PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Paulo Roberto Garcia Medeiros

CLUSTERIZAÇÃO DE TWEETS – O COVID NA REDE SOCIAL

Campinas
2021

Paulo Roberto Garcia Medeiros

CLUSTERIZAÇÃO DE TWEETS – O COVID NA REDE SOCIAL

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Campinas
2021

SUMÁRIO

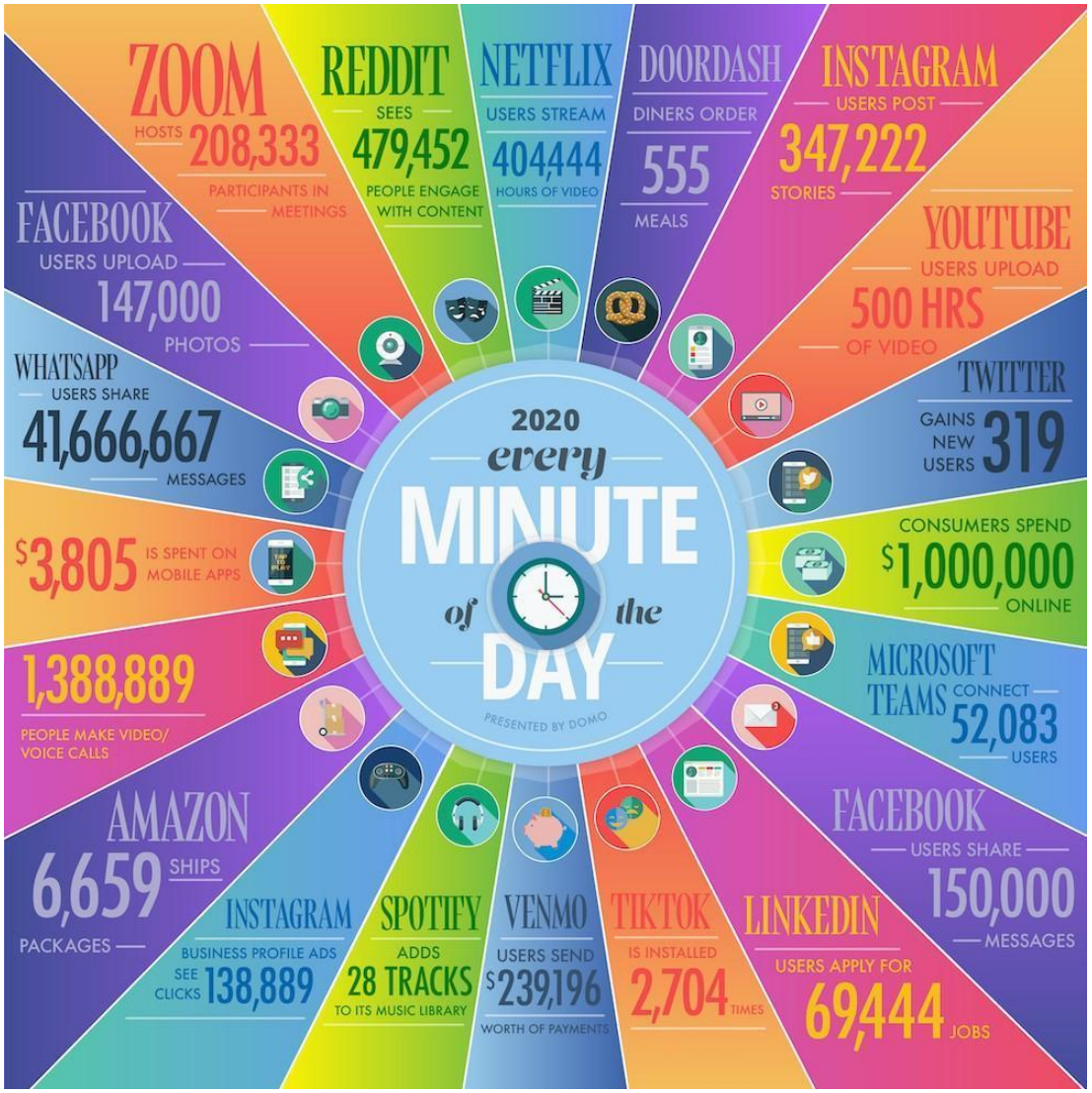
1. Introdução

O trabalho propõe aplicar uma metodologia de clusterização a dados obtidos por meio de integração com a rede social *Twitter, utilizando comentários* que possuam algum relacionamento com o tema Covid. A estruturação do projeto se dá em etapas, sendo elas:

- Introdução – Contexto e abordagem ao problema proposto ao projeto;
- Coleta de dados – Forma de obtenção de dados, apresentando o mecanismo de captura dos mesmos;
- Processamento/Tratamento de dados – Descrição da etapa de tratamento dos dados tanto em relação às técnicas de processamento, quanto às decisões tomadas para viabilizar uma estrutura pertinente a aplicação do modelo estatístico;
- Análise exploração de dados – Momento em que é abordado a análise dos dados a fim de obter *insights;*
- Criação de modelo de *Machine Learning* – Tópico onde é abordado as ferramentas utilizadas, bem como o algoritmo estatístico aplicado para solucionar a questão;
- Apresentação dos resultados obtidos.

1.1. Contextualização

Vivemos em uma época em que um volume gigantesco de dados é gerado diariamente na rede mundial de computadores. Diversas fontes são responsáveis por tamanha quantidade de dados, por exemplo, sensores, satélites e interações entre seres humanos. Recentemente, foi divulgada uma pesquisa que indica que a geração do volume de dados duplica a cada quatro anos [1]. A figura abaixo mostra a volumetria de dados gerada por minuto nas redes sociais no ano passado. O volume realmente é gritante e surpreendente, e abre oportunidades sem precedentes para análises dos mais diversos temas.



- Figura 1 – Quantidade de dados gerados a cada minuto em 2020 [2]

1.2. O problema proposto

Atualmente vivemos uma crise mundial causada pelo Covid. Por definição**,** Covid significa *COrona VIrus Disease* (Doença do Coronavírus), enquanto “19” se refere a 2019, quando os primeiros casos em *Wuhan*, na China, foram divulgados publicamente pelo governo chinês no final de dezembro [3]. Inegavelmente, essa questão impõe inúmeros impactos na sociedade no Brasil e no mundo. Buscando entender um pouco mais essa questão, o projeto propõe obter dados a partir da rede social *twitter* de forma mais atualizada possível e a partir desses dados, obter a percepção das pessoas que tecem comentários na rede social sobre o Covid.

Desse modo, dado o volume gigantesco de dados gerados na rede social, atingir esse objetivo implica em um grande desafio, pois geralmente coexistem uma infinidade de temas distintos em um formato não estruturado. Obter informação a partir desse contexto requer a utilização de *frameworks* de *Big Data* apropriados, em conjunto de algoritmo que facilite tal tarefa. Nesse sentido, foi utilizado as tecnologias abaixo para este projeto:

- Apache Flume 1.7
- Apache Hadoop 3.1.3
- Apache Spark 3.1.1
- Apache Hive 3.1.2
- Apache Maven 3.6.3
- MySql 5.7
- Anaconda 3
- Python 3.8.5
- Java 1.8

Além desse fato, a compreensão de toda essa informação a fim de obter conhecimento para gerar ações que corroborem também é algo complexo, justamente pela criticidade e volatilidade da informação.

2. Coleta de Dados

Como etapa inicial, foi necessário solicitar a abertura de uma conta de desenvolvedor na rede social *Twitter* [4], responder a algumas questões a respeito da finalidade da solicitação e aguardar um período de aprovação. Após a aprovação, foi necessário gerar um conjunto de chaves de acesso para *API* do *Twitter*, sendo*: Consumer Key, ConsumerSecret, Access Token* e *Access Token Secret.*

Como mecanismo de comunicação com a *API do Twitter,* foi utilizado o *framework Apache Flume* [5] na versão 1.7, pois trata-se de uma aplicação de grande performance no transporte e ingestão de volume de dados. Para a sua utilização, foi necessário criar um arquivo de configuração do projeto conforme demonstrado na figura 2.

```
# Definição do source, channel e sink
TwitterAgent.sources = Twitter
TwitterAgent.channels = MemChannel
TwitterAgent.sinks = HDFS

# Definição do data source para obtenção dos dados
TwitterAgent.sources.Twitter.type = com.cloudera.flume.source.TwitterSource
TwitterAgent.sources.Twitter.consumerKey = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
TwitterAgent.sources.Twitter.consumerSecret = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
TwitterAgent.sources.Twitter.accessToken = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
TwitterAgent.sources.Twitter.accessTokenSecret = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

# Termos de pesquisa
TwitterAgent.sources.Twitter.keywords = COVID, CORONA

# Língua alvo
TwitterAgent.sources.Twitter.languages =pt_br

# Deninição do local onde será armazenado as informações (hadoop)
TwitterAgent.sinks.HDFS.type = hdfs
TwitterAgent.sinks.HDFS.hdfs.path = /user/hadoop/twitter_data
TwitterAgent.sinks.HDFS.hdfs.fileType = DataStream
TwitterAgent.sinks.HDFS.hdfs.writeFormat = Text
TwitterAgent.sinks.HDFS.hdfs.batchSize = 100
TwitterAgent.sinks.HDFS.hdfs.rollSize = 0
TwitterAgent.sinks.HDFS.hdfs.rollCount = 1000

# Definiçao das informações do canal
TwitterAgent.channels.MemChannel.type = memory
TwitterAgent.channels.MemChannel.capacity = 1000
TwitterAgent.channels.MemChannel.transactionCapacity = 1000
TwitterAgent.channels.MemChannel.byteCapacity = 6912212

# Definição do bind das configuraões do source e sink no channel
TwitterAgent.sources.Twitter.channels = MemChannel
TwitterAgent.sinks.HDFS.channel = MemChannel
```

- Figura 2 – Arquivo de parametrização do projeto para o Flume.

Dentre as configurações, pode-se observar a definição das chaves de acesso mencionadas anteriormente e a presença do parâmetro *sources type* definido como “com.cloudera.flume.source.TwitterSource”. Esse parâmetro indica ao *Apache Flume* a localização da aplicação que definirá como os dados devem ser obtidos do *Twitter*. Para contemplar o escopo proposto, foi necessário realizar algumas modificações na implementação base dessa aplicação, pois era necessário obter apenas *tweets* em português do Brasil. Para tal, foi definido um parâmetro customizado chamado *sources languages* no arquivo de configuração do projeto e a modificação da implementação base do *sources type*. O projeto base foi obtido no repositório da *Cloudera* [6] e algumas modificações realizadas:

- Na classe “TwitterSourceConstants”, foi incluído uma nova constante chamada “LANGUAGE”;

- Na classe “TwitterSource” foi incluído um trecho de código para obter do *contexto* do parâmetro *sources* *languages* e aplicá-lo como critério de filtro dos dados;

A implementação se observa nas figuras 2.1 e 2.2:

```
String languageString = context.getString(TwitterSourceConstants.LANGUAGE, defaultValue: "");

if (languageString.trim().length() == 0) {
  languages = new String[0];
} else {
  languages = languageString.split( regex: ",");
  for (int i = 0; i < languages.length; i++) {
    languages[i] = languages[i].trim();
  }
}
```

- Figura 2.1 – Obtenção do parâmetro no contexto.

```
// Set up a filter to pull out industry-relevant tweets
if (keywords.length == 0) {
  logger.debug("Starting up Twitter sampling...");
  twitterStream.sample();
} else {
  logger.debug("Starting up Twitter filtering...");

  FilterQuery query = new FilterQuery()
          .language(languages)
          .track(keywords);
  twitterStream.filter(query);
}
super.start();
```

- Figura 2.2 – Aplicação do parâmetro obtido no critério de filtro dos dados.

Posteriormente, foi gerado um novo *build* da aplicação por meio do gerenciador de dependências *Maven* [7] na versão 3.6.3. O artefato gerado no *build* foi adicionado no diretório *lib* contido na raiz da instalação do *Apache Flume.* Também foi necessário atualizar a versão de algumas bibliotecas com o propósito de corrigir alguns problemas de incompatibilidade, sendo elas: guava-30.1-jre.jar, twitter4j-core-4.0.7.jar, twitter4j-media-support-4.0.6.jar, twitter4j-stream-4.0.7.jar. Todas essas bibliotecas foram adicionadas no diretório *lib* e as antigas versões removidas.

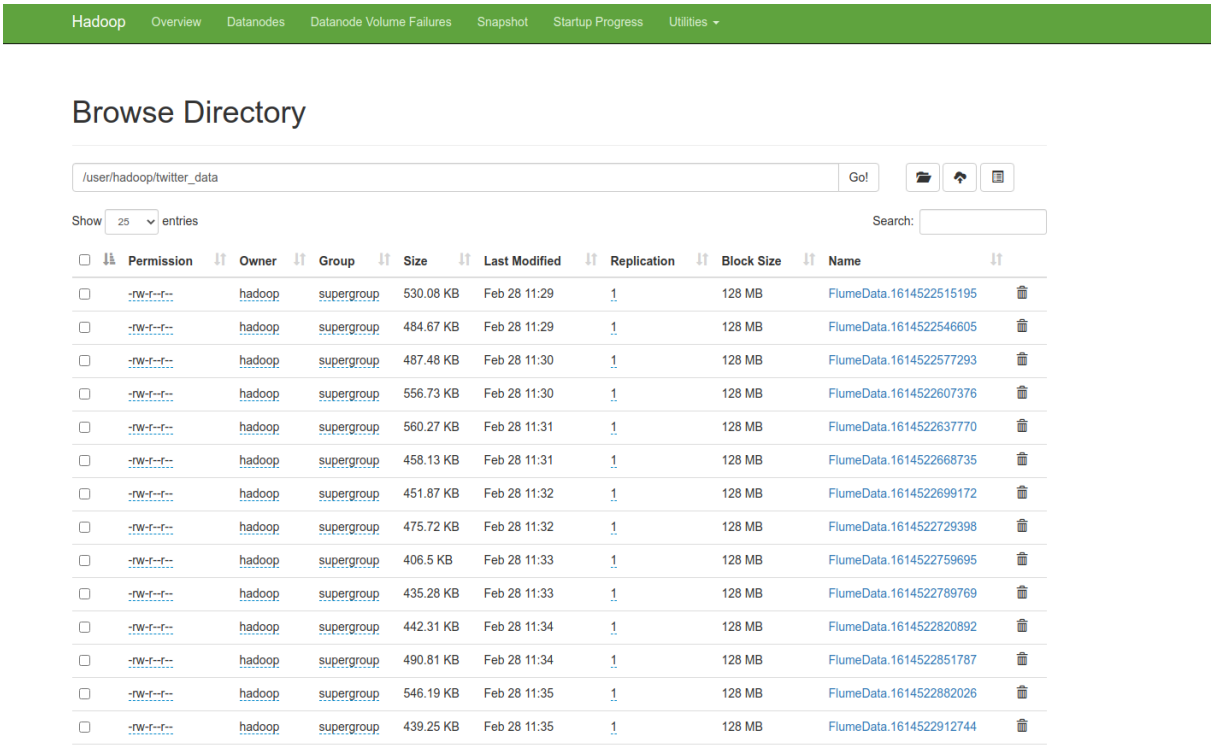
Outro parâmetro importante a ser citado no arquivo de configuração é o *sinks type, pois* define que a ingestão dos dados coletados será realizada no *path* "/user/hadoop/twitter_data" do *HDFS (Hadoop Distributed File System)* do *Apache Hadoop.*

Foi utilizado o *Apache Hadoop* [8] na versão 3.1.3, que é um *framework* escalável e tolerante a falhas para a tarefa de armazenar os dados obtidos. Para a sua instalação, foi necessário realizar uma série de configurações seguindo um tutorial [9] para o sistema operacional *Linux,* além das orientações da documentação oficial da *Apache* [10]. Vale mencionar dois importantes arquivos de configuração do *framework*, sendo: *core-site.xml e hdfs-site.xml.*

No arquivo *core-site.xml* foi criado o parâmetro "fs.defaultFS" definido como "hdfs://localhost:9000" para informar ao *Apache Hadoop* que *HDFS* funcionará na porta 9000 e o parâmetro "hadoop.tmp.dir" definido como "/home/hadoop/hadoop/hadooptmpdata" para informar o *path* em que o *HDFS* armazenará os arquivos temporários.

No arquivo *hdfs-site.xml* foi incluído o parâmetro "dfs.replication" com valor "1" informando ao *Apache Hadoop* para manter apenas uma cópia dos arquivos no sistema de arquivos, um parâmetro chamado "dfs.name.dir" definido como "file:///home/hadoop/hadoop/hdfs/namenode" informando o *path* para o armazenamento das informações do *name node* (master), e o parâmetro "dfs.data.dir" com valor "file:///home/hadoop/hadoop/hdfs/datanode" definindo o *path* para o armazenamento das informações do *data node* (blocos de informações). Por fim um parâmetro chamado "dfs.permissions" que foi definido como "false" para solucionar um problema de permissão que estava ocorrendo ao tentar criar volumes no *HDFS*.

Após concluir as configurações supracitadas, foi possível iniciar o agente do *Apache Flume* para realizar a ingestão dos dados no *HDFS* do *Apache Hadoop* conforme a figura 2.3.



- Figura 2.3 – Dados coletados armazenados no HDFS.

Para facilitar a visualização dos dados coletados, foi utilizado o *framework Apache Hive* [11] na versão 3.1.2. O *framework* provê uma uma interface de consulta em *HQL* (*Hive Query Language*) semelhante ao *SQL*, que abstrai toda complexidade do processamento Map/Reduce para obter os dados do *HDFS* [12]. Sendo assim, podemos filtrar e definir na instrução de consulta quais dados são interessantes na visualização. Durante sua configuração, foi necessário definir algumas propriedades em um arquivo chamado *hive-site.xml* conforme figura 2.4.

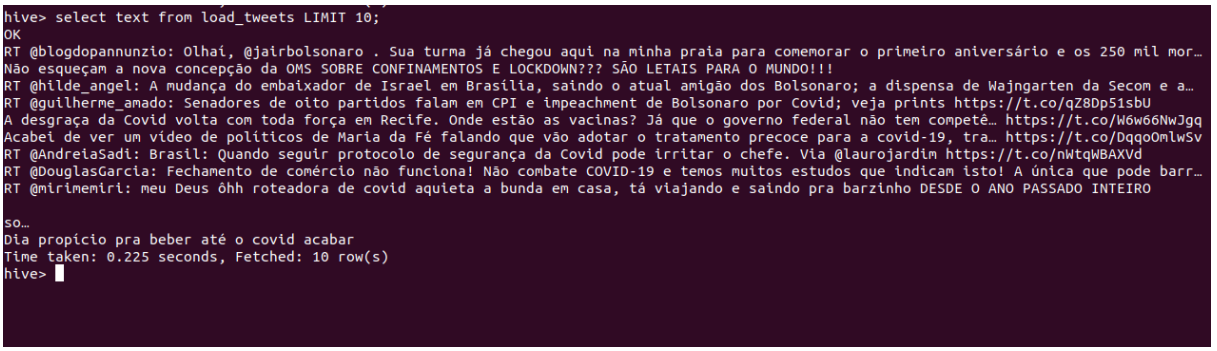
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>

<property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://localhost:3306/metastore?createDatabaseIfNotExist=true</value>
    <description>JDBC connect string for a JDBC metastore</description>
</property>

<property>
  <name>javax.jdo.option.ConnectionDriverName</name>
  <value>com.mysql.jdbc.Driver</value>
</property>

<property>
  <name>javax.jdo.option.ConnectionUserName</name>
  <value>root</value>
</property>

<property>
  <name>javax.jdo.option.ConnectionPassword</name>
  <value>mysql</value>
</property>

<property>
  <name>spark.sql.warehouse.dir</name>
  <value>hdfs://localhost:9000/user/hive/warehouse</value>
</property>

</configuration>
```

- Figura 2.4 – Arquivo de configuração hive-site.xml

Este arquivo contém as propriedades de conexão ao banco de dados *MySQL* utilizado pelo *Apache Hive* para armazenar os metadados, como suas tabelas e partições, contido em um *schema* denominado *metastore.* Também foi necessário adicionar o *driver* do *MySql (mysql-connector-java-5.1.28.jar)* no diretório *lib* existente na instalação do *Apache Hive* para que fosse possível realizar a conexão. Outra propriedade existente é o parâmetro "*spark.sql.warehouse.dir",* que indica o endereço do serviço de acesso ao *HDFS* do *Apache Hadoop.*

Durante o processo de configuração, obtive uma mensagem de erro [13] ao tentar executar o *Apache Hive.* Esse problema ocorria pelo fato do *schema metastore* não estar presente no *MySQL database*. A solução foi executar o comando: *"schematool --dbType mysql –initSchema".*

Após a realização das configurações, foi possível visualizar os dados no *Apache Hive* com sucesso.



- Figura 2.5 – Instrução HQL obtendo 10 registros do HDFS.

Finalmente, a estrutura dos dados ficou da seguinte forma:

Variável	Descrição	Tipo
id	Identificador de cada *tweet* realizado.	String
text	Conteúdo do *tweet* publicado.	String
created_at	Data e hora da publicação do tweet.	String
screen_name	Nome de usuário no *Twitter*.	String
location	Localização do usuário.	String

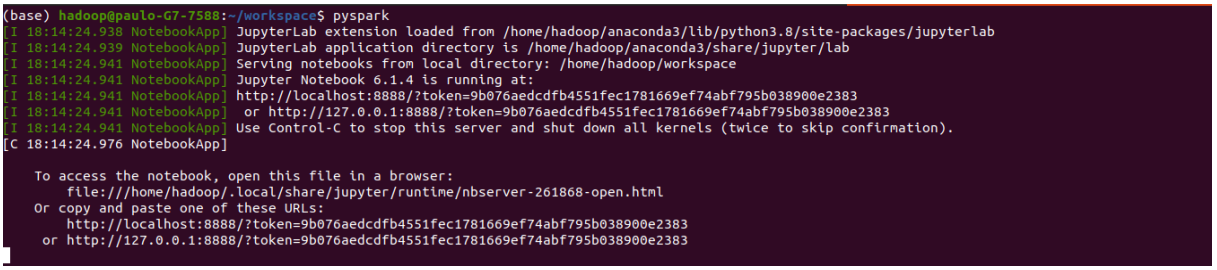
- Figura 2.6 – Variáveis definidas a partir dos dados coletados.

3. Processamento/Tratamento de Dados

Para o processamento dos dados obtidos, foi utilizado o *Apache Spark* na versão 3.1.1 [16] por ser um *framework* que trabalha com processamento em memória, ao contrário do *Apache Hadoop* que gera processos em *batch.* Trabalhando de forma integrada com o *Apache Hadoop,* ele provê uma interface chamada *PySpark,* permitindo assim utilizar a linguagem *Python* [17] para criar programas *Spark*. Como mecanismo importante no processo foi utilizado a plataforma *Anaconda* na versão *3* [18] por possuir integração com o *Apache Spark* e ser uma IDE de desenvolvimento em *Python*.

Após a instalação do *Apache Hadoop* e do *Anaconda*, foi possível iniciar a *IDE* de desenvolvimento ao executar o comando: *pyspark.*

Abaixo a figura demonstra o processo *Jupyter notebook* do *Anaconda* em execução.



- Figura 3.1 - Iniciando o Jupyter do Anaconda por meio do PySpark.

Felizmente, por meio da integração do *Apache Spark* com o *Apache Hadoop*, foi possível obter os dados do *HDFS* de forma simplificada utilizando a mesma sintaxe de consulta utilizada no *Apache Hive.* Basicamente, após gerar um contexto no *Apache Spark*, foi possível executar uma instrução de consulta e criar um *dataframe* do *Pandas* [15] com todos os registros obtidos. A figura a seguir demonstra um trecho do programa implementado em *Python* sendo executado na plataforma *Anaconda*.

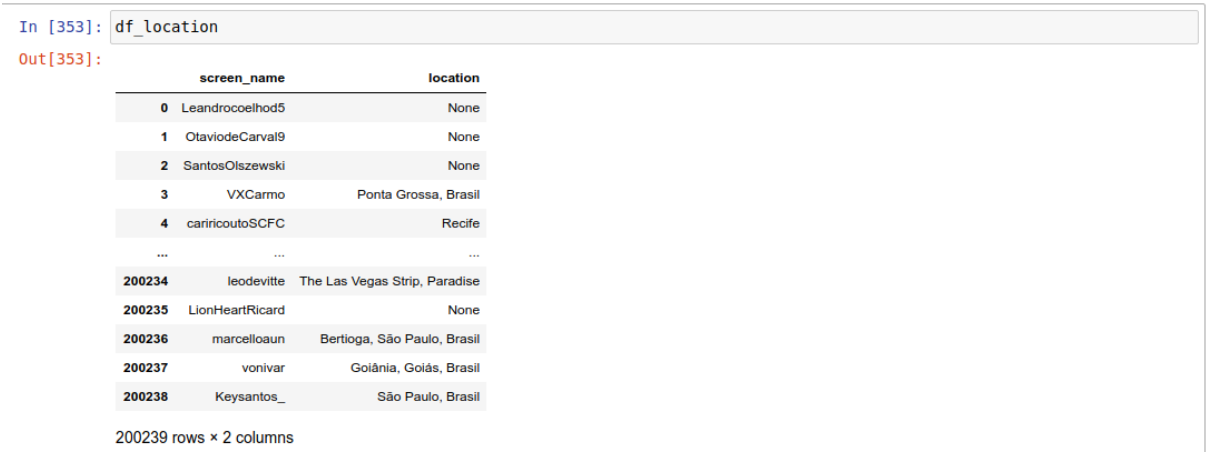
Criação do contexto do Spark e obtenção dos dados no HDFS utilizando sintaxe HQL

```
In [76]: conf = pyspark.SparkConf()
         sc = pyspark.SparkContext.getOrCreate(conf=conf)
         sqlcontext = SQLContext(sc)

In [77]: df_load = sqlcontext.sql("select id, `user`.screen_name, text, `user`.location, created_at from load_tweets").toPan

In [78]: df_load.count()

Out[78]: id             1374597
         screen_name    1374597
         text           1374597
         location        886003
         created_at     1374597
         dtype: int64

In [5]: period_list = pd.to_datetime(df_load.created_at).dt.date.unique()

In [6]: for d in period_list:
            print(d.strftime('%d/%m/%Y'))

         28/02/2021
         13/03/2021
         14/03/2021
         15/03/2021
         16/03/2021
         17/03/2021
         18/03/2021
         19/03/2021
```

- Figura 3.2 – Código em Python importando os tweets.

Conforme ilustrado na imagem acima, foram coletados cerca de 1.374.597 *tweets* nas datas: 28/02/2021, 13/03/2021, 14/03/2021, 15/03/2021, 16/03/2021, 17/03/2021, 18/03/2021 e 19/03/2021. A diversidade de datas de coleta foi propositalmente contemplada a fim de evitar possíveis sazonalidades entre os dias da semana, e contemplar dias de final de semana, onde a concentração de comentários é mais relevante.

De forma complementar, também foi utilizado um *dataset* obtido no *Kaggle* [14] com as informações das cidades brasileiras. O objetivo do uso desse *dataset* é validar os dados obtidos do *Twitter*. Dentre os campos do *dataframe*, *o location* responsável pela informação da localidade do usuário, recebeu um tratamento no seu dado, pois esse é um campo presente na biografia do usuário e nem sempre condiz com a realidade. Em uma amostra inicial, foi possível observar que esse campo não possui um padrão, onde em alguns casos contém o preenchimento com "País, Estado, Cidade", em outros casos "País, Cidade" e até mesmo com dados fictícios ou nulos.



```
In [353]: df_location
Out[353]:
```

	screen_name	location
0	Leandrocoelhod5	None
1	OtaviodeCarval9	None
2	SantosOlszewski	None
3	VXCarmo	Ponta Grossa, Brasil
4	cariricoutoSCFC	Recife
...	...	...
200234	leodevitte	The Las Vegas Strip, Paradise
200235	LionHeartRicard	None
200236	marcelloaun	Bertioga, São Paulo, Brasil
200237	vonivar	Goiânia, Goiás, Brasil
200238	Keysantos_	São Paulo, Brasil

200239 rows × 2 columns

- Figura 3.3 – Dados de localização dos usuários

Para tratar esse dados de localização, foi adotada a estratégia de criar uma nova coluna chamada "*city*" com o conteúdo delimitado de "*location*", pois apenas o dado presente a partir do início da sentença até a primeira vírgula era importante para definir a cidade do usuário. Para tal, foi usado o pacote de manipulação de dados *Pandas*. Basicamente, a partir dos dados da coluna "*location*" de cada linha, por meio da função *split* foi gerado um *array* baseado no delimitador "virgula". Como

resultado dessa operação, obtivemos o dado da primeira posição do *array* de cada linha processada e todo esse resultado foi vinculado a um novo *dataframe* chamado *df_cities.* Na sequência foi aplicado a função de *uppercase* para que todos esses dados ficassem em letra maiúscula.

No passo seguinte esse tratamento foi adicionado como uma nova coluna ao *dataframe* inicial, conforme figura 3.4.

```
In [378]: df_cities = df_location['location'].str.split(',').str[0]

In [386]: df_location['city'] = df_cities.str.upper().copy()

In [380]: df_location

Out[380]:
```

	screen_name	location	city
0	Leandrocoelhod5	None	None
1	OtaviodeCarval9	None	None
2	SantosOlszewski	None	None
3	VXCarmo	Ponta Grossa, Brasil	PONTA GROSSA
4	cariricoutoSCFC	Recife	RECIFE
...	...	...	...
351327	MAGNODO32268484	None	None
351328	_rlorena2	Belford Roxo, Brasil	BELFORD ROXO
351329	Roberto34914681	None	None
351330	ODilkinson	Gramado, Brasil	GRAMADO
351331	zerobarasuol	None	None

351332 rows × 3 columns

- Figura 3.4 – Criação da coluna city com a informação das cidades.

Na sequência, foi utilizado o *dataset* obtido do *Kaggle* com os dados de todas as cidades brasileiras. A partir deste arquivo, foi gerado um novo *dataframe* chamado “df_br_cities” para que esses dados pudessem ser utilizados na validação da informação das cidade dos usuários. O próximo passo foi criar uma lista chamada “*br_cities_list*”. Por fim, foi aplicado uma função *lambda* que avaliou todos os dados da coluna “*city*”, validando se ela estava presente na lista “*br_cities_list*”. Caso estivesse, então a informação era atribuída a uma nova coluna chamada “*validated_city*”, caso contrário, essa nova coluna era preenchida por “*None*”.

Desse modo, toda informação presente na coluna “*validated_city*” passou a ser condizente com as informações verídicas das cidades Brasileiras, conforme imagem 3.5.

```
In [381]: df_br_cities = pd.read_csv("data/BrazilianCities.csv")

In [382]: br_cities_list = df_br_cities["Cidade"].values.tolist();

In [383]: df_location['validated_city'] = df_location['city'].apply(lambda x: x if x in cidades_list else None)

In [384]: df_location
```

Out[384]:

	screen_name	location	city	validated_city
0	Leandrocoelhod5	None	None	None
1	OtaviodeCarval9	None	None	None
2	SantosOlszewski	None	None	None
3	VXCarmo	Ponta Grossa, Brasil	PONTA GROSSA	PONTA GROSSA
4	cariricoutoSCFC	Recife	RECIFE	RECIFE
...	...	...	...	...
351327	MAGNODO32268484	None	None	None
351328	_rlorena2	Belford Roxo, Brasil	BELFORD ROXO	BELFORD ROXO
351329	Roberto34914681	None	None	None
351330	ODilkinson	Gramado, Brasil	GRAMADO	GRAMADO
351331	zerobarasuol	None	None	None

- Figura 3.5 – Coluna validated_city com os dados validados

A fim de reduzir a matriz esparsa, foram realizados tratamentos nos *tweets* visando eliminar dados desnecessários para posterior aplicação do modelo estatístico. De forma auxiliar, foi utilizado uma biblioteca de processamento de linguagem natural chamada *NLKT* [19]. A sua utilização é de grande importância, pois provê ferramentas necessárias para a manipulação de texto de forma descomplicada quando aplicadas em nível de máquina [20]. A sua instalação se deu através do *Jupyter utilizando os comandos:*

!pip install nltk e nltk.download('all').

De forma complementar, algumas importantes funções foram implementadas, conforme se observa na figura 3.6.

```
In [45]: def remove_links(df):
             pattern=r'(?i)\b((?:[a-z][\w-]+:(?:/{1,3}|[a-z0-9%])|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-z]{2,4}/)(?:[^\s()<>]+|\((
             return df.replace(pattern, '', regex=True)

In [5]:  def only_letters(df):
             df = to_lowercase(df)
             df = remove_accentuation(df)
             return df.replace('[^a-zA-Z\\s]', '', regex=True)

In [7]:  portuguese_stops = stopwords.words('portuguese')

         def remove_stopwords(df):
             return df.apply(lambda tweet: ' '.join([word for word in tweet.split() if word not in (portuguese_stops)]))

In [8]:  def to_lowercase(df):
             return df.apply(lambda tweet: tweet.strip().lower())

In [9]:  def remove_accentuation(df):
             return df.apply(lambda tweet: unidecode(tweet))

In [113]: def remove_small_content(df):
              return df.apply(lambda tweet: ' '.join([word for word in tweet.split() if len(word)>=3]))
```

- Figura 3.6 – Funções para tratamentos dos tweets.

A primeira delas, com nome “remove_links”, remove qualquer tipo de endereço de domínio (*URL*) dos *tweets*. Essa função foi necessária, pois muitos usuários compartilham endereços de domínio, também conhecido como URL em suas postagens, fato que não agrega qualquer conteúdo relevante para o modelo estatístico. A segunda função chamada “only_letters” remove qualquer tipo de caractere diferente do esperado, sendo assim, qualquer dado que não seja uma palavra é removido do conteúdo. A terceira função faz uso da biblioteca *NLKT* para remover os chamados *stopwords*. Por definição, *stopwords* são palavras que podem ser consideradas irrelevantes para o conjunto de resultados a ser exibido em uma busca realizada em uma *search engine* [21]. Um exemplo de *stopwords são os conectores verbais* “de”, “para” e “mas”. Devido a alta frequência desses termos, a não remoção dos mesmos poderia impactar sensivelmente o resultado final do modelo de estatístico. A quarta função denominada “*to_lowercase”* transforma todo o conteúdo dos *tweets* para letras minúsculas, permitindo assim que palavras com o mesmo significado não sejam interpretadas de forma diferente. A quinta função remove qualquer tipo de acentuação do conteúdo e a última remove qualquer dado que contenha menos de três caracteres. Como último passo, foi aplicada uma validação para remover do *dataframe* qualquer *tweet* sem nenhum conteúdo. Na figura 2.10 é exibido uma amostra dos *tweets* antes e depois da execução das funções supracitadas.

```
In [121]: df_tweets = df_processed.copy()

In [122]: df_tweets['text']

Out[122]: 1          Não esqueçam a nova concepção da OMS SOBRE CON...
          4          A desgraça da Covid volta com toda força em Re...
          5          Acabei de ver um vídeo de políticos de Maria d...
          9                  Dia propício pra beber até o covid acabar
          10         Esse mofino presume q governadores e prefeitos...
                                                 ...
          1374586    minha mãe vai fazer o teste de covid eu tô com...
          1374587    Agora mais do que nunca estou com ódio da covi...
          1374588    😰😰 gente tô ficando com muito mais medo da cov...
          1374591    eu tô chocada real porque parei pra pesquisar ...
          1374596    É bizarro ver que nos dias de hoje ainda tem g...
          Name: text, Length: 461128, dtype: object

In [123]: df_tweets['text'] = remove_stopwords(df_tweets['text'])
          df_tweets['text'] = remove_links(df_tweets['text'])
          df_tweets['text'] = only_letters(df_tweets['text'])
          df_tweets['text'] = remove_small_content(df_tweets['text'])

In [124]: ## Removendo todos os tweets vazios após tratamento
          df_tweets = df_tweets[df_tweets['text'].str.len() > 0]

In [125]: df_tweets['text']

Out[125]: 1          nao esquecam nova concepcao oms sobre confinam...
          4          desgraca covid volta toda forca recife onde va...
          5          acabei ver video politicos maria falando vao a...
          9                        dia propicio pra beber covid acabar
          10         esse mofino presume governadores prefeitos faz...
                                                 ...
          1374586                      mae vai fazer teste covid medo
          1374587                agora nunca odio covid quero ver meu
          1374588                      gente ficando medo covid antes
          1374591    chocada real porque parei pra pesquisar agora ...
          1374596    bizarro ver dias hoje ainda gente acha terra p...
          Name: text, Length: 460755, dtype: object
```

- Figura 3.7 – Tweets antes e depois das funções aplicadas.

4. Análise e Exploração dos Dados

A rede social *Twitter* é notoriamente uma ferramenta de compartilhamento de informação entre os usuários da plataforma. Nela existe uma funcionalidade em que um usuário pode enviar novamente um *tweet* gerado por outra pessoa. Nessa situação, esse conteúdo é chamado de *retweet*. Querendo ter um panorama a respeito deste ponto, foi realizado um filtro no *dataframe* com os *tweets* obtidos, gerando assim dois novos *dataframe* a partir do original. Em um deles foi incluído todo *tweet* em que o texto publicado não iniciava com o termo “RT”, ou seja, não identificável como um *retweet*. Dessa forma, criamos um novo *dataframe* chamado “df_load_cleaned” com esse conteúdo.

Filtro dos dados no qual o texto publicado inicia com "RT" (Retweets)

```
In [89]: df_load_dirt = df_load[df_load.text.str.startswith('RT')]

In [90]: df_load_dirt.head(5)
```

Out[90]:

	id	screen_name	text	location	created_at
0	1366032532014309379	Leandrocoelhod5	RT @blogdopannunzio: Olhaí, @jairbolsonaro . S...	None	Sun Feb 28 14:28:29 +0000 2021
2	1366032533524262912	SantosOlszewski	RT @hilde_angel: A mudança do embaixador de Is...	None	Sun Feb 28 14:28:30 +0000 2021
3	1366032533771735044	VXCarmo	RT @guilherme_amado: Senadores de oito partido...	Ponta Grossa, Brasil	Sun Feb 28 14:28:30 +0000 2021
6	1366032537701781505	gustacioli	RT @AndreiaSadi: Brasil: Quando seguir protoco...	São Paulo, Brasil	Sun Feb 28 14:28:31 +0000 2021
7	1366032539002023943	LLLGCR	RT @DouglasGarcia: Fechamento de comércio não ...	None	Sun Feb 28 14:28:31 +0000 2021

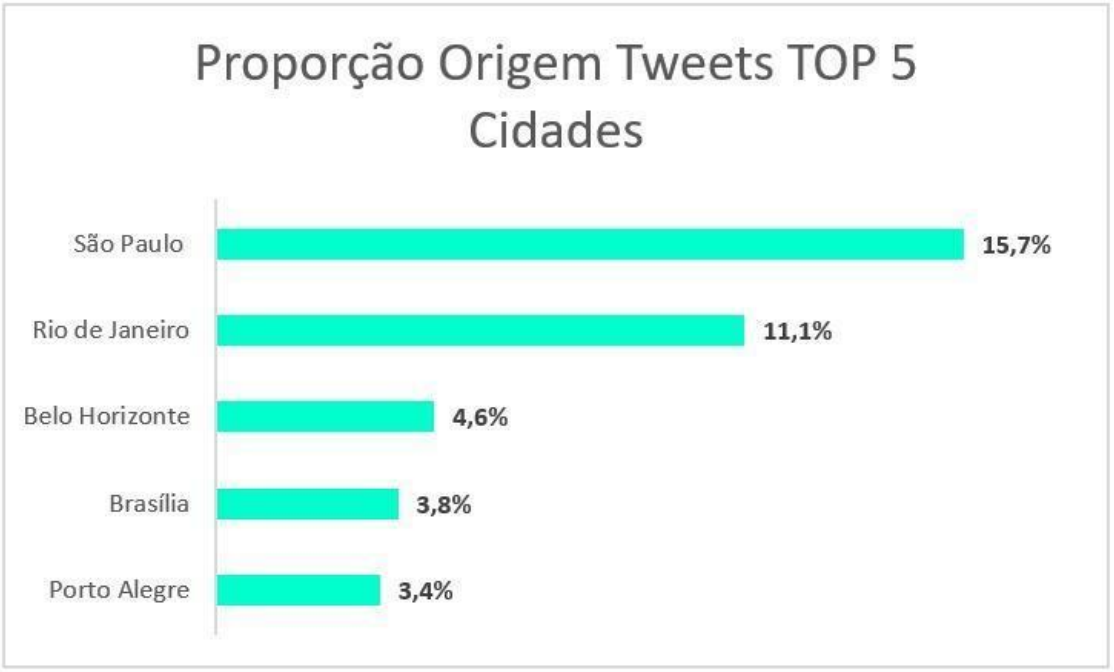
```
In [91]: df_load.count()
Out[91]: id             1374597
         screen_name    1374597
         text           1374597
         location        886003
         created_at     1374597
         dtype: int64

In [92]: df_load_dirt.count()
Out[92]: id             913469
         screen_name    913469
         text           913469
         location       575734
         created_at     913469
         dtype: int64
```

- Figura 4.1 – Código que analisa a quantidade de tweets únicos

A partir da contagem dos dados coletados, foi possível identificar que apenas 33,54 % dos *tweets* são únicos, ou seja, grande parte dos usuários da rede social preferem enviar novamente uma publicação quando o assunto em questão é o *Covid*. Isso também pode ser interpretado como o efeito contagiante na rede, em que um comentário acaba sendo retweetado múltiplas vezes, disseminando com muita facilidade percepções e expectativas.

- Figura 4.2 – Proporção dos tweets únicos relacionados ao Covid.

Outro *insight* obtido foi entender em quais cidades do país ocorreu um maior volume de *tweets* sobre o *Covid* no período coletado. A partir do *dataframe* com os *tweets* únicos, foi obtido a quantidade de publicações agrupadas pelas cidades dos usuários. Com base nos dados obtidos, as cinco cidades com maior quantidade de publicações representam uma concentração de 38,19 % do total.



- Figura 4.3 – As cinco com maior volume de tweets.

Outro ponto analisado, foi verificar a frequência das palavras publicadas nos *tweets*. Para essa análise, foram utilizadas as *features TweetTokenizer* e *FreqDist* disponibilizadas pela biblioteca *NLKT* [22]. O *TweetTokenizer* provê uma interface de processamento de linguagem natural que extrai palavras deixando-as de forma separada a partir de um conteúdo. Basicamente, a ideia foi gerar um *array* de palavras a partir dos *tweets* coletados. Após obter o *array*, foi aplicado uma simples verificação eliminando as palavras menores que três caracteres, pois não agregaram significamente para esse contexto. Dessa forma, foi possível aplicar a função *FreqDist* que teve a responsabilidade de classificar a frequência de cada item presente nesse *array*. Como resultado, foi obtido um *dataframe* com essa frequência computada. Dentre as dez palavras mais frequentes, encontram-se os termos os termos: “covid”, “corona”, “brasil”, “gente”, “contra”, “mortes”, “pessoas”, “deus”, “vacina” e “hoje”. A partir dessas informações fica caracterizado o anseio e preocupação em relação ao tema. Para exemplificar esse contexto foi utilizado a biblioteca *WordCloud* [23] para criar uma nuvem de palavras. Uma nuvem de palavras é uma forma de representação de frequência dentro de um contexto. De forma simples, quanto mais frequente uma palavra, maior destaque ela terá nessa

representação. A figura abaixo contém a nuvem de palavras a partir dos dados coletados.

- Figura 4.5 – Wordcloud obtido a partir dos dados.

5. Criação de Modelos de Machine Learning

Devido ao poder computacional atual, foi possível analisar, tratar e remover dados de forma que pudesse facilitar o propósito de gerar informação e obter conhecimento. Contudo, existem tarefas que podem ser simples para um ser humano, mas extremamente custosas para um computador. Porém, devido ao enorme volume de dados obtido, torna-se inviável que uma pessoa realize todo o trabalho individualmente. Para essa etapa do projeto, em que necessitamos compreender todo esse dado, foi utilizado um algoritmo de clusterização.

Clusterização é o processo computacional capaz de agrupar dados de forma que contém alguma similaridade, em um número *K* grupos geralmente denominado *cluster* [24]. Essa técnica representa uma das principais etapas de processos de análise de dados, denominada análise de *clusters*. A análise de *clusters* envolve, portanto, a organização de um conjunto de padrões (usualmente representados na forma de vetores de atributos ou pontos em um espaço multidimensional) em

clusters, de acordo com alguma medida de similaridade. Intuitivamente, padrões pertencentes a um dado *cluster* devem ser mais “parecidos” entre si do que em relação a padrões pertencentes a outros *clusters*.

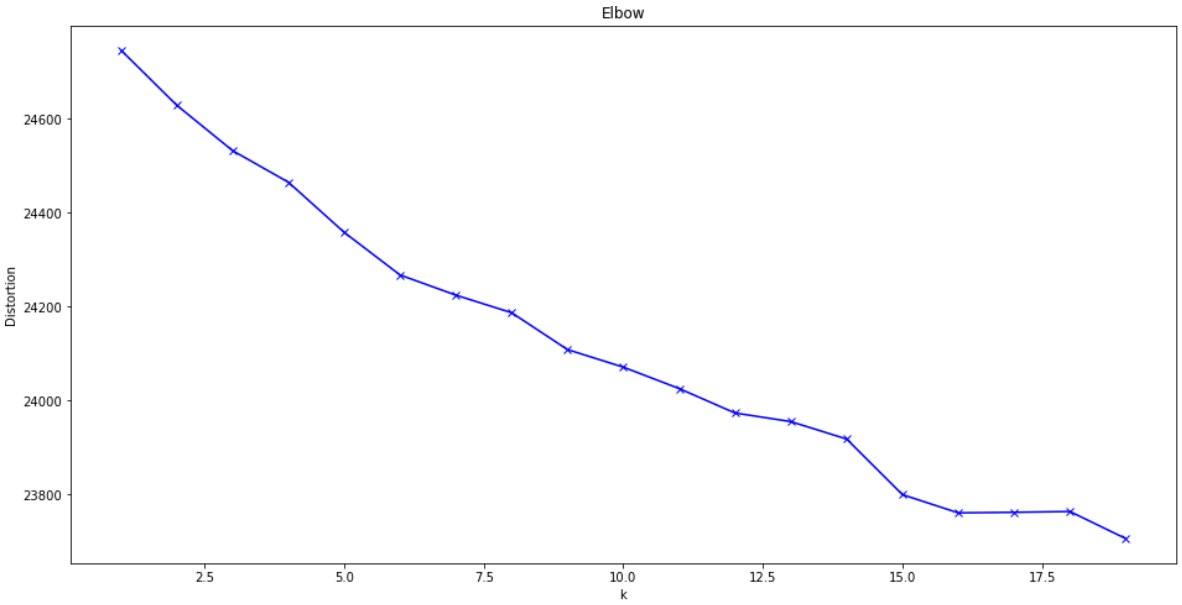
Existem dois tipos de clusterização. A classificação não-supervisionada e a análise discriminante - supervisionada:

• Na classificação supervisionada, são fornecidos padrões rotulados (pré-classificados) e o problema é rotular novos padrões, ainda não rotulados.

• Na classificação não-supervisionada, o problema é agrupar um conjunto de padrões não rotulados em conjuntos que possuam algum significado, ou seja, de tal modo que os padrões apresentem alguma propriedade comum. Sendo assim, uma vez definidos os *clusters*, os padrões também estão “rotulados”, mas o rótulo aqui é ditado pelos próprios padrões que compõem cada conjunto.

Foram realizados experimentos de clusterização não supervisionada de *tweets*, buscando-se verificar a eficácia dos agrupamentos encontrados.

Após o pré-processamento dos dados, dando início à análise, foi plotado o gráfico de *Elbow* a fim de determinar o número de *clusters* ideal que os *tweets* deveriam ser segmentados. O Método de *Elbow* tem esse nome por gerar um gráfico com o formato parecido ao de um "braço", onde procuramos o “cotovelo” para definir um número aceitável de K (*clusters*) a serem criados com base nos dados da amostra. Este método vai crescendo a quantidade de *clusters* a partir de 1 e analisando a melhora do resultado a cada iteração. Quando o benefício parar de ser relevante (um salto entre uma quantidade de *cluster* e a próxima quantidade) ele entra em um modelo platô, no qual a diferença da distância é quase insignificante. E neste momento que entende-se que encontramos uma quantidade K mais interessante. Em um intervalo de 1 a 20, o gráfico sugere que a base deveria ser dividida em cerca de 10 *clusters*. Contudo, foram realizadas várias tentativas e observados os resultados. Na prática, obtiveram se grupos com maior coesão de similaridade e poder de interpretação dividindo-se o conjunto de dados em 9 *clusters*.



- Figura 5.1 – Gráfico de Elbow

Após pré-determinar a melhor divisão do conjunto de dados, optou-se por utilizar o método *k means*, devido a grande oferta de materiais disponíveis e por ser o método de clusterização mais disseminado para problemas complexos de processamento de linguagem natural, *NLP*. Este algoritmo se chama assim pois encontra *k clusters* diferentes no conjunto de dados. O centro de cada *cluster* será chamado centróide e terá a média dos valores neste *cluster*.

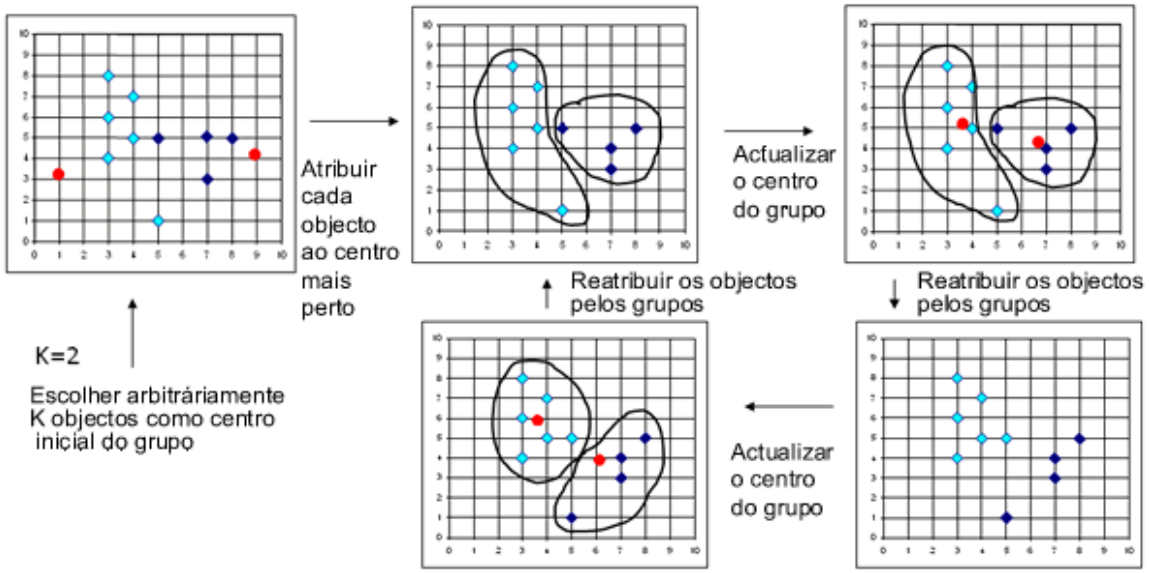
A tarefa do algoritmo é encontrar o centróide mais próximo (por meio de alguma métrica de distância) e atribuir o ponto encontrado a esse *cluster*. Após este passo, os centróides são atualizados sempre tomando o valor médio de todos os pontos naquele *cluster*. Para este método são necessários valores numéricos para o cálculo da distância, os valores nominais então podem ser mapeados em valores binários para o mesmo cálculo. Em caso de sucesso, os dados são separados organicamente podendo assim ser rotulados e centrais viram referência para classificar novos dados.

O algoritmo é repetido inúmeras vezes nos seguintes passos até que se estabilize:

- Associação de cada instância a um centróide - Cada centróide define um *cluster*, então cada instância será associada a seu *cluster* mais semelhante (centróide mais próximo). A distância será calculada por alguma métrica de distância, em geral utiliza-se a distância euclidiana entre as duas instâncias;

- Atualização dos centróides - Centróides dos *clusters* são recalculados, a partir da média entre todas as instâncias associadas àquele *cluster*.

Na figura abaixo é possível elucidar o processo iterativo de otimização e cálculo de distância Euclidiana em relação aos centróides.



- Figura 5.2 – Representação gráfica de otimização de centróides [25]

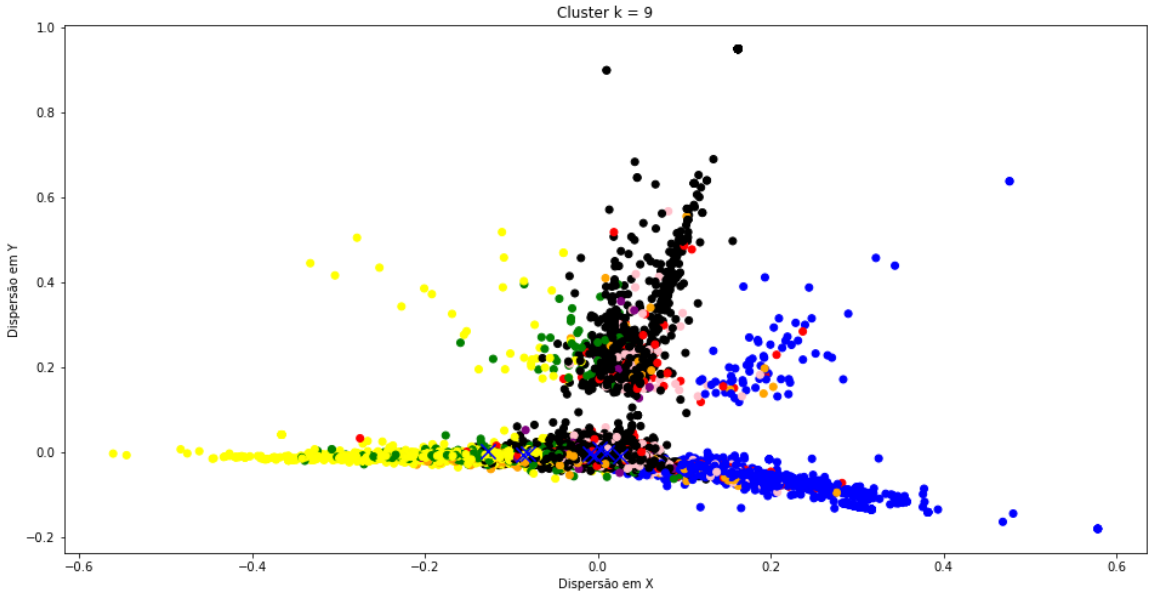
Devido a complexidade e alto volume de dados, foram encontrados alguns desafios relevantes no processo de implementação. Mesmo utilizando um método de redução de dimensionalidade de variáveis, o PCA, que tem por objetivo condensar e resumir múltiplas variáveis em um número menor delas que seja capaz de carregar quantidades semelhantes de informação, computacionalmente foi inviável o processamento da totalidade do conjunto de dados extraído, devido a escassez de memória computacional.

Em face de tal problema, foi retirada amostra aleatória dos dados, por meio do método *sample* presente na biblioteca *Pandas,* totalizando 25 mil exemplares. A escolha do tamanho da amostra se deu pelo limite da capacidade computacional presente. Todos os resultados que serão reportados a seguir são oriundos desta amostra.

6. Apresentação dos Resultados

Devido a complexidade e alto volume de dados, mesmo utilizando apenas uma amostra aleatória do conjunto de dados original, ao plotar o gráfico com a divisão dos *clusters* a imagem não fica claramente dividida em grupos de cores, como habitualmente se encontram exemplos em materiais didáticos. Nota-se a sobreposição de pontos e falta de clareza na comparação de cores.

De qualquer forma, é possível notar tendências claras de comportamento, destacando-se os *clusters* de cor preta, amarela e azul, tanto pela quantidade quanto pela posição no plano cartesiano.



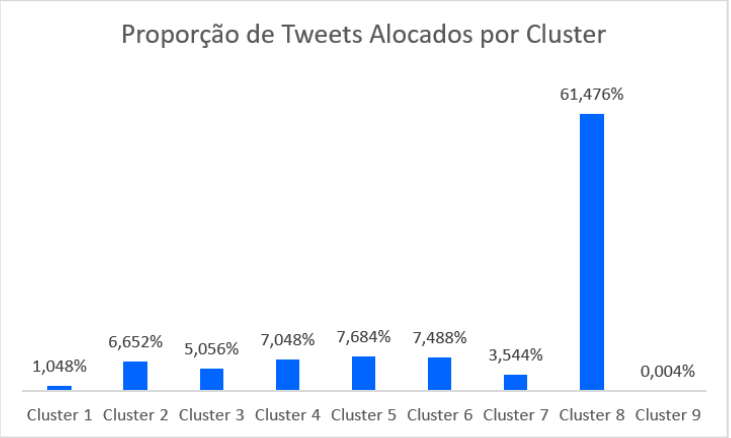
- Figura 6.1 – Representação gráfica dos K-clusters.

No contexto de NLP, a análise e interpretação dos resultados pode ser mais clara entendendo as frases que foram alocadas em *clusters* semelhantes, na busca

de sentidos semânticos comuns. Para isso, foi feita análise da amostra de 25 mil *tweets* com seus respectivos *clusters* flegados.

Primeiramente, notou-se uma distribuição bastante desbalanceada na quantidade de *tweets* alocados por *cluster*. Mais de 61% dos exemplos foram alocados no *cluster* 8, múltiplos *clusters* ficaram com baixa frequência, e o *cluster* 9 ficou com apenas 1 *tweet*, provavelmente *outlier*.

Nº do Cluster	Qtd Tweets	Proporção
Cluster 1	262	1,048%
Cluster 2	1.663	6,652%
Cluster 3	1.264	5,056%
Cluster 4	1.762	7,048%
Cluster 5	1.921	7,684%
Cluster 6	1.872	7,488%
Cluster 7	886	3,544%
Cluster 8	15.369	61,476%
Cluster 9	1	0,004%



- Figura 6.2 – Representação da proporção dos tweets por cluster.

Com isto posto, mesmo identificando-se a necessidade de reprocessar o algoritmo com outras quantidades de “k” experimentalmente, foi plotado as 10 palavras mais frequentes de cada um dos grupos, a fim de ter insumo para as primeiras conclusões. O resultado destas palavras mais frequentemente repetidas foi inserido na tabela abaixo:

	cluster_0	cluster_1	cluster_2	cluster_3	cluster_4	cluster_5	cluster_6	cluster_7	cluster_8
0	onde	covid	covid	covid	corona	covid	covid	covid	mortes
1	covid	morre	corona	mortes	virus	corona	anos	contra	causadas
2	onda	morreu	pode	brasil	ansiedade	casa	passado	deus	falhas
3	brasil	major	gente	morte	cura	fazer	corona	gente	oxigenio
4	corona	olimpio	nada	mortos	pressao	gente	pandemia	vacina	covid
5	mundo	morrer	existe	casos	depressao	minha	hoje	brasil	causam
6	nova	pessoas	pessoas	recorde	alta	todo	contra	pessoas	revolta
7	mortes	senador	deus	horas	baixa	aqui	casa	agora	jordania
8	segunda	morrendo	ainda	registra	aids	pegar	brasil	aqui	mundo
9	parar	morreram	aguento	ultimas	presao	pessoas	pessoas	bolsonaro	

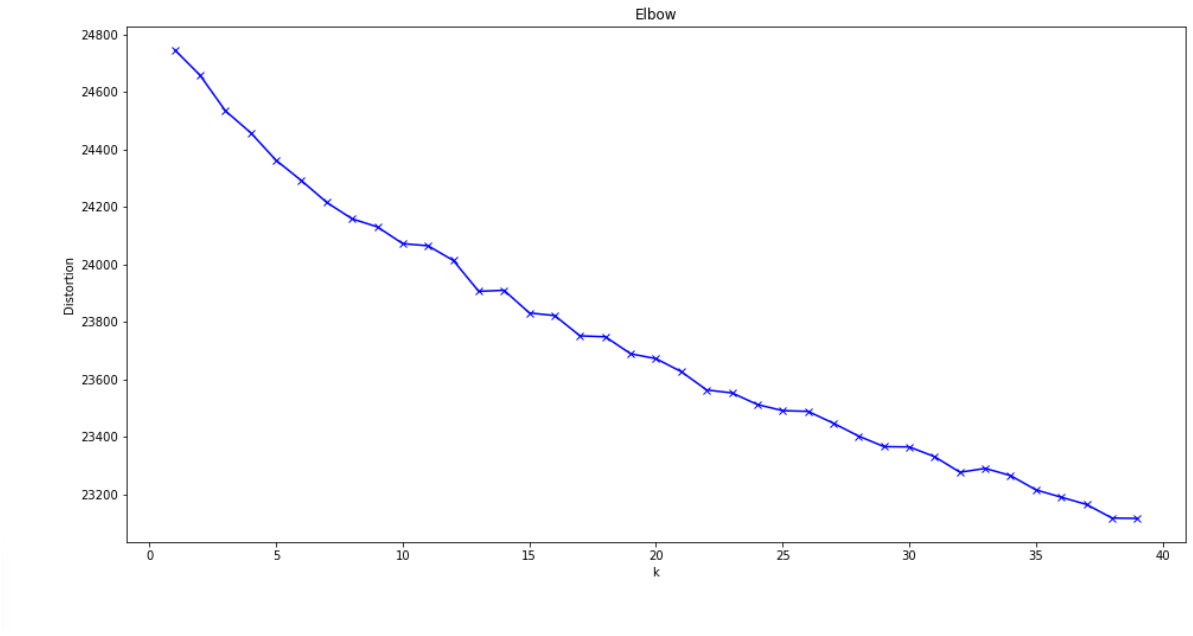
- Figura 6.3 – As dez palavras mais frequentes para cada cluster.

Ao avaliar a imagem 6.3, assuma que o *cluster* denominado como “cluster_0” trata-se do primeiro *cluster*, assim por diante. Em uma primeira análise partindo das principais palavras conjuntamente a leitura amostral aleatória de alguns *tweets*, já se podem tirar algumas conclusões:

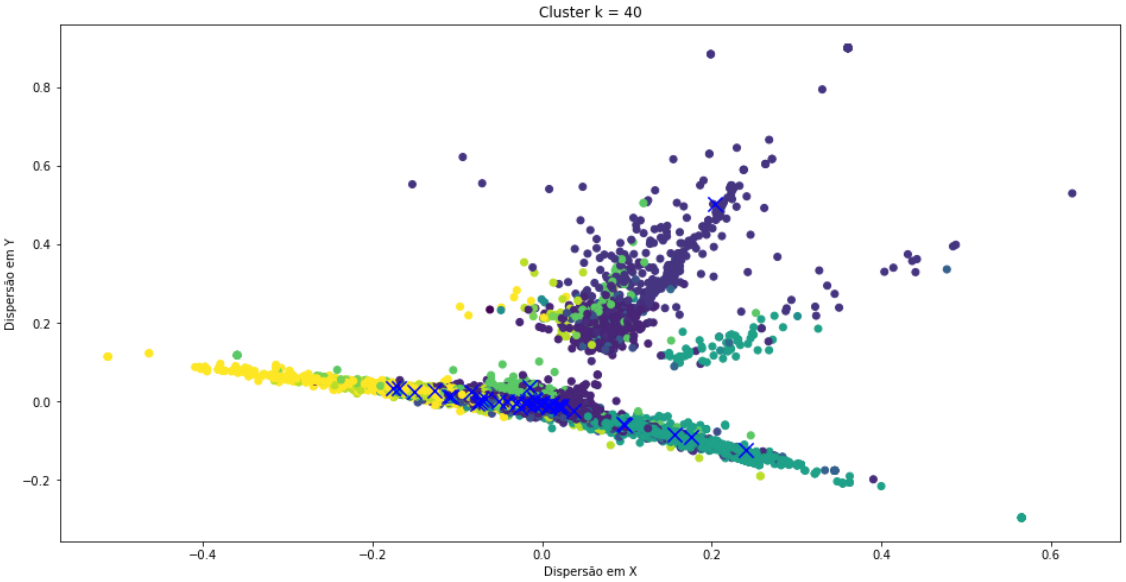
- Cluster 1 - Frases relacionadas a nova variante da doença;
- *Cluster* 2 - Comentários relacionados a doença e morte do Senador Major Olímpio, ocorrida em 18 de março (data compatível às extrações de bases);
- *Cluster* 3 - Frases relacionadas a exaustão, cansaço mediante a triste situação da doença no Brasil;
- *Cluster* 4 - Índices de mortalidade, recorde de morte no país;
- *Cluster* 5 - Estado emocional, stress, depressão, exaustão;
- *Cluster* 6 - Longo período de permanência na quarentena, impossibilidade de seguir as rotinas fora de casa;
- *Cluster* 7 - Mais de um ano de isolamento, comparativo entre a primeira e segunda onda da doença no Brasil;
- *Cluster* 8 - Vacinação, lentidão no processo de imunização da população, posicionamento tardio do governo em providenciar as vacinas;
- *Cluster* 9 - Frase única, diz respeito a mortes causadas pela falta de oxigênio na Jordânia.

Foram executados múltiplos testes com números diferentes de divisão de *clusters*. Esse é um procedimento comum para algoritmos não supervisionados, em que não se tem um *target* disponível e por isso há bastante espaço para subjetividade até encontrar o cenário que possibilite uma melhor possibilidade para a interpretação de resultados, não havendo possibilidade de otimizar matematicamente um resultado ideal.

Realizado teste com 40 grupos, onde a interpretação visual da divisão dos agrupamentos fica ainda mais desafiadora. Para mais de 40 k, não foi possível processar devido a falta de memória computacional. Vê-se que não há comportamento de “cotovelo” no gráfico. De toda forma, os *clusters* foram analisados para interpretação.



- Figura 6.4 – Gráfico de Elbow com os 40 clusters.



- Figura 6.5 – Representação da dispersão dos 40 clusters.

Proporcionalmente, a divisão de *tweets* ficou ainda mais desbalanceada com a divisão em 40 grupos, com múltiplos grupos com apenas 1 *tweet*. Interessante notar inclusive a permanência de mais de 60% da base em apenas um *cluster*.

Número do Cluster	Qtd de Tweets	%
cluster 1	8	0,032%
cluster 2	2	0,008%
cluster 3	1	0,004%
cluster 4	1	0,004%
cluster 5	15649	62,596%
cluster 6	1052	4,208%
cluster 7	350	1,400%
cluster 8	1	0,004%
cluster 9	1	0,004%
cluster 10	1044	4,176%
cluster 11	1	0,004%
cluster 12	278	1,112%
cluster 13	44	0,176%
cluster 14	387	1,548%
cluster 15	1	0,004%
cluster 16	1	0,004%
cluster 17	3	0,012%
cluster 18	3	0,012%
cluster 19	1	0,004%
cluster 20	916	3,664%
cluster 21	1	0,004%
cluster 22	1	0,004%
cluster 23	1876	7,504%
cluster 24	1	0,004%
cluster 25	1	0,004%
cluster 26	1	0,004%
cluster 27	5	0,020%
cluster 28	1	0,004%
cluster 29	10	0,040%
cluster 30	489	1,956%
cluster 31	1	0,004%
cluster 32	284	1,136%
cluster 33	1	0,004%
cluster 34	1	0,004%
cluster 35	1	0,004%
cluster 36	1456	5,824%
cluster 37	1	0,004%
cluster 38	4	0,016%
cluster 39	1	0,004%
cluster 40	1120	4,480%

- Figura 6.6 – proporção dos tweets dentre os 40 clusters.

Com isto posto, conclui-se que a divisão de 9 grupos foi a mais satisfatória e teve maior sentido de interpretabilidade, mesmo tendo 1 *cluster* com apenas um *tweet* de *outlier*.

Finalmente, após análise de todas essas informações, percebe-se uma grande preocupação dos usuários da rede social em relação ao tema. É notório uma percepção bastante negativa e pouco esperançosa em relação ao contexto no período pesquisado. Esse padrão é observado principalmente no 8 *cluster,* tanto devido a sua relevância por possuir um maior volume em relação ao todo, quanto pela característica presente nos seus dados, pois existe uma grande expectativa e apreensão pelo avanço da vacinação da população além de toda evidente insatisfação em relação à situação. Inicialmente, foi considerado a utilização de um algoritmo para avaliar se a percepção era puramente negativa ou positiva em relação ao Covid, contudo, devido aos inestimáveis impactos na sociedade, percebeu-se que o resultado final não seria tão impactante quando comparado aos *insights* obtidos por meio da clusterização.

7. Links

Repositório do projeto:

https://github.com/prgmw/clusterizacao_tweets

Vídeo explicativo:

https://youtu.be/wdsdhwJchG0

Agradecimento:

Agradeço a todos os docentes que me permitiram alcançar um novo patamar de conhecimento. Em especial, agradeço a minha esposa Lígia Nardy de Vasconcellos que me apoiou em inúmeros momentos no qual me encontrava em dificuldade. Sua presença foi substancial para que eu chegasse até o presente momento.

REFERÊNCIAS

[1] RYDNING, John. **IDC's Global StorageSphere Forecast Shows Continued Strong Growth in the World's Installed Base of Storage Capacity**. Disponível em: <https://www.idc.com/getdoc.jsp?containerId=prUS46303920> Acesso: 01/03/2021

[2] HUTCHINSON, Andrew. **What Happens on the Internet Every Minute (2020 Version).** Disponível em: <https://www.socialmediatoday.com/news/what-happens-on-the-internet-every-minute-2020-version-infographic/583340/> Acesso: 01/01/2021

[3] **Por que a doença causada pelo novo coronavírus recebeu o nome de Covid-19?** Disponível em: <https://portal.fiocruz.br/pergunta/por-que-doenca-causada-pelo-novo-coronavirus-recebeu-o-nome-de-covid-19> Acesso: 01/01/2021

[4] **Twitter**. Disponível em: <https://developer.twitter.com/> Acesso: 01/01/2021

[5] **Apache Flume**. Disponível em: <https://flume.apache.org/releases/1.7.0.html/> Acesso: 03/01/2021

[6] **Cloudera**. Disponível em: <https://github.com/cloudera/cdh-twitter-example/>Acesso: 03/01/2021

[7] **Apache Maven**. Disponível em: <https://maven.apache.org/> Acesso: 10/12/2020

[8] **Apache Hadoop**. Disponível em: <https://hadoop.apache.org/docs/r3.1.3/> Acesso: 04/01/2021

[9] BROWN, Korbin. **Ubuntu 20.04 Hadoop**. Disponível em: <https://linuxconfig.org/ubuntu-20-04-hadoop> Acesso: 05/01/2021

[10] **Apache Hadoop.** Disponível em: <https://hadoop.apache.org/docs/r3.1.3/hadoop-project-dist/hadoop-common/SingleCluster.html> Acesso: 05/01/2021

[11] **Apache Hive**. Disponível em: <https://hive.apache.org/index.html/> Acesso: 05/01/2021

[12] **What is Apache Hive.** Disponível em: <https://www.ibm.com/analytics/hadoop/hive> Acesso: 05/02/2021

[13] **HiveException java.lang.RuntimeException: Unable to instantiate org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreClient.** Disponível em: <https://itectec.com/ubuntu/ubuntu-failed-hiveexception-java-lang-runtimeexception-unable-to-instantiate-org-apache-hadoop-hive-ql-metadata-sessionhivemetastoreclient/> Acesso: 06/01/2021

[14] TRINDADE, Gilberto. **Brazilian Cities.** Disponível em: <https://www.kaggle.com/gilbertotrindade/cidades-brasileiras> Acesso: 16/03/2021

[15] **Pandas**. Disponível em: <https://pandas.pydata.org/> Acesso: 16/03/2021

[16] **Apache Spark**. Disponível em: <https://spark.apache.org/releases/spark-release-3-1-1.html> Acesso: 10/02/2021

[17] **PySpark**. Disponível em: <https://spark.apache.org/docs/latest/api/python/> Acesso: 10/02/2021

[18] **Anaconda**. Disponível em: <https://www.anaconda.com/> Acesso: 11/02/2021

[19] **Natural Language Toolkit**. Disponível em: <https://www.nltk.org/> Acesso: 15/02/2021

[20] AURORA, Lady. **NLTK Tutorial.** Disponível em: <https://medium.com/@maelyalways/nltk-tutorial-8175e57fbfda> Acesso: 15/02/2021

[21] **Stop words.** Disponível em: <https://www.computerhope.com/jargon/s/stopword.htm> Acesso: 15/02/2021

[22] **Nltk Package.** Disponível em: <https://www.nltk.org/api/nltk.tokenize.html> Acesso: 15/02/2021

[23] **Wordcloud**. Disponível em: <https://pypi.org/project/wordcloud/> Acesso: 15/02/2021

[24] Ileoh**. Introdução A Clusterização E Os Diferentes Métodos.** Disponível em: <https://portaldatascience.com/introducao-a-clusterizacao-e-os-diferentes-metodos/> Acesso: 15/02/2021

[25] **K-médias.** Disponível em: <http://web.tecnico.ulisboa.pt/ana.freitas/bioinformatics.ath.cx/bioinformatics.ath.cx/index651a.html?id=147> Acesso: 20/02/2021