

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

**Gildo Santos Franco Couto**

***MACHINE LEARNING* NA INTERPRETAÇÃO E CLASSIFICAÇÃO  
DE DECISÕES JUDICIAIS DA JUSTIÇA FEDERAL**

Belo Horizonte

2020

**Gildo Santos Franco Couto**

***MACHINE LEARNING* NA INTERPRETAÇÃO E CLASSIFICAÇÃO  
DE DECISÕES JUDICIAIS DA JUSTIÇA FEDERAL**

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.....	4
1.1. Contextualização .....	5
1.2. O problema proposto .....	6
2. Coleta de Dados .....	7
3. Processamento/Tratamento de Dados .....	16
4. Análise e Exploração dos Dados .....	28
5. Criação de Modelos de <i>Machine Learning</i> .....	37
6. Apresentação dos Resultados .....	65
7. <i>Links</i> .....	68
REFERÊNCIAS.....	69

## 1. Introdução

Ao longo dos últimos anos, os órgãos públicos dos três poderes, em todas as esferas, vêm investindo massivamente em novas tecnologias com o objetivo de digitalizar os seus processos de trabalho, buscando mais celeridade e redução de custos, como o Projeto Sócrates, desenvolvido pela Assessoria de Inteligência Artificial do Superior Tribunal de Justiça (STJ); e o Projeto Victor, desenvolvido pelo Supremo Tribunal Federal (STF) em parceria com a Universidade de Brasília (UnB).

A crise financeira pela qual passa o país, aliada à situação fiscal dos entes federados, com alto nível de endividamento, praticamente inviabiliza a reposição dos servidores e empregados públicos que se aposentam, forçando a busca por soluções inovadoras que minimizem os impactos nos serviços públicos gerados pela redução na mão de obra disponível.

Impulsionados pela crise sanitária imposta pela pandemia da Covid-19, órgãos como a Receita Federal do Brasil reformularam os seus processos de trabalho e a sua estrutura organizacional, com o objetivo de automatizar procedimentos antes realizados manualmente e capacitar os servidores em novas tecnologias, como *ciência de dados* e *big data*, para fazer frente às alterações no cenário externo.

Para os servidores públicos, o cenário interno é extremamente desafiador, com a necessidade imperiosa de se reinventar, adquirir novos conhecimentos e habilidades compatíveis com o mundo virtualizado.

Os processos de trabalho estão sendo automatizados e digitalizados, a exemplo do Decreto nº 10.278, de 18 de março de 2020, que estabelece a técnica e os requisitos para a digitalização de documentos públicos ou privados, a fim de que produzam os mesmos efeitos legais dos originais. Muitos dos atuais procedimentos e atividades poderão ser realizados por meios digitais de forma muito mais célere e precisa. Será necessário repensar as atividades do dia a dia, os postos de trabalho que não serão mais necessários, as novas competências que serão necessárias, *hard skills* e *soft skills*, e mais que capacitação e desenvolvimento, a transformação do perfil dos servidores públicos.

Com a modernização acelerada, mergulhamos na indústria 4.0, chamada de a quarta revolução industrial, cujo desafio é integrar as novas tecnologias – *artificial intelligence* (AI, inteligência artificial), *Internet of things* (IOT, *Internet* das coisas),

*bots, big data, business intelligence* – aos processos tradicionais, transformando-os, modernizando-os, tornando-se imperiosa a utilização de tecnologias como *machine learning*, resultando em um enorme salto de qualidade nos serviços públicos prestados à sociedade.

### 1.1. Contextualização

No período de 16 de março a 19 de julho de 2020, segundo informações publicadas no *site* do Conselho da Justiça Federal (CJF), todas as instâncias e regiões da Justiça Federal aplicaram juntas 1.161.936 sentenças, 1.658.837 decisões, 2.583.009 despachos e 43.072.453 movimentações processuais<sup>1</sup>, sendo que, na maioria dos processos, a União é parte interessada. A Delegacia da Receita Federal de Uberlândia, em Minas Gerais, recebeu mais de 20 mil decisões judiciais em um único tema, 669 – Validade da contribuição a ser recolhida pelo empregador rural pessoa física sobre a receita bruta proveniente da comercialização de sua produção, nos termos do art. 1º da Lei nº 10.256/2001, referente ao Recurso Extraordinário (RE) nº 718.874, que tramitou no Supremo Tribunal Federal, tendo ocorrido mudanças de entendimento no julgamento da matéria ao longo do tempo. O RE nº 596.177/RS, a título de exemplo, declarou inconstitucional a mesma contribuição que foi novamente julgada constitucional no RE nº 718.874.

A partir do julgamento sob a sistemática da repercussão geral (RE nº 718.874), no sentido da constitucionalidade da contribuição instituída pela Lei nº 10.256/2001<sup>2</sup>, a Receita Federal iniciou a cobrança da contribuição previdenciária, sendo que uma única ação promovida em Minas Gerais alcançou o valor de R\$ 260 milhões de contribuição previdenciária devida<sup>3</sup>. Estima-se que o passivo tributário dessa causa chegue a R\$ 15 bilhões, com um número de centenas de milhares de devedores.

Depreende-se, portanto, que, devido ao número elevado de ações judiciais, às mudanças na jurisprudência, ao entendimento divergente sobre a matéria nos tribunais gerando decisões diferentes sobre o mesmo tema, somados à escassez de

---

<sup>1</sup> <https://www.cjf.jus.br/cjf/noticias/2020/07-julho/justica-federal-registra-mais-de-1-5-milhao-de-decisoes-em-regime-de-trabalho-remoto>

<sup>2</sup> [http://www.planalto.gov.br/ccivil\\_03/leis/LEIS\\_2001/L10256.htm](http://www.planalto.gov.br/ccivil_03/leis/LEIS_2001/L10256.htm)

<sup>3</sup> <https://receita.economia.gov.br/noticias/ascom/2018/agosto/receita-federal-cobra-r-260-milhoes-de-funrural-devido-por-produtores-rurais-de-minas-gerais>

mão de obra qualificada nos órgãos federais para análise pontual das decisões judiciais, tornou-se imperiosa a aplicação de modelos de inteligência artificial, *machine learning* e *natural language processing* (NLP) para análise automatizada das decisões judiciais.

## 1.2. O problema proposto

*“Um projeto de Data Science começa com uma necessidade ou ideia. Nesta etapa inicial, deve-se primeiro ter em mente o problema que se deseja resolver, e, em seguida, os objetivos devem ser definidos ...”* (Escovedo, Tatiana)<sup>4</sup>

No presente projeto, o problema selecionado é a classificação das ementas das decisões judiciais de 2º grau do Tribunal Regional Federal da 1ª Região (TRF1), relativas à constitucionalidade da contribuição instituída pela Lei nº 10.256/2001<sup>5</sup>, assunto 6040 – Funrural. Trata-se de verificar se é factível realizar a classificação de decisões judiciais utilizando-se modelos de *machine learning*.

Para melhor visão do problema e da solução optou-se por utilizar a técnica dos 5-Ws<sup>6</sup>, respondendo-se as perguntas a que se refere cada W.

(Why?) Por que esse problema é importante?

Estima-se que o passivo tributário do tema 669 julgado pelo STF chegue a R\$ 15 bilhões, com um número de centenas de milhares de devedores, sendo impossível a análise manual de todas as decisões que a Fazenda Nacional foi intimada, apesar do valor elevado das causas. A partir da análise de decisão, em se decidindo pela constitucionalidade da contribuição, iniciam-se os procedimentos de cobrança dos valores devidos, sendo importante, portanto, a celeridade na análise das decisões, haja vista a decadência/prescrição mês a mês dos fatos geradores da contribuição.

(Who?) De quem são os dados analisados?

<sup>4</sup> Escovedo, Tatiana (2020-02-27T22:58:59). Introdução a Data Science . Casa do Código. Edição do Kindle.

<sup>5</sup> [http://www.planalto.gov.br/ccivil\\_03/leis/LEIS\\_2001/L10256.htm](http://www.planalto.gov.br/ccivil_03/leis/LEIS_2001/L10256.htm)

<sup>6</sup> <https://its.unl.edu/bestpractices/remember-5-ws>

Os dados analisados são públicos, decisões publicadas no Diário Eletrônico da 1ª Região (e-DJF1) e o inteiro teor dos acórdãos, decisões e despachos do Tribunal Regional Federal da 1ª Região disponibilizados para *download* no *site*<sup>7</sup> do TRF1. Foram coletadas as decisões do tipo ementa já analisadas e classificadas como constitucional ou inconstitucional, por serem mais sucintas e terem o resultado da decisão de forma mais clara e objetiva.

(What?) Quais os objetivos com essa análise? O que iremos analisar?

O objetivo do projeto é análise das decisões por meio de técnicas de NLP com vistas à classificação binomial simétrica em duas categorias: constitucional ou inconstitucional, que, na prática, representa se a União ganhou ou perdeu a ação.

(Where?) Quais os aspectos geográficos e logísticos de sua análise?

A delimitação geográfica foi a jurisdição das ações promovidas no âmbito do TRF1, que abrange o Distrito Federal e os seguintes estados: Acre, Amapá, Amazonas, Bahia, Goiás, Maranhão, Mato Grosso, Minas Gerais, Pará, Piauí, Rondônia, Roraima e Tocantins.

(When?) Qual o período está sendo analisado?

Foram analisadas decisões proferidas de fevereiro de 2011 a abril de 2019, período em que houve mudanças no entendimento da corte constitucional, com repercussão geral.

## 2. Coleta de Dados

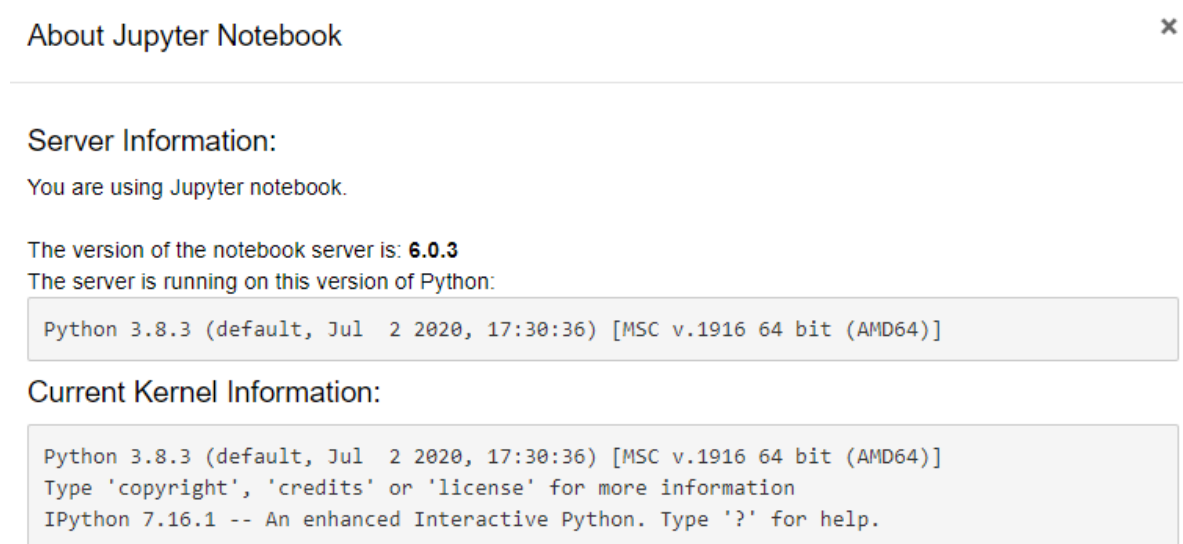
O projeto foi desenvolvido utilizando-se Python versão 3.8.3, uma linguagem de programação interpretada, orientada a objetos e de alto nível com semântica dinâmica<sup>8</sup>, a IDE (*integrated development environment*) ou ambiente de desenvolvimento. Foi utilizado o Jupyter Notebook, versão 6.0.3 (figura 1), um aplicativo cliente-servidor de código aberto usado para criar e executar principalmente projetos de ciência de dados, porque também fornece ferramentas para visualização, simulação numérica e limpeza de dados, dentre outras

<sup>7</sup> <https://arquivo.trf1.jus.br/index.php>

<sup>8</sup> <https://www.python.org/doc/essays/blurb/>

ferramentas.<sup>9</sup> Jupyter é uma sigla de Julia, Python e R, porque foram as primeiras linguagens de programação com suporte para esse editor. Atualmente, o Jupyter oferece suporte a mais de 40 linguagens de programação.

### Figura 1: Versão do Jupyter Notebook e Python



Foram importadas 38 bibliotecas Python (figura 2), além das bibliotecas nativas do Python, para as análises, processamento, tratamento de dados e criação dos modelos de *machine learning*. Basicamente foram utilizadas as bibliotecas: “os” (*miscellaneous operating system interfaces*) para manipulação de diretórios no disco rígido; “Pandas” e “Numpy” para operacionalização e cálculos no *dataset*; “pandas\_profiling” para estatística descrita e outras análises; “BeautifulSoup”, “wget” e “request” para *web scraping* ou *web data extraction*; “win32com” para conversão dos documentos baixados da *Internet*; “Spacy” e “NLTK” para transformação dos textos; “wordcloud”, “matplotlib”, “seaborn” e “statsmodels.graphics” para geração de gráficos; “scipy” para cálculo de medidas estatísticas; e “sklearn” para criação do modelo de *machine learning*.

<sup>9</sup> <https://learnpython.com/blog/jupyter-notebook-python-ide-installation-tips/>



**Figura 2: Bibliotecas Python utilizadas no projeto**

```

import os, unicodedata, re, spacy, nltk, wget, requests
import pandas as pd
import pandas_profiling as pp
import numpy as np
from bs4 import BeautifulSoup
from win32com import client

from spacy.lang.pt.stop_words import STOP_WORDS
import pt_core_news_sm

from nltk.tokenize import WhitespaceTokenizer
from nltk.stem import RSLPStemmer

from wordcloud import WordCloud
import matplotlib as plt
import matplotlib.pyplot as plt
import seaborn as sn

from statsmodels.graphics.gofplots import qqplot
import scipy.stats as stat

from sklearn.datasets import load_files
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn import model_selection
from sklearn.model_selection import KFold, train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

```

A base de dados inicial – em formato XLSX, com os números dos processos e decisões em que as contribuições foram julgadas constitucionais ou inconstitucionais, além das informações de 10.009 decisões – foi complementada com a extração da íntegra das decisões na *web* (*web scraping*) utilizando código Python, que será detalhado mais adiante no capítulo 3.

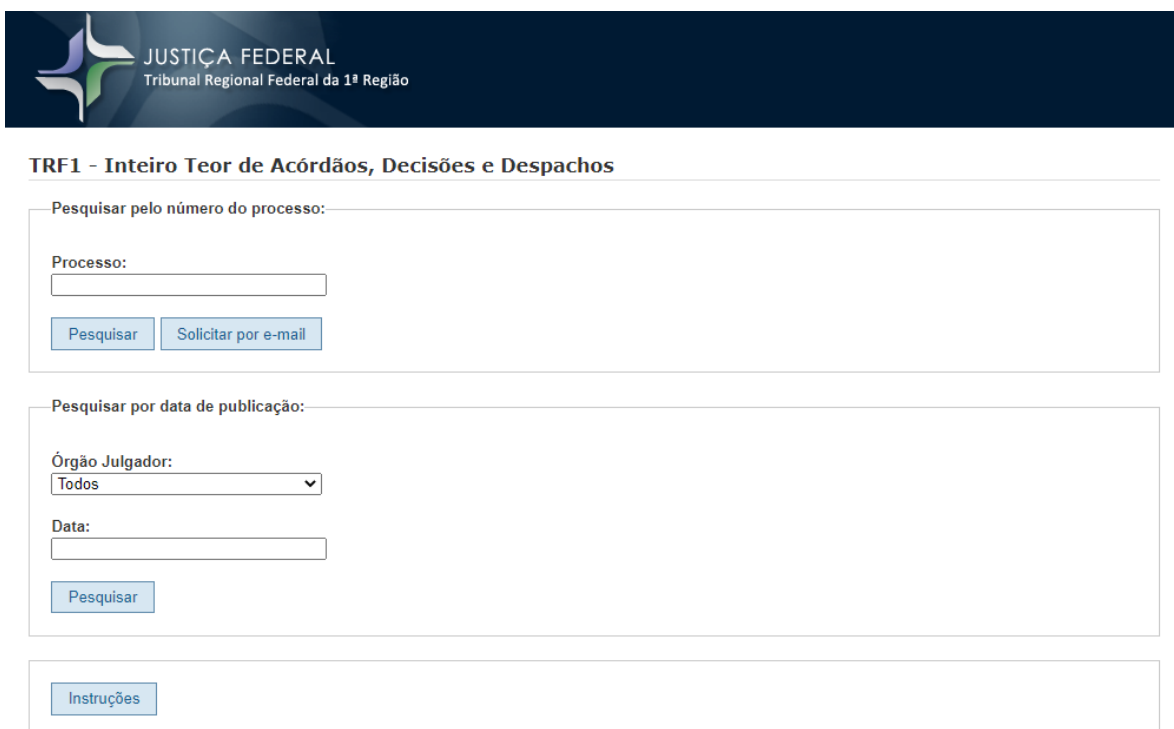
**Tabela 1: Descrição dos campos/colunas do *dataset***

Nome da coluna/campo	Descrição	Tipo
Processo judicial	Número do processo judicial na Justiça Federal	Texto
Decisão	Número da decisão da Justiça Federal de 2º grau	Texto
Resultado da decisão	Resultado da decisão	Categórica

Assunto da petição	Taxonomia processual com o do assunto da discussão judicial	Categórica
Ementa	Decisão judicial extraída do <i>site</i> do TRF1 ( <a href="https://arquivo.trf1.jus.br/index.php">https://arquivo.trf1.jus.br/index.php</a> ) de 16 a 18 de setembro de 2020.	Texto

No *site* do TRF1 (<https://arquivo.trf1.jus.br/index.php>), é possível consultar, pelo número do processo (figura 3), o inteiro teor de acórdãos, decisões e despachos de todos os processos desse tribunal já publicados no Diário da Justiça, proferidos pelas turmas que compõe o 2º grau da Justiça Federal na 1ª Região.

**Figura 3: Site do TRF1 para consulta do inteiro teor das decisões**



The image shows the header of the TRF1 website with the logo and text: "JUSTIÇA FEDERAL Tribunal Regional Federal da 1ª Região". Below the header, the title "TRF1 - Inteiro Teor de Acórdãos, Decisões e Despachos" is displayed. The main content area contains two search sections. The first section, "Pesquisar pelo número do processo:", includes a text input field labeled "Processo:", a "Pesquisar" button, and a "Solicitar por e-mail" button. The second section, "Pesquisar por data de publicação:", includes a dropdown menu labeled "Órgão Julgador:" with "Todos" selected, a text input field labeled "Data:", a "Pesquisar" button, and an "Instruções" button at the bottom.

Conforme instruções na figura 4, os documentos que retornam são arquivos do tipo DOC (Microsoft Word 97-2003) ou TIFF (*tagged image file format*).

**Figura 4: Instruções para realizar a pesquisa**

Instruções para realizar a pesquisa

**Como pesquisar?**

Para ver os documentos de um processo basta preencher o campo "Processo" e pressionar o botão "Pesquisar".

A pesquisa ao inteiro teor dos julgados por e-mail deve ser feita clicando no botão "Solicitar por email" ou diretamente à Divisão de Arquivo Judicial pelo e-mail [acordao@trf1.jus.br](mailto:acordao@trf1.jus.br)

**Quais os processos disponíveis?**

Decisões e acórdãos de todos os processos do TRF 1ª Região já publicados no Diário da Justiça.

**O que é necessário para abrir os documentos?**

Para abrir os arquivos mais recentes, em formato texto, será necessário um visualizador de arquivos do tipo "DOC". Se não possuir o Microsoft Word instalado, poderá instalar apenas o visualizador, que é gratuito e está disponível no site do fornecedor: Visualizador do Word.

Para abrir os arquivos mais antigos, ainda em imagem, será necessário um visualizador de arquivos do tipo "TIFF". Existem vários plug-ins disponíveis para este fim, tais como:

- Tiffsurfer da VisionShape
- AlternaTIFF
- Braval Reader
- Internetiff

Os softwares citados são de responsabilidade de seus fabricantes e a instalação desses componentes é realizada pelo usuário.

Fechar

Os dados com o resultado das decisões, colunas 1 a 5, foram importados da planilha "Decisoes judiciais - treinamento e teste.xlsx" para o *dataframe* Pandas "df\_DecisoesJud", conforme código da figura 6. Todos os campos foram importados como tipo *object* para não se correr o risco de o *dataframe* Pandas assumir formatos incorretos e, por exemplo, números de processos, decisões e CPF serem importados como inteiros e removidos os zeros à esquerda.

Imprimindo-se as informações do *dataframe* gerado, foram importadas, conforme previsto, 10.009 decisões com cinco colunas (figura 5). O código utilizado encontra-se na figura 6.

**Figura 5: *Dataframe* criado com os dados das decisões judiciais**

```

Formato tabela: linhas, colunas
(10009, 5)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10009 entries, 0 to 10008
Data columns (total 5 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   Processo judicial      10009 non-null  object
1   Decisão                10009 non-null  object
2   Tipo de Decisão        10009 non-null  object
3   Resultado da Decisão   2134 non-null   object
4   Assunto                8730 non-null   object
dtypes: object(5)
memory usage: 391.1+ KB
Informações dos dados importados:
None

Amostra dataframe:

```

Out[4]:

	Processo judicial	Decisão	Tipo de Decisão	Resultado da Decisão	Assunto
0	0000007-76.2014.4.01.3802	00000077620144013802_3-1	Ementa	NaN	6040 - Funrural
1	0000007-76.2014.4.01.3802	00000077620144013802_3-2	Ementa	Constitucionalidade	6040 - Funrural
2	0000007-76.2014.4.01.3802	00000077620144013802_3	Ementa	NaN	6040 - Funrural
3	0000049-45.2016.4.01.3806	00000494520164013806_3	Ementa	Constitucionalidade	6040 - Funrural
4	0000064-57.2016.4.01.3824	00000645720164013824_3	Ementa	Constitucionalidade	6040 - Funrural

**Figura 6: Código utilizado para importar os dados da planilha**

```

#Carregando planilha com o resultado das decisões judiciais (grupo de treinamento e teste)
df_DecisoõesJud = pd.read_excel(os.path.join(datasetPuc,
                                             'Decisoões judiciais - treinamento e teste.xlsx'),
                                dtype = 'object')

#Verificando formatos do Dataframe criado
print('Formato tabela: linhas, colunas\n',df_DecisoõesJud.shape,'\n')
print('Informações dos dados importados:\n',df_DecisoõesJud.info(),'\n')
print('Amostra dataframe:')
df_DecisoõesJud.head()

```

Através do número do processo importado foi possível obter e tratar o texto das decisões com um conjunto de funções, uma classe de busca e tratamento de dados na *web* com as seguintes etapas/funções:

1. autenticar(processo): autenticação no *site* do TRF1 com a biblioteca “Request”, passando-se os parâmetros necessários para o *site* e organizando-se o conteúdo da página *web* com a biblioteca “BeautifulSoup”, buscando-se a classe HTML, que contém os *links* para *download* dos documentos das decisões.

**Figura 7: autenticar(processo)**

```
def autenticar(processo):
    #Autenticando no site do TRF1 com parâmetros da página
    chaves = {'numero_processo': processo,
              'pA': processo,
              'pN': processo,
              'p1': processo,
              'orgao': '',
              'nome_orgao': '',
              'data_publicacao': ''}

    page = requests.get('https://arquivo.trf1.jus.br/PesquisaMenuArquivo.asp', params=chaves)

    #Organizando conteúdo da página com o BeautifulSoup
    soup = BeautifulSoup(page.text, "html.parser")

    #Encontrando as classes html que contém os arquivos para download
    documentos = soup.find_all(class_="abreCascata")
    if documentos != None:
        return percorrerDocumentos(documentos)
    return processosSemDecisoes.append(processo)
```

2. percorrerDocumentos(documentos): iteração sobre a lista de *links* encontrados na classe e verificação do tipo ementa.

**Figura 8: percorrerDocumentos(documentos)**

```
def percorrerDocumentos(documentos):
    for documento in documentos: #Percorrendo a classe com os links dos documentos para download
        link = documento.get('href')
        #Se link refere-se a Ementas, codificação igual 3
        if link[link.rfind('_')+1:link.rfind('_')+2] == '3':
            documento = link[link.rfind('/')+1:link.rfind('.')] #Nome do documento
            if documento not in documentosBaixados:
                baixarDocumentos(documento, link)
```

3. baixarDocumentos(documento, link): *download* das decisões em diretórios separados (constitucionalidade ou inconstitucionalidade), conforme resultado da decisão baixada.

**Figura 9: baixarDocumentos(documentos)**

```
def baixarDocumentos(documento, link): #Baixar decisões do site do TRF1
    consulta = 'decisao == "' + str(documento) + '"'
    if df_DecisoesJud.query(consulta)['resultado_decisao'].size > 0:
        resultado_decisao = df_DecisoesJud.query(consulta)['resultado_decisao'].values[0]
        if resultado_decisao == 'Constitucionalidade':
            arquivoDoc = os.path.join(diretorioConstituc, link[link.rfind('/')+1:])
            wget.download(link, out=arquivoDoc)
            documentosBaixados.append(documento)
            #converterDocumentos(arquivoDoc)
        elif resultado_decisao == 'Inconstitucionalidade':
            arquivoDoc = os.path.join(diretorioInconstituc, link[link.rfind('/')+1:])
            wget.download(link, out=arquivoDoc)
            documentosBaixados.append(documento)
            #converterDocumentos(arquivoDoc)
```

4. converterDocumentos(arquivoDoc): conversão dos documentos tipo DOC baixados para o formato TXT para importação pela biblioteca de *machine learning*.

**Figura 10: converterDocumentos(arquivoDoc)**

```
def converterDocumentos(arquivoDoc): #Converter decisões formato word .doc para formato texto
    try:
        arquivoTexto = arquivoDoc.replace('.doc', '.txt')
        word = client.DispatchEx("Word.Application")
        documentoWord = word.Documents.Open(arquivoDoc)
        documentoWord.SaveAs(arquivoTexto, FileFormat = 7)
        documentoWord.Close()
        os.remove(arquivoDoc)
    except (Exception, e):
        print (e)
    finally:
        word.Quit()
```

5. checarDocumentosFaltantes(): checagem para assegurar que todas as decisões do *dataframe* foram baixadas.

**Figura 11: checarDocumentosFaltantes()**

```
def checarDocumentosFaltantes():
    decisoesDataFrame = [x for x in df.DecisoesJud['decisao']]
    documentosBaixados = [x.strip('.txt') for x in (os.listdir(diretorioConstituc) + os.listdir(diretorioInconstituc))]
    documentosNaoBaixados = [y for y in decisoesDataFrame if y not in documentosBaixados]
    print('Decisões DataFrame:', len(decisoesDataFrame))
    print('Documentos Baixados:', len(documentosBaixados))
    print('Documentos Não-baixados:', len(documentosNaoBaixados))
    return documentosNaoBaixados
```

**Figura 12: Código utilizado para *download* das decisões**

```
#Decisões que devem ser baixadas
decisoesDataFrame = [x for x in df.DecisoesJud['decisao']]

#Decisões já baixadas em disco
documentosBaixados = [x.strip('.txt') for x in (os.listdir(diretorioConstituc) + os.listdir(diretorioInconstituc))]

#Lista comparação documentos que devem ser baixados versus já baixados
documentosNaoBaixados = [y for y in decisoesDataFrame if y not in documentosBaixados]

processosSemDecisoes = [] #Lista com processos que não foram encontradas decisões no site do TRF
print('Decisões DataFrame:', len(decisoesDataFrame))
print('Documentos Baixados:', len(documentosBaixados))
print('Documentos Não-baixados:', len(documentosNaoBaixados))

#Percorrendo lista de decisões não baixadas para download da decisão
contador = 1
for documento in documentosNaoBaixados:
    print('\nBaixando decisão ' + str(contador) + '/' + str(len(documentosNaoBaixados)))
    autenticar(documento[:documento.rfind('_')])
    contador += 1
checarDocumentosFaltantes() #Checando se ainda existem documentos não baixados
```

**Figura 13: Output do código utilizado para *download* das decisões**

```
Decisões Dataframe: 1965
Documentos Baixados: 765
Documentos Não-baixados: 1216

Baixando decisão 1/1216
100% [.....] 54495 / 54495
Baixando decisão 2/1216

Baixando decisão 3/1216
100% [.....] 67584 / 67584
Baixando decisão 4/1216
100% [.....] 55338 / 55338
Baixando decisão 5/1216

Baixando decisão 6/1216
100% [.....] 54387 / 54387
Baixando decisão 7/1216
100% [.....] 69632 / 69632
```

Ao final do processo de *download* das decisões – realizado em três dias no período da noite para não prejudicar os serviços do site – e realizada a conferência das decisões com os arquivos gravados em disco, verificou-se que todas as decisões do tipo ementa foram baixadas.

**Figura 14: Conferência das decisões baixadas com os arquivos gravados em disco**

```
Total de decisões:
Decisões Dataframe: 1965
Documentos Baixados: 1965
Documentos Não-baixados: 0

.....
Decisões por resultado: "Constitucionalidade"
Decisões Dataframe: 1061
Documentos Baixados: 1061
Documentos Não-baixados: 0

.....
Decisões por resultado: "Inconstitucionalidade"
Decisões Dataframe: 904
Documentos Baixados: 904
Documentos Não-baixados: 0
```

**Figura 15: Código utilizado para conferência das decisões baixadas com os arquivos gravados em disco**

```
print('Total de decisões:')
decisoesDataFrame = [doc for doc in df_DecisoJud['decisao']] #Decisões que devem ser baixadas
documentosBaixados = [x.strip('.txt') for x in (os.listdir(diretorioConstituc)
+ os.listdir(diretorioInconstituc)) if x.endswith('.txt')] #Decisões já baixadas em disco

#Lista comparação documentos que devem ser baixados versus já baixados
documentosNaoBaixados = [y for y in decisoesDataFrame if y not in documentosBaixados]
print('Decisões DataFrame:',len(decisoesDataFrame))
print('Documentos Baixados:',len(documentosBaixados))
print('Documentos Não-baixados:',len(documentosNaoBaixados))

print('\n.....\nDecisões por resultado: "Constitucionalidade"')
decisoesDataFrameConst = [x for x in df_DecisoJud.query('resultado_decisao == "Constitucionalidade"')['decisao']]
documentosBaixadosConst = [x.strip('.txt') for x in os.listdir(diretorioConstituc) if x.endswith('.txt')]
documentosNaoBaixadosConst = [y for y in decisoesDataFrameConst if y not in documentosBaixadosConst]
print('Decisões DataFrame:',len(decisoesDataFrameConst))
print('Documentos Baixados:',len(documentosBaixadosConst))
print('Documentos Não-baixados:',len(documentosNaoBaixadosConst))

print('\n.....\nDecisões por resultado: "Inconstitucionalidade"')
decisoesDataFrameInconst = [x for x in df_DecisoJud.query('resultado_decisao == "Inconstitucionalidade"')['decisao']]
documentosBaixadosInconst = [x.strip('.txt') for x in os.listdir(diretorioInconstituc) if x.endswith('.txt')]
documentosNaoBaixadosInconst = [y for y in decisoesDataFrameInconst if y not in documentosBaixadosInconst]
print('Decisões DataFrame:',len(decisoesDataFrameInconst))
print('Documentos Baixados:',len(documentosBaixadosInconst))
print('Documentos Não-baixados:',len(documentosNaoBaixadosInconst))
```

Concluída a fase de coleta de dados, após muitos ajustes no código, passou-se à etapa de tratamento dos dados.

### 3. Processamento/Tratamento de Dados

As operações de ETL (*extraction, transformation, loading*) dos dados, atividades de pré-processamento, representam a etapa mais demorada e trabalhosa de um projeto de *data science*, consumindo pelo menos 70% do tempo total do projeto, segundo Escovedo<sup>10</sup>: “Nesta etapa, pode ser necessário remover ou complementar dados faltantes; corrigir ou amenizar dados discrepantes (*outliers*) e desbalanceamento entre classes, e selecionar as variáveis e instâncias mais adequadas para compor o(s) modelo(s) que serão construídos na etapa seguinte.”

No presente projeto foram necessários inúmeros procedimentos de *data cleaning*, assim como um extenso *pipeline* de transformação dos dados brutos das decisões baixados da *web* e importados via planilha:

1. transformação e conversão das colunas do *dataframe*;
2. remoção de linhas que não possuem os dados necessários;

<sup>10</sup> Escovedo, Tatiana (2020-02-27T22:58:59). Introdução a Data Science . Casa do Código. Edição do Kindle.



3. remoção de dados duplicados;
4. rotulagem dos atributos nº do processo e nº da decisão;
5. aplicação de *stop words*;
6. aplicação de dicionário de termos jurídicos;
7. remoção de partes desnecessárias dos documentos, cabeçalho e parte final da decisão;
8. remoção de datas e horas, palavras com menos de 3 caracteres, pontuações, caracteres especiais e acentos; e
9. aplicação de *stemming*

Após a criação do *dataframe* Pandas com os dados importados da planilha, a primeira transformação realizada foi no nome das colunas, retirando-se acentos, espaços, cedilhas, caracteres especiais, e deixando-se tudo em letras minúsculas com o código da figura 16.

**Figura 16: Código utilizado para transformação do nome das colunas**

```
#Normalizando nomes das colunas

#Removendo palavras com menos de 3 caracteres e números maiores que 10 caracteres.
colunas_titulos = [' '.join([palavra for palavra in coluna.split() if len(palavra) > 3])
                    for coluna in df_DecisoJud.columns]

#Removendo pontuações, caracteres especiais e transformando texto em minúsculo
colunas_titulos = [re.sub('[^\\w\\d\\s]+', ' ', palavra).lower() for palavra in colunas_titulos]

#Removendo acentos e cedilha
df_DecisoJud.columns = [' '.join([c for c in unicodedata.normalize('NFKD', coluna)
                                  if not unicodedata.combining(c)].lower().replace(' ', '_'))
                        for coluna in colunas_titulos]

df_DecisoJud.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10009 entries, 0 to 10008
Data columns (total 5 columns):
#   Column                Non-Null Count  Dtype
---  ---                ---
0   processo_judicial      10009 non-null  object
1   decisao                10009 non-null  object
2   tipo_decisao           10009 non-null  object
3   resultado_decisao      2134 non-null   object
4   assunto                8730 non-null   object
dtypes: object(5)
memory usage: 391.1+ KB
```

Na sequência, fez-se a conversão dos tipos das colunas do *dataframe* para os títulos de dados mais adequados às análises (figura 17).

**Figura 17: Conversão das colunas para os tipos adequados às análises**

```
#Converter colunas para os tipos corretos:
df_DecisoesJud = df_DecisoesJud.astype({'processo_judicial':'string',
                                         'decisao':'string',
                                         'tipo_decisao': 'category',
                                         'resultado_decisao':'category',
                                         'assunto': 'category'})

df_DecisoesJud.dtypes

processo_judicial    string
decisao              string
tipo_decisao         category
resultado_decisao    category
assunto              category
dtype: object
```

A contagem de valores nulos mostrou 7.875 linhas sem o resultado da decisão e 1.279 linhas sem o assunto da petição (figura 18).

**Figura 18: Contagem de valores nulos (dados faltantes)**

```
#Contando valores nulos
df_DecisoesJud.isnull().sum()

processo_judicial    0
decisao              0
tipo_decisao         0
resultado_decisao    7875
assunto              1279
dtype: int64
```

Considerando-se que o resultado da decisão e o assunto da petição são dados imprescindíveis para as análises e muito específicos (o assunto da petição será utilizado para filtrar as decisões para um tema específico, 6040 – Funrural, e remover os que não fazem parte do escopo da pesquisa; e o resultado da decisão representa o *y*, o *target* dos modelos de aprendizado supervisionado que serão utilizados), optou-se, portanto, por não inferir os dados faltantes desses atributos e por excluir essas linhas do *dataframe*, uma vez que não há prejuízos nas análises posteriores – pelo contrário, o procedimento torna o *dataset* mais robusto e consistente (figura 19).

**Figura 19: Remoção de valores nulos ou com tipo de decisão e assunto fora do objeto da pesquisa**

```
#Eliminando registros/linhas que não possuem a informação do resultado da decisão
print('Shape antes: ' + str(df_DecisoesModuleJud.shape))
df_DecisoesModuleJud.dropna(subset=['resultado_decisao'],inplace=True)

#Filtrando para decisão tipo "Ementa" e assunto petição "6040 - Funrural"
df_DecisoesModuleJud = df_DecisoesModuleJud.query('tipo_decisao == "Ementa" & assunto == "6040 - Funrural"')

print('Shape depois: ' + str(df_DecisoesModuleJud.shape))
```

Shape antes: (10009, 5)  
Shape depois: (1993, 5)

Pelos resultados do processamento, verificou-se que do total de 10.009, após a depuração do *dataset*, 1.993 linhas representaram decisões do tipo ementa com a informação do resultado da decisão. Por precaução, foi utilizado o método “pivot table” da biblioteca Pandas para conferência das alterações no *dataframe*. Pelos resultados das imagens 20, 21 e 22, vê-se claramente que a remoção dos valores nulos e registros fora do objeto da pesquisa foi implementada corretamente.

**Figura 20: Conferindo *data cleaning* no tipo decisão**

```
#Conferindo data cleaning no tipo decisão
df_DecisoesModuleJud.pivot_table(df_DecisoesModuleJud,columns=['tipo_decisao'],aggfunc={'decisao': 'count'}).T
```

	decisao
tipo_decisao	
Certidão	0
Decisão	0
Decisão ou Despacho	0
Despacho	0
Ementa	1993
Sentença	0
Tutela antecipada	0

**Figura 21: Conferindo *data cleaning* no assunto**

```
#Conferindo data cleaning no assunto
df_DecisoesModuleJud.pivot_table(df_DecisoesModuleJud,columns=['assunto'],aggfunc={'decisao': 'count'}).T
```

	decisao
assunto	
6007 - Repetição de indébito	0
6037 - Salário-Educação	0
6038 - Seguro Acidentes do Trabalho	0
6040 - Funrural	1993
6041 - Contribuição INCRA	0
6048 - Contribuições Previdenciárias	0
6166 - Renúncia ao benefício	0

**Figura 22: Conferindo *data cleaning* no resultado da decisão**

```
#Conferindo data cleaning no resultado da decisão: consticional, inconstitucional
df_DecisoJud.pivot_table(df_DecisoJud,columns=['resultado_decisao'],
                           aggfunc={'decisao': 'count'}).T
```

decisao	
resultado_decisao	
Constitucionalidade	1075
Inconstitucionalidade	918

Na figura 23, a verificação de dados duplicados pelos métodos "nunique" e "duplicated" demonstrou que havia menos processos únicos do que o tamanho do *dataframe*, o que não representa problema, haja vista que pode haver mais de uma decisão em cada processo; no entanto, o número de decisões únicas não pode ser menor que o tamanho do *dataframe*, e cada linha deve corresponder a uma decisão única, evidenciando-se que existem decisões duplicadas no *dataset*, o que foi confirmado pelo método "duplicated" igual a *true*.

**Figura 23 – Verificação de registros duplicados**

```
#Verificando dados duplicados
print('Shape: ' + str(df_DecisoJud.shape))
print('Quant. Processos únicos: ' + str(df_DecisoJud['processo_judicial'].nunique()))
print('Quant. Decisões únicas: ' + str(df_DecisoJud['decisao'].nunique()))
#Número de decisões (UNIQUE) menor que o número de linhas: indicativo de duplicidade de decisões.

#Verificando/confirmando duplicidade de decisões
print('Duplicidade método duplicated: ' + str(any(df_DecisoJud['decisao'].duplicated())))
```

```
Shape: (1993, 5)
Quant. Processos únicos: 1122
Quant. Decisões únicas: 1965
Duplicidade método duplicated: True
```

Para remoção dos registros duplicados foi utilizado o método "drop\_duplicates", resultando, ao final, no *dataframe* "df\_DecisoJud" sem duplicidade de registros.

**Figura 24 – Remoção de registros duplicados**

```
#Eliminando registros com duplicidade de decisões
df_DecisoesModuleJud = df_DecisoesModuleJud.drop_duplicates('decisao')
print('Shape: ' + str(df_DecisoesModuleJud.shape))
print('Quant. Decisões: ' + str(df_DecisoesModuleJud['decisao'].nunique()))
#Resultado total de linhas = total de decisões: Exclusão de duplicidades OK

print('Duplicidade método duplicated: ' + str(any(df_DecisoesModuleJud['decisao'].duplicated())))

Shape: (1965, 5)
Quant. Decisões: 1965
Duplicidade método duplicated: False
```

Apesar de os dados utilizados no projeto serem públicos, as decisões publicadas no e-DJF1, optou-se por rotular (figura 25) o número do processo e o número da decisão, com o objetivo de impossibilitar a identificação dos autores dos processos.

**Figura 25 – Rotulagem do número do processo e da decisão**

```
#Criando rótulo para o número do processo e decisão para impossibilitar a identificação do interessado.

#Label processo
df_DecisoesModuleJud_LabelProcesso = pd.pivot_table(df_DecisoesModuleJud, columns=['processo_judicial'],
                                                    aggfunc={'decisao': 'count'}).T

#Criando coluna com rótulo sequencial para index
df_DecisoesModuleJud_LabelProcesso['processo_label'] = ['processo ' + str(x+1)
                                                    for x in range(len(df_DecisoesModuleJud_LabelProcesso))]

df_DecisoesModuleJud_LabelProcesso.drop('decisao', axis=1, inplace=True)
print('Label Processo:\nFormato tabela: linhas, colunas', df_DecisoesModuleJud_LabelProcesso.shape)
print(df_DecisoesModuleJud_LabelProcesso.head())

#Label decisao
df_DecisoesModuleJud_LabelDecisao = pd.pivot_table(df_DecisoesModuleJud, columns=['decisao'],
                                                    aggfunc={'decisao': 'count'}).T
df_DecisoesModuleJud_LabelDecisao['decisao_label'] = ['decisao ' + str(x+1)
                                                    for x in range(len(df_DecisoesModuleJud_LabelDecisao))]
df_DecisoesModuleJud_LabelDecisao.drop('decisao', axis=1, inplace=True)
print('\nLabel Decisao:\nFormato tabela: linhas, colunas', df_DecisoesModuleJud_LabelDecisao.shape)
print('\n', df_DecisoesModuleJud_LabelDecisao.head())

#Aplicando rótulos criados: processo_judicial e decisao
df_DecisoesModuleJud_Rot = df_DecisoesModuleJud
df_DecisoesModuleJud_Rot = pd.merge(df_DecisoesModuleJud_Rot, df_DecisoesModuleJud_LabelProcesso, on='processo_judicial', how='left')
df_DecisoesModuleJud_Rot = pd.merge(df_DecisoesModuleJud_Rot, df_DecisoesModuleJud_LabelDecisao, on='decisao', how='left')
#Excluindo as colunas substituídas pelos rótulos
df_DecisoesModuleJud_Rot.drop(columns=['processo_judicial', 'decisao'], axis=1, inplace=True)
df_DecisoesModuleJud_Rot.head()
```

	tipo_decisao	resultado_decisao	assunto	processo_label	decisao_label
0	Ementa	Constitucionalidade	6040 - Funrural	processo 1	decisao 1
1	Ementa	Constitucionalidade	6040 - Funrural	processo 2	decisao 2
2	Ementa	Constitucionalidade	6040 - Funrural	processo 3	decisao 3
3	Ementa	Constitucionalidade	6040 - Funrural	processo 4	decisao 4
4	Ementa	Constitucionalidade	6040 - Funrural	processo 5	decisao 6

Após a coleta e conversão dos dados das decisões da *web*, conforme descrição no capítulo 2, os arquivos TXT com os textos das decisões foram importados para um conjunto de dados do tipo “*sklearn.utils.Bunch*”, nomeado como

“df\_DecisoeseJud\_dadosTrein”, utilizando-se o método “load\_files” da biblioteca “Scikit-Learn” (figura 26).

**Figura 26 – Importação dos textos das decisões**

```
#Importação dos arquivos txt gravados em disco, utilizando sklearn load_files
from sklearn.datasets import load_files
df_DecisoeseJud_dadosTrein = load_files(os.path.join(datasetPuc, 'decisoese_trf'),
                                       encoding='latin1',
                                       description='Base de treinamento')
X, y = df_DecisoeseJud_dadosTrein.data, df_DecisoeseJud_dadosTrein.target
type(df_DecisoeseJud_dadosTrein)

sklearn.utils.Bunch
```

Conforme Géron (2019)<sup>11</sup>, os conjuntos de dados carregados pelo “Scikit-Learn” geralmente possuem uma estrutura similar à de dicionário. Iterando sobre o conjunto de dados importados, “df\_DecisoeseJud\_dadosTrein”, encontram-se os seguintes elementos aninhados, conforme figuras 27 e 28:

- “data”: lista com o texto das decisões (X);
- “target”: Narray com a chave-alvo contendo o array com os rótulos (y);
- “filenames”: Narray com a relação de arquivos importados;
- “target\_names”: lista com os nomes dos rótulos y; e
- “DESCR”: *string* com a descrição do conjunto de dados.

**Figura 27 – Elementos aninhados no conjunto de dados “Scikit-Learn”**

```
y #df_DecisoeseJud_dadosTrein.target
array([0, 0, 0, ..., 1, 0, 0])

df_DecisoeseJud_dadosTrein.filenames
array(['D:\\Google Drive\\Ciência de Dados e Big Data - Curso PUC Minas\\Disciplina 13. TCC Ciência d
e Dados e Big Data (2019)\\dataset\\decisoese_trf\\constitucionalidade\\00003952520134013604_3-3.txt',
      'D:\\Google Drive\\Ciência de Dados e Big Data - Curso PUC Minas\\Disciplina 13. TCC Ciência d
e Dados e Big Data (2019)\\dataset\\decisoese_trf\\constitucionalidade\\00070005020104013811_3-2.txt',
      'D:\\Google Drive\\Ciência de Dados e Big Data - Curso PUC Minas\\Disciplina 13. TCC Ciência d
e Dados e Big Data (2019)\\dataset\\decisoese_trf\\constitucionalidade\\00026243620104013806_3-2.txt',
      ...,
      'D:\\Google Drive\\Ciência de Dados e Big Data - Curso PUC Minas\\Disciplina 13. TCC Ciência d
e Dados e Big Data (2019)\\dataset\\decisoese_trf\\inconstitucionalidade\\00063087520104013803_3-1.tx
t',
      'D:\\Google Drive\\Ciência de Dados e Big Data - Curso PUC Minas\\Disciplina 13. TCC Ciência d
e Dados e Big Data (2019)\\dataset\\decisoese_trf\\constitucionalidade\\00042895720154013824_3-1.txt',
      'D:\\Google Drive\\Ciência de Dados e Big Data - Curso PUC Minas\\Disciplina 13. TCC Ciência d
e Dados e Big Data (2019)\\dataset\\decisoese_trf\\constitucionalidade\\00062126020104013803_3-2.tx
t'],
      dtype='<U187')

df_DecisoeseJud_dadosTrein.target_names
['constitucionalidade', 'inconstitucionalidade']

df_DecisoeseJud_dadosTrein.DESCR
'Base de treinamento'
```

<sup>11</sup> Géron, Aurélien. *Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow* (p. 84). Edição do Kindle.”

**Figura 28 – Tipos dos elementos aninhados no conjunto de dados “Scikit-Learn”**

```
for x in df_DecisoesModule_dadosTrein: print(x)

data
filenames
target_names
target
DESCR

print(type(df_DecisoesModule_dadosTrein.data))
print(type(df_DecisoesModule_dadosTrein.filenames))
print(type(df_DecisoesModule_dadosTrein.target_names))
print(type(df_DecisoesModule_dadosTrein.target))
print(type(df_DecisoesModule_dadosTrein.DESCR))

<class 'list'>
<class 'numpy.ndarray'>
<class 'list'>
<class 'numpy.ndarray'>
<class 'str'>
```

Com o objetivo de realizar mais alguns exercícios em um *dataframe* Pandas optou-se por criar um novo *dataset* com o X (previsor) e y (alvo), conforme figura 29.

**Figura 29 – Criação de *dataframe* Pandas com X (previsor) e y (alvo)**

```
df_DecisoesModule = pd.DataFrame(data={'ementa': X, 'resultado_decisao': y})
df_DecisoesModule
```

	ementa	resultado_decisao
0	RELATOR \r\rDESEMBARGADOR FEDERAL JOSÉ AMILCA...	0
1	RELATOR \r\rDESEMBARGADOR FEDERAL JOSÉ AMILCA...	0
2	RELATOR \r\rDESEMBARGADOR FEDERAL ITALO FIOR...	0
3	APELAÇÃO CÍVEL 0002275-33.2010.4.01.3806/MG\r...	1
4	RELATORA \r\rDESEMBARGADORA FEDERAL ÂNGELA CA...	0
...	...	...
1960	Numeração Única: 79127720104013801 \r\rAPELAÇÃ...	0
1961	\r\rAPELAÇÃO CÍVEL 0002016-23.2010.4.01.3811/...	1
1962	\r\rAPELAÇÃO CÍVEL 0006308-75.2010.4.01.3803/...	1
1963	\r\rAPELAÇÃO CÍVEL 0004289-57.2015.4.01.3824/...	0
1964	RELATOR \r\rDESEMBARGADOR FEDERAL ITALO FIOR...	0

1965 rows x 2 columns

Como em todo projeto de NLP que tem por objetivo a classificação de texto, é importante realizar a remoção de *stop words*, palavras irrelevantes que geralmente têm pouco conteúdo lexical e sua presença em um texto não consegue distingui-lo de outros textos<sup>12</sup>. A remoção desses ruídos melhora a precisão dos algoritmos<sup>13</sup>, e

<sup>12</sup> Steven Bird, Ewan Klein, and Edward Loper (2009-06-12). Natural Language Processing with Python (Locais do Kindle 222-224). O'Reilly Media. Edição do Kindle.

também simplifica o modelo, ao mesmo tempo que reduz o tamanho e a complexidade do *corpus*.

A biblioteca “NLTK” tem um *corpus* com uma lista de *stop words* em português, que foi complementado pelo *corpus*, também em português, da biblioteca “Spacy”. Alguns termos específicos do projeto foram acrescentados à lista de *stop words* incorporada pelas *libs*, ao passo que outros foram removidos, já que, no presente projeto, não podem ser ignorados nos documentos sob pena de prejudicar o modelo, a exemplo da palavra “não”, que especifica se uma lei “constitucionalizou a contribuição” ou “não constitucionalizou a contribuição”: “... a Lei n. 10.256/2001 (c/c EC n. 20/98) não constitucionalizou a contribuição...”.

### Figura 30 – Lista de *stop words*

```
#Definição da lista de StopWords. Mesclando StopWords da spaCy e NLTK + Stopwords extras do projeto
stopwordsNltk = nltk.corpus.stopwords.words('portuguese')
stopWordsSpacy = [x for x in STOP_WORDS]
stopWordsExtra = stopwordsNltk + [x for x in stopWordsSpacy if x not in stopwordsNltk]
+ [x for x in stopWordsExtras if x not in stopwordsNltk]
stopWords = [x for x in stopWordsExtra if x not in stopWordsMantidas]

print(stopWords)
```

Para facilitar a compreensão das saídas e relatórios (*n-grams*, nuvem de palavras etc.), foi implementado um dicionário de termos jurídicos do tema, contemplando os recursos extraordinários, súmulas do STF, resoluções do Senado Federal e principais leis, decretos e datas importantes, como o marco temporal para correção das restituições (figura 31).

### Figura 31 – Dicionário de termos jurídicos

```
#Dicionário de termos jurídicos

termosJuridicosDic = {'566621r': 'RE566621', '537623r': 'RE537623', '596177r': 'RE596177', '718874r': 'RE718874',
'821291': 'L8212/91', '854092': 'L8540/92', '952897': 'L9528/97', '925095': 'L9250/95',
'669stf': 'S669', '2098': 'E20/1998', '566621': 'RE566621', '596177': 'RE596177',
'718874': 'RE718874', '82121991': 'L8212/91', '85401992': 'L8540/92', '95281997': 'L9528/97',
'92501995': 'L9250/95', '669stf': 'S669', '669': 'S669', '152017': 'R15/2017',
'152007': 'R15/2017', '15/2017': 'R15/2017', '363852': 'RE363852', '1182005': 'LC118/2005',
'546128rj': 'RE546128', '489156sc': 'RE489156', '363852mg': 'RE363852',
'1164452mg': 'RE1164452', '1059571r': 'RE1059571', '11941': 'L11941',
'10637': 'L10637', '1025601': 'L10256', '102562001': 'L10256'}
```

Foi necessário também limpar o texto das informações-padrão, cabeçalho e fechamento, do início e do final das decisões, com os dados dos processos, autor, interessado, julgador, turma etc., que não contribuem para a classificação dos

<sup>13</sup> Beysolow II, Taweh (2018-09-11). Applied Natural Language Processing with Python (Locais do Kindle 213-215). Apress. Edição do Kindle.



documentos, criando uma lista de termos que identificam o ponto de corte inicial e final das decisões (figura 30).

**Figura 32 – Lista de termos para *strip* do texto inicial e final das decisões**

```
#Retirar início padrão (e desnecessário para o modelo) das decisões
stripEmentaInicio = ['TRIBUTÁRIO', 'PREVIDENCIÁRIO', 'PROCESSUAL CIVIL', 'EMENTA', 'E M E N T A',
                    'TRIBUTÁRIO.', 'PREVIDENCIÁRIO.', 'PROCESSUAL CIVIL.', 'EMENTA ', 'E M E N T A ']

#Retirar final padrão (e desnecessário para o modelo) da decisão
stripEmentaFinal = ['Sétima Turma do TRF da 1ª Região', '8ª Turma do TRF da 1ª Região',
                   '8ª Turma do TRF da 1ª Região', '7ª Turma do TRF da 1ª Região',
                   '7ª Turma do TRF-1ª Região', '7ª Turma do TRF - 1ª Região',
                   'Brasília-DF', 'Brasília/DF', 'Brasília, ', 'TRF da 1ª Região ']
```

Foi realizada a segmentação do texto em *tokens*, separando-se as palavras com o método “split”, nativo do Python, em uma nova coluna do *dataframe*, preparando-se o *dataset* para as próximas tarefas.

Considerando-se que o *corpus* é formado por textos jurídicos, decisões judiciais, há muitas inflexões de palavras com o mesmo sentido, como “constituição”, “constitucionalizar”, “constitucionalizado” e “constitucionalizou”, que representam diferentes formas gramaticais da mesma palavra. Tais derivações de uma mesma palavra, além de aumentar o tamanho e a complexidade do *corpus*, podem reduzir o poder de classificação dos algoritmos.

Reduzir as formas flexionais e as formas derivadas de uma palavra a uma base comum é o objetivo das técnicas de *stemming* e *lemmatization*<sup>14</sup>. Optou-se pela primeira, que retornou um *corpus* mais compacto que a segunda.

Foi utilizado o RSLP Portuguese Stemmer (removedor de sufixos da língua portuguesa) da biblioteca “NLTK” para combinar/agrupar palavras com significado semelhante no mesmo “balde” ou grupo, compactando o vocabulário de *tokens*.

Através da função `processarTexto(ementa)` (figura 33), foram realizados os procedimentos de *data cleaning* e pré-processamento do texto, retornando a ementa das decisões preparada para análise:

<sup>14</sup> <https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html> © 2008 Cambridge University Press

1. remoção do texto não necessário, do início e final das decisões;
2. remoção de datas e horas;
3. remoção de acentos, pontuações e caracteres especiais;
4. substituição dos termos jurídicos;
5. tokenização do texto;
6. remoção de *stop words*;
7. transformação do texto em minúsculo;
8. aplicação de *stemming*.

**Figura 33 – Função para pré-processamento do texto**

```
def processarTexto(ementa):
    stemmer = nltk.stem.RSLPStemmer()

    #Removendo INÍCIO da ementa
    for corte in stripEmentaInicio:
        if corte in ementa:
            ementa = ementa[ementa.index(corte):]
            break

    #Removendo final da ementa
    for corte in stripEmentaFinal:
        if corte in ementa:
            ementa = ementa[:ementa.index(corte)]
            break

    #Removendo datas, horas e outros
    ementa = [palavra for palavra in ementa.split()
               if re.match('^(?:[0-9]{2}[:\/,.-;]){2}[0-9]{2,4}$|^(?:[0-9]{2}[:\/,.-;]){2}[0-9]{2,4}[:\/,).;:~)')
               , palavra) != None]

    #Retirando palavras com menos de 3 caracteres.
    ementa = [palavra for palavra in ementa if len(palavra)>2]

    #Retirando pontuações, caracteres especiais e transformando texto em minúsculo
    ementaSemPontuacao = [re.sub('[^\w\d\s]+', '', palavra).lower() for palavra in ementa]

    #Aplicando Steeming (redução palavras ao radical) e eliminando stopWords
    ementaSteeming = [stemmer.stem(palavra) for palavra in ementaSemPontuacao
                       if palavra not in stopWords and len(palavra)>2 and len(palavra)<20]

    #Substituindo termos jurídicos com dicionário (if else)
    ementaSteeming = [termosJuridicosDic[palavra] if palavra in termosJuridicosDic
                       else palavra for palavra in ementaSteeming]

    #Retornando texto após steeming sem acentos
    ementaProcessada = ["".join([c for c in unicodedata.normalize('NFKD', palavra)
                                 if not unicodedata.combining(c)]) for palavra in ementaSteeming]

    return ementaProcessada
```

A partir do pré-processamento dos textos, com o objetivo de facilitar as análises, foram incluídas quatro novas colunas no *dataframe* “df\_DecisoJud” (figuras 34 e 35):

1. ementaProcessada: ementa pré-processada, após transformações;
2. ementaTokenizada: ementa pré-processada e tokenizada;
3. ementaProcessadaTam: tamanho da ementa pré-processada; e
4. ementaTokenizadaTam: tamanho da ementa tokenizada.

**Figura 34 – Código para criação de colunas com os resultados do pré-processamento do texto**

```
#Incluindo nova coluna com os valores alterados no início do dataframe
df_DecisoesModule.insert(1, 'ementaTokenizada', df_DecisoesModule['ementa'].apply(processarTexto))
#Incluindo nova coluna com os valores alterados no início do dataframe
df_DecisoesModule.insert(1, 'ementaProcessada', df_DecisoesModule['ementaTokenizada'].apply(lambda x: ' '.join(x)))

#Incluindo nova coluna com o tamanho da ementaTokenizada
df_DecisoesModule.insert(1, 'ementaTokenizadaTam', [len(x) for x in df_DecisoesModule['ementaTokenizada']])
#Incluindo nova coluna com o tamanho da ementaProcessada
df_DecisoesModule.insert(1, 'ementaProcessadaTam', [len(x) for x in df_DecisoesModule['ementaProcessada']])

df_DecisoesModule.head()
```

**Figura 35 – Dataframe após pré-processamento do texto**

	ementaProcessadaTam	ementaTokenizadaTam	ementaProcessada	ementaTokenizada	resultado_decisao
0	515	78	tribut embarg declar contribu funr L10256 prod...	[tribut, embarg, declar, contribu, funr, L1025...	0
1	654	101	tribut juiz adequ contribu soc funr empreg rur...	[tribut, juiz, adequ, contribu, soc, funr, emp...	0
2	849	131	tribut process civil juiz retrat repercuss tra...	[tribut, process, civil, juiz, retrat, repercu...	0
3	1737	274	tribut prescr contribu soc funr empreg rural p...	[tribut, prescr, contribu, soc, funr, empreg, ...	1
4	816	123	tribut contribu soc incid comerci produc empre...	[tribut, contribu, soc, incid, comerci, produc...	0

Finalizada a fase de pré-processamento dos textos, passou-se às tarefas de análise e processamento de dados.

## 4. Análise e Exploração dos Dados

Iniciou-se a fase de análise e exploração dos dados com o método "Profile Report" da biblioteca "pandas\_profiling"<sup>15</sup>, que possibilita uma rápida análise exploratória dos dados do *dataframe*: estatística descritiva com medidas de dispersão e medidas de posição, valores extremos, dados faltantes, correlações etc.

No *overview* do "Pandas Profiling Report" (figura 36), são demonstradas as estatísticas do *dataset* e os tipos de variáveis. Mostra-se que não há dados faltantes, o número total de linhas e colunas do *dataframe* foi processado, mas se detectou uma linha duplicada, que foi excluída do *dataframe*. Demonstrou-se também uma variável com o tipo não suportado, a coluna "ementaTokenizada", que contém a lista de tokens de cada decisão, campos tipo lista não têm suporte no "Pandas Profiling", o que não representou problema.

**Figura 36 – Overview do "Pandas Profiling Report"**

```
#Análise exploratória de dados com pandas_profiling
pp.ProfileReport(df_DecisoesJud,n_extreme_obs=5, title='Análise exploratória de dados', explorative=True)
```

Overview		Reproduction	Warnings 7
Dataset statistics		Variable types	
Number of variables	6	CAT	2
Number of observations	1965	NUM	2
Missing cells	0	UNSUPPORTED	1
Missing cells (%)	0.0%	BOOL	1
Duplicate rows	1		
Duplicate rows (%)	0.1%		
Total size in memory	84.6 KiB		
Average record size in memory	44.1 B		

O "Pandas Profiling Report" gerou sete avisos, (figura 37), que foram analisados pontualmente.

1. A coluna duplicada foi excluída do *dataframe*.

<sup>15</sup> <https://pandas-profiling.github.io/pandas-profiling/docs/master/index.html>

2. A alta cardinalidade da coluna “ementa” é normal, haja vista serem valores únicos.
3. A alta cardinalidade da coluna “ementaProcessada” também é normal, todavia uma quantidade menor de valores únicos do que o tamanho do *dataset* chama a atenção e demonstra que há um padrão nas ementas das decisões no âmbito das turmas do TRF1, que após o pré-processamento, passaram a ser decisões idênticas.
4. A alta correlação entre o tamanho da ementa tokenizada, que é o tamanho da lista de *tokens*, e o tamanho da ementa processada, que é o tamanho da *string* do texto, demonstra que não houve distorções na tokenização do *corpus*.
5. Mesmas observações para a correlação entre a coluna “ementaProcessadaTam” e “ementaTokenizadaTam”.
6. A informação da uniformidade da ementa, que são os dados brutos que foram transformados, não agrega valor às análises, porque não fez parte do modelo.
7. O tipo da coluna “ementaTokenizada” sem suporte deve-se ao fato de ser um campo tipo lista, conforme análise do *overview*.

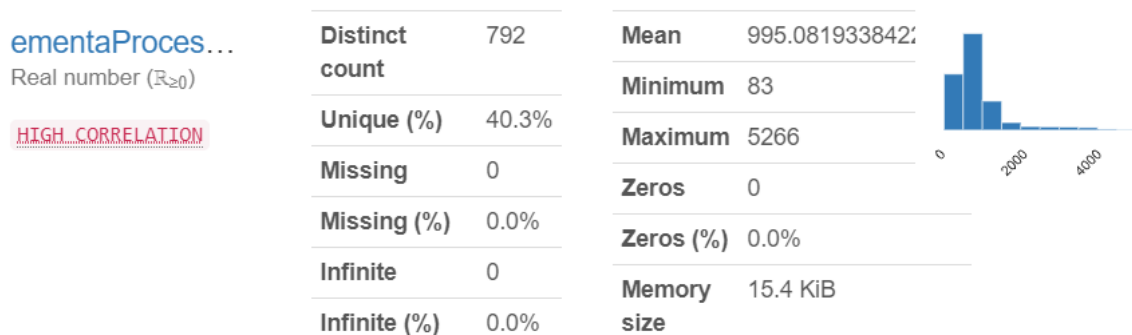
**Figura 37 – Warnings do “Pandas Profiling Report”**

Overview	Reproduction	Warnings 7
Warnings		
Dataset has 1 (0.1%) duplicate rows		Duplicates
ementa has a high cardinality: 1964 distinct values		High cardinality
ementaProcessada has a high cardinality: 1057 distinct values		High cardinality
ementaTokenizadaTam is highly correlated with ementaProcessadaTam		High correlation
ementaProcessadaTam is highly correlated with ementaTokenizadaTam		High correlation
ementa is uniformly distributed		Uniform
ementaTokenizada is an unsupported type, check if it needs cleaning or further analysis		Unsupported

Aprofundando-se na análise da coluna “ementaProcessada”, que representa o *corpus* modelado, observou-se (figura 38) um percentual de 40,3% de valores únicos, o que significa que, após o pré-processamento, grande parte das ementas

ficou igual. Conferindo-se os cálculos com os métodos “nunique” e “duplicated” (figura 39), os resultados foram semelhantes: 53,8% das ementas eram únicas, ou seja, após o pré-processamento, 46,2% das ementas das decisões tornaram-se iguais, sugerindo como promissora a tarefa de classificação, haja vista que a maior parte do *corpus* já está classificada, porque é igual. O desafio, portanto, foi classificar a outra metade do *corpus*, que era diferente.

**Figura 38 – Análise das ementas processadas**



**Figura 39 – Conferência de ementas não únicas**

```
#Verificando ementas processadas únicas
print('Shape: ' + str(df_DecisoesJud.shape))
print('Quant. Ementas processadas Únicas: ' + str(df_DecisoesJud['ementaProcessada'].nunique()))

#Verificando/confirmando ementas não únicas com o método duplicated
print('Ementas iguais pelo método duplicated: ' + str(any(df_DecisoesJud['ementaProcessada'].duplicated())))
```

Shape: (1964, 6)  
 Quant. Ementas processadas Únicas: 1056  
 Ementas iguais pelo método duplicated: True

Não houve dados faltantes nas ementas processadas nem valores zerados (figura 38); no entanto, a amplitude da distribuição ficou muito grande, com valor mínimo de 83 e máximo de 5.266, justificando aprofundar ainda as análises da coluna “ementaProcessada”.

Conforme a figura 40, a estatística descritiva das ementas processadas mostrou um desvio-padrão de 738,22, relativamente alto, haja vista que o coeficiente de variação é igual a 0,74, ou seja, o desvio-padrão representa 74% do valor da média. Aliado ao valor altíssimo da variância e o também elevado valor da amplitude, notou-se que a dispersão dos dados é grande, fato esse que deve ser investigado.

**Figura 40 – Estatística descritiva das ementas processadas**

Statistics   Histogram(s)   Common values   Extreme values			
Quantile statistics		Descriptive statistics	
Minimum	83	Standard deviation	738.2159405
5-th percentile	325.2	Coefficient of variation (CV)	0.7418644791
Q1	564	Kurtosis	7.926563383
median	824	Mean	995.0819338
Q3	1119	Median Absolute Deviation (MAD)	276
95-th percentile	2661.2	Skewness	2.6095794
Maximum	5266	Sum	1955336
Range	5183	Variance	544962.7749
Interquartile range (IQR)	555		

O método “describe” do Pandas para o tamanho da ementa processada (figura 41) chegou aos mesmos valores da estatística descritiva do “Pandas Profiling Report”, e o coeficiente de correlação de Pearson (figura 42), entre a ementa processada e a ementa tokenizada, igual a 0,999, correlação quase perfeita, demonstrou que não houve diferença significativa entre as duas colunas, ou melhor, tem-se praticamente a mesma informação descrita de formas diferentes.

**Figura 41 – Estatística descritiva método Pandas “describe”**

```
df_DecisooesJud['ementaProcessadaTam'].describe()
```

```
count    1965.000000
mean      995.081934
std       738.215941
min        83.000000
25%       564.000000
50%       824.000000
75%      1119.000000
max      5266.000000
Name: ementaProcessadaTam, dtype: float64
```

**Figura 42 – Coeficiente de correlação de Pearson**

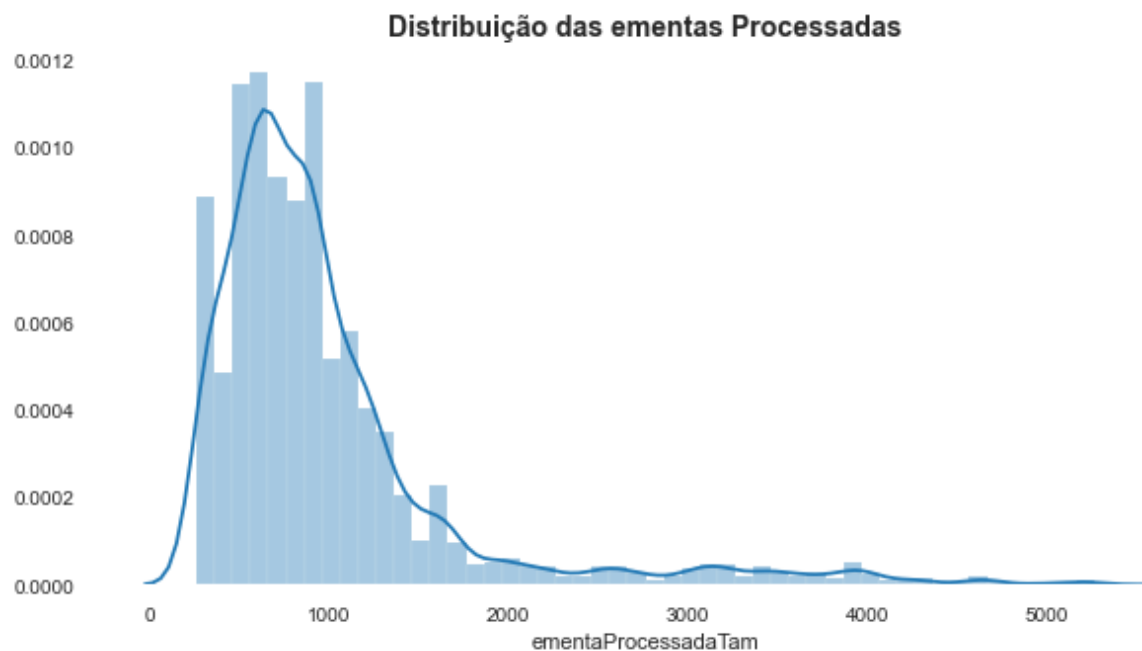
```
# Coeficiente de correlação de Pearson(ementa tokenizada e ementa processada)
#Não há diferença representativa entre as distribuições, Pearson = 0.999
coeficienteCorrPearson = df_DecisoJud['ementaTokenizadaTam'].corr(df_DecisoJud['ementaProcessadaTam'],
                                                                    method='pearson')

coeficienteCorrPearson

0.9996287535072291
```

Analisando-se o histograma da distribuição das ementas processadas, a distribuição aparenta não ser normal, com uma curva assimétrica positiva ou à direita, afunilada, concentrada e com uma cauda positiva extensa (figura 43), demonstrando que existem valores extremos, possivelmente outliers, que devem ser analisados. O valor da “kurtosis” da figura 40, 7,92, conferido pelo método “kurtosis” da biblioteca “Scipy” (figura 44), confirma a análise da figura 43. A título de ilustração, diz-se que a função de probabilidade é chamada leptocúrtica.

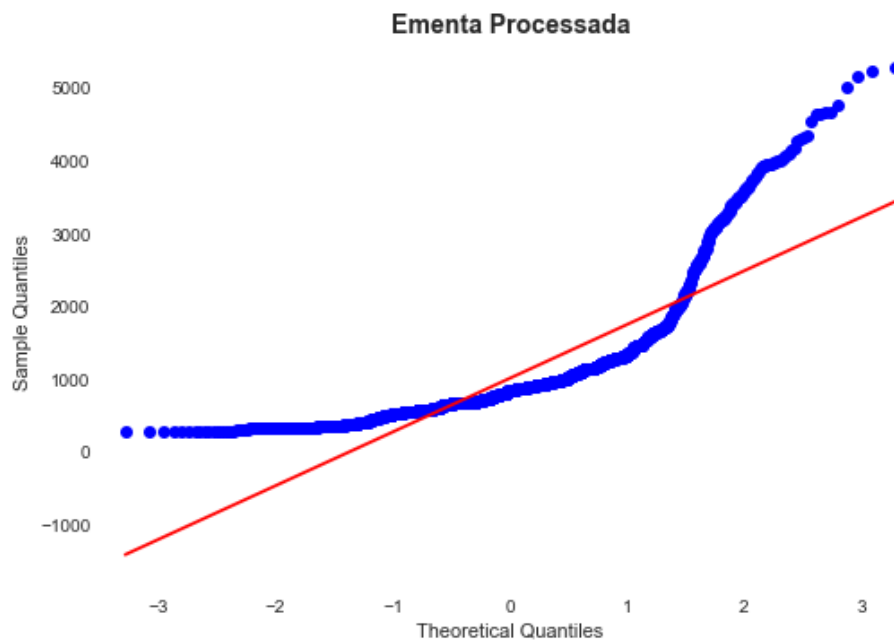
**Figura 43 – Histograma das ementas processadas**



Na figura 44, o gráfico Q-Q da biblioteca “Stats Models” confirmou que a distribuição não se ajusta à distribuição normal.



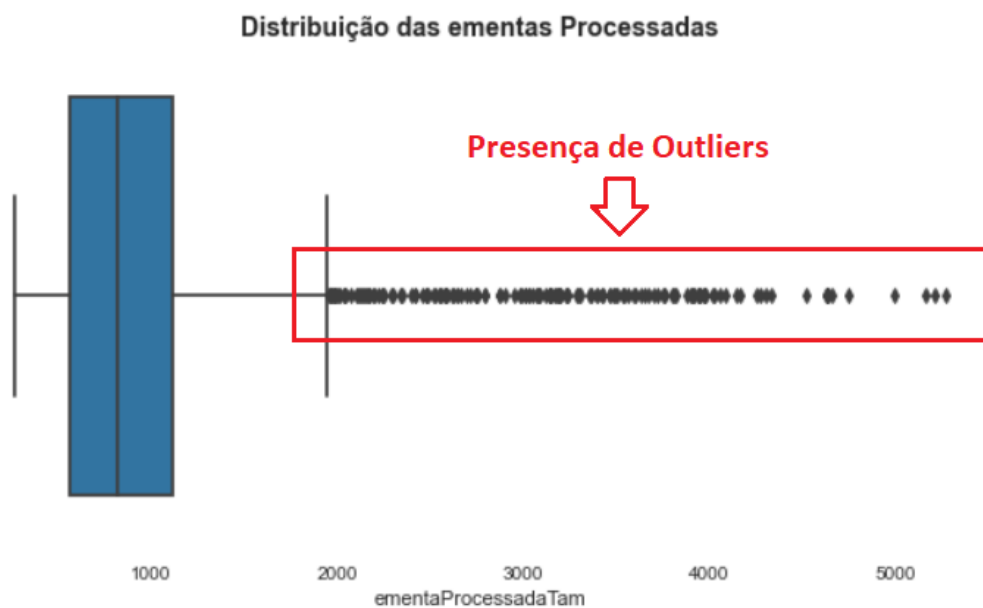
**Figura 44 – Gráfico Q-Q**



Com o objetivo de confirmar a existência de *outliers* foi elaborado o *boxplot* das ementas processadas (figura 45).

**Figura 45 – Boxplot das ementas processadas**

```
#Boxplot das ementas processadas
fig, ax = plt.subplots(figsize=(10,5))
sn.boxplot(x=df_DecisoesJud['ementaProcessadaTam'])
plt.title('Distribuição das ementas Processadas', fontsize=14, fontweight='bold')
plt.show()
```



Existem valores extremos (*outliers*) na distribuição, mas quais são esses valores? Para identificação desses valores foi utilizado o Intervalo-Interquartil (IIQ) ou Amplitude interquartil (figura 46), medida de dispersão estatística que é igual à diferença entre o 75º e o 25º percentis ou entre os quartis superior (3º quartil) e inferior (1º quartil). Os valores 1,5 vez acima ou abaixo do IIQ foram considerados *outliers*.

**Figura 46 – Identificando valores extremos com Intervalo-Interquartil**

```
#Identificando os outliers com Intervalo-Interquartil (IIQ) ou Amplitude interquartil
Q1 = df_DecisoõesJud['ementaTokenizadaTam'].quantile(0.25) #Primeiro Quartil
Q3 = df_DecisoõesJud['ementaTokenizadaTam'].quantile(0.75) #Terceiro Quartil
IIQ = Q3 - Q1 #amplitude interquartil
print('Primeiro Quartil:',Q1)
print('Terceiro Quartil:',Q3)
print('IIQ:',IIQ)

Primeiro Quartil: 86.0
Terceiro Quartil: 178.0
IIQ: 92.0

#Calculando Outliers Inferiores:
limiteOutliersInferiores = (Q1 - 1.5 * IIQ)
df_outliersInferiores = df_DecisoõesJud.query('ementaTokenizadaTam < '+ str(limiteOutliersInferiores))
df_outliersInferiores.shape

(0, 6)

#Calculando Outliers Superiores:
limiteOutliersSuperiores = (Q3 + 1.5 * IIQ)
df_outliersSuperiores = df_DecisoõesJud.query('ementaTokenizadaTam > '+ str(limiteOutliersSuperiores))
df_outliersSuperiores.shape

(142, 6)
```

Nenhum *outlier* abaixo do limite inferior ( $Q1 - 1,5 \text{ IIQ}$ ) foi identificado, todavia foram identificadas 142 linhas acima do limite superior ( $Q3 + 1,5 \text{ IIQ}$ ) com possíveis *outliers* (figura 47). Iterando-se sobre a coluna com as ementas (figura 48), bem como exportando-se os dados para uma planilha XLSX com o código `df_outliersSuperiores.to_excel(diretorioDadosExportados + 'df_outliersSuperiores.xlsx')`, verificou-se que 96,5% dos *outliers* referem-se ao mesmo relator da 7ª Turma do TRF1 e 3,5% referem-se a juízes federais convocados (figura 47), que adotaram uma forma mais extensa de descrever a ementa das decisões, com mais contextualização, mas não foi encontrado nenhum erro de coleta ou processamento de dados na coluna do *dataframe*. Considerando-se que, no processamento dos dados, os *tokens* e *n-grams* menos frequentes no *corpus* são excluídos, a extensão das ementas ou prolixidade torna-se irrelevante. Portanto, decidiu-se manter os “*pseudooutliers*” encontrados para reanálise no processamento dos dados. Vale

ressaltar que esses valores foram classificados como extremos, considerando-se o tamanho das ementas processadas.

**Figura 47 – Outliers identificados (5 primeiras linhas)**

```
df_outliersSuperiores.head()
```

	ementa	ementaProcessadaTam	ementaTokenizadaTam	ementaProcessada	ementaTokenizada	resultado_decisao
6	ypir\nAPELAÇÃO CÍVEL 000...	5206	802	apel civil process orig 40379620104013802 reyn...	[apel, civil, process, orig, 40379620104013802...	1
9	APELAÇÃO CÍVEL 0004400- 83.2010.4.01.3802/MG/r...	4747	743	tribut prescr contribu soc funr empreg rural p...	[tribut, prescr, contribu, soc, funr, empreg, ...	1
13	RELATOR \r\nDESEMBARGADOR FEDERAL REYNALDO FO...	3394	529	tribut constituc contribu soc funr empreg rura...	[tribut, constituc, contribu, soc, funr, empre...	1
14	APELAÇÃO/REEXAME NECESSÁRIO 0004401-68.2010.4....	2756	433	tribut prescr contribu soc funr empreg rural p...	[tribut, prescr, contribu, soc, funr, empreg, ...	1
54	APELAÇÃO/REEXAME NECESSÁRIO 0004123-67.2010.4....	2790	440	tribut contribu soc funr empreg rural pesso na...	[tribut, contribu, soc, funr, empreg, rural, p...	1

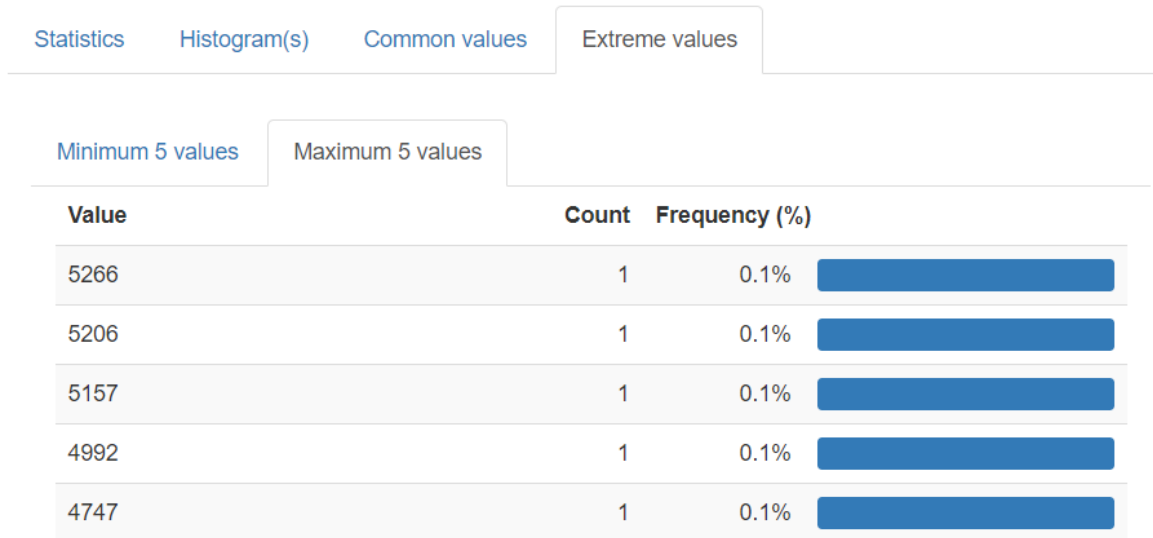
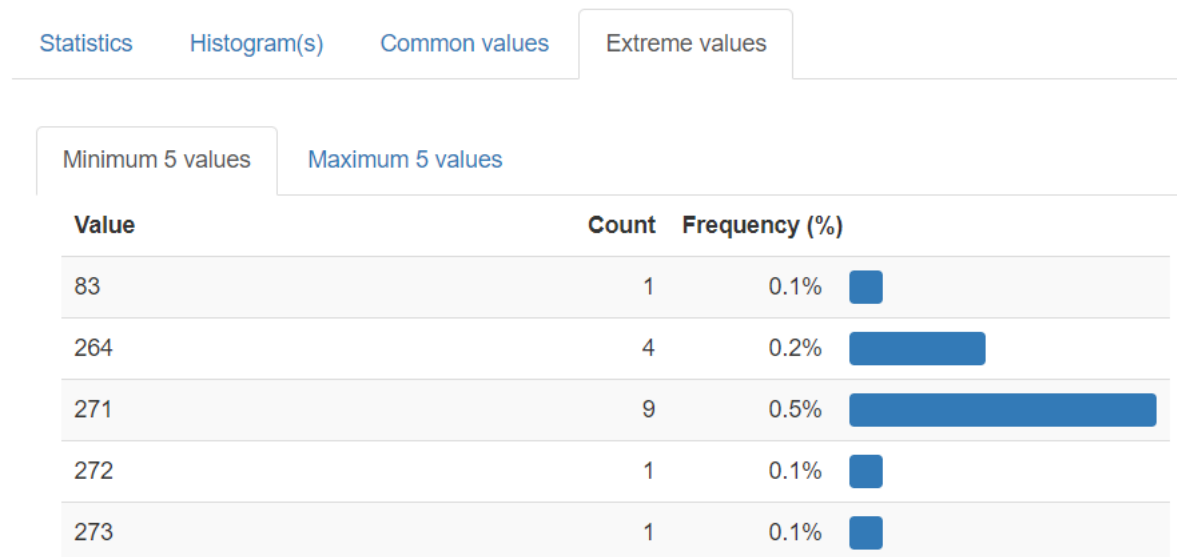
**Figura 48 – Percorrendo lista de ementas com outliers**

```
for outlier in df_outliersSuperiores['ementa']:
    print(outlier)
    input()
```

**Figura 49 – Distribuição de outliers por relator**

Relator	Contagem de ementa	%
DESEMBARGADOR FEDERAL REYNALDO FONSECA	137	96,5%
Juiz Federal ALEXANDRE BUCK MEDRADO SAMPAIO	1	0,7%
Juiz Federal CLODOMIR SEBASTIÃO REIS	3	2,1%
Juiz Federal ROBERTO VELOSO	1	0,7%
<b>Total Geral</b>	<b>142</b>	<b>100,0%</b>

Por fim, analisando-se os valores extremos máximos e mínimos encontrados pelo “Pandas Profiling”, (figuras 50 e 51), foi constatado que todos os valores máximos estão contidos na lista de outliers. Já a lista de valores mínimos não está contida na lista de outliers, porque não há nenhum valor abaixo de  $Q1 - 1,5 * IIQ$ . No entanto, o valor mínimo igual a 83 chama a atenção por ser bastante inferior aos demais. Identificando-se no *dataframe* o texto da decisão, trata-se de uma ementa inválida (figura 53), motivo pelo qual se decidiu retirar esse dado do *dataset*.

**Figura 50 – Valores extremos máximos no “Pandas Profiling”****Figura 51 – Valores extremos mínimos no “Pandas Profiling”****Figura 52 – Verificação dos valores máximos no ‘Pandas Profiling’**

```
#Verificando se os 5 valores extremos máximos do pandas profiling estão na lista de outliers
extremeValues_pandas_profiling_maximum5values = [5266, 5206, 5157, 4992, 4747]
[x for x in extremeValues_pandas_profiling_maximum5values
 if x not in df_outliersSuperiores['ementaProcessadaTam'].values]
```

```
[]
```

**Figura 53 – Identificação e remoção do valor mínimo**

<pre>#Identificação do valor mínimo df_DecisoesJud[df_DecisoesJud['ementaProcessadaTam']==83]</pre>						
	ementa	ementaProcessadaTam	ementaTokenizadaTam	ementaProcessada	ementaTokenizada	resultado_decisao
1071	RELATOR IRVDESEMBARGADOR FEDERAL ITALO FIOR...	83	15	edocx lot lot 3_1 process 2oof141v4 judici reg...	[edocx, lot, lot, 3_1, process, 2oof141v4, jud...	0
<pre>#Identificação do valor mínimo #Ementa inválida: retirar do dataframe df_DecisoesJud.iloc[df_DecisoesJud.index[df_DecisoesJud['ementaProcessadaTam']==83].values[0]]['ementaProcessada']  'edocx lot lot 3_1 process 2oof141v4 judici region reg apel civel lot 3_1 apel civel'</pre>						
<pre>#Remoção do valor extremo - valor mínimo df_DecisoesJud.drop(df_DecisoesJud.index[df_DecisoesJud['ementaProcessadaTam']==83].values[0],inplace=True)</pre>						
<pre>#Verificação df_DecisoesJud[df_DecisoesJud['ementaProcessadaTam']==83]</pre>						
	ementa	ementaProcessadaTam	ementaTokenizadaTam	ementaProcessada	ementaTokenizada	resultado_decisao
<pre>#Verificação df_DecisoesJud.iloc[df_DecisoesJud.index[df_DecisoesJud['ementaProcessadaTam']==83]].values  array([], shape=(0, 6), dtype=object)</pre>						

## 5. Criação de Modelos de *Machine Learning*

Segundo Géron (2019), *machine learning* é a ciência (e a arte) da programação de computadores para que eles possam aprender com os dados<sup>16</sup>. Mas há outras definições mais abrangentes, como o campo de estudo que dá aos computadores a habilidade de aprender sem ser explicitamente programado, de Arthur Samuel (1959).

Em relação à possibilidade de serem ou não treinados com supervisão humana, os sistemas de *machine learning* podem ser classificados em: supervisionados, não supervisionados, semissupervisionados e aprendizado por reforço, podendo ser combinados com outras classificações.

A classificação, objetivo do atual projeto, é uma das tarefas típicas do aprendizado supervisionado, em que os dados de treinamento fornecidos ao algoritmo encontram-se rotulados, ou seja, contêm as soluções desejadas. O *dataset* utilizado contém 1.965 ementas baixadas do site do TRF1, sendo 1.061 decisões nas quais foi declarada a constitucionalidade e 904 decisões nas quais foi declarada a inconstitucionalidade.

<sup>16</sup> Géron, Aurélien. Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow (p. 4). Edição do Kindle.

Tal tarefa de classificação de textos insere-se no contexto de NLP em português, que é, ao mesmo tempo, um subcampo da linguística e da ciência da computação que visa a permitir que os computadores entendam a linguagem falada e escrita de uma forma “natural”, como os humanos fazem<sup>17</sup>. É também descrita como um subcampo da inteligência artificial que tenta processar e analisar dados de linguagem natural<sup>18</sup>, mais especificamente um subcampo de *machine learning* ou *deep learning*, conforme o caso, que utiliza métodos matemáticos e estatísticos para transformar o texto em números, vetorizando o *corpus* do texto para a máquina aprender mediante experiência (treinamento), compreender a linguagem e, a partir disso, gerar análises, *insights*, classificações, agrupamentos etc.

A vetorização dos textos pode ser realizada pela ocorrência ou não dos termos (*tokens*) em cada texto, gerando-se um campo booleano; pela frequência dos termos nos textos, gerando-se um campo inteiro; ou por TF-IDF, a frequência dos termos ponderada pela sua importância no documento. Optou-se pelo método TF-IDF, considerando-se o seu maior poder de classificação/discriminação, haja vista que representa o quão importante é um determinado termo para cada texto, não se considerando apenas a frequência do termo, mas ponderando, também, se a palavra aparece em todos ou em muitos documentos.

TF-IDF é um método estatístico utilizado para avaliar a importância de uma palavra em determinado documento e é definido pelo produto da frequência de termos simples (*term frequency*) pelo inverso da frequência nos documentos (*inverse document frequency*).

TF é a frequência que um termo/palavra  $t$  aparece em um documento  $d$ , dividida pelo número total de palavras no documento:

$$tf(t, d) = \frac{\text{Frequência do termo } t \text{ no documento } d}{\text{Total de palavras no documento}}.$$

IDF é a pontuação de quão esporádico é o termo/palavra no documento, calculada pelo log da divisão do número total de vezes que o termo  $t$  aparece dividido pelo número de documentos com o termo  $t$ :

$$idf(t, d) = \log \left( \frac{\text{Quant.de vezes que o termo } t \text{ aparece}}{\text{Nº de documentos que contém o termo } t}} \right).$$

<sup>17</sup> Beysolow II, Taweh (2018-09-11). Applied Natural Language Processing with Python. Edição do Kindle.

<sup>18</sup> Vasiliev, Yuli (2020-04-27T22:58:59). Natural Language Processing with Python and spaCy . No Starch Press. Edição do Kindle.

Por fim, o TF-IDF( $t,d$ ) é calculado pelo produto da TF pelo IDF:

$$TFIDF(t, d) = tf(t, d) * idf(t, d).$$

Pois bem, os textos já se encontram pré-processados e tokenizados na coluna “ementaTokenizada”, contendo as unidades linguísticas identificadas em cada decisão. Conforme Beysolow II (2018), a forma mais simples de se codificarem os dados de textos é através do algoritmo BoW (bag-of-words), que consiste simplesmente em determinar o número de vezes que uma determinada palavra está presente no corpo do texto<sup>19</sup>, o que é também a base para o cálculo do TF-IDF. A ordem das palavras é totalmente ignorada e o resultado é um vetor cujo comprimento é igual ao número de palavras no vocabulário e cujos elementos são o número de vezes que cada palavra ocorre no documento. Esse esquema pode ser estendido para codificar uma quantidade limitada de dependência ao contar frases únicas em vez de únicas palavras. Uma frase de comprimento  $n$  é referida como um *n-gram*<sup>20</sup>. Por exemplo: “constituc formal mater contribu soc empreg rural” representa o *n-gram-7* do tema 669 do STF, com repercussão geral: “*é constitucional formal e materialmente a contribuição social do empregador rural pessoa física, instituída pela Lei 10.256/01*”, presente em algumas ementas cuja decisão foi pela constitucionalidade.

Utilizando-se os métodos “CountVectorizer” e “TfidfVectorizer” da biblioteca “Scikit-Learn” (figura 54), para aplicação dos algoritmos dos modelos BoW e TF-IDF (figuras 55 e 56), foram geradas as correspondentes matrizes esparsas, com um *range* de *n-grams* de 1 a 7. Optou-se por não limitar a aplicação dos modelos para não se perder informação e trabalhar a redução da dimensionalidade na fase seguinte em camadas, do que resultou uma matriz com a mesma quantidade de linhas do *dataframe* (1.964) e 114.547 colunas, representando o vocabulário aprendido (figura 57). E a frequência dos *n-grams* do vocabulário na figura 56.

<sup>19</sup> Beysolow II, Taweh (2018-09-11). Applied Natural Language Processing with Python. Edição do Kindle.

<sup>20</sup> GENTZKOW, Matthew & KELLY, Bryan & TADDY, Matt: Text as Data, Journal of Economic Literature, 57(3), 535–574, 2019, disponível em <<https://web.stanford.edu/~gentzkow/research/text-as-data.pdf>>. Acesso em: 5 out 2020.

**Figura 54 – Aplicação dos modelos BoW e TF-IDF**

```
# BoW e TF/IDF
corpus = df_DecisoesJud['ementaProcessada']
ngram_range = (1,7)

vetorizador = CountVectorizer(ngram_range=ngram_range)
matrizEsparsa = vetorizador.fit_transform(corpus)
tokens = vetorizador.get_feature_names()
df_matrizEsparsa = pd.DataFrame(matrizEsparsa.toarray(), columns=tokens)

vetorizador_tfidf = TfidfVectorizer(ngram_range=ngram_range)
matrizEsparsa_tfidf = vetorizador_tfidf.fit_transform(corpus)
df_matrizEsparsa_tfidf = pd.DataFrame(matrizEsparsa_tfidf.toarray(), columns=tokens)

df_vocabulario = pd.DataFrame(vetorizador.vocabulary_, index=['vocabulario']).T.sort_values(by='vocabu

#Frequencia de cada palavra/token em cada sentença
df_matrizEsparsa.shape

(1964, 114547)
```

**Figura 55 – Matriz esparsa BoW**

#Matriz esparsa BoW: Frequência de cada palavra/token em cada sentença  
df\_matrizEsparsa

01082011	01082011 plen	01082011 plen public	01082011 plen public dje165	01082011 plen public dje165 divulg	01082011 plen public dje165 divulg 26082011	01082011 plen public dje165 divulg 26082011 public	01122008	01122008 agrg	01122008 agrg resp	...	zavask sec pet inic impetr	zavask sec pet inic impetr formul	zavask sec pet inic impetr formul sucessiv	zel	zel profiss	prof pn
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1959	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1960	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1961	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1962	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1963	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

1964 rows × 114547 columns



**Figura 56 – Matriz esparsa TF-IDF**

```
#Matriz esparsa TF-IDF
df_matrizEsparsa_tfidf
```

	01082011	01082011 plen	01082011 plen public	01082011 plen public dje165	01082011 plen public dje165 divulg	01082011 plen public dje165 divulg 26082011	01082011 plen public dje165 divulg 26082011 public	01122008	01122008 agrg	01122008 agrg resp	...	zavask sec pet inic impetr	zavask sec pet inic impetr formul	zavask sec pet inic impetr formul sucessiv	zel	zel profiss	prof pr
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1959	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
1960	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
1961	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
1962	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
1963	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0

1964 rows × 114547 columns

**Figura 57 – Vocabulário aprendido**

```
df_vocabulario #Vocabulário aprendido com o correspondente índice
```

	vocabulario
zel profiss prest servic natur import caus	114546
zel profiss prest servic natur import	114545
zel profiss prest servic natur	114544
zel profiss prest servic	114543
zel profiss prest	114542
...	...
01082011 plen public dje165 divulg	4
01082011 plen public dje165	3
01082011 plen public	2
01082011 plen	1
01082011	0

114547 rows × 1 columns

**Figura 58 – Frequência dos *n*-grams**

```
df_ngrams_frequencia = pd.DataFrame({'frequencia': [df_matrizEsparsa[coluna].sum() for coluna in df_matrizEsparsa.columns],
                                     'ngrams': vetorizador.get_feature_names()}).sort_values(by='frequencia', ascending=False)
df_ngrams_frequencia.to_excel(diretorioDadosExportados + 'df_ngrams.xlsx')
df_ngrams_frequencia
```

	frequencia	ngrams
	28683	6643 contribu
	100108	6642 rural
	15051	6536 art
	85633	4436 prov
	11523	4406 apel
...	...	...
	61632	1 l8212 91 mant mesm fat
	61633	1 l8212 91 mant mesm fat ger
	61634	1 l8212 91 mant mesm fat ger aliquot
	61638	1 l8212 91 mater estranh lid apel
	70192	1 nao reiter apel nao conheç apel

114547 rows × 2 columns

Gerando-se a nuvem de palavras para verificação dos termos mais frequentes (figura 59), destacam-se os principais temas desse tipo de ação judicial e da jurisprudência do assunto, como: empreg rural pesso físic (empregador rural pessoa física: sujeito passivo da obrigação tributária), produt rural (produtor rural: tipo de contribuinte), contribu soc (contribuição social: nome da contribuição objeto da ação judicial), incid receipt (incidente sobre a receita: fato gerador da contribuição), L10256 (Lei nº 10.256/2001, que constitucionalizou a contribuição), L8212 (Lei nº 8.212/91: previsão legal da contribuição), prov apel (provimento da apelação), juiz retrat (juízo de retratação após o julgamento do RE 718874), civil repercuss (repercussão geral conforme Código Civil), constituc formal mater contribu (constitucional formal e material a contribuição) e bitribut ofens (ofensa ao princípio da bitributação). Naturalmente, as palavras maiores na nuvem são termos mais frequentes nos textos, o que, para o presente projeto, para o tipo de classificação que se pretende, não tem poder de discriminação e, portanto, foram excluídas nas camadas de redução da dimensionalidade a seguir.



Principais (PCA), o mais popular e mais utilizado, além de Kernel PCA, Locally Linear Embedding (LLE)<sup>22</sup>, Escalonamento Multidimensional (MDS, em inglês), Isomap, t-Distributed Stochastic Neighbor Embedding (t-SNE), Análise Discriminante Linear (LDA, em inglês) etc. No entanto, todos esses algoritmos demandam grande poder de processamento quando temos uma matriz com centenas de milhares de termos, o que inviabilizaria a sua execução no contexto do presente projeto, além de tornar a sua *interpretabilidade* muito mais complexa e dificultar a aplicação em outros modelos. Portanto, optou-se por uma solução mais simples para redução do número de termos sem prejudicar o treinamento, que foi um grande desafio e demandou muitas análises até o modelo final de redução de dimensionalidade em camadas, conforme descrito abaixo.

1ª camada de redução de dimensionalidade: remoção dos *n-grams* muito frequentes nos documentos, muito comuns e, portanto, pouco representativos, bem como a remoção dos *n-grams* pouco frequentes, muito raros, com baixa capacidade de discriminação.

Conforme a lei de Zipf, do linguista americano George Kingsley Zipf (1902–1950), segundo a qual a frequência das palavras em um documento é inversamente proporcional ao seu *rank*, combinada com as argumentações posteriores de Luhn<sup>23</sup>, segundo quem as palavras muito frequentes e as pouco frequentes não colaboram para a discriminação e similaridade entre documentos, justificam-se os pontos de corte inferior e superior na distribuição com o intuito de simplificar o modelo sem prejudicar o seu poder de discriminação. No entanto, há certa arbitrariedade na determinação dos pontos de corte, os quais são definidos por tentativa e erro e dependem da linguagem e do estilo linguístico, conforme Gelbukh e Sidorov (2001)<sup>24</sup>. No presente projeto, após inúmeras simulações empíricas, optou-se por estabelecer o limite inferior da frequência dos termos em 2% do tamanho do *corpus* para os termos muito raros e o limite superior em 200% do tamanho do *corpus* para os termos muito comuns, armazenando-se os termos removidos em uma lista de “StopNgrams” (figura 60), reduzindo-se o tamanho da lista de termos para 10,2% do tamanho inicial, de 114.547 *n-grams* para 11.715 *n-grams*.

<sup>22</sup> “Nonlinear Dimensionality Reduction by Locally Linear Embedding”, S. Roweis, L. Saul (2000).

<sup>23</sup> LUHN, H.P., 'The automatic creation of literature abstracts', IBM Journal of Research and Development, 2, 159-165 (1958).

<sup>24</sup> A. Gelbukh, G. Sidorov, International Conference on Intelligent Text Processing and Computational Linguistics, páginas 332-335, Springer, Berlin, Heidelberg

**Figura 60 – Camada 1 de redução da dimensionalidade: Remoção de extremos**

```
#Redução da dimensionalidade - 1ª camada - removendo extremos: limite superior e limite inferior
#Remover ngrams muito frequentes, alta frequência no corpus, ngrams muito comuns, pouco representativos.
#Remover ngrams pouco frequentes, ngrams muito raros, baixa capacidade de discriminação dos objetos.
#Criando lista de N-grams removidos

limiteInferior = (2/100) * len(df_DecisoesJud) # 2% do corpus (número total de decisões len(df_DecisoesJud) )
limiteSuperior = (200/100) * len(df_DecisoesJud) # 200%

df_ngramLista = df_ngrams_frequencia[(df_ngrams_frequencia['frequencia'] < limiteSuperior)
& (df_ngrams_frequencia["frequencia"] > limiteInferior)]
stopNgrams = [gram for gram in df_ngrams_frequencia['ngrams'] if gram not in df_ngramLista.values]

df_ngramLista.to_excel(diretorioDadosExportados + 'df_ngramLista.xlsx')

print('Tamanho da lista StopNgrams: ' + str(len(stopNgrams)))
df_ngramLista.shape

Tamanho da lista StopNgrams: 102832
(11715, 2)
```

2ª camada de redução de dimensionalidade: remoção dos *n-grams* contidos em outros *n-grams* de mesma frequência, indicando que são partes da mesma sentença. Por exemplo, a sentença “*é constitucional formal e materialmente a contribuição social do empregador rural pessoa física, instituída pela Lei 10.256/01*”, mesmo após o pré-processamento, produz dezenas de *n-grams* (range: 1-7) que representam a mesma frase e têm a mesma frequência, sendo possível remover os *n-grams* com menor *range* sem perder a informação. Após a segunda camada de redução de dimensionalidade, o tamanho da lista de termos foi reduzido para 3.213 *n-grams* (figura 61).

**Figura 61 – Camada 2 de redução da dimensionalidade: Remoção de termos da mesma sentença**

```
#Redução da dimensionalidade/multicolinearidade - 2ª camada
#Retirando os N-Grams já contidos em outros N-Grams com a mesma frequência, termos da mesma sentença
frequencia = None
for linha in range(len(df_ngramLista)):
    consulta = ('frequencia == ' + str(df_ngramLista.iloc[linha]['frequencia']))
    grupoFrequencias = df_ngramLista.query(consulta) #Agrupando n-grams com a mesma frequência no corpus
    if len(grupoFrequencias) > 1 and frequencia != grupoFrequencias['frequencia'].iloc[0]:
        frequencia = grupoFrequencias['frequencia'].iloc[0]
        #Criando string com o join dos ngrams da lista da coluna ngrams do Dataframe grupoFrequencias
        ngrams_lista = ' '.join([ngram for ngram in grupoFrequencias['ngrams']])

        for ngram in grupoFrequencias['ngrams']:
            #Criando string sem o ngram da vez
            ngrams_lista2 = ngrams_lista[:ngrams_lista.index(ngram)] + ngrams_lista[ngrams_lista.index(ngram)+len(ngram):]
            if ngram in ngrams_lista2 and ngram not in stopNgrams: stopNgrams.append(ngram)
print('Tamanho da lista StopNgrams: ' + str(len(stopNgrams)))
df_ngramLista = df_ngramLista.loc[~df_ngramLista['ngrams'].isin(stopNgrams)]
df_ngramLista.shape

Tamanho da lista StopNgrams: 111334
(3213, 2)
```

3ª camada de redução de dimensionalidade: remoção dos *n-grams* com baixa correlação com o resultado da decisão, ou seja, no modelo não teriam grande capacidade de discriminação para contribuir com a classificação dos documentos.

Utilizou-se como parâmetro o R2 entre o *n-gram* e o resultado da decisão, coluna “resultado\_decisao” do *dataframe* “df\_DecisoesJud”, calculado pela potência do método “corrcoef” do “Numpy”. O ponto de corte foi uma correlação de 0,30, R2 igual a 0,09, ou seja, uma correlação muito baixa. Após a terceira camada de redução de dimensionalidade, o tamanho da lista de termos foi reduzido para 694 *n-grams* (figura 62).

**Figura 62 – Camada 3 de redução da dimensionalidade: Remoção de termos com baixa correlação com o resultado da decisão**

```
#Redução da dimensionalidade/multicolinearidade - 3ª camada
#Verificação de correlação dos Ngrams com o resultado_decisao,
#mantendo somente Ngrams altamente correlacionados com resultado_decisao

listaCorrelacao = []
df_R2 = pd.DataFrame(columns=['ngram', 'R2', 'correlacao'])

contador = 0
for ngram in df_ngramLista['ngrams']:
    coeficienteCorrPearson = df_matrizEsparsa_tfidf[ngram].corr(df_DecisoesJud['resultado_decisao'], method='pearson')
    #Coeficiente de determinação R2
    R2 = np.corrcoef(df_matrizEsparsa_tfidf[ngram], df_DecisoesJud['resultado_decisao'])[0, 1]**2
    print('ngram: ', ngram, ' R2: ', R2, ' : Pearson', coeficienteCorrPearson)
    #Comparação pelo R2 pode ser apenas positiva, se fosse pela correlação deveria abranger correlação positiva e negativa.
    if R2 < 0.09: #Se R2 baixo, pouco poder de determinação, incluir stopNgrams.
        #Incluindo valores encontrados no dataframe 'df_correlacao' para controle
        df_R2.loc[len(df_R2)] = [ngram, R2, coeficienteCorrPearson]
        if ngram not in stopNgrams: stopNgrams.append(ngram)
        print('\nBaixo Coeficiente de Determinação\n#####\n')
    print('.....\n')
    contador += 1
    #if contador > 10: break
    print(contador)

df_ngramLista = df_ngramLista.loc[~df_ngramLista['ngrams'].isin(stopNgrams)]
print('stopNgrams = ', len(stopNgrams))
df_ngramLista.shape

3211
ngram: process civil contribu previdenciar produc rural / R2: 0.021533307541761422 : Pearson 0.08069444658127638

Baixo Coeficiente de Determinação
#####

.....

3212
ngram: manifest / R2: 0.007881455097659836 : Pearson 0.06252270327316496

Baixo Coeficiente de Determinação
#####

.....

3213
stopNgrams = 113853

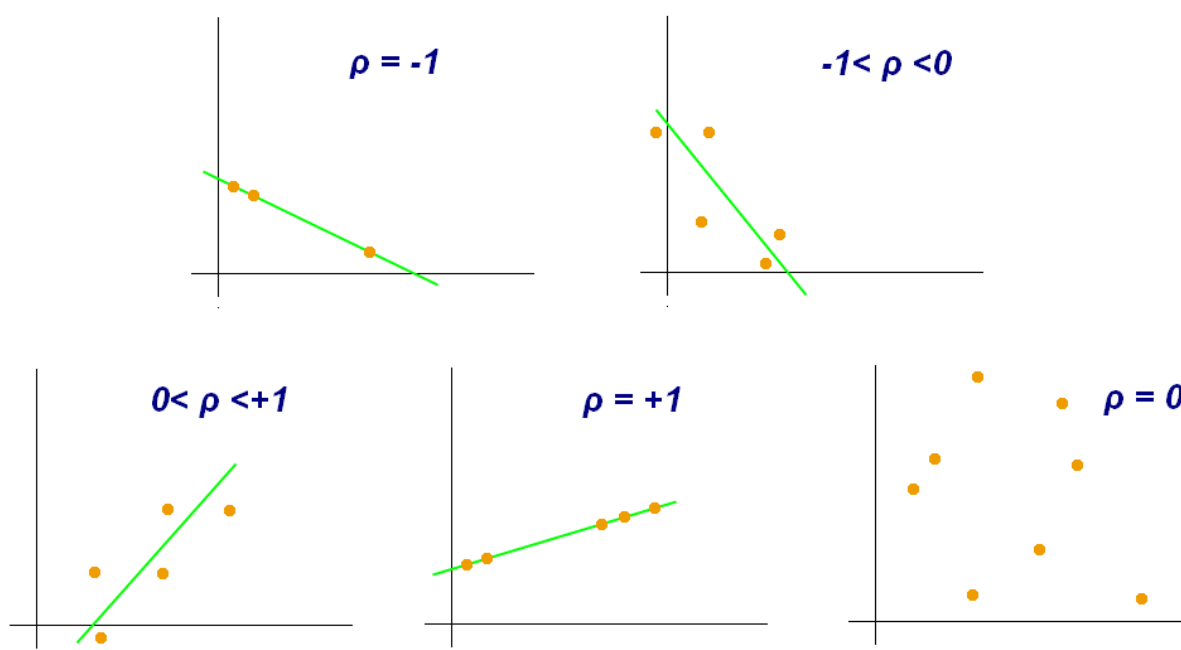
(694, 2)
```

4ª camada de redução de dimensionalidade: remoção dos *n-grams* altamente correlacionados entre si, mantendo-se o *n-gram* com maior correlação com o resultado da decisão.

Quando o *dataset* não é muito grande, pode-se calcular facilmente o coeficiente de correlação padrão, também chamado r de Pearson ou coeficiente de correlação de Pearson, entre cada par de *n-grams* utilizando-se o método “corr” do Pandas. O coeficiente de correlação varia de -1 a 1: próximo de 1, significa que

existe uma forte correlação positiva, ou seja, os valores são diretamente proporcionais; próximo de -1, significa que existe uma forte correlação negativa, ou seja, os valores são inversamente proporcionais. Finalmente, coeficientes próximos de zero significam que não há correlação linear<sup>25</sup>, conforme demonstrado nos diagramas de dispersão da figura 63.

**Figura 63 – Exemplos de diagramas de dispersão com diferentes valores de coeficiente de correlação ( $\rho$ )**



Fonte: Wikipédia, figura de domínio público.

Foram utilizados como parâmetros o índice de correlação de Pearson maior que 0,70, para identificar a alta correlação entre os *n-grams*, e o coeficiente de determinação  $R^2$ , para remoção do *n-grams* com menor poder de determinação do resultado da decisão, coluna “resultado\_decisao” do *dataframe* “df\_DecisoõesJud”, calculado pela potência do método “corrcoef” do “Numpy”. Após a quarta camada de redução de dimensionalidade, o tamanho da lista de termos foi reduzido para apenas 50 *n-grams* (figura 64), que permitirá um rápido processamento dos algoritmos de classificação, não sendo necessário implementar novas camadas de redução de dimensionalidade.

<sup>25</sup> Géron, Aurélien. *Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow* (p. 58). Edição do Kindle.



**Figura 64 – Camada 4 de redução da dimensionalidade: Remoção de termos altamente correlacionados entre si**

```
#Redução da dimensionalidade/multicolinearidade - 4ª camada
#Verificação de correlação dos Ngrams entre si

listaCorrelacao = []
stopNgrams4Camada = []
df_correlacao = pd.DataFrame(columns=['ngram1', 'ngram2', 'correlacao', 'R2_Ngram1', 'R2_Ngram2'])

contador = 1

for ngram1 in df_ngramLista['ngrams']:
    print('Contador: ', contador)
    for ngram2 in df_ngramLista['ngrams']:
        if ngram1 != ngram2 and ngram1+ngram2 not in listaCorrelacao and ngram2+ngram1 not in listaCorrelacao:
            coeficienteCorrPearson = df_matrizEsparsa_tfidf[ngram1].corr(df_matrizEsparsa_tfidf[ngram2], method='pearson')
            if coeficienteCorrPearson > 0.7: #Se Correlação muito alta, incluir stopNgrams altamente correlacionados
                #Coeficiente de determinação R2 Ngram1
                R2_Ngram1 = np.corrcoef(df_matrizEsparsa_tfidf[ngram1], df_DecisoesJud['resultado_decisao'])[0, 1]**2
                #Coeficiente de determinação R2 Ngram2
                R2_Ngram2 = np.corrcoef(df_matrizEsparsa_tfidf[ngram2], df_DecisoesJud['resultado_decisao'])[0, 1]**2
                df_correlacao.loc[len(df_correlacao)] = [ngram1, ngram2, coeficienteCorrPearson, R2_Ngram1, R2_Ngram2]
                if R2_Ngram1 > R2_Ngram2:
                    #Incluindo na lista stopNgrams, Ngram que tiver menor coeficiente de determinação com o resultado_decisao.
                    stopNgrams.append(ngram1)
                    stopNgrams4Camada.append(ngram1)
                else:
                    stopNgrams.append(ngram2)
                    stopNgrams4Camada.append(ngram2)
            else:
                stopNgrams.append(ngram2)
                stopNgrams4Camada.append(ngram2)
        contador += 1

df_correlacao.to_excel(diretorioDadosExportados + 'df_correlacao2.xlsx')

df_ngramLista = df_ngramLista.loc[~df_ngramLista['ngrams'].isin(stopNgrams)]
print('stopNgrams = ', len(stopNgrams))
df_ngramLista.shape
```

Tendo sido identificados os *n-grams* que devem ser removidos do *dataset* para redução da dimensionalidade e resolução do problema da multicolinearidade, foi criado um novo *dataframe*, “df\_matrizEsparsa\_tfidf\_final”, com a matriz esparsa TF-IDF dos 50 termos/tokens/*n-grams* que serão utilizados para treinamento e teste pelos algoritmos de classificação (figura 65).

**Figura 65 – Matriz Esparsa final TF-IDF**

```
stopNgrams = sorted(set(stopNgrams))
df_matrizEsparsa_tfidf_final = df_matrizEsparsa_tfidf.drop(stopNgrams, axis=1)
df_matrizEsparsa_tfidf_final
```

	apreci	art cpc apel	art l8540 92	art l8540 92 inconstituc 110256 201998 nao	assent jurisprud cort mer discuss judic repet	comerci produc rural produt	compens	contribu previdenciar comerci	decid	decid turn unanim juiz retrat prov apel	...	sucumb	susp
0	0.024658	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.024381	0.000000	...	0.0	0.000
1	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.017101	0.000000	...	0.0	0.000
2	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.011764	0.000000	...	0.0	0.000
3	0.000000	0.0	0.012682	0.0	0.0	0.0	0.0	0.0	0.015775	0.000000	...	0.0	0.000
4	0.024118	0.0	0.019171	0.0	0.0	0.0	0.0	0.0	0.011923	0.000000	...	0.0	0.000
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1959	0.022670	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.011207	0.028673	...	0.0	0.000
1960	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	...	0.0	0.000
1961	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	...	0.0	0.000
1962	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.008833	0.000000	...	0.0	0.021
1963	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.007374	0.000000	...	0.0	0.000

1964 rows × 50 columns



Analizando-se a matriz de correlação gerada pela biblioteca “Pandas”<sup>26</sup>, com o método Pearson, do TF-IDF da lista final de *n-grams* (figura 66), bem como o gráfico da figura 67, verificou-se que o problema da multicolinearidade foi tratado, haja vista que não restou nenhum par de TF-IDF altamente correlacionado que estaria representado no gráfico na cor azul. Tem-se, portanto, um *dataset* “df\_matrizEsparsa\_tfidf\_final”, preparado para treinamento.

**Figura 66 – Matriz de correlação TF-IDF**

```
#Correlação Pandas, método Pearson
#https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html

matrizCorrelacao = df_matrizEsparsa_tfidf_final.corr(method='pearson')
matrizCorrelacao
```

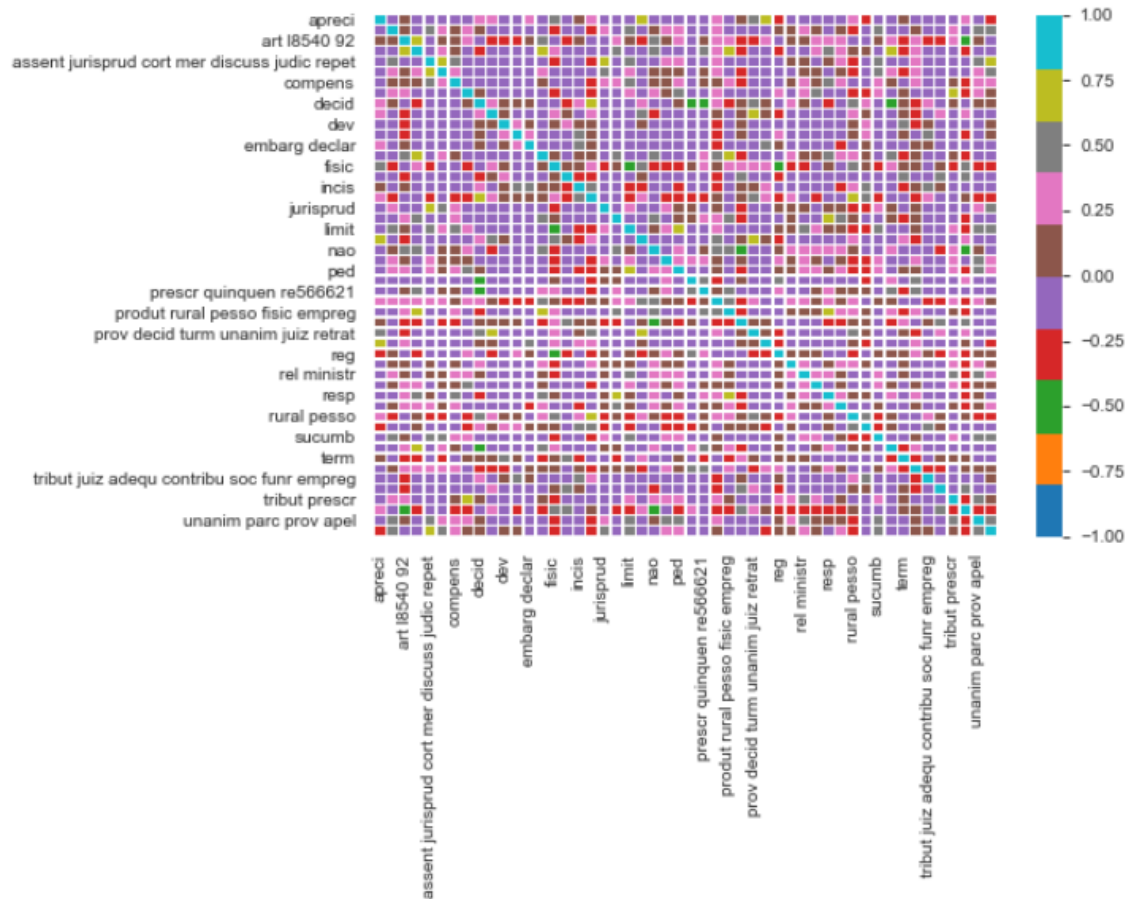
	apreci	art cpc apel	art l8540 92	art l8540 92 inconstituc l10256 201998 nao	assent jurisprud cort mer discuss judic repet	comerci produc rural produt	compens	contribu previdenciar comerci	decid	decid turn unanim juiz retrat prov apel
apreci	1.000000	-0.147688	0.003376	-0.153123	-0.108732	-0.143909	-0.137757	-0.127652	0.360447	0.318
art cpc apel	-0.147688	1.000000	0.050855	-0.095128	0.521066	0.385149	0.182469	0.282789	0.031199	-0.102
art l8540 92	0.003376	0.050855	1.000000	0.634365	-0.137678	0.194481	0.100065	0.340060	-0.115334	-0.213
art l8540 92 inconstituc l10256 201998 nao	-0.153123	-0.095128	0.634365	1.000000	-0.090165	0.269140	0.179771	-0.084594	-0.366957	-0.106
assent jurisprud cort mer discuss judic repet	-0.108732	0.521066	-0.137678	-0.090165	1.000000	0.661507	0.443056	-0.087260	-0.015101	-0.091
comerci produc rural produt	-0.143909	0.385149	0.194481	0.269140	0.661507	1.000000	0.259045	-0.074288	-0.181297	-0.100
compens	-0.137757	0.182469	0.100065	0.179771	0.443056	0.259045	1.000000	-0.066127	-0.164610	-0.128
contribu previdenciar comerci	-0.127652	0.282789	0.340060	-0.084594	-0.087260	-0.074288	-0.066127	1.000000	0.085203	-0.103
decid	0.360447	0.031199	-0.115334	-0.366957	-0.015101	-0.181297	-0.164610	0.085203	1.000000	0.319
decid turn unanim juiz retrat prov apel	0.318574	-0.102196	-0.213671	-0.106841	-0.091807	-0.100650	-0.128499	-0.103398	0.319550	1.000

<sup>26</sup> <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html>

**Figura 67 – Gráfico da matriz de correlação TF-IDF**

```
# Mapa de cores qualitativo seaborn e matplotlib
#https://matplotlib.org/examples/color/colormaps_reference.html

sn.heatmap(matrizCorrelacao, vmin=-1, vmax=1, center=0, cmap='tab10', linewidths=0.4)
plt.figure(figsize=(16,9))
plt.show()
```



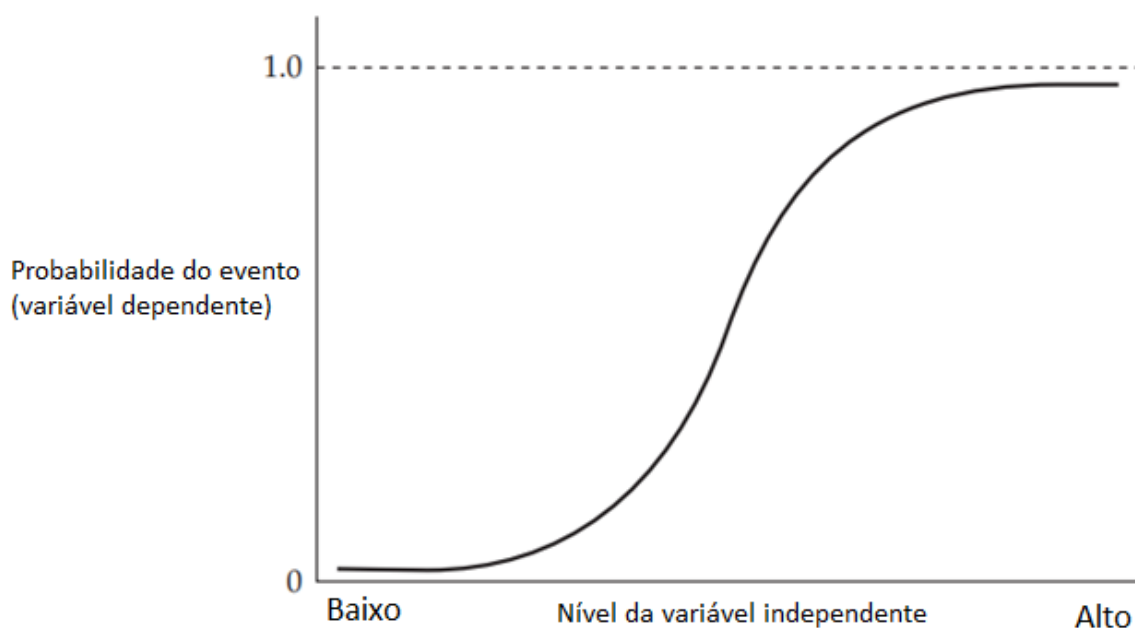
<Figure size 1152x648 with 0 Axes>

Optou-se por comparar o desempenho de sete algoritmos de *machine learning* para classificação:

1. Logistic Regression (RL): Regressão Logística;
2. Linear Discriminant Analysis (AD): Análise Discriminante;
3. k-Nearest Neighbors (k-NN): k-Vizinhos Mais Próximos;
4. Decision Tree Classifier (CART): Árvore de Decisão;
5. Naive Bayes (NB);
6. Support Vector Machines (SVM): Máquinas de Vetores de Suporte; e
7. Random Forest Classifier (RFC): Floresta Aleatória.

A Regressão Logística, também chamada de Regressão Logit, é comumente utilizada para estimar a probabilidade de uma instância pertencer a uma determinada classe (por exemplo, a probabilidade de determinada decisão ter sido pela constitucionalidade). Se a probabilidade estimada for maior que 50%, então o modelo prevê que a instância pertence a essa classe (chamada de classe positiva, rotulada como 1), ou então ela prevê que não (isto é, pertence à classe negativa, rotulada 0). Isso a transforma em um classificador binário<sup>27</sup>, gerando uma curva chamada sigmoide (em forma de S) (figura 68). Por conseguinte, a regressão logística é limitada a prever apenas uma medida dependente de dois grupos (HAIR, 2009), uma classificação dicotômica, por exemplo, constitucional ou inconstitucional, que serve bem ao presente projeto.

**Figura 68 – Curva sigmoide da Regressão Logística**



Fonte: Adaptado de Hair (2009)<sup>28</sup>

A Análise Discriminante envolve determinar uma variável estatística discriminante, a combinação linear das duas (ou mais) variáveis independentes que melhor discriminam entre os objetos (pessoas, documentos, decisões etc.) nos grupos definidos *a priori*. A discriminação é conseguida estabelecendo-se os pesos

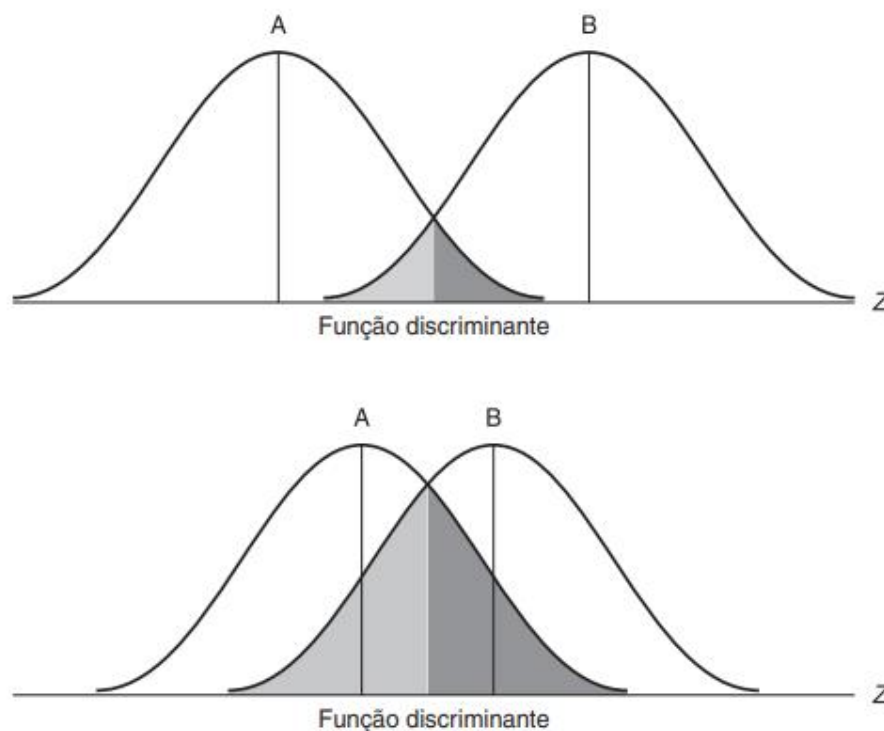
<sup>27</sup> Géron, Aurélien. *Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow* (p. 139). Edição do Kindle.

<sup>28</sup> Hair, Joseph F., William C. Black, Barry J. Babin, e Ronald L. Tatham. 2009. *Análise Multivariada de Dados*. 6a ed. São Paulo: Bookman.

da variável estatística para cada variável independente, para maximizar as diferenças entre os grupos (i.e., a variância entre grupos relativa à variância interna no grupo) – variável estatística para uma análise discriminante, também conhecida como a função discriminante.<sup>29</sup>

A classificação é obtida pelo escore determinante, a soma dos valores obtidos pela multiplicação de cada variável independente por seu peso discriminante, somados ao intercepto, valor fixo semelhante às funções de regressões. Na figura 69 estão representadas duas distribuições de escores discriminantes, uma com pequena área de sobreposição e a outra com grande área de sobreposição – quanto menor a sobreposição, melhor a classificação entre os grupos. A área de sobreposição sombreada representa os casos em que pode acontecer uma classificação ruim. A significância estatística da função discriminante é a medida da distância entre os centroides dos grupos A e B da figura 69 – quanto maior a distância, mais significativa é a função discriminante e melhor a classificação entre os grupos.

**Figura 69 – Representação dos escores Z determinantes**

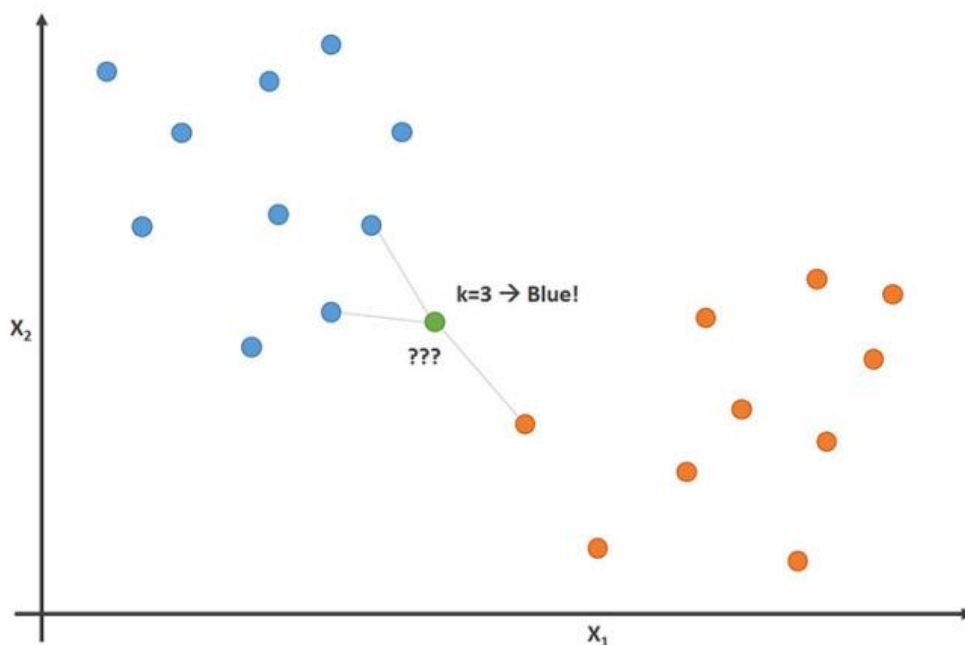


Fonte: Hair (2009).

<sup>29</sup> Hair, Joseph F., William C. Black, Barry J. Babin, e Ronald L. Tatham. 2009. *Análise Multivariada de Dados*. 6a ed. São Paulo: Bookman.

K-Vizinhos Mais Próximos é um dos mais simples algoritmos de aprendizagem baseados em instância. Segue o princípio de que as instâncias dentro de um conjunto de dados geralmente existem próximas a outras instâncias que possuem propriedades semelhantes. Se as instâncias são marcadas com um rótulo de classificação, então o valor do rótulo de uma instância não classificada pode ser determinado observando-se a classe de seus vizinhos mais próximos. O k-NN localiza as k instâncias mais próximas da instância da consulta e determina sua classe, identificando o único rótulo de classe mais frequente<sup>30</sup>, conforme a figura 70.

**Figura 70 – Representação da classificação do algoritmo k-Vizinhos Mais Próximos**



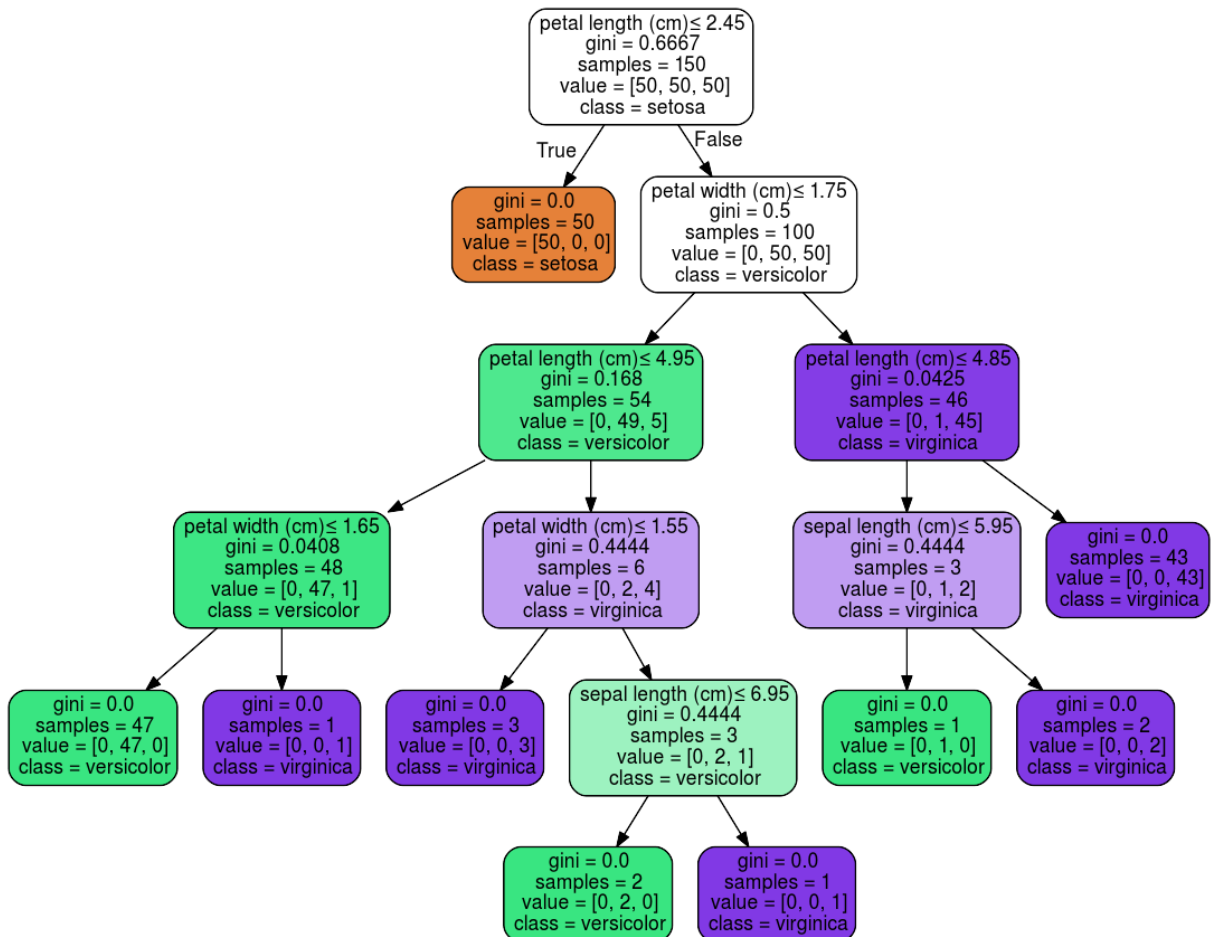
Fonte: <https://rapidminer.com/blog/k-nearest-neighbors-laziest-machine-learning-technique/>

Árvores de Decisão é uma tabela de decisão sob a forma de árvore com nós e folhas sequenciais e interligados que classificam as instâncias, ordenando-as com base nos valores dos recursos. Cada nó em uma árvore de decisão representa uma característica em uma instância a ser classificada, e cada ramo representa um valor que o nó pode assumir. As instâncias são classificadas começando no nó raiz com

<sup>30</sup> KOTSIANTIS, Sotiris B.; ZAHARAKIS, Ioannis D.; PINTELAS, Panayiotis E. Machine learning: a review of classification and combining techniques. Artificial Intelligence Review, v. 26, n. 3, p.159-190, 2006.

base em seus valores de recursos.<sup>31</sup> Trata-se de um dos modelos mais práticos e mais utilizados em inferência por indução (representação visual na figura 71).

**Figura 71 – Representação visual do classificador Árvores de Decisão**

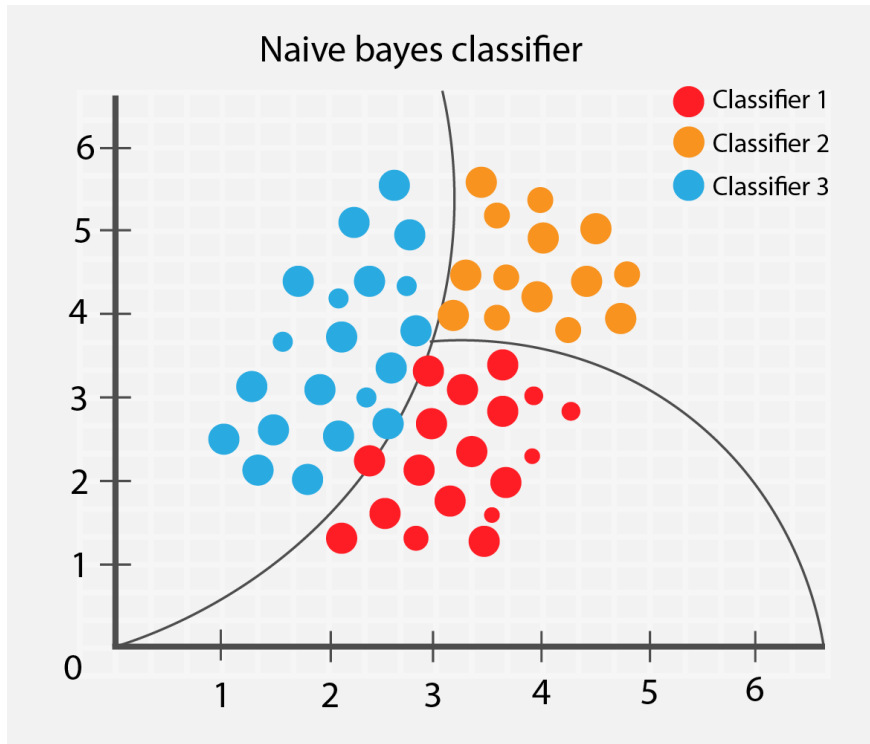


Fonte: <https://scikit-learn.org/stable/modules/tree.html>

Naive Bayes é um conjunto de algoritmos com base na aplicação do teorema de Bayes com a suposição "ingênua" (*naive*), na verdade uma forte suposição de independência condicional entre os nós filhos no contexto de seu pai. São redes muito simples com apenas um pai (representando o nó não observado) e vários filhos (correspondendo aos nós observados).

<sup>31</sup> KOTSIANTIS, Sotiris B.; ZAHARAKIS, Ioannis D.; PINTELAS, Panayiotis E. Machine learning: a review of classification and combining techniques. Artificial Intelligence Review, v. 26, n. 3, p.159-190, 2006.

**Figura 72 – Representação de classificação com Naive Bayes**



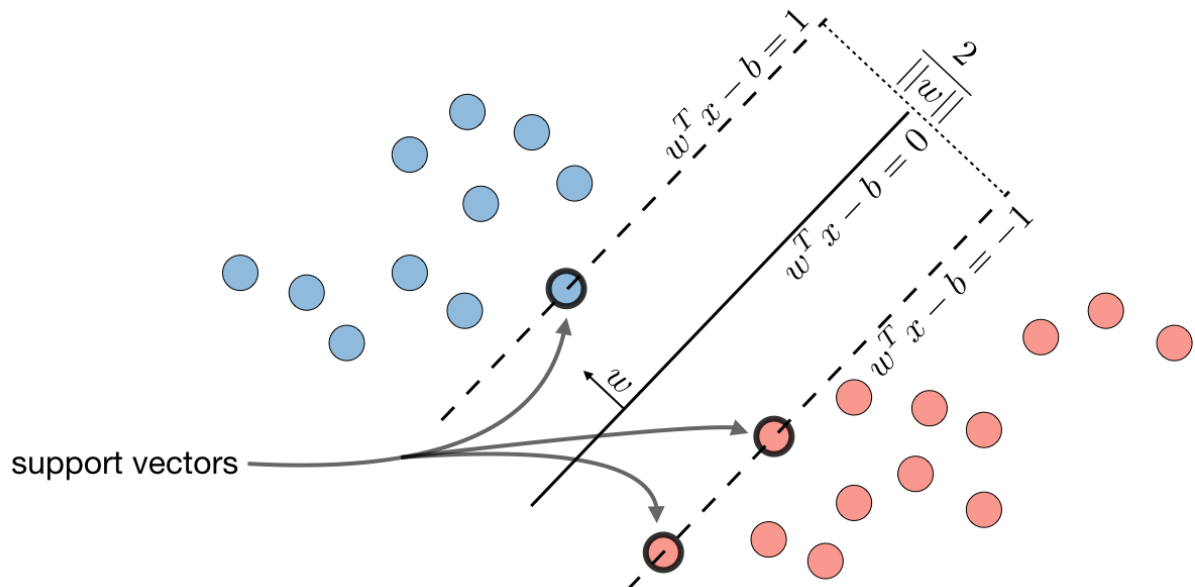
Fonte: <https://towardsdatascience.com/introduction-to-naive-bayes-classifier-fa59e3e24aaf>

As Support Vector Machines, ou Máquinas de Vetores de Suporte em português, estão entre as técnicas mais recentes de aprendizado supervisionado. As últimas pesquisas giram em torno da noção de uma "margem", ou seja, qualquer um dos lados de um hiperplano que separa duas classes de dados, maximizando a margem e criando, assim, a maior distância possível entre o hiperplano de separação.<sup>32</sup>

Na prática, significa encontrar o "hiperplano", uma linha de separação entre os dados de duas classes, buscando maximizar a distância entre os pontos mais próximos em relação a cada uma das classes ou classificações, conforme representação gráfica da figura 73.

<sup>32</sup> KOTSIANTIS, Sotiris B.; ZAHARAKIS, Ioannis D.; PINTELAS, Panayiotis E. Machine learning: a review of classification and combining techniques. *Artificial Intelligence Review*, v. 26, n. 3, p.159-190, 2006.

**Figura 73 – Representação de classificação com Máquinas de Vetores de Suporte**



Fonte: <https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-supervised-learning>

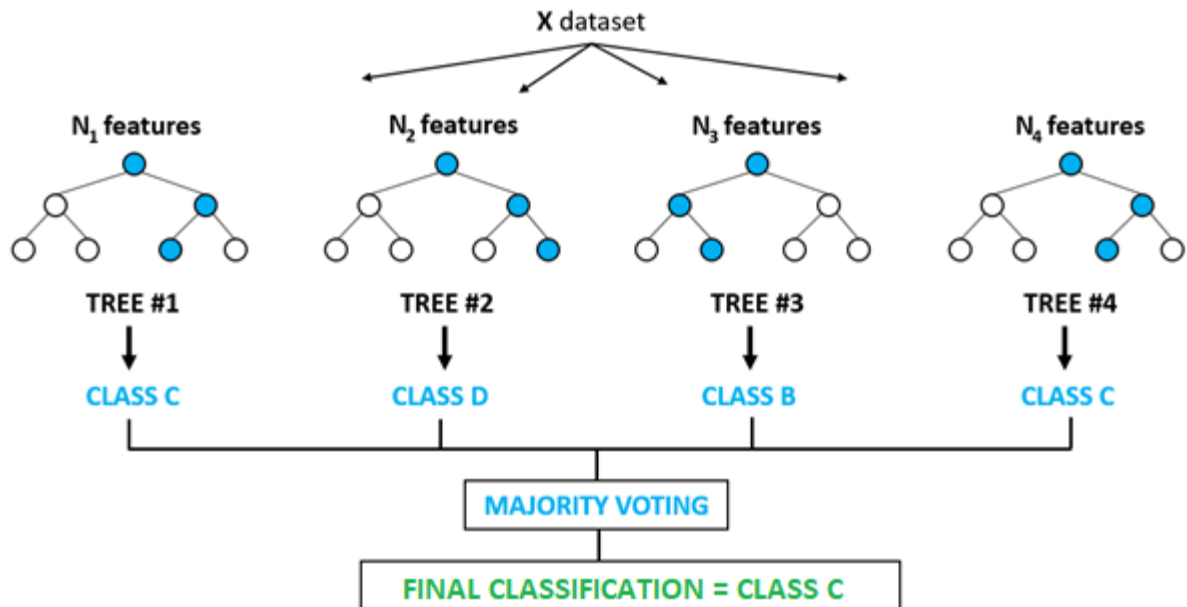
Random Forest Classifier, ou Floresta Aleatória, é um *ensemble*, um conjunto de Árvores de Decisão que, apesar da sua simplicidade, é um dos mais poderosos algoritmos de aprendizado de máquina disponíveis atualmente. Baseado na chamada “sabedoria das multidões”, em que a resposta agregada de milhares de pessoas aleatórias a uma pergunta complexa é melhor do que a resposta de um especialista.

Na prática, significa treinar um conjunto de classificadores de árvores de decisão, cada um em um subconjunto aleatório diferente do conjunto de treinamento, e fazer as previsões, obtendo-as de todas as árvores individuais, e, então, prever a classe que obtém a maioria dos votos.<sup>33</sup> No exemplo da figura 74 vemos um *ensemble* de árvores de decisão, em que a Class C teve o maior número de resultados nas árvores individuais, sendo, portanto classificada como resultado final.

<sup>33</sup> Géron, Aurélien. Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow (p. 185). Edição do Kindle.



**Figura 74 – Representação de Floresta Aleatória**



Fonte: adaptado de <https://www.kaggle.com/getting-started/176257>

Para a comparação dos algoritmos, foi escolhida a biblioteca “Scikit-Learn”, e, para a criação dos objetos (figura 75), os métodos “LogisticRegression”, “LinearDiscriminantAnalysis”, “KNeighborsClassifier”, “DecisionTreeClassifier”, “GaussianNB”, “SVC” e “RandomForestClassifier”.

**Figura 75 – Objetos dos algoritmos**

```

#Criando lista de objetos dos modelos que serão comparados e siglas para boxplot
modelos = []
modelos.append(('Regressão Logística', LogisticRegression()))
modelos.append(('Análise Discriminante', LinearDiscriminantAnalysis()))
modelos.append(('K-Nearest Neighbours', KNeighborsClassifier()))
modelos.append(('Árvores de Decisão', DecisionTreeClassifier()))
modelos.append(('Naive Bayes', GaussianNB()))
modelos.append(('Máquinas de Vetores de Suporte', SVC()))
modelos.append(('Florestas Aleatórias', RandomForestClassifier()))

modelosSiglasDic = {'Regressão Logística': 'RL',
                    'Análise Discriminante': 'AD',
                    'K-Nearest Neighbours': 'KNC',
                    'Árvores de Decisão': 'CART',
                    'Naive Bayes': 'NB',
                    'Máquinas de Vetores de Suporte': 'SVM',
                    'Florestas Aleatórias': 'RFC'}
siglasModelosBoxplot = [modelosSiglasDic[x] for x in modelosSiglasDic]

print(siglasModelosBoxplot)
modelos
  
```

Para avaliação dos algoritmos, utilizam-se vários métodos, tais como “F-measure”, “K-fold”, “Hold-out”, “Leave-One-Out” e “Receiver Operating Characteristics”. No presente trabalho, optou-se pela validação cruzada do método “K-fold”, que utiliza todas as amostras disponíveis como amostras de treinamento e teste<sup>34</sup>, conseguindo chegar a resultados mais precisos em comparação com outros métodos, como “Hold-out” e “Leave-One-Out”, que ainda exigem um maior poder computacional para processamento.

Foi definido  $k=10$ , ou seja, o *dataset* foi dividido em 10 subconjuntos de tamanhos iguais, cada subconjunto é utilizado para validação (teste) do modelo, e os conjuntos restantes são utilizados para treinamento. Esse processo foi repetido 10 vezes, porque  $k=10$ , uma vez para cada subconjunto criado, com o objetivo de maximizar a confiabilidade do classificador. O resultado final da validação “K-fold” é a média dos resultados encontrados. No entanto, pode-se calcular outras medidas para avaliação, tais como desvio-padrão, coeficiente de variação, valor mínimo e valor máximo. Foram utilizados os seguintes parâmetros de pontuação da validação “K-fold”<sup>35</sup>:

1. “accuracy”: a pontuação da acurácia do modelo representa o quanto o conjunto de rótulos previsto para uma amostra corresponde exatamente ao conjunto correspondente de rótulos em  $y\_true$ ;
  2. “precision”: a precisão é a proporção  $tp / (tp + fp)$ , em que  $tp$  é o número de verdadeiros positivos, e  $fp$ , o número de falsos positivos. A precisão é, intuitivamente, a capacidade do classificador de não rotular como positiva uma amostra negativa. O melhor valor é 1, e o pior, 0;
  3. “recall”: é a razão  $tp / (tp + fn)$ , em que  $tp$  é o número de verdadeiros positivos, e  $fn$ , o número de falsos negativos. O “recall” é, intuitivamente, a capacidade do classificador de encontrar todas as amostras positivas. O melhor valor é 1, e o pior, 0;
  4. “f1”: a pontuação f1 pode ser interpretada como uma média ponderada de “precision” e “recall”, em que uma pontuação f1 atinge seu melhor valor em 1 e o pior em 0; e
- $$f1 = 2 * (precisão * recuperação) / (precisão + recuperação)$$

<sup>34</sup> DUCHESNE, Pierre; RÉMILLARD, Bruno (Ed.). Statistical modeling and analysis for complex data problems. Springer Science & Business Media, 2005.

<sup>35</sup> [https://scikit-learn.org/stable/modules/model\\_evaluation.html](https://scikit-learn.org/stable/modules/model_evaluation.html)

5. “jaccard”: índice de Jaccard, ou coeficiente de similaridade de Jaccard, definido como o tamanho da interseção dividido pelo tamanho da união de dois conjuntos de rótulos, é usado para comparar o conjunto de rótulos previstos para uma amostra com o conjunto correspondente de rótulos em `y_true`.

Os resultados obtidos através do código da figura 76 foram incluídos em um novo *dataframe*, “`df_avaliacaoAlgoritmos`” (figura 77), ordenado pela medida em primeiro nível e por média da medida em segundo nível, em ordem decrescente. Em todas as medidas, exceto “precision”, o algoritmo florestas aleatórias teve o melhor desempenho, apesar de os resultados estarem muito próximos e todos com valores muito altos, acurácia média mínima de 0,96 (regressão logística) e máxima de 0,99 (florestas aleatórias), bem como a baixa dispersão dos resultados, revelada pelos valores do coeficiente de variação com valores muito baixos em todos os modelos e medidas, denotam um excelente desempenho dos modelos de classificação.

O *dataset* utilizado de fato contribuiu para o alto desempenho dos algoritmos por se tratar de decisões judiciais da Justiça Federal de 2º grau, em que determinados temas têm um entendimento jurídico definido em cada turma e, por via de regra, seguindo-se a jurisprudência dos tribunais superiores, ou seja, há um padrão linguístico nas decisões que foi vetorizado permitindo à máquina compreender e classificar os textos. O pré-processamento, realizado em várias camadas, também contribuiu muito para o resultado. Sem essa etapa bem concluída, seria impossível classificar os documentos. O pior desempenho dos modelos em quase todas as medidas, novamente exceto “precision”, foi a regressão logística, revelando ser a melhor medida para identificar falsos positivos.

**Figura 76 – Código para avaliação dos algoritmos**

```
modelosSiglasDic = {'Regressão Logística': 'RL',
                   'Análise Discriminante': 'AD',
                   'K-Nearest Neighbours': 'KNC',
                   'Árvores de Decisão': 'CART',
                   'Naive Bayes': 'NB',
                   'Máquinas de Vetores de Suporte': 'SVM',
                   'Florestas Aleatórias': 'RFC'}

medidaAvaliacaoLista = ['accuracy', 'f1', 'precision', 'recall', 'jaccard']

df_avaliacaoAlgoritmos = pd.DataFrame(columns=['Medida', 'Algoritmo', 'Sigla', 'Média', 'Desvio-Padrão',
                                              'Coeficiente de variação', 'Valor Mínimo', 'Valor Máximo'])

for medidaAvaliacao in medidaAvaliacaoLista:
    resultados = []
    modelosLista = []

    for modeloNome, modelo in modelos:
        metodo_CrossValidator = model_selection.KFold(n_splits=10, shuffle=False, random_state=None)
        CrossValidator_resultados = model_selection.cross_val_score(modelo, X, y, cv=metodo_CrossValidator,
                                                                    scoring=medidaAvaliacao)

        resultados.append(CrossValidator_resultados)
        modelosLista.append(modeloNome)
        df_avaliacaoAlgoritmos.loc[len(df_avaliacaoAlgoritmos)] = [medidaAvaliacao, modeloNome,
                                                                    modelosSiglasDic[modeloNome],
                                                                    CrossValidator_resultados.mean(),
                                                                    CrossValidator_resultados.std(),
                                                                    CrossValidator_resultados.std() / CrossValidator_resultados.mean(),
                                                                    CrossValidator_resultados.min(), CrossValidator_resultados.max()]

    df_avaliacaoAlgoritmos.sort_values(['Medida', 'Média'], ascending=False)
```

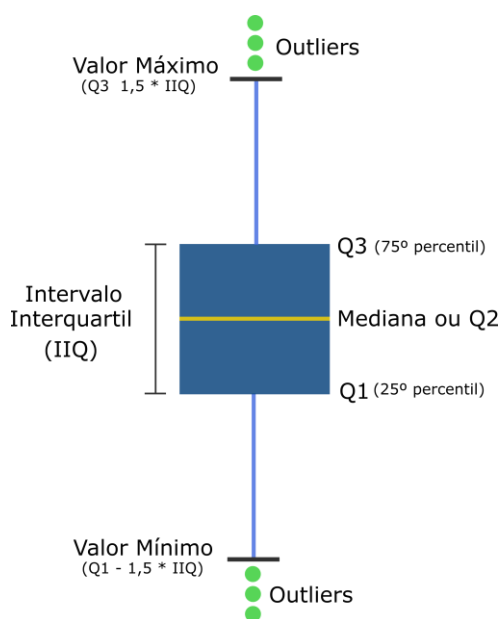
**Figura 77 – Medidas de avaliação dos algoritmos**

	Medida	Algoritmo	Sigla	Média	Desvio-Padrão	Coeficiente de variação	Valor Mínimo	Valor Máximo
27	recall	Florestas Aleatórias	RFC	0.995546	0.007191	0.007224	0.979167	1.000000
23	recall	K-Nearest Neighbours	KNC	0.991877	0.012081	0.012180	0.962963	1.000000
26	recall	Máquinas de Vetores de Suporte	SVM	0.990804	0.014568	0.014703	0.962963	1.000000
24	recall	Árvores de Decisão	CART	0.979156	0.015081	0.015402	0.949495	1.000000
22	recall	Análise Discriminante	AD	0.978784	0.012204	0.012469	0.962963	1.000000
25	recall	Naive Bayes	NB	0.956470	0.022962	0.024007	0.917647	1.000000
21	recall	Regressão Logística	RL	0.927426	0.035999	0.038816	0.885417	1.000000
19	precision	Máquinas de Vetores de Suporte	SVM	0.992204	0.006802	0.006856	0.980198	1.000000
14	precision	Regressão Logística	RL	0.991670	0.010572	0.010661	0.974359	1.000000
20	precision	Florestas Aleatórias	RFC	0.991162	0.008021	0.008092	0.977778	1.000000
16	precision	K-Nearest Neighbours	KNC	0.990963	0.008505	0.008583	0.975000	1.000000
15	precision	Análise Discriminante	AD	0.989883	0.005914	0.005975	0.978495	1.000000
17	precision	Árvores de Decisão	CART	0.983532	0.011737	0.011933	0.960000	1.000000
18	precision	Naive Bayes	NB	0.977641	0.013658	0.013971	0.954023	1.000000
34	jaccard	Florestas Aleatórias	RFC	0.988155	0.010913	0.011044	0.969072	1.000000
33	jaccard	Máquinas de Vetores de Suporte	SVM	0.983177	0.016668	0.016953	0.951220	1.000000
30	jaccard	K-Nearest Neighbours	KNC	0.983111	0.018186	0.018498	0.939759	1.000000
29	jaccard	Análise Discriminante	AD	0.969119	0.014330	0.014786	0.951220	0.989691
31	jaccard	Árvores de Decisão	CART	0.965259	0.019573	0.020278	0.931373	1.000000
32	jaccard	Naive Bayes	NB	0.935980	0.025384	0.027121	0.902174	0.977273
28	jaccard	Regressão Logística	RL	0.920451	0.039283	0.042678	0.873563	1.000000
13	f1	Florestas Aleatórias	RFC	0.992754	0.005023	0.005059	0.987654	1.000000
12	f1	Máquinas de Vetores de Suporte	SVM	0.991445	0.008540	0.008613	0.975000	1.000000
9	f1	K-Nearest Neighbours	KNC	0.991398	0.009355	0.009436	0.968944	1.000000
8	f1	Análise Discriminante	AD	0.984264	0.007374	0.007492	0.975000	0.994819
10	f1	Árvores de Decisão	CART	0.979530	0.012681	0.012946	0.954315	0.994536
11	f1	Naive Bayes	NB	0.966754	0.013540	0.014006	0.948571	0.988506
7	f1	Regressão Logística	RL	0.958147	0.021065	0.021986	0.932515	1.000000
6	accuracy	Florestas Aleatórias	RFC	0.993896	0.004434	0.004461	0.989796	1.000000
5	accuracy	Máquinas de Vetores de Suporte	SVM	0.992368	0.007287	0.007343	0.979592	1.000000
2	accuracy	K-Nearest Neighbours	KNC	0.992360	0.007980	0.008042	0.974490	1.000000
1	accuracy	Análise Discriminante	AD	0.985748	0.006346	0.006438	0.979592	0.994924
3	accuracy	Árvores de Decisão	CART	0.982689	0.013527	0.013765	0.948980	0.994924
4	accuracy	Naive Bayes	NB	0.969960	0.011474	0.011829	0.954315	0.989848
0	accuracy	Regressão Logística	RL	0.962820	0.018354	0.019063	0.943878	1.000000

Aprofundando-se um pouco mais na análise dos resultados da acurácia dos modelos, com o gráfico *boxplot* (figura 78), que permite a visualização dos *outliers* (valores extremos) acima do valor máximo e abaixo do valor mínimo, calculados a partir do intervalo interquartil (IIQ), 3º quartil, abaixo do qual estão 75%, 1º quartil, abaixo do qual estão 25%, mediana ou 2º quartil que dividem a distribuição ao meio, foi possível observar (figura 79) que não houve *outliers* em nenhum modelo. No entanto, as distribuições divergiram em alguns pontos, especialmente no modelo da regressão logística, que teve a maior diferença entre os valores máximo e mínimo, bem como entre o Q1 e Q3, e Q3 e valor máximo, confirmando ser o modelo com o menor desempenho, apesar de ser um resultado muito bom com acurácia mínima de 0,91 na validação cruzada “K-fold”.

Os melhores desempenhos foram os de K-Vizinhos Mais Próximos, Máquinas de Vetores de Suporte e Floresta Aleatória, que tiveram, dentre os modelos avaliados, as maiores médias e medianas, bem como os maiores valores mínimos e máximos, sendo Floresta Aleatória o de melhor resultado, confirmando a tendência atual de ser um dos mais poderosos algoritmos de aprendizado de máquina disponíveis atualmente.

**Figura 78 – Boxplot**



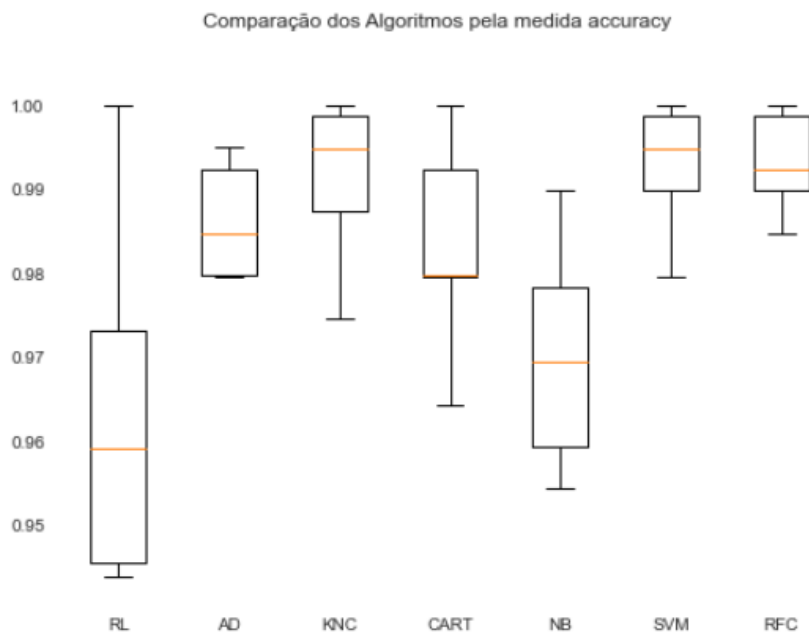
Fonte: Elaboração própria

**Figura 79 – Boxplot de comparação dos algoritmos**

```
# Gráfico boxplot de comparação dos algoritmos, medida: Acurácia

siglasModelosBoxplot = [modelosSiglasDic[x] for x in modelosLista]

imagemComparacao = plt.figure()
imagemComparacao.suptitle('Comparação dos Algoritmos pela medida '+medidaAvaliacao)
ax = imagemComparacao.add_subplot(111)
plt.boxplot(resultados)
ax.set_xticklabels(siglasModelosBoxplot)
plt.show()
```



Selecionado o algoritmo com melhor desempenho, qual seja, o Random Forest Classifier (Floresta Aleatória)<sup>36</sup>, passou-se ao treinamento (figura 80), dividindo-se o *dataset* em 80% para treinamento e 20% para testes, *test\_size*=0.2, do total de linhas do *dataframe* “*df\_matrizEsparsa\_tfidf\_final*” igual a 1.964. Foram utilizados portanto, 1.571 documentos para treino e 393 para testes.

O resultado da predição da base de testes está representada na figura 81, tendo obtido um escore de 0,997, ou seja, quase toda a base de testes foi classificada corretamente, conforme os “*target\_names*” (figura 27), em “*constitucionalidade*” = 0 e “*inconstitucionalidade*” = 1, sendo que o modelo obteve um escore de 0,997 (figura 80).

<sup>36</sup> <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

**Figura 80 – Treinamento com Random Forest Classifier (Floresta Aleatória)**

```
#Treinamento com Random Forest Classifier (Floresta Aleatória)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
classificador = RandomForestClassifier(n_estimators=1000, random_state=0)
classificador.fit(X_train, y_train)
classificador.score(X, y)

0.9979633401221996
```

**Figura 81 – Array com o resultado da predição**

```
y_pred = classificador.predict(X_test)
y_pred

array([0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1,
       0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1])
```

Construída a matriz de confusão com o código da figura 82, que mostra as quantidades de classificações corretas e incorretas como verdadeiros positivos, falsos positivos, falsos verdadeiros e falsos negativos, figura 81, sendo:

1. verdadeiro positivo (*true positive* - TP): ocorre quando a classificação que se busca é prevista corretamente (no presente projeto é a quantidade de constitucionalidades prevista corretamente);
2. falso positivo (*false positive* - FP): ocorre quando a classificação que se busca é prevista incorretamente (no presente projeto é a quantidade de constitucionalidade prevista incorretamente como inconstitucionalidade);
3. falso verdadeiro (*true negative* - TN): ocorre quando a classificação que não se está buscando é prevista corretamente (no presente projeto, é a quantidade de inconstitucionalidade prevista corretamente); e

4. falso negativo (*false negative* - FN): ocorre quando a classificação que não se está buscando foi prevista incorretamente (no presente projeto, é a quantidade de inconstitucionalidade prevista incorretamente como constitucionalidade).

Conforme a matriz de confusão da figura 83 com os resultados da classificação do modelo, apenas uma decisão de constitucionalidade foi classificada incorretamente, 0,48% do total, enquanto apenas duas decisões de inconstitucionalidade foram classificadas incorretamente, 1,08% do total, com “accuracy” de 0,992, “precision”, “recall” e “f1” da inconstitucionalidade igual a 0,99, “precision” e “f1” da constitucionalidade igual a 0,99 e “recall” igual a 1,00.

**Figura 82 – Código para matriz de confusão, acurácia e relatório da classificação**

```
#Avaliação do modelo: matriz de confusão, acurácia e relatório da classificação
print('Matriz de confusão: \n', confusion_matrix(y_test, y_pred))
print('\nRelatório da classificação: \n', classification_report(y_test, y_pred))
print('Acurácia: ', accuracy_score(y_test, y_pred))
```

Matriz de confusão:

```
[[207  1]
 [  2 183]]
```

Relatório da classificação:

	precision	recall	f1-score	support
0	0.99	1.00	0.99	208
1	0.99	0.99	0.99	185
accuracy			0.99	393
macro avg	0.99	0.99	0.99	393
weighted avg	0.99	0.99	0.99	393

Acurácia: 0.9923664122137404

**Figura 83 – Matriz de confusão**

	CONSTITUCIONALIDADE	INCONSTITUCIONALIDADE	
CONSTITUCIONALIDADE	VERDADEIRO POSITIVO	FALSO POSITIVO	
INCONSTITUCIONALIDADE	FALSO NEGATIVO	FALSO VERDADEIRO	

	CONSTITUCIONALIDADE	INCONSTITUCIONALIDADE	
CONSTITUCIONALIDADE	207	1	0,48%
INCONSTITUCIONALIDADE	2	183	
	1,08%		

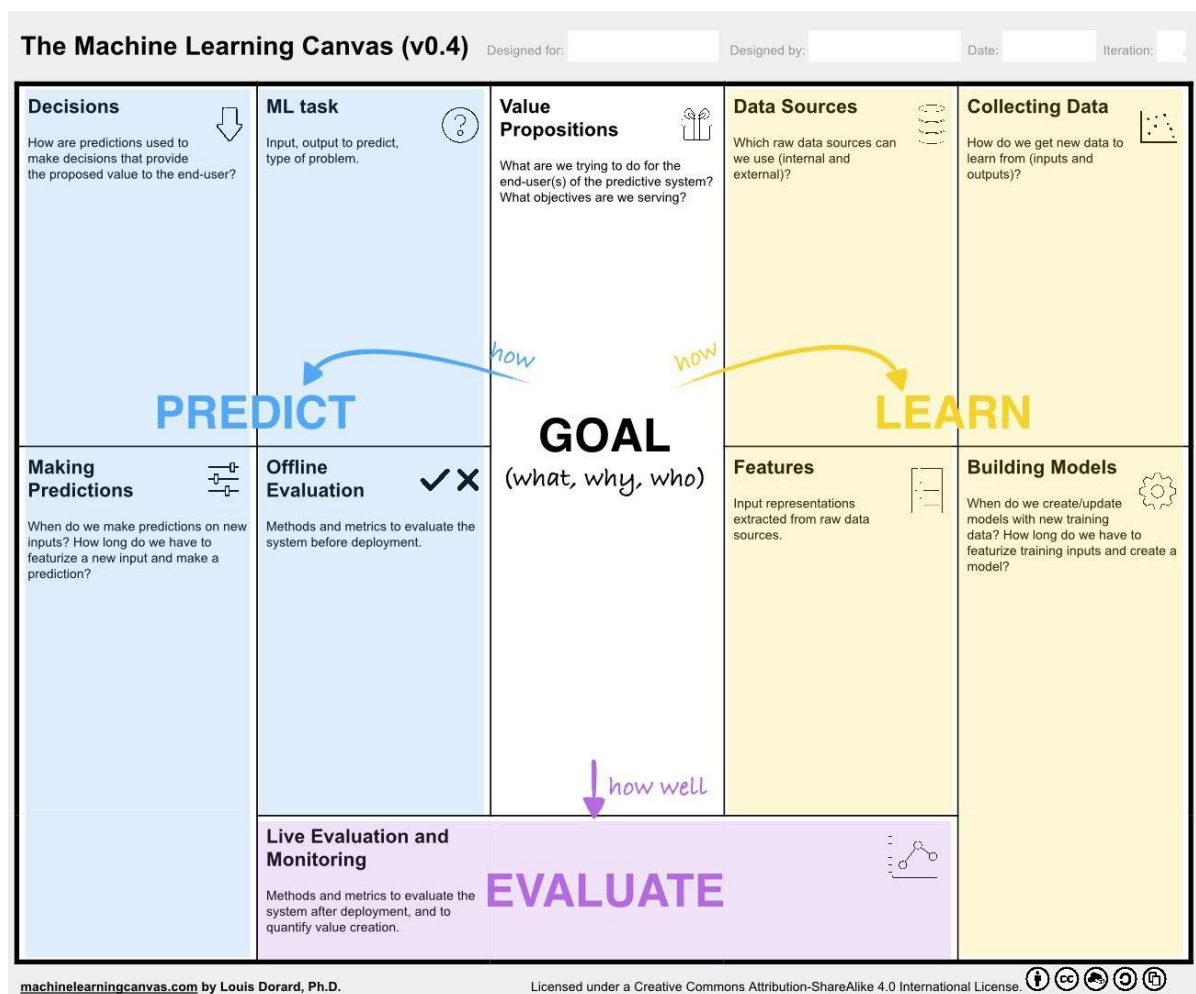
Fonte: elaboração própria.



## 6. Apresentação dos Resultados

Para a apresentação dos resultados obtidos foi utilizado o modelo de Canvas proposto por Dourard<sup>37</sup> (imagem 84).

**Figura 84 – Modelo de Canvas para projetos de *machine learning* proposto por Dourard**



<sup>37</sup> The Machine Learning Canvas (v0.4) by Louis Dorard, Licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.






## The Machine Learning Canvas (v0.4)






Designed for: PUC Minas

Designed by: Gildo S. F. Couto

Date: 10/08/2020

Iteration: .

<p><b>Decisions</b></p>  <p>O resultado do modelo é a predição do resultado das decisões judiciais, possibilitando determinar os encaminhamentos necessários com eficiência e celeridade.</p>	<p><b>ML task</b></p>  <p>A tarefa de <i>machine learning</i> (ML) é a classificação das decisões, tendo como X (previsor) o TF-IDF dos <i>n-grams</i> que resultarem ao final e o resultado das decisões como rótulo ou variável y (alvo).</p> <p>O <i>output</i> é a classificação binomial das decisões em constitucionalidade e inconstitucionalidade.</p>	<p><b>Value Propositions</b></p>  <p>O modelo produzirá a classificação binomial do resultado das decisões judiciais da Justiça Federal de 2º grau com o assunto 6040 - Funrural, em relação à discussão da constitucionalidade da contribuição previdenciária para o Funrural em duas classes: “constitucional” e “inconstitucional”, não sendo necessário aos responsáveis pelos processos judiciais ler e analisar as decisões para saber o resultado.</p> <p>Valores gerados:</p> <ul style="list-style-type: none"> <li>- celeridade: o modelo tem uma velocidade de análise muito maior que a análise manual das ações. Podem ser analisadas e classificadas milhares de ações judiciais em alguns minutos.</li> <li>- eficiência: o percentual de erro na classificação do modelo é muito inferior à análise humana.</li> <li>- redução de custos: alocação da mão de obra técnica e especializada em outras atividades, mais complexas.</li> <li>- otimização da recuperação de créditos tributários: a análise extremamente ágil e em massa das ações judiciais com o modelo possibilitará a cobrança de um número muito maior de processos.</li> </ul>	<p><b>Data Sources</b></p>  <p>As fontes de dados serão os dados de uma planilha XLSX com os números dos processos e decisões em que as contribuições que foram julgadas constitucionais ou inconstitucionais, bem como o texto das decisões extraídas da <i>web</i>.</p>	<p><b>Collecting Data</b></p>  <p>Através do número do processo serão extraídos os textos das decisões judiciais do <i>site</i> do TRF1, via <i>web scraping</i> com código Python, utilizando as libs “BeautifulSoup”, “wget” e “request”.</p> <p>Os dados serão processados com técnicas de NLP, stemizando e vetorizando as palavras do <i>corpus</i>, reduzindo a dimensionalidade e tratando <i>outliers</i>, colinearidades, inconsistências, dados faltantes e outros problemas.</p>
--	---	---	--	--

<div><div>Making Predictions</div><div></div><div>As predições serão realizadas após o processamento dos dados e de forma muito rápida.</div></div>	<div><div>Offline Evaluation</div><div></div><div>As técnicas de validação cruzada poderão ser utilizadas para avaliação do modelo, bem como as métricas de desvio-padrão, coeficiente de variação, valor mínimo e valor máximo dos parâmetros:<div><div>1. “accuracy”</div><div>2. “precision”</div><div>3. “recall”</div><div>4. “f1”</div><div>5. “jaccard”</div></div></div></div>		<div><div>Features</div><div></div><div>Os dados brutos de entrada disponíveis serão:<div><div>Número do processo judicial na Justiça Federal</div><div>Número da decisão da Justiça Federal de 2º grau</div><div>Resultado da decisão</div><div>Taxonomia processual com o do assunto da discussão judicial</div><div>Texto da decisão judicial extraída do site do TRF1.</div></div></div></div>	<div><div>Building Models</div><div></div><div>Serão testados sete algoritmos de ML:<div><div>1. Logistic Regression (RL): Regressão Logística</div><div>2. Linear Discriminant Analysis (AD): Análise Discriminante</div><div>3. K-Nearest Neighbors (K-NN): K-Vizinhos mais próximos</div><div>4. Decision Tree Classifier (CART): Árvore de Decisão</div><div>5. Naive Bayes (NB)</div><div>6. Support Vector Machines (SVM): Máquinas de Vetores de Suporte</div><div>7. Random Forest Classifier (RFC): Floresta Aleatória</div></div><div>O modelo será construído com o algoritmo que tiver o melhor desempenho na classificação.</div></div></div>
<div><div>Live Evaluation and Monitoring</div><div></div><div>A avaliação e monitoramento do modelo poderá ser realizada através da classificação de novas decisões judiciais, medindo a acurácia das classes, bem como pela medição do tempo gasto para classificação de novos datasets pelo modelo. Por limitações de tempo, essa etapa não será realizada no presente projeto.</div></div>				

## 7. Links

Link para o vídeo de apresentação do projeto: [youtu.be/-F6eEasnfJE](https://youtu.be/-F6eEasnfJE)

Link para o *notebook* do projeto: [rebrand.ly/JupyterNotebook](https://rebrand.ly/JupyterNotebook)

Link para os arquivos do projeto: [rebrand.ly/TCC-Ciencia-de-Dados](https://rebrand.ly/TCC-Ciencia-de-Dados)

## REFERÊNCIAS

DUCHESNE, Pierre; RÉMILLARD, Bruno (Ed.). **Statistical modeling and analysis for complex data problems**. Springer Science & Business Media, 2005.

Escovedo, Tatiana (2020-02-27T22:58:59). **Introdução a Data Science**. Casa do Código. Edição do Kindle.

Géron, Aurélien. **Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow**. Edição do Kindle.

Steven Bird, Ewan Klein, and Edward Loper (2009-06-12). **Natural Language Processing with Python**. O'Reilly Media. Edição do Kindle.

Beysolow II, Taweh (2018-09-11). **Applied Natural Language Processing with Python**. Apress. Edição do Kindle.

Vasiliev, Yuli (2020-04-27T22:58:59). **Natural Language Processing with Python and spaCy**. No Starch Press. Edição do Kindle.

GENTZKOW, Matthew & KELLY, Bryan & TADDY, Matt: **Text as Data**, Journal of Economic Literature, 57(3), 535–574, 2019, disponível em <<https://web.stanford.edu/~gentzkow/research/text-as-data.pdf>>. Acesso em: 5 out 2020.

**Nonlinear Dimensionality Reduction by Locally Linear Embedding**, S. Roweis, L. Saul (2000).

LUHN, H.P., **The automatic creation of literature abstracts**, IBM Journal of Research and Development, 2, 159-165 (1958).

A. Gelbukh, G. Sidorov, **International Conference on Intelligent Text Processing and Computational Linguistics**, páginas 332-335, Springer, Berlin, Heidelberg

Hair, Joseph F., William C. Black, Barry J. Babin, e Ronald L. Tatham. 2009. **Análise Multivariada de Dados**. 6a ed. São Paulo: Bookman.

KOTSIANTIS, Sotiris B.; ZAHARAKIS, Ioannis D.; PINTELAS, Panayiotis E. **Machine learning: a review of classification and combining techniques**. *Artificial Intelligence Review*, v. 26, n. 3, p.159-190, 2006.

DUCHESNE, Pierre; RÉMILLARD, Bruno (Ed.). **Statistical modeling and analysis for complex data problems**. Springer Science & Business Media, 2005.

Escovedo, Tatiana (2020-02-27T22:58:59). **Introdução a Data Science**. Casa do Código. Edição do Kindle.

**Lei nº 10.256, de 09 de julho 2011**, disponível em <[http://www.planalto.gov.br/ccivil\\_03/leis/LEIS\\_2001/L10256.htm](http://www.planalto.gov.br/ccivil_03/leis/LEIS_2001/L10256.htm)>. Acesso em: 21 jul 2020.

**Remember the 5 W's**, disponível em <<https://its.unl.edu/bestpractices/remember-5-ws>>. Acesso em: 21 jul 2020.

**Justiça Federal registra mais de 1,5 milhão de decisões em regime de trabalho remoto**, disponível em <<https://www.cjf.jus.br/cjf/noticias/2020/07-julho/justica-federal-registra-mais-de-1-5-milhao-de-decisoes-em-regime-de-trabalho-remoto>>. Acesso em: 21 jul 2020.

**Receita Federal cobra R\$ 260 milhões de Funrural devido por produtores rurais de Minas Gerais**, disponível em <<https://receita.economia.gov.br/noticias/ascom/2018/agosto/receita-federal-cobra-r-260-milhoes-de-funrural-devido-por-produtores-rurais-de-minas-gerais>>. Acesso em: 21 jul 2020.

**Stemming and lemmatization**, disponível em <<https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html>> © 2008 Cambridge University Press>. Acesso em: 30 jul 2020.

**TRF1 - Inteiro Teor de Acórdãos, Decisões e Despachos**, disponível em <<https://arquivo.trf1.jus.br/index.php>>. Acesso em: 15 a 18 set 2020.

**Consulta aos diários eletrônicos da Justiça Federal da 1a Região, TRIBUNAL REGIONAL FEDERAL DA 1a REGIÃO**, disponível em <<https://edj.trf1.jus.br/>>. Acesso em: 15 set 2020.

**Documentação Pandas Profiling**, disponível em <<https://pandas-profiling.github.io/pandas-profiling/docs/master/index.html>>/. Acesso em: 15 set 2020.

**Documentação do método padrão de correlação do Pandas DataFrame**, disponível em <<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html>>. Acesso em: 21 set 2020.

**Metrics and scoring: quantifying the quality of predictions**, disponível em <[https://scikit-learn.org/stable/modules/model\\_evaluation.html](https://scikit-learn.org/stable/modules/model_evaluation.html)>. Acesso em: 23 set 2020.

**Documentação Scikit-learn, Machine Learning in Python, Random Forest Classifier**, disponível em <<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>>. Acesso em: 23 set 2020.

@article{scikit-learn, title={**Scikit-learn: Machine Learning in {P}ython**}, author={Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E.}, journal={Journal of Machine Learning Research}, volume={12}, pages={2825--2830}, year={2011}}