Desenvolvimento de Software para Nuvem
Profs.: Fernando Trinta e Paulo Antonio Rego

Relatório do Trabalho Prático Haddop

Pedro Almir Martins de Oliveira (434728)

Descrição do Ambiente

Para execução das aplicações[1] Hadoop desenvolvidas no contexto deste trabalho, utilizou uma instância Hadoop configurada sobre um Docker Windows por meio das imagens disponibilizadas no repositório https://github.com/big-data-europe/docker-hadoop. A máquina Windows utilizada possui as seguintes configurações: processador intel core i7-8565U, 16GB de memória RAM, HD de 1TB e SSD de 128GB, e Windows 10 de 64 bits. Quanto a configuração do Hadoop, optou-se pela configuração *default* do big-data-europe com 1 namenode, 1 datanote, 1 resourcemanager, 1 nodemanager e 1 historyserver (conforme Figura 1).

```
ppedr@DESKTOP-66D5LBD /cygdrive/d/dev/docker/docker-hadoop
$ docker-compose up -d --build
Creating network "docker-hadoop_default" with the default driver
Creating historyserver   ... done
Creating namenode        ... done
Creating resourcemanager ... done
Creating nodemanager     ... done
Creating datanode        ... done
```

Figura 1. Containers em execução após o comando `docker-compose up -d --build`.

A título de registro, a versão atualizada das imagens do Hadoop Docker, proposta por Nathan Johnson no Github[2], não funcionou localmente. Além disso, não foi possível a correta configuração do hadoop diretamente no windows por conta de dificuldades quanto a pacotes de atualizações que necessitam ser instalados para que o hadoop funcione nesse sistema operacional.

Para garantir a reprodutibilidade dos resultados apresentados a seguinte, disponibilizamos os comandos realizados desde o deploy do hadoop, até a execução dos jobs e coleta dos resultados. Para isso, basta acessar o link: https://github.com/pedroalmir/cloud_study/blob/master/hadoop/commads.txt. Além disso, os resultados brutos encontram-se em uma planilha do google que pode ser acessada com o link: https://docs.google.com/spreadsheets/d/101yKlcCiKP_UKdooVOYElfqMVa-IGt7grLNL2nIu2is. Por fim, destaca-se o uso do software Tableau 2020 na geração de algumas visualizações de dados.

---

[1]Projeto com jobs: https://github.com/pedroalmir/cloud_study/tree/master/hadoop/tweets-reviews-counter
[2] Docker-compose aprimorado: https://gist.github.com/nathan815/a938b3f7a4d06b2811cf2b1a917800e1

Questão 1. Dado um dataset de arquivos texto com o conteúdo de vários livros, deseja-se distribuir a tarefa para se descobrir:

Procedimento: para essa questão, criou-se um único job hadoop. O mapper associou a cada palavra um mapa com duas características dessa palavra: contagem e tamanho. Além disso, criou-se uma chave especial ("my-word-average") para contabilizar o tamanho de todas as palavras (Figura 2).

```
public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
        MapWritable map = new MapWritable();
        MapWritable map2 = new MapWritable();

        String token = itr.nextToken().trim();
        map.put(new Text("count"), new IntWritable(1));
        map.put(new Text("length"), new IntWritable(token.length()));
        context.write(new Text(token), map);

        map2.put(new Text("length"), new IntWritable(token.length()));
        context.write(new Text("my-word-average"), map2);
    }
}
```

Figura 2. Mapper para Questão 1.

No reducer, para cada palavra realizou-se a soma do atributo contagem, mantendo o atributo tamanho. Foi adicionado um condicional para lidar com a chave ("my-word-average"). Nesse último caso, realizou-se a soma dos tamanhos e uma posterior divisão pela quantidade de palavras registradas (Figura 3). Assim, com o mesmo job foi possível responder os dois itens dessa questão. Ressalta-se que não foi realizado nenhum tipo de pré-processamento para remover tokens compostos por caracteres especiais.

```
public void reduce(Text key, Iterable<MapWritable> values, Context context) throws IOException, InterruptedException {
    if(key.toString().equals("my-word-average")) {
        int vals = 0, sum = 0;
        for (MapWritable val : values) {
            sum += ((IntWritable) val.get(new Text("length"))).get();
            vals++;
        }

        context.write(new Text(String.join("\t", new String[] {key.toString(), sum + "", vals + "", (sum/vals)  + ""})), null);
    }else {
        int count = 0, length = 0;
        for (MapWritable val : values) {
            count += ((IntWritable) val.get(new Text("count"))).get();
            length = ((IntWritable) val.get(new Text("length"))).get();
        }
        context.write(new Text(String.join("\t", new String[] {key.toString(), count + "", length + ""})), null);
    }
}
```

Figura 3. Reducer para Questão 1.

a) Qual ou quais são as palavras com maior número de letras?

Resposta: Como as palavras foram obtidas por meio da classe StringTokenizer usando o espaço como separador, foram obtidas muitas palavras compostas apenas por caracteres especiais. Isso, inclusive, pode ser uma melhoria em versões futuras. Assim, utilizamos a regex ([a-zA-Z])+ para obter palavras compostas apenas por letras. As três maiores palavras em português encontradas foram: magnificentissimamente (22 letras), inconstitucionalidade (21 letras) e principalissimamente (20 letras).

b) Qual a média do tamanho das palavras?

Resposta: O tamanho médio das palavras registro pelo job foi 4,74.

Questão 2. Dado um dataset com tweets relacionados à campanha eleitoral presidencial de 2014, responda:

a) Quais foram as hashtags mais usadas pela manhã, tarde e noite?

Procedimento: para esse item, criou-se um mapper que associava as hashtags (usadas como chave) com um mapa contendo três contagens: número de ocorrências nos turnos da manhã (00:00 até 11:59), tarde (12:00 até 17:59) e noite (18:00 até 23:59). O reducer apenas sumariza os valores por turno. A Figura 4 apresenta a codificação para esse item.

```
public static class MyMapper extends Mapper<Object, Text, Text, MapWritable> {
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        Tweet tweet = ReadTSV.parse(value.toString());
        ArrayList<String> tags = tweet.getHashTags();
        for (String tag : tags) {
            MapWritable map = new MapWritable();
            map.put(new Text("morning"), (tweet.isMorning()) ? new IntWritable(1) : new IntWritable(0));
            map.put(new Text("afternoon"), (tweet.isAfternoon()) ? new IntWritable(1) : new IntWritable(0));
            map.put(new Text("night"), (tweet.isNight()) ? new IntWritable(1) : new IntWritable(0));
            context.write(new Text(tag), map);
        }
    }
}

public static class MyReducer extends Reducer<Text, MapWritable, Text, NullWritable> {
    public void reduce(Text key, Iterable<MapWritable> values, Context context) throws IOException, InterruptedException {
        int morning = 0, afternoon = 0, night = 0;
        for (MapWritable val : values) {
            morning   += ((IntWritable) val.get(new Text("morning"))).get();
            afternoon += ((IntWritable) val.get(new Text("afternoon"))).get();
            night     += ((IntWritable) val.get(new Text("night"))).get();
        }
        context.write(new Text(key + "\t" + morning + "\t" + afternoon + "\t" + night), null);
    }
}
```

Figura 4. Mapper e Reducer para a contagem de hashtags por período do dia.

Resposta: a Tabela 1 apresenta as cinco hashtags mais utilizadas por turno. Os dados completos podem ser encontrados na planilha Respostas_Hadoop aba Q2a (link mencionado anteriormente).

Tabela 1. Top 5 Hashtags mais utilizadas por turno.

Turno	Hashtag
Manhã	#EMABiggestFansJustinBieber (94475), #EMABiggestFans1D (88214), #camilasayshi (9946), #EMABiggestFansJustinBieber" (2936), #TheVoiceBrasil (2649)
Tarde	#EMABiggestFans1D (57711), #EMABiggestFansJustinBieber (48951), #QueroNoTVZ (4430), #StealMyGirl (1991), #EMABiggestFansJustinBieber" (1466)
Noite	#EMABiggestFans1D (64075), #EMABiggestFansJustinBieber (54028), #StealMyGirl (4816), #bigpaynodanceoff (4085), #DebateNoSBT (3122)

b) Quais as hashtags mais usadas em cada dia?

Procedimento: para esse item, criou-se um mapper que associava as hashtags com o dia de criação da postagem. O reducer soma os valores por dia. A Figura 5 apresenta a codificação para esse item. Para esse item, gerou-se também um vídeo[3] com o ranking evolutivo das 10 hashtags mais utilizadas durante o período da campanha eleitoral de 2014.

[3] Vídeo 2.b: https://drive.google.com/file/d/1zfpAxd9y2YJ9gBjBZ53Nd2W7Fs3H3U3-/view?usp=sharing

```
public static class MyMapper extends Mapper<Object, Text, Text, Text> {
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        Tweet tweet = ReadTSV.parse(value.toString());
        ArrayList<String> tags = tweet.getHashTags();
        for (String tag : tags) {
            context.write(new Text(tag), new Text(tweet.getFormattedDate()));
        }
    }
}

public static class MyReducer extends Reducer<Text, Text, Text, NullWritable> {
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        HashMap<String, Integer> map = new HashMap<>();
        for (Text val : values) {
            String mkey = val.toString();
            if(map.containsKey(mkey)) {
                map.put(mkey, map.get(mkey) + 1);
            }else {
                map.put(mkey, 1);
            }
        }
        for(String mkey : map.keySet()) {
            context.write(new Text(key + "\t" + mkey + "\t" + map.get(mkey)), null);
        }
    }
}
```

Figura 5. Codificação do map-reduce para obter as hashtags mais usadas por dia.

Resposta: a Tabela 2 apresenta as dez hashtags mais utilizadas por dia. Os dados completos podem ser encontrados na planilha Respostas_Hadoop aba Q2b (link mencionado anteriormente).

Tabela 2. Top 10 Hashtags mais utilizadas por dia.

Turno	Hashtags
15/10	#EMABiggestFans1D #EMABiggestFansJustinBieber #StealMyGirl #bigpaynodanceoff #AssistamODR #BuyLoveMeHarderOniTunes, #Vote5HEMA #UnlockMockingjay #EMABiggestFansJustinBieber #QueroMuitosSeguidoresComValentino
16/10	#EMABiggestFans1D #EMABiggestFansJustinBieber #camilasayshi #DebateNoSBT #CartersNewVideo #AustinMahoneChile #EMABiggestFansJustinBieber" #AustinMahone #EMABiggestFans1D" #MasterChefBR
17/10	#EMABiggestFansJustinBieber #EMABiggestFans1D #QueroNoTVZ #TheVoiceBrasil #EMABiggestFansJustinBieber" #FlyNoMixDiario #LançamentoDoClipeVocêSeFoiFLY #AustinMahoneChile #AustinMahone #SextaTodosSDVcomValentino
18/10	#EMABiggestFans1D #EMABiggestFansJustinBieber #QueroNoTVZ #demiyourstorydoesntdefineyou #WeWantZaynsSongsInFOUR #trndnl #MaratonaTwilight #SigaTiuMarkitoNesseSabadoSdv #EMABiggestFansJustinBieber" #KCAArgentina
19/10	#EMABiggestFansJustinBieber #EMABiggestFans1D #LuanSantanaNaHoraDoFaro #LinkinParkNoMultishow #demiyourstorydoesntdefineyou #VoteVampsTeenAwards #ComeToBrazilMattEspinosa #trndnl #VamosLaU #DomingoPreguiçosoDoSDVcomValentino
20/10	#EMABiggestFansJustinBieber #EMABiggestFans1D #DebateNaRecord #QueroDilmaTreze #debatenarecord #CongratsOn1MChris #AecioEmTodoBrasil #EmTodoBrasilAecio45 #KCAArgentina #Los80

c) Qual o número de tweets por hora a cada dia?

Procedimento: similar aos itens anteriores, criou-se um mapper para associar a chave (dia e hora) ao número de tweets. O reducer apenas soma os valores por chave. A Figura 6 apresenta a codificação para esse item.

```
public static class MyMapper extends Mapper<Object, Text, Text, IntWritable> {
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        Tweet tweet = ReadTSV.parse(value.toString());
        context.write(new Text(tweet.getDayAndHour()), new IntWritable(1));
    }
}

public static class MyReducer extends Reducer<Text, IntWritable, Text, NullWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int total = 0;
        for (IntWritable val : values) {
            total += val.get();
        }
        context.write(new Text(key + "\t" + total), null);
    }
}
```

Figura 6. Codificação do map-reduce para obter o número de tweets por hora a cada dia.

Resposta: a Figura 7 apresenta a distribuição dos tweets por hora a cada dia. Os dados completos podem ser encontrados na planilha Respostas_Hadoop aba Q2c (link mencionado anteriormente).

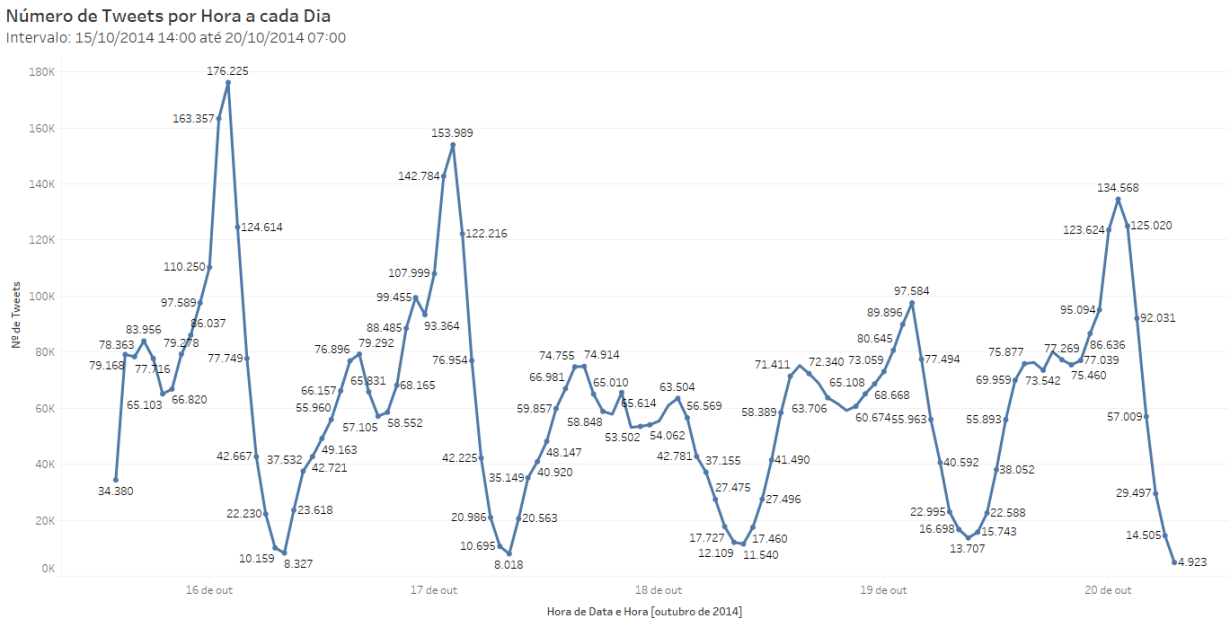

Figura 7. Distribuição de tweets por hora a cada dia.

d) Quais as principais sentenças relacionadas à palavra “Dilma”?

e) Quais as principais sentenças relacionadas à palavra “Aécio”?

Procedimento: os itens d) e e) possuem um procedimento bem similar. Com isso, serão apresentados em conjunto. Por sentenças, definimos a combinação de palavras próximas considerando os tamanhos 2, 3 e 4. Para definir a relação com a Dilma ou Aécio verificamos a existência desses termos no tweet. Para esses itens, nós realizamos um pré-processamento no texto. As atividades de pré-processamento foram: remoção de caracteres especiais, remoção de

URLs e remoção de palavras com tamanho menor que 3. Os dados completos estão na planilha Respostas_Hadoop abas Q2d e Q2e. Quanto ao job, o mapper usou como chave as sentenças e vinculou cada sentença ao número de ocorrência. O reducer além de somar as ocorrências também inseriu a informação do tamanho da sentença para facilitar a análise dos resultados. A Figura 8 apresenta a codificação para o caso das sentenças relacionadas a palavra Dilma.

```
public static class MyMapper extends Mapper<Object, Text, Text, IntWritable> {
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        Tweet tweet = ReadTSV.parse(value.toString());
        if(tweet.isRelatedDilma()) {
            ArrayList<String> nGrams = tweet.getNGrams();
            for (String ngram : nGrams) {
                context.write(new Text(ngram), new IntWritable(1));
            }
        }
    }
}

public static class MyReducer extends Reducer<Text, IntWritable, Text, NullWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int count = 0;
        for (IntWritable val : values) {
            count += val.get();
        }
        if(count > 5) {
            int length = new StringTokenizer(key.toString()).countTokens();
            context.write(new Text(key + "\t" + length + "\t" + count), null);
        }
    }
}
```

Figura 8. Codificação do map-reduce para contabilizar sentenças relacionadas ao termo Dilma.

Resposta: a Tabela 3 apresenta sentenças mais utilizadas considerando os termos Dilma e Aécio.

Tabela 3. Top 10 sentenças mais utilizadas em relação a Dilma e Aécio.

Termo	Tam.	Sentenças
Dilma	2	dilma aecio, aecio dilma, cara dilma, dilma fala, dilma esta, votar dilma, essa dilma, minas gerais, bolsa familia, dilma rousseff
	3	entre dilma aecio, aprovado pela dilma, aecio aprovado pela, cala boca dilma, dilma sabe falar, debate aecio neves, aecio neves aconversa, dilma nocauteada vivo, nocauteada vivo debate, vivo debate aecio
	4	aecio aprovado pela dilma, debate aecio neves aconversa, dilma nocauteada vivo debate, nocauteada vivo debate aecio, vivo debate aecio neves, aecio neves aconversa debatenosbt, entre dilma aecio prefiro, pelo marqueteiro joao santana, aecioneves esta postos acovardar, dilma orientada pelo marqueteiro
Aécio	2	aecio neves, dilma aecio, aecio dilma, votar aecio, esse aecio, minas gerais, aecio fala, vota aecio, aecio ganhar, aecio falando
	3	entre dilma aecio, aecio aprovado pela, aprovado pela dilma, debate aecio neves, aecio neves aconversa, dilma nocauteada vivo, nocauteada vivo debate, vivo debate aecio, quem conhece vota, neves aconversa debatenosbt
	4	aecio aprovado pela dilma, debate aecio neves aconversa, dilma nocauteada vivo debate, nocauteada vivo debate aecio, vivo debate aecio neves, aecio neves aconversa debatenosbt, entre dilma aecio prefiro, aecio quem conhece vota, aecioneves esta postos acovardar, dilmabr fuja raia aecioneves

Questão 3. Dado um dataset com tweets relacionados à visita da Torre Eiffel, responda:

a) Encontre as palavras mais utilizadas nas avaliações.

Procedimento: cada palavra foi utilizada como chave para o mapper e o reducer conta o número de ocorrências por chave. A Figura 9 apresenta a codificação para esse item.

```
public static class MyMapper extends Mapper<Object, Text, Text, IntWritable> {
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        Review review = ReadJson.getReview(value.toString());
        if(review.getText() != null) {
            StringTokenizer tokenizer = new StringTokenizer(review.getCleanedContent());
            while (tokenizer.hasMoreTokens()) {
                String token = tokenizer.nextToken();
                context.write(new Text(token), new IntWritable(1));
            }
        }
    }
}

public static class MyReducer extends Reducer<Text, IntWritable, Text, NullWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int count = 0;
        for (IntWritable val : values) {
            count += val.get();
        }
        context.write(new Text(key + "\t" + count), null);
    }
}
```

Figura 9. Codificação do map-reduce para contar as palavras mais utilizadas nas avaliações.

Resposta: a Tabela 4 apresenta os resultados.

Tabela 4. Lista com as 30 palavras mais utilizadas nas avaliações.

Palavra	Ocorrências	Palavra	Ocorrências	Palavra	Ocorrências
tower	5916	night	1868	amazing	1238
paris	3461	with	1860	line	1227
eiffel	3395	were	1777	long	1198
there	2524	visit	1711	great	1176
from	2521	views	1480	worth	1175
this	2223	very	1478	went	1160
time	2154	tickets	1420	your	1092
that	2042	when	1400	beautiful	1058
view	1907	must	1327	level	1051
have	1882	just	1244	floor	1016

b) Encontre as expressões mais usadas. Considere uma expressão um conjunto de palavras na sequência. O tamanho da sequência pode ser determinado por você.

Procedimento: por sentenças, definimos a combinação de palavras próximas considerando os tamanhos 2, 3 e 4. Para esse item, realizou-se um pré-processamento no texto com a seguintes atividades: remoção de caracteres especiais, remoção de URLs e remoção de palavras com tamanho menor que 3. A Figura 10 apresenta a codificação para esse item.

```java
public static class MyMapper extends Mapper<Object, Text, Text, IntWritable> {
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        Review review = ReadJson.getReview(value.toString());
        if(review.getText() != null) {
            ArrayList<String> nGrams = review.getNGrams();
            for (String ngram : nGrams) {
                context.write(new Text(ngram), new IntWritable(1));
            }
        }
    }
}

public static class MyReducer extends Reducer<Text, IntWritable, Text, NullWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int count = 0;
        for (IntWritable val : values) {
            count += val.get();
        }
        if(count > 5) {
            int length = new StringTokenizer(key.toString()).countTokens();
            context.write(new Text(key + "\t" + length + "\t" + count), null);
        }
    }
}
```

Figura 10. Codificação do map-reduce para encontrar as expressões mais usadas nas avaliações.

Resposta: a Tabela 4 apresenta os resultados.

Tabela 4. Top 10 sentenças mais utilizadas nas avaliações por tamanho.

Tamanho 2		Tamanho 3		Tamanho 4	
Sentença	**Qnt.**	**Sentença**	**Qnt.**	**Sentença**	**Qnt.**
eiffel tower	2938	visit eiffel tower	200	without visiting eiffel tower	23
view from	362	visited eiffel tower	135	must when visiting paris	20
second floor	329	eiffel tower must	112	paris visit eiffel tower	20
second level	269	paris eiffel tower	99	eiffel tower first time	19
well worth	261	visiting eiffel tower	90	must visit when paris	19
views from	238	eiffel tower night	76	must visit eiffel tower	17
skip line	235	from eiffel tower	59	paris without visiting eiffel	17
visit paris	231	went eiffel tower	58	eiffel tower must when	14
visit eiffel	216	seeing eiffel tower	56	have been eiffel tower	14
make sure	215	must when paris	55	paris eiffel tower must	14

c) Encontre os principais tópicos relacionados às revisões.

Procedimento: esse, com certeza, foi o item que gerou maior dificuldade para esse trabalho e também um maior consumo de recursos computacionais para obtenção de uma resposta adequada ao item. A definição de tópicos não é uma atividade trivial, uma vez que espera-se que tais tópicos possam caracterizar o conjunto de documentos analisados.

Na literatura, alguns algoritmos foram propostas para tal atividade. Dentre os algoritmo mais utilizados destaca-se o LDA (Latent Dirichlet Allocation)[4]. Esse algoritmo possui implementações em diversas linguagens, entretanto não encontrei nenhuma versão pronta e funcional para uso com Hadoop. Assim, após algumas buscas, decidi adaptar uma versão simplificada do LDA desenvolvida pelo usuário hankcs que foi disponibilizada no github[5].

Além da adaptação do LDA para o modelo de programação map-reduce do Hadoop, foi necessário um pré-processamento mais acurado para a obtenção de tópicos significativos. Para essa tarefa, decidimos utiliza a biblioteca CoreNLP[6] desenvolvida por pesquisadores da universidade de Stanford. Dentre as atividades do pipeline, destaca-se a tokenização, lematização e remoção de palavras compostas apenas por números, datas, e-mails, valores monetários, URLs, além de uma extensa lista de stop words[7] para a língua inglesa. As Figuras 11 e 12 apresentam, respectivamente, a codificação para a classe mapper e para a classe reducer.

```
public static class MyMapper extends Mapper<LongWritable, Text, LongWritable, Text> {

    public static int reviewCount = 0;
    private static List<String> stopWords = Arrays.asList(new String[]{"n't", "'ll", "'ve", "1-1", "a", "a's"

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        Review review = ReadJson.getReview(value.toString());
        if(review.getText() != null && !review.getText().isEmpty()) {
            ArrayList<String> list = new ArrayList<>();

            // NLP
            Properties props = new Properties();
            props.setProperty("annotators", "tokenize, ssplit, pos, lemma, ner");
            StanfordCoreNLP pipeline = new StanfordCoreNLP(props);

            // create an empty Annotation just with the given text
            Annotation document = new Annotation(review.getText());

            // run all annotators on this text
            pipeline.annotate(document);

            // these are all the sentences in this document a CoreMap is essentially
            // a Map that uses class objects as keys and has values with custom types
            List<CoreMap> sentences = document.get(SentencesAnnotation.class);

            for (CoreMap sentence : sentences) {
                // traversing the words in the current sentence a CoreLabel
                // is a CoreMap with additional token-specific methods
                for (CoreLabel token : sentence.get(TokensAnnotation.class)) {
                    // this is the text of the token
                    String word = token.get(TextAnnotation.class);
                    String ne = token.get(NamedEntityTagAnnotation.class);
                    if(!ne.equals("NUMBER") && !ne.equals("ORDINAL")
                            && !ne.equals("PERCENT")  && !ne.equals("DATE")
                            && !ne.equals("EMAIL")    && !ne.equals("MONEY")
                            && !ne.equals("TIME")     && !ne.equals("URL")
                            && !word.startsWith("http://")
                            && word.length() > 2
                            && !stopWords.contains(word)) {
                        list.add(word.toLowerCase().trim());
                    }
                }
            }
            context.write(key, new Text(String.join(",", list)));
            reviewCount++;
        }
    }
}
```

Figura 11. Classe Mapper para o problema da modelagem de tópicos com LDA.

---

[4] BLEI, David M.; NG, Andrew Y.; JORDAN, Michael I. Latent dirichlet allocation. Journal of machine Learning research, v. 3, n. Jan, p. 993-1022, 2003.
[5] LDA4j: https://github.com/hankcs/LDA4j
[6] Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60.
[7] PUURULA, Antti. Cumulative progress in language models for information retrieval. In: Proceedings of the Australasian Language Technology Association Workshop 2013 (ALTA 2013). 2013. p. 96-100. https://www.aclweb.org/anthology/U13-1013/

```java
public static class MyReducer extends Reducer<LongWritable, Text, Text, NullWritable> {

    public static int resultCount = 0;
    private static Corpus corpus = new Corpus();

    public void reduce(LongWritable key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        Text array = values.iterator().next();
        ArrayList<String> document = new ArrayList<>();
        String[] words = array.toString().split(",");
        for (String text : words) {
            document.add(text.toString().trim());
        }
        resultCount++;

        if(MyMapper.reviewCount != resultCount) {
            return;
        }

        // 1. Load corpus
        corpus.addDocument(document);
        // 2. Create a LDA sampler
        LdaGibbsSampler ldaGibbsSampler = new LdaGibbsSampler(corpus.getDocument(), corpus.getVocabularySize());
        // 3. Train it
        ldaGibbsSampler.gibbs(10);
        // 4. The phi matrix is a LDA model, you can use LdaUtil to explain it.
        double[][] phi = ldaGibbsSampler.getPhi();
        Map<String, Double>[] topicMap = LdaUtil.translate(phi, corpus.getVocabulary(), 10);

        int i = 0;
        for (Map<String, Double> topics : topicMap) {
            ArrayList<String> result = new ArrayList<>();
            result.add(String.format("Topic %d", i++));

            for (Map.Entry<String, Double> entry : topics.entrySet()) {
                result.add(entry.getKey() + " (" + String.format("%.4f", entry.getValue()) + ")");
            }

            context.write(new Text(String.join("\t", result)), null);
        }
        //LdaUtil.explain(topicMap);
    }
}
```

Figura 12. Classe Reducer para o problema da modelagem de tópicos com LDA.

Resposta: a Tabela 5 apresenta os resultados. Cada tópico possui 10 termos e os valores entre parênteses representa a relevância do termo para cada tópico. Não houve tempo hábil para a realização de testes que permitissem validar a corretude dos tópicos, mas em uma análise superficial os tópicos parecem representar o conteúdo das avaliações.

Tabela 5. Tópicos obtidos a partir dos tweets sobre visitas à Torre Eiffel.

Tópicos	Termos
Topic 0	the (0,0488) visited (0,0480) view (0,0479) experience (0,0479) eiffel (0,0479) spectacular (0,0476) make (0,0474) tower (0,0471) great (0,0465) apartment (0,0463)
Topic 1	eiffel (0,0520) great (0,0502) buy (0,0483) apartment (0,0473) thomas (0,0469) tiny (0,0464) level (0,0459) visited (0,0458) experience (0,0456) make (0,0453)
Topic 2	eiffel (0,0497) level (0,0486) ticket (0,0477) great (0,0477) spectacular (0,0476) skim (0,0476) access (0,0470) there (0,0462) apartment (0,0454) buy (0,0453)
Topic 3	eiffel (0,0538) view (0,0500) edison (0,0487) elevator (0,0487) tower (0,0478) people (0,0476) level (0,0471) ticket (0,0465) patient (0,0460) buy (0,0459)
Topic 4	eiffel (0,0598) elevator (0,0498) thomas (0,0481) great (0,0464) visited (0,0464) level (0,0463) patient (0,0463) tiny (0,0456) ticket (0,0456) the (0,0454)
Topic 5	tiny (0,0520) apartment (0,0495) eiffel (0,0491) make (0,0490) thomas (0,0470) visited (0,0468) buy (0,0467) tower (0,0464) skim (0,0459) access (0,0459)

Topic 6	eiffel (0,0583) visited (0,0500) experience (0,0496) make (0,0475) edison (0,0469) patient (0,0459) there (0,0456) skim (0,0455) tower (0,0454) spectacular (0,0454)
Topic 7	there (0,0516) edison (0,0487) the (0,0481) tower (0,0479) people (0,0473) access (0,0467) view (0,0467) ticket (0,0463) experience (0,0463) tiny (0,0457)
Topic 8	eiffel (0,0552) people (0,0501) level (0,0485) buy (0,0478) spectacular (0,0474) tiny (0,0467) ticket (0,0465) there (0,0464) the (0,0455) edison (0,0450)
Topic 9	edison (0,0506) eiffel (0,0491) skim (0,0481) tower (0,0476) patient (0,0475) access (0,0474) thomas (0,0473) spectacular (0,0465) ticket (0,0464) the (0,0458)

d) Mapeie a distribuição temporal das revisões.

Procedimento: para finalizar, a distribuição temporal das revisões foi realizada usando como chave a data das revisões e como valor o número de ocorrências naqueles dias. A Figura 13 apresenta a codificação para esse item.

```
public static class MyMapper extends Mapper<Object, Text, Text, IntWritable> {
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        Review review = ReadJson.getReview(value.toString());
        context.write(new Text(review.getFormattedDate()), new IntWritable(1));
    }
}

public static class MyReducer extends Reducer<Text, IntWritable, Text, NullWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int count = 0;
        for (IntWritable val : values) {
            count += val.get();
        }
        context.write(new Text(key + "\t" + count), null);
    }
}
```

Figura 13. Codificação do map-reduce para análise temporal das revisões.

Resposta: a Figura 14 apresenta os resultados.

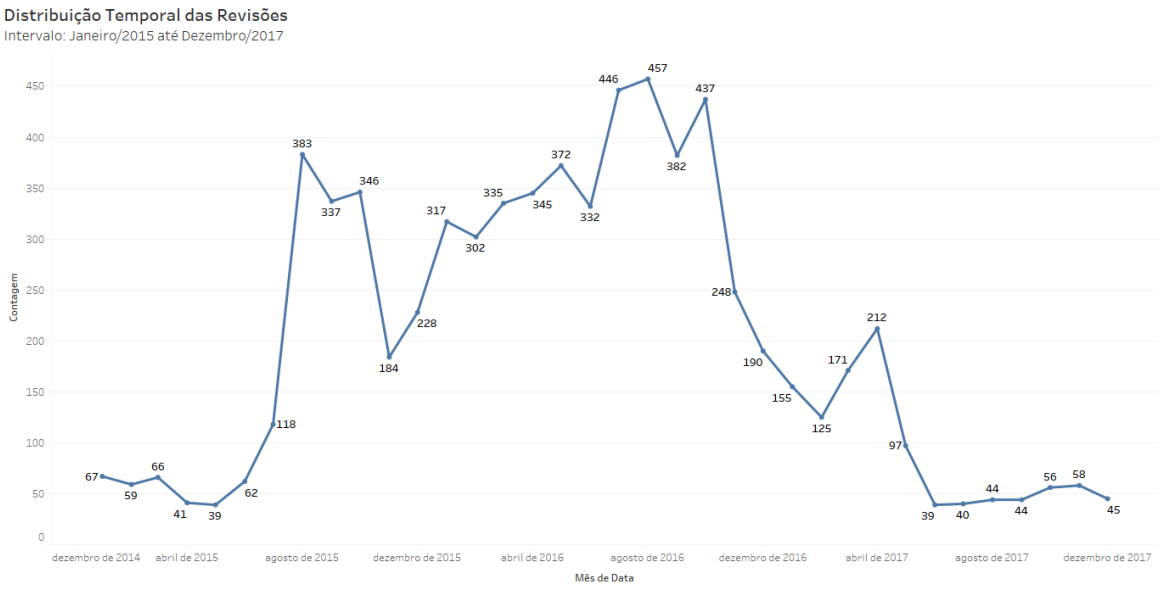

Figura 14. Distribuição temporal das revisões.