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

Thais Carvalho Areias

Análise de Sentimento em Avaliações de Telefones Celulares

Belo Horizonte
2021

Thais Carvalho Areias

Análise de Sentimento em Avaliações de Telefones Celulares

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.

Belo Horizonte
2021

SUMÁRIO

1. Introdução

1.1. Contextualização

Novas tecnologias vêm sendo criadas para conectar e diminuir a distância entre as pessoas. Com a ascensão das tecnologias houve um grande crescimento na geração de dados, disponibilizados de diferentes formas na internet. A análise desses dados pode trazer grande valor e auxiliar em tomadas de decisão empresariais. Esse tipo de análise pode trazer informações para a empresa como: reputação da marca na internet; avaliar a receptividade de novos produtos no mercado; e conhecer os pontos fortes e fracos da empresa.

A análise de dados de forma automatizada, sem interação humana, é muito importante nesse novo cenário de crescimento constante e aumento na complexidade dos dados.Redes sociais, como o twitter, possuem um alto número de usuários, com abrangência mundial de comentários. Nos últimos anos as compras online se popularizaram, e muitos sites de compras incentivam seus clientes a fazerem avaliações dos produtos adquiridos. Essa avaliação é disponibilizada no site e influencia a opinião de possíveis compradores. O Processamento de língua natural (PLN) é uma subárea da ciência da computação, inteligência artificial e da linguística que estuda os problemas da geração e compreensão automática de línguas humanas naturais.

1.2. O problema proposto

Neste trabalho, apresentamos uma análise de sentimento voltada para o mercado de telefones celulares. Para auxiliar no entendimento do problema vamos utilizar a ferramenta 5W, que consiste em responder às seguintes perguntas:

Por que *(Why)*: Auxiliar empresas do setor de telefonia celular a conhecer a reputação da empresa, avaliar a receptividade de novos produtos, conhecer seus pontos positivos e negativos e com isso tomar decisões empresariais efetivas.

Quem (*Who*): Com o intuito de fazer uma análise de sentimento voltada ao mercado de telefones celulares utilizamos avaliações de clientes da Amazon em telefones celulares, disponibilizada na plataforma kaggle.

O que (*What*): O objetivo desse projeto é analisar uma base de dados com informações de avaliações de telefones celulares do site da Amazon.com, com o interesse em construir um processo de classificação de polaridade. Para tanto, os dados serão pré processados por meio de aplicação dos modelos *TF-IDF* (Frequência do Termo–Inverso da Frequência nos Documentos) e aprendizado de máquina supervisionado, os quais serão detalhados mais à frente

Onde (*Where*): A análise não considera a localidade, foram coletadas informações de sobre os telefones celulares comercializados pela Amazon.com.

Quando (*When*): 1 de Abril de 2011 a 9 de Setembro de 2019.

2. Coleta de Dados

A base de dados utilizada neste trabalho foi obtida através da Plataforma Kaggle. Os dados representam o histórico de 9 anos em avaliações de telefones celulares vendidos no site da Amazon.

Tabela 1: 20191226-items.csv

Coluna	Descrição	Tipo
asin	Número único de identificação de produtos vendidos pela Amazon.	Alfanumérico, normalizado
brand	Fabricante de telefone celular.	Texto
title	Título do produto.	Texto
url	Url do produto.	Url
image	Url com foto do produto.	Url
rating	Média das avaliações do produto de 1 a 5.	Numérico
reviewUrl	Url da página de avaliação do produto.	Url
totalReviews	Número total de avaliações.	Numérico
price	Preço do produto	Numérico

originalPrice	Preço original do produto.	Numérico

Tabela 2: 20191226-reviews.csv

Coluna	Descrição	Tipo
asin	Número único de identificação de produtos vendidos pela Amazon.	Alfanumérico, normalizado
name	Nome do avaliador.	Texto
rating	Nota da avaliação do review numa escala de 1 a 5.	Numérico
date	Data em que a avaliação foi escrita.	data
verified	Verificação se o usuário é válido.	Booleano
title	Título da avaliação.	Texto
body	Comentário da avaliação.	Texto
helpfulVotes	Número total de avaliações avaliadas como úteis.	Numérico

Referência: “https://www.kaggle.com/grikomsn/amazon-cell-phones-reviews”

3. Processamento/Tratamento de Dados

O dataset possui 59.815 diferentes avaliações, às quais procedemos a análise, limpeza e tratamento dos dados. O projeto foi executado por meio da

linguagem de programação interpretada Python no Jupyter Notebook– ambiente de desenvolvimento iterativo baseado na Web.

3.1 Importação dos dados e filtragem dos dados

Os *datasets* foram importados no formato csv e convertidos em um dataframe do Dask, comumente utilizado para grandes *datasets,* pela sua capacidade de processamento em paralelo, como pode ser visto na Figura 1.

```
import dask.dataframe as dd

df_review = dd.read_csv(r'C:\Users\thais\Google Drive\
                          Ciencia de dados e Big data\13.TCC\archive\20191226-reviews.csv')
df_item = dd.read_csv(r'C:\Users\thais\Google Drive\
                          Ciencia de dados e Big data\13.TCC\archive\20191226-items.csv')
```

Figura 1: Leitura dos *datasets 20191226-items.csv* e *20191226-reviews.csv*

O dataframe “df_review” e “df_item” foram utilizados como *raw data*, ou seja, dado original sem alterações. Com o intuito de facilitar a manipulação dos dados, foi criado um novo *data frame,* resultante da junção de “df_review” com a informação de fabricante (*brand*) contida em “df_item”, utilizando a identificação única do produto (*asim)* como chave primária. E em seguida foi realizada a remoção das colunas com informações não relevantes para essa análise. São elas: *name, date, verified, title and helpfulVotes.* Como pode ser observado na Figura 2.

```
df=df_review.merge(df_item[['asin', 'brand']])
df=df.drop(['name','date','verified','title','helpfulVotes'], axis=1)
df
```

Dask DataFrame Structure:

	asin	rating	body	brand
npartitions=1				
	object	int64	object	object
	...	...	...	...

Dask Name: drop_by_shallow_copy, 5 tasks

Figura 2: Novo dask dataframe df com as colunas asin, rating, body and brand.

3.1 Tratamento de dados ausentes

Uma vez que o dataframe de junção foi obtido, “df” possui todas as informações relevantes para análise dos dados. A Figura 3 mostra o número de

valores ausentes. Como a quantidade de valores ausentes é pequena com relação ao tamanho do *dataset* (menor que 1 por cento), foram removidas todas as linhas que possuem valores nulos.

```
missing_value=df[['rating', 'body', 'brand']].isnull().sum().compute()
print(missing_value)
```

```
rating      0
body       21
brand     200
dtype: int64
```

```
df=df.dropna(subset=['rating', 'body', 'brand'])
```

Figura 3: Remoção das linhas com valores nulos.

3.2 Pré-Processamento do texto

Antes de aplicar algoritmos de classificação de sentimento é necessário realizar um pré-processamento, a fim de gerar uma versão simplificada, que melhor represente o texto, para facilitar o processamento computacional. O pré-processamento aplicado irá impactar na performance do modelo de aprendizado de máquina que será aplicado posteriormente.

O processamento dos dados irá seguir a sequência apresentada na Figura 4.

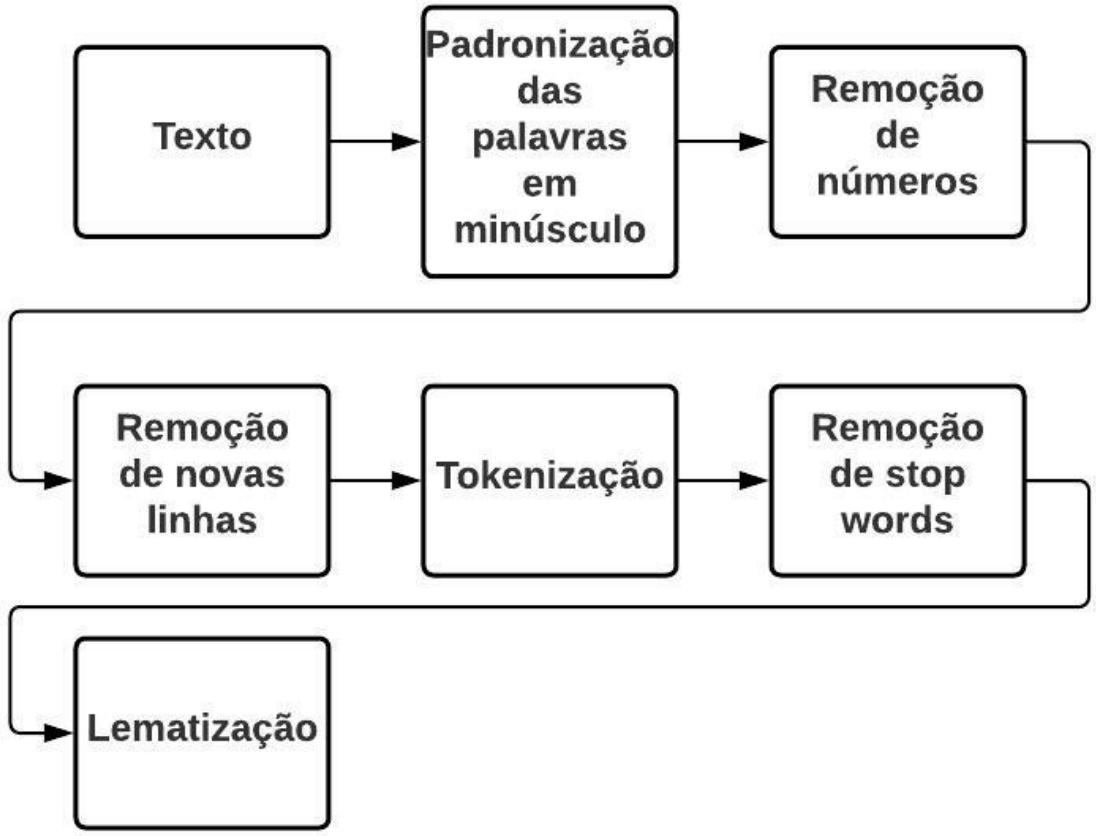

Figura 4: Etapas de Pré-Processamento dos dados.

A primeira etapa consiste em converter todos os caracteres do documento para um único tamanho, de forma a conferir maior agilidade no processo de indexação. Para isso, foi utilizada a função Map do Python, que recebe uma lista e transforma numa nova, executando em toda a lista o método *.lower(),* que retorna o texto com caracteres em minúsculo. Segue abaixo a função e um exemplo:

```
df['body'].map(lambda text:str(text).lower())
```

Texto original	'I have been with nextel for nearly a year now I started out this time last year with the Motorola i205 and just upgraded to the i265 it is one of the best phones I have ever had the service is the best I have ever had I have no problems making or reciving calls. If you are considering nextel give it a shot they are in my opinion the best cell company out there.'
Palavras em minúsculo	'i have been with nextel for nearly a year now i started out this time last year with the motorola i205 and just upgraded to the i265 it is one of the best phones i have ever had the service is the best

	i have ever had i have no problems making or reciving calls. if you are considering nextel give it a shot they are in my opinion the best cell company out there.'

A seguir é realizada a remoção de números, pois não agregam ao entendimento do texto e podem adicionar ruído nos algoritmos. Foi utilizada a função Map, aplicando o método *re.sub()*, que irá buscar um padrão de string e substituí-lo por uma nova sub-string. Caso o padrão não seja encontrado, a string é retornada sem mudanças. Para a remoção dos números, o método re.sub irá procurar pelo padrão *r'\d+'*, que identifica dígitos de 0 a 9 e substituir por um espaço vazio. O método que faz a remoção dos números e um exemplo estão apresentados abaixo:

```
import re
def remove_numbers(text):
      return re.sub(r'\d+', '', text)
```

Remoção de números	'i have been with nextel for nearly a year now i started out this time last year with the motorola i and just upgraded to the i it is one of the best phones i have ever had the service is the best i have ever had i have no problems making or reciving calls. if you are considering nextel give it a shot they are in my opinion the best cell company out there.'

Em um texto, novas linhas são representadas por caracteres, a etapa a seguir irá remover os caracteres referentes a novas linhas utilizando a função Map aplicando o método re.sub para encontrar a substring *r'\s+'* e substituir por um espaço. O método que faz a remoção dos novas linhas e um exemplo estão apresentados abaixo:

```
import re
def remove_newlinechars(text):
    return re.sub(r'\s+', ' ', text)
```

Remoção do novas linhas	'i have been with nextel for nearly a year now i started out this time last year with the motorola i and just upgraded to the i it is one of the best phones i have ever had the service is the best i have ever had i have no problems making or reciving calls. if you

	are considering nextel give it a shot they are in my opinion the best cell company out there.'

O método de tokenização separa os termos em unidades chamadas *tokens*, importante para estruturar o texto para os processamentos a seguir. A maneira como os tokens são separados pode variar, dependendo da língua sendo processada. O dataset que é tratado neste trabalho utiliza o espaço como separador. A biblioteca NLTK do python possui o método *word_tokenize()* que associado a função Map irá gerar os *tokens*. O método que faz a tokenização e o exemplo estão apresentados abaixo:

```
import nltk
def tokenize(text):
    tokens = nltk.word_tokenize(text)

    return list(
        filter(lambda word: word.isalnum(), tokens)
    )
```

Tokenização	['i','have','been','with','nextel','for','nearly','a','year','now','i','started','out','this','time','last','year','with','the','motorola','i','and','just','upgraded','to','the','i','it','is','one','of','the','best','phones','i','have','ever','had','the','service','is','the','best','i','have','ever','had','i','have','no','problems','making','or','reciving','calls','if','you','are','considering','nextel','give','it','a','shot','they','are','in','my','opinion','the','best','cell','company','out','there']

A etapa de remoção de *stop words*, que são palavras que sozinhas não agregam informação ao texto e que podem ser removidas, tais como preposições, pronomes, artigos e advérbios. Dentre as *stop words* da língua inglesa estão palavras como: the, a, about, etc. A biblioteca NLTK do python possui uma lista *stop words* pré definida. Com isso foi utilizada a função Map associada ao método *remove_stopwords()* para remover todas as palavras classificadas como *stop words*. O método que faz a remoção das *stop words* e o exemplo estão apresentados a seguir:

```
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = stopwords.words("english")
```

```
def remove_stopwords(words):
    filtered = filter(lambda word: word not in stop_words, words)
    return list(filtered)
```

Remoção de stopwords	['nextel','nearly','year','started','time','last','year','motorola','upgraded','one','best','phones','ever','service','best','ever','problems','making','reciving','calls','considering','nextel','give','shot','opinion','best','cell','company']

O processo de Lematização tem como objetivo reduzir uma palavra à sua forma base do dicionário e agrupar diferentes formas da mesma palavra, para que possam ser analisadas como um único item, o lema. A biblioteca Spacy do python foi utilizada no processo de lematização associada com a função Map. O método que faz a lematização e o exemplo estão apresentados abaixo:

```
import spacy
nlp = spacy.load("en_core_web_sm")

def lemmatize(text, nlp=nlp):
    doc = nlp(" ".join(text))
    lemmatized = [token.lemma_ for token in doc]

    return lemmatized
```

Lematização	['nextel','nearly','year','start','time','last','year','motorola','upgrade','one','good','phone','ever','service','well','ever','problem','make','reciving','call','consider','nextel','give','shot','opinion','good','cell','company']

Uma nova coluna chamada *df['cleaned_text']* foi criada no Dask data frame para armazenar o texto pré-processamento seguindo as etapas descritas anteriormente, como pode ser vista nas Figura 5 e 6.

```
from dask.diagnostics import ProgressBar
with ProgressBar():
    df['cleaned_text']=df['body'].map(
        lambda text:str(text).lower()).map(
        remove_numbers).map(
        remove_newlinechars).map(
        tokenize).map(
        remove_stopwords).map(
        lemmatize).map(
        ' '.join)
```

Figura 5: Nova coluna no dask data frame com o texto pré-processado.

```
import re
import spacy
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = stopwords.words("english")
nlp = spacy.load("en_core_web_sm")

def lemmatize(text, nlp=nlp):
    doc = nlp(" ".join(text))
    lemmatized = [token.lemma_ for token in doc]

    return lemmatized

def tokenize(text):
    tokens = nltk.word_tokenize(text)
    return list(
        filter(lambda word: word.isalnum(), tokens)
    )
def remove_stopwords(words):
    filtered = filter(lambda word: word not in stop_words, words)
    return list(filtered)

def remove_newlinechars(text):
    return re.sub(r'\s+', ' ', text)

def remove_numbers(text):
    return re.sub(r'\d+', '', text)
```

Figura 6: Código Python das funções aplicadas no pré-processamento.

4. Análise e Exploração dos Dados

Temos definidos alguns tópicos a serem respondidos que nos ajudarão a transformar os dados em informações por meio de análises quantitativas. São eles:

quantidade de avaliações por marca, quantidade de produtos por marca, média das avaliações por marca, quantidade de notas por marca e agrupadas por classe.

4.1 Quantidade de avaliações por marca

Para obter a quantidade de avaliações por marca é necessário agrupar os dados por marca, usando o *groupby(['brand'])* e contando cada uma das entradas com o *['asin'].count, como pode ser observado na Figura 7. No gráfico apresentado na Figura 8 podemos observar que a Samsung é a empresa que mais possui avaliações.*

```
from matplotlib import pyplot as plt

plt.figure(figsize=(15,4))
plt.title("Number of Reviews por Marca", size=15, weight="bold")
plt.xlabel("Brand")
plt.ylabel("Reviews")
df.groupby(['brand'])['asin'].count().compute().plot.bar()
```

Figura 7: Código Python para gerar o gráfico para quantidade de avaliações por marca.

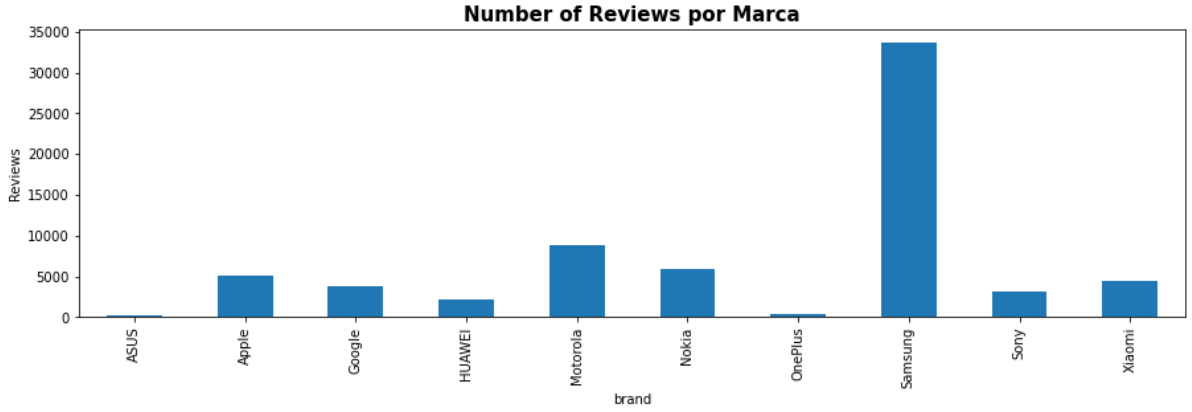

Figura 8: Gráfico de barras com a quantidade de avaliações por marca.

4.2 Quantidade de produtos por marca

Para obter a quantidade de modelos por Marca é necessário agrupar os dados por marca, usando o *groupby(['brand'])* e contando os valores únicos do identificador do produto com o *['asin'].nunique, como pode ser observado na Figura 9. No gráfico apresentado na Figura 10 podemos observar que a Samsung é a empresa que possui a maior quantidade de produtos diferentes, o que justifica o maior número de avaliações.*

```
plt.figure(figsize=(15,4))
plt.title("Number of Models by Brand", size=15, weight="bold")
plt.xlabel("Brand")
plt.ylabel("Reviews")
df.groupby(['brand'])['asin'].nunique().compute().plot.bar()
```

Figura 9: Código Python para gerar o gráfico para quantidade de produtos por marca.

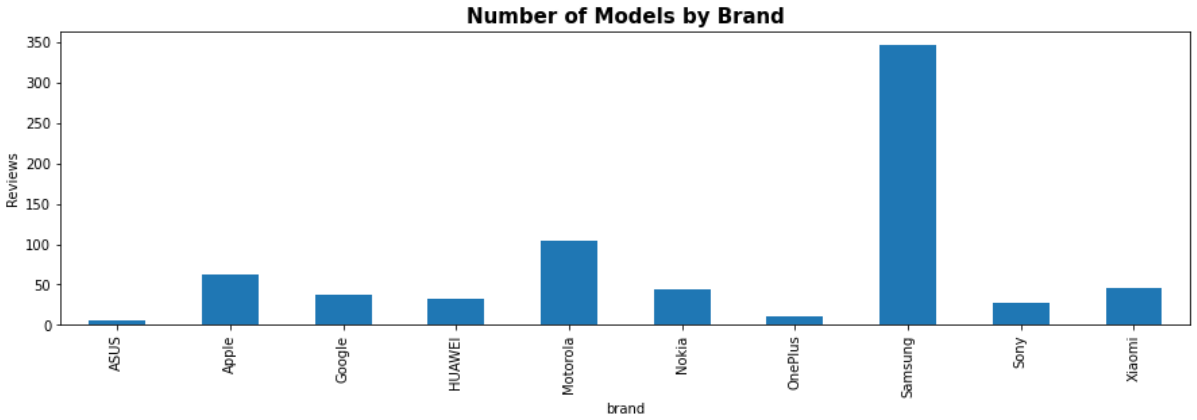

Figura 10: Gráfico de barras com a quantidade de produtos por marca.

4.3 Média das avaliações por marca

Para obter a média das avaliações por marca é necessário agrupar os dados por marca e calcular a média das avaliações com *['rating'].mean()*, como pode ser observado na Figura 11. No resultado, apresentado na Figura 12, é possível observar que na média a Xiaomi é a empresa que possui as melhores avaliações, seguida da Huawei. No entanto, a média de todas as empresas possuem boas avaliações.

```
plt.figure(figsize=(15,5))
plt.title("Mean Rate by Brand", size=15, weight="bold")
plt.xlabel("Brand", size=13)
plt.ylabel("Mean Rate", size=13)
df.groupby(['brand'])['rating'].mean().compute().plot.bar()
```

Figura 11: Código Python para gerar o gráfico para quantidade de produtos por marca.

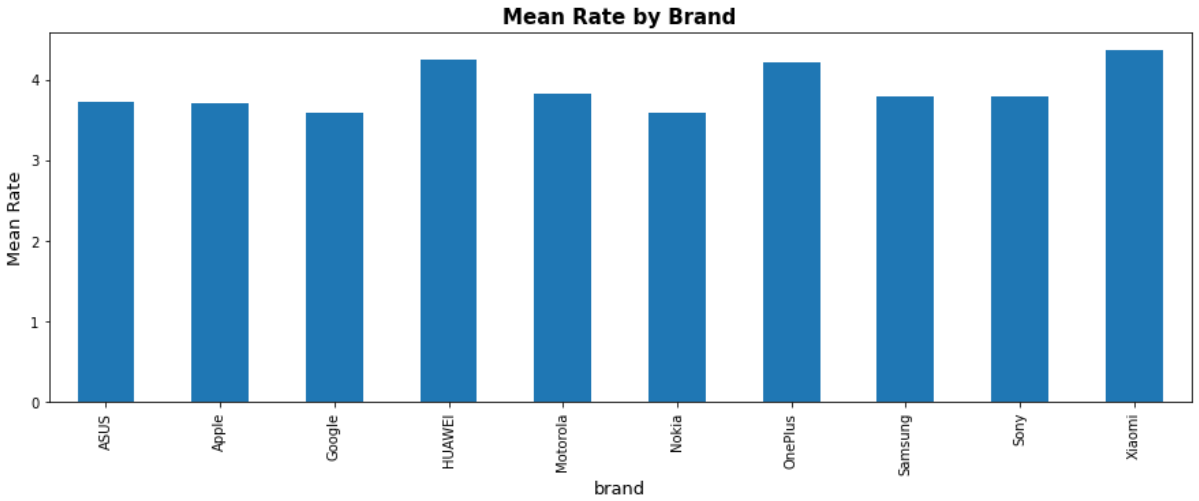

Figura 12: Gráfico de barras com a quantidade de produtos por marca.

4.4 Quantidade de notas por Marca

Para obter a quantidade de notas para cada marca, é necessário agrupar os dados por marca e nota, com o *groupby(['brand','rating'])* e verificar o tamanho com o *.size()* e dividir pela quantidade total de review através do *.apply(lambda x: 100 * x / df['asin'].count()).* O código e os resultados estão apresentados na figura 13 e 14. As figuras 15 e 16 apresentam a mesma informação de forma normalizada e é possível concluir que a base de dados é consistente e apresenta avaliação de diversos tipos de clientes, tanto de marcas quanto de faixa de preço.

```
import matplotlib.ticker as mtick
color_map = plt.cm.get_cmap('coolwarm')
df.groupby(['brand','rating']).size().compute().groupby(level=0).apply(
    lambda x: 100 * x / x.sum()).unstack().plot.bar(
    stacked=True, cmap =color_map.reversed())
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())
plt.title("Review Rate by Brand", size=15, weight="bold")
plt.xlabel("Brand" , size=13)
plt.ylabel("Rate", size=13)
```

Figura 13: Código python para gerar a quantidade de notas por marca.

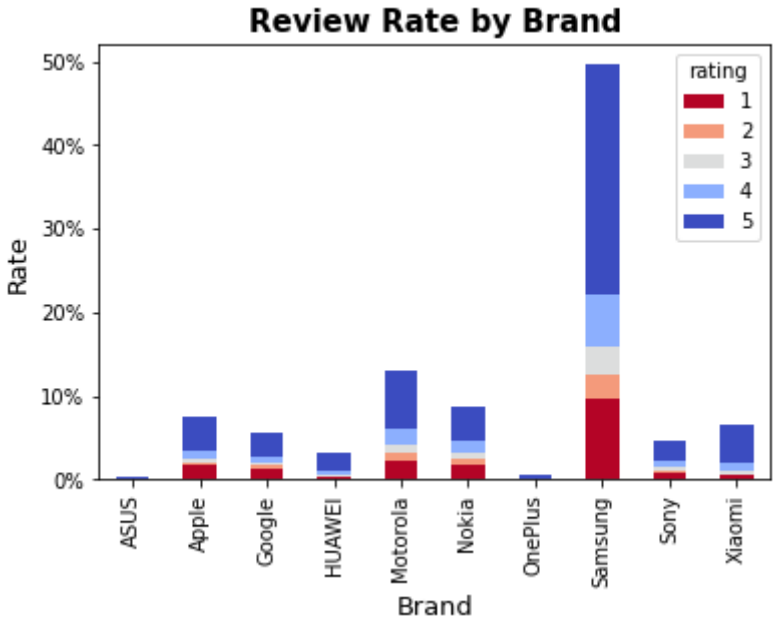

Figura 14: Gráfico de barras da quantidade de notas por marca.

```
import matplotlib.ticker as mtick
color_map = plt.cm.get_cmap('coolwarm')
df.groupby(['brand','rating']).size().compute().groupby(level=0).apply(
    lambda x: 100 * x / x.sum()).unstack().plot.bar(
    stacked=True, cmap =color_map.reversed())
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())
plt.title("Review Rate by Brand", size=15, weight="bold")
plt.xlabel("Brand" , size=13)
plt.ylabel("Rate", size=13)
```

Figura 15: Código python para gerar a quantidade de notas por marca normalizado.

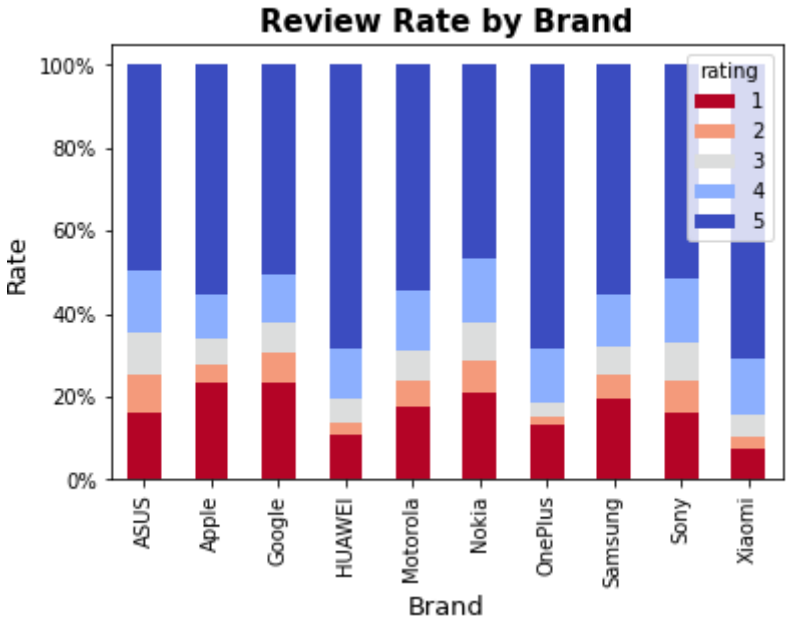

Figura 16: Gráfico de barras da quantidade de notas por marca normalizado.

4.5 Quantidade de avaliações agrupadas por classe

Para calcular a quantidade todas de reviews positivos e negativos foi criada uma nova coluna no dataframe classificando todos os review com nota maior ou igual a 3 como positivo e o restante como negativo aplicando a seguinte função *apply(lambda x: 'positive' if x >= 3 else 'negative', meta=float),* e verificando o tamanho do dataframe agrupado por opinião com a função *df.groupby(['opinion']).size()*. O código e os resultados estão apresentados nas Figuras 17, 18 e 19.

```
df['opinion']=df['rating'].apply(
    lambda x: 'positive' if x >= 3 else 'negative', meta=float)
df.groupby(['opinion']).size().compute()
```

```
opinion
negative    16602
positive    51163
dtype: int64
```

Figura 17: Código python para criar nova coluna no dataframe para classificar a avaliação em positivo e negativo.

```
explode = (0, 0.1)
labels='negative','positive'
colors = ['tab:red','tab:blue']
textprops=dict(color="w",size=12, weight="bold")

df.groupby(['opinion']).size().compute().plot.pie(
    explode=explode, label=labels, colors=colors, textprops=textprops,figsize=(
        5, 5), autopct='%1.1f%%', startangle=0, shadow = True)
plt.legend(labels, title="Opinion", loc="center left", bbox_to_anchor=(1, 0, 0.5, 1))
plt.title("Reviews", size=15, weight="bold")
```

Figura 18: Código python para criar o gráfico de avaliações positivas e negativas.

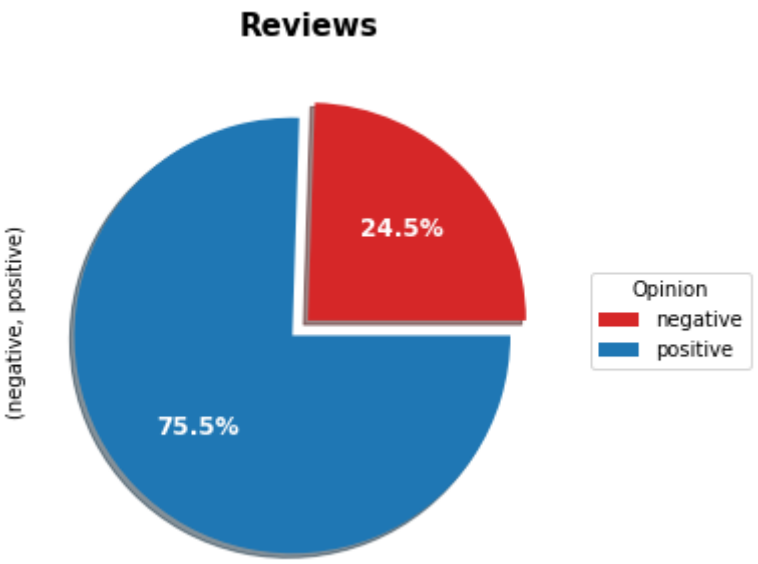

Figura 19:Gráfico de pizza de avaliações positivas e negativas.

5. Criação de Modelos de Machine Learning

Como já visto, o foco deste trabalho é na análise de sentimento, que busca identificar os sentimentos expressos no documento e determinar sua polaridade. Para classificar o texto será utilizado uma abordagem de aprendizado de máquina supervisionado, que necessita de dados previamente classificados.

Os textos e duas polaridades disponíveis na base de dados serão utilizados para treinar o algoritmo de classificação. No entanto, as mesmas palavras podem aparecer em textos pré-classificados como positivos e negativos. Nesse caso,

podemos levar em consideração a frequência em que as palavras aparecem em cada documento.

Neste trabalho iremos utilizar o modelo TF-IDF (Frequência do Termo–Inverso da Frequência nos Documentos), que tem como objetivo indicar a importância de uma palavra no texto em relação a uma coleção de documentos. O TF-IDF possui duas componentes, são elas: Frequência do termo (TF) que representa a frequência com que uma determinada palavra aparece em um documento; e Freqüência inversa do documento (IDF) que reduz as palavras que aparecem em muitos documentos. Isso auxilia a distinguir o fato da ocorrência de algumas palavras serem geralmente mais comuns que outras.

O cálculo do TF-IDF será realizado com auxílio da biblioteca *Scikit-Learn* do Python utilizando a classe"*TfidfVectorizer",* como parâmetro selecionamos min_df=10 que irá ignorar termos que aparecem em menos que dez documentos, removendo palavras muito raras entre os documentos. Em seguida utilizamos o método *"fit_transform()"* que irá gerar os valores de TF e IDF. O método *".get_feature_names()"* retorna as as palavras selecionadas e o *".vocabulary_" retorna as palavras com* sua respectiva frequência.

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
X = vectorizer.fit_transform(df['cleaned_text'].compute())
features_tfidf=vectorizer.get_feature_names()
vocabulary=vectorizer.vocabulary_
print(vocabulary)
```

Figura 20: Código Python que calcula os valores TF-IDF.

```
{'buy': 659, 'service': 4395, 'nearly': 3220, 'year': 5626, 'star
rade': 5309, 'one': 3378, 'good': 2118, 'phone': 3613, 'ever': 16
2, 'consider': 973, 'give': 2085, 'shot': 4458, 'opinion': 3401,
y': 1501, 'use': 5334, 'hear': 2253, 'person': 3594, 'talk': 4907
ay': 3543, 'additional': 70, 'fee': 1818, 'unlock': 5265, 'produc
er': 5565, 'definitely': 1222, 'recommend': 4005, 'anyone': 216,
96, 'get': 2069, 'go': 2107, 'new': 3249, 'day': 1165, 'contract'
```

Figura 21: Parte do vocabulário gerado com suas respectivas frequências.

Com o intuito de propiciar uma visualização direta dos termos mais frequentes foi gerada uma nuvem de palavra utilizando a biblioteca WordCloud do Python utilizando o método *".generate_from_frequencies()"* que criará a nuvem de palavras com base na frequência do termo gerado pelo *".vocabulary_".*

```
from matplotlib import pyplot as plt
from wordcloud import WordCloud

wordcloud = WordCloud (
    max_font_size=100, width=1520, height=535, colormap='viridis',
    background_color='white').generate_from_frequencies(
    vocabulary)
plt.figure(figsize=(16,9))
plt.axis("off")
plt=plt.imshow(wordcloud)
```

Figura 21: Código para gerar a nuvem de palavras com os termos mais frequentes.

Figura 22: Nuvem de palavras gerada a partir do vocabulário.

Para treinar e avaliar um modelo de aprendizado de máquina é necessário dividir o dataset em duas partes, uma para treinamento e outra para teste. Na base de treinamento, o modelo irá identificar os padrões nos dados e esses padrões serão utilizados para fazer previsão em novos dados. A biblioteca *scikit-learn* possui o método *"train_test_split()"* que divide a base de dados em duas partes aleatoriamente. No método "*train_test_split",* dois parametros importantes foram definidos: *test_size=0.33,* indica a porcentagem do tamanho dos dados para teste e; *random_state=42,* para que a divisão seja a mesma toda vez que o algoritmo seja executado.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
x = df['cleaned_text']
y = df['opinion']
```

```
with ProgressBar():
    X_train, X_test, y_train, y_test = train_test_split(
        x.compute(),y.compute(),test_size=0.33,random_state=42)
```

```
[########################################] | 100% Completed | 21min 55.7s
[########################################] | 100% Completed | 37min 50.8s
```

Figura 23:Código para divisão da base de dados em treinamento e teste.

Esta seção apresenta alguns dos algoritmos mais utilizados na construção de classificadores de texto: LinearSVC, Naive Bayes, Random Forest e Logistic Regression. Com o objetivo de experimentar diferentes parâmetro no projeto vamos utilizar o *Pipeline* da biblioteca *sklearn*, que vai nos permitir automatizar os fluxos de processos

5.1 Métricas de Avaliação do modelo de Machine Learning

Para análise do resultado do modelo iremos utilizar método *classification_report()* e *confusion_matrix()* do módulo *sklearn.metrics*. As métricas utilizadas são:

- False Positive (FP): É o número de vezes que uma classe foi avaliada como positiva de forma incorreta - era negativa.
- False Negative (FN): É o número de vezes que uma classe foi avaliada como negativa de forma incorreta - era positiva.
- True Positive (TP): É o número de vezes que uma classe foi avaliada como positiva de forma correta - era positiva.
- True Negative (TN): É o número de vezes que uma classe foi avaliada como negativa de forma correta - era negativa.
- Precisão: Mede o desempenho do classificador para classificar uma classe específica.

  $Precisão = TP/(TP + FP)$ ou $Precisão = TN/(TN + FN)$
- Cobertura (recall): É a capacidade de um classificador encontrar todas as instâncias corretas do modelo.

$$Cobertura = TP/(TP + FN)$$

- Acurácia: Mede o desempenho do classificador para a combinação das duas classes, ou seja, mede a exatidão do classificador.

$$Acuracia = TP + TN / (TP + TN + FP + FN)$$

- F1-Score: É uma métrica derivada da precisão e cobertura, sendo a média harmônica entre elas. Assim, podemos dizer que o classificar chega ao seu melhor resultado quando o F1-Score atinge o seu ponto máximo.

$$F1 - Score = 2 * (Cobertura * Precisão) / (Cobertura + Precisão)$$

5.2 LinearSVC:

O modelo LinearSVC implementa um algoritmo de classificação, o SVM, ou máquina de vetores de suporte. O objetivo desse algoritmo é encontrar uma linha que separa as duas classes. A Figura 24 apresenta o código desenvolvido para aplicar o modelo na base de treinamento. A Tabela 3 apresenta o desempenho do modelo aplicado na base de teste.

```
from sklearn.svm import LinearSVC
from sklearn.pipeline import Pipeline

LinearSVC_text = Pipeline([('tfidf', TfidfVectorizer()),(
    'clf', LinearSVC())])
LinearSVC_text.fit(X_train,y_train)
LinearSVC_predictions = LinearSVC_text.predict(X_test)
LinearSVC_cm = confusion_matrix(y_test,LinearSVC_predictions)
print(classification_report(y_test,LinearSVC_predictions))
```

Figura 24: Código para avaliação do algoritmo Linear SVC.

Tabela 3: Desempenho do modelo LinearSVC e TfidfVectorizer.

LinearSVC e TfidfVectorizer				
	precision	recall	f1-score	support
negative	0.78	0.74	0.76	5397
positive	0.92	0.93	0.93	16966
accuracy			0.89	22363
macro avg	0.85	0.83	0.84	22363

weighted avg	0.88	0.89	0.89	22363

5.2 Naive Bayes:

O modelo Naive Bayes é um método de construir classificadores que analisa termos e documentos pelo teorema de bayes. Esse teorema calcula a probabilidade de ocorrência de um evento baseado em conhecimento sobre ele.

No contexto de análise de sentimentos, o classificador Naive Bayes calcula a probabilidade de um tempo ser considerado positivo e negativo e classifica o documento. Por exemplo, se um documento é composto por termos considerados positivos, será classificado como positivo; seguindo o mesmo princípio para termos e documentos negativos.

A Figura 25 apresenta o código desenvolvido para aplicar o modelo na base de treinamento. A Tabela 4 apresenta o desempenho do modelo aplicado na base de teste.

```
from sklearn.naive_bayes import MultinomialNB

MultinomialNB_text = Pipeline([('tfidf',TfidfVectorizer()),(
    'clf',MultinomialNB())])
MultinomialNB_text.fit(X_train,y_train)
MultinomialNB_predictions = MultinomialNB_text.predict(X_test)
cm = confusion_matrix(y_test,MultinomialNB_predictions)
print(classification_report(y_test,MultinomialNB_predictions))
```

Figura 25: Código para avaliação do algoritmo Naive Bayes.

Tabela 4: Desempenho do modelo Naive Bayes e TfidfVectorizer.

Naive Bayes e TfidfVectorizer				
	precision	recall	f1-score	support
negative	0.90	0.38	0.54	5397
positive	0.83	0.99	0.90	16966
accuracy			0.84	22363

macro avg	0.86	0.68	0.72	22363
weighted avg	0.85	0.84	0.81	22363

5.3 Random Forest:

O modelo Random Forest é um algoritmo de aprendizagem de máquina baseado em árvore de decisão. Uma árvore de decisão cria internamente uma representação gráfica das alternativas possíveis, similar a um fluxograma, com o objetivo de descobrir a classificação de uma nova entrada, baseada em variáveis decisórias seguindo o caminho da raiz até as folhas. No entanto, a mesma palavra pode aparecer em classes positivas e negativas. Então, construir apenas uma árvore não é o suficiente para a criar um modelo eficiente. Sendo assim, o algoritmo Random Forest propõe a criação de várias árvores de decisão baseadas em subconjuntos aleatórios de uma base de dados. Dessa forma, podemos classificar um documento baseado não só em uma árvore, mas, sim em uma “floresta”.

A Figura 26 apresenta o código desenvolvido para aplicar o modelo na base de treinamento. A Tabela 5 apresenta o desempenho do modelo aplicado na base de teste.

```
from sklearn.ensemble import RandomForestClassifier

RandomForest_text = Pipeline([('tfidf', TfidfVectorizer()),(
    'clf', RandomForestClassifier())])
RandomForest_text.fit(X_train,y_train)
RandomForest_predictions = RandomForest_text.predict(X_test)
RandomForest_cm = confusion_matrix(y_test,RandomForest_predictions)
print(classification_report(y_test,RandomForest_predictions))
```

Figura 26: Código para avaliação do algoritmo Random Forest.

Tabela 5: Desempenho do modelo Random Forest e TfidfVectorizer.

Random Forest e TfidfVectorizer				
	precision	recall	f1-score	support
negative	0.85	0.64	0.73	5397
positive	0.89	0.96	0.93	16966

accuracy			0.89	22363
macro avg	0.87	0.80	0.83	22363
weighted avg	0.88	0.89	0.88	22363

5.4 Logistic Regression:

O modelo Logistic Regression é um algoritmo utilizado de análise preditiva baseado no conceito de probabilidade, utilizado para problemas de classificação. Na Regressão logística a função Sigmoid é utilizada para mapear previsões de probabilidades entre 0 e 1.

A Figura 27 apresenta o código desenvolvido para aplicar o modelo na base de treinamento. A Tabela 6 apresenta o desempenho do modelo aplicado na base de teste.

```
from sklearn.linear_model import LogisticRegression

LogisticRegression_text = Pipeline([('tfidf',TfidfVectorizer()),(
    'clf',LogisticRegression(max_iter=200))])
LogisticRegression_text.fit(X_train,y_train)
LogisticRegression_predictions = LogisticRegression_text.predict(X_test)
LogisticRegression_cm = confusion_matrix(y_test,LogisticRegression_predictions)
print(classification_report(y_test,LogisticRegression_predictions))
```

Figura 27: Código para avaliação do algoritmo Logistic Regression.

Tabela 6: Desempenho do modelo Logistic Regression e TfidfVectorizer.

Logistic Regression e TfidfVectorizer				
	precision	recall	f1-score	support
negative	0.81	0.71	0.76	5397

positive	0.91	0.95	0.93	16966
accuracy			0.89	22363
macro avg	0.86	0.83	0.84	22363
weighted avg	0.89	0.89	0.89	22363

5.4 Comparação dos classificadores

Analisando a Tabela 7 de desempenho de cada um dos modelos de machine learning aplicados é possível observar que o Naive Bayes e Random Forest foram os modelos que apresentaram o menor desempenho. Enquanto que os modelos Linear SVC e Logistic Regression apresentaram desempenhos similares, e conseguiram classificar bem a base de teste.

Tabela 7: Comparação dos Modelos de Machine Learning.

	Accuracy	**F1-Score Positive**	**F1-Score Negative**	**Precision Positive**	**Precision Negative**	**Recall Positive**	**Recall Negative**
LinearSVC	0.89	0.93	0.76	**0.92**	0.78	0.93	**0.74**
Naive Bayes	**0.84**	**0.90**	**0.54**	**0.83**	0.90	0.99	0.38
Random Forest	0.89	0.93	**0.73**	**0.89**	0.85	0.96	0.64
Logistic Regression	0.89	0.93	0.76	0.91	**0.81**	**0.95**	0.71

5.5 Ajuste de hiperparâmetros

Com o objetivo de experimentar diferentes parâmetros no projeto, vamos utilizar o *Pipeline* e *GridSearchCV* da biblioteca sklearn, que vão nos permitir automatizar os fluxos de processos e testar combinações de parâmetros de ajuste do modelo.

Os parâmetros testados serão: *“max_df”* que é usado para remover termos que aparecem com muita frequência; *“ngram_range”* que irá agrupar as palavras de maneira a manter o contexto; *“C”* que indica o quanto é necessário evitar uma classificação incorreta, para grandes valores o hiperplano escolhido poderá ter uma margem menor se os valores forem corretamente classificados; e o *“class_weight”* que pode balancear pesos diferentes para as classes inversamente proporcionais a frequência da classe.

'tfidf__max_df': (0.25, 0.5, 0.75),

'tfidf__ngram_range': [(1, 1), (1, 2), (1, 3)]

"clf__estimator__C": [0.01, 0.1, 1],

"clf__estimator__class_weight": ['balanced', None]

```
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV
from sklearn.multiclass import OneVsRestClassifier
from sklearn.pipeline import Pipeline

pipeline = Pipeline([('tfidf', TfidfVectorizer()),(
    'clf', OneVsRestClassifier(LinearSVC())),])

parameters = {
    'tfidf__max_df': (0.25, 0.5, 0.75),
    'tfidf__ngram_range': [(1, 1), (1, 2), (1, 3)],
    'clf__estimator__C': [0.01, 0.1, 1],
    'clf__estimator__class_weight': ['balanced', None],
}

grid_search_tune = GridSearchCV(
    pipeline, parameters, cv=2, n_jobs=2, verbose=3)
grid_search_tune.fit(X_train, y_train)

print("Best parameters set:")
print (grid_search_tune.best_estimator_.steps)

print ("Applying best classifier on test data:")
best_clf = grid_search_tune.best_estimator_
predictions = best_clf.predict(X_test)

print (classification_report(y_test, predictions))
```

Figura 28: Código para ajuste de hiperparâmetros no algoritmo Linear SVC.

```
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('clf', OneVsRestClassifier(LogisticRegression(solver='sag'))),
])
parameters = {
    'tfidf__max_df': (0.25, 0.5, 0.75),
    'tfidf__ngram_range': [(1, 1), (1, 2), (1, 3)],
    "clf__estimator__C": [0.01, 0.1, 1],
    "clf__estimator__class_weight": ['balanced', None],
}

grid_search_tune = GridSearchCV(
    pipeline, parameters, cv=2, n_jobs=2, verbose=3)
grid_search_tune.fit(X_train, y_train)

print("Best parameters set:")
print (grid_search_tune.best_estimator_.steps)

print ("Applying best classifier on test data:")
best_clf = grid_search_tune.best_estimator_
predictions = best_clf.predict(X_test)

print (classification_report(y_test, predictions))
```

Figura 29: Código para ajuste de hiperparâmetros no algoritmo Logistic Regression.

Os melhores parâmetros para o algoritmo Linear SVC foram max_df=0.25, ngram_range=(1, 2) e C=1. No caso do Logistic Regression foram max_df=0.25 e C=1. Na Tabela 8 apresentamos os comparativos entres os modelos Linear SVC e Logistic Regression com o ajuste de hiperparâmetros. Podemos observar que o Linear SVC apresentou uma melhora no desempenho, enquanto o Logistic Regression não houve alteração.

Tabela 8: Comparação dos Modelos de Machine Learning LinearSVC e Logistic Regression.

	Accuracy	**F1-Score Positive**	**F1-Score Negative**	**Precision Positive**	**Precision Negative**	**Recall Positive**	**Recall Negative**
LinearSVC	**0.90**	0.93	0.79	0.93	0.81	0.94	0.77
Logistic Regression	0.89	0.93	**0.76**	**0.91**	0.81	0.95	0.71

6. Apresentação dos Resultados

O processamento de linguagem natural associado a algoritmos de machine learning são amplamente utilizados para análise de sentimento em redes sociais, graças ao alto número de informações disponíveis, que abrange todo o tipo de consumidor. Comentários em sites de venda são fundamentais para conhecer os pontos fracos e fortes do seu produto, além de que essas avaliações impactam diretamente na opinião de consumidores em potencial. A seguir é apresentado o modelo Canvas com a *workflow* da análise de dados apresentada neste trabalho.

7. Links

Link para o vídeo, documento e código está disponível no repositório do GitHub.