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

Diego de Borba Barbosa

APRENDIZADO DE MÁQUINA APLICADO AO COMÉRCIO EXTERIOR
MODELO PREDITIVO PARA DETERMINAÇÃO DO PAÍS DE ORIGEM DE MERCADORIAS IMPORTADAS

Belo Horizonte
2022

Diego de Borba Barbosa

APRENDIZADO DE MÁQUINA APLICADO AO COMÉRCIO EXTERIOR
MODELO PREDITIVO PARA DETERMINAÇÃO DO PAÍS DE ORIGEM DE MERCADORIAS IMPORTADAS

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
2022

SUMÁRIO

1. Introdução

1.1. Contextualização

A proposta deste trabalho é a criação de um modelo de aprendizado de máquina que possa ser útil nas atividades da Receita Federal do Brasil (RFB). Mais especificamente, um modelo preditivo que possa ser utilizado no gerenciamento de riscos de comércio exterior, especialmente importações.

Diversas variáveis do processo de importação podem ser trabalhadas em um modelo de aprendizado de máquina. Propõe-se, a criação de um modelo preditivo, em linguagem Python, para determinar o país de origem (Argentina, China, Chile, Estados Unidos, Alemanha, Índia etc) em função de outros dados das importações, os quais serão obtidos exclusivamente a partir de **fontes abertas** neste trabalho.

O modelo proposto poderá auxiliar na identificação de irregularidades, ou mesmo fraudes, no processo de importação ou exportação, pois, o **país de origem das mercadorias importadas** é um fator essencial na correta aplicação de margens de preferências tarifárias, alíquotas de impostos, medidas de defesa comercial e restrições de compra/venda internacional.

1.2. O problema proposto

Este Trabalho de Conclusão de Curso tem por objetivo desenvolver e comparar modelos de aprendizado de máquina para determinar o país de origem das mercadorias importadas em função de informações gerais, dados abertos, sobre a operação de importação.

(Why?) Por que esse problema é importante?

A Facilitação do Comércio Internacional veio à tona no final do século XX, culminando com a assinatura da Convenção Revisada de Kyoto em 1999 (RKC). O RKC foi um marco im-

portante que trouxe princípios orientadores para facilitar e impulsionar o comércio internacional de maneira segura, eficiente e transparente. Posteriormente, com o Acordo de Facilitação do Comércio (TFA) da Organização Mundial do Comércio (Conferência Ministerial da OMC de 2013 em Bali), que entrou em vigor em fevereiro de 2017, tais princípios foram detalhados de forma a possibilitar a concretização da facilitação de comércio por todos os membros. Essas medidas incluem: transparência e disponibilidade de procedimentos e normas, ambientes em janela única, benefícios para operadores autorizados, reduções de tempos, coordenação entre agentes públicos, uso intensivo de tecnologia da informação e gerenciamento de riscos, entre outros.

As concessões de medidas de facilitação de comércio devem ser contrabalanceadas pelo uso do gerenciamento de riscos, ou seja, quanto mais concessões são feitas maior deve ser a atuação do país com o gerenciamento de riscos para aferição da conformidade das operações. Por esse motivo a identificação de irregularidades, ou mesmo fraudes, no processo de importação, a variável **país de origem das mercadorias importadas** é um fator essencial na correta aplicação de margens de preferências tarifárias, alíquotas de impostos, medidas de defesa comercial e restrições de compra/venda.

As margens de preferências tarifárias[1] são concessões, em termos percentuais e para produtos específicos, promovidas pelos países-membros de um determinado Acordo Comercial. Quanto maior a margem de preferência, menor será a alíquota do Imposto de Importação efetivamente cobrada. Dessa forma, para um mesmo produto, a depender do país de origem da mercadoria e dos acordos bilaterais ou multilaterais dos países envolvidos, isso pode influenciar sobremaneira no cálculo da alíquota final que deve ser aplicada e o tributo a ser recolhido.

---

[1] http://siscomex.gov.br/aprendendo-a-exportar/identificando-mercados/acordos-comerciais/margens-de-preferencia/ ou https://static.portaldaindustria.com.br/media/filer_public/68/98/6898a38f-2e99-4b34-88db-4abc0af107da/cartilha_acordoscomerciais.pdf

Os instrumentos de defesa comercial[2], como direitos antidumping, medidas compensatórias e salvaguardas, visam resguardar a indústria nacional de distorções do comércio internacional e assegurar uma competição justa entre produtores domésticos e estrangeiros. Podemos entender esses instrumentos como uma sobretaxa, um adicional de Alíquota ad valorem (uma alíquota proporcional ao valor do bem) ou de valor fixo para uma unidade do bem (por exemplo, US$ 4,10/Kg). Compete a Secretaria de Comércio Exterior (SECEX) a investigação e a proposição de medidas de defesa comercial e a RFB a fiscalização da conformidade das operações para as quais há imposição.

Embora a tendência mundial seja pelo livre comércio, há os regimes de sanções multilaterais[3], geralmente definido no âmbito do Conselho de Segurança da Organização das Nações Unidas (CSNU), que são proibições de importação ou exportação determinados produtos e para determinados países. No âmbito do Brasil tais sanções têm sido internalizadas na legislação federal como é o caso do Art. 66 e Art. 254 da Portaria Secex n° 23, de 14 de julho de 2011. Isso significa dizer que em âmbito internacional o Brasil assume e aplicará as proibições de importação ou exportação em suas operações de comércio exterior. Um exemplo atual deste instrumento tem sido as sanções internacionais aplicadas à Rússia tendo em vista a guerra Rússia-Ucrânia.

(Who?) De quem são os dados analisados?

Os dados utilizados neste Trabalho de Conclusão de Curso foram extraídos do site governamental do Ministério da Economia que oferece informações

[2] Produtos com medidas de defesa comercial em vigor https://www.gov.br/produtividade-e-comercio-exterior/pt-br/assuntos/comercio-exterior/defesa-comercial-e-interesse-publico/medidas-em-vigor/medidas-em-vigor ou https://www.gov.br/produtividade-e-comercio-exterior/pt-br/assuntos/comercio-exterior/defesa-comercial-e-interesse-publico

[3] Importações proibidas são tratadas no Art. 66 e exportações proibidas no Art. 254 da Portaria Secex n° 23, de 14 de julho de 2011. Geralmente, trata-se de um regime de sanções adotadas pelo Conselho de Segurança da Organização das Nações Unidas (ONU) e internalizadas pelos países membros como o Brasil.

abertas para o público: *dataset* de dados agregados de importações brasileiras[4] para períodos entre 1997 a 2022, além de *datasets* auxiliares de NCM, PAISES, VIA e URF (UNIDADE DA RFB).

(What?): Quais os objetivos com essa análise?

O objetivo principal do trabalho é montar um modelo preditivo que, a partir dos *datasets* de fontes abertas, seja capaz de prever o país de origem das mercadorias importadas. Sob aspectos quantitativos e qualitativos, serão analisadas as informações contidas nesses *datasets* de forma que seja possível montar e otimizar um modelo de aprendizado de máquina. Importante mencionar que este mesmo modelo poderá ser potencializado internamente na organização visto que há dados sigilosos específicos de cada operação e que não constam em dados abertos.

(Where?): Trata dos aspectos geográficos e logísticos de sua análise.

O *dataset* é oriundo de agregação pelos atributos de que é composto e apresenta a maioria das operações de importação do comércio exterior brasileiro.

(When?): Qual o período está sendo analisado?

Apesar da base de dados aberta disponibilizar informações desde 1997 até março de 2022, selecionamos o período 2019 a 2021 para fase de exploração, *insights* e montagem do modelo preditivo.

[4] https://www.gov.br/produtividade-e-comercio-exterior/pt-br/assuntos/comercio-exterior/estatisticas/base-de-dados-bruta

1.3. Objetivos

O presente trabalho tem por objeto identificar a falsa prestação de informação relativa ao país de origem das mercadorias, podendo se estender aos país de procedência e país de fabricação.

Para tanto, vamos elaborar um modelo preditivo baseando em *Machine Learning* (ML) para determinação do país de origem da mercadoria importada com base apenas em dados abertos e agregados de estatísticas de comércio exterior.

O desenvolvimento dos conhecimentos adquiridos neste curso de especialização também são importantes objetivos para que posteriormente as técnicas e ferramentas de predição possam ser utilizadas em projetos produtivos dentro da organização.

1.4. Metodologia e ferramentas

O projeto foi desenvolvido no *google Colab* criando um Notebook em *python*. O treinamento e testes foram realizados utilizando ambiente *Jupyter Notebook* em máquina local tendo em vista limitação de *RAM no Colab*.

As principais bibliotecas utilizadas foram *pandas, numpy, Scikit-Learn, matplotlib, wordcloud e seaborn*. O primeiro trecho de código demostra a importação das bibliotecas utilizadas:

Importar bibliotecas

```
In [2]: #Carrega as bibliotecas do modelo e algumas utilizadas em testes
        #!pip install pydotplus
        #!pip install dtreeviz
        #from sklearn.inspection import permutation_importance
        #import sklearn
        #print('The scikit-learn version is {}.'.format(sklearn.__version__))
        import functools
        import pandas as pd
        import numpy as np
        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import cross_val_score
        from sklearn.model_selection import GridSearchCV
        from sklearn.model_selection import RandomizedSearchCV
        from sklearn.linear_model import LogisticRegression
        from sklearn import datasets, tree
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.naive_bayes import CategoricalNB
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, mean_absolute_error, r2_score
        from sklearn.metrics import precision_score, recall_score, f1_score
        from sklearn.feature_extraction import DictVectorizer
        from sklearn.preprocessing import LabelEncoder
        from sklearn.model_selection import KFold
        import time
        from functools import wraps
        import pydotplus
        from IPython.display import Image
        import matplotlib.pyplot as plt
        from matplotlib.colors import ListedColormap
        from matplotlib.pyplot import subplots
        from PIL import Image
        from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
        import seaborn as sns
        import warnings
        warnings.filterwarnings("ignore")

        %matplotlib inline
        plt.rcParams["figure.figsize"] = [15, 12]
```

2. Coleta de Dados

2.1. Considerações gerais

O *dataset* principal utilizado e enriquecido utilizado neste trabalho traz **informações agregadas sobre as importações brasileiras.** Os *datasets* foram obtidos da página **Estatísticas de Comércio Exterior em Dados Abertos**, disponibilizado pelo Ministério da Economia no site do gov.br[5].

Os dados abertos de comércio exterior disponíveis não trazem informações completas sobre as operações de importação. A base disponível apresenta os dados agregados, que permitem conhecer o perfil das importações brasileiras em vários aspectos, porém sem de-

[5] https://www.gov.br/produtividade-e-comercio-exterior/pt-br/assuntos/comercio-exterior/estatisticas/base-de-dados-bruta

talhes que permitam identificar importadores (empresas ou pessoas físicas) ou detalhamentos de cada operação real de importação realizada.

Essa restrição decorre do mandamento constitucional, legal e infra legado do instituto do sigilo fiscal a que estão sujeitas as operações de comércio exterior. Este trabalho permitirá construir um modelo de aprendizado sem, no entanto, esgotar as possibilidades de *features* que podem ser utilizadas e que trariam grande desempenho nas previsões de um modelo em produção.

2.2. *Datasets* utilizados

Em uma análise inicial podemos descrever a estrutura dos *dataset* utilizados, incluindo o nome da coluna ou *feature*, sua descrição e tipo.

A tabela 1 trata do *dataset* principal o qual descreve as informações agregadas das operações de importação para o Brasil.

Tabela 1 - COLUNAS DO *DATASET* "IMP" (arquivo IMP_AAAA.CSV)

Nome da coluna	**Descrição**	**Tipo**
CO_ANO	Ano em que foi registrada a declaração de importação	Inteiro
CO_MES	Mês em que foi registrada a declaração de importação	Inteiro
CO_NCM	Código do subitem da mercadoria importada (8 dígitos), segundo a NCM (Nomenclatura Comum do Mercosul)	Inteiro
CO_UNID	Código da unidade estatística utilizada para quantificar as mercadorias importadas	Inteiro
CO_PAIS	Código do país de origem das mercadorias importadas	Inteiro
SG_UF_NCM	Sigla da unidade da federação de destino da mercadoria importada (independentemente de onde esteja localizada a sede da empresa que realizou a operação de importação)	String
CO_VIA	Código da via de transporte utilizada na operação internacional	Inteiro
CO_URF	Código da unidade da Receita Federal em que ocorreu a entrada da mercadoria no País (desembarque)	Inteiro

QT_ESTAT	Quantidade de mercadoria na unidade estatística	Inteiro
KG_LIQUIDO	Peso líquido das mercadorias importadas, em quilogramas	Inteiro
VL_FOB	Valor FOB (Free on board) das mercadorias importadas, em dóla res americanos	

As tabelas 2 a 6 descrevem os *datasets* complementares ao *dataset* principal e que serão utilizados na fase de enriquecimento de dados e análise exploratória.

Tabela 2 - COLUNAS DO *DATASET* "VIA" (arquivo VIA.CSV)

Nome da coluna	**Descrição**	**Tipo**
CO_VIA	Código da via de transporte internacional	Inteiro
NO_VIA	Nome da via de transporte internacional	String

Tabela 3 - COLUNAS DO *DATASET* "URF" (arquivo URF.CSV)

Nome da coluna	**Descrição**	**Tipo**
CO_URF	Código da unidade da Receita Federal do Brasil	Inteiro
NO_URF	Nome da unidade da Receita Federal do Brasil	String

Tabela 4 - COLUNAS DO *DATASET* "PAIS" (arquivo PAIS.CSV)

Nome da coluna	**Descrição**	**Tipo**
CO_PAIS	Código do país nas tabelas do Ministério da Economia	Inteiro
CO_PAIS_ISON3	Código do país no sistema ISON3	Inteiro
CO_PAIS_ISOA3	Código do país no sistema ISOA3	String
NO_PAIS	Nome do país em português	String

Tabela 5 - COLUNAS DO *DATASET* "NCM" (arquivo NCM.CSV)

Nome da coluna	**Descrição**	**Tipo**
CO_NCM	Código do subitem da NCM, com 8 dígitos	Inteiro
CO_UNID	Código da unidade estatística relacionada à NCM	Inteiro
CO_SH6	Código da posição no Sistema Harmonizado, com 6 dígitos	Inteiro

CO_PPE	Código na pauta de produtos exportados	Inteiro
CO_PPI	Código na pauta de produtos importados	Inteiro
CO_FAT_AGREG	Código do fator de agregação	Inteiro
CO_CUCI_ITEM	Código na classificação CUCI	Inteiro
CO_GGCE_N3	Código na classificação GGCE	Inteiro
CO_SIIT	Código na classificação SIIT	Inteiro
CO_ISIC_CLASSE	Código na classificação ISIC	Inteiro
CO_EXP_SUBSET	Código SUBSET nas exportações	Inteiro
NO_NCM_POR	Descrição do subitem da NCM em português	String
NO_NCM_ESP	Descrição do subitem da NCM em espanhol	String
NO_NCM_ING	Descrição do subitem da NCM em inglês	String

Tabela 6 - COLUNAS DO *DATASET* "NCM_SH" (arquivo NCM_SH.CSV)

Nome da coluna	**Descrição**	**Tipo**
CO_SH6	Código da subposição no Sistema Harmonizado, com 6 dígitos	Inteiro
NO_SH6_POR	Descrição da subposição no SH em português	String
NO_SH6_ESP	Descrição da subposição no SH em espanhol	String
NO_SH6_ING	Descrição da subposição no SH em inglês	String
CO_SH4	Código da posição no Sistema Harmonizado, com 4 dígitos	Inteiro
NO_SH4_POR	Descrição da posição no SH em português	String
NO_SH4_ESP	Descrição da posição no SH em espanhol	String
NO_SH4_ING	Descrição da posição no SH em inglês	String
CO_SH2	Código do capítulo no Sistema Harmonizado, com 2 dígitos	Inteiro
NO_SH2_POR	Descrição do capítulo no SH em português	String
NO_SH2_ESP	Descrição do capítulo no SH em espanhol	String
NO_SH2_ING	Descrição do capítulo no SH em inglês	String
CO_NCM_SECROM	Código no SECROM	String
NO_SEC_POR	Descrição do item no SECROM em português	String
NO_SEC_ESP	Descrição do item no SECROM em espanhol	String
NO_SEC_ING	Descrição do item no SECROM em inglês	String

3. Processamento/Tratamento de Dados

3.1. Carga dos *datasets*

O *dataset* principal foi carregado no *Jupyter Notebook* com os **dados agregados das operações de importação dos anos 2019 a 2021** a partir da leitura dos respectivos arquivos CSV.

```
#Anos utilizados para análise exploratória
ANOS = [2019, 2020, 2021]
```

Carga dos datasets

In [3]:
```
#Carregamento do dataset
start_time = time.time()

#dataset principal com dados brutos
baseComex = pd.DataFrame()
for ano in ANOS:
    baseComex = baseComex.append(pd.read_csv(".//Dados//IMP_{0}.csv".format(ano), sep = ';', encoding='latin-1'))

print("Informações do dataset principal IMP_2021")
print("\nTipo: {0}".format(type(baseComex)))
print("Dimensões: {0}".format(baseComex.shape))
print("Campos: {0}".format(baseComex.keys()))
baseComex.describe()
```

```
Informações do dataset principal IMP_2021

Tipo: <class 'pandas.core.frame.DataFrame'>
Dimensões: (4765387, 13)
Campos: Index(['CO_ANO', 'CO_MES', 'CO_NCM', 'CO_UNID', 'CO_PAIS', 'SG_UF_NCM',
       'CO_VIA', 'CO_URF', 'QT_ESTAT', 'KG_LIQUIDO', 'VL_FOB', 'VL_FRETE',
       'VL_SEGURO'],
      dtype='object')
```

Out[3]:

	CO_ANO	CO_MES	CO_NCM	CO_UNID	CO_PAIS	CO_VIA	CO_URF	QT_ESTAT	KG_LIQUIDO	VL_FOB	VL_
count	4.765387e+06	4.765387e+06	4.765387e+06	4.765387e+06	4.765387e+06	4.765387e+06	4.765387e+06	4.765387e+06	4.765387e+06	4.765387e+06	4.7653
mean	2.019814e+03	6.558142e+00	6.884258e+07	1.046850e+01	3.118397e+02	2.625731e+00	7.893985e+05	1.296454e+05	8.197801e+04	9.656024e+04	4.7325

Foram importados ao todo 4.765.387 registros com 13 colunas, ou seja, quase 5 milhões de registros foram carregados. Não há informação de registros duplicados. É possível que as *features* QT_ESTAT, KG_LIQUIDO, VL_FOB, VL_FRETE ou VL_SEGURO possuam valores zerados, porém não há ocorrência de valores ausentes.

Abaixo seguem exemplos dos registros que compõe esta base de dados:

```
In [4]: #computar tempo de carregamento
elapsed_time = time.time() - start_time
print("\nLoad Dataset: %.2f" %elapsed_time, "segundos")
baseComex.head()
```

```
Load Dataset: 13.23 segundos
```

Out[4]:

	CO_ANO	CO_MES	CO_NCM	CO_UNID	CO_PAIS	SG_UF_NCM	CO_VIA	CO_URF	QT_ESTAT	KG_LIQUIDO	VL_FOB	VL_FRETE	VL_SEGURO
0	2019	6	85439090	10	399	SC	4	817600	1	1	15	7	0
1	2019	12	56041000	10	160	SC	1	927800	453	453	1889	81	4
2	2019	2	85365090	11	87	AM	4	227700	4	0	203	4	0
3	2019	7	39269090	10	247	PR	1	917800	167	167	1477	499	0
4	2019	12	62021300	11	160	ES	4	817600	16	5	1603	54	2

Os seguintes relacionamentos podem ser feitos com os *datasets* complementares:

- coluna CO_NCM, verificada junto ao *dataset* auxiliar NCM;
- coluna CO_PAIS, verificada junto ao *dataset* auxiliar PAIS;
- coluna SG_UF_NCM, verificada junto ao *dataset* auxiliar UF;
- coluna CO_VIA, verificada junto ao *dataset* auxiliar UF;
- coluna CO_URF, verificada junto ao *dataset* auxiliar URF;

Os *datasets* complementares também foram carregados a partir de arquivos CSV. Os labels advindos destes *datasets* deixam mais inteligível a análise exploratória. Abaixo seguem os códigos *python* e exemplos dos registros que compõe cada uma das bases:

```
In [5]: #carrega domínios e dataset acessórios
base_pais = pd.read_csv('.//Dados//PAIS.csv', sep = ';' , encoding='latin-1')
print("Informações do dataset PAIS")
print("\nDimensões: {0}".format(base_pais.shape))
print("Campos: {0}".format(base_pais.keys()))
base_pais.head()
```

```
Informações do dataset PAIS

Dimensões: (281, 6)
Campos: Index(['CO_PAIS', 'CO_PAIS_ISON3', 'CO_PAIS_ISOA3', 'NO_PAIS', 'NO_PAIS_ING',
       'NO_PAIS_ESP'],
      dtype='object')
```

Out[5]:

	CO_PAIS	CO_PAIS_ISON3	CO_PAIS_ISOA3	NO_PAIS	NO_PAIS_ING	NO_PAIS_ESP
0	0	898	ZZZ	Não Definido	Not defined	No definido
1	13	4	AFG	Afeganistão	Afghanistan	Afganistan
2	15	248	ALA	Aland, Ilhas	Aland Islands	Alans, Islas
3	17	8	ALB	Albânia	Albania	Albania
4	20	724	ESP	Alboran-Perejil, Ilhas	Alboran-Perejil, Islands	Alboran-Perejil, Islas

In [6]:
```
base_via = pd.read_csv('.//Dados//VIA.csv', sep = ';', encoding='latin-1')
print("Informações do dataset VIA")
print("\nDimensões: {0}".format(base_via.shape))
print("Campos: {0}".format(base_via.keys()))
base_via
```

```
Informações do dataset VIA

Dimensões: (17, 2)
Campos: Index(['CO_VIA', 'NO_VIA'], dtype='object')
```

Out[6]:

	CO_VIA	NO_VIA
0	99	VIA DESCONHECIDA
1	13	POR REBOQUE
2	11	COURIER
3	15	VICINAL FRONTEIRICO
4	14	DUTOS
5	12	EM MAOS
6	0	VIA NAO DECLARADA
7	1	MARITIMA
8	2	FLUVIAL
9	3	LACUSTRE
10	4	AEREA
11	5	POSTAL
12	6	FERROVIARIA

In [7]:
```
base_URF = pd.read_csv('.//Dados//URF.csv', sep = ';', encoding='latin-1')
print("Informações do dataset URF")
print("\nDimensões: {0}".format(base_URF.shape))
print("Campos: {0}".format(base_URF.keys()))
base_URF.head()
```

```
Informações do dataset URF

Dimensões: (276, 2)
Campos: Index(['CO_URF', 'NO_URF'], dtype='object')
```

Out[7]:

	CO_URF	NO_URF
0	510353	0510353 - IRF ILHEUS
1	710251	0710251 - IRF CAMPOS DOS GOYTACAZES
2	1010351	1010351 - IRF SANTANA DO LIVRAMENTO
3	1017504	1017504 - IRF PORTO MAUA
4	1017505	1017505 - IRF PORTO XAVIER

In [8]:
```
base_NCM = pd.read_csv('.//Dados//NCM.csv', sep = ';', encoding='latin-1')
print("Informações do dataset NCM")
print("\nDimensões: {0}".format(base_NCM.shape))
print("Campos: {0}".format(base_NCM.keys()))
base_NCM.head()
```

```
Informações do dataset NCM

Dimensões: (13161, 14)
Campos: Index(['CO_NCM', 'CO_UNID', 'CO_SH6', 'CO_PPE', 'CO_PPI', 'CO_FAT_AGREG',
       'CO_CUCI_ITEM', 'CO_CGCE_N3', 'CO_SIIT', 'CO_ISIC_CLASSE',
       'CO_EXP_SUBSET', 'NO_NCM_POR', 'NO_NCM_ESP', 'NO_NCM_ING'],
      dtype='object')
```

Out[8]:

	CO_NCM	CO_UNID	CO_SH6	CO_PPE	CO_PPI	CO_FAT_AGREG	CO_CUCI_ITEM	CO_CGCE_N3	CO_SIIT	CO_ISIC_CLASSE	CO_EXP_SUBSET	NO_NCM_
0	29398000	10	293980	3329	3329	3	54149	240	1000	2100	1402.0	C alcal natur reprod
1	30021100	10	300211	3990	3221	3	54163	322	1000	2100	1406.0	Estoj diagnósti m (palud
2	30021211	10	300212	3990	3990	3	54163	322	1000	2100	1406.0	Antiofíd (antivene
3	30021212	10	300212	3990	3990	3	54163	322	1000	2100	1406.0	Antitet

```
In [9]: base_SH6 = pd.read_csv('.//Dados//NCM_SH.csv', sep = ';', encoding='latin-1')
print("Informações do dataset NCM_SH")
print("\nDimensões: {0}".format(base_SH6.shape))
print("Campos: {0}".format(base_SH6.keys()))
base_SH6.head()
```

```
Informações do dataset NCM_SH

Dimensões: (6308, 16)
Campos: Index(['CO_SH6', 'NO_SH6_POR', 'NO_SH6_ESP', 'NO_SH6_ING', 'CO_SH4',
       'NO_SH4_POR', 'NO_SH4_ESP', 'NO_SH4_ING', 'CO_SH2', 'NO_SH2_POR',
       'NO_SH2_ESP', 'NO_SH2_ING', 'CO_NCM_SECROM', 'NO_SEC_POR', 'NO_SEC_ESP',
       'NO_SEC_ING'],
      dtype='object')
```

Out[9]:

	CO_SH6	NO_SH6_POR	NO_SH6_ESP	NO_SH6_ING	CO_SH4	NO_SH4_POR	NO_SH4_ESP	NO_SH4_ING	CO_SH2	NO_SH2_POR	NO_SH2_ESP	NO_SH2_
0	10110	Animais vivos das espécies cavalar, asinina e ...	Caballos y asnos, reproductores de raza pura	Pure-bred breeding horses and asses	101	Cavalos, asininos e muares, vivos	Caballos, asnos, mulos y burdéganos, vivos	Live horses, asses, mules and hinnies	1	Animais vivos	Animales vivos	Live ani
1	10111	Cavalos reprodutores, de raça pura	Caballos reproductores de raza pura	Pure-bred breeding horses	101	Cavalos, asininos e muares, vivos	Caballos, asnos, mulos y burdéganos, vivos	Live horses, asses, mules and hinnies	1	Animais vivos	Animales vivos	Live ani
2	10119	Outros cavalos, vivos	Demás caballos, vivos	Other live horses	101	Cavalos, asininos e muares, vivos	Caballos, asnos, mulos y burdéganos, vivos	Live horses, asses, mules and hinnies	1	Animais vivos	Animales vivos	Live ani
3	10129	Asininos e	Asnos, mulos	Asses, mules	101	Cavalos,	Caballos, asnos, mulos	Live horses,	1	Animais	Animales	

3.2. Filtragens

Em que pese termos criado o *dataset* de 2019 a 2021 para análise exploratória, *insights* e criação do modelo, precisamos restringir o treinamento e criação do modelo para prever o país de origem dos capítulos 0 até 30 de mercadorias. Essa ação foi necessária tendo em vista memória *RAM* insuficiente para trabalhar com o imenso volume de dados (são quase 5 milhões de registros):

```
#Filtro por capítulos de 0 até 99 - filtrar para reduzir consumo de memória
CAPITULO_INICIO = 0
CAPITULO_FIM = 30
```

3.3. Remoções de colunas

Apesar do enriquecimento do *dataset* utilizamos apenas as colunas apresentadas na seção 3.5 deste trabalho para análise exploratória e apenas as colunas apresentadas na seção 3.6 para criação e treinamento do modelo. Outras análises foram feitas durante a fase de otimização do modelo na tentativa de eliminar *features* que demonstraram não agregar na criação do modelo, isto pode ser observado na seção 5.7.1.2 deste trabalho.

Criação do Modelo

Ajustes dataset (drop e encoder)

```
In [ ]: #Conforme análises pode ser necessário desconsiderar essas colunas antes do treinamento
#baseComexModelo.drop("VL_SEGURO", axis=1, inplace=True)
#baseComexModelo.drop("CO_UNID", axis=1, inplace=True)
#baseComexModelo.drop("CO_SH4", axis=1, inplace=True)
#baseComexModelo.drop("CO_SH2", axis=1, inplace=True)
#baseComexModelo.drop("CO_UNID", axis=1, inplace=True)

baseComexModelo.shape
#baseComexModelo.head()
```

3.4. Tratamento de colunas

Utilizamos o método *"fit_transform()"* da classe *"LabelEncoder"* para transformar a *feature* não numérica (SG_UF_NCM) em *feature* numérica. O seguinte código *python* demonstra esta transformação:

```
baseComexModelo.head()
```

Out[30]:

	CO_ANO	CO_MES	CO_SH2	CO_SH4	CO_NCM	CO_UNID	CO_PAIS	SG_UF_NCM	CO_VIA	CO_URF	QT_ESTAT	KG_LIQUIDO	VL_FOB	VL_FRETE
162606	2019	12	30	3006	30067000	10	399	SP	4	817600	469	469	4473	2227
162607	2019	4	30	3006	30067000	10	399	SP	4	817600	310	310	2999	1643
162608	2019	2	30	3006	30067000	10	399	SP	4	817600	0	0	7	3
162609	2019	11	30	3006	30067000	10	399	SP	4	817600	14	14	158	89
162610	2019	7	30	3006	30067000	10	399	SP	4	817600	6	6	62	85

```
In [31]: #encoder da coluna label
le = LabelEncoder()
baseComexModelo["SG_UF_NCM"] = le.fit_transform(baseComexModelo["SG_UF_NCM"])
baseComexModelo.head()
```

Out[31]:

	CO_ANO	CO_MES	CO_SH2	CO_SH4	CO_NCM	CO_UNID	CO_PAIS	SG_UF_NCM	CO_VIA	CO_URF	QT_ESTAT	KG_LIQUIDO	VL_FOB	VL_FRETE
162606	2019	12	30	3006	30067000	10	399	27	4	817600	469	469	4473	2227
162607	2019	4	30	3006	30067000	10	399	27	4	817600	310	310	2999	1643
162608	2019	2	30	3006	30067000	10	399	27	4	817600	0	0	7	3
162609	2019	11	30	3006	30067000	10	399	27	4	817600	14	14	158	89
162610	2019	7	30	3006	30067000	10	399	27	4	817600	6	6	62	85

3.5. *Dataset* análise exploratória

O *dataset* principal foi enriquecido com os *datasets* auxiliares de forma que **totalizaram 4.765.387 registros com 21 colunas utilizados na análise exploratória**, ou seja, quase 5 milhões de registros foram carregados. Realizamos um filtro de colunas sendo que das 47 colunas apenas 21 colunas foram utilizadas para análise exploratória. Consideramos

as demais colunas repetitivas, por exemplo, as mesmas informações em línguas diferentes, códigos equivalentes mas de outros padrões de identificação, dentre outros.

Enriquecimento de dataset

```
In [29]: baseComexEnriquecida.shape

Out[29]: (4765387, 47)
```

```
In [25]: #dataset enriquecido com domínios e dataset acessórios
baseComexEnriquecida = pd.merge(baseComex, base_pais, on="CO_PAIS", how="inner")
baseComexEnriquecida = pd.merge(baseComexEnriquecida, base_via, on="CO_VIA", how="inner")
baseComexEnriquecida = pd.merge(baseComexEnriquecida, base_URF, on="CO_URF", how="inner")
baseComexEnriquecida = pd.merge(baseComexEnriquecida, base_NCM, on=["CO_NCM","CO_UNID"], how="inner")
baseComexEnriquecida = pd.merge(baseComexEnriquecida, base_SH6, on="CO_SH6", how="inner")

#Filtrar dataset para manter apenas colunas para análise exploratória
lista = ['CO_ANO', 'CO_MES', 'CO_SH2', 'NO_SH2_POR', 'CO_SH4', 'NO_SH4_POR', 'CO_SH6',
         'CO_NCM', 'CO_UNID', 'CO_PAIS', 'NO_PAIS', 'SG_UF_NCM','CO_VIA','NO_VIA',
         'CO_URF', 'NO_URF', 'QT_ESTAT', 'KG_LIQUIDO', 'VL_FOB', 'VL_FRETE', 'VL_SEGURO']

baseComexEnriquecidaAmostra = baseComexEnriquecida[lista]
print("Informações do dataset principal enriquecido para análise exploratória")
#print("\nFiltros por capítulo SH2: {0} a {1}".format(CAPITULO_INICIO, CAPITULO_FIM))
print("Dimensões: {0}".format(baseComexEnriquecidaAmostra.shape))
print("Campos: {0}".format(baseComexEnriquecidaAmostra.keys()))

Informações do dataset principal enriquecido para análise exploratória
Dimensões: (4765387, 21)
Campos: Index(['CO_ANO', 'CO_MES', 'CO_SH2', 'NO_SH2_POR', 'CO_SH4', 'NO_SH4_POR',
       'CO_SH6', 'CO_NCM', 'CO_UNID', 'CO_PAIS', 'NO_PAIS', 'SG_UF_NCM',
       'CO_VIA', 'NO_VIA', 'CO_URF', 'NO_URF', 'QT_ESTAT', 'KG_LIQUIDO',
       'VL_FOB', 'VL_FRETE', 'VL_SEGURO'],
      dtype='object')
```

3.6. *Dataset* para treinamento dos modelos

Conforme mencionado na seção 3.2, realizamos a criação de um *dataset* para treinamento, validação e comparação de modelos de predição do países de origem de mercadorias dos capítulos 0 a 30. Esta restrição tornou-se necessária pois não havia memória *RAM* suficiente para criação e otimização dos modelos considerando a base total, ou seja, quase 5 milhões de registros.

Após a aplicação deste filtro, **o *dataset* utilizado para criação dos estimadores contou com 477.396 registros e 17 colunas**.

In [26]:
```
#Filtrar dataset para manter apenas colunas para criação do modelo
lista = ['CO_ANO', 'CO_MES', 'CO_SH2', 'CO_SH4', 'CO_NCM',
         'CO_UNID', 'CO_PAIS', 'SG_UF_NCM','CO_VIA', 'CO_URF',
         'QT_ESTAT', 'KG_LIQUIDO', 'VL_FOB', 'VL_FRETE', 'VL_SEGURO']
baseComexModelo = baseComexEnriquecida[lista]

#Filtrando dataset para selecionar um intervalo de mercadorias por capitulo
baseComexModelo = baseComexModelo.loc[
                                (baseComexModelo['CO_SH2'] >= CAPITULO_INICIO)
                                & (baseComexModelo['CO_SH2'] <= CAPITULO_FIM)]

#Filtrar dataset para criar modelo - reduzir consumo de memória
#Criar colunas derivadas no dataset do modelo
baseComexModelo["VL_FRETE/KG_LIQUIDO"]= baseComexModelo["VL_FRETE"]/ baseComexModelo["KG_LIQUIDO"]
baseComexModelo["VL_FRETE/KG_LIQUIDO"].replace(np.inf, 0, inplace=True)
baseComexModelo["VL_FRETE/KG_LIQUIDO"].replace(np.nan, 0, inplace=True)

baseComexModelo["VL_FOB/KG_LIQUIDO"]= baseComexModelo["VL_FOB"]/ baseComexModelo["KG_LIQUIDO"]
baseComexModelo["VL_FOB/KG_LIQUIDO"].replace(np.inf, 0, inplace=True)
baseComexModelo["VL_FOB/KG_LIQUIDO"].replace(np.nan, 0, inplace=True)

print("\nInformações do dataset principal para criação do modelo")
print("\nFiltros por capítulo SH2: {0} a {1}".format(CAPITULO_INICIO, CAPITULO_FIM))
print("Dimensões: {0}".format(baseComexModelo.shape))
print("Campos: {0}".format(baseComexModelo.keys()))
```

```
Informações do dataset principal para criação do modelo

Filtros por capítulo SH2: 0 a 30
Dimensões: (477396, 17)
Campos: Index(['CO_ANO', 'CO_MES', 'CO_SH2', 'CO_SH4', 'CO_NCM', 'CO_UNID', 'CO_PAIS',
       'SG_UF_NCM', 'CO_VIA', 'CO_URF', 'QT_ESTAT', 'KG_LIQUIDO', 'VL_FOB',
       'VL_FRETE', 'VL_SEGURO', 'VL_FRETE/KG_LIQUIDO', 'VL_FOB/KG_LIQUIDO'],
      dtype='object')
```

É importante ressaltar que as 17 colunas mantidas são especialmente relevantes em termos de negócio para o propósito de identificação do país de origem. Tais colunas englobam dados categóricos (NCM, país de origem, unidade da federação de destino dos produtos, via de transporte, unidade da Receita Federal de entrada da mercadoria, mês e ano) e valores escalares (Quantidade estatística de mercadoria, Peso líquido, Valor da mercadoria, valor do frete, valor do seguro).

Criamos 2 colunas derivadas, a “VL_FRETE/KG_LIQUIDO” e a “VL_FOB/KG_LIQUIDO” pois contém um relacionamento importante para fins de reconhecimento de padrões. Na seção 5.7.1.2 deste trabalho é possível verificar que a tais *features* os modelos atribuíram grande valor de *“feature_importances_”*.

4. Análise e Exploração dos Dados

Importante mencionar que utilizamos um *dataset* criado especialmente para a análise e exploração de dados. Este *dataset* foi enriquecido com outros *datasets* complementares para inclusão dos *labels* de vários códigos. Dessa forma, a análise de negócio pode se dar de maneira mais intuitiva.

4.1. Análise dos dados categóricos

Com o objetivo de conhecer e apresentar visualmente o peso das variáveis que compõe o *dataset* criamos gráficos que nos permitem a comparação. Criamos *dataset* agregando as informações conforme cada tipo de gráfico a ser analisado. O código abaixo foi adotado para produzir os gráficos a seguir:

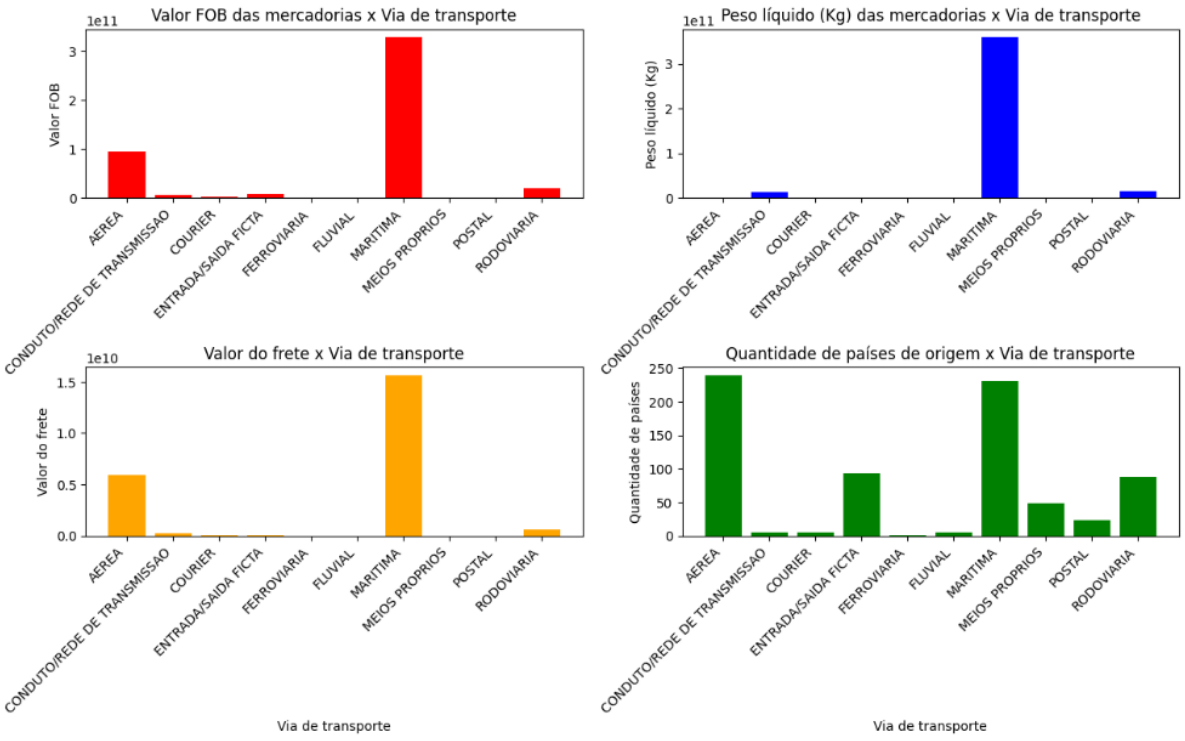
Análise exploratória

Gráficos

In [12]:
```
df_graf_via_fob = baseComexEnriquecidaAmostra.groupby(["NO_VIA"]).agg({"VL_FOB":['sum']})
df_graf_via_kg = baseComexEnriquecidaAmostra.groupby(["NO_VIA"]).agg({"KG_LIQUIDO":['sum']})
df_graf_via_frete = baseComexEnriquecidaAmostra.groupby(["NO_VIA"]).agg({"VL_FRETE":['sum']})
df_graf_via_pais = baseComexEnriquecidaAmostra.groupby(["NO_VIA"]).agg({"NO_PAIS":['nunique']})

#plt.style.use(style ="seaborn")
plt.rcdefaults()

# Cada plot terá o mesmo tamanho de figuras
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15,7))

x1 = df_graf_via_fob.index.tolist()
x2 = df_graf_via_kg.index.tolist()
x3 = df_graf_via_frete.index.tolist()
x4 = df_graf_via_pais.index.tolist()

y1 = df_graf_via_fob['VL_FOB'][ 'sum'].tolist()
y2 = df_graf_via_kg['KG_LIQUIDO']['sum'].tolist()
y3 = df_graf_via_frete['VL_FRETE']['sum'].tolist()
y4 = df_graf_via_pais['NO_PAIS']['nunique'].tolist()

ax1.bar(x1, y1,color='red')
ax2.bar(x2, y2,color='blue')
ax3.bar(x3, y3,color='orange')
ax4.bar(x4, y4,color='green')

ax1.set(title="Valor FOB das mercadorias x Via de transporte", xlabel="Via de transporte", ylabel="Valor FOB")
ax2.set(title="Peso líquido (Kg) das mercadorias x Via de transporte", xlabel="Via de transporte", ylabel="Peso líquido (Kg)")
ax3.set(title="Valor do frete x Via de transporte", xlabel="Via de transporte", ylabel="Valor do frete")
ax4.set(title="Quantidade de países de origem x Via de transporte", xlabel="Via de transporte", ylabel="Quantidade de países")

ax1.set_xticklabels(x1, rotation=45, ha='right')
ax2.set_xticklabels(x2, rotation=45, ha='right')
ax3.set_xticklabels(x3, rotation=45, ha='right')
ax4.set_xticklabels(x4, rotation=45, ha='right')

plt.subplots_adjust(wspace=0.2, hspace=1)
#plt.legend()
plt.show()
```

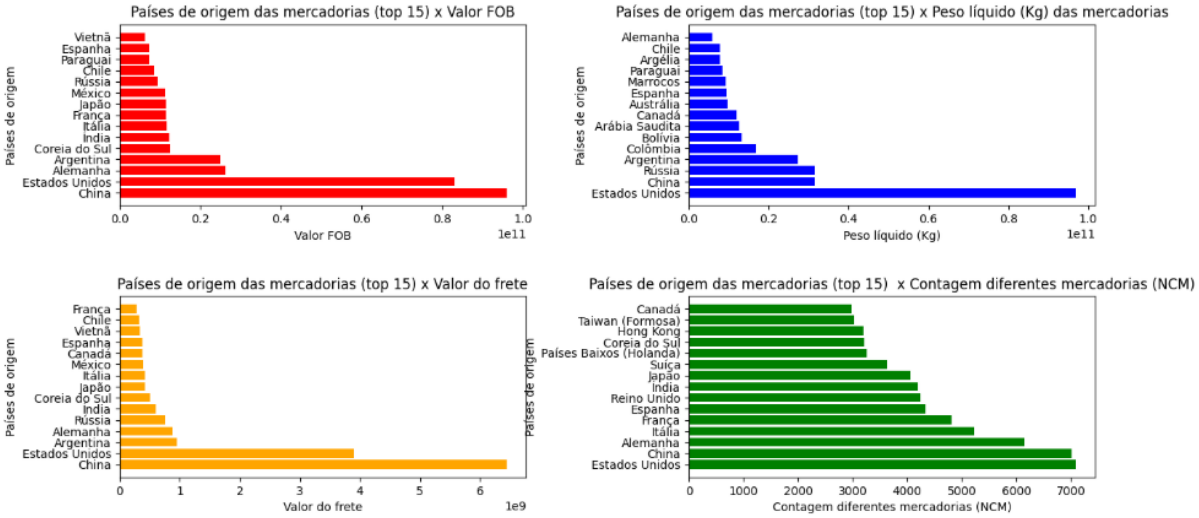
Tomando por base a Via de Transporte (*feature* CO_VIA) verificamos sua relevância em função do Valor FOB (*feature* VL_FOB), Peso líquido (*feature* KG_LIQUIDO), Valor do frete (*feature* VL_FRETE) e Quantidade de países de origem (*feature* CO_PAIS).

A via de transporte marítima é a mais relevante por Valor FOB, Peso líquido ou Valor do frete seguida pela via aérea. Em relação à quantidade de países de origem de mercadorias podemos constatar que a via aérea é a mais diversificada seguida pela via marítima. Em terceiro lugar temos a via de transporte rodoviária. Importante destacar que a via entrada/saída ficta é utilizada em situações permitidas pela legislação em que não há efetiva movimentação de cargas tendo em vista já terem ocorrido anteriormente.



Continuando a exploração do *dataset* podemos visualizar os 15 países de origem mais relevantes em função do Valor FOB (*feature* VL_FOB), Peso líquido (*feature* KG_LIQUIDO), Valor do frete (*feature* VL_FRETE) e Contagem de diferentes NCMs (*feature* CO_NCM).

As diferenças comparativas podem ser decorrentes de muitos fatores. Por exemplo, Valor do frete está relacionado à distância geográfica e também a demanda de mercado; Valor FOB está relacionado ao valor de venda da mercadoria e flutuações de mercado; Peso líquido diretamente relacionado ao tipo de produto comercializado revelando valores altos na comercialização de produtos primários ou máquinas muito pesadas; e Contagem de diferentes NCMs possui relação com a variedade de mercadorias que determinado país exporta para o Brasil.



Criamos 2 nuvens de palavras para verificar a relevância das unidades da RFB de desembarque de mercadorias (CO_URF) em função do valor das mercadorias (VL_FOB) e também em função do peso líquido (KG_LIQUIDO).

Wordcloud

In [16]:
```
#CRIA DICT para wordcloud
df_graf_urf_fob = baseComexEnriquecidaAmostra.groupby(["NO_URF"]).agg({"VL_FOB":['sum']})
df_graf_urf_kg = baseComexEnriquecidaAmostra.groupby(["NO_URF"]).agg({"KG_LIQUIDO":['sum']})

df_graf_urf_fob.columns = ["VL_FOB_sum"]
df_graf_urf_kg.columns = ["KG_LIQUIDO_sum"]

df_graf_urf_fob["VL_FOB_sum"].apply(int)
data_urf_fob = df_graf_urf_fob.to_dict('dict')
data_urf_fob = data_urf_fob['VL_FOB_sum']

df_graf_urf_kg["KG_LIQUIDO_sum"].apply(int)
data_urf_kg = df_graf_urf_kg.to_dict('dict')
data_urf_kg = data_urf_kg['KG_LIQUIDO_sum']
```

In [17]:
```
# gerar uma wordcloud
wordcloud = WordCloud(#stopwords=STOPWORDS,
                      background_color="black",
                      width=1600, height=800).generate_from_frequencies(data_urf_fob)

# mostrar a imagem final
fig, ax = plt.subplots(figsize=(20,10))
ax.imshow(wordcloud, interpolation='bilinear')
ax.set_axis_off()

plt.imshow(wordcloud)
```

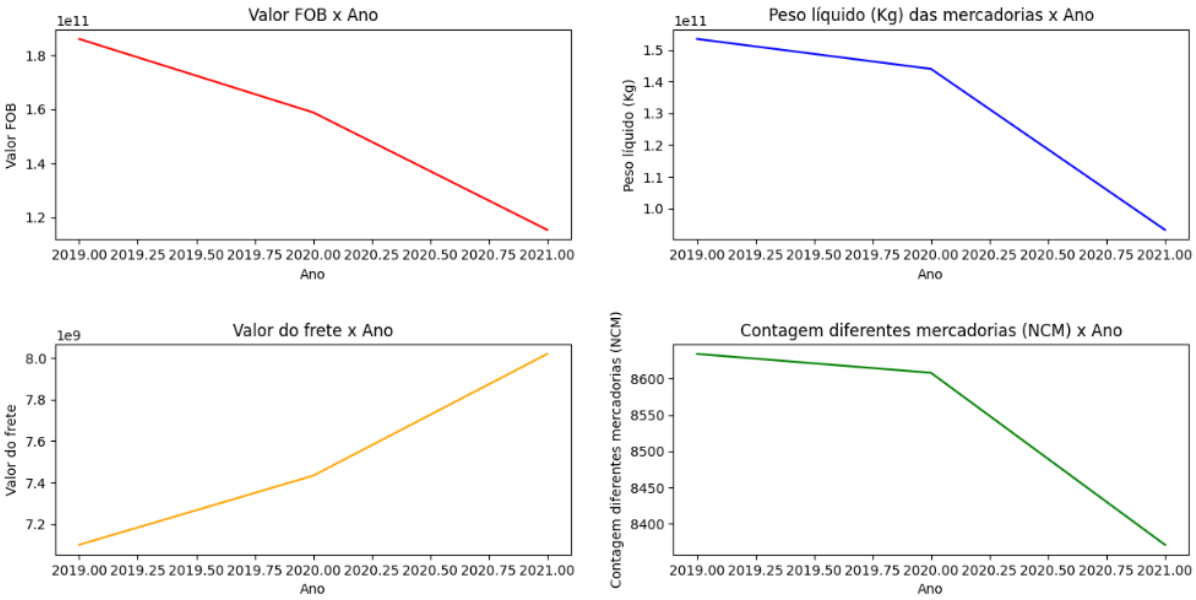
Out[17]: <matplotlib.figure.AxesImage at 0x7f4e94e53a58>

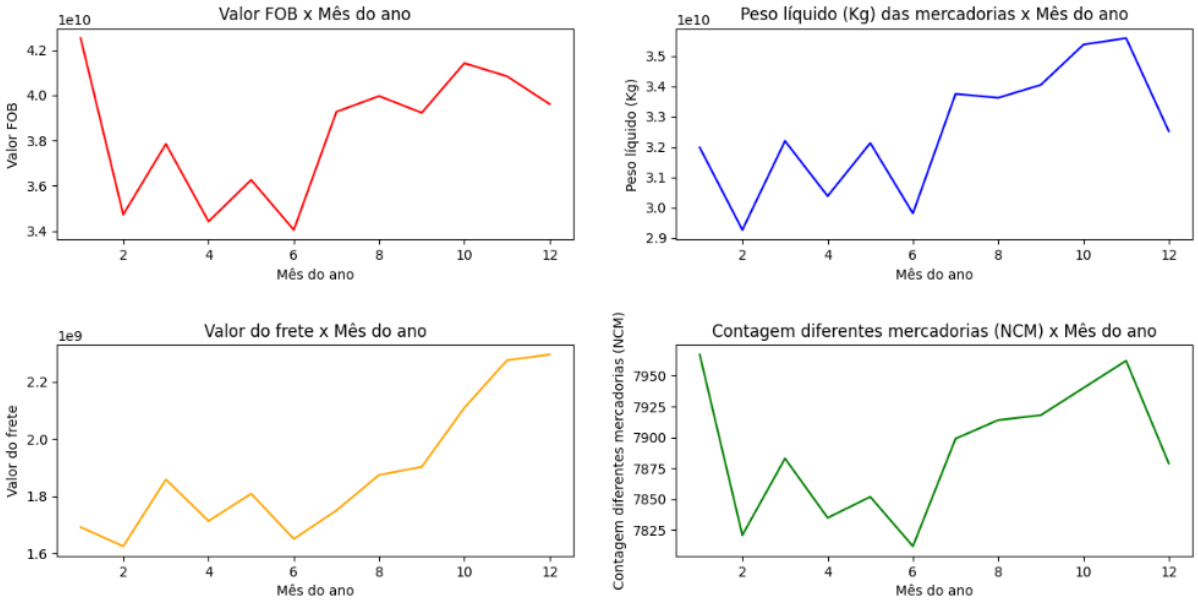
Out[17]: <matplotlib.figure.AxesImage at 0x7f4e94e53a58>

Podemos confirmar que o Porto de Santos tem amplo destaque em ambas análises entretanto a relevância dos portos é diferente quando comparamos peso líquido ao invés do valor. Estes padrões podem ser especialmente úteis para que o algoritmo ML construa uma função com boa predição.

4.2. Análise temporal

Abaixo podemos visualizar as sazonalidades de ano e mês do ano em função do Valor FOB (*feature* VL_FOB), Peso líquido (*feature* KG_LIQUIDO), Valor do frete (*feature* VL_FRETE) e Contagem de diferentes NCMs (*feature* CO_NCM).





4.3. Influência das variáveis sobre o país de origem

Elaboramos um gráfico *pairplot* para comparar a maioria das variáveis entre si buscando identificar padrões que possam auxiliar a identificação do país de origem. Abaixo segue o código *python* gerador desse gráfico:

Pairplot

```
In [22]: df_pair_plot = baseComexEnriquecidaAmostra.groupby(['CO_SH2', 'CO_PAIS', 'SG_UF_NCM', 'CO_VIA'], as_index=False).agg({'KG_LIQUID(
               'VL_FOB':['sum'], 'VL_FRETE':['sum']})

         df_pair_plot.columns = ['CO_SH2', 'CO_PAIS', 'SG_UF_NCM', 'CO_VIA', 'KG_LIQUIDO_sum', 'VL_FOB_sum', 'VL_FRETE_sum']
         df_pair_plot.shape

Out[22]: (61510, 7)

In [23]: #análise
         plt.figure(figsize=(36,20))

         sns.pairplot(df_pair_plot[['CO_SH2', 'CO_PAIS', 'SG_UF_NCM', 'CO_VIA', 'KG_LIQUIDO_sum',
               'VL_FOB_sum', 'VL_FRETE_sum']], hue="CO_SH2", diag_kind="hist", palette="bright")
         plt.show()

         <Figure size 3600x2000 with 0 Axes>
```

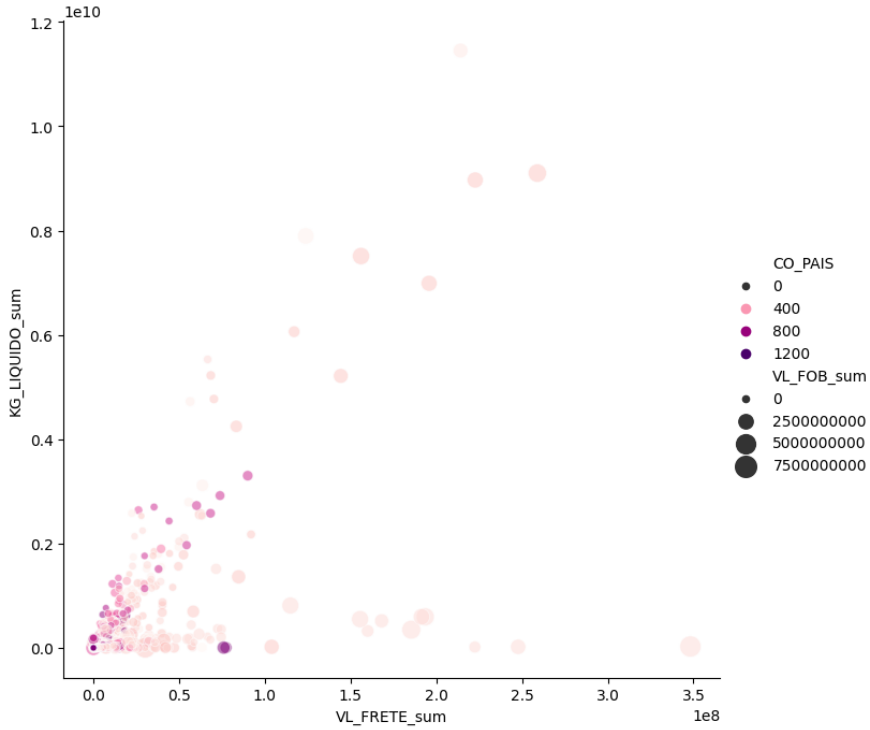

CO_SH2
CO_PAIS
CO_VIA
KG_LIQUIDO_sum
VL_FOB_sum
VL_FRETE_sum
1e10
1e9
1e8

Podemos observar que a legenda está por código SH2, o que significa que é o capítulo de classificação de mercadoria. Este é um código macro de 2 dígitos.

Por meio dessa legenda podemos identificar, por exemplo, a relação entre VL_FOB(valor da mercadoria) e KG_LIQUIDO(Peso líquido). É possível identificar mercadorias com alto valor e baixo peso líquido, pontos verdes e vermelhos, embora a tendência seja de quando maior o peso líquido também maior é o valor da mercadoria.

Esta mesma relação diretamente proporcional pode ser encontrada entre VL_FRETE (valor do frete) e KG_LIQUIDO (peso líquido). Isso é natural pois geralmente o peso é fator determinante do valor frete. Segue um *relplot* mais detalhado:

```
In [24]: sns.relplot(x='VL_FRETE_sum', y='KG_LIQUIDO_sum', data = df_pair_plot, kind='scatter', size ='VL_FOB_sum', hue ='CO_PAIS',
                     sizes =(20,200), palette = 'RdPu', alpha=.5, height = 7)
         plt.show()
```



5. Criação de Modelos de Machine Learning

5.1. Escolha dos algoritmos

O objetivo do trabalho é desenvolver um preditor para determinar, a partir de diversos parâmetros de importação, qual será o país de origem da mercadoria mais provável. Trata-se, portanto, de um problema de classificação. Foram escolhidos 5 algoritmos classificadores para análise e comparação: o **DecisionTreeClassifier,** o **RandomForestClassifier,** o **Naive Bayes** (**CategoricalNB**) e **KNeighborsClassifier.**

5.2. Escolha das ferramentas

O projeto foi desenvolvido no *google Colab* criando um *Notebook* em *python*. O treinamento e testes foram realizados utilizando ambiente *Jupyter Notebook* em máquina local tendo em vista limitação de *RAM* no *Colab*.

Além da linguagem *Python*, utilizamos suas principais bibliotecas (*numpy*, *pandas*, *seaborn* e, especificamente para os modelos de aprendizado de máquina, a biblioteca *scikit-learn* (*sklearn*), e para visualização, as bibliotecas *matplotlib* e *wordcloud*.

Jupyter Notebook[6]

Jupyter Notebook ou caderno *Jupyter* é uma ferramenta criada para se trabalhar com programação literária. Neste paradigma de programação há uma intersecção entre a codificação e a documentação em forma de narrativa, ao invés de manipulá-los como elementos independentes.

[6] https://jupyter.org/

Pandas[7]

Fornece uma estrutura de dados e funções avançadas projetada para o trabalho com grandes quantidades de dados de forma mais rápida.

NumPy[8]

Seu nome se refere a *Numerical Python* e permite calcular com eficiência funções matemáticas de grandes *arrays* (multidimensionais). O *NumPy* um pacote básico de computação científica.

Matplotlib[9]

Biblioteca popular para a produção de gráficos e outras visualizações 2D. *Matplotlib* se integra muito bem com *Python* proporcionando criar visualizações estáticas, animadas ou interativas para a exploração de dados em *Python*.

Scikit-Learn[10]

O scikit-learn é uma biblioteca de aprendizado de máquina de código aberto para a linguagem de programação *Python*. Ela inclui diversos algoritmos utilizados para resolver problemas de classificação, regressão e agrupamento.

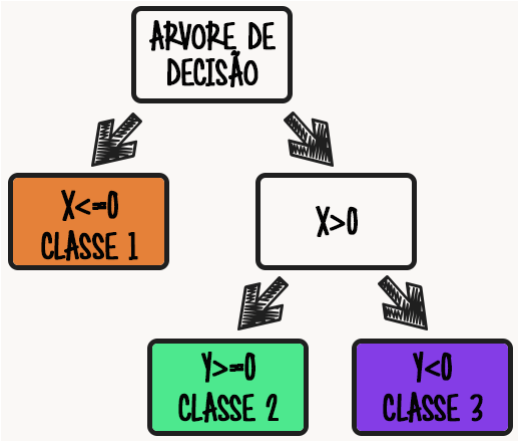
[7] https://pandas.pydata.org/
[8] https://numpy.org/
[9] https://matplotlib.org/
[10] https://scikit-learn.org/stable/

5.3. Modelos

Como bem definido pela citação de Mike Roberts no livro *Introducing Data Science*: *"O aprendizado de máquina é o processo pelo qual um computador pode trabalhar com mais precisão à medida que recolhe e aprende com os dados fornecidos"*[11]. Em *Data Science* esse recurso pode deixar explícito informações que passariam desapercebidas pelo olhar humano ao executar a análise dos dados.

5.3.1. *DecisionTreeClassifier*[12]

Um classificador de árvore de decisão que funciona realmente como uma árvore. Cada galho da árvore se divide em dois até chegar às folhas, que são as decisões da árvore. Vejamos a figura a seguir:



No exemplo figurado acima, temos duas variáveis de entradas e três classes possíveis como saída. Primeiramente o algoritmo verifica a variável **X**, caso ela seja **menor ou igual a 0** a saída é **CLASSE 1**. Se a variável **X** for **maior que 0**, ele verifica a segunda variável, caso **Y** seja **maior ou igual a 0** a saída é **CLASSE 2**, mas se **Y** for **menor que 0** a saída será **CLASSE 3**. De forma simples é este o funcionamento do algoritmo.

---

[11] CIELEN, Davy; MEYSMAN, Arno D. B.; ALI, Mohamed. Introducing Data Science: Big Data, Machine Learning, and more using Python Tools, 1. pub. Shelter Island - Estados Unidos: Editor Manning Publication, 2016

[12] sklearn.tree.DecisionTreeClassifier — scikit-learn 1.0.2 documentation e Decision Tree e Random Forest (carlosbaia.com)

Utilizado em problemas de classificação e regressão.

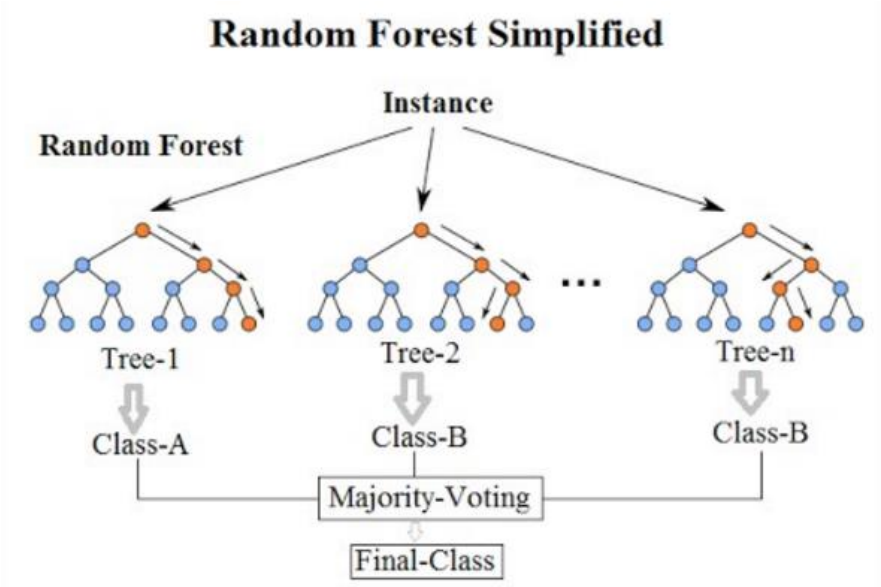
5.3.2. *RandomForestClassifier*[13]

O algoritmo *Random Forest* é um tipo de *ensemble learning*, método que gera muitos classificadores e combina o seu resultado.

A palavra *random* significa aleatório e denota o comportamento do algoritmo **selecionar aleatoriamente algumas amostras dos dados de treino** e também subconjuntos de *features* e montar uma "mini" árvores de decisão. *Forest* significa floresta, já que são geradas várias árvores de decisão.

No caso do *Random Forest*, ele gera vários *decision trees*, cada um com suas particularidades e combinada o resultado da classificação de todos eles. Essa combinação de modelos, torna ele um algoritmo muito mais poderoso do que o *Decision Tree*.

Utilizado em problemas de classificação e regressão



[13] sklearn.ensemble.RandomForestClassifier — scikit-learn 1.0.2 documentation e Decision Tree e Random Forest (carlosbaia.com) e Random forest classifier from scratch in Python - Lior Sinai

5.3.3. *Naive Bayes (CategoricalNB)*[14]

O *CategoricalNB* é um classificador probabilístico baseado no "**Teorema de Bayes**". Baseia-se na probabilidade de cada evento ocorrer, desconsiderando a correlação entre *features*. Por ter uma parte matemática relativamente simples, possui um bom desempenho e precisa de poucas observações para ter uma boa acurácia.

Utilizado em problemas de classificação

5.3.4. *KNeighborsClassifier*[15]

Classificador que implementa o algoritmo *K Nearest Neighbors* (KNN). O algoritmo do KNN (como é mais conhecido), ou k-vizinhos mais próximos se traduzido, ele busca estimar o valor de uma classe observando os pontos mais próximos. Por padrão essa distância é calculada a **distância euclidiana**, mas pode também ser calculadas com outras medidas. Vejamos a figura a seguir:

[14] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.CategoricalNB.html e Modelos de Predição | Naive Bayes | by Ana Laura Moraes Tamais | Turing Talks | Medium

[15] sklearn.neighbors.KNeighborsClassifier — scikit-learn 1.0.2 documentation e Classificação com scikit-learn - Dados ao Cubo - Data Science - Python, em 16 de fevereiro

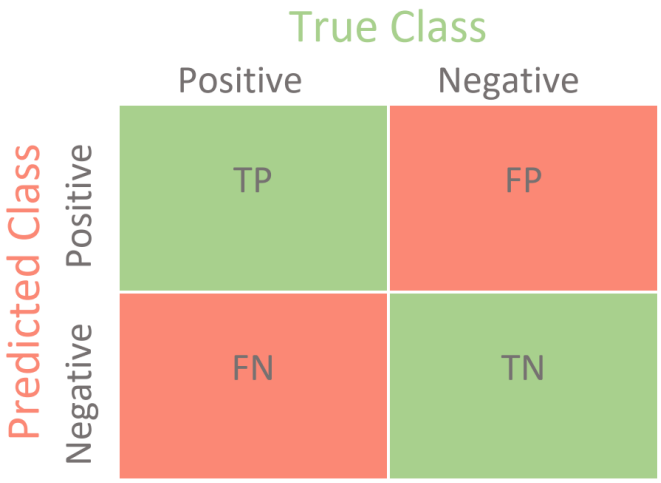
A "**?**" representa o nosso **NOVO EXEMPLO** que vamos classificar, **quadrados azuis são a CLASSE1 e triângulos vermelhos são a CLASSE2**. Então temos a **DISTÂNCIA** para os pontos mais próximos e o **NÚMERO DE VIZINHOS** que vão ser levados em consideração para classificar. Caso o algoritmo seja configurado como **K=1** o **NOVO EXEMPLO** seria classificado como **quadrado azul**, mas se for configurado como **K=3** o **NOVO EXEMPLO** seria classificado como **triângulo vermelho**. Portanto o hiperparâmetro **K** deve ser cuidadosamente escolhido e testado.

Utilizado em problemas de classificação e regressão

5.4. Métricas de avaliação de performance

5.4.1. Matriz de confusão[16]

A matriz de confusão é uma tabela que representa os acertos e erros de uma classificação. Dessa forma é possível fazer cálculos de performance através destes resultados obtidos como vamos ver a seguir:



[16] Consulta no website: Entendendo o que é Matriz de Confusão com Python | by Emanuel G de Souza | Data Hackers | Medium e Confusion Matrix for Your Multi-Class Machine Learning Model | by Joydwip Mohajon | Towards Data Science, em 15 de fevereiro de 2022

Verdadeiro positivo (*true positive* — TP): ocorre quando no conjunto real, a classe que estamos buscando foi prevista corretamente. Por exemplo, quando a mulher está grávida e o modelo previu corretamente que ela está grávida.

Falso positivo (*false positive* — FP): ocorre quando no conjunto real, a classe que estamos buscando prever foi prevista incorretamente. Exemplo: a mulher não está grávida, mas o modelo disse que ela está.

Falso verdadeiro (*true negative* — TN): ocorre quando no conjunto real, a classe que não estamos buscando prever foi prevista corretamente. Exemplo: a mulher não estava grávida, e o modelo previu corretamente que ela não está.

Falso negativo (*false negative* — FN): ocorre quando no conjunto real, a classe que não estamos buscando prever foi prevista incorretamente. Por exemplo, quando a mulher está grávida e o modelo previu incorretamente que ela não está grávida.

Entendido os conceitos da matriz de confusão e seus possíveis erros podemos seguir para os cálculos de performance.

Com a finalidade de avaliar modelos de classificação a matriz de confusão é um instrumento muito importante. Dessa forma, vamos entender cada um desses avaliadores de desempenho dos classificadores.

5.4.2. Acurácia[17]

Acurácia – A acurácia é a proporção das **observações classificadas corretas** dentre o **total de observações**.

$$accurary = \frac{TP + TN}{TP + FP + TN + FN} = \frac{predições\ corretas}{todas\ as\ predições}$$

5.4.3. Precisão

Precisão – A precisão é a proporção das **observações positivas classificadas corretamente** dentre as observações **positivas que foram previstas**.

$$precision = \frac{TP}{TP + FP}$$

5.4.4. Recall

Recall – O *recall* é a proporção das **observações positivas classificadas corretamente** dentre as **observações positivas verdadeiras**.

$$recall = \frac{TP}{TP + FN}$$

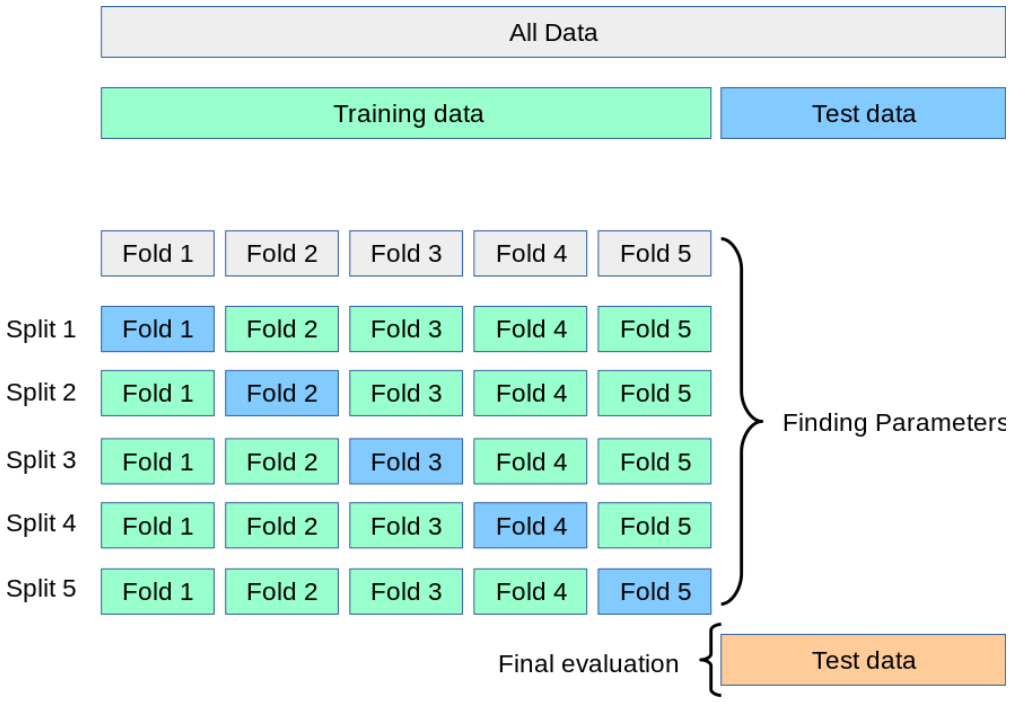
5.4.5. F1 Score

F1 Score – O F1 Score é a média harmônica entre precisão e *recall.*

$$2 * \frac{precision * recall}{precision + recall}$$

[17] Consulta no website: Entendendo o que é Matriz de Confusão com Python | by Emanuel G de Souza | Data Hackers | Medium e Confusion Matrix for Your Multi-Class Machine Learning Model | by Joydwip Mohajon | Towards Data Science, em 15 de fevereiro de 2022

5.4.6. Validação cruzada[18]



Nossos dados de treino são divididos em, por exemplo, 5 partes (*folds*). O primeiro modelo é treinado utilizando os *folds* 2, 3, 4 e 5 e o teste é feito sob o *fold* 1, o segundo modelo é treinado utilizando os *folds* 1, 3, 4 e 5 e testado no *fold* 2, e assim sucessivamente.

Nesse caso, teremos cinco métricas diferentes, ou seja, vamos ter uma noção melhor da performance do modelo e sem usar os dados de teste.

Para fazer a validação cruzada utilizamos a função "*cross_val_score*", que faz todo esse processo de forma automatizada.

---

[18] Consulta no website: Como avaliar seu modelo de classificação | by Marcelo Randolfo | Data Hackers | Medium e 3.1. Validação cruzada: avaliação do desempenho estimador — documentação scikit-learn 1.0.2, em 15 de fevereiro de 2022

5.5. Otimização

Além disso, utilizamos as funções "*GridSearchCV*" e "*RandomizedSearchCV*" para elaborar outras versões de teste com diferentes hiperparâmetros para o mesmo *dataset*.

5.5.1. GridSearchCV[19]

Vimos durante os exemplos que todos os algoritmos possuem diversos parâmetros. Os melhores valores para esses parâmetros mudam conforme os dados mudam, conforme você adiciona ou tira *features* e conforme muda os outros parâmetros também.

Mas então, como definir quais os melhores valores? Uma das técnicas que pode ser utilizada para isso é o *Grid Search Cross Validation*. É possível passar como parâmetro uma lista de possíveis valores de hiperparâmetros e o tipo de score que deve ser utilizado para medir a eficiência do modelo. Essa função vai rodar o *Cross Validation* com todas as possíveis combinações e no final vai te dizer qual a combinação apresentou o melhor score.

5.5.2. RandomizedSearchCV[20]

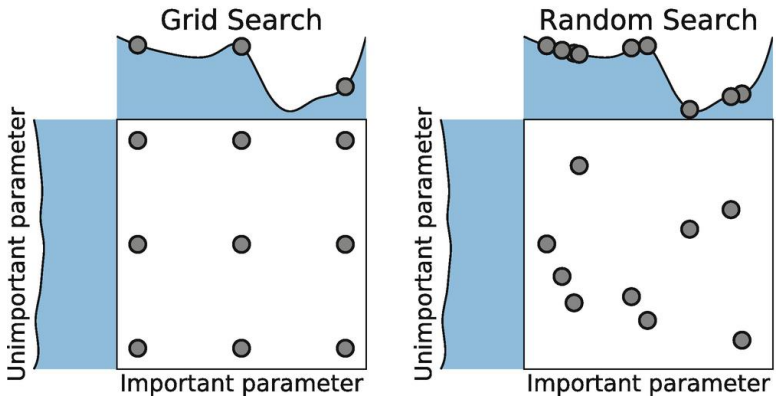
Este algoritmo é muito parecido com o *Grid Search*, exceto por um motivo: ao invés de testar todas as combinações na vizinhança, **o *Random Search*, ou busca aleatória, testa combinações aleatórias de hiperparâmetros**, conforme um número especificado de amostras a tirar.

Ele é uma alternativa para o *Grid Search* quando o conjunto de dados é muito grande, ou há um número muito grande de hiperparâmetros para otimizar.

---

[19] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[20] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html



5.6. Particionamento em treino e teste

Primeiramente definimos o *target*, nossa variável alvo multiclasse, para "CO_PAIS" que é o código do país de origem da mercadoria importada. Posteriormente, dividimos o *dataset* em dados de treino, usados no treinamento do modelo de ML, e dados de teste, usados para apurar as métricas de performance do modelo treinado. O tamanho da base de teste foi de 15% do *dataset*, conforme o seguinte trecho do script:

Particionamento do dataset

```
In [39]: # Particiona a base de dados
label_target = "CO_PAIS"
X = baseComexModelo.drop(label_target, axis=1)
y = baseComexModelo[label_target]

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.15)

In [40]: #Quantidade países (target)
unique= np.unique(y)
print('Contagem de target ({0}): {1}'.format(label_target, unique.size))

Contagem de target (CO_PAIS): 176
```

É possível apurar que há 176 possibilidades de *target* no *dataset* objeto de análise. Isso demonstra os desafios de se conseguir um modelo eficaz com uma quantidade pequena de *features* disponíveis.

5.7. Criação e comparação de modelos

Os quatro modelos foram preparados com hiperparâmetros típicos. Após a execução da VERSAO_DO_TESTE = "1.0" e, principalmente, da utilização de *insights* da análise exploratória então realizamos a otimização dos hiperparâmetros e *features.*

5.7.1. *DecisionTreeClassifier*

Instanciamos cada modelo com hiperparâmetros default criando o *"ml_estimador"* que implementa o estimador que fará nossa classificação.

Após a criação do estimador então utilizamos o método *"fit"* para efetuar o treinamento do estimador. O conjunto de treinamento, as *features* e *target*, são passados por parâmetro

Importante ressaltar que todos os modelos seguem os padrões de instanciação, treinamento e predição apresentados, conforme trecho do script abaixo:

DecisionTreeClassifier

```
In [138]: if label_estimador == "DecisionTreeClassifier":
            ml_estimador = DecisionTreeClassifier(criterion='gini',
                                                  splitter='best',
                                                  random_state=0,
                                                  min_samples_leaf=2)

            #fit construindo a árvore, nosso modelo
            ml_estimador_model = ml_estimador.fit(X_train, y_train)
```

5.7.1.1. Avaliação de performance

Passamos a avaliação do estimador *DecisionTreeClassifier* utilizando as *features* e hiperparâmetros da VERSAO_DO_TESTE = "1.0".

Avaliação dos modelos

```
In [47]: #predizer y a partir de um X
         y_pred = ml_estimador_model.predict(X_test)
```

É possível verificar na matriz de confusão que há uma diversidade de classes targets, 176 conforme a seção 5.6, e que a diagonal principal contém pontos de acertos do modelo em destaque. Tendo em vista a diversidade de classes, o que prejudica a visualização, e como não identificamos uma área padrão de concentração de erros então deixamos de apresentar a matriz de confusão para os próximos modelos.

A seguir as principais métricas de avaliação de performance dos modelos que testamos. Tendo em vista a reutilização do código nos próximos modelos será exibido apenas o output desses códigos.

Acurácia

```
In [49]: # Acurácia
         print_label_estimador(label_estimador)
         accuracy_test = accuracy_score(y_test, y_pred)
         print('Acurácia: {0:.2f}% no treino, {1:.2f}% no teste'.
               format(ml_estimador_model.score(X_train, y_train) * 100, accuracy_test * 100))

         Classificador: DecisionTreeClassifier
         Acurácia: 87.11% no treino, 55.73% no teste
```

Precisão

```
In [50]: # Precision
         print_label_estimador(label_estimador)
         precision = precision_score(y_test, y_pred, average='weighted')
         print('Precision: %f' % precision)

         Classificador: DecisionTreeClassifier
         Precision: 0.562060
```

Recall

```
In [51]: # Recall
         print_label_estimador(label_estimador)
         recall = recall_score(y_test, y_pred, average='weighted')
         print('Recall: %f' % recall)

         Classificador: DecisionTreeClassifier
         Recall: 0.557310
```

F1-Score

```
In [52]: # F1-Score
         print_label_estimador(label_estimador)
         f1 = f1_score(y_test, y_pred, average='weighted')
         print('F1-Score: %f' % f1)

         Classificador: DecisionTreeClassifier
         F1-Score: 0.556506
```

Validação cruzada

```
In [56]: #executar Cross Validation com 5 partes

         cv = KFold(n_splits = 5, shuffle = True)
         scores = cross_val_score(ml_estimador, X_train, y_train, cv=cv, scoring="accuracy" )
```

```
In [57]: print_label_estimador(label_estimador)

         scores
         print(scores.mean())
         print(scores.std())
         mean_accuracy = scores.mean()
         dv = scores.std()
         print('Acurácia média: {:.2f}%'.format(mean_accuracy*100))
         print('Intervalo de acurácia: [{:.2f}% ~ {:.2f}%]'
                   .format((mean_accuracy - 2*dv)*100, (mean_accuracy + 2*dv)*100))

         Classificador: DecisionTreeClassifier
         0.5351761796366801
         0.001529626553254526
         Acurácia média: 53.52%
         Intervalo de acurácia: [53.21% ~ 53.82%]
```

Tendo em vista que o *dataset* está ordenado então utilizamos a função "*KFold"*. Ao invés de passarmos no parâmetro *"cv"* apenas o número de folds, utilizamos a função "*KFold"* que mistura os dados antes da segregação dos *folds*.

A acurácia média utilizando a validação cruzada foi de 53.52% o que permite a aceitação da acurácia simples de 55.73 obtida.

Importante ressaltar que não houve variação relevante comparando acurácia com precisão, recall e F1-Score.

Em resumo foram obtidos as seguintes métricas na VERSAO_DO_TESTE = "1.0":

```
Classificador: DecisionTreeClassifier
Acurácia: 87.11% no treino, 55.73% no teste
Precision: 0.562060
Recall: 0.557310
F1-Score: 0.556506

Acurácia média CV: 53.52%
Intervalo de acurácia CV: [53.15% ~ 53.88%]
```

5.7.1.2. Otimização

Utilizando a função "*feature*_importances_" criamos um *DataFrame* e ordenamos as *features* utilizadas na predição pelo modelo.

Feature Importance

In [55]:
```
print_label_estimador(label_estimador)

try:
    if label_estimador == "GaussianNB" or label_estimador == "KNeighborsClassifier" or label_estimador == "LogisticRegression" or
        #feature_importances_ para Naive Bayes
        imps = permutation_importance(ml_estimador_model, X_test, y_test)
        importancia = pd.DataFrame({"Feature":X.columns.values, "Importância (%)": 100 * imps.importances_mean})
    else:
        #feature para Tree
        importancia = pd.DataFrame({"Coluna":X.columns.values, "Importância (%)": 100 * ml_estimador_model.feature_importances_})
except BaseException:
    print("feature_importances_ indisponível na versão sklearn.__version__")

importancia.sort_values(by=["Importância (%)"], ascending=False)
```

```
Classificador: DecisionTreeClassifier
```

Out[55]:

	Coluna	Importância (%)
4	CO_NCM	21.746813
15	VL_FOB/KG_LIQUIDO	13.529292
14	VL_FRETE/KG_LIQUIDO	9.575172
11	VL_FOB	9.151441
13	VL_SEGURO	6.296114
12	VL_FRETE	6.257703
8	CO_URF	5.706621
3	CO_SH4	5.146638
6	SG_UF_NCM	5.004035
10	KG_LIQUIDO	4.123890
1	CO_MES	4.100762
9	QT_ESTAT	3.237101
7	CO_VIA	2.982372
0	CO_ANO	2.266330
2	CO_SH2	0.639352
5	CO_UNID	0.236364

Em termos de análise de negócio podemos observar o seguinte:

- É de conhecimento que as *features* “CO_NCM”, “CO_SH2” e “CO_SH4” possuem relação direta sendo a NCM o código mais específico com 8 casas, “CO_SH4” o mesmo código com apenas 4 casas e a SH2 o mesmo código com apenas 2 casas;

- A *feature* “CO_UNID” possui 13 valores exclusivos já a “CO_NCM” possui mais de 13 mil valores exclusivos, ou seja, “CO_UNID” tem relação direta e automática com “CO_NCM” não explicando ou agregando qualquer alteração de padrão no mundo fático.

Logo, é coerente que as *features* “CO_SH4”, “CO_SH2” e “CO_UNID” sejam removidas na próxima versão de teste (2.0) de todos os modelos pois são variáveis que não explicam ou agregam na predição da classificação do país de origem.

Executamos o *GridSearchCV* e o *RandomizedSearchCV* tentando buscar os melhores hiperparâmetros, conforme os seguintes códigos:

GridSearchCV

```
Classificador: DecisionTreeClassifier
Melhores parametros {'criterion': 'entropy','min_samples_leaf': 4} com o valor de acurácia 0.5670336391437309
```

RandomizedSearchCV

```
Classificador: DecisionTreeClassifier
Melhores parametros {'min_samples_leaf': 4, 'criterion': 'entropy'} com o valor de acurácia 0.5570336391437309
```

Treinamos novamente o modelo com novas *features* (“CO_SH4”, “CO_SH2” e “CO_UNID” removidas) e os novos hiperparâmetros. Dessa forma, foram obtidas as seguintes métricas na VERSAO_DO_TESTE = "2.0":

DecisionTreeClassifier

```
In [138]: if label_estimador == "DecisionTreeClassifier":
            ml_estimador = DecisionTreeClassifier(criterion='entropy',
                                                  splitter='best',
                                                  random_state=0,
                                                  min_samples_leaf=4)

            #fit construindo a árvore, nosso modelo
            ml_estimador_model = ml_estimador.fit(X_train, y_train)
```

```
Classificador: DecisionTreeClassifier
Acurácia: 80.02% no treino, 59.32% no teste
Precision: 0.593930
Recall: 0.593199
F1-Score: 0.592300

Acurácia média CV: 56.67%
Intervalo de acurácia CV: [56.10% ~ 57.23%]
```

5.7.2. *RandomForestClassifier*

Instanciamos o classificador *RandomForestClassifier* com hiperparâmetros default, conforme o trecho de código abaixo:

RandomForestClassifier

```
In [35]: if label_estimador == "RandomForestClassifier":
           ml_estimador = RandomForestClassifier(
                                    random_state=0,
                                    criterion='gini',
                                    max_depth=None,
                                    n_estimators=100,
                                    min_samples_leaf=1,
                                    n_jobs=-1)

           ml_estimador_model = ml_estimador.fit(X_train, y_train)
```

5.7.2.1. Avaliação de performance

Em resumo foram obtidas as seguintes métricas na VERSAO_DO_TESTE = "1.0":

```
Classificador: RandomForestClassifier
Acurácia: 99.98% no treino, 61.28% no teste
Precision: 0.625746
Recall: 0.612833
F1-Score: 0.608268

Acurácia média CV: 60.77%
Intervalo de acurácia CV: [58.36% ~ 62.14%]
```

5.7.2.2. Otimização

Executamos o *GridSearchCV* e o *RandomizedSearchCV* tentando buscar os melhores hiperparâmetros:

GridSearchCV

```
Classificador: RandomForestClassifier
Melhores parametros {'criterion': 'entropy', 'min_samples_leaf': 1} com o valor de acurácia 0.6461971573904179
```

RandomizedSearchCV

```
Classificador: RandomForestClassifier
Melhores parametros {'min_samples_leaf': 1, 'criterion': 'entropy'} com o valor de acurácia 0.6391972573904179
```

Treinamos novamente o modelo com novas *features* (“CO_SH4”, “CO_SH2” e “CO_UNID” removidas) e os novos hiperparâmetros. Dessa forma, foram obtidas as seguintes métricas na VERSAO_DO_TESTE = "2.0":

RandomForestClassifier

```
In [*]: if label_estimador == "RandomForestClassifier":
          ml_estimador = RandomForestClassifier(
                                    random_state=0,
                                    criterion='entropy',
                                    max_depth=None,
                                    n_estimators=120,
                                    min_samples_leaf=1,
                                    n_jobs=-1)

          ml_estimador_model = ml_estimador.fit(X_train, y_train)
```

Como o parâmetro “n_estimators” não foi testado no *GridSearchCV* e tampouco no *RandomizedSearchCV* utilizamos um valor ligeiramente maior (*n_estimators=120*) para verificar o % de aumento de acurácia. Os resultados foram os seguintes:

```
Classificador: RandomForestClassifier
Acurácia: 99.97% no treino, 67.68% no teste
Precision: 0.664746
Recall: 0.652833
F1-Score: 0.638268

Acurácia média CV: 63.10%
Intervalo de acurácia CV: [60.45% ~ 65.31%]
```

5.7.3. *Naive Bayes (CategoricalNB)*

Instanciamos o classificador ***CategoricalNB*** com hiperparâmetros default, conforme o trecho de código abaixo:

Categorical Naive Bayes

```
In [*]: if label_estimador == "CategoricalNB":
          ml_estimador = CategoricalNB(alpha = 0.1, fit_prior = False)
          ml_estimador_model = ml_estimador.fit(X_train, y_train)
```

5.7.3.1. Avaliação de performance

Em resumo foram obtidas as seguintes métricas na VERSAO_DO_TESTE = "1.0":

```
Classificador: CategoricalNB
Acurácia: 80.63% no treino, 42.68% no teste
Precision: 0.391543
Recall: 0.406759
F1-Score: 0.406168

Acurácia média CV: 37.59%
Intervalo de acurácia CV: [35.52% ~ 39.65%]
```

5.7.3.2. Otimização

Não realizamos otimização.

5.7.4. *KNeighborsClassifier*

Instanciamos o classificador *KNeighborsClassifier* com hiperparâmetros default, conforme o trecho de código abaixo:

KNeighborsClassifier

```
In [81]: if label_estimador == "KNeighborsClassifier":
             ml_estimador = KNeighborsClassifier(n_neighbors=5)
             ml_estimador_model = ml_estimador.fit(X_train, y_train)
```

5.7.4.1. Avaliação de performance

Em resumo foram obtidas as seguintes métricas na VERSAO_DO_TESTE = "1.0":

```
Classificador: KNeighborsClassifier
Acurácia: 61.17% no treino, 44.43% no teste
Precision: 0.442954
Recall: 0.444254
F1-Score: 0.438404

Acurácia média CV: 42.47%
Intervalo de acurácia CV: [41.96% ~ 42.98%]
```

5.7.4.2. Otimização

Executamos o *GridSearchCV* e o *RandomizedSearchCV* tentando buscar os melhores hiperparâmetros:

GridSearchCV

```
Classificador: KNeighborsClassifier
Melhores parametros {'metric': 'hamming', 'n_neighbors': 9} com o valor de acurácia 0.4557152565409447
```

RandomizedSearchCV

```
Classificador: KNeighborsClassifier
Melhores parametros {'n_neighbors': 9, 'metric': 'hamming'} com o valor de acurácia 0.4457152565409447
```

Treinamos novamente o modelo com novas *features* (“CO_SH4”, “CO_SH2” e “CO_UNID” removidas) e os novos hiperparâmetros. Dessa forma, foram obtidas as seguintes métricas na VERSAO_DO_TESTE = "2.0":

KNeighborsClassifier

```
In [50]: if label_estimador == "KNeighborsClassifier":
           ml_estimador = KNeighborsClassifier(n_neighbors=9, metric= 'hamming')
           ml_estimador_model = ml_estimador.fit(X_train, y_train)
```

```
Classificador: KNeighborsClassifier
Acurácia: 68.17% no treino, 47.43% no teste
Precision: 0.471254
Recall: 0.452251
F1-Score: 0.459491

Acurácia média CV: 44.47%
Intervalo de acurácia CV: [43.33% ~ 46.61%]
```

6. Interpretação dos Resultados

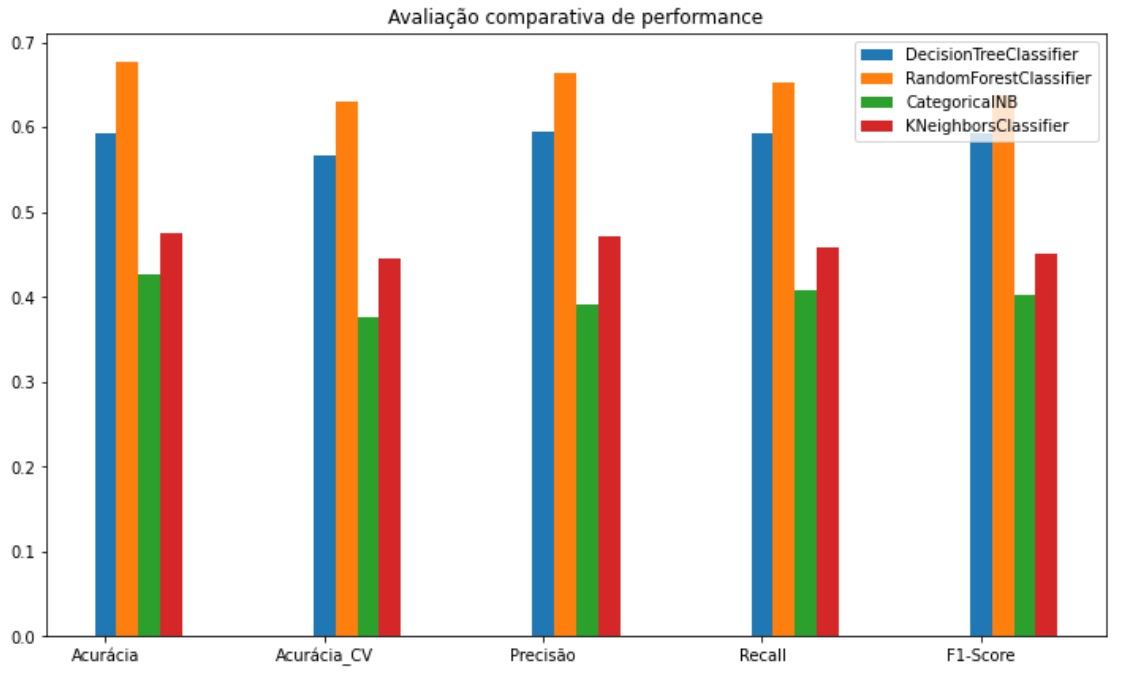
No decorrer da análise confirmamos que as *features* CO_NCM, VL_FOB/KG_LIQUIDO, VL_FRETE/KG_LIQUIDO, VL_FOB, VL_SEGURO, VL_FRETE e CO_URF são as mais relevantes para a construção do modelo de predição. Além do conhecimento do negócio, as leituras do método *"features_importantes"* trouxeram esta indicação mesmo utilizando diferentes hiperparâmetros.

A otimização decorrente de alterações nos hiperparâmetros e a remoção de *features* com importância próximo de nulo contribuíram na melhoria das métricas de performance dos os estimadores utilizados.

Ficou evidenciada a importância de buscar a otimização dos hiperparâmetros de cada estimador bem como a remoção de *features* sem importância. O melhor estimador para o presente trabalho, ***RandomForestClassifier***, partiu de uma acurácia média apurada em validação cruzada de 60.77%, e, após as otimizações, tal estimador atingiu 63.10% de acurácia média.

Elaboramos um gráfico comparativo para avaliar a performance dos estimadores utilizados com suas respectivas métricas: acurácia em teste, acurácia média com validação cruzada, precisão, recall e F1-score. Segue o código e o gráfico:

```
#Criar gráfico comparativo
x = np.arange(5)
plt.figure(figsize=(12,7))
width = 0.1

for i in range(0, len(lista_ml)):
  if lista_ml[i] in y_metricas.keys():
    plt.bar(x +(width*(i)), y_metricas[lista_ml[i]], width, label=lista_ml[i])

plt.xticks(x, ["Acurácia", "Acurácia_CV", "Precisão", "Recall", "F1-Score"])
plt.legend()
plt.title('Avaliação comparativa de performance')
plt.show()
```

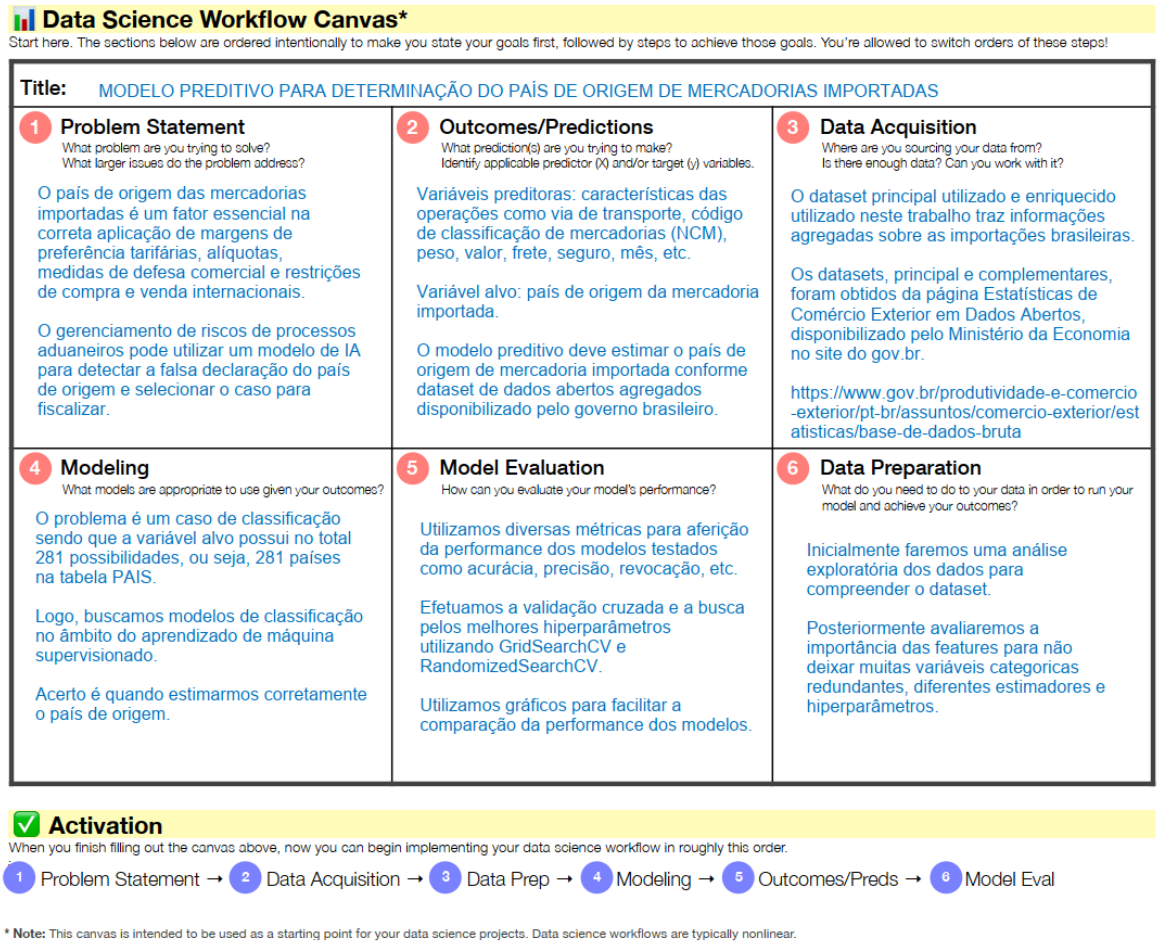


Importante destacar que a acurácia média foi mais alta do que a inicialmente esperada pois, como demonstramos, foram utilizados dados estatísticos abertos e que se referem a dados agregados e não são dados de operações individualizadas. Em que pese isso, foi possível obter *insights* relevantes para criação de um modelo de predição minimamente satisfatório de modo a atender a proposta do presente estudo.

Conclui-se, portanto, que todos os aprendizados gerados pela condução do presente trabalho foram capazes de produzir *insights* e conhecimentos relevantes acerca dos dados disponíveis, das formas tratamento e, principalmente, da criação e comparação de modelos ML capazes de contribuir no gerenciamento de risco e na fiscalização de operações aduaneiras.

7. Apresentação dos Resultados

O modelo conceitual Business Model Canvas, proposto por Jasmine Vasandani[21], foi utilizado para definir as principais fases do trabalho, assim como os insumos necessários e os resultados esperados em cada uma delas.



Data Science Workflow Canvas*

Start here. The sections below are ordered intentionally to make you state your goals first, followed by steps to achieve those goals. You're allowed to switch orders of these steps!

Title: MODELO PREDITIVO PARA DETERMINAÇÃO DO PAÍS DE ORIGEM DE MERCADORIAS IMPORTADAS

1 Problem Statement
What problem are you trying to solve?
What larger issues do the problem address?

O país de origem das mercadorias importadas é um fator essencial na correta aplicação de margens de preferência tarifárias, alíquotas, medidas de defesa comercial e restrições de compra e venda internacionais.

O gerenciamento de riscos de processos aduaneiros pode utilizar um modelo de IA para detectar a falsa declaração do país de origem e selecionar o caso para fiscalizar.

2 Outcomes/Predictions
What prediction(s) are you trying to make?
Identify applicable predictor (X) and/or target (y) variables.

Variáveis preditoras: características das operações como via de transporte, código de classificação de mercadorias (NCM), peso, valor, frete, seguro, mês, etc.

Variável alvo: país de origem da mercadoria importada.

O modelo preditivo deve estimar o país de origem de mercadoria importada conforme dataset de dados abertos agregados disponibilizado pelo governo brasileiro.

3 Data Acquisition
Where are you sourcing your data from?
Is there enough data? Can you work with it?

O dataset principal utilizado e enriquecido utilizado neste trabalho traz informações agregadas sobre as importações brasileiras.

Os datasets, principal e complementares, foram obtidos da página Estatísticas de Comércio Exterior em Dados Abertos, disponibilizado pelo Ministério da Economia no site do gov.br.

https://www.gov.br/produtividade-e-comercio-exterior/pt-br/assuntos/comercio-exterior/estatisticas/base-de-dados-bruta

4 Modeling
What models are appropriate to use given your outcomes?

O problema é um caso de classificação sendo que a variável alvo possui no total 281 possibilidades, ou seja, 281 países na tabela PAIS.

Logo, buscamos modelos de classificação no âmbito do aprendizado de máquina supervisionado.

Acerto é quando estimarmos corretamente o país de origem.

5 Model Evaluation
How can you evaluate your model's performance?

Utilizamos diversas métricas para aferição da performance dos modelos testados como acurácia, precisão, revocação, etc.

Efetuamos a validação cruzada e a busca pelos melhores hiperparâmetros utilizando GridSearchCV e RandomizedSearchCV.

Utilizamos gráficos para facilitar a comparação da performance dos modelos.

6 Data Preparation
What do you need to do to your data in order to run your model and achieve your outcomes?

Inicialmente faremos uma análise exploratória dos dados para compreender o dataset.

Posteriormente avaliaremos a importância das features para não deixar muitas variáveis categoricas redundantes, diferentes estimadores e hiperparâmetros.

Activation

When you finish filling out the canvas above, now you can begin implementing your data science workflow in roughly this order.

1 Problem Statement → 2 Data Acquisition → 3 Data Prep → 4 Modeling → 5 Outcomes/Preds → 6 Model Eval

* **Note:** This canvas is intended to be used as a starting point for your data science projects. Data science workflows are typically nonlinear.

Conceptualized by Jasmine Vasandani using notes from General Assembly's Data Science Immersive. Format inspired by Business Model Canvas.

Na seção anterior já apresentamos a interpretação dos resultados, incluindo um quadro comparativo da performance dos modelos testados.

[21] Artigo publicado no site Toward Data Science disponível em https://towardsdatascience.com/a-data-science-workflow-canvas-to-kickstart-your-projects-db62556be4d0

O melhor estimador para o presente trabalho, ***RandomForestClassifier***, partiu de uma acurácia média apurada em validação cruzada de 60.77%, e, após as otimizações, tal estimador atingiu 63.10% de acurácia média.

A conclusão do trabalho é que há possibilidade de construção de um modelo de aprendizado de máquina focado na identificação da probabilidade de erro na prestação de informação em declaração aduaneira buscando identificar riscos associados à falsa declaração do país de origem de mercadoria importada. Isso é possível tendo em vista a diversidade de *features* disponíveis em sistemas do governo e que se relacionam a cada operação de importação diferentemente dos dados abertos que constituem um conjunto pequeno de *features* e, ainda, agregados.

8. Links

Disponibilizamos os links para o vídeo com a apresentação resumida deste trabalho e para o repositório contendo os dados e scripts criados no projeto.

Link para o vídeo:
https://youtu.be/BWL1G6LqT7c
Link para o repositório GIT (sem *datasets*):
https://github.com/diegobbarbosa/tcc
Link para repositório dropBox (com *datasets*):
https://www.dropbox.com/sh/8u27g1j40vifp0t/AACrMbcw4NI0gKhA3E_GillDa?dl=0

APÊNDICE

As páginas seguintes apresentam a versão integral do script em linguagem Python desenvolvidos neste Trabalho de Conclusão de Curso. Esse script também está disponível em "https://github.com/diegobbarbosa/tcc/blob/main/baseComexTcc.ipynb"

Trabalho de Conclusão de Curso - TCC

MODELO PREDITIVO PARA DETERMINAÇÃO DO PAÍS DE ORIGEM DE MERCADORIAS IMPORTADAS

Diego de Borba Barbosa

Curso de Especialização em Ciência de Dados e Big Data, Puc Minas, abril de 2022

Coleta de dados

Importar bibliotecas

In [ ]:
```
#Carrega as bibliotecas do modelo e algumas utilizadas em testes
#!pip install pydotplus
#!pip install dtreeviz
#from sklearn.inspection import permutation_importance
#import sklearn
#print('The scikit-learn version is {}.'.format(sklearn.__version__))
import functools
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn import datasets, tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import CategoricalNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, mean_absolute_error, r2_score
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.feature_extraction import DictVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import KFold
import time
from functools import wraps
import pydotplus
from IPython.display import Image
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from matplotlib.pyplot import subplots
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

%matplotlib inline
plt.rcParams["figure.figsize"] = [15, 12]
```

Parâmetros

In [ ]:
```
# Define parâmetros para exploração e criação do modelo
#versão do teste com colunas e hiperparâmetros específicos
VERSAO_DO_TESTE = "1.0"
#Anos utilizados para análise exploratória
ANOS = [2019, 2020, 2021]
#Filtro por capítulos de 0 até 99 - filtrar para reduzir consumo de memória
CAPITULO_INICIO = 0
CAPITULO_FIM = 30
#Estrutura y_metricas acumula métricas de avaliação de performance dos modelos
#Deve-se executar seção "Treinamento de modelos" e células posteriores para cada um dos modelos
y_metricas = dict()
```

Carga dos datasets

In [ ]:
```
#Carregamento do dataset
start_time = time.time()

#dataset principal com dados brutos
baseComex = pd.DataFrame()
for ano in ANOS:
    baseComex = baseComex.append(pd.read_csv(".//Dados//IMP_{0}.csv".format(ano), sep = ';', encoding='latin-1'))

print("Informações do dataset principal IMP_2021")
print("\nTipo: {0}".format(type(baseComex)))
print("Dimensões: {0}".format(baseComex.shape))
print("Campos: {0}".format(baseComex.keys()))
baseComex.describe()
```

In [ ]:
```
baseComex.memory_usage().sum()/1024**2
#baseComex.dtypes
```

In [ ]:
```
#computar tempo de carregamento
elapsed_time = time.time() - start_time
print("\nLoad Dataset: %.2f" %elapsed_time, "segundos")
baseComex.head()
```

In [ ]:
```
#carrega domínios e dataset acessórios
base_pais = pd.read_csv('.//Dados//PAIS.csv', sep = ';' , encoding='latin-1')
print("Informações do dataset PAIS")
print("\nDimensões: {0}".format(base_pais.shape))
print("Campos: {0}".format(base_pais.keys()))
base_pais.head()
```

In [ ]:
```
base_via = pd.read_csv('.//Dados//VIA.csv', sep = ';', encoding='latin-1')
print("Informações do dataset VIA")
print("\nDimensões: {0}".format(base_via.shape))
print("Campos: {0}".format(base_via.keys()))
base_via
```

In [ ]:
```
base_URF = pd.read_csv('.//Dados//URF.csv', sep = ';', encoding='latin-1')
print("Informações do dataset URF")
print("\nDimensões: {0}".format(base_URF.shape))
print("Campos: {0}".format(base_URF.keys()))
base_URF.head()
```

In [ ]:
```
base_NCM = pd.read_csv('.//Dados//NCM.csv', sep = ';', encoding='latin-1')
print("Informações do dataset NCM")
print("\nDimensões: {0}".format(base_NCM.shape))
print("Campos: {0}".format(base_NCM.keys()))
base_NCM.head()
```

In [ ]:
```
base_SH6 = pd.read_csv('.//Dados//NCM_SH.csv', sep = ';', encoding='latin-1')
print("Informações do dataset NCM_SH")
print("\nDimensões: {0}".format(base_SH6.shape))
print("Campos: {0}".format(base_SH6.keys()))
base_SH6.head()
```

Enriquecimento de dataset

In [ ]:
```
#dataset enriquecido com domínios e dataset acessórios
baseComexEnriquecida = pd.merge(baseComex, base_pais, on="CO_PAIS", how="inner")
baseComexEnriquecida = pd.merge(baseComexEnriquecida, base_via, on="CO_VIA", how="inner")
baseComexEnriquecida = pd.merge(baseComexEnriquecida, base_URF, on="CO_URF", how="inner")
baseComexEnriquecida = pd.merge(baseComexEnriquecida, base_NCM, on=["CO_NCM","CO_UNID"], how="inner")
baseComexEnriquecida = pd.merge(baseComexEnriquecida, base_SH6, on="CO_SH6", how="inner")

#Filtrar dataset para manter apenas colunas para análise exploratória
lista = ['CO_ANO', 'CO_MES', 'CO_SH2', 'NO_SH2_POR', 'CO_SH4', 'NO_SH4_POR', 'CO_SH6',
         'CO_NCM', 'CO_UNID', 'CO_PAIS', 'NO_PAIS', 'SG_UF_NCM','CO_VIA','NO_VIA',
         'CO_URF', 'NO_URF', 'QT_ESTAT', 'KG_LIQUIDO', 'VL_FOB', 'VL_FRETE', 'VL_SEGURO']

baseComexEnriquecidaAmostra = baseComexEnriquecida[lista]
print("Informações do dataset principal enriquecido para análise exploratória")
#print("\nFiltros por capítulo SH2: {0} a {1}".format(CAPITULO_INICIO, CAPITULO_FIM))
print("Dimensões: {0}".format(baseComexEnriquecidaAmostra.shape))
print("Campos: {0}".format(baseComexEnriquecidaAmostra.keys()))
```

In [ ]:
```
#Filtrar dataset para manter apenas colunas para criação do modelo
lista = ['CO_ANO', 'CO_MES', 'CO_SH2', 'CO_SH4', 'CO_NCM',
         'CO_UNID', 'CO_PAIS', 'SG_UF_NCM','CO_VIA', 'CO_URF',
         'QT_ESTAT', 'KG_LIQUIDO', 'VL_FOB', 'VL_FRETE', 'VL_SEGURO']
baseComexModelo = baseComexEnriquecida[lista]

#Filtrando dataset para selecionar um intervalo de mercadorias por capitulo
baseComexModelo = baseComexModelo.loc[
                                (baseComexModelo['CO_SH2'] >= CAPITULO_INICIO)
                                & (baseComexModelo['CO_SH2'] <= CAPITULO_FIM)]

#Filtrar dataset para criar modelo - reduzir consumo de memória
#Criar colunas derivadas no dataset do modelo
baseComexModelo["VL_FRETE/KG_LIQUIDO"]= baseComexModelo["VL_FRETE"]/ baseComexModelo["KG_LIQUIDO"]
baseComexModelo["VL_FRETE/KG_LIQUIDO"].replace(np.inf, 0, inplace=True)
baseComexModelo["VL_FRETE/KG_LIQUIDO"].replace(np.nan, 0, inplace=True)

baseComexModelo["VL_FOB/KG_LIQUIDO"]= baseComexModelo["VL_FOB"]/ baseComexModelo["KG_LIQUIDO"]
baseComexModelo["VL_FOB/KG_LIQUIDO"].replace(np.inf, 0, inplace=True)
baseComexModelo["VL_FOB/KG_LIQUIDO"].replace(np.nan, 0, inplace=True)

print("\nInformações do dataset principal para criação do modelo")
print("\nFiltros por capítulo SH2: {0} a {1}".format(CAPITULO_INICIO, CAPITULO_FIM))
print("Dimensões: {0}".format(baseComexModelo.shape))
print("Campos: {0}".format(baseComexModelo.keys()))
```

In [ ]:
```
baseComexModelo.head()
```

Análise exploratória

Gráficos

In [ ]:
```
df_graf_via_fob = baseComexEnriquecidaAmostra.groupby(["NO_VIA"]).agg({"VL_FOB":['sum']})
df_graf_via_kg = baseComexEnriquecidaAmostra.groupby(["NO_VIA"]).agg({"KG_LIQUIDO":['sum']})
df_graf_via_frete = baseComexEnriquecidaAmostra.groupby(["NO_VIA"]).agg({"VL_FRETE":['sum']})
df_graf_via_pais = baseComexEnriquecidaAmostra.groupby(["NO_VIA"]).agg({"NO_PAIS":['nunique']})

#plt.style.use(style ="seaborn")
plt.rcdefaults()

# Cada plot terá o mesmo tamanho de figuras
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15,7))

x1 = df_graf_via_fob.index.tolist()
x2 = df_graf_via_kg.index.tolist()
x3 = df_graf_via_frete.index.tolist()
x4 = df_graf_via_pais.index.tolist()

y1 = df_graf_via_fob['VL_FOB'][ 'sum'].tolist()
y2 = df_graf_via_kg['KG_LIQUIDO']['sum'].tolist()
y3 = df_graf_via_frete['VL_FRETE']['sum'].tolist()
y4 = df_graf_via_pais['NO_PAIS']['nunique'].tolist()

ax1.bar(x1, y1,color='red')
ax2.bar(x2, y2,color='blue')
ax3.bar(x3, y3,color='orange')
ax4.bar(x4, y4,color='green')

ax1.set(title="Valor FOB das mercadorias x Via de transporte", xlabel="Via de transporte", ylabel="Valor FOB")
ax2.set(title="Peso líquido (Kg) das mercadorias x Via de transporte", xlabel="Via de transporte", ylabel="Peso líquido
ax3.set(title="Valor do frete x Via de transporte", xlabel="Via de transporte", ylabel="Valor do frete")
ax4.set(title="Quantidade de países de origem x Via de transporte", xlabel="Via de transporte", ylabel="Quantidade de pa

ax1.set_xticklabels(x1, rotation=45, ha='right')
ax2.set_xticklabels(x2, rotation=45, ha='right')
ax3.set_xticklabels(x3, rotation=45, ha='right')
ax4.set_xticklabels(x4, rotation=45, ha='right')

plt.subplots_adjust(wspace=0.2, hspace=1)
#plt.legend()
plt.show()
```

In [ ]:
```
df_graf_pais_fob = baseComexEnriquecidaAmostra.groupby(["NO_PAIS"]).agg({"VL_FOB":['sum']})
df_graf_pais_kg = baseComexEnriquecidaAmostra.groupby(["NO_PAIS"]).agg({"KG_LIQUIDO":['sum']})
df_graf_pais_frete = baseComexEnriquecidaAmostra.groupby(["NO_PAIS"]).agg({"VL_FRETE":['sum']})
df_graf_pais_ncm = baseComexEnriquecidaAmostra.groupby(["NO_PAIS"]).agg({"CO_NCM":['nunique']})

df_graf_pais_fob.columns = ["VL_FOB_sum"]
df_graf_pais_kg.columns = ["KG_LIQUIDO_sum"]
df_graf_pais_frete.columns = ["VL_FRETE_sum"]
df_graf_pais_ncm.columns = ["CO_NCM_nunique"]

top = 15
df_graf_pais_fob = df_graf_pais_fob.sort_values(by="VL_FOB_sum", ascending=False).head(top)
df_graf_pais_kg = df_graf_pais_kg.sort_values(by="KG_LIQUIDO_sum", ascending=False).head(top)
df_graf_pais_frete = df_graf_pais_frete.sort_values(by="VL_FRETE_sum", ascending=False).head(top)
df_graf_pais_ncm = df_graf_pais_ncm.sort_values(by="CO_NCM_nunique", ascending=False).head(top)

plt.rcdefaults()

# Cada plot terá o mesmo tamanho de figuras
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15,7))

x1 = df_graf_pais_fob.index.tolist()
x2 = df_graf_pais_kg.index.tolist()
x3 = df_graf_pais_frete.index.tolist()
x4 = df_graf_pais_ncm.index.tolist()

y1 = df_graf_pais_fob['VL_FOB_sum'].tolist()
y2 = df_graf_pais_kg['KG_LIQUIDO_sum'].tolist()
y3 = df_graf_pais_frete['VL_FRETE_sum'].tolist()
y4 = df_graf_pais_ncm['CO_NCM_nunique'].tolist()

ax1.barh(x1, y1,color='red')
ax2.barh(x2, y2,color='blue')
ax3.barh(x3, y3,color='orange')
ax4.barh(x4, y4,color='green')

ax1.set(title="Países de origem das mercadorias (top {0}) x Valor FOB".format(top), xlabel="Valor FOB", ylabel="Países d
ax2.set(title="Países de origem das mercadorias (top {0}) x Peso líquido (Kg) das mercadorias".format(top), xlabel="Peso
ax3.set(title="Países de origem das mercadorias (top {0}) x Valor do frete".format(top), xlabel="Valor do frete", ylabel
ax4.set(title="Países de origem das mercadorias (top {0})  x Contagem diferentes mercadorias (NCM)".format(top), xlabel=

plt.subplots_adjust(wspace=0.4, hspace=0.5)
#plt.legend()
plt.show()
```

In [ ]:
```
#FAZER POR ANO
df_graf_ano_fob = baseComexEnriquecidaAmostra.groupby(["CO_ANO"]).agg({"VL_FOB":['sum']})
df_graf_ano_kg = baseComexEnriquecidaAmostra.groupby(["CO_ANO"]).agg({"KG_LIQUIDO":['sum']})
df_graf_ano_frete = baseComexEnriquecidaAmostra.groupby(["CO_ANO"]).agg({"VL_FRETE":['sum']})
df_graf_ano_ncm = baseComexEnriquecidaAmostra.groupby(["CO_ANO"]).agg({"CO_NCM":['nunique']})

plt.rcdefaults()

# Cada plot terá o mesmo tamanho de figuras
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15,7))

x1 = df_graf_ano_fob.index.tolist()
x2 = df_graf_ano_kg.index.tolist()
x3 = df_graf_ano_frete.index.tolist()
x4 = df_graf_ano_ncm.index.tolist()

y1 = df_graf_ano_fob['VL_FOB']['sum'].tolist()
y2 = df_graf_ano_kg['KG_LIQUIDO']['sum'].tolist()
y3 = df_graf_ano_frete['VL_FRETE']['sum'].tolist()
y4 = df_graf_ano_ncm['CO_NCM']['nunique'].tolist()

ax1.plot(x1, y1,color='red')
ax2.plot(x2, y2,color='blue')
ax3.plot(x3, y3,color='orange')
ax4.plot(x4, y4,color='green')

ax1.set(title="Valor FOB x Ano", ylabel="Valor FOB", xlabel="Ano")
ax2.set(title="Peso líquido (Kg) das mercadorias x Ano", ylabel="Peso líquido (Kg)", xlabel="Ano")
ax3.set(title="Valor do frete x Ano", ylabel="Valor do frete", xlabel="Ano")
ax4.set(title="Contagem diferentes mercadorias (NCM) x Ano", ylabel="Contagem diferentes mercadorias (NCM)", xlabel="Ano

plt.subplots_adjust(wspace=0.2, hspace=0.5)
#plt.legend()
plt.show()
```

In [ ]:
```
#FAZER POR MES
df_graf_mes_fob = baseComexEnriquecidaAmostra.groupby(["CO_MES"]).agg({"VL_FOB":['sum']})
df_graf_mes_kg = baseComexEnriquecidaAmostra.groupby(["CO_MES"]).agg({"KG_LIQUIDO":['sum']})
df_graf_mes_frete = baseComexEnriquecidaAmostra.groupby(["CO_MES"]).agg({"VL_FRETE":['sum']})
df_graf_mes_ncm = baseComexEnriquecidaAmostra.groupby(["CO_MES"]).agg({"CO_NCM":['nunique']})

plt.rcdefaults()

# Cada plot terá o mesmo tamanho de figuras
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15,7))

x1 = df_graf_mes_fob.index.tolist()
x2 = df_graf_mes_kg.index.tolist()
x3 = df_graf_mes_frete.index.tolist()
x4 = df_graf_mes_ncm.index.tolist()

y1 = df_graf_mes_fob['VL_FOB'][ 'sum'].tolist()
y2 = df_graf_mes_kg['KG_LIQUIDO']['sum'].tolist()
y3 = df_graf_mes_frete['VL_FRETE']['sum'].tolist()
y4 = df_graf_mes_ncm['CO_NCM']['nunique'].tolist()

ax1.plot(x1, y1,color='red')
ax2.plot(x2, y2,color='blue')
ax3.plot(x3, y3,color='orange')
ax4.plot(x4, y4,color='green')

ax1.set(title="Valor FOB x Mês do ano", ylabel="Valor FOB", xlabel="Mês do ano")
ax2.set(title="Peso líquido (Kg) das mercadorias x Mês do ano", ylabel="Peso líquido (Kg)", xlabel="Mês do ano")
ax3.set(title="Valor do frete x Mês do ano", ylabel="Valor do frete", xlabel="Mês do ano")
ax4.set(title="Contagem diferentes mercadorias (NCM) x Mês do ano", ylabel="Contagem diferentes mercadorias (NCM)", xlab

plt.subplots_adjust(wspace=0.2, hspace=0.5)
#plt.legend()
plt.show()
```

Wordcloud

In [ ]:
```
#CRIA DICT para wordcloud
df_graf_urf_fob = baseComexEnriquecidaAmostra.groupby(["NO_URF"]).agg({"VL_FOB":['sum']})
df_graf_urf_kg = baseComexEnriquecidaAmostra.groupby(["NO_URF"]).agg({"KG_LIQUIDO":['sum']})

df_graf_urf_fob.columns = ["VL_FOB_sum"]
df_graf_urf_kg.columns = ["KG_LIQUIDO_sum"]

df_graf_urf_fob["VL_FOB_sum"].apply(int)
data_urf_fob = df_graf_urf_fob.to_dict('dict')
data_urf_fob = data_urf_fob['VL_FOB_sum']

df_graf_urf_kg["KG_LIQUIDO_sum"].apply(int)
data_urf_kg = df_graf_urf_kg.to_dict('dict')
data_urf_kg = data_urf_kg['KG_LIQUIDO_sum']
```

In [ ]:
```
# gerar uma wordcloud
wordcloud = WordCloud(#stopwords=STOPWORDS,
                      background_color="black",
                      width=1600, height=800).generate_from_frequencies(data_urf_fob)

# mostrar a imagem final
fig, ax = plt.subplots(figsize=(20,10))
ax.imshow(wordcloud, interpolation='bilinear')
ax.set_axis_off()

plt.imshow(wordcloud)
```

In [ ]:
```
# gerar uma wordcloud
wordcloud = WordCloud(#stopwords=STOPWORDS,
                      background_color="white",
                      width=1600, height=800).generate_from_frequencies(data_urf_kg)

# mostrar a imagem final
fig, ax = plt.subplots(figsize=(20,10))
ax.imshow(wordcloud, interpolation='bilinear')
ax.set_axis_off()
plt.imshow(wordcloud)
```

Pairplot

In [ ]:
```
df_pair_plot = baseComexEnriquecidaAmostra.groupby(['CO_SH2', 'CO_PAIS', 'SG_UF_NCM', 'CO_VIA'], as_index=False).agg({'K
       'VL_FOB':['sum'], 'VL_FRETE':['sum']})

df_pair_plot.columns = ['CO_SH2', 'CO_PAIS', 'SG_UF_NCM', 'CO_VIA', 'KG_LIQUIDO_sum', 'VL_FOB_sum', 'VL_FRETE_sum']
df_pair_plot.shape
```

In [ ]:
```
#análise
plt.figure(figsize=(36,20))

sns.pairplot(df_pair_plot[['CO_SH2', 'CO_PAIS', 'SG_UF_NCM', 'CO_VIA', 'KG_LIQUIDO_sum',
       'VL_FOB_sum', 'VL_FRETE_sum']], hue="CO_SH2", diag_kind="hist", palette="bright")
plt.show()
```

In [ ]:
```
sns.relplot(x='VL_FRETE_sum', y='KG_LIQUIDO_sum', data = df_pair_plot, kind='scatter', size ='VL_FOB_sum', hue ='CO_PAIS
            sizes =(20,200), palette = 'RdPu', alpha=.5, height = 7)
plt.show()
```

In [ ]:
```
sns.catplot(x='CO_SH2', y='VL_FOB_sum', data = df_pair_plot[['CO_SH2', 'VL_FOB_sum']], kind='swarm')
plt.show()
```

Criação do Modelo

Ajustes dataset (drop e encoder)

In [ ]:
```
#Conforme análises pode ser necessário desconsiderar essas colunas antes do treinamento
#baseComexModelo.drop("VL_SEGURO", axis=1, inplace=True)
#baseComexModelo.drop("CO_UNID", axis=1, inplace=True)
#baseComexModelo.drop("CO_SH4", axis=1, inplace=True)
#baseComexModelo.drop("CO_SH2", axis=1, inplace=True)
#baseComexModelo.drop("CO_UNID", axis=1, inplace=True)

baseComexModelo.shape
#baseComexModelo.head()
```

In [ ]:
```
#encoder da coluna label
le = LabelEncoder()
baseComexModelo["SG_UF_NCM"] = le.fit_transform(baseComexModelo["SG_UF_NCM"])
baseComexModelo["CO_ANO"] = le.fit_transform(baseComexModelo["CO_ANO"])
baseComexModelo["CO_NCM"] = le.fit_transform(baseComexModelo["CO_NCM"])
baseComexModelo["CO_URF"] = le.fit_transform(baseComexModelo["CO_URF"])
baseComexModelo.head()
```

Particionamento do dataset

In [ ]:
```
# Particiona a base de dados
label_target = "CO_PAIS"
X = baseComexModelo.drop(label_target, axis=1)
y = baseComexModelo[label_target]

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.15)
```

In [ ]:
```
#Quantidade países (target)
unique= np.unique(y)
print('Contagem de target ({0}): {1}'.format(label_target, unique.size))
```

Treinamento de modelos

In [ ]:
```
def print_label_estimador(label):
    print('Classificador: {0}'.format(label))

#inserir label do modelo a treinar e "Executar após"
#DecisionTreeClassifier, RandomForestClassifier, CategoricalNB, KNeighborsClassifier e LogisticRegression
lista_ml = ["DecisionTreeClassifier","RandomForestClassifier", "CategoricalNB", "KNeighborsClassifier", "LogisticRegress
label_estimador = lista_ml[3]
print_label_estimador(label_estimador)
```

DecisionTreeClassifier

In [ ]:
```
if label_estimador == "DecisionTreeClassifier":
  ml_estimador = DecisionTreeClassifier(criterion='entropy',
                                        splitter='best',
                                        random_state=0,
                                        min_samples_leaf=1)

  #fit construindo a árvore, nosso modelo
  ml_estimador_model = ml_estimador.fit(X_train, y_train)
```

RandomForestClassifier

In [ ]:
```
if label_estimador == "RandomForestClassifier":
  ml_estimador = RandomForestClassifier(
                             random_state=0,
                             criterion='entropy',
                             max_depth=None,
                             n_estimators=100,
                             min_samples_leaf=1,
                             n_jobs=-1)

  ml_estimador_model = ml_estimador.fit(X_train, y_train)
```

Categorical Naive Bayes

In [ ]:
```
if label_estimador == "CategoricalNB":
  ml_estimador = CategoricalNB(alpha = 0.1, fit_prior = False)
  ml_estimador_model = ml_estimador.fit(X_train, y_train)
```

KNeighborsClassifier

In [ ]:
```
if label_estimador == "KNeighborsClassifier":
  ml_estimador = KNeighborsClassifier(n_neighbors=9, metric= 'hamming')
  ml_estimador_model = ml_estimador.fit(X_train, y_train)
```

LogisticRegression

In [ ]:
```
if label_estimador == "LogisticRegression":
  ml_estimador = LogisticRegression(random_state=0)
  ml_estimador_model = ml_estimador.fit(X_train, y_train)
```

Avaliação dos modelos

In [ ]:
```
#predizer y a partir de um X
y_pred = ml_estimador_model.predict(X_test)
```

In [ ]:
```
X_train.dtypes
```

Acurácia

In [ ]:
```
# Acurácia
print_label_estimador(label_estimador)
accuracy_test = accuracy_score(y_test, y_pred)
print('Acurácia: {0:.2f}% no treino, {1:.2f}% no teste'.
      format(ml_estimador_model.score(X_train, y_train) * 100, accuracy_test * 100))
```

Precisão

In [ ]:
```
# Precision
print_label_estimador(label_estimador)
precision = precision_score(y_test, y_pred, average='weighted')
print('Precision: %f' % precision)
```

Recall

In [ ]:
```
# Recall
print_label_estimador(label_estimador)
recall = recall_score(y_test, y_pred, average='weighted')
print('Recall: %f' % recall)
```

F1-Score

In [ ]:
```
# F1-Score
print_label_estimador(label_estimador)
f1 = f1_score(y_test, y_pred, average='weighted')
print('F1-Score: %f' % f1)
```

Classification report

In [ ]:
```
print_label_estimador(label_estimador)
print(classification_report(y_test, y_pred))
```

Matriz de confusão

In [ ]:
```
plt.figure(figsize=(20,20))
cnf_matrix = confusion_matrix(y_test, y_pred)
cnf_table = pd.DataFrame(data=cnf_matrix)
data = {
    'Ocorreu': y_test,
    'Predito': y_pred
}
df = pd.DataFrame(data, columns=['Ocorreu','Predito'])
conf = pd.crosstab(df['Ocorreu'], df['Predito'], rownames=['Ocorreu'], colnames=['Predito'])
sns.heatmap(conf, annot=True, annot_kws={"size":5}, cmap=plt.cm.Blues)

plt.title('Matriz de Confusão')
plt.show()
```

Feature Importance

In [ ]:
```
print_label_estimador(label_estimador)

try:
    if label_estimador == "CategoricalNB" or label_estimador == "KNeighborsClassifier" or label_estimador == "LogisticRe
        #feature_importances_ para Naive Bayes
        imps = permutation_importance(ml_estimador_model, X_test, y_test)
        importancia = pd.DataFrame({"Feature":X.columns.values, "Importância (%)": 100 * imps.importances_mean})
    else:
        #feature para Tree
        importancia = pd.DataFrame({"Coluna":X.columns.values, "Importância (%)": 100 * ml_estimador_model.feature_impor
except BaseException:
    print("feature_importances_ indisponível na versão sklearn.__version__")

importancia.sort_values(by=["Importância (%)"], ascending=False)
```

Validação cruzada

In [ ]:
```
#executar Cross Validation com 5 partes

cv = KFold(n_splits = 5, shuffle = True)
scores = cross_val_score(ml_estimador, X_train, y_train, cv=cv, scoring="accuracy" )
```

In [ ]:
```
print_label_estimador(label_estimador)

scores
print(scores.mean())
print(scores.std())
mean_accuracy = scores.mean()
dv = scores.std()
print('Acurácia média: {:.2f}%'.format(mean_accuracy*100))
print('Intervalo de acurácia: [{:.2f}% ~ {:.2f}%]'
          .format((mean_accuracy - 2*dv)*100, (mean_accuracy + 2*dv)*100))
```

GridSearchCV

In [ ]:
```
print_label_estimador(label_estimador)

list_min_samples_leaf = list(range(1, 7, 1))

k_list = list(range(1,21))

if label_estimador == "CategoricalNB":
  param_grid = {'var_smoothing': np.logspace(0,-9, num=30)}

elif label_estimador == "KNeighborsClassifier":
  param_grid = {'n_neighbors': k_list,
                'metric': ['minkowski','canberra', 'hamming', 'euclidean', 'manhattan']}
elif label_estimador == "LogisticRegression":
  param_grid = {'C': np.logspace(-5, 8, 5), 'penalty': ['l1', 'l2']}
else:
  param_grid = {"criterion": ['entropy', 'gini'],
                "min_samples_leaf": list_min_samples_leaf}
```

In [ ]:
```
print_label_estimador(label_estimador)
grid_search = GridSearchCV(ml_estimador_model, param_grid, scoring="accuracy", cv=5)
grid_search.fit(X_train, y_train)

classifier_rf = grid_search.best_estimator_
print("Melhores parametros {} com o valor de acurácia {} ".
      format(grid_search.best_params_, grid_search.best_score_))
```

RandomizedSearchCV

In [ ]:
```
print_label_estimador(label_estimador)

randomized_search = RandomizedSearchCV(ml_estimador_model, param_grid, scoring="accuracy", cv=5, n_iter=10,)
randomized_search.fit(X_train, y_train)

randomized_search.best_estimator_
randomized_search.best_params_, randomized_search.best_score_
print("Melhores parametros {} com o valor de acurácia {} ".
      format(randomized_search.best_params_, randomized_search.best_score_))
```

Erro médio absoluto

In [ ]:
```
print_label_estimador(label_estimador)

mean = mean_absolute_error(y_test, y_pred)
print(mean)
```

Visualizar árvore

In [ ]:
```
### Teste visualização pequenas árvores
    if label_estimador == "DecisionTreeClassifier - false":
        name_classes = [str(val) for val in ml_estimador_model.classes_]
        # Create DOT data
        dot_data = tree.export_graphviz(ml_estimador_model, out_file=None,
                                        #proportion=True,
                                        rounded = True,
                                        filled = True,
                                        feature_names = ml_estimador_model.feature_names_in_,
                                        class_names = name_classes)
        # Draw graph
        graph = pydotplus.graph_from_dot_data(dot_data)
        # Show graph
        Image(graph.create_png())
```

Avaliação comparativa de performance

In [ ]:
```
#Estrutura y_metricas acumula métricas de avaliação de performance dos modelos
#Deve-se executar após seção "Treinamento de modelos" para cada um dos modelos
y_metricas[label_estimador] = [accuracy_test, mean_accuracy, precision, recall, f1]
print(y_metricas)
```

In [ ]:
```
#Criar gráfico comparativo
x = np.arange(5)
plt.figure(figsize=(12,7))
width = 0.1

for i in range(0, len(lista_ml)):
  if lista_ml[i] in y_metricas.keys():
    plt.bar(x +(width*(i)), y_metricas[lista_ml[i]], width, label=lista_ml[i])

plt.xticks(x, ["Acurácia", "Acurácia_CV", "Precisão", "Recall", "F1-Score"])
plt.legend()
plt.title('Avaliação comparativa de performance')
plt.show()
```

In [ ]:
```

```