# Inteligência Computacional — COC 361
# 2021/2

# Trabalho Computacional

Gabriel de Oliveira da Fonseca, Gustavo Pires Machado

March 13, 2022

## 1 Introdução

Para o presente trabalho, o conjunto de dados escolhido reúne dados coletados entre 1º de Julho de 2015 e 31 de Agosto de 2017 por uma rede hoteleira que incluem diversos atributos relacionados às reservas efetuadas por seus clientes. Utilizando-se das diversas metodologias de Inteligência Computacional discutidas ao longo do curso, o trabalho tem por objetivo a construção de um modelo de classifcação para a previsão de reservas canceladas. A partir deste modelo, espera-se que a rede hoteleira possa se beneficiar de uma maior previsibilidade das reservas que serão efetivamente concretizadas, por fim aumentando sua margem de lucro.

## 2 Dataset e Tecnologia

O dataset escolhido possui 36 colunas, foi retirado da plataforma Kaggle [1], e conta com 119390 entradas. Os dados presentes compreendem diversos aspectos relacionados às reservas, como número de hóspedes em diferentes faixas etárias, dados pessoais dos clientes (como nome, e-mail e telefone), além de nuances mais promissoras, como indicadores de que um determinado cliente já cancelou reservas anteriormente. De forma geral, este dataset se encontra razoavelmente organizado e possui um baixo número de dados faltantes (nulos).

Um outro importante aspecto a ser analisado diz respeito à qual classe pertence o conjunto de dados: balanceada ou desbalanceada. Isso pode ser visto através da predominância das classes alvo, que nesse estudo são as reservas canceladas ou não. Contando cada uma das classes, chegamos à proporção de 39% das reservas canceladas para 61% não canceladas. Dessa forma, conclui-se que o dataset é razoavelmente balanceado.

Além disso, conforme disposto na tabela abaixo, 20 de suas colunas são numéricas, e portanto 16 categóricas.

| # | Coluna | # entradas não-nulas | Tipo |
|---|---|---|---|
| 0 | hotel | 119390 | object |
| 1 | is_canceled | 119390 | int64 |
| 2 | lead_time | 119390 | int64 |
| 3 | arrival_date_year | 119390 | int64 |
| 4 | arrival_date_month | 119390 | object |
| 5 | arrival_date_week_number | 119390 | int64 |
| 6 | arrival_date_day_of_month | 119390 | int64 |
| 7 | stays_in_weekend_nights | 119390 | int64 |
| 8 | stays_in_week_nights | 119390 | int64 |
| 9 | adults | 119390 | int64 |
| 10 | children | 119386 | float64 |
| 11 | babies | 119390 | int64 |
| 12 | meal | 119390 | object |
| 13 | country | 118902 | object |
| 14 | market_segment | 119390 | object |
| 15 | distribution_channel | 119390 | object |
| 16 | is_repeated_guest | 119390 | int64 |
| 17 | previous_cancellations | 119390 | int64 |
| 18 | previous_bookings_not_canceled | 119390 | int64 |
| 19 | reserved_room_type | 119390 | object |
| 20 | assigned_room_type | 119390 | object |
| 21 | booking_changes | 119390 | int64 |
| 22 | deposit_type | 119390 | object |
| 23 | agent | 103050 | float64 |
| 24 | company | 6797 | float64 |
| 25 | days_in_waiting_list | 119390 | int64 |
| 26 | customer_type | 119390 | object |
| 27 | adr | 119390 | float64 |
| 28 | required_car_parking_spaces | 119390 | int64 |
| 29 | total_of_special_requests | 119390 | int64 |
| 30 | reservation_status | 119390 | object |
| 31 | reservation_status_date | 119390 | object |
| 32 | name | 119390 | object |
| 33 | email | 119390 | object |
| 34 | phone-number | 119390 | object |
| 35 | credit_card | 119390 | object |

Table 1: Atributos do dataset.

Em relação às tecnologias utilizadas para a implementação da solução deste trabalho, foram adotadas essencialmente a plataforma Google Collaboratory [2], onde o trabalho foi desenvolvido no formato de um notebook Jupyter, e diversas bibiliotecas em Python. Dentre as bibliotecas utilizadas, vale citar o Pandas, que implementa abstrações para facilitar o manuseio de data frames, bem como Scikit-Learn, que disponibiliza uma vasta gama de ferramentas para visualização de dados e treinamento de modelos. Especificamente, também foi utilizada a API Keras do TensorFlow para o treinamento dos modelos de redes neurais.

# 3 Metodologia

A solução construída para o problema consistiu na implementação de um ferramental de modelos de classificação baseados em diversos algoritmos apresentados ao longo da disciplina de Inteligência Computacional. Como tarefa preliminar, conforme apresentado posteriormente na seção 4, foi realizada também uma série de ajustes de pré-processamento no dataset a fim de otimizar os modelos treinados. Nas subseções a seguir, são apresentados detalhadamente os modelos adotados para construção da solução.

## 3.1 Regressão Logística

A Regressão Logística é um modelo de classificação binária, cuja função discriminante é definida pela probablidade *a posteriori* como [3]:

$$\begin{aligned} P(v(t)) = 1|\mathbf{x}(t), \boldsymbol{\theta}) = g(\mathbf{x}(t), \boldsymbol{\theta}) \\ P(v(t)) = 0|\mathbf{x}(t), \boldsymbol{\theta}) = 1 - g(\mathbf{x}(t), \boldsymbol{\theta}) \end{aligned} \tag{3.1}$$

Tal função de probabilidade em representar os valores observados é maximizada pelo modelo através da seguinte função objetivo:

$$l(\boldsymbol{\theta}) = -\sum_{t=1}^{N} v(t) log(\hat{v}(t)) + (1 - v(t)) log(1 - \hat{v}(t)) \tag{3.2}$$

Onde,

$$\hat{v}(t) = \frac{1}{1 + e^{-\theta Tx}} \tag{3.3}$$

Pela equação (3.2), observa-se que quando $v(t) = 0$, apenas o segundo termo do somatório prevalece e o custo tende a infinito ($log(0)$) no caso em que $\hat{v}(t) = 1$. Situação similar ocorre quando $v(t) = 1$ e $\hat{v}(t) = 0$. Sabendo que esta função objetivo determina o erro entre a classe predita e a classe observada, pode-se minimizá-la utilizando métodos de PNL como o do gradiente. Dessa forma, construímos uma superfície de separação entre as classes, discriminada por uma curva sigmóide, como dado pela equação (3.3).

## 3.2 Classificação Bayesiana

O teorema de Bayes relaciona o conhecimento prévio do problema na forma da seguinte equação:

$$P(C_i|\mathbf{x}) = \frac{p(\mathbf{x}|C_i)P(C_i)}{p(\mathbf{x})} \tag{3.4}$$

Onde $P(C_i|\mathbf{x})$ é a probabilidade de observar a classe $C_i$ dado que os valores das variáveis $\mathbf{x}$ são conhecidos, chamada **probabilidade *a posteriori***; $p(\mathbf{x}|C_i)$ representa a distribuição de probabilidades das variáveis $\mathbf{x}$ quando a classe observada é $C_i$, chamada de distribuição de **probabilidade condicional**; e $P(C_i)$ é a probabilidade de ocorrência da classe $C_i$ na ausência de qualquer observação, chamada **probabilidade *a priori***. Por fim, o denominador é apenas a probabilidade de observar valores das variáveis $\mathbf{x}$, e funciona como um fator de padronização [3].

O modelo de Classificação Bayesiana faz uso do teorema de Bayes para decidir qual classe tem a maior probabilidade condicional, e portanto será escolhida. Para isso, as probabilidades condicionais são calculadas diretamenta pela definição quando se tratando de atributos nominais, e através da PDF do atributo aplicada em $x$ para variáveis numéricas. Por fim, a probabilidade condicional geral para cada classe será dada pelo produtório das probabilidades condicionais de cada atributo.

## 3.3 Árvores de Decisão

Árvores de Decisão são modelos de aprendizado supervisionado que podem ser utilizados tanto para classificação quanto para regressão. De forma ampla, buscam definir um modelo de predição através do aprendizado de regras inferidas pelos atributos do dataset. São chamadas de árvore pois o modelo é construído como um diagrama de decisão, cuja representação é similar a uma árvore, conforme ilustrado abaixo:

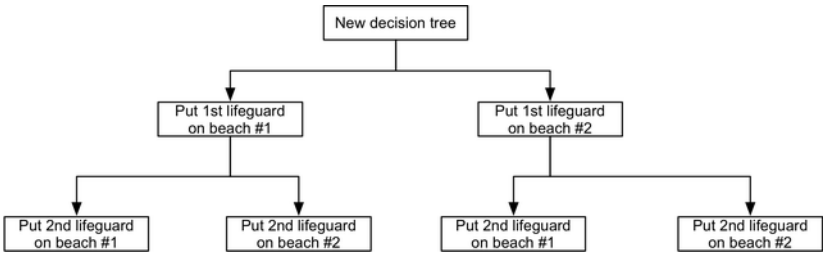


Figure 1: Exemplo de Árvore de Decisão [4]

A predição de uma classe é baseada essencialmente em percorrer o diagrama utilizando os valores dos atributos conhecidos como entrada para as regras inferidas pelo modelo. A acurácia do modelo está, portanto, diretamente correlacionada à inferência das regras e também ao tamanho que a árvore pode adotar, o que permite uma maior especificidade das regras inferidas.

## 3.4 Random Forest

Modelos do tipo Random Forest realizam a agregação de diversas Árvores de Decisão (DT, do inglês *decision tree*), já explicadas na subseção 3.3, para composição de classificadores ou regressores. Para o problema específico de classificação, a saída de uma Random Forest é a classe selecionada pela maioria das DTs. Já para regressões, são retornadas as médias dos valores retornados por cada DT [5].

O objetivo central na construção do modelo é diminuir a variância entre as saídas das DTs e ao mesmo tempo evitar *overfitting* - que ocorre quando o modelo é específico demais para o conjunto de treinamento e apresenta baixa acurácia para conjuntos de teste. Para isso, o número de DTs utilizadas é configurável através de um hiperparâmetro, o que permite realizar um ajuste que propicie a menor variância possível para análise.

## 3.5 Gradient Boosting

O método de Gradient Boosting preconiza a construção de um modelo preditivo forte a partir de modelos intermediários fracos - ideia geral dos algoritmos de Boosting-, normalmente DTs. É similar, portanto, ao de Random Forest no que tange a utilização de DTs

como elementos de construção do modelo preditivo final. Diferem, entretanto, na forma como os diferentes modelos intermediários (DTs) são agregados, já que no Gradient Boosting eles são sequencialmente adicionados, enquanto em Random Forest são utilizados em paralelo.

O algoritmo de Gradient Boosting consiste em realizar o treinamento de um modelo fraco inicial e aplicar à ele próprio uma função de correção a fim de melhorar a acurácia de um novo modelo a ser criado para o mesmo conjunto de dados. Nesse caso, como o nome do método sugere, a função de correção é derivada do gradiente da função em cada etapa da interação, conforme a seguinte equação:

$$F_{m+1}(x) = F_m(x) + h_m(x) \tag{3.5}$$

Onde $h_m(x)$ é a função de correção, que será o gradiente da função de avaliação utilizada para o problema. Além disso, o método possui 3 hiperparâmetros: tamanho da árvore, taxa de aprendizado e subamostra. Os dois últimos servem, respectivamente, para controlar a proporção em que a função de correção é adicionada e o particionamento da amostra original durante a construção do modelo.

## 3.6 SVM

O método de Support Vector Machine (SVM) consiste na utilização das chamadas funções de núcleo para manipular o espaço de características de um dado problema de forma indireta a fim de obter uma medida de similaridade entre diferentes entradas do conjunto de dados. A partir da aplicação da função de núcleo, é construída então uma matriz de núcleo, que contém uma representação do conjunto inicial com suas respectivas similaridades em relação a todos os outros registros. Esse matriz é, posteriormente, utilizada para construção de um hiperplano que separa as diferentes classes alvo.

A fim de minimizar o erro de classificação produzido pelo método, o conceito de margem de separação é adotado no SVM, e consiste em definir um limiar definido a partir do hiperplano para o qual o modelo é capaz de realizar classificações corretas. Com isso, ao maximizarmos a margem de separação, chegamos ao modelo com a maior capacidade de classficar corretamente os registros.

O modelo disponibiliza 2 hiperparâmetros, C e $\gamma$. O primeiro, C, é um parâmetro de regularização da SVM, e define a margem utilizada pelo modelo para determinar o quanto de erro é aceitável. Isso permite controlar a troca entre a fronteira de decisão e o termo de classificação errônea. Quando C estiver alto, o modelo classificará um número maior de pontos, o que também pode ocasionar perda de acurácia e overfitting. Já o hiperparâmetro $\gamma$ define o quanto a distância entre os pontos influencia o cálculo dos hiperplanos. Quanto maior for Gamma, mais influência possuem pontos próximos e maior a probabilidade de overfitting. Por outro lado, quanto menor for Gamma, maior a influência de pontos mais distantes e mais generalista o modelo.

## 3.7 Redes Neurais

O método de Redes Neurais se inspira em uma das mais complexas máquinas existentes no mundo real, o cérebro humano. Ao pensarmos no cérebro humano, pensamos em como os neurônios são responsáveis pela recepção e transmissão de informações. Em redes neurais, o neurônio artificial, conhecido como Perceptron, é responsável pela recepção

de diversas entradas, tais são multiplicadas por diferentes pesos e esse valores somados. A partir da soma ponderada, tal valor é posto sob uma função de ativação para gerar uma saída. Basicamente, essas funções de ativação definem se o neurônio dever ser ativado ou não. Alguns exemplos de funcão de ativação são a função linear, função retificadora linear(relu), função tangente hiperbólica (tanH), sigmóide, etc. As redes neurais possuem camadas e essas são conectadas de forma que as suas conexões formam uma rede que realizam o processamento da saída. Para problemas mais complexos e não linearmente separáveis, é possível utilizar de múltiplas camadas chamada de perceptron multicamadas, Multilayer Perceptron (MLP).

No treinamento, os pesos gerados de forma aleatória são ajustados de forma a tornar o modelo um bom aproximador da função que gera a saída. A divergência entre a saída do modelo e o valor esperado é indicada pela função de avaliação que vai ser minimizada. Contudo, ao tratarmos de rede neurais lidamos como uma quantidade grande parâmetros que precisam ser ajustados o que ocasiona numa maior complexidade e custo computacional. A partir do método gradiente, em cada iteração são calculados os pesos que possuem uma direção de redução da função avaliadora que vai ser otimizada. O ajuste feito para os pesos em cada iteração são propagados para as camadas anteriores, de forma que em toda iteração teremos o reajuste dos pesos anteriores. Essa caracteristica do algoritmo de backpropagation possbilita um ajuste de uma grande quantidade de parametros de forma menos custosa e obtendo um erro mínimo.

A taxa de aprendizado possui uma forte influêcia no comportamento do treinamento de uma rede neural e tal indica o tamanho do passo no ajuste dos parãmetros de cada iteração presente no método do gradiente. Caso esse parâmetro seja muito baixo, o aprendizado da rede vai ser lento; enquanto se esse parametro for muito alto, podemos ter a presença de oscilações no treinamento que impossibilitam a convergência no aprendizado. Existem otimizadores que usam de uma taxa de aprendizado adaptativa, de forma que a convergência seja garantida sem a necessidade de tempos muito grandes. Um exemplo desse tipo de otimizador é o Adam.

Não existe uma bala de prata quando tratamos da melhor topologia de rede para os diversos problemas, logo cada problema é um problma e a topologia mais adequada deve ser obtida através de meios empíricos.

# 4 Resultados

## 4.1 Visualização e Caracterização dos dados

Conforme apresentado na matriz de variáveis nulas (figura 2), algumas colunas como *agent* e *company* possuem entradas com valores nulos, cuja manutenção afetaria as análises efetuadas. Como a variável que define a empresa associada à reserva é nula em 94% das entradas, optou-se por removê-la. Aproveitou-se também para remover variáveis intuitivamente irrelevantes para o modelo, como nome, e-mail e telefone. Após isso, foi possível remover todas as entradas contendo valores nulos, já que representavam uma pequena parte do dataset utilizado ($\approx$ 10%).

A figura 3 apresenta a matriz de correlação dos atributos, pela qual observa-se a ausência de fortes correlações, exceto entre as variáveis *arrival_date_week_number* e *arrival_date_year*.

Como uma etapa de engenharia de atributos, optou-se por tratar algumas informações

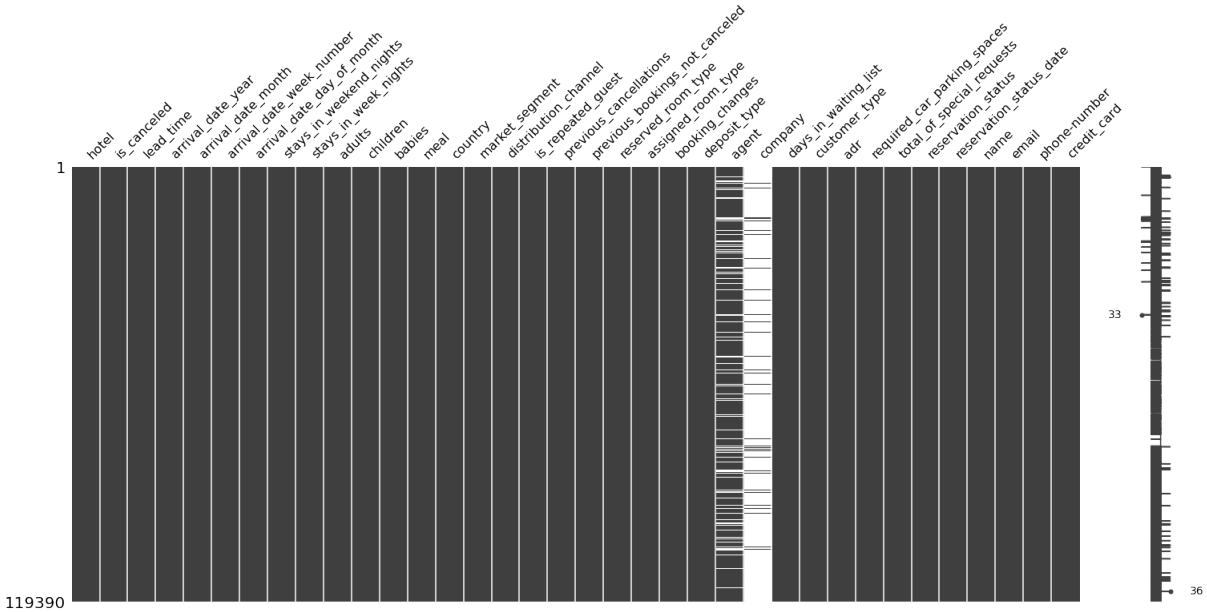


Figure 2: Matriz de variáveis nulas.

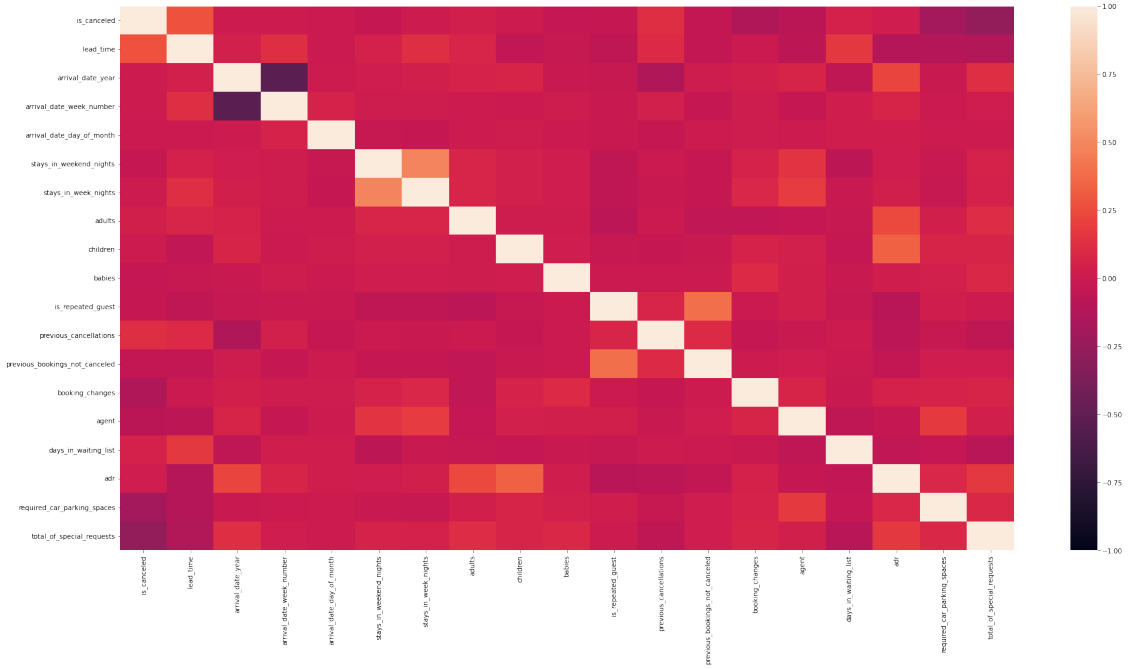


Figure 3: Matriz de correlação de atributos.

pouco relevantes para o modelo em sua forma original. Para isso, um novo atributo de localização (*guest_location*) foi criado para substituir o atributo de país, mapeando Portugal em "Local" e outros países em "Internacional", já que o número de reservas provenientes de Portugal era inicialmente muito maior que as de outros países individualmente. Além disso, as variáveis *children* e *babies* foram somadas em uma nova variável *kids*. Por final, dois novos atributos *total_guests* e *total_stays* foram criados a fim de representar, respectivamente, as somas de hóspedes (adultos, crianças e bebês) e diárias (diárias em finais de semana e dias de semana).

Após as modificações citadas acima, realizou-se o procedimento de conversão de variáveis categóricas para variáveis indicadoras. Por exemplo, a variável *meal* foi dividida em 4 novas variáveis: *meal_FB*, *meal_HB*, *meal_SC* e *meal_Undefined*, que representam as diferentes possibilidades de refeições a serem escolhidas pelos clientes.

Logo após a etapa de tratamento das variáveis, como tarefa de pré-processamento e com o objetivo de ajustar as escalas das diferentes variáveis, padronizou-se o dataset. Para isso, foi utilizada a funcionalidade de `StandardScaler` do *scitkit learn* [6]. Sendo assim, prosseguiu-se para a spearação dos subconjuntos de treinamento e testes e posterior treinamento dos modelos, cujos resultados são apresentados nas subseções a seguir.

Antes de apresentar os resultados dos treinamentos propriamente ditos, é importante ressaltar o procedimento de validação adotado. Essencialmente, foi utilizada a técnica de validação cruzada, que consiste na divisão do conjunto de treinamento em $n$ subconjuntos, os quais são utilizados para treinar os modelos $n$ vezes, validando o modelo para diferentes entradas e gerando estatísticas que permitem avaliar seu comportamento de forma mais aprofundada. Tal técnica é especialmente relevante para os modelos que disponibilizam hiperparâmetros, pois permite avaliar diferentes conjuntos de hiperparâmetros e decidir o que propicia o melhor modelo. Para o caso de modelos com hiperparâmetros, utilizou-se a função de `GridSearchCV` do *scikit learn* [7], que possibilita a definição de uma matriz de hiperparâmetros a serem estudados. Para todas as validações cruzadas foi utilizado o mesmo split e a mesma semente de randomização.

## 4.2 Métricas de comparação

Diversas métricas foram calculadas para comparar os modelos, as quais estão descritas abaixo, onde $VP$, $VN$, $FP$ e $FN$ representam, respectivamente, os verdadeiros positivos, verdadeiros negativos, falso positivos e falso negativos.

- **Acurácia**: dentre todas as classificações, quantas o modelo classificou corretamente. É calculada pela seguinte equação:

$$A = \frac{VP + FN}{VP + VN + FP + FN} \tag{4.1}$$

- **Precisão**: dentre todas as classificações positivas que o modelo fez, quantas são de fato positivas. É calculada pela seguinte equação:

$$P = \frac{VP}{VP + FP} \tag{4.2}$$

- **Revocação**: dentre todos os registros que são de fato positivos, quantos o modelo classificou como positivo. Dada pela seguinte equação:

$$R = \frac{VP}{VP + FN} \tag{4.3}$$

- **F1-score**: média harmônica entre precisão e revocação. Dado pela seguinte equação:

$$F1 = \frac{2 \cdot P \cdot R}{P + R} \tag{4.4}$$

Dentre as estatísticas apresentadas acima, adotou-se o *F1-score* como métrica de comparação entre os diferentes modelos. Isso, pois ele indica a capacidade do modelo apresentar uma boa sensibilidade sem prejudicar a precisão e vice-versa. À princípio, havia-se adotado a acurácia como métrica de comparação, mas acabou-se rejeitando-a por conta do pequeno desbalanceamento apresentado pelo dataset, o que faz com quê a acurácia seja tendenciosa para a classe dominante.

## 4.3 Modelos lineares

| | **Regressão Logística** | **Classificação Bayesiana** |
|---|---|---|
| **Precisão** | 0.794 ± 0.005 | 0.700 ± 0.005 |
| **F1-score** | 0.788 ± 0.005 | 0.484 ± 0.005 |
| **Acurácia** | 0.793 ± 0.005 | 0.526 ± 0.005 |
| **Recall** | 0.793 ± 0.005 | 0.526 ± 0.005 |

Table 2: Resultados para modelos lineares.

A tabela 2 apresenta os resultados para os modelos lineares de Regressão Logística e Classificação Bayesiana. Como observa-se, o segundo teve uma performance muito baixa, com *F1-score* de apenas 0.484. Isso é esperado dada a característica da Classificação Bayesiana de considerar apenas as probabilidades condicionais individuais de cada atributo e a variável alvo, ignorando portanto as correlações entre os atributos. A regressão logística também apresentou performance elevada, com *F1-score* de 0.788, indicando que a superfície de decisão do problema não é bem aproximada por um sistema linear.

Como os modelos lineares estudados não disponibilizam hiperparâmetros, não há mais discussões a serem feitas sobre eles.

## 4.4 Modelos não lineares

| | **Árvore de decisão** | **Random Forest** | **Gradient Boosting** |
|---|---|---|---|
| **Precisão** | 0.779 ± 0.009 | 0.993 ± 0.005 | 0.765 ± 0.005 |
| **F1-score** | 0.764 ± 0.013 | 0.995 ± 0.005 | 0.802 ± 0.005 |
| **Acurácia** | 0.772 ± 0.008 | 0.996 ± 0.005 | 0.853 ± 0.005 |
| **Recall** | 0.772 ± 0.008 | 0.996 ± 0.005 | 0.844 ± 0.005 |

| | **SVM** | **Redes Neurais** |
|---|---|---|
| **Precisão** | 0.824 ± 0.005 | 0.843 ± 0.005 |
| **F1-score** | 0.850 ± 0.005 | 0.790 ± 0.005 |
| **Acurácia** | 0.886 ± 0.005 | 0.850 ± 0.005 |
| **Recall** | 0.877 ± 0.005 | 0.755 ± 0.005 |

Table 3: Resultados para modelos não lineares.

A tabela 3 apresenta os resultados para os modelos não lineares estudados. Observa-se que a performance demonstrada é consideravelmente superior que as apresentadas pelos modelos lineares, confirmando a suspeita do final da subseção anterior de que o problema é melhor aproximado por uma superfície de decisão não linear. O único modelo cujo treinamento

não envolveu estudo de hiperparâmetros foi o de Árvore de Decisão. Para os outros, os resultados e estudo de hiperparâmetros será feito a seguir.

Para o treinamento da Random Forest, variou-se o número de estimadores utilizados para sua construção, que é basicamente o número de árvores de decisão combinadas. Os dois valores utilizados foram de 10 e 100 estimadores, que apresentaram respectivamente, *F1-score* médio na validação cruzada de $0.871 \pm 0.003$ e $0.881 \pm 0.002$. Como era esperado, o aumento do número de estimadores aumenta a performance do modelo, mas prejudica o tempo de treinamento. Com isso, o modelo utilizado para a comparação foi o com 100 estimadores.

Já para o Gradient Boosting, o estudo dos hiperparâmetros envolveu uma maior variedade, que são apresentados na tabela 4, disponível abaixo.

| **Número de estimadores** | **Taxa de aprendizado** | **Subamostras** | **F1-score médio** |
|---|---|---|---|
| 10 | 0.1 | 0.5 | $0.713 \pm 0.008$ |
| 10 | 0.1 | 1.0 | $0.713 \pm 0.011$ |
| 10 | 1 | 0.5 | $0.828 \pm 0.003$ |
| 10 | 1 | 1.0 | $0.830 \pm 0.003$ |
| 100 | 0.1 | 0.5 | $0.840 \pm 0.003$ |
| 100 | 0.1 | 1.0 | $0.839 \pm 0.002$ |
| 100 | 1 | 0.5 | $0.850 \pm 0.004$ |
| 100 | 1 | 1.0 | $0.857 \pm 0.005$ |

Table 4: Validação cruzada de hiperparâmetros para Gradient Boosting.

Observa-se, portanto, que o melhor conjunto de hiperparâmetros é dado por número de estimadores = 100, taxa de aprendizado = 1 e Subamostras = 1. Isso faz sentido, pois um maior número de estimadores tende a aumentar a acurácia do modelo, e a taxa de aprendizado funciona como uma penalidade para cada árvore de decisão que compõe o modelo.

Para o modelo de SVM, não foi possível realizar um estudo muito aprofundado dos hiperparâmetros, pois o tempo de treinamento de cada modelo foi muito elevado para o conjunto de treinamento utilizado, que é consideravelmente grande. Portanto, variou-se apenas o hiperparâmetro C, para o qual adotou-se o valor de 1 e 100, obtendo-se respectivamente *F1-score* médio de $0.841 \pm 0.001$ e $0.850 \pm 0.002$. Logo, o aumento de C levou a uma melhor acurácia, o que pode se justificar pelo fato de que este hiperparâmetro funciona como regularizador, definindo a margem utilizada pelo modelo para determinar o quanto de erro é aceitável. Logo, quanto maior C, mais pontos são classificados. Importante se atentar que um aumento indiscriminado de C pode levar a uma eventual perda de acurácia com ocorrência de overfitting.

Por fim, para o modelo de Redes Neurais foram testadas diferentes topologias, cujas descrições e resultados de *F1-score* médio são apresentados na tabela 5. Para as 3 tropologias adotadas, utilizou-se otimizador *Adam*, função de custo de Entropia Cruzada Binária e regularizadores do tipo L2 a fim de evitar overfitting e consequente perda de acurácia.

Nesse caso, concluiu-se que a topologia onde foi utilizada largura de 133 e 2 camadas escondidas foi a que apresentou melhor performance. A conclusão a que chegamos foi de que o aumento do número de camadas internas, bem como da largura da rede neural foi inicialmente benéfico, mas ao seguir com o aumento ocasiona-se a ocorrência de overfitting e perda de acurácia devido à construção de caminhos complexos pouco eficientes.

| Número de estimadores | Taxa de aprendizado |
|---|---|
| Largura 12, 1 camada escondida | 0.774 ± 0.003 |
| Largura 133, 2 camadas escondidas | 0.790 ± 0.003 |
| Largura 133, 5 camadas escondidas | 0.784 ± 0.002 |

Table 5: Validação cruzada de hiperparâmetros para Rede Neural.

# 5 Conclusões

| Modelo | Melhor *F1-score* |
|---|---|
| Random Forest | 0.995 ± 0.005 |
| SVM | 0.850 ± 0.005 |
| Gradient Boosting | 0.802 ± 0.005 |
| Redes Neurais | 0.790 ± 0.005 |
| Regressão Logística | 0.788 ± 0.005 |
| Árvore de Decisão | 0.764 ± 0.013 |
| Classificação Bayesiana | 0.484 ± 0.005 |

Table 6: Resumo de resultados dos diferentes modelos treinados.

A partir da tabela 6, conclui-se que o modelo que apresentou a melhor performance (avaliada através da métrica de *F1-score*) foi o de Random Forest. Além disso, os 4 melhores modelos treinados foram do tipo não-linear, o que leva à conclusão de que a superfície de decisão do problema em questão é melhor ajustada por um sistema não-linear.

# References


[1] https://www.kaggle.com/mojtaba142/hotel-booking. Acessado em 20/02/2022.

[2] https://colab.research.google.com/.

[3] Evsukoff, A. Inteligência Computacional, Fundamentos e Aplicações, 2017.

[4] M. Wagner, H. Principles of Operations Research, 1975.

[5] https://en.wikipedia.org/wiki/Random_forest.

[6] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

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