Universidade Federal de São Carlos

Centro de Ciências e de Tecnologia

Departamento de Computação

Aprendizado de Máquina - Trabalho 1 - Estudo e Classificação de dados

Professor: Murilo Coelho Naldi

Integrantes do grupo

Eduardo de Souza da Silva, 759495, Engenharia de Computação

Jhon Wislin Ribeiro Citron, 776852, Engenharia de Computação

João Paulo Sampaio Góes Costa 760646, Engenharia de Computação

Juan Pablo Gonzalez Jimenez, 769449, Engenharia de Computação

Lucas Henrique Marchiori, 769787, Engenharia de Computação

São Carlos, SP

2023

1. Introdução

O objetivo deste trabalho foi analisar o conjunto de dados (Dataset) sobre a satisfação de passageiros de uma companhia aérea, estudar esse conjunto de dados, observando os atributos mais relevantes para determinar a satisfação dos clientes, recorrendo às técnicas de aprendizado de máquina, E a partir desse estúdio, classificar o conjunto de dados.

2. Conjunto de dados

O conjunto de dados escolhido pelo grupo para a análise, apresenta informações relacionadas a um formulário de satisfação de passageiros de uma companhia aérea. Esse Dateset possui alguns atributos como: o atraso de chegada em minutos do Voo, Idade dos passageiros, sexo, idade, tipo de cliente e entre outras informações que expressam a relação desses passageiros com essa companhia. Esse Dataset pode ser consultado no link [1] das bibliografias.

3. Estudo do Conjunto

Todo o código aqui utilizado encontra-se disponível no *Google Colab*[1]

A primeira problemática deste *dataset*, é que o mesmo já vem dividido em dois arquivos .csv, um de teste e outro de treino, logo para a leitura e limpeza do mesmo, é necessário fazer a leitura de ambos os arquivos e concatená-los utilizando a biblioteca Pandas[2] e armazenando o resultado em uma variável tipo *Dataframe* chamada “df”. Após esta concatenação, o conjunto que pode ser visualizado através do comando *df.head()* está ilustrado na figura 1

[1] https://colab.research.google.com/drive/1irgz7UzmKD3Zo7ON4fsLBi8ZP1vacLKf?usp=sharing
[2] https://pandas.pydata.org/

	Unnamed: 0	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	...	Inflight entertainment	On-board service	Leg room service	Baggage handling	Checkin service	Inflight service	Cleanliness	Departure Delay in Minutes	Arrival Delay in Minutes	satisfaction
0	0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	4	...	5	4	3	4	4	5	5	25	18.0	neutral or dissatisfied
1	1	5047	Male	disloyal Customer	25	Business travel	Business	235	3	2	...	1	1	5	3	1	4	1	1	6.0	neutral or dissatisfied
2	2	110028	Female	Loyal Customer	26	Business travel	Business	1142	2	2	...	5	4	3	4	4	4	5	0	0.0	satisfied
3	3	24026	Female	Loyal Customer	25	Business travel	Business	562	2	5	...	2	2	5	3	1	4	2	11	9.0	neutral or dissatisfied
4	4	119299	Male	Loyal Customer	61	Business travel	Business	214	3	3	...	3	3	4	4	3	3	3	0	0.0	satisfied

5 rows × 25 columns

Figura 1: Inicio do *Dataset*

Além disto, outra forma de visualização que irá ajudar antes da aplicação do aprendizado de máquina é a visualização de dados faltantes, para isto pode-se utilizar tanto o comando df.info(), onde irá retornar a quantidade de entradas total, e quantos objetos não nulos há em cada coluna, além do tipo de entrada desta coluna(inteiro, *object*, *string*, etc…).

Outra forma de realizar isto, é com combinações de funções do Pandas e do Python, utilizando o df.isna(), que retorna True para todo valor nulo, com o comando .*sum(),* que irá somar as saídas *True*, ficando assim *df.isna().sum().* A saída de ambas as funções está representadas na figura 2 e 3 respectivamente

Figura 2: Retorno do *df.info()*

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 129880 entries, 0 to 25975
Data columns (total 25 columns):
 #   Column                            Non-Null Count   Dtype
---  ------                            --------------   -----
 0   Unnamed: 0                        129880 non-null  int64
 1   id                                129880 non-null  int64
 2   Gender                            129880 non-null  object
 3   Customer Type                     129880 non-null  object
 4   Age                               129880 non-null  int64
 5   Type of Travel                    129880 non-null  object
 6   Class                             129880 non-null  object
 7   Flight Distance                   129880 non-null  int64
 8   Inflight wifi service             129880 non-null  int64
 9   Departure/Arrival time convenient 129880 non-null  int64
 10  Ease of Online booking            129880 non-null  int64
 11  Gate location                     129880 non-null  int64
 12  Food and drink                    129880 non-null  int64
 13  Online boarding                   129880 non-null  int64
 14  Seat comfort                      129880 non-null  int64
 15  Inflight entertainment            129880 non-null  int64
 16  On-board service                  129880 non-null  int64
 17  Leg room service                  129880 non-null  int64
 18  Baggage handling                  129880 non-null  int64
 19  Checkin service                   129880 non-null  int64
 20  Inflight service                  129880 non-null  int64
 21  Cleanliness                       129880 non-null  int64
 22  Departure Delay in Minutes        129880 non-null  int64
 23  Arrival Delay in Minutes          129487 non-null  float64
 24  satisfaction                      129880 non-null  object
dtypes: float64(1), int64(19), object(5)
memory usage: 25.8+ MB
```

Figura 3: retorno do *df.isna().sum()*

```
Unnamed: 0                            0
id                                    0
Gender                                0
Customer Type                         0
Age                                   0
Type of Travel                        0
Class                                 0
Flight Distance                       0
Inflight wifi service                 0
Departure/Arrival time convenient     0
Ease of Online booking                0
Gate location                         0
Food and drink                        0
Online boarding                       0
Seat comfort                          0
Inflight entertainment                0
On-board service                      0
Leg room service                      0
Baggage handling                      0
Checkin service                       0
Inflight service                      0
Cleanliness                           0
Departure Delay in Minutes            0
Arrival Delay in Minutes            393
satisfaction                          0
dtype: int64
```

Com essas análises inicias, percebe-se que o atributo *Arrival Delay in Minutes* contem 393 entradas nulas, este problema poderia ser tratado de diversas maneiras, como a exclusão dessas linhas por um completo, ou o

preenchimento delas com algum valor previamente escolhido. Neste trabalho, optou-se pelo preenchimento dos valores nulos com a média do *Arrival Delay in Minutes,* justamente para não perder muitos dados, através da exclusão, e isto foi feito pelo comando: *df['Arrival Delay in Minutes'] = df['Arrival Delay in Minutes'].fillna(df['Arrival Delay in Minutes'].mean()).*

Percebe-se também nessa análise inicial, que há dois atributos para o Atraso (*Delay)*, o de chegada (*Arrival*) e o de Partida(*Departure).* A princípio, foi discutido que o atraso total do voo seria a soma destes dois atributos de atraso, originando assim o atributo “Delay”, resultante desta soma com o comando: *“df['Delay'] = df['Arrival Delay in Minutes'] + df['Departure Delay in Minutes']”*. Porém, após uma análise visual dos dados, foram encontrados casos onde o atraso de chegada (Arrival Delay) era de muitas horas, como na primeira linha da Tabela 1, onde o atraso de chegada é de 485 minutos. Isto significa que, após decolar, o avião ficaria voando por oito horas além do tempo previsto de viagem.

Tabela 1: Recorte dos dados de teste em que o atraso de chegada é de várias horas

	id	Gender	Customer Type	Age	Type of Travel	…	…	…	…	…	…	…	…	..	…	…	…	…	…	…	Departure Delay in Minutes	Arrival Delay in Minutes	satisfaction
696	88102	Female	Loyal Customer	26	Business travel	Business	1947	4	4	4	4	5	5	5	5	4	4	4	4	5	5	5	493	485	satisfied
279	24628	Female	Loyal Customer	14	Personal Travel	Eco	666	3	3	3	2	3	1	1	4	2	1	5	1	4	1		243	251	neutral or dissatisfied
865	102099	Female	Loyal Customer	46	Business travel	Business	2961	3	3	3	3	4	4	4	5	3	4	5	4	4	4		416	407	satisfied

Pode-se concluir que ambos os atrasos dizem respeito ao horário previsto inicialmente pela companhia, ou seja, o atraso real do voo é o atraso de chegada.

Antes de realizar qualquer análise de correlação, apenas de ver o *Database*, percebem-se atributos que podem ser removidos, pois os mesmos não irão influenciar na nossa classificação, sendo eles:

- *Unnamed: 0* criado após a concatenação dos dois .csv
- *id* - Id do passageiro, não é necessário para classificação
- *Gender*- gênero do passageiro, também não influencia

Feito esses processamentos iniciais, agora é necessário modificar os atributos que estão em formato de *string,* para isto utiliza-se *Label encoders* do SKLearn[3] no atributo alvo "*satisfaction",* que transforma os dados "*neutral or dissatisfied"* e "*satisfied"* em 0 e 1 respectivamente. Foi utilizado esta abordagem para o atributo alvo, pois ele mantém tudo em um mesmo atributo, ao invés de criar vários como o *dummies* do Pandas, ou o *One H*ot encoder do *SKLearn,* e ao criar vários atributos novos, o problema de classificação simples se tornaria um problema de classificação com múltiplos alvos, complicando assim a abordagem.

Já para o atributo *Customer Type* foi utilizado o *Label Encoder, pois* ocasionalmente esta abordagem pode gerar um atributo ordenado, onde um exerce mais peso que o outro, e como este atributo são clientes Leais e clientes não Leais, entende-se que os clientes Leais tem mais peso na avaliação que os clientes não leais, assim ao utilizar o *Label Encoder* para os clientes leais foi atribuído o valor "1" e para os não leais "0".

Outra possível abordagem é o *One Hot Encoder* que cria vários atributos, um para cada valor do atributo em que ele está sendo utilizado, e aplica 1 ou 0 se aquela linha do *dataframe* tem aquele atributo ou não, como, por exemplo, para a variável "Class", que no head era da forma representada pela tabela 2, e após a aplicação do *One Hot Encoder* ficou como na tabela 3. Esta abordagem foi aplicada tanto para o atributo *Class*, quanto para o *Type of Travel,* e foi escolhida, pois não há relação de ordenação nestes 2 atributos, então o *Label Encoder* poderia causar esta relação erroneamente, enquanto o *One Hot Encoder* não tem este risco

[3] https://scikit-learn.org/

Tabela 2: Atributo *Class* antes do *Encoder* nas primeiras linhas do data frame

	Class
0	*Eco Plus*
1	Business
2	Business
3	Business
4	Business

Tabela 3: Atributo *Class* Pós *Encoder* nas primeiras linhas do data frame

	Class_Bussines	Class_Eco	Class_Eco Plus
0	0	0	1
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0

Destaca-se também, a criação de uma variável df_original, cujos únicos pré-processamentos realizados na mesma foram o tratamento de valores *NaN* e do *Label enconder/One Hot Encoder*, para poder ser feita a comparação de resultados entre um *dataframe* pré-processado, com um que foi feito apenas o básico para o algoritmo conseguir rodar sem qualquer tipo de erro.

Após a realização destes processamentos mais triviais, realiza-se uma matriz de correlação(disponível no *Colab*[4]) para poder analisar quais atributos podem ser excluídos da classificação e quais podem ser unidos mutuamente , e a maior relação que se retira desta matriz é que os atributos *Departure Delay in Minutes, Arrival*

[4] https://colab.research.google.com/drive/1irgz7UzmKD3Zo7ON4fsLBi8ZP1vacLKf?usp=sharing

Delay in Minutes, Gate location e *Departure/Arrival time convenient* não influenciam na satisfação do cliente, assim podendo ser removidos da classificação.

Transformados todos os atributos em números, confere-se a escala deles, se estão distantes uma da outra, um método para realizar isso é utilizando a biblioteca pandas, através do comando *df.describe(),* parte de seu *output* está ilustrado na figura 4. Nele percebe-se que grande parte dos atributos deste *dataset* estão em uma escala de 0 a 5, por se tratar de notas dadas a estes serviços no formulário, porém o atributo *Flight Distance* tem um valor mínimo de 31 e um máximo de 4983, por se tratar da distância de voo. Esta discrepância na escala pode se tornar um problema dependendo do algoritmo de classificação a ser utilizado.

Figura 4: *output df.describe()*

	Flight Distance	Inflight wifi service	Ease of Online booking	Food and drink	Online boarding	Seat comfort	Inflight entertainment	On-board service	Leg room service	Baggage handling	Checkin service	Inflight service	Cleanliness
count	79661.000000	79661.000000	79661.000000	79660.000000	79660.000000	79660.000000	79660.000000	79660.000000	79660.000000	79660.000000	79660.000000	79660.000000	79660.000000
mean	1187.103652	2.728864	2.753970	3.209315	3.253653	3.444238	3.357620	3.383881	3.348117	3.633932	3.313784	3.643799	3.288037
std	996.043591	1.332745	1.404492	1.332447	1.352593	1.320083	1.335366	1.286074	1.314671	1.179111	1.265012	1.177500	1.316115
min	31.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000
25%	413.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.000000	3.000000	3.000000	3.000000	2.000000
50%	842.000000	3.000000	3.000000	3.000000	3.000000	4.000000	4.000000	4.000000	4.000000	4.000000	3.000000	4.000000	3.000000
75%	1740.000000	4.000000	4.000000	4.000000	4.000000	5.000000	4.000000	4.000000	4.000000	5.000000	4.000000	5.000000	4.000000
max	4983.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000

Uma das técnicas utilizadas para a resolução deste problema é a Padronização dos dados, pois ao contrário da Normalização de dados, ela não tem uma limitação de escala (apenas de 0 a 1, por exemplo), então mesmo que tenha *outliers* em nosso database eles não serão removidos.

Para a padronização de dados, optou-se pela utilização do *StandardScaler* da biblioteca SKlearn, que padroniza subtraindo a média e, em seguida, escalando para variância unitária. Onde o *Standard Score* de uma amostra x é calculado da seguinte forma:

$$z = \frac{x - \mu}{\sigma}$$

Onde z é o *Standard Score* (*z-score*) da amostra, x é o valor da amostra, μ é a média dos dados e σ é o desvio padrão dos dados.

Após a aplicação do *StandardScaler,* a diferença de escala dos dados apresentados na figura 4 diminuiu consideravelmente, seus dados encontram-se ilustrados na figura 5.

Figura 5: *output df.describe()*

	Flight Distance	Inflight wifi service	Ease of Online booking	Food and drink	Online boarding	Seat comfort	Inflight entertainment	On-board service	Leg room service	Baggage handling	Checkin service	Inflight service	Cleanliness
count	7.966100e+04	7.966100e+04	7.966100e+04	7.966000e+04	7.966000e+04	7.966000e+04	7.966000e+04	7.966000e+04	7.966000e+04	7.966000e+04	7.966000e+04	7.966000e+04	7.966000e+04
mean	4.218962e-17	-1.605525e-18	-9.035535e-17	6.511376e-17	7.051017e-17	-1.594841e-16	4.870152e-17	1.416001e-16	-7.858249e-17	-1.050071e-16	1.370065e-16	-1.010601e-16	-1.141721e-16
std	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00
min	-1.160703e+00	-2.047565e+00	-1.960843e+00	-2.408601e+00	-2.405509e+00	-2.609123e+00	-2.514397e+00	-2.631188e+00	-2.546750e+00	-2.233843e+00	-2.619584e+00	-3.094539e+00	-2.498306e+00
25%	-7.771834e-01	-5.468926e-01	-5.368310e-01	-9.075946e-01	-9.268577e-01	-1.094058e+00	-1.016671e+00	-1.076058e+00	-1.025447e+00	-5.376388e-01	-2.480495e-01	-5.467537e-01	-9.786720e-01
50%	-3.464766e-01	2.034435e-01	1.751748e-01	-1.570913e-01	-1.875322e-01	4.210080e-01	4.810547e-01	4.790722e-01	4.958558e-01	3.104633e-01	-2.480495e-01	3.025082e-01	-2.188551e-01
75%	5.550960e-01	9.537795e-01	8.871807e-01	5.934120e-01	5.517934e-01	1.178541e+00	4.810547e-01	4.790722e-01	4.958558e-01	1.158565e+00	5.424620e-01	1.151770e+00	5.409618e-01
max	3.810998e+00	1.704116e+00	1.599187e+00	1.343915e+00	1.291119e+00	1.178541e+00	1.229918e+00	1.256637e+00	1.256507e+00	1.158565e+00	1.332974e+00	1.151770e+00	1.300779e+00

4. Classificação

Após a análise e tratamento dos dados, foi feita a classificação, ou seja, o processo de predizer os dados que queremos. Para o nosso *dataset*, o objetivo era predizer a satisfação dos clientes da companhia aérea. Para isso, recorremos a três métodos de classificação. O método da árvore de decisão, o método de *knn* e o método Bayesiano Ingênuo que serão abordados a seguir na implementação no *dataset* nos tópicos 4.1, 4.2 e 4.3 abaixo.

Também foi utilizado o Kfold, onde o mesmo divide os exemplos em folds(estratos) de mesmo tamanho , onde os k-1 fold são utilizados para treinamento e o remanscente em teste, e assim sucessivamente até acabar os folds, isto é feito para evitar que na divisão de teste e treinamento aleatoriamente tenha sido pego o melhor ou o pior cenário, e evitar *underfitting* e *overfitting*.

Para analisar a influência do pré-processamento de dados no dataset, e nos métodos de classificação. A aplicação dos métodos foi feita no dataset pré-processado (df) e não processado (df_original). Esses datasets foram divididos nos subconjuntos X e Y, e X_original e Y_original respectivamente.

Como foi aplicado mais de um método de classificação, para o dataset pré-processado e não processado, foram definidas duas funções. A função scores, responsável por fazer a predição, conforme o método escolhido é

passado como parâmetro na chamada da função, juntamente aos subconjuntos. A função pode ser observada abaixo (Figura 6), onde basicamente ela divide os subconjuntos em conjuntos de treino e teste e faz o treinamento conforme o modelo adotado, retornando a predição e o atual. A segunda função (Figura 7), foi a plot_confusion_matrix, que recebe a classe atual e a predição, e cria a matriz de confusão e calcula os valores de acurácia, precisão, revocação e medida-f.

Figura 6: *scores*

```
def scores(model, X, y):
    kf = KFold(10,shuffle=False) #shuffle false pois queremos sempre repetir os mesmos valores para todo
    predicted = np.empty([0], dtype=int)
    actual = np.empty([0], dtype=int)
    for _, (train_index, test_index) in enumerate(kf.split(X)):
        X_train, X_test, y_train, y_test = X[train_index], X[test_index], y[train_index], y[test_index]
        model = model.fit(X_train, y_train)
        predict = model.predict(X_test)

        predicted = np.append(predicted, predict)
        actual = np.append(actual, y_test)

    return actual, predicted
```

Figura 7: *plot_confusion_matrix*

```
def plot_confusion_matrix(actual_classes, predicted_classes):

    matrix = confusion_matrix(actual_classes, predicted_classes)

    plt.figure(figsize=(12.8,6))
    fig = px.imshow(matrix, text_auto=True,aspect="auto", labels=dict(x='Pr
    fig.show()

    AC1 = (matrix[0,0]+matrix[1,1])/matrix.sum()
    Prec1 = matrix.diagonal()/matrix.sum(axis=0)
    Rev1 = matrix.diagonal()/matrix.sum(axis=1)
    MF1 = 2*(Prec1*Rev1)/(Prec1+Rev1)

    print('Acurácia: {:.2f}'.format(AC1))
    print('Precisão: {:.2f}'.format(Prec1.mean()))
    print('Revocação: {:.2f}'.format(Rev1.mean()))
    print('Medida-F: {:.2f}'.format(MF1.mean()))
```

4.1 Árvore de Decisão

Conforme feita as definições descritas anteriormente, tópico 4. Para o modelo de árvore de decisão, foram inicialmente utilizados os dados com pré-processamento. Abaixo (Figura 8), é possível observar as chamadas das

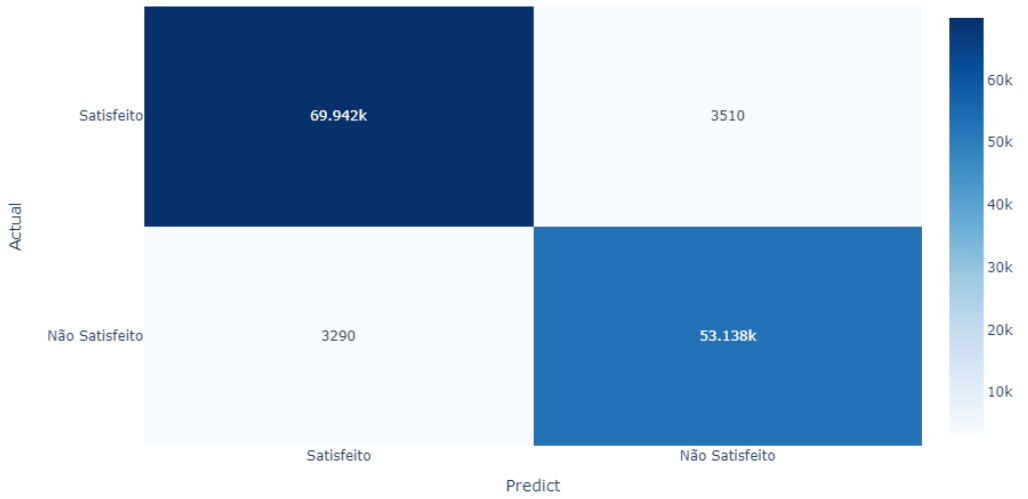
funções definidas anteriormente para a predição e matriz de confusão, com o modelo de árvore de decisão como modelo entrada.

Figura 8: *Modelo de árvore de decisão com pré-processamento*

```
tree = DecisionTreeClassifier(criterion='entropy', random_state=42)
actual, predicted  = scores(tree, X, y)
plot_confusion_matrix(actual, predicted)
```

Por conseguinte, após a aplicação do método, foi obtido a matriz de confusão abaixo (Figura 9). Onde, conforme a coloração de cada posição da matriz for mais próxima do azul-escuro, mais acertos ou erros ocorreram. No caso, o verdadeiro positivo (VP), chegou bem próximo a essa tonalidade, significado que houve um alto número de acertos na predição positiva, conforme a escala de cores. O mesmo ocorreu para o verdadeiro negativo (VN), possuindo uma tonalidade um pouco mais clara em relação ao VP.

Figura 9: *Matriz de confusão árvore de decisão com pré-processamento*



Juntamente a matriz de confusão, também foi obtido os valores de precisão, acurácia, revocação e medida-f (Figura 10). Onde, todos obtiveram um valor de 95%.

Figura 10: Resultados árvore de decisão com pré-processamento

```
Acurácia: 0.95
Precisão: 0.95
Revocação: 0.95
Medida-F: 0.95
```

Após isso, o mesmo processo descrito anteriormente neste tópico foi replicado para o conjunto sem pré-processamento. Com a matriz de confusão e os valores de acurácia, precisão, revocação e medida-f sendo mostrados abaixo (Figura 11 e Figura 12).

Figura 11: Matriz de confusão árvore de decisão sem pré-processamento

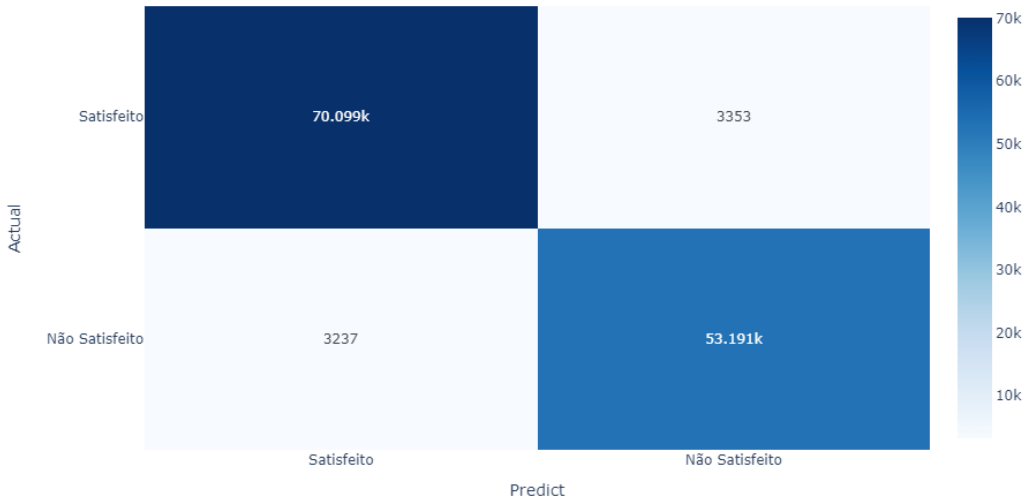


Figura 12: Resultados árvore de decisão sem pré-processamento

```
Acurácia: 0.95
Precisão: 0.95
Revocação: 0.95
Medida-F: 0.95
```

Comparando os resultados, foi possível observar que houve pouca discrepância entre os resultados obtidos com e sem pré-processamento. Isso ocorre, provavelmente pelo fato dos dados estarem balanceados ou talvez pela qualidade dos dados que estão sendo utilizados. Mais afrente será possível observar como o processamento influencia em outros métodos.

4.2 Knn

O mesmo processo descrito anteriormente no tópico 4.1, foi replicado para o modelo de Knn, para o dataset com pré-processamento e sem pré-processamento. Os resultados obtidos para Knn se encontram nas figuras abaixo (Figura 13, Figura 14, Figura 15 e Figura 16).

Figura 13: *Matriz de confusão Knn com pré-processamento*

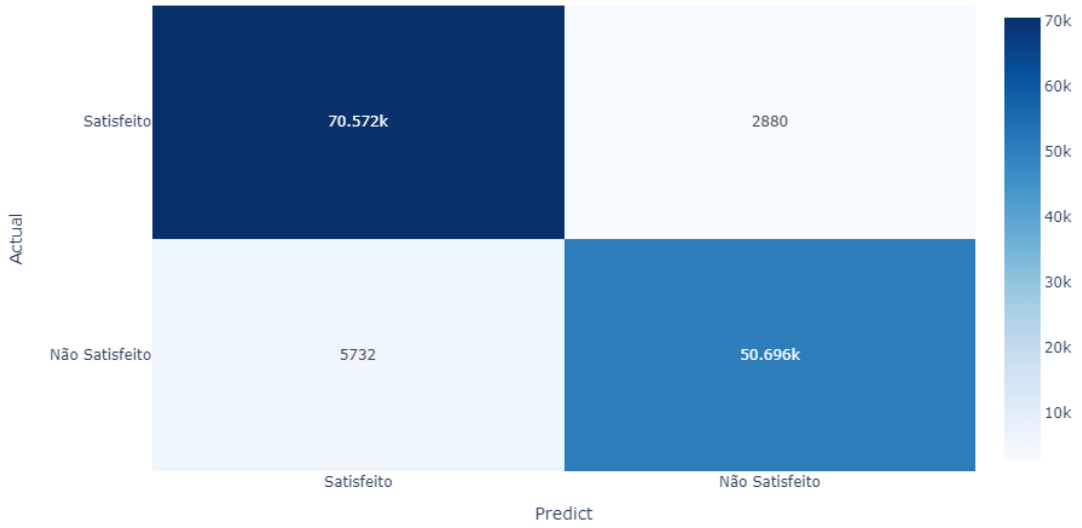


Figura 14: *Resultados Knn com pré-processamento*

```
Acurácia: 0.93
Precisão: 0.94
Revocação: 0.93
Medida-F: 0.93
```

Figura 14: *Matriz de confusão Knn sem pré-processamento*

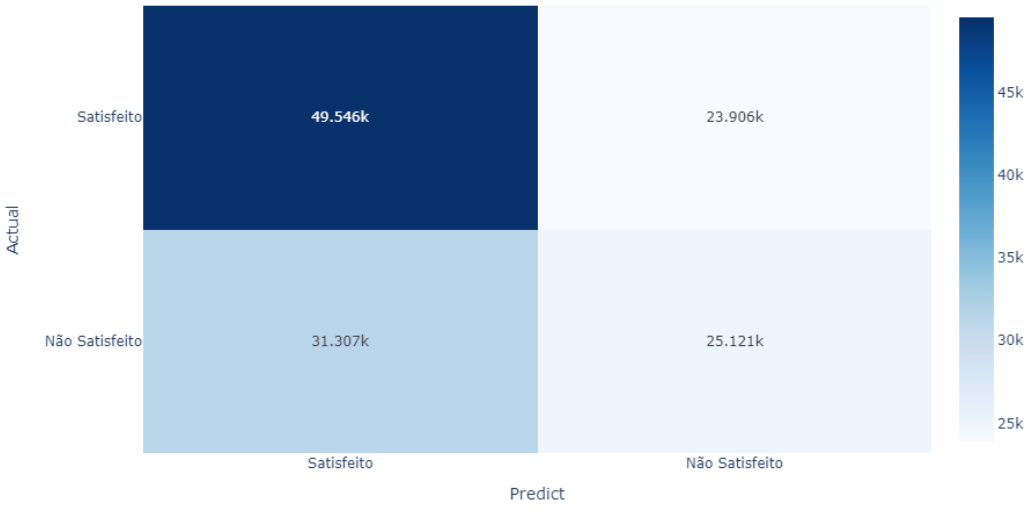


Figura 15: *Resultados Knn sem pré-processamento*

```
Acurácia: 0.57
Precisão: 0.56
Revocação: 0.56
Medida-F: 0.56
```

Comparando os resultados obtidos, foi possível observar uma grande discrepância entre os resultados para o dataset com pré-processamento e sem pré-processamento. Essa discrepância muito provavelmente é devido à sensibilidade do modelo de Knn em relação à escala dos dados, dado que foi feita a normalização do dataset. Outro ponto que influenciou nessa discrepância, foi o tratamento feito para os valores faltantes no dataset, como o Knn possui essa sensibilidade, provavelmente foi outro fator que pode ter influenciado. Dessa forma o dataset com pré-processamento obteve melhores resultados em relação ao sem pré-processamento.

4.3 Algoritmo Ingênuo de Bayes

O mesmo processo descrito nos tópicos 4.1 e 4.2, foi replicado para o algorítimo de Bayes. Sendo obtido os resultados apresentados nas figuras abaixo (Figura 16 a Figura 21).

Figura 16: *Matriz de confusão Bayes com pré-processamento*

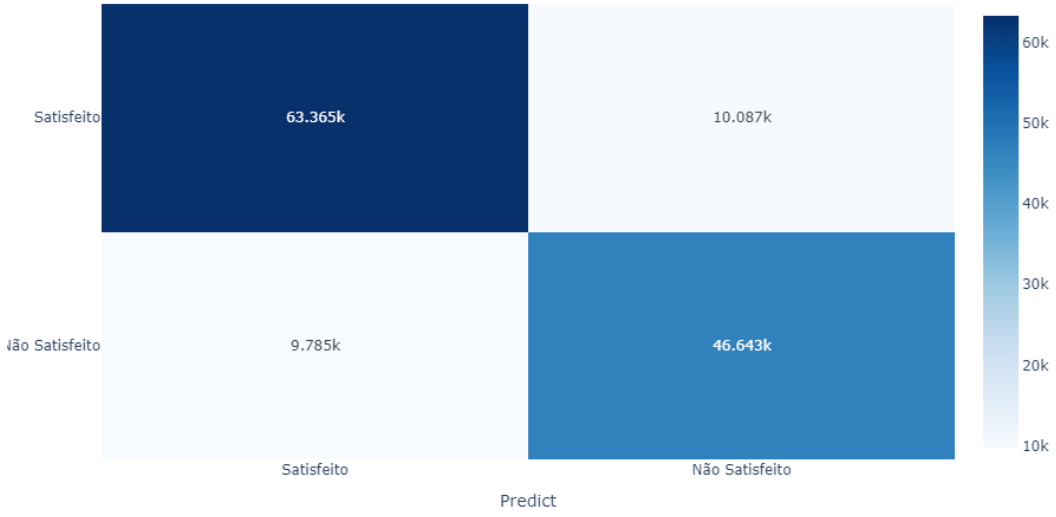


Figura 17: *Resultados Bayes Gaussiano com pré-processamento*

```
Acurácia: 0.85
Precisão: 0.84
Revocação: 0.84
Medida-F: 0.84
```

Figura 18: *Matriz de confusão Bayes Gaussiano sem pré-processamento*

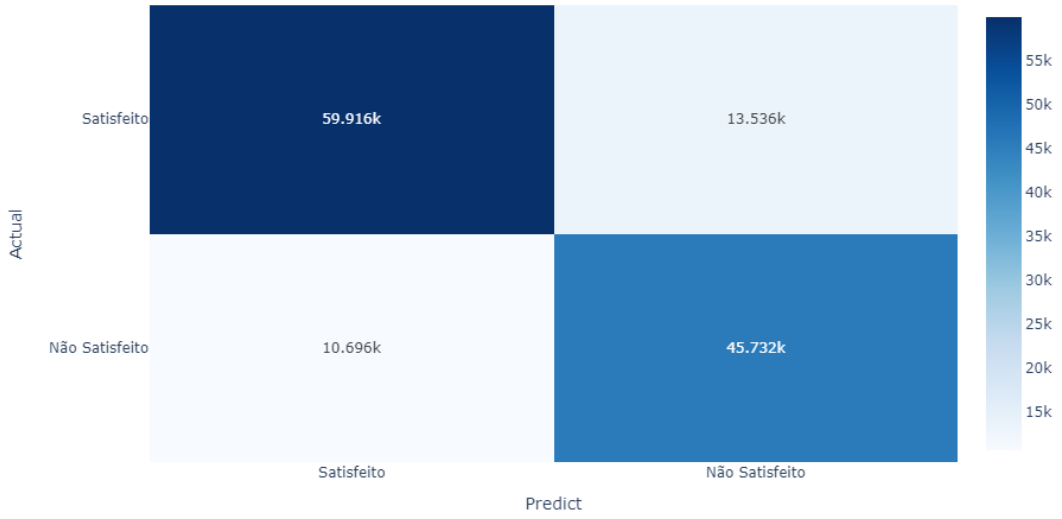
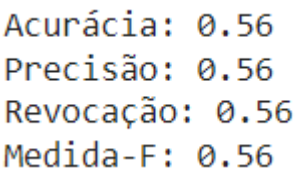


Figura 19: *Resultados Bayes sem pré-processamento*

```
Acurácia: 0.81
Precisão: 0.81
Revocação: 0.81
Medida-F: 0.81
```

Figura 20: *Matriz de confusão Bayes Multinomial sem pré-processamento*

Figura 21: *Resultados Bayes Multinomial sem pré-processamento*



Analisando os dados obtidos para o algoritmo de bayes, foi possível observar uma pequena discrepância nos resultados obtidos para o dataset com e sem pré-processamento no método Gaussiano. Essa discrepância, provavelmente é devido ao algoritmo ingênuo bayes, assim como o Knn, ser sensível à escala dos dados. Outro fator que pode ser considerado é a distribuição dos dados. Como o bayes considera que os dados não possuem uma dependência, existe essa necessidade de tratamento no pré-processamento.

Para o método multinomial, que é mais utilizado para dados discretos, o resultado foi o mais baixo de todos os algoritmos. Isto pode acontecer pois alguns atributos como idade, distância e atraso contêm muitos valores distintos, e com distribuição normal, o que prejudica o algoritmo multinomial. Por outro lado, os atributos de satisfação, com entradas de 1 a 5, e os atributos binários (0 ou 1) funcionam melhor neste algoritmo, devido a sua distribuição multinomial. Retirando os atributos ID, idade, distância de voo e Atraso, o algoritmo multinomial tem um resultado muito melhor do que o anterior, como pode ser observado nas figuras 22 e 23.

Figura 22: *Resultados Bayes Multinomial retirando dados de distribuição normal*

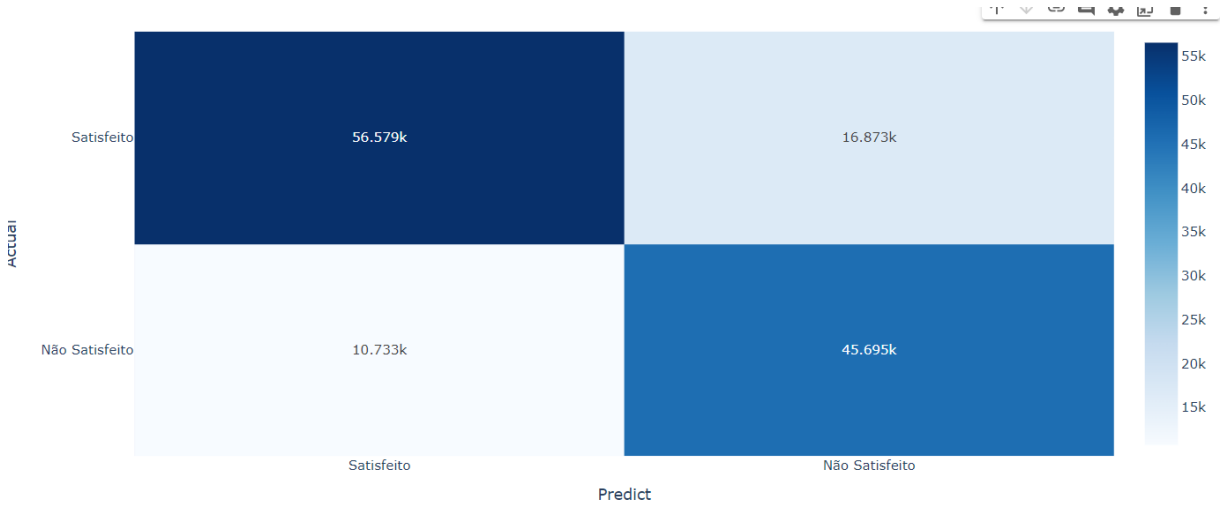


Figura 23: *Resultados Bayes Multinomial retirando dados de distribuição normal*

```
Acurácia: 0.79
Precisão: 0.79
Revocação: 0.79
Medida-F: 0.79
```

5. Conclusão

Por fim, analisando os resultados obtidos anteriormente para os métodos de classificação (Árvore de Decisão, Knn e Bayes). Foi possível observar que o modelo aplicado para a árvore de decisão, obteve os melhores resultados de acurácia, precisão, revocação e media em relação aos outros modelos. Isso devido à árvore de decisão ser um modelo capaz de detectar as relações entre diferentes atributos, além de ser menos sensível a outliers.

6. Bibliografia

[1] https://www.kaggle.com/datasets/teejmahal20/airline-passenger-satisfaction

[2]
https://ava2.ead.ufscar.br/pluginfile.php/953393/mod_resource/content/4/Trabalho1.pdf