Previsão de Vendas de Mercado

Trabalho de AMMCI

Equipe:
Gabriel de Melo Osório - 107862
Henrique Shiguemoto Felizardo - 115207
Matheus Augusto Schiavon - 107115

Roteiro

- Dataset Escolhido
- Árvore de Decisão
- Naive Bayes
- Support Vector Machine

Dataset Escolhido

O modelo do dataset escolhido foi o conjunto de dados de vendas da Adidas, uma coleção de dados que inclui informações sobre as vendas de produtos da Adidas. Esse tipo de conjunto de dados incluir detalhes como número de unidades vendidas, receita total de vendas, local das vendas, tipo de produto vendido, entre outras informações relevantes.

Link: https://www.kaggle.com/datasets/heemalichaudhari/adidas-sales-dataset?select=Adidas+US+Sales+Datasets.xlsx

Tratamento de dados base

Foi realizado um tratamento de dados no dataset da adidas para corrigir alguns erros e para adequa-lo a um dataframe:

Adequação ao dataset:

- Ler as informações do dataset, mas apenas as colunas de B até N do excel.
- Utilizar os valores da quarta linha para nomear as colunas.
- Apagar as 4 primeiras linhas.

Erro:

- Dividir o valor da venda e o lucro por 10 pois as contas do dataset original estão erradas.

Importando os Dados

[3] 24s

```
# Carregue o auxiliar do Drive e monte o Drive
from google.colab import drive

# Isso solicitará autorização.até N do excel.
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[4] 3s

```
import pandas as pd # É usado para manipulção do dataframe

# Local do DataFrame da Adidas no meu Drive
local_dataset = '/content/drive/MyDrive/AMMCI (DIN-9806)/Trabalho 2/Adidas US Sales Datasets.xlsx'

# Ler as informações do dataframe, mas apenas as colunas de B até N do excel.
df=pd.read_excel(local_dataset, usecols="B:N")

# Utilizar os valores da quarta linha para nomear as colunas.
df.columns = df.iloc[3]

# Apagar as 4 primeiras linhas.
df = df.drop([0, 1, 2, 3])

# Divide o valor da venda e o lucro por 10 pois as contas do dataframe original estão erradas.
df['Total Sales'] = df['Total Sales']/10
df['Operating Profit'] = df['Operating Profit']/10
df_svm = df.copy()
df
```

Importando os Dados

3	Retailer	Retailer ID	Invoice Date	Region	State	City	Product	Price per Unit	Units Sold	Total Sales	Operating Profit	Operating Margin	Sales Method
4	Foot Locker	1185732	2020-01-01 00:00:00	Northeast	New York	New York	Men's Street Footwear	50	1200	60000.0	30000.0	0.5	In-store
5	Foot Locker	1185732	2020-01-02 00:00:00	Northeast	New York	New York	Men's Athletic Footwear	50	1000	50000.0	15000.0	0.3	In-store
6	Foot Locker	1185732	2020-01-03 00:00:00	Northeast	New York	New York	Women's Street Footwear	40	1000	40000.0	14000.0	0.35	In-store
7	Foot Locker	1185732	2020-01-04 00:00:00	Northeast	New York	New York	Women's Athletic Footwear	45	850	38250.0	13387.5	0.35	In-store
8	Foot Locker	1185732	2020-01-05 00:00:00	Northeast	New York	New York	Men's Apparel	60	900	54000.0	16200.0	0.3	In-store
...	...	...	...	...	...	...	...	...	...	...	...	...	...
9647	Foot Locker	1185732	2021-01-24 00:00:00	Northeast	New Hampshire	Manchester	Men's Apparel	50	64	320.0	89.6	0.28	Outlet
9648	Foot Locker	1185732	2021-01-24 00:00:00	Northeast	New Hampshire	Manchester	Women's Apparel	41	105	430.5	137.76	0.32	Outlet
9649	Foot Locker	1185732	2021-02-22 00:00:00	Northeast	New Hampshire	Manchester	Men's Street Footwear	41	184	754.4	279.128	0.37	Outlet
9650	Foot Locker	1185732	2021-02-22 00:00:00	Northeast	New Hampshire	Manchester	Men's Athletic Footwear	42	70	294.0	123.48	0.42	Outlet
9651	Foot Locker	1185732	2021-02-22 00:00:00	Northeast	New Hampshire	Manchester	Women's Street Footwear	29	83	240.7	64.989	0.27	Outlet

9648 rows × 13 columns

Árvore de Decisão

Árvores de decisão são modelos de aprendizado de máquina que representam um conjunto de regras de decisão hierarquicamente organizadas em forma de uma árvore. Essas árvores ajudam a visualizar as possíveis decisões e resultados decorrentes de uma série de condições e eventos.

Cada nó da árvore representa uma decisão ou teste a ser tomado, e as ramificações que saem dele representam as possíveis respostas. O processo continua até que se chegue a um ponto em que não há mais testes a serem realizados, ou uma decisão final seja tomada.

Árvore de Decisão: Pré-requisitos

```python
import numpy as np # Raiz quadrada e calculos
import matplotlib.pyplot as plt # Desenhar matriz de confusão
#from datetime import datetime # para a conversão do datetime para time
from sklearn.preprocessing import LabelEncoder # Para codificar as "strings" em números

# Para Calcular erro médio quadrático (Bom para modelo de regressão) é especialmente útil quando se deseja penalizar erros grandes e outliers
from sklearn.metrics import mean_squared_error
```

```python
# Desenhar a arvore bonitinho
def plot_tree(fitted_tree, feature_name, label_names):
  import graphviz
  from sklearn.tree import export_graphviz

  dot_data = export_graphviz(clf, out_file=None,
                             feature_names=feature_name,
                             class_names=label_names,
                             filled=True, rounded=True,
                             special_characters=True)

  graph = graphviz.Source(dot_data)
  return graph
```

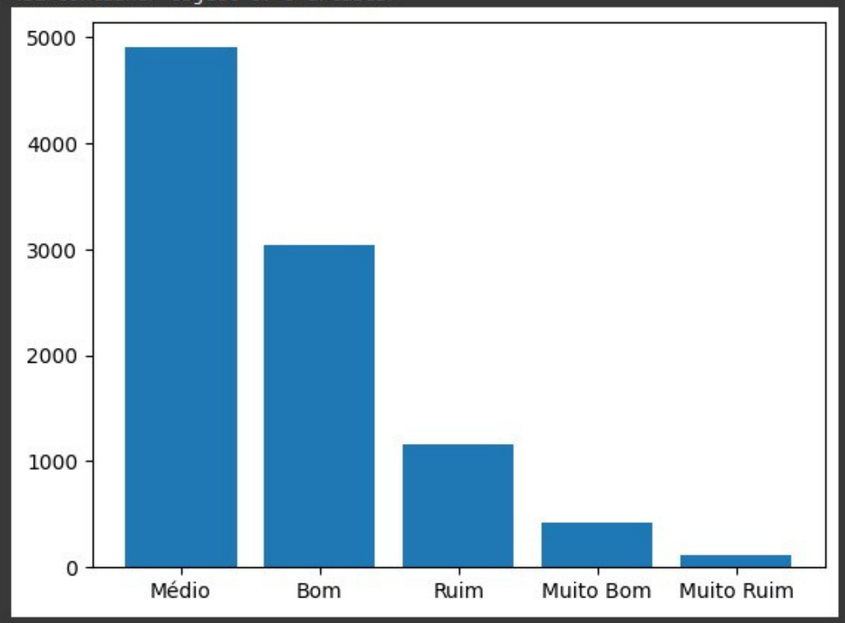
Árvore de Decisão: Categorias

Categorias

0. Muito Ruim (margem <=0.2)
1. Ruim (0.2 < margem <= 0.3)
2. Médio (0.3 < margem <= 0.45)
3. Bom (0.45 < margem <= 0.6)
4. Muito Bom (0.6 < margem)

```
[ ]  1 # Valor das categorias a serem decididas
     2 # 0. Muito Ruim (margem <=0.2)
     3 # 1. Ruim (0.2 < margem <= 0.3)
     4 # 2. Médio (0.3 < margem <= 0.45)
     5 # 3. Bom (0.45 < margem <= 0.6)
     6 # 4. Muito Bom (0.6 < margem)
     7 df['Sales Class Arvore']=df['Operating Margin'].apply(lambda x: 'Muito Ruim' if x<=0.2 else
     8                                                                 'Ruim'       if 0.2<x<=0.3 else
     9                                                                 'Médio'      if 0.3<x<=0.45 else
    10                                                                 'Bom'        if 0.45<x<=0.6 else
    11                                                                 'Muito Bom')
    12
    13 # Imprimindo proporção dos resultados
    14 print(df['Sales Class Arvore'].value_counts())
    15 plt.bar(df['Sales Class Arvore'].value_counts().index.values, df['Sales Class Arvore'].value_counts())
```

```
Médio         4902
Bom           3040
Ruim          1163
Muito Bom      423
Muito Ruim     120
Name: Sales Class Arvore, dtype: int64
<BarContainer object of 5 artists>
```



Árvore de Decisão: Encoder

[31] 0s

```
# A partir daqui precisamos converter as informações que utilizaremos em números inteiros ou ponto flutuante
df['Invoice Date TS']=df['Invoice Date'].apply(lambda x: x.timestamp()) # Converte DateTime pra timestamp(float64)

# Imprime os valores categóricos originais e os valores numéricos atribuídos
def imprimir_encoder(lb):
  print("\nRelação original-numérico: ")
  for i, valor in enumerate(lb.classes_):
      print("%s -> %d" % (valor, i))

# Utilizo LabelEncoder para codificar as Regioes, Estados, Cidades, Produtos, Metodos de Venda e Classificação final
# em números inteiros
lb = LabelEncoder()
df['Region LB Encoder']= lb.fit_transform(list(df['Region']))
df['State LB Encoder']= lb.fit_transform(list(df['State']))
df['City LB Encoder']= lb.fit_transform(list(df['City']))
df['Product LB Encoder']= lb.fit_transform(list(df['Product']))
df['Sales Method LB Encoder']= lb.fit_transform(list(df['Sales Method']))
df['Sales Class Arvore LB Encoder']= lb.fit_transform(list(df['Sales Class Arvore']))
imprimir_encoder(lb)

# Printa as informações novas do dataframe
df.iloc[:, -8:] # Pega as ultima 8 colunas adicionadas
```

Árvore de Decisão: Encoder

```
Relação original-numérico:
Bom -> 0
Muito Bom -> 1
Muito Ruim -> 2
Médio -> 3
Ruim -> 4
```

	3	Invoice Date TS	Sales Class Arvore	Region LB Encoder	State LB Encoder	City LB Encoder	Product LB Encoder	Sales Method LB Encoder	Sales Class Arvore LB Encoder
4		1.577837e+09	Bom	1	31	35	2	0	0
5		1.577923e+09	Ruim	1	31	35	1	0	4
6		1.578010e+09	Médio	1	31	35	5	0	3
7		1.578096e+09	Médio	1	31	35	4	0	3
8		1.578182e+09	Ruim	1	31	35	0	0	4
...		...	...	...	...	...	...	...	...
9647		1.611446e+09	Ruim	1	28	30	0	2	4
9648		1.611446e+09	Médio	1	28	30	3	2	3
9649		1.613952e+09	Médio	1	28	30	2	2	3
9650		1.613952e+09	Médio	1	28	30	1	2	3
9651		1.613952e+09	Ruim	1	28	30	5	2	4

9648 rows × 8 columns

Árvore de Decisão: Separação de Colunas

Nesta etapa é feita a separação das informações classificadas em **y_original** e das informações usadas para classificação **x_original**. Em seguida é feita a verificação do ganho de informação de cada atributo utilizado para classificar a coluna **y_original**.

```
# x_original é o que eu uso para classificar y_original, nele eu dropo tudo que não vou usar + a coluna da classificação final
x_original = df.drop(['Retailer','Invoice Date','Region','State','City','Product','Sales Method','Sales Class Arvore','Sales Class Arvore LB Encoder'], axis=1)

# y_original é o que será classificado
y_original = df['Sales Class Arvore LB Encoder']
```

Árvore de Decisão: Ganho de informação

```
[51]  0s
```

```python
# Calculando o ganho de informação de cada atributo
from sklearn.feature_selection import mutual_info_classif

# Calcula a pontuação de informação mútua para cada atributo
mutual_info = mutual_info_classif(x_original, y_original)
lista_col = list(x_original.columns) # lista das colunas

# Imprime a pontuação de informação mútua para cada atributo
for i, score in enumerate(mutual_info):
    print("Atributo %s: %.2f" % (lista_col[i], score))

```

```
Atributo Retailer ID: 0.04
Atributo Price per Unit: 0.15
Atributo Units Sold: 0.16
Atributo Total Sales: 0.18
Atributo Operating Profit: 0.28
Atributo Operating Margin: 1.13
Atributo Invoice Date TS: 0.01
Atributo Region LB Encoder: 0.03
Atributo State LB Encoder: 0.14
Atributo City LB Encoder: 0.14
Atributo Product LB Encoder: 0.04
Atributo Sales Method LB Encoder: 0.11
```

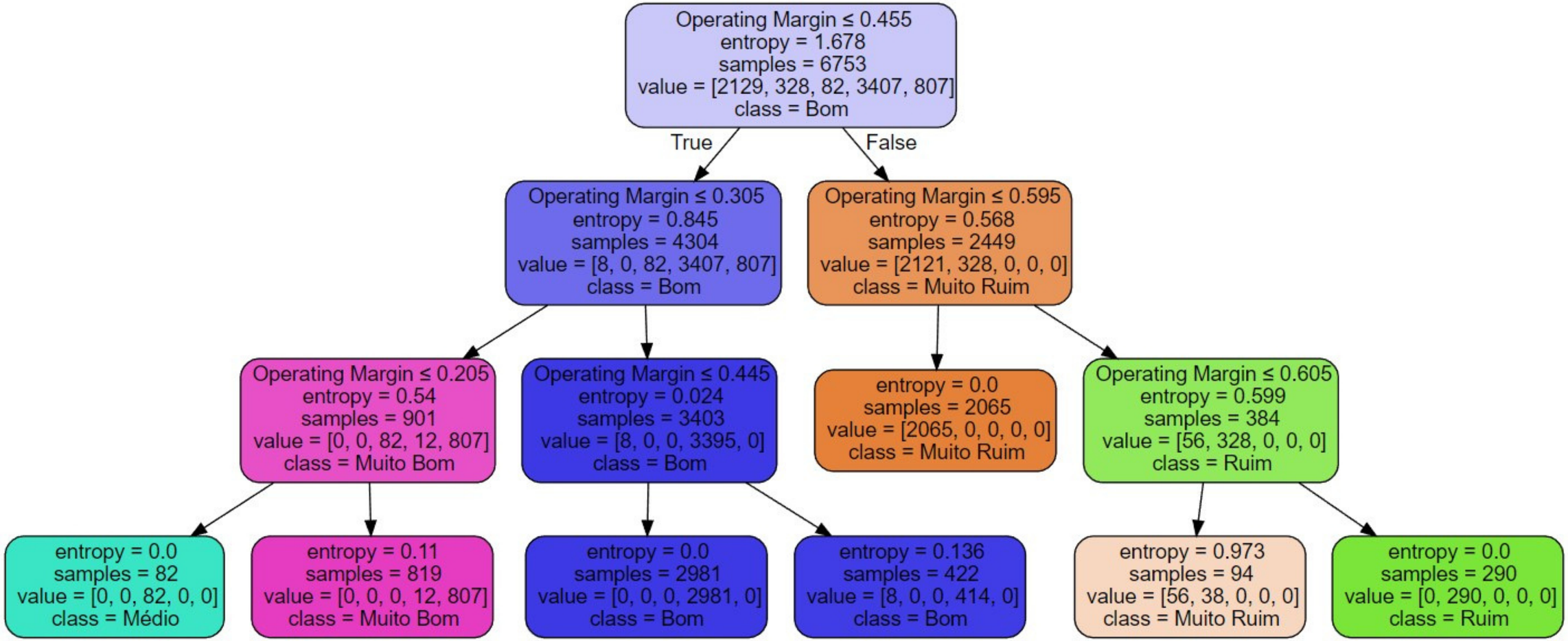
Árvore de Decisão: Treino

```
[52]  1 from sklearn.model_selection import train_test_split
      2
      3 # Realiza a divisão entre de treino e teste
      4 x_train, x_test, y_train, y_test = train_test_split(x_original, y_original, train_size = 0.7)
      5
      6 # Nomes das classes
      7 target_names = ['Muito Ruim', 'Ruim', 'Médio','Bom','Muito Bom']
```

```
 1 from sklearn.tree import DecisionTreeClassifier # Biblioteca para implementação da arvore de classificação
 2
 3 # Implementação da arvore de classificação por entropia com no maximo 3 niveis de profundidade
 4 clf = DecisionTreeClassifier(criterion = 'entropy', max_depth=3) # max_leaf_nodes sendo 3 ou 4, outros valores não são bons
 5
 6 #  Faz o treinamento
 7 clf = clf.fit(x_train, y_train)
 8
 9 # Predizendo os dados de teste
10 y_pred = clf.predict(x_test)
11
12 # Erro quadratico médio
13 print("Erro quadratico médio:",np.sqrt(mean_squared_error(y_test, y_pred)))
14
15 # Desenha Arvore de Classificação
16 plot_tree(clf, list(x_original), target_names)
```

```
Erro quadratico médio: 0.18304656485443435
```

Árvore de Decisão: Árvore Final



Árvore de Decisão: Cross-Validation

```
[54]  1 # Cross-Validation
      2 from sklearn.model_selection import cross_val_score
      3
      4 lst = []
      5 maximo = 0
      6 minimo = 1
      7 k_max, k_min = 0, 0
      8 # Fazer K-folds com os seguintes valores
      9 for cv in [3,5,7,10,12,15]:
     10   scores = cross_val_score(clf, x_original, y_original, cv=cv)
     11   lst.append(scores)
     12   print("K = %d Folds com %0.2f precisão com um desvio padrão de %0.2f" % (cv, scores.mean(), scores.std()))
     13   if maximo < np.max(scores):
     14     maximo =  np.max(scores)
     15     k_max = cv
     16   if minimo > np.min(scores):
     17     minimo = np.min(scores)
     18     k_min = cv
     19
     20 print("Maximo:", maximo, " com k =", k_max)
     21 print("Minimo:", minimo, " com k =", k_min)
     22 lst
```

Árvore de Decisão: K-Folds

```
K = 3 Folds com 0.99 precisão com um desvio padrão de 0.01
K = 5 Folds com 0.99 precisão com um desvio padrão de 0.01
K = 7 Folds com 0.99 precisão com um desvio padrão de 0.01
K = 10 Folds com 0.99 precisão com um desvio padrão de 0.01
K = 12 Folds com 0.99 precisão com um desvio padrão de 0.01
K = 15 Folds com 0.99 precisão com um desvio padrão de 0.01
Maximo: 1.0  com k = 10
Minimo: 0.9394409937888198  com k = 15
[array([0.98196517, 0.99564677, 0.99160448]),
 array([0.97409326, 0.97564767, 0.99326425, 0.9994816 , 0.98600311]),
 array([0.96446701, 0.99637418, 0.99709724, 0.99274311, 0.99709724,
        0.99927431, 0.98911466]),
 array([0.95440415, 0.99378238, 0.99481865, 0.99689119, 1.        ,
        0.9865285 , 0.99896373, 1.        , 0.99481328, 0.99377593]),
 array([0.94776119, 0.99004975, 0.99502488, 0.99502488, 1.        ,
        1.        , 0.98756219, 0.99502488, 1.        , 1.        ,
        0.99129353, 0.99502488]),
 array([0.93944099, 0.98757764, 0.99534161, 0.99377916, 0.99377916,
        1.        , 1.        , 0.98600311, 0.99377916, 0.99844479,
        1.        , 1.        , 0.99844479, 0.99066874, 0.99377916])]
```

Árvore de Decisão: Precisão e Acurácia

```
[58]  1 # Retornando a precisão média nos dados e rótulos de teste fornecidos.
      2 clf.score(x_test, y_test)
```

```
0.9913644214162349
```

```
[59]  1 # Agora vamos testar a acuracia
      2 from sklearn.metrics import accuracy_score
      3
      4 # Mostra a acuracia consideração a predição do teste e os valores do teste
      5 accuracy_score(y_test, y_pred)
```

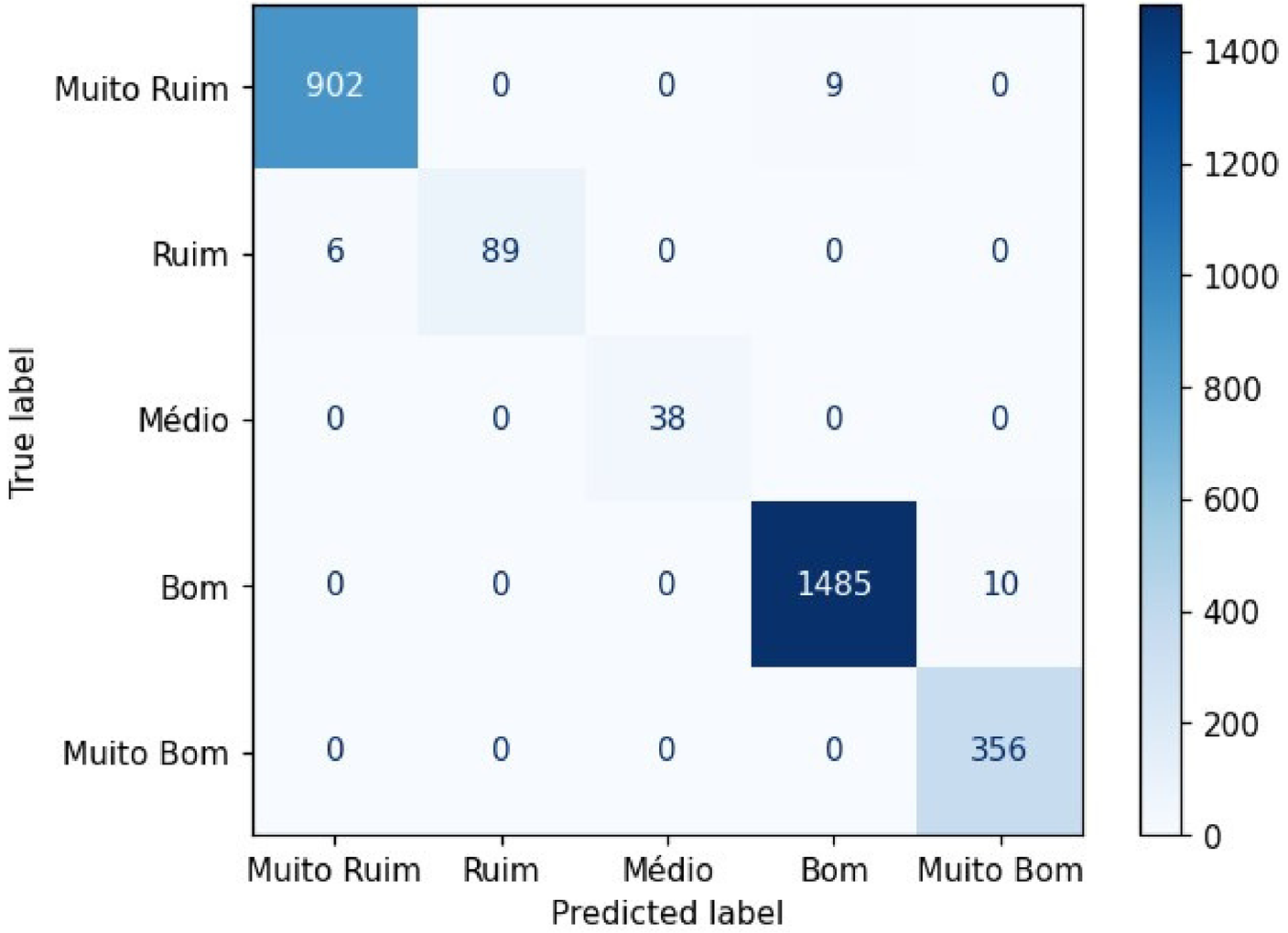
```
0.9913644214162349
```

Árvore de Decisão: Matriz de confusão

```
[60]  0s
```

```python
# Vamos desenhar e mostrar a matrix de confusão
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix

# Cria a figura e os eixos
fig, ax = plt.subplots(dpi = 110)

# y_pred já foi calculado, então basta fazer a matriz
cm = confusion_matrix(y_test, y_pred)

# Cria a classe com a matriz de confusão e os nomes das classes
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=target_names)

# Desenha a matriz
disp.plot(cmap=plt.cm.Blues, ax=ax)
```

Árvore de Decisão: Matriz de confusão



Naïve Bayes

A abordagem do Naïve Bayes requer uma perspectiva diferente sobre a base de dados. Isso se deve ao fato a suposição do modelo Naïve Bayes, que assume que as variáveis das instâncias são todas independentes das outras.

Dessa forma, é necessário escolher colunas que não possuem alta coorelação com outras colunas.

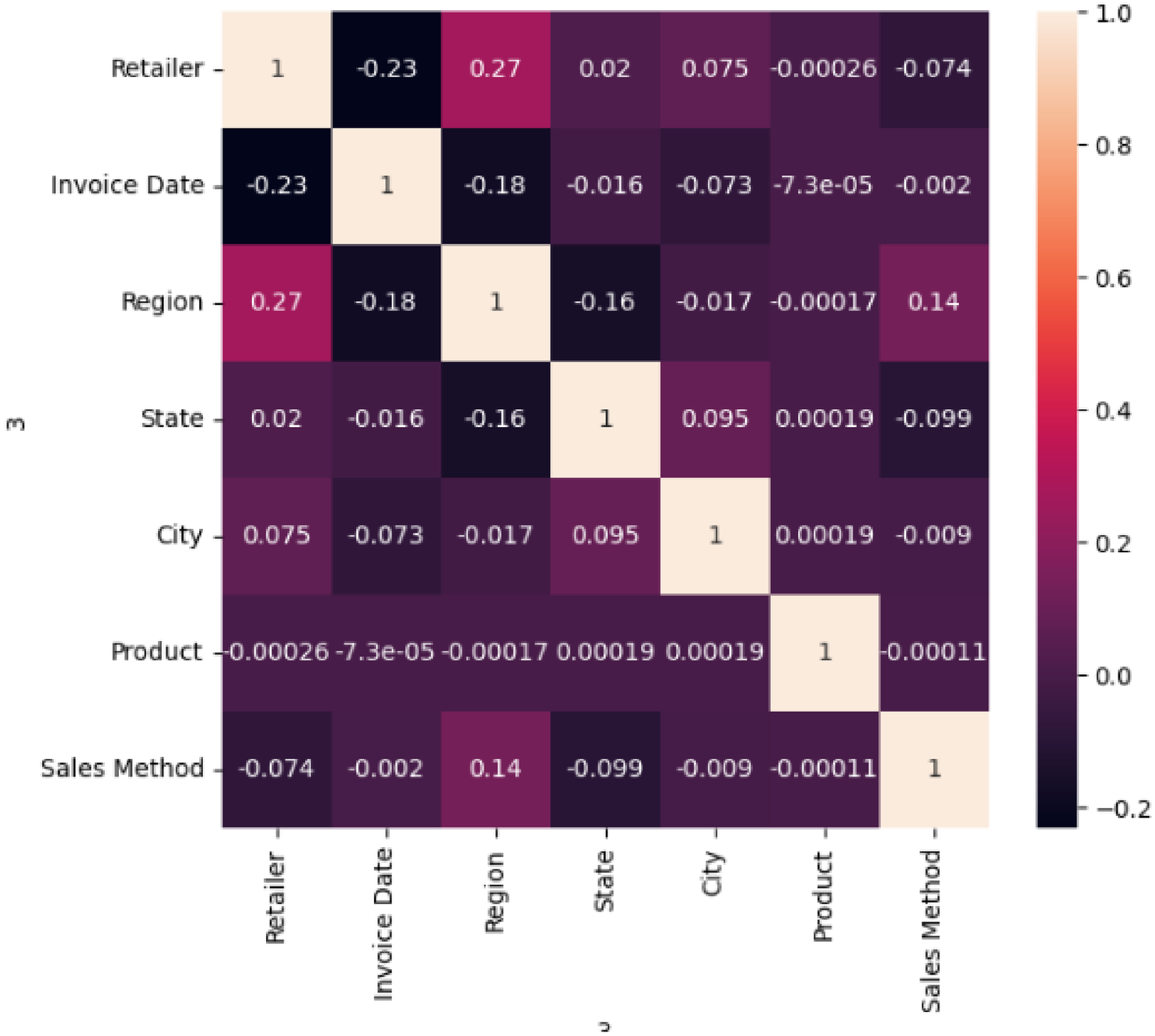
Naïve Bayes - Pré-processamento

Algumas colunas foram removidas da tabela original, como as colunas:

- **RetailerID**
- **Total Sales e**
- **Operating Profit.**

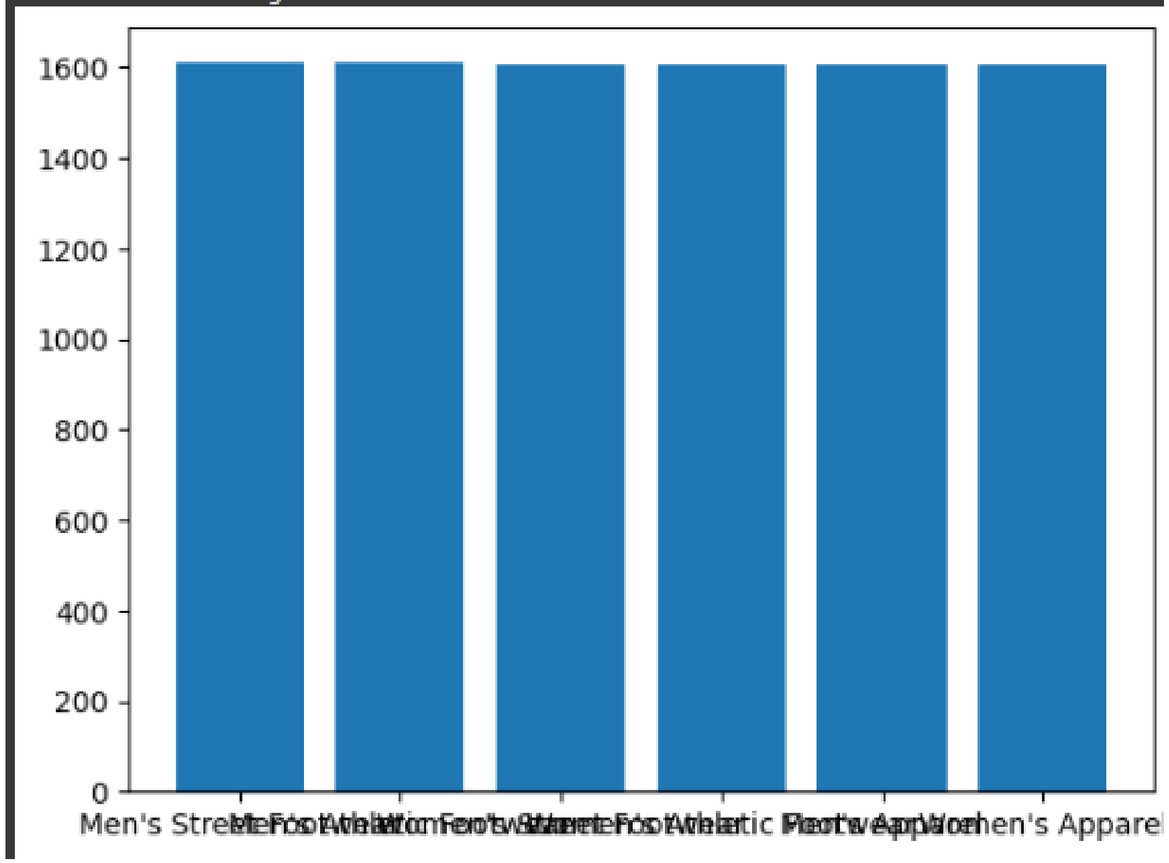
3	Retailer	Invoice Date	Region	State	City	Product	Price per Unit	Units Sold	Operating Margin	Sales Method
0	Foot Locker	2020-01-01 00:00:00	Northeast	New York	New York	Men's Street Footwear	50	1200	0.5	In-store
1	Foot Locker	2020-01-02 00:00:00	Northeast	New York	New York	Men's Athletic Footwear	50	1000	0.3	In-store
2	Foot Locker	2020-01-03 00:00:00	Northeast	New York	New York	Women's Street Footwear	40	1000	0.35	In-store

Naïve Bayes - Coorelação de Atributos

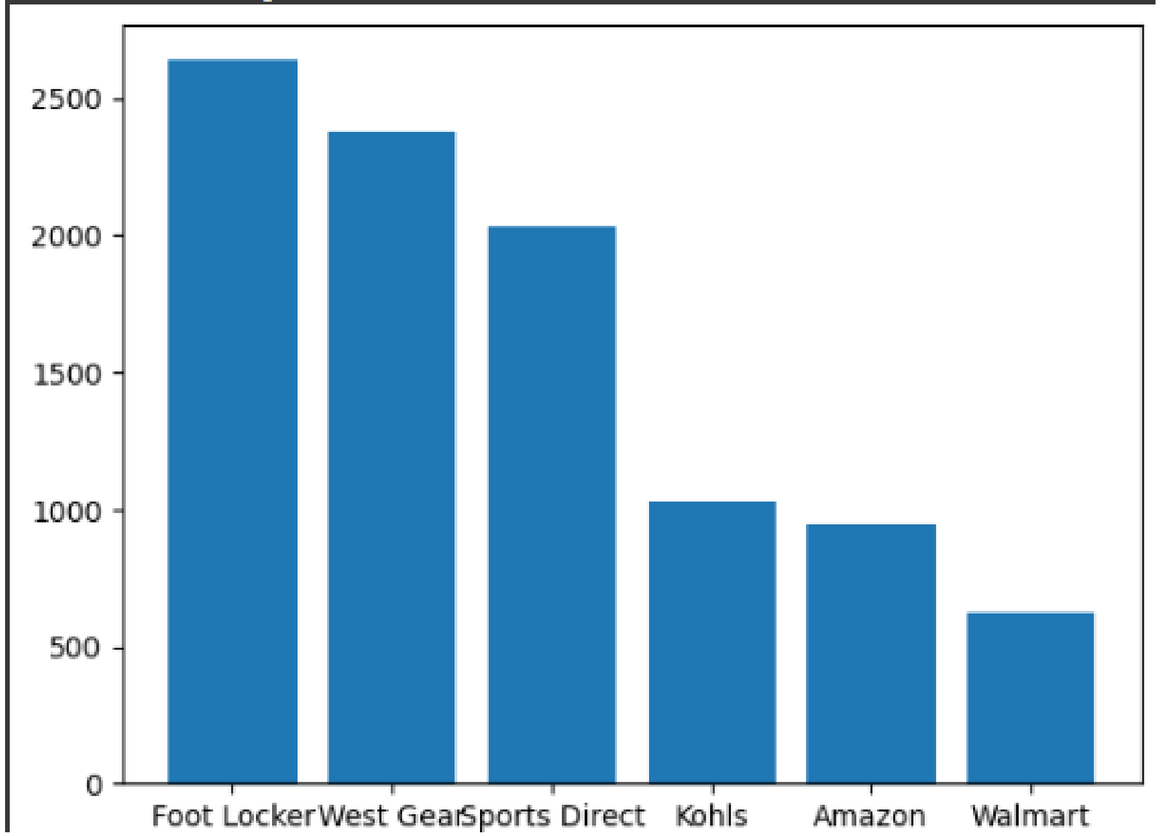


Naïve Bayes - Estatísticas

```
Men's Street Footwear        1610
Men's Athletic Footwear      1610
Women's Street Footwear      1608
Women's Apparel              1608
Women's Athletic Footwear    1606
Men's Apparel                1606
Name: Product, dtype: int64
<BarContainer object of 6 artists>
```




```
Foot Locker      2637
West Gear        2374
Sports Direct    2032
Kohl's           1030
Amazon            949
Walmart           626
Name: Retailer, dtype: int64
<BarContainer object of 6 artists>
```



Naïve Bayes - Discretização

- **Foi decidido que a coluna Operating Margin deveria ser utilizada como atributo para previsão (interpretado como meta-atributo das instâncias)**

- **Para isso, para utilizar modelos de classificação, foi decidido discretizar a coluna inteira. A discretização foi realizada da seguinte maneira:**

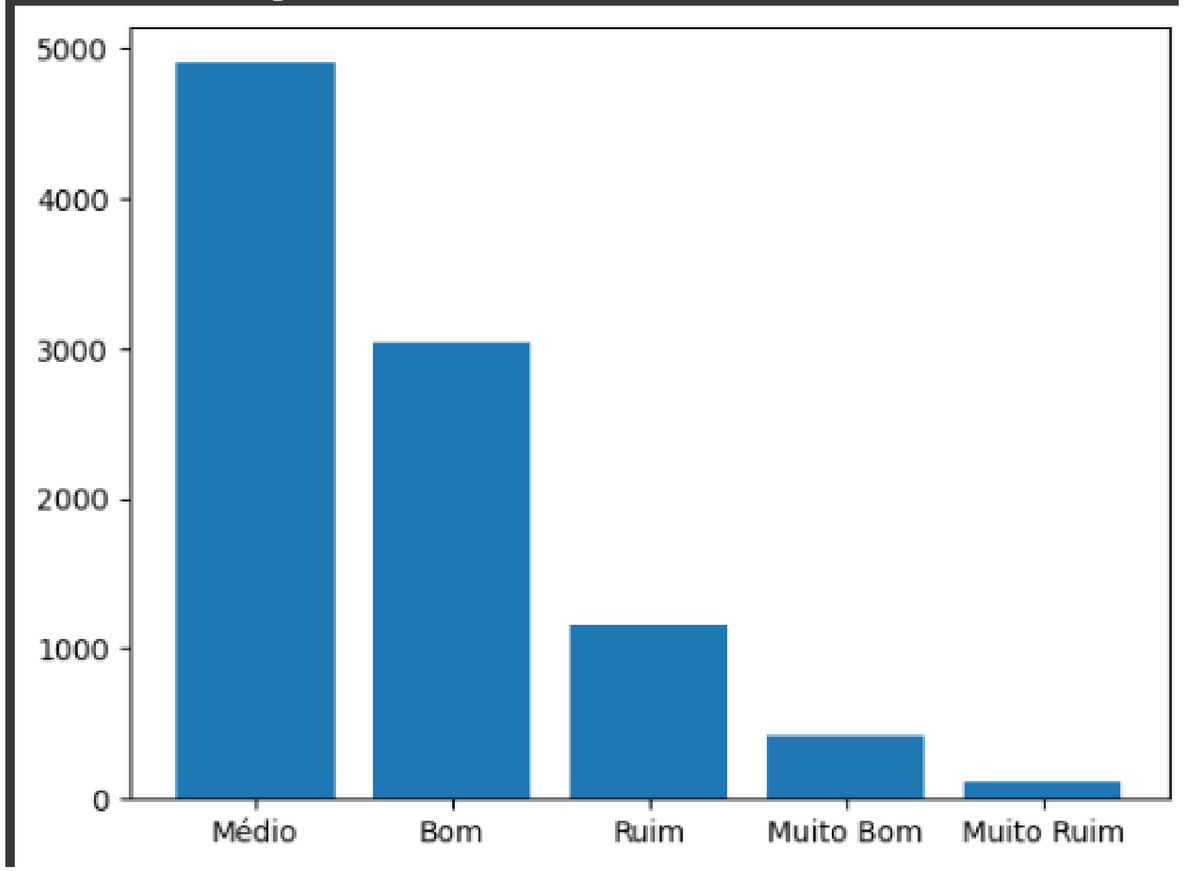
Naïve Bayes - Discretização

- Se a margem de lucro (Operating Margin) estiver:
- Operating Margin <= 0.2, então "Muito Ruim"
- 0.2 < Operating Margin <= 0.3, então "Ruim"
- 0.3 < Operating Margin <= 0.45, então "Médio"
- 0.45 < Operating Margin <= 0.6, então "Bom"
- 0.6 < Operating Margin, então "Muito Bom"

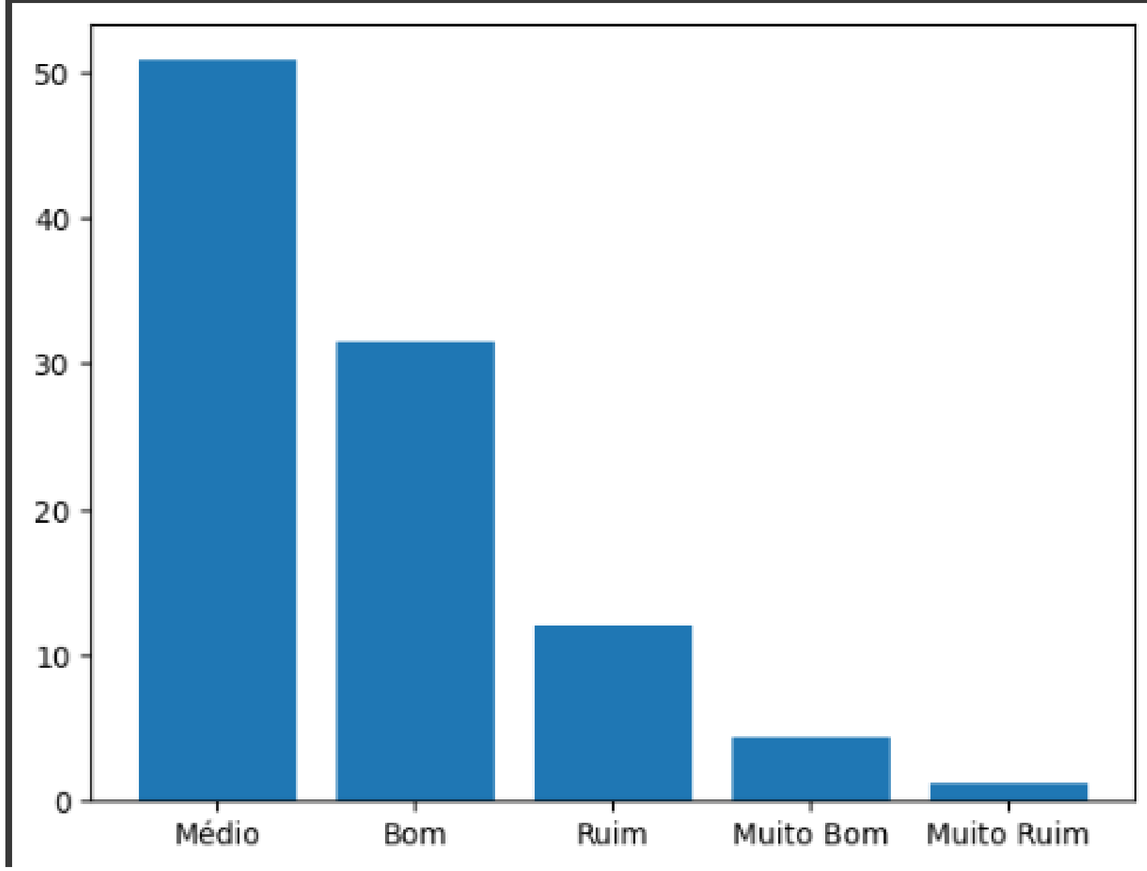
	Sales Class
0	Bom
1	Ruim
2	Médio
3	Médio
4	Ruim
...	...
9643	Ruim
9644	Médio
9645	Médio
9646	Médio
9647	Ruim

Naïve Bayes - Distribuição de Classes

```
Médio         4902
Bom           3040
Ruim          1163
Muito Bom      423
Muito Ruim     120
Name: Sales Class, dtype: int64
<BarContainer object of 5 artists>
```




```
Médio         50.808458
Bom           31.509121
Ruim          12.054312
Muito Bom      4.384328
Muito Ruim     1.243781
Name: Sales Class, dtype: float64
<BarContainer object of 5 artists>
```



Naïve Bayes - Codificação de Valores Categóricos

Os valores categóricos da tabela devem ser transformados em valores numéricos. Foi usado a classe LabelEncoder do módulo sklearn.preprocessing (from sklearn.preprocessing import LabelEncoder)

	Retailer	Invoice Date	Region	State	City	Product	Price per Unit	Units Sold	Operating Margin	Sales Method
0	1	0	1	31	35	2	50	1200	0.5	0
1	1	1	1	31	35	1	50	1000	0.3	0
2	1	2	1	31	35	5	40	1000	0.35	0
3	1	3	1	31	35	4	45	850	0.35	0
4	1	4	1	31	35	0	60	900	0.3	0
...	...	...	...	...	...	...	...	...	...	...
9643	1	382	1	28	30	0	50	64	0.28	2
9644	1	382	1	28	30	3	41	105	0.32	2
9645	1	411	1	28	30	2	41	184	0.37	2
9646	1	411	1	28	30	1	42	70	0.42	2
9647	1	411	1	28	30	5	29	83	0.27	2

Naïve Bayes - Cross Validation

Foi utilizada a técnica de Cross Validation com os valores de K iguais a 3, 5, 7, 10, 12 e 15

Também foi calculado os valores de K que resultaram em maior acurácia e o que resultou na menor acurácia (com seus respectivos valores de acurácia)

Por último, existem várias versões de classificadores Naïve Bayes, foi utilizado o do tipo Gaussiano (Gaussian Naïve Bayes)

```python
# Criação do modelo e configuração de Cross Validation
ClassificadorGaussNB = GaussianNB()

melhorK = -1
piorK = -1
melhorScore = 0
piorScore = 1

# Teste de predição com diferentes valores de K para cross validation
for i in [3,5,7,10,12,15]:
  scores = cross_val_score(ClassificadorGaussNB, X_Atributos, np.ravel(Y_Atributos), cv=i, scoring="accuracy")
  mediaScores = scores.mean()
  if melhorScore < mediaScores:
    melhorScore = mediaScores
    melhorK = i
  if piorScore > mediaScores:
    piorScore = mediaScores
    piorK = i

# Imprimindo melhores e piores resultados
print("Melhor acurácia =", melhorScore*100, "com k =", melhorK)
print("Pior acurácia   =", piorScore*100, "com k =", piorK)
```

```
Melhor acurácia = 91.04564525113584 com k = 15
Pior acurácia   = 86.18366500829188 com k = 3
```

Support Vector Machine

Popular em problemas de classificação onde não existe conhecimento prévio especializado do domínio

PROPRIEDADES

1. **SVMs constroem um separador de margem máxima**
2. **Truque de Kernel**
3. **Método não paramétrico**

Support Vector Machine - Pré processamento

Para a realização da classificação, algumas colunas foram descartadas do dataset original, por exemplo:

. "Total Sales"

. "Retailer ID"

. Por conta do dataset possuir atributos textuais, fez-se necessário transformá-los em atributos numéricos para que o algoritmo conseguisse utilizá-lo

Support Vector Machine - Pré processamento

3	Retailer	Invoice Date	Region	State	City	Product	Price per Unit	Units Sold	Operating Profit	Sales Method
0	Foot Locker	2020-01-01 00:00:00	Northeast	New York	New York	Men's Street Footwear	50	1200	30000.0	In-store
1	Foot Locker	2020-01-02 00:00:00	Northeast	New York	New York	Men's Athletic Footwear	50	1000	15000.0	In-store
2	Foot Locker	2020-01-03 00:00:00	Northeast	New York	New York	Women's Street Footwear	40	1000	14000.0	In-store
3	Foot Locker	2020-01-04 00:00:00	Northeast	New York	New York	Women's Athletic Footwear	45	850	13387.5	In-store
4	Foot Locker	2020-01-05 00:00:00	Northeast	New York	New York	Men's Apparel	60	900	16200.0	In-store
...	...	...	...	...	...	...	...	...	...	...

ANTES

	Retailer	Invoice Date	Region	State	City	Product	Price per Unit	Units Sold	Operating Profit	Sales Method
0	1	0	1	31	35	2	50	1200	30000.0	0
1	1	1	1	31	35	1	50	1000	15000.0	0
2	1	2	1	31	35	5	40	1000	14000.0	0
3	1	3	1	31	35	4	45	850	13387.5	0
4	1	4	1	31	35	0	60	900	16200.0	0
...	...	...	...	...	...	...	...	...	...	...

DEPOIS

Support Vector Machine - Classe

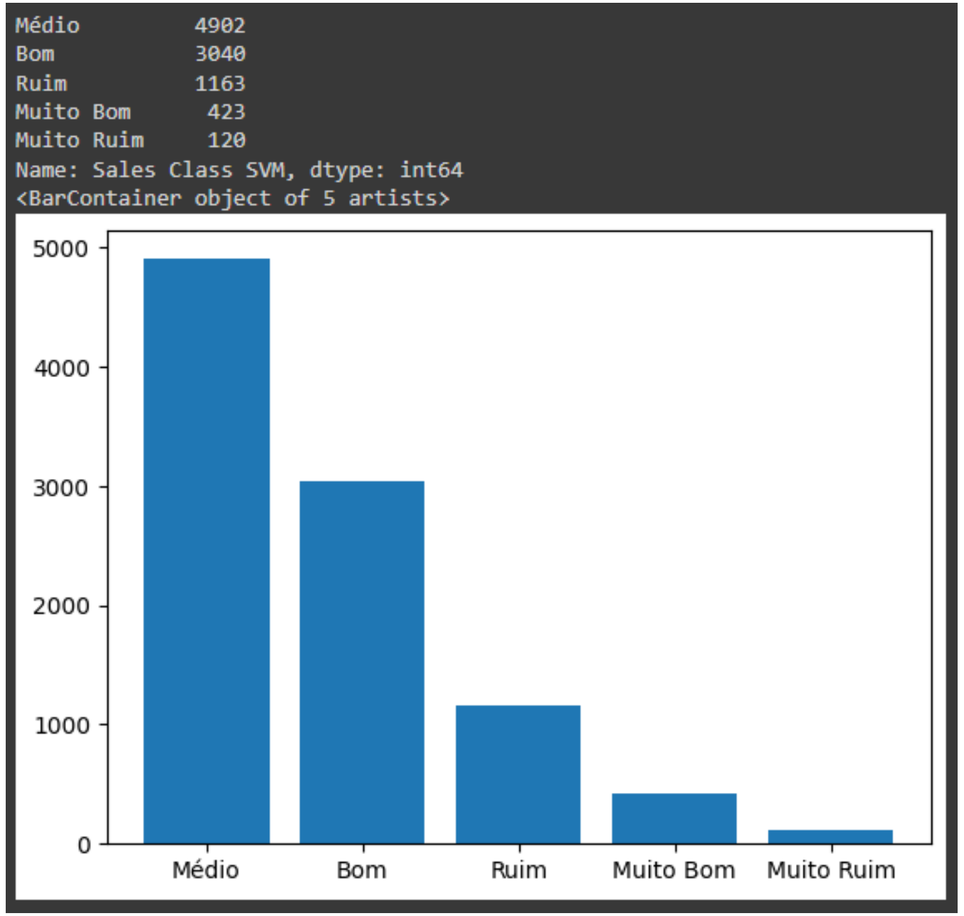
. O atributo "Operating Margin" foi definido como a classe para o problema de classificação.

. Este atributo é um atributo contínuo. Portanto, para realizar a classificação, seus valores foram discretizados da seguinte forma:

- **Se o Operating Margin estiver:**
- **Operating Margin <= 0.2, então "Muito Ruim"**
- **0.2 < Operating Margin <= 0.3, então "Ruim"**
- **0.3 < Operating Margin <= 0.45, então "Médio"**
- **0.45 < Operating Margin <= 0.6, então "Bom"**
- **0.6 < Operating Margin, então "Muito Bom"**

Support Vector Machine - Classe

```
Médio         4902
Bom           3040
Ruim          1163
Muito Bom      423
Muito Ruim     120
Name: Sales Class SVM, dtype: int64
<BarContainer object of 5 artists>
```



Support Vector Machine - Cross Validation

Para o treinamento do algoritmo, utilizou-se o cross validation com 3, 5,7,10,12 e 15 folds.

Para os testes, foram selecionados 30% dos dados. Sendo calculados por uma SVM com uma função sigmoide, avaliando sua acurácia.

Support Vector Machine - Cross Validation

```python
from sklearn.model_selection import ShuffleSplit # utilizado para personalizar o cross validation
# Criação do modelo e configuração de Cross Validation
supportVector = svm.SVC(kernel='sigmoid', C=1, random_state=42)

melhorK = -1
piorK = -1
melhorScore = 0
piorScore = 1

# Teste de predição com diferentes valores de K para cross validation
for i in [3,5,7,10,12,15]:
  crossSettings = ShuffleSplit(n_splits=i, test_size=0.3, random_state=0)
  scores = cross_val_score(supportVector, dataset, np.ravel(ClasseY), cv=crossSettings, scoring="accuracy")
  mediaScores = scores.mean()
  if melhorScore < mediaScores:
    melhorScore = mediaScores
    melhorK = i
  if piorScore > mediaScores:
    piorScore = mediaScores
    piorK = i
  if i == 3:
    fold3 = mediaScores
  if i == 5:
    fold5 = mediaScores
  if i == 7:
    fold7 = mediaScores
  if i == 10:
    fold10 = mediaScores
  if i == 12:
    fold12 = mediaScores
  if i == 15:
    fold15 = mediaScores

# Imprimindo melhores e piores resultados
print("Melhor acurácia =", melhorScore*100, "com k folds=", melhorK)
print("Pior acurácia   =", piorScore*100, "com k folds=", piorK)
print("Acurácia 3 folds   =", fold3*100)
print("Acurácia 5 folds   =", fold5*100)
print("Acurácia 7 folds   =", fold7*100)
print("Acurácia 10 folds   =", fold10*100)
print("Acurácia 12 folds   =", fold12*100)
print("Acurácia 15 folds   =", fold15*100)
```

```
Melhor acurácia = 44.72078295912493 com k folds= 3
Pior acurácia   = 43.44929681717247 com k folds= 7
Acurácia 3 folds   = 44.72078295912493
Acurácia 5 folds   = 44.359240069084635
Acurácia 7 folds   = 43.44929681717247
Acurácia 10 folds   = 43.78929188255613
Acurácia 12 folds   = 44.0587219343696
Acurácia 15 folds   = 44.50201496833621
```

Support Vector Machine - Resultados

- 3 folds = 44,72%
- 5 folds = 44,36%
- 7 folds = 43,45%
- 10 folds = 43,79%
- 12 folds = 44,06%
- 15 folds = 44,5%

Com os resultados em mãos, é possível observar que a melhor acurácia é alcançada ao utilizar 3 folds, atingindo 44,72% de acurácia.

A pior acurácia é alcançada ao utilizar 7 folds, atingindo 43,45% de acurácia.

Conclusão

De maneira geral, ao analisar comparativamente os algoritmos, percebe-se que o algoritmo de maior acurácia é a árvore de decisão. Seu maior problema dá-se pela generalização ruim.

O SVM teve uma performance abaixo de 50% em todos os seus testes, desta forma, ele não é uma abordagem muito interessante para o problema.

Desta forma, com a precisão alta de mais de 90%, o algoritmo de nayve bayes é o mais indicado para uso, dado sua boa generalização e acertos.

Referências

PROF. DR. IGARASHI, Wagner. Métodos probabilísticos de aprendizagem. Maringá: Uem, 2023. 41 slides, color.

LEARN, Scikit. Sklearn.naive_bayes.GaussianNB. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html. Acesso em: 25 mar. 2023.

LEARN, Scikit. Sklearn.tree.DecisionTreeClassifier. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html. Acesso em: 25 mar. 2023.

LEARN, Scikit. Sklearn.preprocessing.LabelEncoder. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html. Acesso em: 25 mar. 2023.

LEARN, Scikit. Cross-validation: evaluating estimator performance. Disponível em: https://scikit-learn.org/stable/modules/cross_validation.html Acesso em: 25 mar. 2023.

Referências

IGARASHI, Prof. Dr. Wagner. MÁQUINAS DE VETORES DE SUPORTE. Maringá: Uem, 2023. Color.

LEARN, Scikit. Support Vector Machines. Disponível em: https://scikit-learn.org/stable/modules/svm.html. Acesso em: 29 mar. 2023.