do sistema de processar vídeos eficientemente no contexto do MrJob. Embora o código para análise de vídeos já esteja bastante completo e avançado (apresentado nos Anexos na secção VII-K, identificamos uma dificuldade específica relacionada à leitura de ficheiros de vídeo com a biblioteca OpenCV dentro do ambiente MrJob no Hadoop. O código apresentado a seguir foi o usado para ler o ficheiro vídeo.

```
class VideoFileInputFormat(RawValueProtocol):

    def read(cls, file_path):
        # Implement video file reading
        # Return (key, value) for each frame in the
        # format (frame_number, image_frame_bytes)

        video_capture = cv2.VideoCapture(str(file_path),
        cv2.CAP_FFMPEG)

        record = np.array([])

        frame_number = 0

        while True:
            success, frame = video_capture.read()
            if not success:
                break

            frame_number += 1
            record.append(frame.tobytes())
            #yield (frame_number, frame.tobytes())

        return None, record.tobytes()
```

Ao correr este código, este apresenta alguns erros de leitura do ficheiro, aparecendo bastantes caracteres em bytes, apresentado na figura 13.

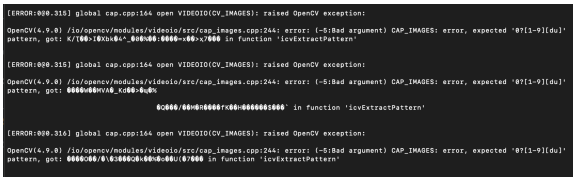

Figura 13: Erro de processamento de vídeo

Resolver essa questão, que poderia demorar algum tempo pois não existe documentação suficiente disponível, permitiria ter resultados bastante interessantes do projeto, possibilitando a análise de grandes conjuntos de dados de vídeo de maneira distribuída e paralela. Essa melhoria seria especialmente valiosa para casos de uso que envolvem análise de vídeos em larga escala, como era o nosso objetivo de deteção de faces em cada *frame* do vídeo.

VI. Conclusões

Este estudo demonstrou a utilização do modelo **MapReduce** na análise de grandes conjuntos de dados de texto. Com o auxílio da biblioteca **mrjob** em Python, foram realizadas três tarefas fundamentais: a contagem de palavras, a identificação e contagem de bigramas, e o cálculo de TF-IDF para realçar a importância de termos específicos nas revisões. No fim, foi possível acrescentar um outra tarefa, a análise de sentimentos, que foi uma parte crucial deste processo, permitindo uma compreensão mais profunda das emoções e opiniões expressas nos textos analisados.

A flexibilidade do código modular e do programa principal (main.py) permitiu a execução das tarefas tanto no ambiente Hadoop quanto localmente. Além disso, o projeto abordou o problema do processamento em tempo real, discutindo as vantagens e desvantagens do Hadoop e apresentando trabalhos relacionados na área.

Os resultados obtidos demonstram a eficácia do modelo **MapReduce** na análise de grandes volumes de dados de texto. Através da contagem de palavras, foi possível identificar as palavras mais frequentes nas revisões. A identificação e contagem de bigramas permitiu a análise de pares de palavras consecutivas, proporcionando uma visão mais contextualizada do conteúdo das revisões. Por fim, o cálculo de TF-IDF destacou a relevância de termos específicos, ajudando a identificar os tópicos mais discutidos nas revisões.

No entanto, apesar dos resultados promissores, o projeto também revelou alguns desafios associados ao processamento distribuído. A integração do Python com o Hadoop apresentou algumas dificuldades, principalmente devido a alguma dificuldade no tratamento de alguns tipos de dados, e também devido à necessidade de lidar com diferentes bibliotecas de processamento de texto. Além disso, a análise em tempo real de grandes volumes de dados continua a ser um desafio significativo, exigindo soluções mais eficientes.

Em conclusão, este projeto ofereceu uma abordagem prática para lidar com os desafios associados ao processamento distribuído de grandes conjuntos de dados de texto. O conhecimento adquirido durante o desenvolvimento deste projeto fornecem uma base sólida para futuras investigações e aplicações nesta área. A análise de sentimentos, em particular, emergiu como uma ferramenta valiosa para extrair *insights* de grandes volumes de dados de texto, com potencial para uma ampla gama de aplicações, desde a melhoria do serviço ao cliente até a informação de decisões de negócios.

REFERÊNCIAS

[1] Apache Hadoop - [Em linha] [Consult. 4 jan. 2024]. Disponível em https://hadoop.apache.org/.

[2] Big Data e Hadoop explicados: uma visão geral para todos - [Em linha] [Consult. 4 jan. 2024]. Disponível em https://www.tableau.com/pt-br/learn/articles/big-data-hadoop-explained.

[3] 5 vantagens do HADOOP – AGTECH - , [s.d.]. [Consult. 4 jan. 2024]. Disponível em https://www.agtech.com.br/vantagens-do-hadoop.

[4] Hadoop - Prós e Contras – Acervo Lima - [Em linha] [Consult. 4 jan. 2024]. Disponível em https://acervolima.com/hadoop-pros-e-contras/

[5] DBA, Dani Monteiro - Hadoop ou Bancos de Dados Relacional? [Em linha], atual. 16 jul. 2017. [Consult. 4 jan. 2024]. Disponível em https://code.likeagirl.io/hadoop-ou-bancos-de-dados-relacional-f19b5e223ae5.

[6] RATHORE, M. Mazhar et al. - Real-time video processing for traffic control in smart city using Hadoop ecosystem with GPUs. Soft Computing. 22:5 (2018) 1533–1544. doi: 10.1007/s00500-017-2942-7.

[7] SAYAR, Ahmet - Hadoop Optimization for Massive Image Processing: Case Study Face Detection. International Journal of Computers, Communications & Control (IJCCC). 9:2014) 664–671. doi: 10.15837/ijccc.2014.6.285.

[8] Post Event Investigation of Multi-stream Video Data Utilizing Hadoop Cluster — Parsola — International Journal of Electrical and Computer Engineering (IJECE) - [Em linha] [Consult. 3 jan. 2024]. Disponível em https://ijece.iaescore.com/index.php/IJECE/article/view/11754.

[9] KOUSALYA, K.; SHAIK, Javed - Effective processing of unstructured data using python in Hadoop map reduce. International Journal of Engineering & Technology. 7:2018) 417. doi: 10.14419/ijet.v7i2.21.12456.

[10] LIN, Jimmy; DYER, Chris - Data-Intensive Text Processing with MapReduceSynthesis Lectures on Human Language Technologies. [Em linha]. Cham : Springer International Publishing, 2010 [Consult. 9 jan. 2024]. Disponível em https://link.springer.com/10.1007/978-3-031-02136-7. ISBN 9783031010088 9783031021367.

[11] Application of HADOOP Technologies on Text Processing - IJSER Journal Publication - [Em linha] [Consult. 9 jan. 2024]. Disponível em www.ijser.org/onlineResearchPaperViewer.aspx?Application-of-HADOOP-Technologies-on-Text-Processing.pdf.

[12] SHVACHKO, Konstantin et al. - The Hadoop Distributed File System. Em 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST) [Em linha] [Consult. 9 jan. 2024]. Disponível em https://ieeexplore.ieee.org/document/5496972.

[13] LEUNG, Carson K.; PAZDOR, Adam G. M.; ZHENG, Hao - Scalable Mining of Big Data. Em 2021 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/IOP/SCI) [Em linha] [Consult. 9 jan. 2024]. Disponível em https://ieeexplore.ieee.org/document/9604400.

[14] Text Processing using Hadoop MapReduce — Anunay Rao - [Em linha] [Consult. 9 jan. 2024]. Disponível em https://anunayrao.github.io/dic2.html.

[15] mrjob — mrjob v0.7.4 documentation - [Em linha] [Consult. 9 jan. 2024]. Disponível em https://mrjob.readthedocs.io/en/latest/.

[16] Yelp/mrjob - [Em linha]. [S.l.] : Yelp.com, 2024 [Consult. 9 jan. 2024]. Disponível em https://github.com/Yelp/mrjob.

[17] HDFS - javatpoint - [Em linha] [Consult. 8 jan. 2024]. Disponível em https://www.javatpoint.com/hdfs.

[18] MapReduce Architecture - GeeksforGeeks, 8 set. 2020. [Consult. 11 jan. 2024]. Disponível em https://www.geeksforgeeks.org/mapreduce-architecture/.

[19] NERO, Rafael Del - When to Use Map Reduce Technologies for Systems Design Interview? [Em linha], atual. 4 dez. 2023. [Consult. 11 jan. 2024]. Disponível em https://javachallengers.com/map-reduce-fundamentals/.

[20] What is YARN (Yet Another Resource Negotiator), uses and advantages - Nixon Data - , 17 dez. 2022. [Consult. 11 jan. 2024]. Disponível em https://nixondata.com/knowledge/hadoop-fundamentals/what-is-yarn/.

[21] Hadoop YARN Architecture - GeeksforGeeks, 11 dez. 2018. [Consult. 11 jan. 2024]. Disponível em https://www.geeksforgeeks.org/hadoop-yarn-architecture/.

[22] Apache Hadoop 3.3.6 – Apache Hadoop YARN - [Em linha] [Consult. 11 jan. 2024]. Disponível em https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.

[23] HDFS Tutorial - [Em linha] [Consult. 10 jan. 2024]. Disponível em https://www.simplilearn.com/tutorials/hadoop-tutorial/hdfs.

[24] TECHNOLOGIES, Mindmajix - HDFS Architecture, Features & How to Access HDFS - Hadoop [Em linha], atual. 22 abr. 2021. [Consult. 12 jan. 2024]. Disponível em https://mindmajix.com/hadoop/hdfs-architecture-features-how-to-access-hdfs.

[25] MANNING, Christopher D.; RAGHAVAN, Prabhakar; SCHUTZE, Hinrich - Introduction to information retrieval. New York: Cambridge University Press, 2008. ISBN 9780521865715.

[26] What is Sentiment Analysis? - Sentiment Analysis Explained - AWS - [Em linha] [Consult. 13 jan. 2024]. Disponível em https://aws.amazon.com/what-is/sentiment-analysis/.

[27] NRC Emotion Lexicon - [Em linha] [Consult. 03 jan. 2024]. Disponível em https://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm.

VII. Anexos

A. Ficheiro - main.py

```
import os
import sys


def get_user_choice():
    print("Choose an operation:")
    print("1. Word Count TXT")
    print("2. Word Count JSON")
    print("3. Word Count JSON REGEX")
    print("4. Frequency Table Bigrams")
    print("5. Singletons - Bigrams")
    print("6. Singletons")
    print("7. TF-IDF")
    print("8. Sentiment Analysis with Model")
    print("9. Sentiment Analysis (Dictionary)")
    print("10. Video Analysis")

    choice = input("Enter the number corresponding to your choice: ")
    if str(choice) not in ["1", "2", "3", "4", "5", "6", "7", "8", "9", "10"]:
        print("Invalid choice. Please run again, and enter a number between 1 and 10.")
        sys.exit()
    return str(choice)


choices_paths = {
    "1": "1_word_count_txt/mapreduce.py",
    "2": "2_word_count_json/mapreduce.py",
    "3": "3_word_count_json_regex/mapreduce.py",
    "4": "4_frequency_table_bigrams/mapreduce.py",
    "5": "5_bigram_singletons/mapreduce.py",
    "6": "6_singletons/mapreduce.py",
    "7": "7_tfidf/mapreduce.py",
    "8": "8_sentimento_model/mapreduce.py",
    "9": "9_sentimento_dict/mapreduce.py",
    "10": "10_video/mapreduce.py",
}

print(choices_paths["1"].split("/")[1])

list_of_arguments = sys.argv

selected_operation = get_user_choice()
list_of_arguments.append(selected_operation)
dirname_main, filename_main = os.path.split(os.path.abspath(__file__))

print("List of arguments:")
print(list_of_arguments)


localorhadoop = list_of_arguments[1]
inputfile = list_of_arguments[2]
output = list_of_arguments[3]
pythonfilename = choices_paths[selected_operation]
nrreducers = list_of_arguments[4]
compressioncodec = list_of_arguments[5]
compressioncodec = "GzipCodec" # assuming it's always gzip to make things easier

if localorhadoop == "hadoop":
    # Remove files so it can generate the new ouputs
    destination_directory = (
        "/user/usermr/projecttextanalysis/input/"
        + choices_paths[selected_operation].split("/")[0]
    )
    destination_directory_outhome = (
        "/user/usermr/projecttextanalysis/output/"
        + choices_paths[selected_operation].split("/")[0]
    )
    os.system("hadoop fs -rm -r " + destination_directory_outhome)
    print("CMD: hadoop fs -rm -r " + destination_directory_outhome)
    os.system("rm -r " + output + "*")
    print("CMD: rm -r " + output + "*")
```

```
    check_dir_command = "hadoop fs -test -e " + destination_directory
    directory_exists = os.system(check_dir_command) == 0
    if not directory_exists:

        print(
            f"Destination directory {destination_directory} does not exist in HDFS. Creating it..."
        )
        os.system("hadoop fs -mkdir -p " + destination_directory)

    # added file to hadoop
    os.system("hadoop fs -put -f " + inputfile + " " + destination_directory)

    os.system(
        "python3 " + dirname_main + "/" + pythonfilename +
        " -r hadoop hdfs://" + destination_directory + "/" + inputfile.split("/")[-1]
        + " -o /" + destination_directory_outhome + "/"
        + " -nr " + nrreducers + " -cc " + compressioncodec
    )

    os.system(
        "hadoop fs -copyToLocal " + destination_directory_outhome + " " + output
        + choices_paths[selected_operation].split("/")[0]
    )

else:

    os.system("rm -r " + output + "*")
    if "/" in inputfile:
        os.system(
            "python3 " + dirname_main + "/" + pythonfilename + " file://" + inputfile
            + " -o " + output + choices_paths[selected_operation].split("/")[0] + "/"
            + " -nr " + nrreducers + " -cc " + compressioncodec
        )
    else:
        os.system(
            "python3 " + dirname_main + "/" + pythonfilename + " file://" + dirname_main + "/" + inputfile
            + " -o " + output + choices_paths[selected_operation].split("/")[0] + "/"
            + " -nr " + nrreducers + " -cc " + compressioncodec
        )
```

B. Ficheiro - 1_word_count_txt/mapreduce.py

```python
import re
from mrjob.job import MRJob
from mrjob.step import MRStep


class WordCounter(MRJob):
    """Job to count the number of words in a text file.
    Also, it sorts the words by frequency.
    """

    def configure_args(self):
        super(WordCounter, self).configure_args()
        self.add_passthru_arg("-nr", "--numreducers", help="Number of reducers")
        self.add_passthru_arg("-cc", "--compressioncodec", help="Compression codec (e.g., gzip)" )

    def mapper(self, _, value):
        """Uses regex pattern to find all words in the text.
        It doesn't clean the text from digits, but the regex pattern
        used detects automatically words with digits as one word.
        """
        # get words
        tokens = re.findall(
            r"\b\w+\b", value.lower()
        )  # Regex to find all words in the line
        for token in tokens:
            yield token, 1

    def combiner(self, key, values):
        """Prepare data for the Reducer.
        Sum all the values '1' for each key (word).
        """
        yield key, sum(values)

    def reducer(self, key, value):
        """Reduce the data. To get all words count, for the sorting step."""
        yield None, (sum(value), key)

    def reducer_sort(self, _, values):  # key was None
        """Sort the words by frequency.
        The ouput will the the word counts,
        starting with the less frequent words (singletons).
        """
        for count, key in sorted(values):
            yield count, key

    def steps(self):
        """Steps to run the MapReduce job.
        The first step is to count the words.
        The second step is to sort the words by frequency.
        """
        return [
            MRStep(
                mapper=self.mapper,
                combiner=self.combiner,
                reducer=self.reducer,
                jobconf={
                    "mapreduce.job.reduces": self.options.numreducers  # Set the number of reducers
                },
            ),
            MRStep(
                reducer=self.reducer_sort,
                jobconf={
                    "mapreduce.output.fileoutputformat.compress": "true",
                    "mapreduce.output.fileoutputformat.compress.codec": "org.apache.hadoop.io.compress."
                    + self.options.compressioncodec,  # Set compression codec
                },
            ),
        ]


if __name__ == "__main__":
    WordCounter.run()
```

C. Ficheiro - 2_word_count_json/mapreduce.py

```python
import re, json
from mrjob.job import MRJob
from mrjob.step import MRStep

class WordCounterJson(MRJob):
    """Job to count the number of words of a json dataset called 'Amazon Musical Instruments Reviews',
    available in https://www.kaggle.com/datasets/eswarchandt/amazon-music-reviews. Also, it sorts the words
    by frequency. """

    def configure_args(self):
        super(WordCounterJson, self).configure_args()
        self.add_passthru_arg("-nr", "--numreducers", help="Number of reducers")
        self.add_passthru_arg("-cc", "--compressioncodec", help="Compression codec (e.g., gzip)")

    def mapper(self, _, value):
        """Uses regex pattern to find all words in the json datatset field 'reviewText'.
        It doesn't clean the text from digits, but the regex pattern
        used detects automatically words with digits as one word.
        """
        review = json.loads(value)  # loads instead of load to load just a string
        review_text = review["reviewText"]  # get only the review text
        # get words
        tokens = re.findall(
            r"\b\w+\b", review_text.lower()
        )  # Regex to find all words in the review text
        for token in tokens:
            yield token, 1

    def combiner(self, key, values):
        """Prepare data for the Reducer.
        Sum all the values '1' for each key (word).
        """
        yield key, sum(values)

    def reducer(self, key, values):
        """Reduce the data. To get all words count, for the sorting step."""
        yield None, (sum(values), key)

    def reducer_sorter(self, _, values):
        """Sort the words by frequency.
        The ouput will the the word counts,
        starting with the less frequent words (singletons).
        """
        for count, key in sorted(values):
            yield count, key

    def steps(self):
        """Steps to run the MapReduce job. The first step is to count the words.
        The second step is to sort the words by frequency."""
        return [
            MRStep(
                mapper=self.mapper,
                combiner=self.combiner,
                reducer=self.reducer,
                jobconf={
                    "mapreduce.job.reduces": self.options.numreducers  # Set the number of reducers
                },
            ),
            MRStep(
                reducer=self.reducer_sorter,
                jobconf={
                    "mapreduce.output.fileoutputformat.compress": "true",
                    "mapreduce.output.fileoutputformat.compress.codec": "org.apache.hadoop.io.compress."
                    + self.options.compressioncodec,  # Set compression codec
                },
            ),
        ]


if __name__ == "__main__":
    WordCounterJson.run()
```

D. Ficheiro - 3_word_count_json_regex/mapreduce.py

```python
import re, json
from mrjob.job import MRJob
from mrjob.step import MRStep

WORD_RE = re.compile(r"\b\w+\b")  # re.compile(r"[\w']+")


class WordCounterClean(MRJob):
    """Job to count the number of words of a json dataset called
    'Amazon Musical Instruments Reviews', available in
    https://www.kaggle.com/datasets/eswarchandt/amazon-music-reviews.
    Also, it sorts the words by frequency.
    """

    def configure_args(self):
        super(WordCounterClean, self).configure_args()
        self.add_passthru_arg("-nr", "--numreducers", help="Number of reducers")
        self.add_passthru_arg(
            "-cc", "--compressioncodec", help="Compression codec (e.g., gzip)"
        )

    def mapper(self, _, value):
        """Uses regex pattern to find all words in the json datatset field 'reviewText'.
        It cleans the text from digits and punctuation.
        """
        review = json.loads(value)  # loads instead of load to load just a string
        review_text = review["reviewText"]  # get only the review text
        # remove digits and punctuation
        review_text = re.sub(r"\d+", "", review_text)  # remove digits
        review_text = re.sub(
            r"[!\"#$%&()*+-./:;<=>?@\[\\\]^_`{|}~\n]", "", review_text
        )  # remove punctuation
        # get words
        tokens = re.findall(WORD_RE, review_text.lower())
        for token in tokens:
            yield token, 1

    def combiner(self, key, values):
        """Prepare data for the Reducer.
        Sum all the values '1' for each key (word).
        """
        yield key, sum(values)

    def reducer(self, key, values):
        """Reduce the data. To get all words count, for the sorting step."""
        yield None, (sum(values), key)

    def reducer_sorter(self, _, values):
        """Sort the words by frequency.
        The ouput will the the word counts,
        starting with the less frequent words (singletons).
        """
        for count, key in sorted(values):
            yield count, key

    def steps(self):
        """Steps to run the MapReduce job.
        The first step is to count the words.
        The second step is to sort the words by frequency.
        """
        return [
            MRStep(
                mapper=self.mapper,
                combiner=self.combiner,
                reducer=self.reducer,
                jobconf={
                    "mapreduce.job.reduces": self.options.numreducers  # Set the number of reducers
                },
            ),
```

```
            MRStep(
                reducer=self.reducer_sorter,
                jobconf={
                    "mapreduce.output.fileoutputformat.compress": "true",
                    "mapreduce.output.fileoutputformat.compress.codec": "org.apache.hadoop.io.compress."
                    + self.options.compressioncodec,  # Set compression codec
                },
            ),
        ]


if __name__ == "__main__":
    WordCounterClean.run()
```

E. Ficheiro - 4_frequency_table_bigrams/mapreduce.py

```
import re
from mrjob.job import MRJob
from mrjob.step import MRStep
import json

WORD_RE = re.compile(r"\b\w+\b")  # re.compile(r"[\w']+")


class BigramFrequencyTable(MRJob):
    """Job to count the number of bigrams (co-occurrence of words) of the reviews
    in the dataset 'Amazon Musical Instruments Reviews', available in
    https://www.kaggle.com/datasets/eswarchandt/amazon-music-reviews.
    Also, it sorts the words by frequency.
    """

    def configure_args(self):
        super(BigramFrequencyTable, self).configure_args()
        self.add_passthru_arg("-nr", "--numreducers", help="Number of reducers")
        self.add_passthru_arg(
            "-cc", "--compressioncodec", help="Compression codec (e.g., gzip)"
        )

    def mapper(self, _, value):
        """Uses regex pattern to find all words in the json datatset field 'reviewText'.
        It cleans the text from digits and punctuation.
        It yields bigrams (co-occurrence of words) instead of singular tokens.
        """
        review = json.loads(value)  # loads instead of load to load just a string
        review_text = review["reviewText"]  # get only the review text
        # remove digits and punctuation
        review_text = re.sub(r"\d+", "", review_text)  # remove digits
        review_text = re.sub(
            r"[!\"#$%&()*+-./:;<=>?@\[\\\]^_`{|}~\n]", "", review_text
        )  # remove punctuation
        # get words
        tokens = re.findall(WORD_RE, review_text.lower())
        # yield bigrams as one word
        for i in range(len(tokens) - 1):
            yield f"{tokens[i]} {tokens[i+1]}", 1

    def combiner(self, key, values):
        """Prepare data for the Reducer.
        Sum all the values '1' for each key (bigram).
        """
        yield key, sum(values)

    def reducer(self, key, values):
        """Reduce the data. To get all bigrams count, for the sorting step."""
        yield None, (sum(values), key)

    def reducer_sorter(self, _, values):
        """Sort the bigrams by frequency.
        The ouput will the the bigram counts,
        starting with the most frequent bigrams.
        """
        for count, key in sorted(values, reverse=True):
```

```
            yield count, key

    def steps(self):
        """Steps to run the MapReduce job.
        The first step is to count the bigrams.
        The second step is to sort the bigrams by frequency.
        """
        return [
            MRStep(
                mapper=self.mapper,
                combiner=self.combiner,
                reducer=self.reducer,
                jobconf={
                    "mapreduce.job.reduces": self.options.numreducers  # Set the number of reducers
                },
            ),
            MRStep(
                reducer=self.reducer_sorter,
                jobconf={
                    "mapreduce.output.fileoutputformat.compress": "true",
                    "mapreduce.output.fileoutputformat.compress.codec": "org.apache.hadoop.io.compress."
                    + self.options.compressioncodec,  # Set compression codec
                },
            ),
        ]


if __name__ == "__main__":
    BigramFrequencyTable.run()
```

F. Ficheiro - 5_bigram_singletons/mapreduce.py

```
import re
from mrjob.job import MRJob
from mrjob.step import MRStep
import json

WORD_RE = re.compile(r"\b\w+\b")  # re.compile(r"[\w']+")


class SingletonBigrams(MRJob):
    """Job to get the singletons of a json dataset called
    'Amazon Musical Instruments Reviews', available in
    https://www.kaggle.com/datasets/eswarchandt/amazon-music-reviews.
    Also, it sorts the words by frequency.
    """

    def configure_args(self):
        super(SingletonBigrams, self).configure_args()
        self.add_passthru_arg("-nr", "--numreducers", help="Number of reducers")
        self.add_passthru_arg(
            "-cc", "--compressioncodec", help="Compression codec (e.g., gzip)"
        )

    def mapper(self, _, value):
        """Uses regex pattern to find all words in the json datatset field 'reviewText'.
        It cleans the text from digits and punctuation.
        It yields bigrams (co-occurrence of words) instead of singular tokens.
        """
        review = json.loads(value)  # loads instead of load to load just a string
        review_text = review["reviewText"]  # get only the review text
        # remove digits and punctuation
        review_text = re.sub(r"\d+", "", review_text)  # remove digits
        review_text = re.sub(
            r"[!\"#$%&()*+-./:;<=>?@\[\\\]^_`{|}~\n]", "", review_text
        )  # remove punctuation
        # get words
        tokens = re.findall(WORD_RE, review_text.lower())
        # yield bigrams as one word
        for i in range(len(tokens) - 1):
            yield f"{tokens[i]} {tokens[i+1]}", 1
```

```
    def combiner(self, word, counts):
        """Prepare data for the Reducer.
        Sum all the values '1' for each key (word).
        """
        yield word, sum(counts)

    def reducer(self, key, values):
        """Reduce the data. Get all words count,
        for the singleton identification step
        """
        yield None, (sum(values), key)

    def reducer_init(self):
        """Initialize the singletons list"""
        self.singletons = []

    def reducer_final(self, _, counts):
        """"""
        total_counts = 0
        total_uniques = 0
        for value, key in counts:
            # add total number of words
            total_counts += value
            # add to total number of unique words
            total_uniques += 1
            # if it is a singleton, add it to the list
            if value == 1:
                self.singletons.append(key)
        # yield the results
        yield "Percentage of bigram singletons in all counts", f"{round(len(self.singletons)/total_counts*100, 3)}%"
        yield "Percentage of bigram singletons in unique counts", f"{round(len(self.singletons)/total_uniques*100, 3)}
        for word in self.singletons:
            yield "Singleton", word

    def steps(self):
        """Steps to run the MapReduce job.
        The first step is to count the words.
        The second step is to get the singletons and the percentage of singletons.
        """
        return [
            MRStep(
                mapper=self.mapper,
                combiner=self.combiner,
                reducer=self.reducer,
                jobconf={
                    "mapreduce.job.reduces": self.options.numreducers  # Set the number of reducers
                },
            ),
            MRStep(
                reducer_init=self.reducer_init,
                reducer=self.reducer_final,
                jobconf={
                    "mapreduce.output.fileoutputformat.compress": "true",
                    "mapreduce.output.fileoutputformat.compress.codec": "org.apache.hadoop.io.compress."
                    + self.options.compressioncodec,  # Set compression codec
                },
            ),
        ]


if __name__ == "__main__":
    SingletonBigrams.run()
```

G. Ficheiro - 6_singletons/mapreduce.py

```python
import re
from mrjob.job import MRJob
from mrjob.step import MRStep
import json

WORD_RE = re.compile(r"\b\w+\b")  # re.compile(r"[\w']+")


class SingletonWords(MRJob):
    """Job to get the singletons of a json dataset called
    'Amazon Musical Instruments Reviews', available in
    https://www.kaggle.com/datasets/eswarchandt/amazon-music-reviews.
    Also, it sorts the words by frequency.
    """

    def configure_args(self):
        super(SingletonWords, self).configure_args()
        self.add_passthru_arg("-nr", "--numreducers", help="Number of reducers")
        self.add_passthru_arg(
            "-cc", "--compressioncodec", help="Compression codec (e.g., gzip)"
        )

    def mapper(self, _, value):
        """Uses regex pattern to find all words in the json datatset field 'reviewText'.
        It cleans the text from digits and punctuation.
        """
        review = json.loads(value)  # loads instead of load to load just a string
        review_text = review["reviewText"]  # get only the review text
        # remove digits and punctuation
        review_text = re.sub(r"\d+", "", review_text)  # remove digits
        review_text = re.sub(
            r"[!\"#$%&()*+-./:;<=>?@\[\\\]^_`{|}~\n]", "", review_text
        )  # remove punctuation
        # get words
        tokens = re.findall(WORD_RE, review_text.lower())
        for token in tokens:
            yield token, 1

    def combiner(self, word, counts):
        """Prepare data for the Reducer.
        Sum all the values '1' for each key (word).
        """
        yield word, sum(counts)

    def reducer(self, key, values):
        """Reduce the data. Get all words count,
        for the singleton identification step
        """
        yield None, (sum(values), key)

    def reducer_init(self):
        """Initialize the singletons list"""
        self.singletons = []

    def reducer_final(self, _, counts):
        """"""
        total_counts = 0
        total_uniques = 0
        for value, key in counts:
            # add total number of words
            total_counts += value
            # add to total number of unique words
            total_uniques += 1
            # if it is a singleton, add it to the list
            if value == 1:
                self.singletons.append(key)
        # yield the results
        yield "Percentage of singletons in all counts", f"{round(len(self.singletons)/total_counts*100, 3)}%"
        yield "Percentage of singletons in unique counts", f"{round(len(self.singletons)/total_uniques*100, 3)}%"
        for word in self.singletons:
            yield "Singleton", word

    def steps(self):
```

```
        """Steps to run the MapReduce job.
        The first step is to count the words.
        The second step is to get the singletons and the percentage of singletons.
        """
        return [
            MRStep(
                mapper=self.mapper,
                combiner=self.combiner,
                reducer=self.reducer,
                jobconf={
                    "mapreduce.job.reduces": self.options.numreducers  # Set the number of reducers
                },
            ),
            MRStep(
                reducer_init=self.reducer_init,
                reducer=self.reducer_final,
                jobconf={
                    "mapreduce.output.fileoutputformat.compress": "true",
                    "mapreduce.output.fileoutputformat.compress.codec": "org.apache.hadoop.io.compress."
                    + self.options.compressioncodec,  # Set compression codec
                },
            ),
        ]


if __name__ == "__main__":
    SingletonWords.run()
```

H. Ficheiro - 7_tfidf/mapreduce.py

```
import re  # Regex library
import json
from mrjob.job import MRJob  # MapReduce library
from mrjob.step import MRStep

WORD_RE = re.compile(r"\b\w+\b")  # re.compile(r"[\w']+")
class WordTFIDF(MRJob):
    """Job to calculate TF-IDF from the words in the reviews.
    JSON dataset 'Amazon Musical Instruments Reviews', available in
    https://www.kaggle.com/datasets/eswarchandt/amazon-music-reviews.
    """

    def configure_args(self):
        super(WordTFIDF, self).configure_args()
        self.add_passthru_arg("-nr", "--numreducers", help="Number of reducers")
        self.add_passthru_arg(
            "-cc", "--compressioncodec", help="Compression codec (e.g., gzip)"
        )

    def mapper_init(self):
        """Initialize the variables to calculate TF-IDF.
        This method is called before the mapper and it's needed
        because we can't manipulate mutable objects inside the mapper.
        """
        self.df_dict = {}
        self.tf_dict = {}
        self.terms_idf = None
        self.terms_tf_idf = None

    def mapper_tf(self, _, value):

        review = json.loads(value)  # loads instead of load to load just a string
        # get the review text and ID
        review_text = review["reviewText"]
        review_id = review["reviewerID"]
        # remove digits and punctuation
        review_text = re.sub(r"\d+", "", review_text)  # remove digits
        review_text = re.sub(
            r"[!\"#$%&()*+-./:;<=>?@\[\\\]^_`{|}~\n]", "", review_text
        )  # remove punctuation
        # get words
```

```
        tokens = re.findall(WORD_RE, review_text.lower())
        # yeld the TF for each word, in each review
        for token in tokens:
            yield f"{review_id} {token}", 1 / len(tokens)

    def combiner_tf(self, key, values):
        """Prepare data for the Reducer.
        Sum all the values '1' for each key (doc_id word).
        """
        yield key, sum(values)

    def reducer_tf(self, key, values):
        """Reduce the data. To get all words TF."""
        yield None, (key, sum(values))

    def reducer_init_tfidf(self):
        """Initialize the variables to calculate TF-IDF.
        This method is called before the reducer and it's needed
        because we can't manipulate mutable objects inside the reducer.
        """
        self.df_dict = {}  # document frequency
        self.tf_dict = {}  # term frequency
        self.terms_idf = {}  # inverse document frequency
        self.terms_tf_idf = {}  # TF-IDF

    def reducer_tfidf(self, _, values):
        """
        Populate the dictionaries to calculate TF-IDF.
        """
        for key, value in values:
            # get the review ID and the word
            review_id, word = key.split()
            # populate the dictionaries
            self.df_dict[word] = self.df_dict.get(word, [])
            self.df_dict[word].append(review_id)
            self.tf_dict[word] = self.tf_dict.get(word, 0) + value

        # get terms IDF
        self.terms_idf = {k: 1 / (len(v) + 1) for k, v in self.df_dict.items()}
        # get terms TF-IDF
        self.terms_tf_idf = {k: v * self.terms_idf[k] for k, v in self.tf_dict.items()}

        # yield by order (descending)
        for k, v in sorted(
            self.terms_tf_idf.items(), key=lambda item: item[1], reverse=True
        ):
            yield k, v

    def steps(self):
        """Steps to run the MapReduce job. The first step is to count the words. The second step is to calculate TF-ID
        and order the terms by descending order. """
        return [
            MRStep(
                mapper=self.mapper_tf,
                combiner=self.combiner_tf,
                reducer=self.reducer_tf,
                jobconf={
                    "mapreduce.job.reduces": self.options.numreducers  # Set the number of reducers
                },
            ),
            MRStep(
                reducer_init=self.reducer_init_tfidf,
                reducer=self.reducer_tfidf,
                jobconf={
                    "mapreduce.output.fileoutputformat.compress": "true",
                    "mapreduce.output.fileoutputformat.compress.codec": "org.apache.hadoop.io.compress."
                    + self.options.compressioncodec,  # Set compression codec
                },
            ),
        ]


if __name__ == "__main__":
    WordTFIDF.run()
```

I. Ficheiro - 8_sentimento_model/mapreduce.py

```
#!/usr/bin/env python

from mrjob.job import MRJob
from mrjob.step import MRStep

import nltk
from nltk.sentiment import SentimentIntensityAnalyzer


class SentimentAnalysis(MRJob):
    """Job to perform sentiment analysis on the reviews, using a NLTK model."""

    def configure_options(self):
        super(SentimentAnalysis, self).configure_options()
        self.add_file_arg("--nltk-archive", help="Path to NLTK archive")

    def configure_args(self):
        super(SentimentAnalysis, self).configure_args()
        self.add_passthru_arg("-nr", "--numreducers", help="Number of reducers")
        self.add_passthru_arg(
            "-cc", "--compressioncodec", help="Compression codec (e.g., gzip)"
        )
        self.add_file_arg("--nltk-archive")

    def mapper_init(self):
        """Initialize the mapper, by initializing the SentimentIntensityAnalyzer."""
        # Download the VADER lexicon (if not already downloaded)
        nltk.download("vader_lexicon")
        # Create a SentimentIntensityAnalyzer
        self.sid = SentimentIntensityAnalyzer()
        # Unpack NLTK archive and set it up in the mapper's initialization
        nltk_archive = self.options.nltk_archive
        if nltk_archive:
            # Unpack NLTK archive or perform any setup needed
            pass

    def mapper(self, key, value):
        """Perform sentiment analysis on the review text.
        Yield the sentiment and the count of 1.
        """
        sent = self.sid.polarity_scores(value)[
            "compound"
        ]  # Get the sentiment score for the review
        if sent > 0.5:
            pol = "positive"
        else:
            pol = "negative"

        yield pol, 1

    def combiner(self, key, values):
        """Prepare data for the Reducer.
        Sum all the values '1' for each key (polarity).
        """
        yield key, sum(values)

    def reducer(self, key, value):
        """Reduce the data. To get all polarities count."""
        yield None, (sum(value), key)

    def reducer_final(self, _, values):
        """Calculate the percentage of each polarity.
        Yield the count and the percentage of each polarity.
        """
        total_counts = 0
        for count, key in values:
            total_counts += count
            yield key, count
        yield "total", total_counts
        yield f"Percentage of {key}", f"{count/total_counts*100}%"
```

```
    def steps(self):
        """Steps to run the MapReduce job.
        The first step is to classify the sentiment of each review
         and to count the polarities.
        The second step is to return polarities statistics.
        """
        return [
            MRStep(
                mapper_init=self.mapper_init,
                mapper=self.mapper,
                combiner=self.combiner,
                reducer=self.reducer,
                jobconf={
                    "mapreduce.job.reduces": self.options.numreducers  # Set the number of reducers
                },
            ),
            MRStep(
                reducer=self.reducer_final,
                jobconf={
                    "mapreduce.output.fileoutputformat.compress": "true",
                    "mapreduce.output.fileoutputformat.compress.codec": "org.apache.hadoop.io.compress."
                    + self.options.compressioncodec,  # Set compression codec
                },
            ),
        ]


if __name__ == "__main__":
    SentimentAnalysis.run()
```

J. Ficheiro - 9_sentimento_dict/mapreduce.py

```
import re  # Regex library
import json
from mrjob.job import MRJob  # MapReduce library
from mrjob.step import MRStep

# from sentimento import sentiment_dict

WORD_RE = re.compile(r"\b\w+\b")  # re.compile(r"[\w']+")


class SentimentDict(MRJob):
    """Job to perform sentiment analysis on the reviews, using a dictionarity."""

    def configure_args(self):
        super(SentimentDict, self).configure_args()
        self.add_passthru_arg("-nr", "--numreducers", help="Number of reducers")
        self.add_passthru_arg(
            "-cc", "--compressioncodec", help="Compression codec (e.g., gzip)"
        )

    def mapper_init(self):
        self.sentiment_dict = {"positive": [], "negative": [], "neutral": []}
        # open json file with word polarities
        with open("words_sentiment.json") as f:
            words_sentiment = json.load(f)

        """
        Each word has this values:

        "anger": 0,
        "anticipation": 0,
        "disgust": 0,
        "fear": 0,
        "joy": 0,
        "negative": 0,
        "positive": 1,
        "sadness": 0,
        "surprise": 0,
        "trust": 1
```

```
        """

        positive_scores = ["anticipation", "joy", "positive", "surprise", "trust"]

        negative_scores = ["anger", "disgust", "fear", "negative", "sadness"]

        for word, value in words_sentiment.items():
            pos, neg = 0, 0
            for score, val in value.items():
                if score in positive_scores:
                    pos += val
                elif score in negative_scores:
                    neg += val

            if pos > neg:
                self.sentiment_dict["positive"].append(word)
            elif neg > pos:
                self.sentiment_dict["negative"].append(word)
            else:
                self.sentiment_dict["neutral"].append(word)
        # self.sentiment_dict = sentiment_dict

    def mapper(self, key, value):
        """Extract words from the review text and perform sentiment analysis."""
        review = json.loads(value)  # loads instead of load to load just a string
        review_text = review["reviewText"]  # get only the review text
        # remove digits and punctuation
        review_text = re.sub(r"\d+", "", review_text)  # remove digits
        review_text = re.sub(
            r"[!\"#$%&()*+-./:;<=>?@\[\\\]^_`{|}~\n]", "", review_text
        )  # remove punctuation
        tokens = re.findall(
            WORD_RE, review_text.lower()
        )  # Regex to find all words in the review text

        sentence_polarity = 0
        for token in tokens:
            if token in self.sentiment_dict["positive"]:
                sentence_polarity += 1
            elif token in self.sentiment_dict["negative"]:
                sentence_polarity -= 1

        if sentence_polarity > 0:
            yield "positive", 1
        elif sentence_polarity < 0:
            yield "negative", 1
        else:
            yield "neutral", 1

    def combiner(self, key, values):
        """Prepare data for the Reducer.
        Sum all the values '1' for each key (polarity).
        """
        yield key, sum(values)

    def reducer(self, key, values):
        """Reduce the data. To get all words count, for the sorting step."""
        yield None, (sum(values), key)

    def reducer_final(self, _, values):
        """Calculate the percentage of each polarity.
        Yield the count and the percentage of each polarity.
        """
        total_counts = 0
        for count, key in values:
            total_counts += count
            yield key, count
        yield "total", total_counts
        yield f"Percentage of {key}", f"{count/total_counts}%"

    def steps(self):
        """Steps to run the MapReduce job.
        The first step is to classify the sentiment of each review,
```

```
        using a dictionary and to count the polarities.
        The second step is to return polarities statistics.
        """
        return [
            MRStep(
                mapper=self.mapper,
                combiner=self.combiner,
                reducer=self.reducer,
                jobconf={
                    "mapreduce.job.reduces": self.options.numreducers  # Set the number of reducers
                },
            ),
            MRStep(
                reducer=self.reducer_final,
                jobconf={
                    "mapreduce.output.fileoutputformat.compress": "true",
                    "mapreduce.output.fileoutputformat.compress.codec": "org.apache.hadoop.io.compress."
                    + self.options.compressioncodec,  # Set compression codec
                },
            ),
        ]


if __name__ == "__main__":
    SentimentDict.run()
```

K. Ficheiro - 10_video/mapreduce.py

```
from mrjob.job import MRJob, MRStep
from mrjob.protocol import BytesValueProtocol
import cv2
import numpy as np


class VideoFileInputFormat(BytesValueProtocol):
    """Allows reading video files as input for MapReduce jobs."""

    def read(cls, file_path):
        """Reads a video file and returns a generator of (key, value) pairs.
        Return (key, value) for each frame in the format (frame_number, image_frame_bytes)
        """

        # read the video file
        video_capture = cv2.VideoCapture(str(file_path), cv2.CAP_FFMPEG)

        # add frames to the record
        record = np.array([])
        frame_number = 0
        while True:
            success, frame = video_capture.read()
            if not success:
                break
            frame_number += 1
            record.append(frame)
            # yield (frame_number, frame.tobytes())

        return None, record.tobytes()


class FaceDetector(MRJob):
    """Job to detect faces in a video file."""

    INPUT_PROTOCOL = VideoFileInputFormat
    INTERNAL_PROTOCOL = BytesValueProtocol

    def mapper_init(self):
        """Initializes the face detection model."""
        # Load the cascade classifier at the beginning of the mapper
        self.face_cascade = cv2.CascadeClassifier(
            cv2.data.haarcascades + "haarcascade_frontalface_default.xml"
        )
```

```
    def mapper(self, _, record):
        """Detects faces in the video file."""

        # convert the record to a numpy array from
        # record_array = np.array(record)
        # Convert the bytes to a numpy array
        # buffer_img = cv2.imencode('.jpg', image_bytes)
        # image_array = np.frombuffer(buffer_img)
        # frame = cv2.imdecode(image_array) #, cv2.IMREAD_COLOR)
        # Convert the bytes to a numpy array
        record_buffer = np.frombuffer(record, dtype=np.uint8)

        # record_array = record_buffer.reshape((record_buffer.shape[0], record_buffer.shape[1], record_buffer.shape[2]

        frame_number = 0

        record_array = np.array(record_buffer)
        for frame in record_array:
            # frame_buffer = np.frombuffer(frame_bytes, dtype=np.uint8)
            # frame = cv2.imdecode(frame_buffer, cv2.IMREAD_COLOR)
            frame_number += 1

            yield (frame_number, frame)

    def reducer(self, frame_number, frame):
        """Count the number of faces in the video frames."""

        faces = self.face_cascade.detectMultiScale(
            frame, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30)
        )
        # count_faces = 0
        # for face in faces:
        # count_faces += 1

        # percentage = 100 * count_faces / count_frames

        yield frame_number, len(faces)

    def steps(self):
        """Steps to run the MapReduce to detect faces in a video file."""
        return [
            MRStep(
                mapper_init=self.mapper_init,
                mapper=self.mapper,
                reducer=self.reducer,
                # jobconf={
                #
                #     'mapreduce.job.reduces': self.options.numreducers  # Set the number of reducers
                # }
            )
        ]


if __name__ == "__main__":
    FaceDetector.run()
```