Notebook Jupyter 5_NLP_modeloClassificador

Classificação da Aplicação por aprendizado de máquina

Importando bibliotecas

In [1]:
```
import pandas as pd, numpy as np, time
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
```

In [2]:
```
# Data e hora da execução do script
initot=time.time()
print(f'Código executado em {time.strftime("%d/%m/%Y às %H:%M", time.localtime(tim
e.time()))}')
```

```
Código executado em 25/01/2022 às 13:00
```

Importando a lista de Aplicações

In [3]:
```
df_aplicacoes = pd.read_csv(r'./bases/Aplicacoes.csv')
```

In [4]:
```
df_aplicacoes.head(2)
```

Out[4]:

	APLICACOES
0	ACELLERA ACX 250F 250
1	ACELLERA FRONTLANDER 500

In [5]:
```
df_aplicacoes.tail(2)
```

Out[5]:

	APLICACOES
852	ZONGSHEN ZS 125
853	ZONGSHEN ZS 200

In [6]:
```
df_aplicacoes.shape
```

Out[6]: (854, 1)

CountVectorizer

CountVectorizer do DataSet das Aplicaçãoes

Análise do uso do TF-IDF

Um ponto muito importante desse trabalho foi a decisão quanto à **utilização ou não do TF-IDF** (*term frequency–inverse document frequency*), que serve para indicar o grau de importância de uma palavra em relação ao conjunto total de palavras existentes. Na prática, funciona com um ponderador dos termos na classificação.
Especificamente no nosso trabalho, o objetivo é determinar a aplicação a que se refere uma descrição, então a existência de um termo mais raro, que ganharia peso com o TF-IDF, não pode se sobrepor à aparição de vários termos que indicam uma classificação.
Após diversos testes observou-se que o uso do TF-IDF trazia várias classificações inadequadas ao conjunto de dados. Por exemplo, com a utilização do TF-IDF, a descrição já realizada a limpeza “yamaha fazer 150 gt” era classificada pelo modelo Linear SVC como *“SUZUKI GT”*, mas a aplicação mais adequada seria *“YAMAHA FAZER YS150 150”*.
Decidiu-se, então, pela não utilização do ponderador TF-IDF.

In [7]:
```
# Criação da função CountVectorizer
cvta = CountVectorizer(strip_accents='ascii', lowercase=True)
X_cvta = cvta.fit_transform(df_aplicacoes['APLICACOES'])
```

Treinando os Modelos com o DataSet Aplicações

Definindo os parâmetros

Utilizaremos toda a base no treinamento, pois a intenção é criar uma função de classificação para um dataset onde a classificação é inexistente.

In [8]:
```
X_train=X_cvta.toarray()
y_train=np.array(df_aplicacoes['APLICACOES'])
y_train1=df_aplicacoes['APLICACOES'].index.to_numpy()
```

Modelo LinearSVC

In [9]:
```
# Criando modelo
clfsvc = LinearSVC()
# Treinamento do modelo
clfsvc.fit(X_train, y_train)
```

Out[9]:
```
LinearSVC()
```

Função de classificação LinearSVC

A função para utilização do modelo, recebe a descrição filtrada Modelo e retorna a aplicação.

In [10]:
```
def classificaAplicacaoSVC(modelo):
    novo_cvta = cvta.transform(pd.Series(modelo))
    aplicacao = clfsvc.predict(novo_cvta)[0]
    return aplicacao
```

No final o nosso resultado mostrando (Modelo: descrição filtrada para o modelo e Aplicação: aplicação prevista).

In [11]:
```
# Lista de exemplos de novos produtos
modelos = ['150 CG HONDA TITAN',
           '125 CARGO CG HONDA TITAN',
           'BIZ C100 HONDA',
           '100 HONDA BIZ',
           '100 BIZ BRAVO HONDA',
           '125 YBR GT YAMAHA',
           '250F TWISTER HONDA']
# Loop for para fazer a predição do departamento de novos produtos
for modelo in modelos:
    print('Modelo:', modelo, ' Aplicação:', classificaAplicacaoSVC(modelo))
```

```
Modelo: 150 CG HONDA TITAN  Aplicação: HONDA CG TIT TITAN 125 150 160
Modelo: 125 CARGO CG HONDA TITAN  Aplicação: HONDA CG TIT TITAN 125 150 160
Modelo: BIZ C100 HONDA  Aplicação: HONDA BIZ C100 125 C125
Modelo: 100 HONDA BIZ  Aplicação: HONDA BIZ C100 125 C125
Modelo: 100 BIZ BRAVO HONDA  Aplicação: SHINERAY BRAVO 200
Modelo: 125 YBR GT YAMAHA  Aplicação: SUZUKI GT
Modelo: 250F TWISTER HONDA  Aplicação: HONDA TWISTER CBX 250
```

Modelo Multinomial Naive Bayes

In [12]:
```
# Criando modelo
clfmnb = MultinomialNB()
# Treinamento do modelo
clfmnb.fit(X_train, y_train1)
```

Out[12]:
```
MultinomialNB()
```

Função de classificação

A função para utilização do modelo, recebe a descrição filtrada Modelo e retorna a aplicação.

In [13]:
```
def classificaAplicacaoMNB(modelo):
    novo_cvta = cvta.transform(pd.Series(modelo))
    aplicacao = df_aplicacoes['APLICACOES'][clfmnb.predict(novo_cvta)[0]]
    return aplicacao
```

No final o nosso resultado mostrando (Modelo: descrição filtrada para o modelo e Aplicação: aplicação prevista).

In [14]:
```
# Lista de exemplos de novos produtos
modelos = ['150 CG FAN HONDA TITAN',
           '125 CARGO CG HONDA TITAN',
           'BIZ C100 HONDA',
           '100 HONDA BIZ',
           '100 BIZ BRAVO HONDA',
           '125 YBR GT YAMAHA',
           '250F TWISTER HONDA']
# Loop for para fazer a predição do departamento de novos produtos
for modelo in modelos:
    print('Modelo:', modelo, 'Aplicação:', classificaAplicacaoMNB(modelo))
```

```
Modelo: 150 CG FAN HONDA TITAN Aplicação: HONDA CG TIT TITAN 125 150 160
Modelo: 125 CARGO CG HONDA TITAN Aplicação: HONDA CG TIT TITAN 125 150 160
Modelo: BIZ C100 HONDA Aplicação: HONDA BIZ C100 125 C125
Modelo: 100 HONDA BIZ Aplicação: HONDA BIZ C100 125 C125
Modelo: 100 BIZ BRAVO HONDA Aplicação: HONDA BIZ C100 125 C125
Modelo: 125 YBR GT YAMAHA Aplicação: YAMAHA FACTOR YBR 125 YBR125
Modelo: 250F TWISTER HONDA Aplicação: HONDA CB 250 250F
```

Modelo de Regressão Logística

In [15]:
```
# Criando modelo
clflgr = LogisticRegression(solver='lbfgs',multi_class='multinomial')
# Treinamento do modelo
clflgr.fit(X_train, y_train)
```

Out[15]:
```
LogisticRegression(multi_class='multinomial')
```

Função de classificação

A função para utilização do modelo, recebe a descrição filtrada Modelo e retorna a aplicação.

In [16]:
```
def classificaAplicacaoLGR(modelo):
    novo_cvta = cvta.transform(pd.Series(modelo))
    aplicacao = clflgr.predict(novo_cvta)[0]
    return aplicacao
```

No final o nosso resultado mostrando (Modelo: descrição filtrada para o modelo e Aplicação: aplicação prevista).

In [17]:
```
# Lista de exemplos de novos produtos
modelos = ['150 CG FAN HONDA TITAN',
           '125 CARGO CG HONDA TITAN',
           'BIZ C100 HONDA',
           '100 HONDA BIZ',
           '100 BIZ BRAVO HONDA',
           '125 YBR GT YAMAHA',
           '250F TWISTER HONDA']
# Loop for para fazer a predição do departamento de novos produtos
for modelo in modelos:
    print('Modelo:', modelo, 'Aplicação:', classificaAplicacaoLGR(modelo))
```

```
Modelo: 150 CG FAN HONDA TITAN Aplicação: HONDA CG TIT TITAN 125 150 160
Modelo: 125 CARGO CG HONDA TITAN Aplicação: HONDA CG TIT TITAN 125 150 160
Modelo: BIZ C100 HONDA Aplicação: HONDA BIZ C100 125 C125
Modelo: 100 HONDA BIZ Aplicação: HONDA BIZ C100 125 C125
Modelo: 100 BIZ BRAVO HONDA Aplicação: HONDA BIZ C100 125 C125
Modelo: 125 YBR GT YAMAHA Aplicação: YAMAHA FACTOR YBR 125 YBR125
Modelo: 250F TWISTER HONDA Aplicação: HONDA TWISTER CBX 250
```

Utilização do modelo

Carregando dataset

In [18]:
```
# Importa base de dados com os modelos já determinados para um dataframe
df = pd.read_excel(r'./bases/dataframe_modelos_class0.xlsx')
df.iloc[:,-2:].head()
```

Out[18]:

	Modelo	APLICACAO
0	150 cg fan honda titan	HONDA CG FAN
1	125 cargo cg honda titan	HONDA CG TIT TITAN 125 150 160
2	125 cg fan honda	HONDA CG FAN
3	biz c100 honda	HONDA BIZ C100 125 C125
4	150 kasinski mirage	KASINSKI MIRAGE 150 250

In [19]:
```
# Verifica o tamnanho do dataframe
df.shape
```

Out[19]: (17484, 6)

Classificando os modelos do DataSet

In [20]:
```
# Modelo Linear SVC
ini=time.time()
now = time.strftime("%H:%M:%S", time.localtime(time.time()))
t1=5/df.shape[0]*1000 # tempo para cada mil registros
print(f"Hora de início: {now}")
print(f"Tempo estimado: {int(t1*df.shape[0]/1000)} segundos")

df['APLICACAOSVC']=df['Modelo'].apply(classificaAplicacaoSVC)

now = time.strftime("%H:%M:%S", time.localtime(time.time()))
fim=time.time()
print("Hora de término:" + str(now))
print("Tempo decorrido: " + str(round((fim-ini),1)) + " segundos.")
```

```
Hora de início: 13:00:06
Tempo estimado: 5 segundos
Hora de término:13:00:11
Tempo decorrido: 4.9 segundos.
```

In [21]:
```
# Modelo Multinomial NB
ini=time.time()
now = time.strftime("%H:%M:%S", time.localtime(time.time()))
t1=53/df.shape[0]*1000 # tempo para cada mil registros
print(f"Hora de início: {now}")
print(f"Tempo estimado: {int(t1*df.shape[0]/1000)} segundos")

df['APLICACAOMNB']=df['Modelo'].apply(classificaAplicacaoMNB)

now = time.strftime("%H:%M:%S", time.localtime(time.time()))
fim=time.time()
print("Hora de término:" + str(now))
print("Tempo decorrido: " + str(round((fim-ini),1)) + " segundos.")
```

```
Hora de início: 13:00:11
Tempo estimado: 53 segundos
Hora de término:13:01:04
Tempo decorrido: 53.1 segundos.
```

In [22]:
```
# Modelo Regressão Logística
ini=time.time()
now = time.strftime("%H:%M:%S", time.localtime(time.time()))
t1=68/df.shape[0]*1000 # tempo para cada mil registros
print(f"Hora de início: {now}")
print(f"Tempo estimado: {int(t1*df.shape[0]/1000)} segundos")

df['APLICACAOLGR']=df['Modelo'].apply(classificaAplicacaoLGR)

now = time.strftime("%H:%M:%S", time.localtime(time.time()))
fim=time.time()
print("Hora de término:" + str(now))
print("Tempo decorrido: " + str(round((fim-ini),1)) + " segundos.")
```

```
Hora de início: 13:01:04
Tempo estimado: 68 segundos
Hora de término:13:02:05
Tempo decorrido: 61.1 segundos.
```

In [23]:
```
df.shape
```

Out[23]:
```
(17484, 9)
```

In [24]:
```
df.iloc[:,-6:].sample(5)
```

Out[24]:

	DESCRICAO	Modelo	APLICACAO	APLICACAOSVC	APLICACAOMNB	APLICACAOLGR
12718	ybr 125 factor	125 factor yamaha ybr	YAMAHA FACTOR YBR 125 YBR125	YAMAHA FACTOR YBR 125 YBR125	YAMAHA FACTOR YBR 125 YBR125	YAMAHA FACTOR YBR 125 YBR125
6455	cg 150 titan 160 cargo 17 fan start	150 160 cargo cg fan honda titan	HONDA CG FAN	HONDA CG FAN	HONDA CG TIT TITAN 125 150 160	HONDA CG TIT TITAN 125 150 160
2956	fazer 250	250 fazer yamaha	YAMAHA FAZER YS250 250	YAMAHA FAZER YS250 250	YAMAHA FAZER YS250 250	YAMAHA FAZER YS250 250
23	fan 125	125 fan honda	HONDA CG FAN	HONDA CG FAN	HONDA CB 125	HONDA CG FAN
4468	darom pop	honda pop	HONDA POP	HONDA POP	HONDA POP	HONDA POP

Comparação das classificações

In [25]:
```
# Definição dos filtros
f1 = df['APLICACAOSVC']==df['APLICACAOMNB'] # Modelos SVC e MNB iguais
f2 = df['APLICACAOSVC']==df['APLICACAOLGR'] # Modelos SVC e LGR iguais
f3 = df['APLICACAOSVC']==df['APLICACAO'] # Modelo e SVC iguais
f4 = df['APLICACAOMNB']==df['APLICACAOLGR'] # Modelos MNB e LGR iguais
f5 = df['APLICACAOMNB']==df['APLICACAO'] # Modelo e MNB iguais
f6 = df['APLICACAOLGR']==df['APLICACAO'] # Modelo e LGR iguais
```

In [26]:
```
# Quantidade de registros divergentes entre os modelos SVC e Multinomial NB
df[~f1].shape
```

Out[26]:
```
(3815, 9)
```

In [27]:
```
# Quantidade de registros divergentes entre os modelos SVC e Regressão Logística
df[~f2].shape
```

Out[27]:
```
(3251, 9)
```

In [28]:
```
# Quantidade de registros divergentes entre a extração e o modelo SVC
df[~f3].shape
```

Out[28]: (1624, 9)

In [29]:
```
# Quantidade de registros divergentes entre os modelos MNB e Regressão Logística
df[~f4].shape
```

Out[29]: (1166, 9)

In [30]:
```
# Quantidade de registros divergentes entre a extração e o Modelo MNB
df[~f5].shape
```

Out[30]: (3622, 9)

In [31]:
```
# Quantidade de registros divergentes entre a extração e o Modelo LGR
df[~f6].shape
```

Out[31]: (3014, 9)

In [32]:
```
# Quantidade de registros divergente entre os quatro
df[~(f1 & f2 & f3)].shape
```

Out[32]: (4589, 9)

In [33]:
```
# Quantidade de registros iguais nos quatro
df[f1 & f2 & f3].shape
```

Out[33]: (12895, 9)

Em virtude do tempo de processamento maior e também da maior quantidade de divergências, decidiu-se por abandonar o uso da modelo de regressão logística.

In [34]:
```
df.drop('APLICACAOLGR', axis=1, inplace=True)
```

In [35]:
```
df[df['APLICACAOSVC']!=df['APLICACAOMNB']].iloc[:5,-5:]
```

Out[35]:

	DESCRICAO	Modelo	APLICACAO	APLICACAOSVC	APLICACAOMNB
0	honda cg 150 titan ks es mix fan	150 cg fan honda titan	HONDA CG FAN	HONDA CG FAN	HONDA CG TIT TITAN 125 150 160
2	honda cg 125 fan	125 cg fan honda	HONDA CG FAN	HONDA CG FAN	HONDA CG 125
11	hunter max 125	125 hunter max sundown	XXX	SUNDOWN MAX	SUNDOWN HUNTER
17	yes intruder 125	125 intruder suzuki yes	SUZUKI YES EN 125	SUZUKI INTRUDER	SUZUKI YES EN 125
22	jet 49cc	jet shineray	['SHINERAY JET 125', 'SHINERAY JET 50']	SHINERAY JET 50	SHINERAY JET 125

In [36]:
```
df['DESCRICAO DO PRODUTO'].iloc[11]
```

Out[36]:
```
'80360 KIT DE TRANSMISSÃO, COMPOSTO DE CORRENTE, COROA E PINHÃO PARA MOTOCICLETA H
UNTER / MAX 125, MARCA ALLEN.'
```

Melhorando a classificação

Para chegar à classificação final a ser utilizada na função de classificação, utilizaremos algumas regras:

1. SE APLICACAOMNB==APLICACAOSVC ==> APLICACAOFIM=APLICACAOSVC
2. SE APLICACAOMNB!=APLICACAOSVC
   A. SE APLICACAO=='XXX' ==> APLICACAOFIM=[APLICACAOSVC,APLICACAOMNB]
   B. SE APLICACAO==APLICACAOSVC ==> APLICACAOFIM=APLICACAOSVC
   C. SE APLICACAO==APLICACAOMNB ==> APLICACAOFIM=APLICACAOMNB
   D. SE APLICACAO for uma lista:
      - SE APLICACAOSVC estiver na lista ==> APLICACAOFIM=APLICACAOSVC
      - SE APLICACAOMNB estiver na lista ==> APLICACAOFIM=APLICACAOSMNB
      - SENÃO ==> APLICAFIM=LISTA+[APLICACAOSVC,APLICACAOMNB]
   E. SE APLICACAO!=APLICACAOSVC!=APLICACAOMNB ==> APLICACAOFIM= [APLICACAO,APLICACAOSVC,APLICACAOMNB]

In [37]:
```
df=df.assign(APLICACAOFIM=df.APLICACAOMNB.tolist())
df.shape
```

Out[37]: (17484, 9)

In [38]:
```
df.iloc[:,-5:].sample(10)
```

Out[38]:

	Modelo	APLICACAO	APLICACAOSVC	APLICACAOMNB	APLICACAOFIM
4348	150 broz honda nx	HONDA NXR 150 160 BROZ	HONDA NX 150 200 250	HONDA NX 150 200 250	HONDA NX 150 200 250
2631	250 fazer yamaha	YAMAHA FAZER YS250 250	YAMAHA FAZER YS250 250	YAMAHA FAZER YS250 250	YAMAHA FAZER YS250 250
12424	150 fazer yamaha	YAMAHA FAZER YS150 150	YAMAHA FAZER YS150 150	YAMAHA FAZER YS150 150	YAMAHA FAZER YS150 150
14705	125 cg fan honda	HONDA CG FAN	HONDA CG FAN	HONDA CG 125	HONDA CG 125
6496	125 biz c125 honda	HONDA BIZ C100 125 C125	HONDA BIZ C100 125 C125	HONDA BIZ C100 125 C125	HONDA BIZ C100 125 C125
6883	125 biz honda	HONDA BIZ C100 125 C125	HONDA BIZ C100 125 C125	HONDA BIZ C100 125 C125	HONDA BIZ C100 125 C125
10051	falcon honda nx	HONDA NX 400 FALCON	HONDA NX 400 FALCON	HONDA NX 400 FALCON	HONDA NX 400 FALCON
12506	125 honda xl	HONDA XL XLS 125 XL125 XL125S	HONDA VARADERO XL 1000V	HONDA XL XLS 125 XL125 XL125S	HONDA XL XLS 125 XL125 XL125S
4788	biz honda	HONDA BIZ C100 125 C125	HONDA BIZ C100 125 C125	HONDA BIZ C100 125 C125	HONDA BIZ C100 125 C125
1184	150 160 cargo cg fan honda titan	HONDA CG FAN	HONDA CG FAN	HONDA CG TIT TITAN 125 150 160	HONDA CG TIT TITAN 125 150 160

In [39]:
```
colindex = df.columns.get_loc("APLICACAOFIM")
for i in range(df.shape[0]):
    #print(i)
    # se APLICACAOSVC==APLICACAOMNB ==> já aplicado pelo df.assign
    # se APLICACAOSVC!=APLICACAOMNB
    if df['APLICACAOSVC'][i]!=df['APLICACAOMNB'][i]:
        # se APLICAÇÃO for sem valor ('XXX') ==> APLICACAOFIM=[APLICACAOSVC,APLICACAOMNB]
        if df['APLICACAO'][i]=='XXX':
            # APLICACAOFIM será a lista com os dois valores
            df.iloc[i,colindex]=str([df['APLICACAOSVC'][i], df['APLICACAOMNB'][i]])
            continue # passa para o próximo i
        elif df['APLICACAO'][i]==df['APLICACAOSVC'][i]:
            df.iloc[i,colindex]=df['APLICACAOSVC'][i]
            continue # passa para o próximo i
        elif df['APLICACAO'][i]==df['APLICACAOMNB'][i]:
            df.iloc[i,colindex]=df['APLICACAOMNB'][i]
            continue # passa para o próximo i
        elif "," in df['APLICACAO'][i]: # se tiver vírgula, a APLICACAO é uma lista
            aplictemp=df['APLICACAO'][i].replace("[","").replace(", ",";").replace(",",";").replace("'","").replace("]","")
            aplictemp=aplictemp.split(';')
            # se APLICACAOSVC estiver na lista
            if df['APLICACAOSVC'][i] in aplictemp:
                df.iloc[i,colindex]=df['APLICACAOSVC'][i]
                continue # passa para o próximo i
            # se APLICACAOMNB estiver na lista
            elif df['APLICACAOMNB'][i] in aplictemp:
                df.iloc[i,colindex]=df['APLICACAOMNB'][i]
                continue # passa para o próximo i
            else:
                aplictemp.append(df['APLICACAOSVC'][i])
                aplictemp.append(df['APLICACAOMNB'][i])
                df.iloc[i,colindex]=str(aplictemp)
                continue # passa para o próximo i
        else: # caso seja um valor (diferente de ambos os classificadores)
            # APLICACAO!=APLICACAOSVC!=APLICACAOMNB ==> APLICACAOFIM=[APLICACAO,APLICACAOSVC,APLICACAOMNB]
            df.iloc[i,colindex]=str([df['APLICACAO'][i],df['APLICACAOSVC'][i], df['APLICACAOMNB'][i]])
            continue # passa para o próximo i
```

Recomparação das classificações

In [40]:
```
# Definição dos filtros
f1 = df['APLICACAOSVC']==df['APLICACAOMNB'] # Modelos SVC e MNB iguais
f2 = df['APLICACAOSVC']==df['APLICACAOFIM'] # Modelo manual e SVC iguais
f3 = df['APLICACAOMNB']==df['APLICACAOFIM'] # Modelo manual e MNB iguais
```

In [41]:
```
# Quantidade de registros divergentes entre os modelos SVC e Multinomial NB
df[~f1].shape
```

Out[41]: (3815, 9)

In [42]:
```
# Quantidade de registros divergentes entre a extração e o modelo SVC
df[~f2].shape
```

Out[42]: (911, 9)

In [43]:
```
# Quantidade de registros divergentes entre a extração e o modelo SVC
df[~f3].shape
```

Out[43]: (3066, 9)

In [44]:
```
# Quantidade de registros divergente entre os três
df[~(f1 & f2)].shape
```

Out[44]: (3815, 9)

In [45]:
```
# Quantidade de registros iguais nos três
df[f1 & f2].shape
```

Out[45]: (13669, 9)

In [46]:
```
# Registros ainda não definidos (lista de opções)
# O filtro definirá True se APLICACAOFIM iniciar com '[' ou False caso contrário.
filtro=[]
for i, aplicacao in enumerate(df['APLICACAOFIM']):
    if aplicacao[0]=='[':
        filtro.append(True)
    else:
        filtro.append(False)
```

In [47]:
```
df[filtro].iloc[:,-5:].head()
```

Out[47]:

	Modelo	APLICACAO	APLICACAOSVC	APLICACAOMNB	APLICACAOFIM
11	125 hunter max sundown	XXX	SUNDOWN MAX	SUNDOWN HUNTER	['SUNDOWN MAX', 'SUNDOWN HUNTER']
103	200 cbx honda xr	XXX	HONDA XR	HONDA STRADA CBX 200	['HONDA XR', 'HONDA STRADA CBX 200']
124	150 fazer sm yamaha	XXX	HUSQVARNA SM	YAMAHA FAZER YS150 150	['HUSQVARNA SM', 'YAMAHA FAZER YS150 150']
282	125 hunter max sundown	XXX	SUNDOWN MAX	SUNDOWN HUNTER	['SUNDOWN MAX', 'SUNDOWN HUNTER']
304	125 hunter max sundown	XXX	SUNDOWN MAX	SUNDOWN HUNTER	['SUNDOWN MAX', 'SUNDOWN HUNTER']

In [48]:
```
df[filtro].shape
```

Out[48]: (162, 9)

Classificando pela quantidade de termos em comum

In [49]:
```
# determina o(s) índice(s) do valor máximo de uma lista
def indicesMaxLista(lista):
    maximo=max(lista)
    indices=[]
    for i, elemento in enumerate(lista):
        if elemento==maximo:
            indices.append(i)
    return indices
```

In [50]:
```
colindex = df.columns.get_loc("APLICACAOFIM") # índice da coluna
for i, row in df[filtro].iterrows(): #para cada linha do dataset
    qtds=[]
    aplicMOD=row['Modelo'].upper().split()
    aplicFIM=row['APLICACAOFIM'].replace("[","").replace(", ",";").replace(",",";")
.replace("'","").replace("]","").split(';')
    for a, aplicacao in enumerate(aplicFIM):
        aplicFIMlista=aplicacao.split()
        qtds.append(len(set(aplicFIMlista).intersection(set(aplicMOD)))) # determin
ar len da interseção
    indicesMax = indicesMaxLista(qtds)
    if len(indicesMax)==1:
        df.iloc[i,colindex]=aplicFIM[indicesMax[0]]
    else:
        df.iloc[i,colindex]=str([aplicFIM[x] for x in indicesMax])
```

In [51]:
```
# Registros ainda não definidos (lista de opções)
# O filtro definirá True se APLICACAOFIM iniciar com '[' ou False caso contrário.
filtro=[]
for i, aplicacao in enumerate(df['APLICACAOFIM']):
    if aplicacao[0]=='[':
        filtro.append(True)
    else:
        filtro.append(False)
```

In [52]:
```
print(f'Registros a classificar: {df[filtro].shape[0]}')
```

```
Registros a classificar: 67
```

Precisamos agora fazer a observação manual dos registros divergentes para correção.
A seguir, exportaremos o arquivo em excel para fazer a classificação manual em outro notebook.

Exportando o DataSet para classificação "manual"

Exportando para um arquivo CSV

In [53]:
```
df.to_csv(r'./bases/dataframe_modelos_classificado_manual-tf.csv', index = False, h
eader = True)
```

Exportando para um arquivo de planilha do Excel

In [54]:
```
df.to_excel(r'./bases/dataframe_modelos_classificado_manual-tf.xlsx', index = False
, header = True)
```

In [55]:
```
tempotot=time.time()-initot
if tempotot>60:
    print(f'Tempo total de execução: {tempotot/60:.2f} minutos.')
else:
    print(f'Tempo total de execução: {tempotot:.2f} segundos.')
```

```
Tempo total de execução: 2.20 minutos.
```