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

Alexandre Henriques Nunes

MODELOS DE ANÁLISE DE SENTIMENTO UTILIZANDO AVALIAÇÕES ONLINE DE PRODUTOS E SERVIÇOS

Belo Horizonte
2020

Alexandre Henriques Nunes

MODELOS DE ANÁLISE DE SENTIMENTO UTILIZANDO AVALIAÇÕES ONLINE DE PRODUTOS E SERVIÇOS

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
2020

SUMÁRIO

1. Introdução

1.1. Contextualização

Desde sua criação, a internet tem se mostrado um instrumento essencial para a sociedade. Segundo pesquisa publicada pela DataReportal[1], dos 7.79 bilhões de habitantes mundialmente, 4.57 bilhões têm acesso a internet, totalizando uma penetração de 59%. Além disso, a pesquisa também mostra a importância das mídias sociais, com 3.96 bilhões de usuários ativos, totalizando uma penetração de 51%.

No Brasil os números são ainda maiores. De acordo com a TIC Domicílios 2019[2], 80% da população brasileira utiliza a internet e 76% desses usuários utilizam redes sociais, 59% procurou informações sobre produtos e serviços e 39% compraram produtos e serviços pela internet.

Os números comprovam que a internet e as mídias sociais tem cada vez mais impacto nas nossas vidas. Segundo uma pesquisa do Instituto QualiBest[3] com internautas brasileiros, 50% dos entrevistados dizem confiar em influenciadores digitais na sua tomada de decisão de compra de um produto e apenas 16% confiam nas propagandas de TV e Rádio. Além disso, conforme o relatório Social Media Trends 2019 da Rock Content[4], 96,2% das empresas estão presentes nas redes sociais e 62,6% consideram que as redes sociais têm um papel muito importante para as empresas.

Com o crescimento da utilização da internet e das mídias sociais a quantidade de dados disponíveis cresce em ritmo acelerado. Segundo o relatório da Seagate[5], teremos aproximadamente 175 Zettabytes[A] de dados até o ano de 2025.

Dessa forma fica inviável para um a empresa fazer o processamento manual de toda essa informação, sendo crucial a utilização da tecnologia para nos auxiliar na tomada de decisão. Assim uma empresa pode monitorar a reação do público aos seus produtos e serviços.

[A] Um Zettabyte é equivalente a 1 trilhão de Gigabytes

A análise de sentimentos das opiniões na internet assume grande importância nessa tarefa. O Processamento de Linguagem Natural (PLN) – área responsável pelo desenvolvimento de modelos computacionais relacionados a informações expressas em língua natural – atrelado a abordagens baseadas em aprendizado de máquina constituem uma ferramenta com bastante eficacia para solução desse problemas.

1.2. O Problema Proposto

As mídias sociais estão cheias de opiniões sobre produtos e serviços e são muito utilizadas no processo de decisão de compra pelos seus usuários. Assim monitorar essas opiniões e ter certeza que seu produto ou serviço tem boa receptividade pelo consumidor é essencial para empresas que querem ter vantagem competitiva. Segundo os dados da Garnet[6], não responder corretamente os consumidores em redes sociais pode gerar um aumento de 15% nas taxas de cancelamentos.

Monitorar as mídias sócias envolve um grande volume de dados não estruturados presentes principalmente em redes sociais e sites de avaliação de produtos e serviços.

Para utilização de algorítimos de aprendizado de máquina se faz necessário a utilização de dados históricos para alimentar tais algoritmos, possibilitando o “aprendizado” – ou calibragem. Portanto é necessário ter uma base de dados de avaliações históricas de tal forma a avaliar novas avaliações publicadas na rede. Assim é necessário utilizar algoritmos para capturar os dados e montar essa base, já que seu tamanho afetará a eficacia do aprendizado, assim sendo, fica inviável montar a base de dados manualmente.

Além da criação da base, outro desafio é estruturar os dados, já que cada site tem sua própria estrutura. Outro detalhe importante é a maneira de comunicação dos usuários em redes sociais, que muitas vezes é feita de maneira informal e com erros de gramática e ortografia, dificultando uma análise sistemática dessa comunicação.

Este trabalho tem como motivação descrever uma possível abordagem para a criação de um modelo de análise de sentimento em língua portuguesa utilizando o processamento de linguagem natural em avaliações de produtos e serviços disponíveis na internet.

2. Coleta de Dados

Os dados utilizados foram extraídos das principais empresas brasileiras de comércio eletrônico que permitem comentários sobre seus produtos. As empresas selecionadas foram baseados no valor de mercado conforme tabela abaixo:

Tabela 1: Valor de mercado de empresas varejistas com presença online

Empresa	Valor de Mercado (30/07/20)
Magazine Luiza - magazineluiza.com.br	R$ 135 bilhões
B2W - americanas.com.br	R$ 68 bilhões
Via Varejo - extra.com.br	R$ 32 bilhões

Fonte: B3 - Brasil Bolsa Balcão

Como os sites possuem uma quantidade muito grande de produtos a extração focou nos produtos do seguintes segmentos:

- Eletroeletrônicos: TVs, celular e notebooks
- Eletrodomésticos: máquina de lavar
- Beleza: perfume
- Móveis: sofás

Também foram extraídos dados referentes a avaliação de restaurantes do site Zomato (zomato.com.br) das cidades com mais restaurantes na plataforma:

- São Paulo
- Rio de Janeiro
- Salvador
- Porto Alegre
- Brasília

E por último foram extraídos avaliações do site ReclameAqui (reclameaqui.com.br) das empresas com mais reclamações segundo o ranking[7] do próprio site:

- Correios
- Netshoes
- Caixa Econômica Federal
- Vivo
- Uber
- C&A
- Decolar
- NET
- Mercado Pago

A extração foi feita utilizando scripts em Python[8] versão 3.7.5 (64 bits) em conjunto com a biblioteca Scrapy[9] versão 2.2.0, que é especializada em extração de dados de sites.

O Scrapy permite a criação de *Spiders*, que são classes com as definições de como a informação de um site deve ser extraída. Basicamente o *Spider* baixa o conteúdo *HTML* do site e com isso é possível extrair o dado via *XPath* ou seletores *CSS (Cascading Style Sheets)*.

Nesse trabalho foram criados 5 *Spiders* distintos. Abaixo segue o código do *Spider* do Magazine Luiza que utiliza seletores *CSS* e *JSON (JavaScript Object Notation)* para obter as avaliações dos produtos:

```
import json
import scrapy
from scraper.items import ScraperItem

class Magalu(scrapy.Spider):
    name = 'Magalu'
    categories = [
        "smartphone/celulares-e-smartphones/s/te/tcsp",  # Celulares
        "tvs/tv-e-video/s/et/tves",  # TVs
        "notebook/informatica/s/in/note",  # Notebook
        "maquina-de-lavar/eletrodomesticos/s/ed/lava",  # Máquina de Lavar
        "sofas/moveis/s/mo/msof",  # Sofas
        "perfume/beleza-e-perfumaria/s/pf/pftm",  # Perfume
    ]

    def start_requests(self):
        """ Generate initial requests """
        for category in self.categories:
            for page in range(50): # Load 50 pages
                url = self.search_url(category, page)
                yield scrapy.Request(url)

    def search_url(self, category, page):
        """ Return search URL """
        return 'https://www.magazineluiza.com.br/{}?sort=type%3AsoldQuantity
%2Corientation%3Adesc&page={}'.format(category, page + 1)

    def review_url(self, product_id, page):
        """ Return review URL """
```

```
        if len(product_id) == 7:
            product_id += "00"

        return 'https://www.magazineluiza.com.br/review/{}/?
page={}'.format(product_id, page)

    def parse_reviews(self, response):
        """ Parse review page """
        try:
            data = json.loads(response.text)['data']
        except Exception:  # pylint: disable=broad-except
            return

        for review in data['objects']:
            yield ScraperItem(
                title=review['title'],
                text=review['review_text'],
                rating=review['rating'],
            )

        if data['pages'] > response.meta['page']:
            product_id = response.meta['id']
            next_page = response.meta['page'] + 1
            url = self.review_url(id, next_page)

            yield scrapy.Request(url, callback=self.parse_reviews, meta={'page':
next_page, 'id': product_id})

    def parse(self, response):
        """ Parse search page """
        scripts = response.css('script[type="application/ld+json"]')

        for script in scripts:
            if script is not None:
                content = script.css('::text').get()

                if "sku" in content:
                    try:
                        product_id = json.loads(content)['sku']
                        url = self.review_url(id, 1)
                        yield scrapy.Request(url, callback=self.parse_reviews,
meta={'page': 1, 'id': product_id})
                    except Exception:  # pylint: disable=broad-except
                        continue
```

Os *Spiders* produzem um *dataset* em arquivo *CSV (Comma-separated values)* com os dados abaixo:

Tabela 2: Descrição do dataset

Nome da coluna/campo	Descrição	Tipo
Title	Título da avaliação	String
Text	Texto da avaliação	String
Rating	Nota da avaliação	Double
Source	Site de origem	String

Fonte: Autor

O formato *CSV* foi escolhido pela sua simplicidade e por ser suportado por aplicativos de planilha, como o Microsoft Excel e o LibreOffice Calc. Abaixo segue um *snapshot* do arquivo puro e da exibição do LibreOffice Calc*:*

Figura 1: Arquivo *CSV* com as avaliações

```
"source","title","text","rating"
"B2W","Super bom","Jogo de telefone muito bacana, com otimo alcance e facil
instalacao.",5.0
"B2W","Goste ido produto.","Gostei, de acordo com as especificações do site. Funciona
perfeitamente",5.0
"B2W","Otimo aparelho","Produto muito bom. O volume de ligação é bom. O de toque, muito
bom tb. Podem confiar. Fácil instalação.",4.0
"B2W","Gostei, boa qualidade","Gostei, boa qualidade, produto chegou em perfeitas condi-
ções de uso.",3.0
"B2W","Ótimo","Ótimo Produto e chegou no prazo estabelecido.....",5.0
"B2W","Ruim pois n foi possível usar lo , pois n registra ligações recebidas.","Ruim pois
infelizmente n possível usar lo , pois n registra ligações recebidas.. Estou devolvendo n
deu a opcao de trocar lo.",1.0
"B2W","Gostei muito do produto.","Ótimo atendimento. Entrega rápida e eficiente. Recomen-
do o produto e a loja.",5.0
"B2W","Bina.","Presados Senhores. Atende as minhas necessidades!. Abs. Joaquim",3.0
"B2W","Chegou com defeito","Minha expectativa era identificar a chamada e saber a que ho-
ras a ligação ocorreu. Não há essa possibilidade pois o botão para ajuste de data e horá-
rio veio com defeito. #chateada",1.0
"B2W","Excelete custo beneficio","Atendeu perfeitamente minhas necessidades, no meu caso
um pequeno escritorio com com 8 usuarios",5.0
```

Fonte: Autor

Figura 2: Exibição do arquivo *CSV* no LibreOffice Calc

	A	B	C	D
1	source	title	text	rating
2	B2W	Super bom	Jogo de telefone muito bacana, com otimo alcance e facil instalacao.	5
3	B2W	Goste ido produto.	Gostei, de acordo com as especificações do site. Funciona perfeitamente	5
4	B2W	Otimo aparelho	Produto muito bom. O volume de ligação é bom. O de toque, muito bom tb. Podem confiar. Fácil instalação.	4
5	B2W	Gostei, boa qualidade	Gostei, boa qualidade, produto chegou em perfeitas condições de uso.	3
6	B2W	Ótimo	Ótimo Produto e chegou no prazo estabelecido.....	5
7	B2W	Ruim pois n foi possível usar lo , pois n registra ligações recebidas.	Ruim pois infelizmente n possível usar lo , pois n registra ligações recebidas.. Estou devolvendo n deu a opcao de trocar lo.	1
8	B2W	Gostei muito do produto.	Ótimo atendimento. Entrega rápida e eficiente. Recomendo o produto e a loja.	5
9	B2W	Bina.	Presados Senhores. Atende as minhas necessidades!. Abs. Joaquim	3
10	B2W	Chegou com defeito	Minha expectativa era identificar a chamada e saber a que horas a ligação ocorreu. Não há essa possibilidade pois o botão para ajuste de data e horário veio com defeito. #chateada	1
11	B2W	Excelete custo beneficio	Atendeu perfeitamente minhas necessidades, no meu caso um pequeno escritorio com com 8 usuarios	5

Fonte: Autor

3. Processamento/Tratamento de Dados

3.1. Ferramentas Utilizadas

Todo o processamento e tratamento dos dados foi feito utilizando scripts em Python em conjunto com as bibliotecas listadas abaixo:

Tabela 3: Bibliotecas utilizadas

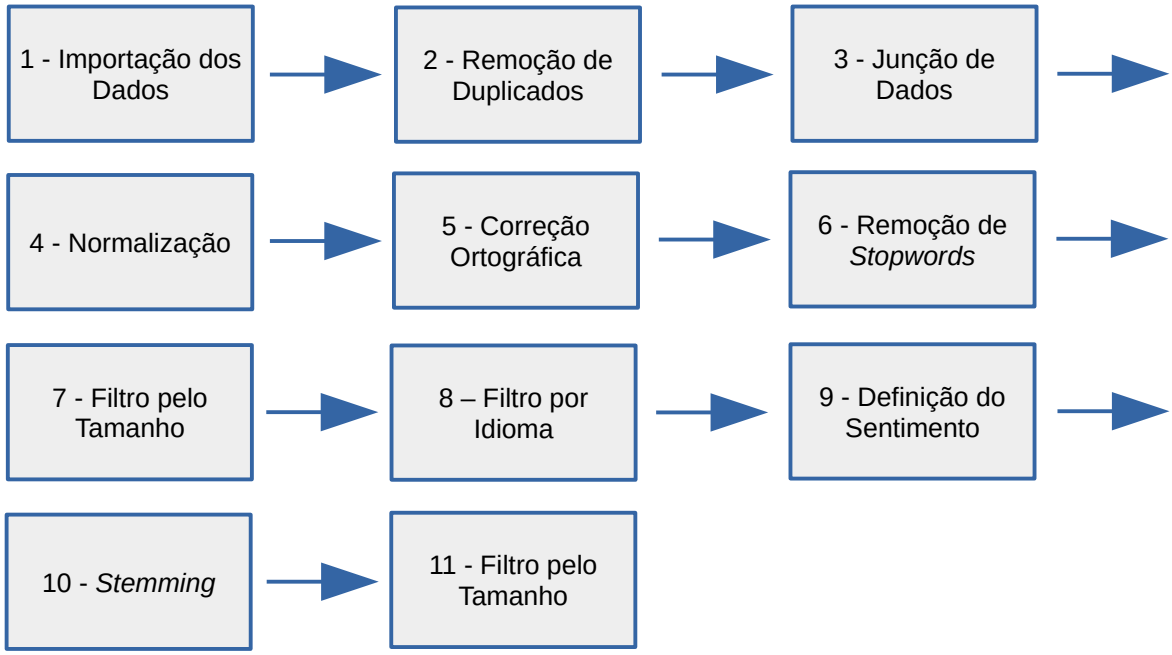
Biblioteca	Descrição
Pandas[10]	Biblioteca para manipulação e análise de dados. Em particular, oferece estruturas e operações para manipular tabelas numéricas e séries temporais.
unidecode[11]	Biblioteca para conversão de texto em Unicode para o formato ASCII.
Regex[12]	Regex provê uma forma concisa e flexível de identificar cadeias de caracteres de interesse, como caracteres particulares, palavras ou padrões de caracteres.
Spacy[13]	Biblioteca para processamento avançado de linguagem natural.
NLTK[14]	Plataforma para processamento de dados em linguagem humana.
symspellpy[15]	Biblioteca de correção ortográfica em Python.
WhatTheLang[16]	Biblioteca para rápida detecção de idioma.

Fonte: Autor

3.2. Importação e Tratamento dos Dados

A importação e tratamento dos dados é feita em 11 estapas que podem ser visualizadas no fluxograma abaixo.

Figura 3: Fluxo de Importação e Tratamento dos Dados



Fonte: Autor

3.2.1. Importação dos Dados

A importação dos dados é feita utilizando a biblioteca Pandas, que suporta a importação de arquivos de *CSV* de maneira simplificada. Abaixo segue o código da importação do arquivo em um *dataframe* Pandas. No total são importados 125.771 registros no dataframe do Pandas.

```
1 import pandas as pd
2 df = pd.read_csv("data/data.csv")
```

3.2.2. Remoção de Duplicados

A remoção de registro duplicados é feita utilizando novamente a biblioteca Pandas que tem o método drop_duplicates. Assim podemos alterar a importação para já remover os registros duplicados.

```
1 import pandas as pd
2 df = pd.read_csv("data/data.csv").drop_duplicates()
```

São removidos 13.453 registros duplicados, totalizando agora 112.318 registros no *dataframe* Pandas.

3.2.3. Junção de Dados

Para a análise de sentimento vamos considerar tanto o título quanto o texto das avaliações. O código abaixo faz a junção de ambos campos:

```
df["text"] = df.apply(lambda row: f"{row.title} {row.text}", axis=1)
```

3.2.4. Normalização

A normalização de dados é necessária para remover inconsistências no texto, possibilitando seu processamento posteriormente. Ela consiste em diversas sub-etapas descritas abaixo:

1. Conversão para minúsculas
2. Substituição de entidades *HTML*, como “ç” por “ç”
3. Remoção de tags *HTML*
4. Remoção de *URLs*
5. Remoção de palavras com números, como por exemplo códigos de rastreio “SR12345BR”
6. Remoção de acentos e caracteres que não sejam alfabéticos
7. Remoção de espaços duplos
8. Remoção de duplicações de uma mesma vogal, como por exemplo “amo-oooo” será convertida em “amo”
9. Remoção de palavras com menos de 3 caracteres
10. Remoção dos espaços no começo e no final da *string*

O código abaixo realiza todas as sub-etapas descritas acima:

```
import html
import re
import unicodedata
import unidecode

def normalize_text(text):
    # 1 - Conversão para minúsculas
    text = text.lower()

    # 2 - Substituição de entidades HTML
    text = html.unescape(text)

    # 3 - Remoção de tags HTML
    text = re.sub(r"<[^.]*>", " ", text)

    # 4 - Remoção de URLs
    text = re.sub(
        r"https?:\/\/(www\.)?[-a-zA-Z0-9@:%._\+~#=]{1,256}\.[a-zA-Z0-9()]{1,6}\b([-a-zA-Z0-9()@:%_\+.~#?&//=]*)",
```

```
        " ", text)

    # 5 - Remoção de palavras com números
    text = re.sub(r"\b[0-9a-z]*[0-9]+[0-9a-z]*\b", " ", text)

    # 6 - Remoção de acentos e caracteres que não sejam alfabéticos
    text = unidecode.unidecode(text)
    text = "".join(ch for ch in text if unicodedata.category(ch)[0] != "C")
    text = re.sub(r"[^a-z ]", " ", text)

    # 7 - Remoção de espaços duplos
    text = re.sub(r"\s\s+", " ", text)

    # 8 - Remoção de duplicações de uma mesma vogal
    text = re.sub(r"([aeiou])\1+", r"\1", text)

    # 9 - Remoção de palavras com menos de 3 caracteres
    text = re.sub(r"\b[a-z][a-z]?\b", "", text)

    # 10 - Remoção dos espaços no começo e no final da string
    text = text.strip()

df["text"] = df.apply(lambda row: normalize_text(row.text), axis=1)
```

3.2.5. Correção Ortográfica

Existem diversas formas de implementação para corretores ortográficos, talvez o mais conhecido e simples seja o método proposto por NORVING, Peter (2007)[17], que corrige os erros baseado no menor número de edições. Porém esse método é relativamente lento e dado o tamanho da base de dados ficaria inviável utilizar o mesmo.

A solução foi utilizar a biblioteca symspellpy[15], que é a implementacão em Python do método SymSpell proposta dor GARBE, Wolf (2019)[18] que diminui a quantidade possível de edições para correção de palavras e possibilida fazer a correção de diversas palavras ao mesmo tempo, ao invés de uma palavra por vez conforme o método de NORVING.

Para utilizarmos o symspellpy precisamos criar dois arquivos que serão utilizados como base para a correção ortográfica. O primeiro arquivo deve conter os unigrams do idioma em conjunto com a frequência, o segundo arquivo é bastante semelhante, porém ao invés de unigrams ele é composto por bigrams. O código abaixo foi utilizado para criar os arquivos baseado nos *corpus* Floresta, Machado e Mac-Morpho[19] da biblioteca *NLTK*:

```
import nltk
from nltk.corpus import floresta, machado, mac_morpho

nltk.download("floresta")
nltk.download("machado")
nltk.download("mac_morpho")

sents  = [normalize_text(" ".join(x)).split() for x in floresta.sents()]
sents += [normalize_text(" ".join(x)).split() for x in machado.sents()]
sents += [normalize_text(" ".join(x)).split() for x in mac_morpho.sents()]
```

```
11
12 unigrams = [item for sublist in sents for item in sublist]
13 unigrams = nltk.probability.FreqDist(unigrams)
14
15 file = open("data/unigrams.txt", "w")
16 for k, v in unigrams.items():
17     file.write(f"{k} {v}\n")
18 file.close()
19
20 bigrams = []
21
22 for sent in sents:
23     bigrams += list(nltk.bigrams(sent))
24
25 bigrams = nltk.probability.FreqDist(bigrams)
26
27 file = open("data/bigrams.txt", "w")
28 for k, v in bigrams.items():
29     file.write(f"{' '.join(k)} {v}\n")
30 file.close()
```

Com os arquivos criados podemos instanciar o SymSpell, carregar os arquivos e chamando os métodos `load_dictionary` e `load_bigram_dictionary`:

```
1  from symspellpy.symspellpy import SymSpell
2
3  spell = SymSpell()
4
5  spell.load_dictionary("data/unigrams.txt", 0, 1)
6  spell.load_bigram_dictionary("data/bigrams.txt", 0, 2)
7
8  def spell_check(text, spell):
9      suggestions = spell.lookup_compound(text, 2)
10     return suggestions[0].term if suggestions else text
11
12 df["text"] = df.apply(lambda row: spell_check(row.text, spell), axis=1)
```

3.2.6. Remoção de *Stopwords*

Uma etapa muito comum no processamento de linguagem natural é a remoção de *stopwords*, que são palavras que tem pouca importância no significado das sentenças. Podemos citar como exemplo de *stopwords* em português os artigos “a”, “o”, “as” e “os”.

As bibliotecas NLTK e a Spacy contém listas de *stopwords* em português. O autor também criou uma lista de *stopwords* adicionais baseada em listas disponíveis na internet[20] e palavras adicionadas manualmente. Abaixo segue o código para carregar a lista de *stopwords* que também são normalizadas utilizando a mesma função definida na etapa 3.2.4:

```
1 import spacy
2 from nltk.corpus import stopwords
3
4 spacy.load("pt")
5 nltk.download("stopwords")
6
```

```
stop_words  = stopwords.words("portuguese")
stop-words += list(spacy.lang.pt.stop_words.STOP_WORDS)

stop_words = [normalize_text(x) for x in result]

with open("data/stop-words.txt", "r") as reader:
    stop_words += [normalize_text(x) for x in reader.read().splitlines()]

stop_words = set(stop_words)

def remove_words(text, words):
    text_split = text.split()
    return " ".join([x for x in text_split if x not in words])

df["text"] = df.apply(lambda row: remove_words(row.text, stop_words), axis=1)
```

3.2.7. Filtro pelo Tamanho

Para remover textos com pouca informação foi aplicado um filtro que remove registros com tamanho menor que 11 caracteres ou com menos do que 4 palavras. O código abaixo executa esse filtro:

```
df = df[(df["text"].str.len() > 10) & (df["text"].str.count(" ") >= 4)]
```

Esse filtro remove 10.023 registros, totalizando 102.295 registros restantes.

3.2.8. Filtro por Idioma

Alguma avaliações não estão escritas em português e como o foco deste trabalho é a análise de sentimento para o idioma português foi aplicado um filtro nas avaliações que não estavam nesse idioma.

Para identificar o idioma de uma avaliação foi utilizado a biblioteca WhatTheLang[16] que mostrou ter uma boa performance de processamento, porém com alguns falso-positivos, portanto foi utilizado uma lista de palavras adicionais em português para aumentar a eficácia da identificação. Abaixo está o código para a identificação e o filtro de idioma dos registros:

```
from whatthelang import WhatTheLang

def language(text, wtl):
    words = [
        "recomendo", "amei", "entrega", "otim[ao]", "excelente", "rapida",
        "celular", "gostei", "facil", "lindo", "bonito", "comprei", "legal",
        "perfume", "preco", "tela", "pra", "lento", "problema", "pelicula",
        "memoria", "cabelo", "ultima",
    ]

```

```
    if re.search(rf'\b({"|".join(words)})\b', text):
        result = "pt"
    else:
        result = wtl.predict_lang(text)

    return result

wtl = WhatTheLang()

df["language"] = df.apply(lambda row: language(row.text, wtl), axis=1)
df = df[df["language"] == "pt"]
```

Esse filtro remove 1.255 registros, totalizando 101.040 registros restantes.

3.2.9. Definição do Sentimento

A base de dados possui avaliações com suas respectivas notas que variam normalmente de 0 a 5, com exceção do ReclameAqui que tem suas notas entre 0 e 10.

O primeiro passo foi normalizar as notas na escala de 0 e 5, portanto para isso as notas do ReclameAqui foram divididas por 2. A seguir foi definido que notas abaixo de 3 significavam um sentimento “negativo” e notas maiores ou iguais a 3 sinigicavam um sentimento “positivo”. O codigo abaixo adiciona a coluna sentiment a base de dados:

```
def sentiment(rating, source):
    if source == "ReclameAqui":
        rating /= 2

   return "negative" if rating < 3 else "positive"

df["sentiment"] = df.apply(lambda row: sentiment(row.rating, row.source), axis=1)
```

3.2.10. *Stemming*

Assim como a remoção de *stopwords*, o *stemming* – ou stemização – do texto também é uma etapa muito comum no processamento de linguagem natural. Ela consiste em reduzir palavras flexionadas (ou derivadas) à sua raiz. Como exemplo podemos citar as palavras “pagar” e “pagamento” que ambas tem como raiz “pag”, portanto após o *stemming* somente a raiz continuaria presente no texto.

Utilizamos o stemmer em português da biblioteca NLTK, chamado RSLPStemmer. Abaixo está o codigo da etapa de *stemming*:

```
from nltk.stem import RSLPStemmer

stemmer = RSLPStemmer()

def stemmed_text(text, stemmer):
    return " ".join([stemmer.stem(x) for x in text.split()])

df["text"] = df.apply(lambda row: stemmed_text(row.text, stemmer), axis=1)
```

3.2.11. Filtro pelo Tamanho

Finalmente aplicamos novamente o mesmo filtro da etapa 3.2.7. Nesta etapa nenhum registro é removido, portanto o número final de registro se mantem em 101,040.

3.2.12. Etapas Descartadas

Outra etapa comum no processamento de linguagem natural é o *Lemmatization* – ou lematização –, que é aplicado após o *Stemming* com o intuito convertar as palavras para sua forma inflecionada. Está etapa não foi adicionada no modelo, pois após alguns testes foi identificado que ela agregava pouco ou nada para a sua eficâcia.

Também é comum a utilização do *Part-of-speech tagging (POS)*, que consiste em agregar as propriedades gramaticais das palavras no dataset a ser analisado. O pacote NLTK também consegue um component para *POS* em português, mas o autor não conseguiu obter nehuma melhoria com esta etapa, portanto ela também não foi inclusa no modelo.

4. Análise e Exploração dos Dados

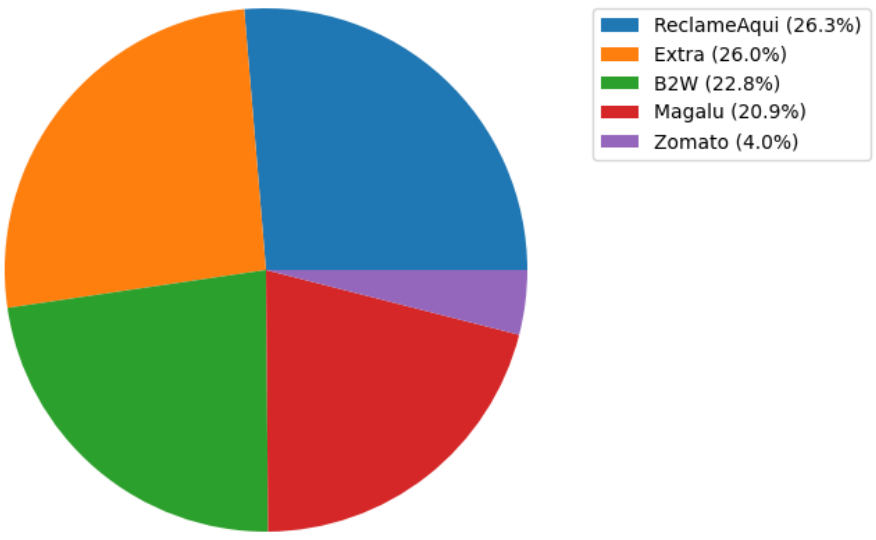
Com a base montada podemos gerar uma nuvem de palavras para identificarmos as raízes das palavras mais comuns. Podemos perceber que raízes palavras relacionadas a sentimentos, como “gost” e “otim” e raízes de palavras relacionados a produtos e serviços, como “produt”, “atend” e “compr”.

Figura 4: Núvem de Palavras

Fonte: Autor

A maior parte das avaliações foi obtida do RelameAqui, porém as outras fontes também possuem uma quantidade expressiva de avaliações, sendo a única exceção o Zomato, já que o site não possui tanta avaliações comparado às outras fontes.

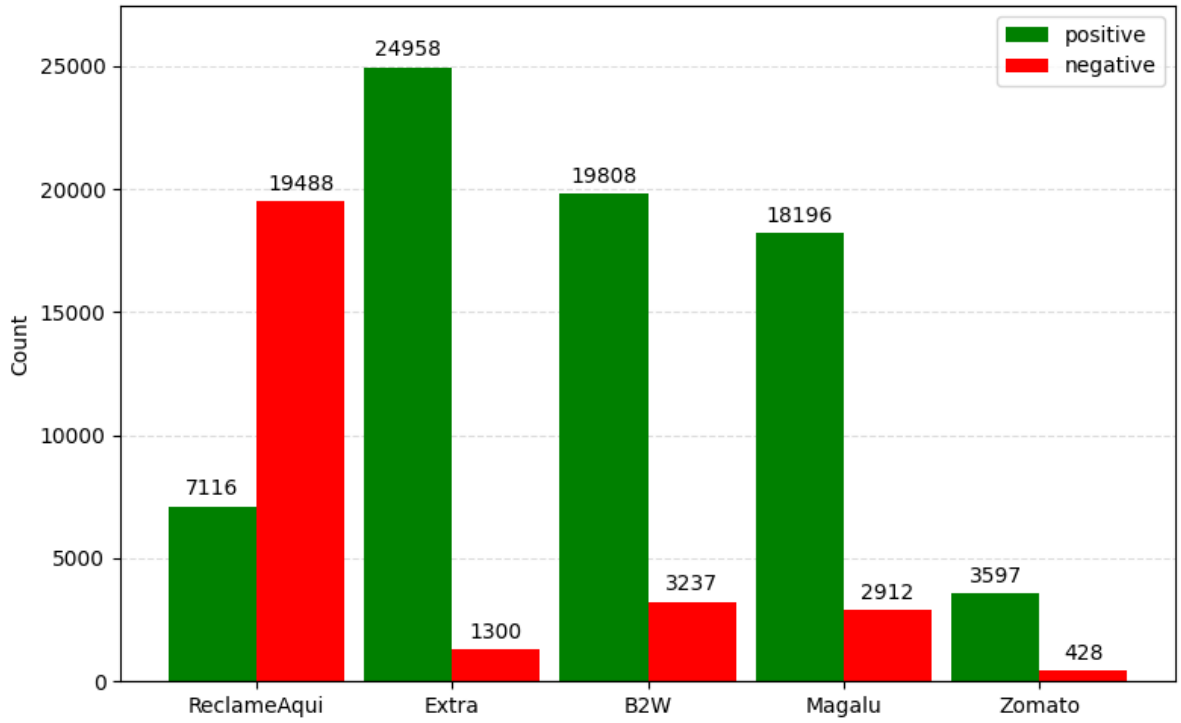
Figura 5: Distribuição de Avaliações por Origem



Fonte: Autor

Analisando a distribuição das avaliações por origem e sentimento na figura 6, podemos observar que a única fonte que possui uma quantidade maior de avaliações negativas é o ReclameAqui, o que faz sentido, pois é uma plataforma especializada em reclamações sobre produtos e serviços. Já as outras fontes não tem esse foco e de maneira geral as avaliações são feitas pelos consumidores dos produtos e serviços, portanto quanto mais consumidores mais avaliações serão feitas. Considerando que os produtos e serviços mais vendidos normalmente são os com melhor custo benefício para o consumidor é de se esperar uma quantidade maior de avaliações positivas conforme observado.

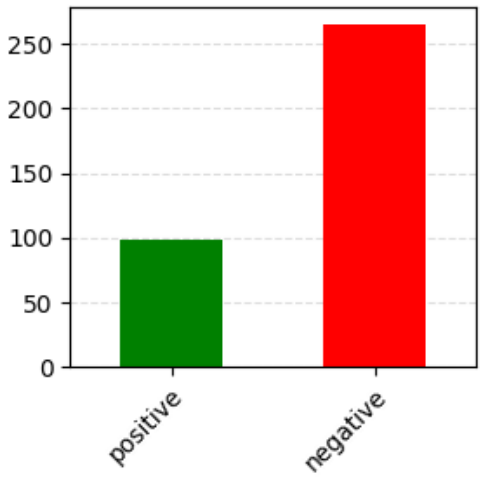
Figura 6: Distribuição de Avaliações por Origem e Sentimento



Fonte: Autor

Também é interessante notar que as avaliações negativas tem um tamanho 2,7 vezes maior, sendo em média que uma avaliação negativa tem 265 caracteres enquanto uma positiva tem 98 caracteres.

Figura 7: Quantidade média de caracteres por sentimento



Fonte: Autor

5. Criação de Modelos de Machine Learning

5.1. Ferramentas Utilizadas

Existem diversas bibliotecas em Python para Machine Learning. Abaixo estão as ferramentas selecionadas para utilização neste trabalho:

Tabela 4: Bibliotecas utilizadas

Biblioteca	Descrição
scikit-learn[21]	Biblioteca de aprendizado de máquina que inclui vários algoritmos de classificação, regressão e agrupamento. É projetada para interagir com as bibliotecas NumPy e SciPy.
TensorFlow[22]	Biblioteca de aprendizado de máquina para criação e treinamento de redes neurais para detectar e decifrar padrões e correlações, análogo à forma como humanos aprendem e raciocinam. Ele é usado tanto para a pesquisa quanto produção no Google.
XGBoost[23]	Biblioteca focada no algoritmo de classificação Gradient Boosting.
LightGBM[24]	Biblioteca desenvolvida pela Microsoft focada no algoritmo de classificação Gradient Boosting.

CatBoost[25]	Biblioteca focada no algoritmo de classificação Gradient Boosting.

Fonte: Autor

A biblioteca scikit-learn possui diversos componentes auxiliares que são utilizados em conjunto com os algorítmos de aprendizado de máquina. Isso faz com que ela seja uma das bibliotecas mais utilizadas nessa área, sendo que todas as outras bibliotecas listadas acima tem classes especiais que se integram de maneira transparente à biblioteca scikit-learn. Desta forma este trabalho utiliza como base a biblioteca scikit-learn e as classes de integração das outras bibliotecas.

5.2. Vetorizadores

Alguns dos componentes auxiliares da biblioteca scikit-learn são os vetorizadores – ou vectorizers –, que possibilitam a representação de textos em vetores informativos. Abaixo será descrito os vetorizadores utilizados nesse trabalho:

5.2.1. Unigram

Unigrams são vetores com palavras individuais. Abaixo está a configuração utilizada para gerar os unigrams em conjunto com suas frequências:

```
1 from sklearn.feature_extraction.text import CountVectorizer
2 CountVectorizer(max_features=2500, min_df=8, max_df=0.8)
```

A tabela 5 mostra os 5 unigrams mais frequentes na base de dados:

Tabela 5: Unigrams mais frequentes

Termo	Frequência
nao	116.084
muit	52.565
produt	46.605
compr	35.956
atend	29.629

Fonte: Autor

5.2.2. Bigram

Bigrams são vetores com 2 palavras consecutivas. Neste trabalho utilizamos o conjunto de unigrams mais bigrams e suas frequências, conforme código abaixo:

```
1 from sklearn.feature_extraction.text import CountVectorizer
2 CountVectorizer(max_features=2500, min_df=8, max_df=0.8, ngram_range=(1, 2))
```

A tabela 6 mostra os 5 bigrams mais frequentes na base de dados:

Tabela 6: Bigrams mais frequentes

Termo	Frequência
muit bom	11.796
entr contat	7.745
cust benefici	5.030
merc pag	4.395
gost muit	4.361

Fonte: Autor

5.2.3. Trigram

Trigrams são vetores com 3 palavras consecutivas. Neste trabalho utilizamos o conjunto de unigrams mais bigrams mais trigrams e suas frequências, conforme código abaixo:

```
1 from sklearn.feature_extraction.text import CountVectorizer
2 CountVectorizer(max_features=2500, min_df=8, max_df=0.8, ngram_range=(1, 3))
```

A tabela 7 mostra os 5 trigrams mais frequentes na base de dados:

Tabela 7: Trigrams mais frequentes

Termo	Frequência
produt muit bom	2.400
otim cust benefici	1.493
entreg sup rap	1.471
gost muit produt	1.185
muit bom produt	1.017

Fonte: Autor

5.2.4. 2-Skip-Bigram

Skip-grams são representações que tentam diminuir o problema de esparcividade no dados gerados pelo n-grams (unigram, bigram, trigram, etc). Para fazer isso os skip-grams não consideram somente palavras consecutivas na sua formulação, mas também pulam algumas palavras de tal forma a obter uma maior combinação de palavras do texto.

Abaixo segue um exemplo de 2-skip-bigram para o texto “o livro está sobre a mesa”:

- o livro
- o está
- o sobre
- livro está
- livro sobre
- livro a
- está sobre
- está a
- está mesa
- sobre a
- sobre mesa
- a mesa

A biblioteca scikit-learn não possui uma implementação para skip-grams, portanto foi utilizado uma implementação disponibilizada no StackOverflow[26]. Neste trabalho utilizamos o conjunto de unigrams mais 2-skip-bigrams e suas frequências, conforme código abaixo:

```
1 SkipGramVectorizer(max_features=2500, min_df=8, max_df=0.8, k=2, ngram_range=(1, 2))
```

A tabela 8 mostra os 5 2-skip-bigrams mais frequentes na base de dados:

Tabela 8: 2-skip-bigrams mais frequentes

Termo	Frequência
muit bom	13.147
nao nao	10.333
entr cont	7.889
produt muit	6.797
nao produt	5.918

Fonte: Autor

5.2.5. 2-Skip-Trigram

Similarmente aos trigrams e aos 2-skip-bigrams, utilizamos o conjunto de unigrams mais 2-skip-bigrams mais 2-skip-trigrams e suas frequências, conforme código abaixo:

```
1 SkipGramVectorizer(max_features=2500, min_df=8, max_df=0.8, k=2, ngram_range=(1, 3))
```

A tabela 9 mostra os 5 2-skip-trigrams mais frequentes na base de dados:

Tabela 9: 2-skip-trigrams mais frequentes

Termo	Frequência
produt muit bom	3.269
muit bom produt	1.886
otim cust benefici	1.803
nao entr conat	1.729
muit bom muit	1.703

Fonte: Autor

5.2.6. TF-IDF Unigram

O modelo TF-IDF tenta expressar a “importância” de uma palavra dentro de uma coleção. Para isso o modelo se utiliza das seguintes equações:

$$idf\left(t\right)=\log\left(\frac{N}{n_t}\right)$$

$$w\left(d\,,t\right)=tf\left(d\,,t\right)\times idf\left(t\right)$$

Sendo:

- *t* = palavra
- *N* = número total de documentos na coleção
- n_t = número de documentos onde a palavra *t* ocorreu
- *tf(d,t)* = frequência da palavra *t* no documento *d*
- *idf(t)* = importância de *t* na coleção

Utilizamos a seguinte configuração para TF-IDF Unigram:

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2 TfidfVectorizer(max_features=2500, min_df=8, max_df=0.8)
```

A tabela 10 mostra os 5 TF-IDF unigrams mais frequentes na base de dados:

Tabela 10: TD-IDF unigrams mais frequentes

Termo	Frequência
nao	4.276,73
muit	3.351,40
produt	3.215,17
bom	2.306,08
compr	2.195,03

Fonte: Autor

5.2.7. TF-IDF Bigram

Similarmente utilizamos no TF-IDF o conjunto de unigrams mais bigrams, conforme código abaixo:

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2 TfidfVectorizer(max_features=2500, min_df=8, max_df=0.8, ngram_range=(1, 2))
```

A tabela 11 mostra os 5 TF-IDF bigrams mais frequentes na base de dados:

Tabela 11: TD-IDF bigrams mais frequentes

Termo	Frequência
muit bom	1.255,22
cust benefici	657,49
gost muit	621,30
otim produt	573,27
produt muit	542,36

Fonte: Autor

5.2.8. TF-IDF Trigram

Similarmente utilizamos no TF-IDF o conjunto de unigrams mais bigrams mais trigrams, conforme código abaixo:

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2 TfidfVectorizer(max_features=2500, min_df=8, max_df=0.8, ngram_range=(1, 3))
```

A tabela 12 mostra os 5 TF-IDF trigrams mais frequentes na base de dados:

Tabela 12: TD-IDF trigrams mais frequentes

Termo	Frequência
produt muit bom	389,29
otim cust benefici	256,28
entreg sup rap	252,33
gost muit produt	222,30
muit bom produt	199,94

Fonte: Autor

5.3. Classificadores

Foram selecionados somente os classificadores que suportavam matrizes esparsas (scipy.sparse), essa representação necessita de uma quantidade menor de memória e em geral os classificadores operam mais rápido com esse tipo de dado.

Cada classificador será brevemente descrito juntamente com sua respectiva configuração nas próximas sessões.

5.3.1. KerasClassifier

O KerasClassifier faz parte da biblioteca TensorFlow e utiliza técnicas de aprendizagem produnda – ou deep learning – para modelar abstrações de alto nível de dados usando um grafo com várias camadas de processamento e trransformações lineares e não lineares.

```
1 from tf.keras.wrappers.scikit_learn import KerasClassifier
2
3 def keras_model(optimizer="Adamax", activation="softplus", units=32):
4     model = Sequential()
5     model.add(Dense(units, activation="relu", input_dim=2500))
6     model.add(Dense(2, activation=activation))
7     model.compile(loss="categorical_crossentropy", optimizer=optimizer,
```

```
    metrics=["accuracy"])
    return model

KerasClassifier(build_fn=keras_model, epochs=5, batch_size=10)
```

5.3.2. XGBClassifier

O XGBClassifier faz parte da biblioteca XGBoost e implementa o algoritimo de Gradient Boosting, que utiliza modelo de predição Decision Tree divididos em estágios de tal forma a otimizar uma função de erro.

```
from xgboost import XGBClassifier
XGBClassifier(base_score=0.5)
```

5.3.3. XGBRFClassifier

O XGBRFClassifier também faz parte da biblioteca XGBoost e implementa o algoritimo de Gradient Boosting utilizando como sub-modelo o Random Forest Classification.

```
from xgboost import XGBRFClassifier
XGBRFClassifier(base_score=0.5)
```

5.3.4. LGBMClassifier

O LGBMClassifier faz parte da biblioteca LightGBM e implementa o algoritimo de Gradient Boosting utilizando o modelo de predição Decision Tree, similar ao XGBClassifier.

```
from lightgbm import LGBMClassifier
LGBMClassifier()
```

5.3.5. CatBoostClassifier

O CatBoostClassifier faz parte da biblioteca CatBoost e implementa o algoritimo de Gradient Boosting utilizando o modelo de predição Decision Tree, similar ao XGBClassifier.

```
from catboost import CatBoostClassifier
CatBoostClassifier()
```

5.3.6. LogisticRegression

O LogisticRegression faz parte da bibliotaca scikit-learn, assim como todos os classificadores que serão apresentados posteriormente. Ele implementa um modelo linear utilizando a função logística:

$$f(x) = \frac{L}{1 + \mathrm{e}^{-k(x-x_0)}}$$

O código abaixo demonstra a configuração utilizada:

```
1 from sklearn.linear_model import LogisticRegression
2 LogisticRegression(max_iter=2000)
```

5.3.7. SGDClassifier

O SGDClassifier utiliza o Support Vector Machine (SVM) como modelo de predição, calculando o gradiente da perda em cada etapa do algorítmo. O código abaixo demonstra a configuração utilizada:

```
1 from sklearn.linear_model import SGDClassifier
2 SGDClassifier(max_iter=2000)
```

5.3.8. RidgeClassifier

O RidgeClassifier se assemelha ao método dos mínimos quadrados, porém adicionando uma penalidade no tamanho dos coeficientes. Os coeficientes de ridge minizam a soma dos minimos quadados com penalidade:

$$\min_{w} \| Xw - y \|_2^2 + \alpha \| w \|_2^2$$

O código abaixo demonstra a configuração utilizada:

```
1 from sklearn.linear_model import RidgeClassifier
2 RidgeClassifier()
```

5.3.9. PassiveAggressiveClassifier

Classificador linear que não necessita de uma taxa de aprendizado, mas inclui penalidades e só atualiza o modelo no caso de erros. Tem boa performance para grande quantidade de dados. O código abaixo demonstra a configuração utilizada:

```
1 from sklearn.linear_model import PassiveAggressiveClassifier
2 PassiveAggressiveClassifier(max_iter=2000)
```

5.3.10. Perceptron

Classificador semelhante ao PassiveAggressiveClassifier, porém não inclui penalidades no modelo. O código abaixo demonstra a configuração utilizada:

```
1 from sklearn.linear_model import Perceptron
2 Perceptron(max_iter=2000)
```

5.3.11. DecisionTreeClassifier

O DecisionTreeClassifier é um classificador baseado numa árvore de decisões. Apesar de criar modelos de fácil entendimento, está sujeito a overfitting. O código abaixo demonstra a configuração utilizada:

```
1 from sklearn.tree import DecisionTreeClassifier
2 DecisionTreeClassifier()
```

5.3.12. ExtraTreeClassifier

O ExtraTreeClassifier também é um classificador baseado numa árvore de decisões, com a diferença que as amostras são separadas de maneira aleatória na montagem da árvore. O código abaixo demonstra a configuração utilizada:

```
1 from sklearn.tree import ExtraTreeClassifier
2 ExtraTreeClassifier()
```

5.3.13. MultinomialNB

O classificador MultinomialNB implementa uma variação do algorítmo de Naive Bayes, que segue o seguinte teorema de relacionamento de uma classe *y* e vetores de atributos dependentes x_1 até x_n:

$$P\left(y \mid x_1, \ldots, x_n\right) = \frac{P\left(y\right) P\left(x_1, \ldots, x_n \mid y\right)}{P\left(x_1, \ldots, x_n\right)}$$

O MultinomialNB adiciona um parametro de suavização baseado na frequência relativa dos atributos. O código abaixo demonstra a configuração utilizada:

```
1 from sklearn.naive_bayes import MultinomialNB
2 MultinomialNB()
```

5.3.14. BernoulliNB

Outra variação do algorítmo de Naive Bayes que penaliza a não ocorrencia de um atributo em uma determinada classe. O código abaixo demonstra a configuração utilizada:

```
1 from sklearn.naive_bayes import BernoulliNB
2 BernoulliNB()
```

5.3.15. RandomForestClassifier

Classificador baseado em árvores de decisão com aleatoriadade na seleção de atributos para a construção da árvore de decisão. O código abaixo demonstra a configuração utilizada:

```
1 from sklearn.ensemble import RandomForestClassifier
2 RandomForestClassifier()
```

5.3.16. ExtraTreesClassifier

Classificador similar ao RandomForestClassifier, porém adicionando um nível de aleatoriedade na divisão das amostras para construção da árvore de decisão. O código abaixo demonstra a configuração utilizada:

```
1 from sklearn.ensemble import ExtraTreesClassifier
2 ExtraTreesClassifier()
```

5.3.17. GradientBoostingClassifier

Implementação do algorítimo de Gradient Boosting da biblioteca scikit-learn utilizando o modelo de Decision Tree. O código abaixo demonstra a configuração utilizada:

```
1 from sklearn.ensemble import GradientBoostingClassifier
2 GradientBoostingClassifier(n_estimators=100)
```

5.3.18. BaggingClassifier

O classificador BaggingClassifier utiliza modelos de predição pré-definidos, gerando a predição de um sub-conjunto de dados e agregando o resultado pela média ou pela maioria das predições. O código abaixo demonstra a configuração utilizada:

```
1 from sklearn.ensemble import BaggingClassifier
2 BaggingClassifier()
```

5.3.19. AdaBoostClassifier

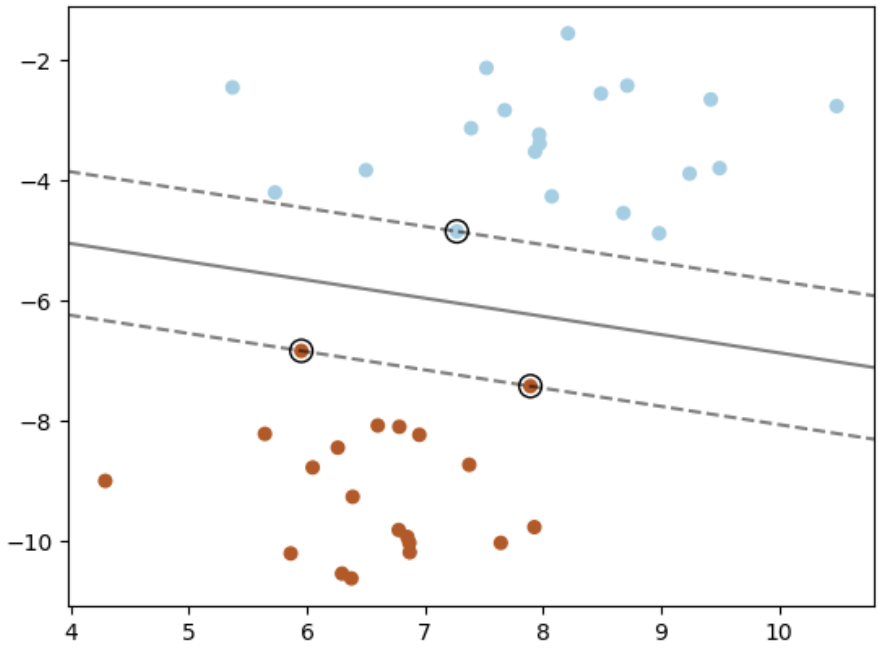
O classificador AdaBoostClassifier também utiliza modelos de predição pré-definidos, calibrando-os baseados num conjunto de dados modificados com pesos maiores nas predições erradas, fazendo assim com que os modelos de predição foquem em resolver os casos mais difíceis. O código abaixo demonstra a configuração utilizada:

```
1 from sklearn.ensemble import AdaBoostClassifier
2 AdaBoostClassifier()
```

5.3.20. SVC

O classificador SVC (Support Vector Machines) constrói um hiperplano ou um conjunto de hiperplanos que são utilizados na predição. A figura 8 mostra um exemplo de dois conjuntos de dados separados pelo hiperplano, aonde 3 pontos estão na fronteira dos “vetores de suporte”.

Figura 8: Exemplo de classificação usando SVC



Fonte: scikit-learn

Este classificador suporta diversos *kernels*, que são as funções que mapeiam os dados no hiperplano. O código abaixo demonstra a configuração utilizada:

```
1 from sklearn.svm import SVC
2 SVC(max_iter=2000)
```

5.3.21. LinearSVC

Similar ao classificador SVC, porém utilizando um *kernel* linear. O código abaixo demonstra a configuração utilizada:

```
1 from sklearn.svm import LinearSVC
2 LinearSVC(max_iter=2000)
```

5.3.22. MLPClassifier

O classificador MLPClassifier utiliza uma rede neural multi-camadas do modelo Perceptron para fazer as predições. O código abaixo demonstra a configuração utilizada:

```
1 from sklearn.neural_network import MLPClassifier
2 LinearSVC(max_iter=20)
```

5.4. Otimização de Hiperparâmetros

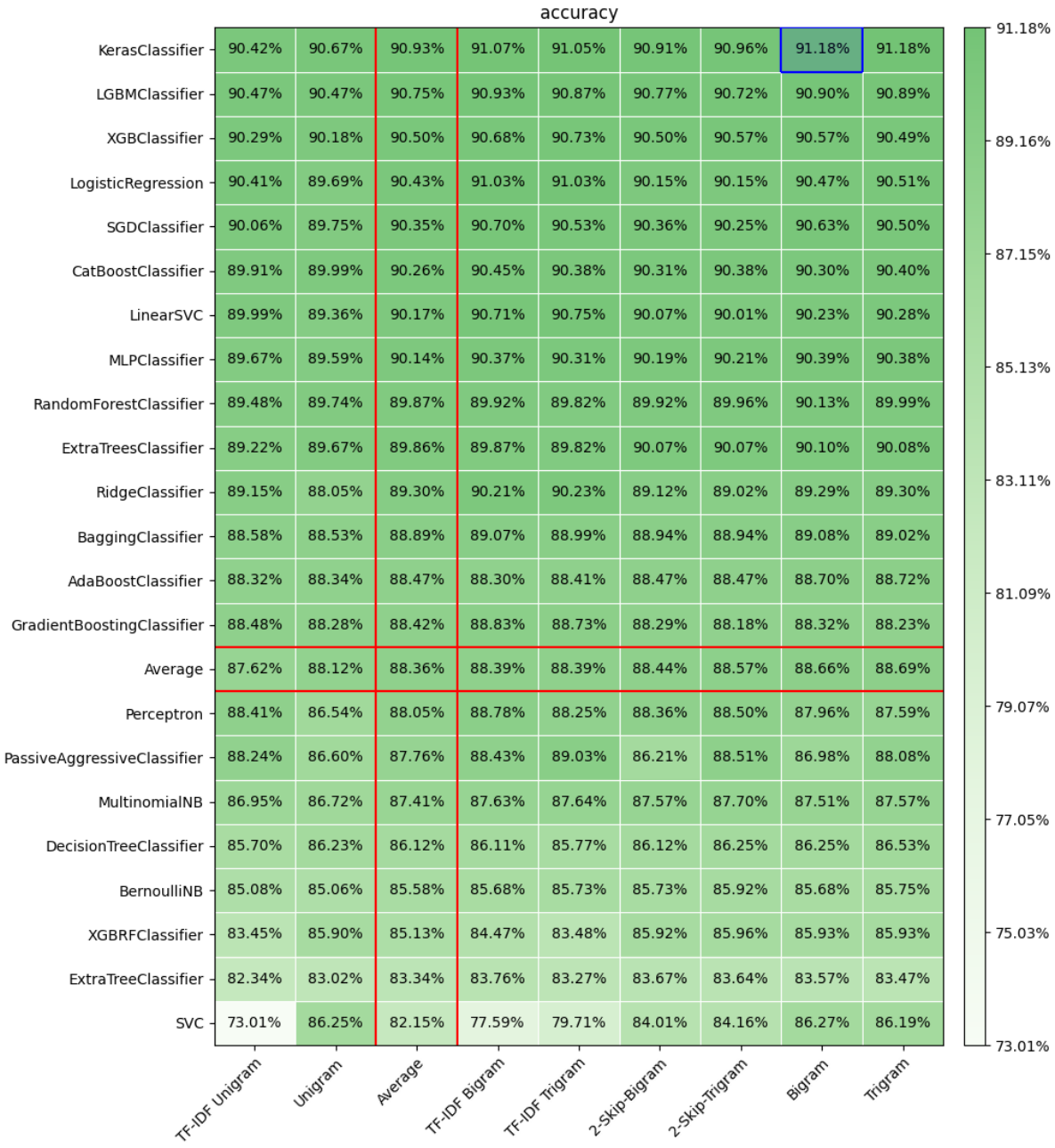
Primeiramente os dados foram vetorizados utilizando os vetorizadores descritos acima e então a base foi separada de tal forma a utilizar 50% para o trainamento e 50% para o teste dos classificadores. Posteriormente foram removidos os *outliers* dos dados de treino utilizando o LocalOutlierFactor e os dados foram escalados utilizando o RobustScaler. Finalmente rodamos o classificador selecionado e calculamos a acurácia da predição. Abaixo está um exemplo dessas operações para o vetorizador Unigram e o classificador LogisticRegression:

```
 1 from sklearn.feature_extraction.text import CountVectorizer
 2 from sklearn.linear_model import LogisticRegression
 3 from sklearn.metrics import accuracy_score
 4 from sklearn.model_selection import train_test_split
 5 from sklearn.neighbors import LocalOutlierFactor
 6 from sklearn.preprocessing import LabelEncoder, RobustScaler
 7
 8 texts = df["text"].values
 9 sentiments = LabelEncoder().fit_transform(df["sentiment"].values)
10
11 x_train, x_test, y_train, y_test = train_test_split(
12     texts, sentiments, train_size=train_size)
13
14 outlier_detection = LocalOutlierFactor(n_neighbors=5, contamination=.01)
15 outliers = outlier_detection.fit_predict(x_train)
```

```
16
17 x_train = x_train[outliers > 0]
18 y_train = y_train[outliers > 0]
19
20 scaler = RobustScaler(with_centering=False).fit(x_train)
21 x_train = scaler.transform(x_train)
22 x_test = scaler.transform(x_test)
23
24 vectorizer = CountVectorizer(max_features=2500, min_df=8, max_df=0.8).fit(x_train)
25
26 x_train = vectorizer.transform(x_train)
27 x_test = vectorizer.transform(x_test)
28
29 classifier = LogisticRegression(max_iter=2000).fit(x_train, y_train)
30 y_predict = classifier.predict(x_test)
31
32 accuracy = accuracy_score(y_test, y_predict)
```

O mesmo procedimento foi executado para a combinação de todos os vetorizadores e classificadores descritos nas sessões 5.2 e 5.3. O resultado é exibido na figura 9 como uma tabela ordenada pela acurácia da esqueda para direita e de baixo para cima. Foi incluido a média das acurácia por vetorizador e classificador em destaque vermelho e a melhor acurácia está em destaque azul.

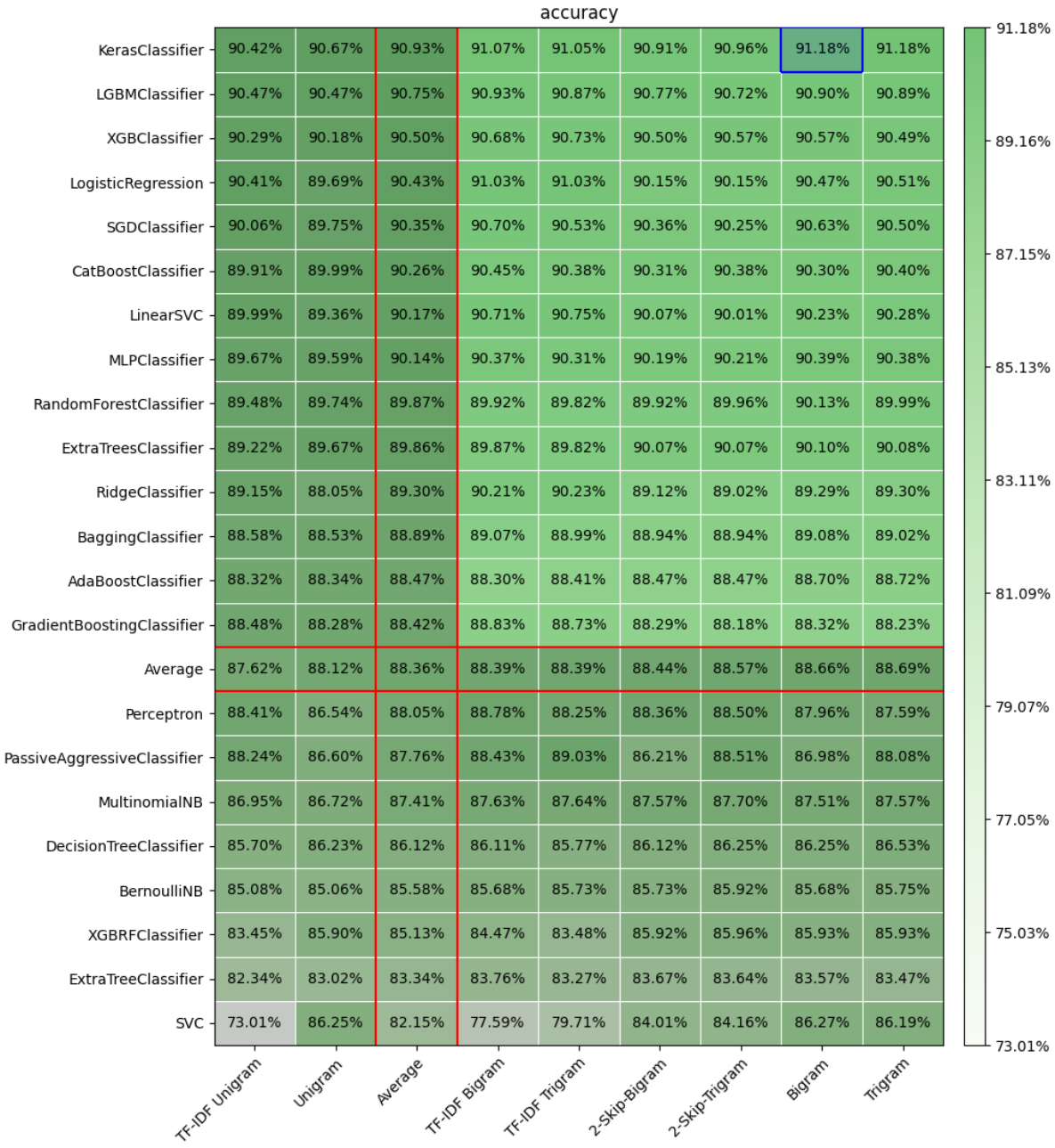
Figura 9: Matriz de acurácia



Fonte: Autor

O resultado acima foi obtido com as configurações definidas anteriormentes, assim para conseguir um resultado melhor devemos fazer a otimização dos hiperparâmetros. Por ser uma operação muito demorada foram excluidos os vetorizadores e classificadores que estavam abaixo da média de acurácia. A figura 10 mostra os vetorizadores e classificadores excluídos com uma sombra escura.

Figura 10: Matriz com os vetorizadores e classificadores excluídos



Fonte: Autor

Para os classificadores que não foram excluídos, os hiperparâmetros apresentados na tabela 13 foram testados e a melhor combinação foi selectionada utilizando o componente GridSearchCV.

Tabela 13: Hiperparâmetros para cada classificador

Classificador	Hiperparâmetros
KerasClassifier	`epochs=[5, 10, 15]` `batch_size=[3, 5, 10]`

LGBMClassifier	`n_estimators=[100, 200]`
	`learning_rate=[0.1, 0.5, 1.0]`
	`max_depth=[5, 6, 7, -1]`

XGBClassifier	`n_estimators=[100, 200]`
	`gamma=[1, 0.1, 0.001, 0.0001]`
	`max_depth=[5, 10]`

LogisticRegression	`solver=[`
	`"lbfgs", "newton-cg", "liblinear",`
	`"sag", "saga"`
	`]`
	`C=[0.05, 0.5, 1.0]`
	`penalty=["l1", "l2"]`

SGDClassifier	`loss=[`
	`"hinge", "log", "modified_huber",`
	`"squared_hinge", "perceptron"`
	`]`
	`learning_rate=["optimal", "invscaling", "adaptive"]`
	`eta0=[0.01, 0.05, 0.1, 0.25, 0.75, 1]`

CatBoostClassifier	`n_estimators=[100, 200]`
	`learning_rate=[0.03, 0.5, 1.0]`
	`max_depth=[5, 6, 7]`

LinearSVC	`penalty=["l1", "l2"]`
	`loss=["hinge", "squared_hinge"]`
	`C=[1, 10, 100, 1000]`

MLPClassifier	`activation=["logistic", "tanh", "relu"]`
	`solver=["lbfgs", "adam"]`
	`learning_rate=["constant", "invscaling"]`
	`alpha=[0.0001, 0.01, 1.0]`

RandomForestClassifier	`n_estimators=[100, 200]`
	`criterion=["entropy", "gini"]`
	`min_samples_leaf=[1, 2, 5, 7]`
	`max_depth=[5, 6, 7, None]`

ExtraTreesClassifier	`n_estimators=[100, 200]`
	`criterion=["entropy", "gini"]`
	`min_samples_leaf=[1, 2, 5, 7]`
	`max_depth=[5, 6, 7, None]`

RidgeClassifier	`alpha=[0, 0.001, 0.01, 0.25, 0.5, 0.75, 1.0]`

BaggingClassifier	`n_estimators=[10, 20],`
	`base_estimator=[`
	`None,`

	LogisticRegression(max_iter=2000), SGDClassifier(max_iter=2000),],
AdaBoostClassifier	n_estimators=[10, 50, 100] algorithm=["SAMME", "SAMME.R"] learning_rate=[0.5, 1.0] base_estimator=[None, LogisticRegression(max_iter=2000), SGDClassifier(max_iter=2000),]
GradientBoostingClassifier	loss=["deviance", "exponential"] criterion=["friedman_mse", "mse"] min_samples_leaf=[1, 3, 7] max_depth=[3, 5, 7]

Fonte: Autor

Abaixo segue um exemplo de código do GridSearchCV para o LogisticRegression:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

classifier = GridSearchCV(
    LogisticRegression(max_iter=2000),
    dict(
        loss=["hinge", "log", "modified_huber", "squared_hinge", "perceptron"],
        learning_rate=["optimal", "invscaling", "adaptive"],
        eta0=[0.01, 0.05, 0.1, 0.25, 0.75, 1],
    )
)

y_predict = classifier.fit(x_train, y_train).predict(x_test)
accuracy = accuracy_score(y_test, y_predict)

print(classifier.best_params_)
print(classifier.best_estimator_)
```

Após a otimização dos hiperparâmetros foi utilizando o VotingClassifier com todas as combinações possíveis de classificadores de tal forma a achar a combinação com melhor acurácia. Abaixo segue o código com a utilização do VotingClassifier:

```
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression, SGDClassifier, RidgeClassifier
from sklearn.metrics import accuracy_score

classifier = VotingClassifier([
    ("logistic", LogisticRegression(max_iter=2000)),
    ("sgd", SGDClassifier(max_iter=2000)),
    ("ridge", RidgeClassifier(max_iter=2000)),
])

```

```
y_predict = classifier.fit(x_train, y_train).predict(x_test)
accuracy = accuracy_score(y_test, y_predict)
```

5.5. Modelo Final

A figura 11 mostra o resultado da otimização dos hiperparâmetros e do VotingClassifier, sendo a melhor acurácia destacada em azul. As células com asterísco são as que foram selecionadas como melhor combinação no VotingClassifier.

Figura 11: Matriz com a otimização dos hiperparâmetros e o VotingClassifier

Fonte: Autor

O melhor resultado ficou com o VotingClassifier rodando com o vetorizador de Trigram e com os classificadores LGBMClassifier, KerasClassifier, XGBClassifier, MLPClassifier, e LinearSVC. A acurácia do modelo foi de 91,42%, uma melhora de 0,25% comparado ao melhor modelo na etapa anterior, o KerasClassifier com Bigrams, com 91.18% de acurácia.

Os hiperparâmetros otimizados nos classificadores selecionados para o modelo final são descritos na tabela 14.

Tabela 14: Hiperparâmetros otimizados

Classificador	Hiperparâmetros
KerasClassifier	`epochs=5` `batch_size=10`

LGBMClassifier	`n_estimators=200`
	`learning_rate=0.1`
	`max_depth=-1`

XGBClassifier	`n_estimators=200`
	`gamma=0.1`
	`max_depth=10`

MLPClassifier	`activation="relu"`
	`solver="adam"`
	`learning_rate="constant"`
	`alpha=1.0`

LinearSVC	`penalty="l2"`
	`loss="hinge"`
	`C=1`

Fonte: Autor

6. Apresentação dos Resultados

O modelo final tem uma acurâcia de 91,42% para classificar a base de dados em sentimentos positivos e negativos. Nas sessões seguintes iremos analisar algumas características do modelo para termos certeza que o mesmo não apresenta nenhuma deficiência evidente.

6.1. Termos com mais Influência

A biblioteca scikit-learn nos permite identificar quais são os termos que mais influência nas predições. A tabela 15 mostra os termos mais influentes para as classificações positivas e negativas.

Tabela 15: Termos com mais Influência

Positivo	Negativo
efici	nao recom
gost bast	tot
praz cert	bom sup
excel atend	pess
delici	lix

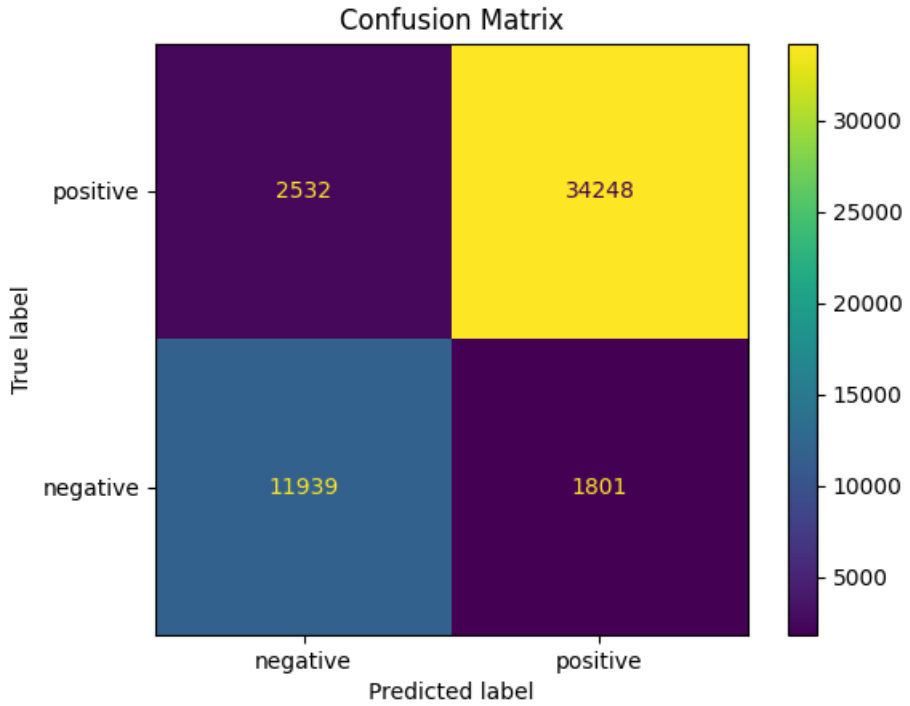
Fonte: Autor

6.2. Matriz de Confusão

Analisando a matrix de confusão da figura 12 podemos observar que o número de falsos negativos é de 2.532 e de falsos positivos é de 1.801. O número maior de falsos negativos está ligado ao fato de haver mais avaliações positivas na base de dados. Através da matriz de confusão também podemos calcular a acurácia do modelo:

$$\frac{(34.248 + 11.939)}{(2.532 + 34.248 + 11.939 + 1.801)} = 91{,}42\ \%$$

Figura 12: Matriz de Confusão



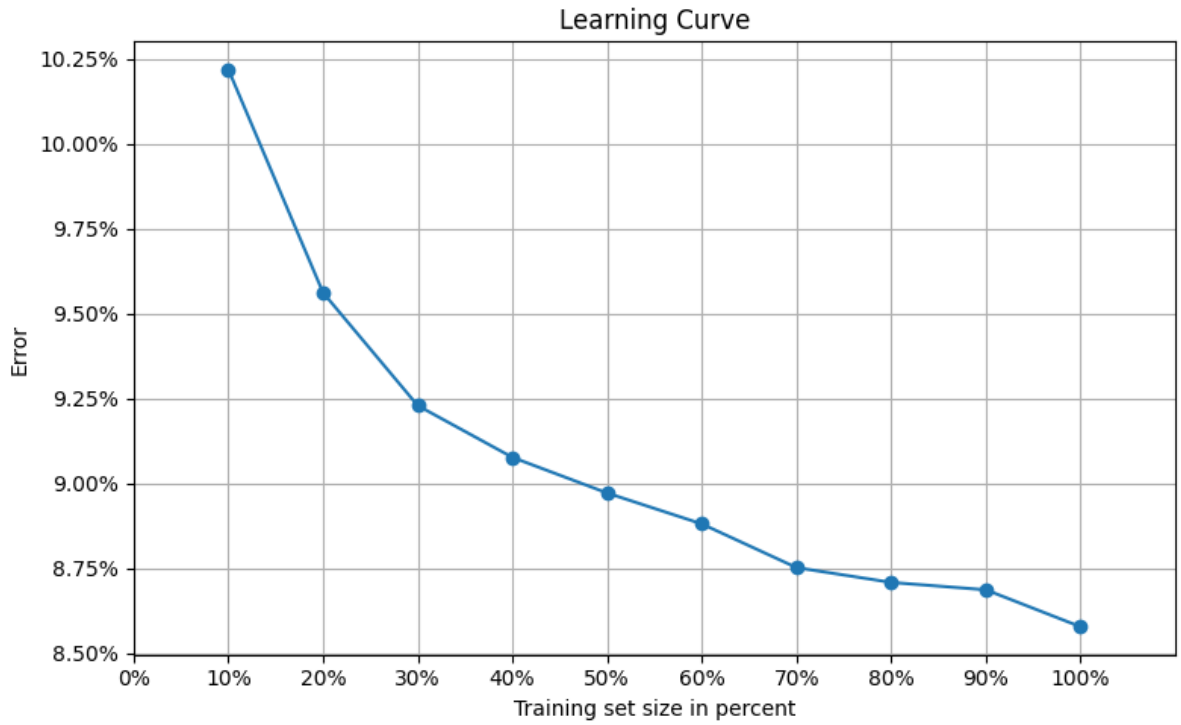
Fonte: Autor

6.3. Curva de Aprendizado

A curva de aprendizado mostra a perda ou erro conforme aumentamos as amostras de treinamento. Dessa forma podemos identificar se há *overfitting* checando se a curva tem inclinação positiva para aumentos no tamanho da amostra.

Podemos identificar na figura 13 que não ocorreu *overfitting*, já que a curva tem inclinação negativa em todos os pontos, sempre diminuindo o erro para um aumento no tamanho da amostra de treino.

Figura 13: Curva de Aprendizado



Fonte: Autor

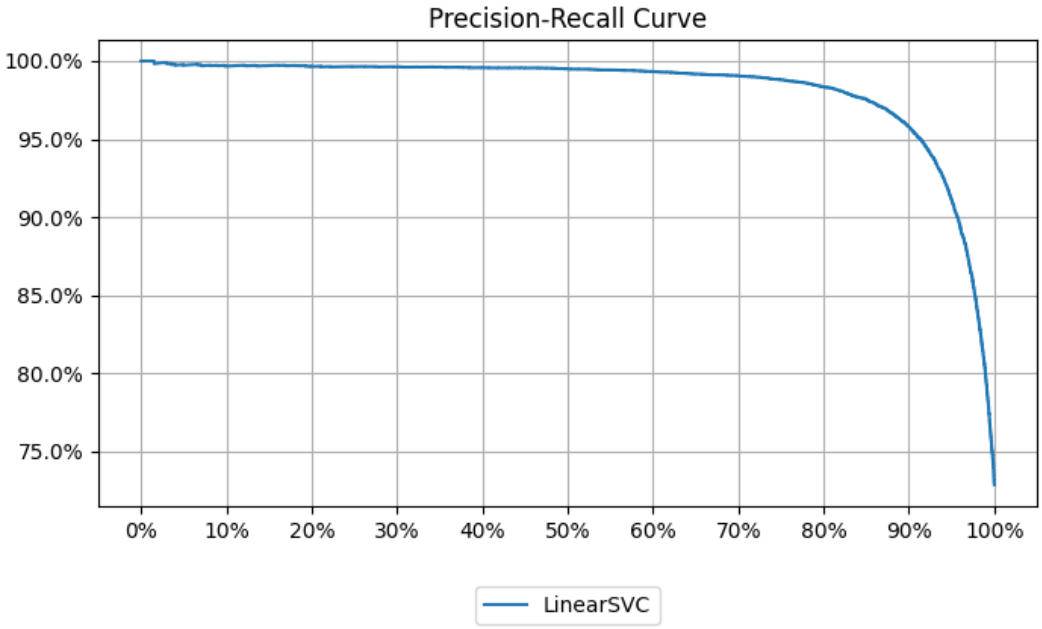
6.4. Curva de Precisão e Revocação

A precisão – ou *precision* – de um classificador pode ser definida como o quanto os resultados da pesquisa são úteis, enquanto a revocação – ou *recall* – é o quão completos os resultados estão.

A curva ideal de Precisão e Revocação seria uma reta horizontal em 100% de precisão e uma reta vertical em 100% de Revocação.

Na figura 14 vemos uma curvatura na precisão a partir de 50%, sendo mais acentuada a partir de 80%, porém ainda conseguimos observar que a curva fica em sua maior parte próximo de 100% de precisão e 100% de revocação, evidenciando um boa qualidade do modelo.

Figura 14: Curva de Precisão e Revocação



Fonte: Autor

Uma observação a ser feita é que para calcular a curva de Precisão e Revocação pela biblioteca scikit-learn o classificador precisa implementar o método `decision_function`, portanto a curva só foi calculada para os classificadores que implementavam este método.

6.5. Curva de Característica de Operação do Receptor

A curva de Característica de Operação do Receptor – ou *Receiver Operating Characteristic curve, ROC* – mede os verdadeiros positivos em comparação com os falsos positivos.

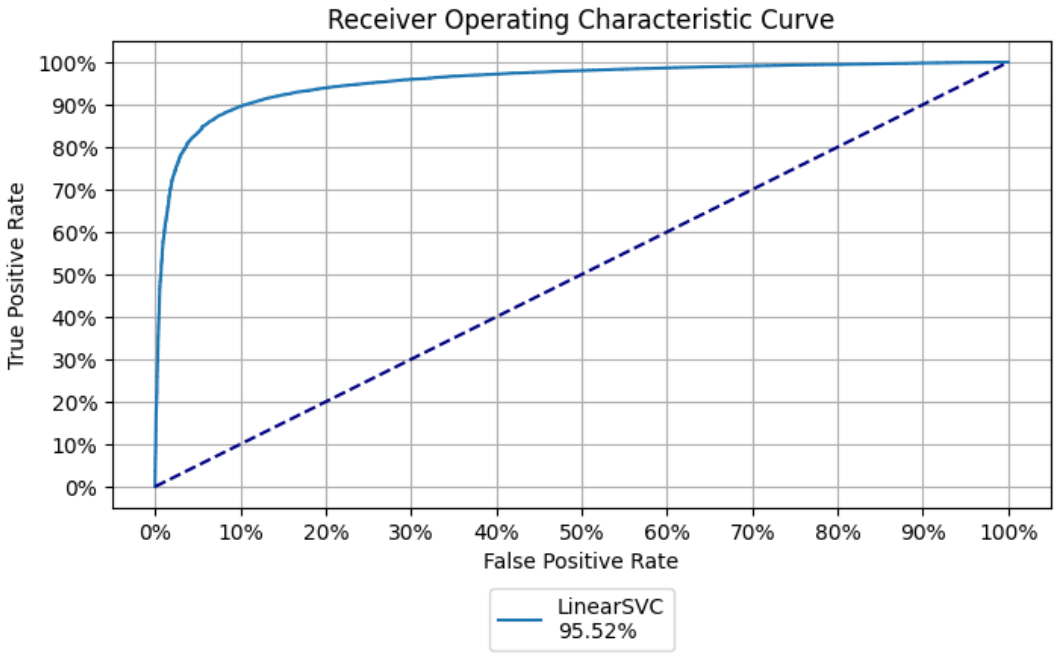
A curva ideal seria uma reta horizontal em 100% nos verdadeiros positivos e uma reta vertical em 0% nos falsos positivos.

Similarmente à curva de Precisão e Revocação, vemos na figura 15 uma curvatura nos verdadeiros positivos a partir de 50%, sendo mais acentuada a partir de 70%, porém a curva fica em sua maior parte próximo de 100% de verdadeiros positivos e 0% de falsos positivos.

Uma outra métrica associada a curva de Característica de Operação do Receptor é calcular a área abaixo da curva, sendo que a área ideal seria de 100% e vemos que o classificador tem área de 95,52%.

Ambas métricas demostram que a qualidade do modelo é boa e não há nenhum defeito evidente.

Figura 15: Curva de Característica de Operação do Receptor



Fonte: Autor

Para calcularmos esta curva os classificadores também precisavam implementar o método `decision_function`, portanto a curva só foi calculada para os classificadores que implementavam este método.

6.6. Comparação com Outros Modelos

Além das análises do modelo apresentadas nas sessões anteriores também foram feitas comparações com 2 modelos determinísticos para checar se o modelo apresentado é ou não superior.

O primeiro modelo de comparação é um modelo muito simples incluso na biblioteca scikit-learn chamado DummyClassifier, que simplesmente pega o resultado mais frequente na base de treino e sempre retorna esse resultado como predição. Como nossa base tem mais avaliações positivas este modelo sempre predizia que qualquer amostra era positiva.

A acurácia do classificador DummyClassifier foi de 72,80%, muito abaixo dos 91,42% do modelo descrito neste trabalho.

O segundo modelo utilizado como comparação foi o LeIA (Léxico para Inferência Adaptada) proposto por ALMEIDA, Rafael (2018)[27] que é uma adaptação para o português da ferramenta VADER (Valence Aware Dictionary and sEntiment Reasoner) de Hutto, C.J. & Gilbert, E.E. (2014)[28] que é um analisador de sentimentos em inglês para redes sociais.

O modelo LeIA teve acurácia de 84,43%, ainda abaixo do modelo proposto neste trabalho.

Uma última comparação foi feita com relação a classificação manual do autor, aonde apenas uma frase teve predição errada. Essa comparação está na tabela 16.

Tabela 16: Comparação com classificação manual

Frase	Predição	Real
produto muito bom, adorei! recomendo a todos!	positivo	positivo
o produto é ótimo, vale a pena comprar!	positivo	positivo
muito bom, superou minhas expectativas! compraria novamente	positivo	positivo
a entrega demorou muito e o produto chegou quebrado. muito ruim	negativo	negativo
terrível, não funciona, tentei de tudo mas não liga	negativo	negativo
acabamento ruim, muito frágil. não vale a pena.	negativo	negativo
o produto não é bom como anunciado	**positivo**	**negativo**

Fonte: Autor

6.7. Considerações Finais

Podemos concluir que é possível criar um modelo de aprendizado de máquina com boa acurácia utilizando as avaliações de produtos e serviços disponíveis na internet.

Segundo a plataforma Lexalytics[29], especializada em análise de sentimento, os pesquisadores concordam que uma boa acurácia para uma classificador de sentimentos (positivo, negativo e neutro) deve variar entre 80 e 85%.

Por questões de simplificação, o modelo proposto nesse trabalho classifica somente sentimentos positivos e negativos. Uma sugestão para trabalho posteriores seria estender os possíveis sentimentos, incluindo o neutro ou aumentando ainda

mais a granularidade dos sentimentos, como a inclusão de “raiva”, “triste”, “alegre” e outros sentimentos.

Outro desafio é melhorar a identificação de negações, ironias e sarcarmos de maneira genérica e evitando *overfitting*. Esse é um problema muito comum em algorítmos de análise de sentimentos e dado sua complexidade, não foi abordado neste trabalho.

Apesar do modelo proposto ser simples, a acurácia obtida comprova que o mesmo pode ser utilizado por empresas ou qualquer pessoa que queira monitorar a reação dos usuários no ambiente da internet.

7. Links

Os códigos demostrados nesse trabalho são simplificação do código que foi utilizado para rodar os modelos. O código completo pode ser encontrado no link abaixo juntamente com um vídeo com uma breve apresentação sobre este trabalho.

Link para o vídeo: https://youtu.be/6mXQeqDCoJE
Link para o repositório: https://github.com/ahnunes/tcc-puc

REFERÊNCIAS

[1] DIGITAL 2020: JULY GLOBAL STATSHOT. Disponível em: <https://datareportal.com/reports/digital-2020-july-global-statshot>. Acesso em 31/07/2020.

[2] TIC DOMICÍLIOS – 2019. Disponível em: <https://cetic.br/pt/tics/domicilios/2019/individuos/>. Acesso em 31/07/2020.
[3] O POST É PAGO, E AÍ?. Disponível em: <https://www.institutoqualibest.com/download/influenciadores-digitais-o-post-e-pago-e-ai/>. Acesso em 31/07/2020.
[4] SOCIAL MEDIA TRENDS 2019. Disponível em: <https://materiais.rockcontent.com/social-media-trends>. Acesso em 31/07/2020.
[5] DATA AGE 2025 – THE DIGITALIZATION OF THE WORLD. Disponível em: <https://www.seagate.com/br/pt/our-story/data-age-2025/>. Acesso em 31/07/2020.
[6] 20 CUSTOMER EXPERIENCE STATISTICS YOU NEED TO KNOW. Disponível em: <https://www.getresponse.com/blog/20-customer-experience-statistics-need-know>. Acesso em 31/07/2020.
[7] RANKING DE RECLAMAÇÕES - RECLAME AQUI. Disponível em: <https://www.reclameaqui.com.br/ranking/>. Acesso em 06/07/2020.
[8] PYTHON. Disponível em: <https://www.python.org/>. Acesso em 31/07/2020.
[9] SCRAPY. Disponível em: <https://scrapy.org/>. Acesso em 31/07/2020.
[10] PANDAS - PYTHON DATA ANALYSIS LIBRARY. Disponível em: <https://pandas.pydata.org/>. Acesso em 31/07/2020.
[11] UNIDECODE · PYPI. Disponível em: <https://pypi.org/project/Unidecode/>. Acesso em 31/07/2020.
[12] RE - REGULAR EXPRESSION OPERATIONS - PYTHON 3.7.8 DOCUMENTATION. Disponível em: <https://docs.python.org/3.7/library/re.html>. Acesso em 31/07/2020.
[13] SPACY · INDUSTRIAL-STRENGTH NATURAL LANGUAGE PROCESSING IN PYTHON. Disponível em: <https://spacy.io/>. Acesso em 31/07/2020.
[14] NATURAL LANGUAGE TOOLKIT — NLTK 3.5 DOCUMENTATION. Disponível em: <https://www.nltk.org/>. Acesso em 31/07/2020.
[15] MAMMOTHB/SYMSPELLPY: PYTHON PORT OF SYMSPELL. Disponível em: <https://github.com/mammothb/symspellpy>. Acesso em 31/07/2020.
[16] INDIX/WHATTHELANG: LIGHTNING FAST LANGUAGE PREDICTION. Disponível em: <https://github.com/indix/whatthelang>. Acesso em 31/07/2020.
[17] NORVIG, Peter. How to Write a Spelling Corrector, 2007. Disponível em: <https://norvig.com/spell-correct.html>. Acesso em 01/08/2020.
[18] GARBE, Wolf. wolfgarbe/SymSpell: SymSpell: 1 million times faster through Symmetric Delete spelling correction algorithm, 2019. Disponível em: <https://github.com/wolfgarbe/SymSpell>. Acesso em 01/08/2020.
[19] EXAMPLES FOR PORTUGUESE PROCESSING. Disponível em: <http://www.nltk.org/howto/portuguese_en.html>. Acesso em 01/08/2020.
[20] STOPWORDS EM PORTUGUÊS. Disponíveis em <https://gist.githubusercontent.com/alopes/5358189/raw/2107d809cca6b83ce3d8e04dbd9463283025284f/stopwords.txt> e <https://raw.githubusercontent.com/stopwords-iso/stopwords-pt/master/stopwords-pt.txt>. Acesso em 06/07/2020.

[21] SCIKIT-LEARN: MACHINE LEARNING IN PYTHON — SCIKIT-LEARN 0.23.1 DOCUMENTATION. Disponível em: <https://scikit-learn.org/stable/>. Acesso em 01/08/2020.

[22] TENSORFLOW. Disponível em: <https://www.tensorflow.org/>. Acesso em 01/08/2020.

[23] XGBOOST. Disponível em: <https://xgboost.ai/>. Acesso em 01/08/2020.

[24] WELCOME TO LIGHTGBM'S DOCUMENTATION! Disponível em <https://lightgbm.readthedocs.io/en/latest/>. Acesso em 01/08/2020.

[25] CATBOOST - OPEN-SOURCE GRADIENT BOOSTING LIBRARY. Disponível em <https://catboost.ai/>. Acesso em 01/08/2020.

[26] PYTHON - IMPLEMENTING SKIP GRAM WITH SCIKIT-LEARN? - STACK OVERFLOW. Disponível em <https://stackoverflow.com/questions/39725052/implementing-skip-gram-with-scikit-learn>.

Acesso em 01/08/2020.

[27] ALMEIDA, Rafael. LeIA - Léxico para Inferência Adaptada, 2018. Disponível em <https://github.-com/rafjaa/LeIA>. Acesso em 20/07/2020.

[28] HUTTO, C.J. & GILBERT, E.E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014. Disponível em <https://github.com/cjhutto/vaderSentiment>. Acesso em 20/07/2020.

[29] Sentiment Accuracy: Explaining the Baseline and How to Test It – Lexalytics. Disponível em <https://www.lexalytics.com/lexablog/sentiment-accuracy-baseline-testing>. Acesso em 20/07/2020.

APÊNDICE

Código para Termos com mais Influência

Código baseado em:
https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers

```
import numpy as np

feature_names = vectorizer.get_feature_names()
top = np.argsort(classifier.coef_[0])

print(", ".join(feature_names[j] for j in top[:10]))
print(", ".join(feature_names[j] for j in top[-10:]))
```

Código para Matriz de Confusão

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix

labels =["negative", "positive"]
plot_confusion_matrix(classifier, x, y, display_labels=labels, values_format="d")

plt.show()
```

Código para Curva de Aprendizado

Código baseado no mlxtend disponível em:
https://github.com/rasbt/mlxtend/blob/master/mlxtend/plotting/learning_curves.py

```
import matplotlib.pyplot as plt
import numpy as np

def misclf_err(y_predict, y):
    return (y_predict != y).sum() / float(len(y))

rng = [int(i) for i in np.linspace(0, x_train.shape[0], 11)][1:]
for r in rng:
    clf.fit(x_train[:r], y_train[:r])
    y_predict = clf.predict(x_test)

    error = misclf_err(y_test, y_predict)
    errors.append(error)

plt.plot(np.arange(10, 101, 10), errors, label="Error")
plt.show()
```

Código para Curva de Precisão e Revocação

```
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve
from sklearn.multiclass import OneVsRestClassifier

clf = OneVsRestClassifier(estimator)
y_score = clf.fit(x_train_t, y_train_t).decision_function(x_test)

precision, recall, _ = precision_recall_curve(y_test.ravel(), y_score.ravel())

plt.plot(recall, precision)
plt.show()
```

Código para Curva de Característica de Operação do Receptor

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve
from sklearn.multiclass import OneVsRestClassifier

clf = OneVsRestClassifier(estimator)
y_score = clf.fit(x_train_t, y_train_t).decision_function(x_test)

fpr, tpr, _ = roc_curve(y_test.ravel(), y_score.ravel())

plt.plot(fpr, tpr)
plt.show()
```