# 

INSTITUIÇÃO DE ENSINO SUPERIOR CENTRO UNIVERSITÁRIO RITTER DOS REIS

**Predição de Diabetes com**

**Aprendizado de Máquina**

**SUMÁRIO**

[**Introdução 3**](#_7whmf681hday)

[Objetivo 3](#_qb8bbpjx2cbb)

[Alunos participantes 3](#_4rfkqr9mde8s)

[**Conjuntos de dados 3**](#_vtmhzej041)

[Estatísticas e visualizações importantes 3](#_phl3ywj17g8r)

[**Dependências de código 5**](#_g35x74nj4hc4)

[Packages necessárias 5](#_rgcjqmfw51wi)

[**Experimento - Árvore de decisão 5**](#_xorycole0tt6)

[Random Forest 5](#_mh04sfr0kbgd)

[Extra tree classifier 6](#_mg2pu1nu13up)

[Decision tree classifier 8](#_cdbrtehnixqt)

[XGBoost 10](#_ccmj0e9l05h7)

[Decision tree Regressor 11](#_g24gj2umys6s)

[**Experimento - Redes Neurais 12**](#_19hk13l9ysm6)

[FFN 12](#_ggpw92ba4qu)

[MLP 15](#_8yz95a60cad0)

[Perceptron 16](#_5s3lvdobetc7)

[RBN 17](#_mcy398e4h5cp)

[DFF 18](#_sqm2vsklbke5)

[**Análise comparativa entre modelos 20**](#_o6eucfioh2we)

[Modelos com melhor desempenho 20](#_xsslt29rsh3o)

[Modelos com pior desempenho 20](#_cifbtm4njc0w)

[**Recomendação de modelo 21**](#_nj64sbycgq9o)

[Rede Neural - Modelo FFN 21](#_bgg7ms3ceecp)

## 

## Introdução

### Objetivo

Doenças como a diabetes exigem um diagnóstico precoce e tratamento adequado, visto ser um grande problema (mas comum) em nossas vidas. Portanto, visando solucionar/aprimorar o diagnóstico da diabetes, propomos treinamentos de modelos de Inteligência Artificial. Através de dados coletados de pacientes diabéticos e não diabéticos, como gravidez, espessura da pele, níveis de glicose e insulina no sangue e idade, estes dados serão usados por modelos que fornecerão uma ferramenta poderosa para auxiliar na triagem e diagnóstico precoce da diabetes.

### Alunos participantes

Nosso grupo é composto por três participantes:

* Gabriel Schumann Moreira - RA 1292225971 ;
* Nicolas Marcolin Heberle - RA 1292226066 ;
* Vitor Silva de Antoni - RA 1292219829 ;

## Conjuntos de dados

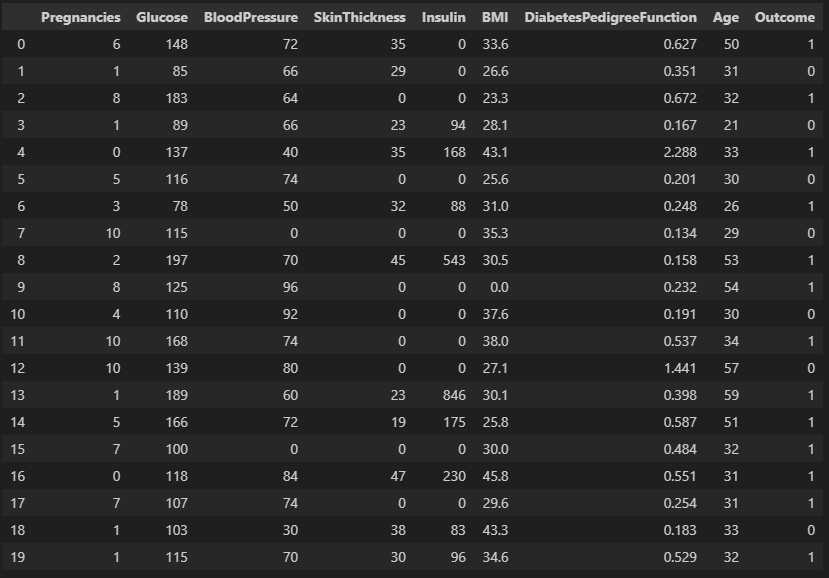
### Estatísticas e visualizações importantes

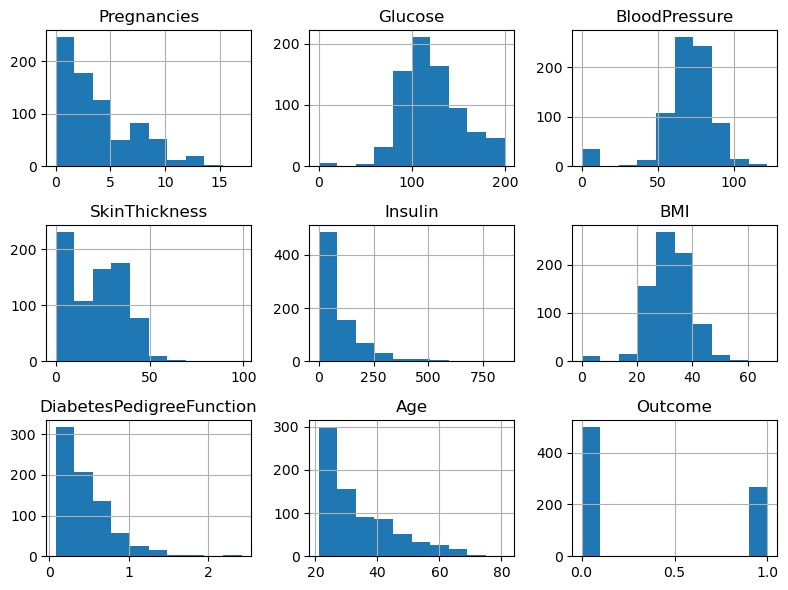
Nosso conjunto de dados reúne informações de características humanas que corroboram a indicativa de possibilidade de um indivíduo possuir diabetes . Todavia, abaixo segue duas imagens, sendo elas a **primeira** referente a uma parte do nosso conjunto de dados cujo apresenta algumas estatísticas de características humanas com base em dados coletados e disponibilizados através do ***Kaggle***. A **segunda imagem**, todavia, diz respeito à visualizações importantes que abrange diversos gráficos com base em cada característica abordada no nosso conjunto.

**Observação ¹:** Para baixar o dataset, acesse o link [clicando aqui](https://drive.google.com/file/d/1XnYjOIpNgF28FnJmI3zlNuErvjqxFQtM/view?usp=drive_link).

**Observação ²:** Na segunda imagem, a linha vertical refere-se ao número de pessoas (0 - 200+) que possuem aquelas características, enquanto a linha horizontal refere-se a estatística de cada característica.

**Observação ³:** Por padronização, mantemos os nomes de cada característica em **inglês**, mas em tradução direta, seriam elas (respectivamente de acordo com a tabela abaixo): Gravidez, glicose, pressão no sangue, espessura da pele, insula, IMC, função de pedigree de diabetes, idade e resultado (se tem diabete ou não).





## Dependências de código

### Packages necessárias

Para o desenvolvimento do nosso código, foram utilizados os seguintes pacotes abaixo, sendo necessário baixá-los em suas respectivas versões para correto funcionamento do programa.

*python 3.9.18*

*numpy: 1.26.4*

*pandas: 2.2.1*

*matplotlib.pyplot: 3.8.4*

*seaborn: 0.12.2*

*sklearn: 1.3.0*

*imblearn: 0.8.0*

*xgboost: 2.0.3*

*tensorflow: 2.10.0*

*keras: 2.10.0*

*pickle: 4.0*

## Experimento - Árvore de decisão

### Random Forest

**Parâmetros**

Os parâmetros utilizados foram *max\_depth=2* e *min\_samples\_leaf=40*, *min\_samples\_split*=30 e *random\_state*=42.

**Justificativa**

É um algoritmo de aprendizado de máquina que combina múltiplas árvores de decisão para melhorar a precisão e a robustez das previsões. No contexto do dataset ele proporciona um equilíbrio entre precisão, interpretabilidade e robustez .

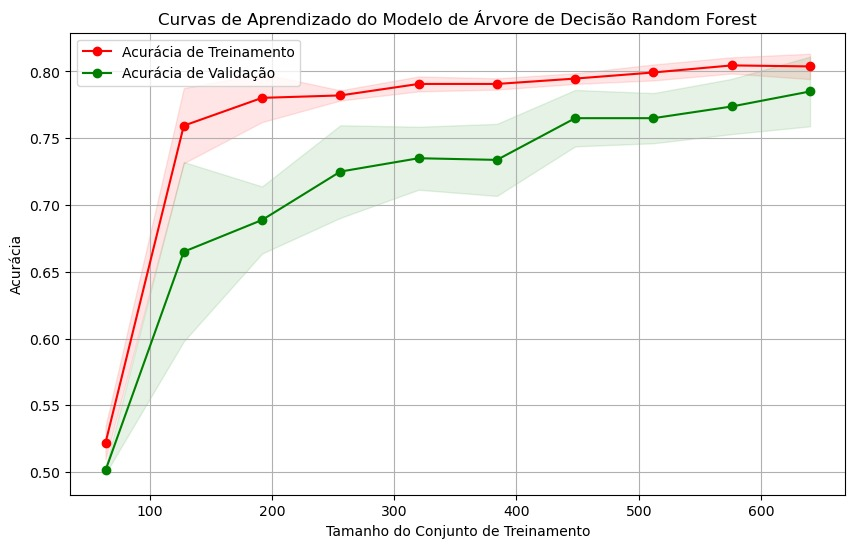
**Estratégia**

A estratégia foi usar o GridSearchCV para encontrar os melhores parâmetros e assim chegar no melhor modelo, foi notado que ainda assim tinha um certo nível de overfitting, para avaliarmos se tinha ou não overfitting usamos um gráfico de curva de aprendizado para verificar a acurácia de treinamento e de validação conforme o aumento dos dados, por isso foi necessário fazer diversos testes alterando os valores do min\_samples\_leaf e min\_samples\_split ou seja o min\_samples\_split é a quantidade de amostras necessárias para dividir um nó interno e o min\_samples\_leaf é o número mínimo de amostras necessárias para estar em um nó folha, colocamos os valores de min\_samples\_leaf 40 e min\_samples\_split 30, a profundidade máxima (max\_depth) colocamos o valor de 2, assim o modelo se mostrou estável e sem overfitting.

**Modelo Resultante**

Para visualização do modelo resultante, podemos rodar o código Python disponível no [link](https://drive.google.com/file/d/1PQHPgzRghJ9vrrIymfBTV7Lc09yJ_mtJ/view?usp=drive_link).

**Desempenho**



**Código fonte**

O código fonte está disponível através do [link](https://drive.google.com/file/d/1X_Am9OmIhwQbsbURvtaQ5ck-ayjz-Edr/view?usp=drive_link) e o dataset através do [link](https://drive.google.com/file/d/1XnYjOIpNgF28FnJmI3zlNuErvjqxFQtM/view?usp=drive_link). Para executar o código fonte, podemos utilizar o Google Colab.

### Extra tree classifier

**Parâmetros**

Os parâmetros utilizados foram *max\_depth*=6*, min\_samples\_leaf*=52*, min\_samples\_split*=80*, random\_state=42.*

**Justificativa**

Semelhante ao Random Forest, também é um método de árvores de decisão que combina múltiplas árvores, mas ele utiliza um método de divisão mais aleatório. Isso pode resultar em uma maior variabilidade, possibilitando capturar diferentes padrões de dados.

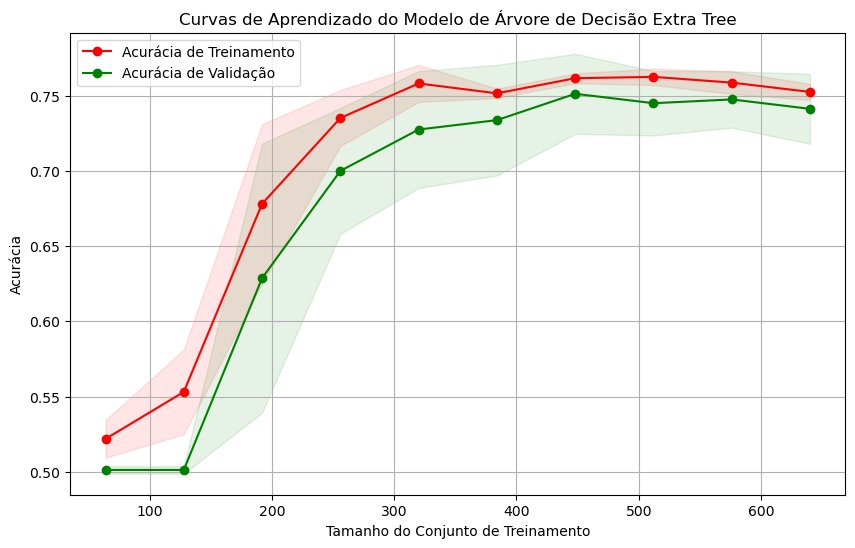
**Estratégia**

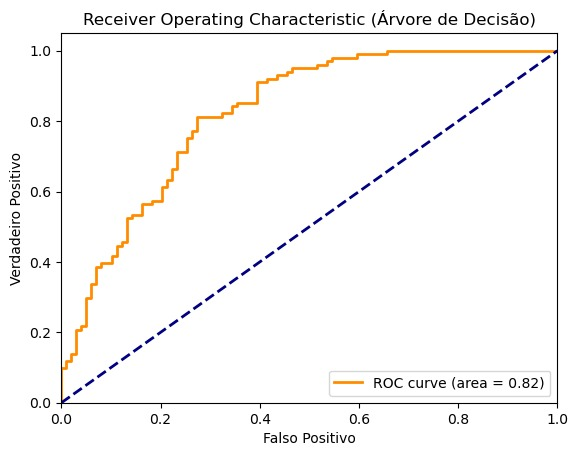
Nesse modelo também utilizamos o GridSearchCV mas tivemos que mudar alguns valores pra diminuir o Overfitting, para avaliar se tinha overfitting utilizamos de novo um gráfico de curva de aprendizado e foi notado que a acurácia do treinamento estava sempre muito acima da acurácia da validação(teste), caracterizando overfitting, os valores ficaram, max\_depth=6, min\_samples\_leaf=52, min\_samples\_split=80, após esses ajustes o modelo se mostrou estável, a curva de acurácia do treinamento ficou extremamente próxima da curva de acurácia de teste e a acurácia apresentou um valor razoável.

**Modelo Resultante**

Para visualização do modelo resultante, podemos rodar o código Python disponível no [link](https://drive.google.com/file/d/1RSqw5J6xO27JqCUvnt_lmrXh9KsTx7pf/view?usp=drive_link).

**Desempenho**





**Código fonte**

O código fonte está disponível através do [link](https://drive.google.com/file/d/1C0tVcM8mDwaalziJ3GvzSBbxlpGISBui/view?usp=drive_link) e o dataset através do [link](https://drive.google.com/file/d/1XnYjOIpNgF28FnJmI3zlNuErvjqxFQtM/view?usp=drive_link). Para executar o código fonte, podemos utilizar o Google Colab.

### Decision tree classifier

**Parâmetros**

Os parâmetros utilizados foram *max\_depth*=4, *min\_samples\_leaf*=40, *min\_samples\_split*=50, *random\_state*=42.

**Justificativa**

Sendo simples de entender e interpretar, este modelo é útil para entender a importância das variáveis e como elas afetam o “Outcome” no contexto do dataset.

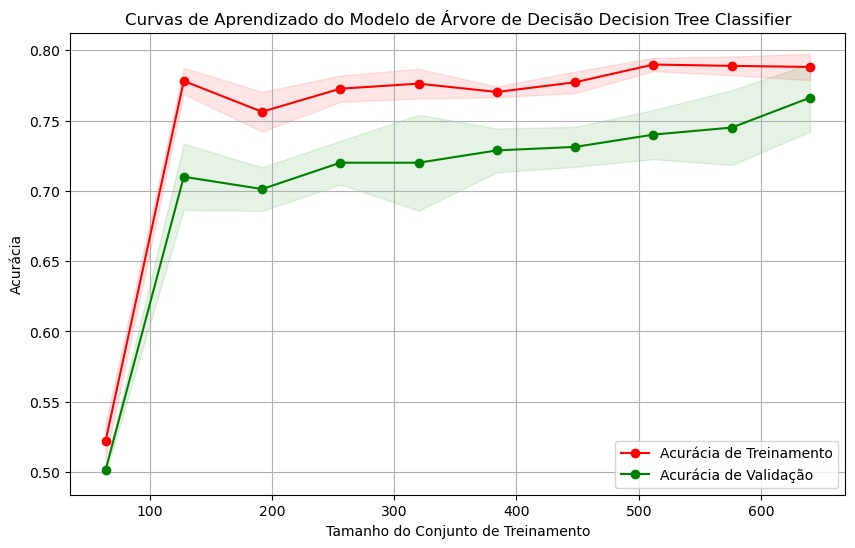
**Estratégia**

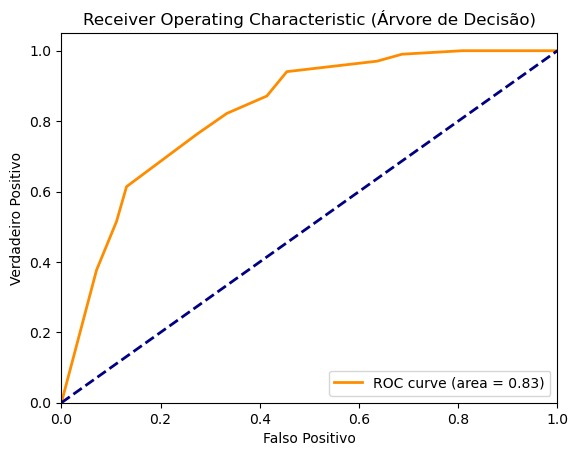
Nesse modelo não utilizamos o GridSearchCV pois nos outros modelos tivemos que fazer ajustes e geralmente rodar o GridSearchCV é algo demorado, então usamos primeiramente o modelo padrão sem alterar os parâmetros e novamente acompanhando a curva de acurácia de treinamento e teste, notamos uma diferença bem alta, então começamos a fazer testes com valores menor do max\_depth e começamos a aumentar os valores de min\_samples\_leaf e min\_samples\_split, os valores que chegaram em um bom modelo foram max\_depth=4, min\_samples\_leaf=40 e min\_samples\_split=50.

**Modelo Resultante**

Para visualização do modelo resultante, podemos rodar o código Python disponível no [link](https://drive.google.com/file/d/12ERXvPmNJtweLdZyTBHTMjJMc7-U-pE8/view?usp=drive_link).

**Desempenho**





**Código fonte**

O código fonte está disponível através do [link](https://drive.google.com/file/d/1RtbMlQKFo2-0M3hunk2ZRoOHPwnu53C2/view?usp=drive_link) e o dataset através do [link](https://drive.google.com/file/d/1XnYjOIpNgF28FnJmI3zlNuErvjqxFQtM/view?usp=drive_link). Para executar o código fonte, podemos utilizar o Google Colab.

### XGBoost

**Parâmetros**

Os parâmetros utilizados foram *max\_depth*: 1, *eta*: 0.1, *objective*: ‘binary:logistic', *eval\_metric*: 'logloss', *alpha*: 0.1, *lambda*: 0.1.

**Justificativa**

Este modelo oferece alta precisão em tarefas de classificação e é capaz de lidar bem com dados muitas features e valores ausentes. Sua capacidade de lidar com dados desbalanceados é a vantagem mais chamativa para.

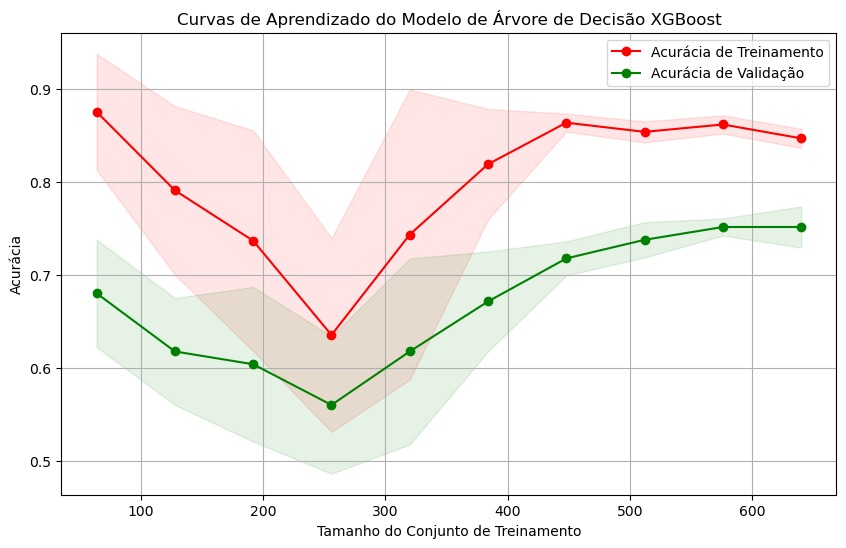
**Estratégia**

Nesse modelo não utilizamos o GridSearchCV, testamos primeiro com os valores padrões do modelo, mostrou um alto nível de overfitting, então fomos testando com valores diferentes, os valores que chegamos para ter um modelo equilibrado e com boa acurácia foram, max\_depth=5, learning\_rate=0.002, n\_estimators=50, objective=’binary:logistic’, eval\_metric=’logloss’, reg\_alpha=0.1 e reg\_lambda=0.1, explicando o que seriam esses parâmetros, max\_depth é a profundidade máxima da árvore, learning\_rate é a taxa de aprendizado, n\_estimators é o número de estágios de boosting a serem executados, objective binary é a classificação binária, o eval\_metric=’logloss’ define a métrica de avaliação, reg\_alpha define a regularização L1 com penalidade de 0.1, e o reg\_lambda define a regularização L2 com penalidade de 0.1, assim o modelo ainda apresentou uma diferença na evolução da acurácia do treinamento para o teste, mas houve uma melhora e os gráficos ficaram próximos.

**Modelo Resultante**

Para visualização do modelo resultante, podemos rodar o código Python disponível no [link](https://drive.google.com/file/d/1YezQheLDKXJud3ZxKz5es_F9StLiclz5/view?usp=drive_link).

**Desempenho**



**Código fonte**

O código fonte está disponível através do [link](https://drive.google.com/file/d/1n9qbVP5AFNninD7VXKji-AZlBzxO8PvT/view?usp=drive_link) e o dataset através do [link](https://drive.google.com/file/d/1XnYjOIpNgF28FnJmI3zlNuErvjqxFQtM/view?usp=drive_link). Para executar o código fonte, podemos utilizar o Google Colab.

### Decision tree Regressor

**Parâmetros**

Os parâmetros utilizados foram *max\_depth*=20, *random\_state*=42.

**Justificativa**

É útil para entender a relação linear ou não-linear entre as variáveis preditoras e o “Outcome”.

**Estratégia**

Nesse modelo também não utilizamos o GridSearch, testamos com uma profundidade máxima (max\_depth) de 20 e fizemos um crossvalidation para avaliar se o modelo tinha algum overfitting representativo, a acurácia média teve um bom valor, então deixamos esse parâmetro.

**Modelo Resultante**

Para visualização do modelo resultante, podemos rodar o código Python disponível no [link](https://drive.google.com/file/d/1Am1O8lkoeNNb_0YuWGSHxuQc_OWMZgWx/view?usp=drive_link).

**Desempenho**





**Código fonte**

O código fonte está disponível através do [link](https://drive.google.com/file/d/1HbOdjiNoWA9Fhok1Nisz8hWsP2v50Ade/view?usp=drive_link) e o dataset através do [link](https://drive.google.com/file/d/1XnYjOIpNgF28FnJmI3zlNuErvjqxFQtM/view?usp=drive_link). Para executar o código fonte, podemos utilizar o Google Colab.

## Experimento - Redes Neurais

### FFN

**Parâmetros**

Os parâmetros utilizados foram:

*Dense(64,activation='relu',name='a1',input\_shape=(X\_trainOver.shape[1],),*

*kernel\_regularizer=regularizers.l2(0.01)),*

*Dense(32,activation='relu',name='a2',kernel\_regularizer=regularizers.l2(0.01)),*

*Dense(1,activation='sigmoid',name='a3',kernel\_regularizer=regularizers.l2(0.01))*

**Justificativa**

Este modelo consiste de camadas densamente conectadas. O que a torna poderosa para capturar padrões não-lineares nos dados e são úteis para tarefas de classificação complexas, como o caso do nosso dataset.

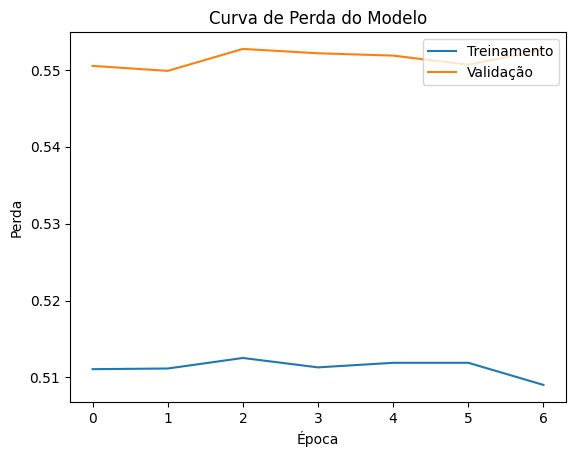
**Estratégia**

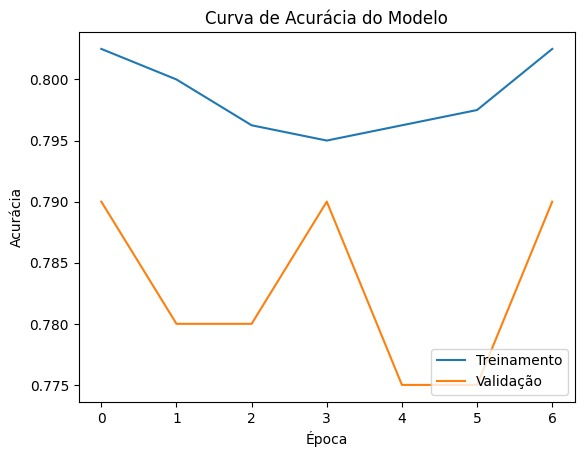
Nesse modelo testamos várias camadas e quantidades de neurônios, chegamos no melhor resultado usando 3 camadas, na primeira camada colocamos 64 neurônios, na segunda 32 neurônios e na última 1 neurônio(saída binária), o modo de ativação usado nas duas primeiras camadas foram relu e na última sigmoid, usamos também um penalizador l2 de 0.01, o otimizador usado foi o Adam com learning\_rate de 0.001, sendo assim o modelo se apresentou estável e sem overfitting, generalizando bem os dados e tendo a melhor acurácia dentre os modelos e a melhor curva ROC AUC, a curva de acurácia e perda acompanharam bem tanto no treinamento quanto no teste.

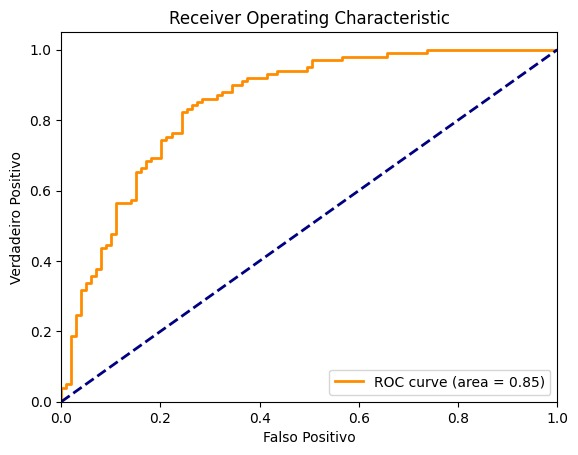
**Modelo Resultante**

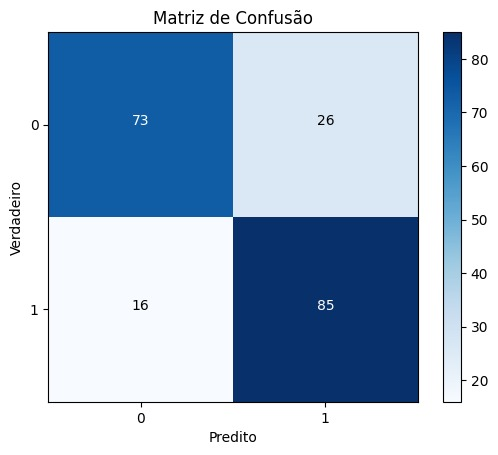
Para visualização do modelo resultante, podemos rodar o código Python disponível no [link](https://drive.google.com/file/d/1iTzp0ouotwZDE59Pq5kt2nFb_X0qZEQk/view?usp=drive_link).

**Desempenho**









**Código fonte**

O código fonte está disponível através do [link](https://drive.google.com/file/d/1lPNoKNqiR8_t0nD0pfJ7QXoda9KcZOdX/view?usp=drive_link) e o dataset através do [link](https://drive.google.com/file/d/1XnYjOIpNgF28FnJmI3zlNuErvjqxFQtM/view?usp=drive_link). Para executar o código fonte, podemos utilizar o Google Colab.

### MLP

**Parâmetros**

Os parâmetros utilizados foram *hidden\_layer\_sizes*=(100, 50), *max\_iter*=500, *alpha*=0.0001, *solver*='adam', *random\_state*=42, *verbose*=False

**Justificativa**

Este modelo sendo similar ao FFN, mas possuindo uma ou mais camadas ocultas, capaz de de aprender representações complexas e abstrações dos dados, o que pode aumentar a precisão do “Outcome” do nosso dataset.

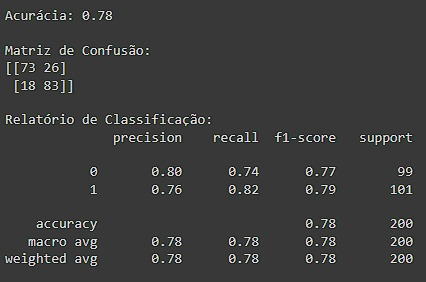
**Estratégia**

Nesse modelo também testamos diversos valores, chegamos no hidden\_layer\_sizes de (50,50), activation=’tanh’, alpha=0.01, learning\_rate=’constant’, e solver=’adam’, o hidden\_layer\_sizes é a configuração das camadas ocultas, com esses parâmetros fizemos um crossvalidation para verificar como estava e deu uma diferença considerável tivemos 0.62 na média do crossvalidation e desvio padrão de acurácia de 0.08, algo não considerado bom mas foi o melhor que chegou nesse modelo.

**Modelo Resultante**

Para visualização do modelo resultante, podemos rodar o código Python disponível no [link](https://drive.google.com/file/d/15aIbq-9BNcbkg4mmPUSuIpeVB8bA08-u/view?usp=drive_link).

**Desempenho**





**Código fonte**

O código fonte está disponível através do [link](https://drive.google.com/file/d/1eHwxW_lxHypwLeuo_31qVlM4zTdTbnZf/view?usp=drive_link) e o dataset através do [link](https://drive.google.com/file/d/1XnYjOIpNgF28FnJmI3zlNuErvjqxFQtM/view?usp=drive_link). Para executar o código fonte, podemos utilizar o Google Colab.

### Perceptron

**Parâmetros**

Os parâmetros utilizados foram *max\_iter*=100, *eta0*=0.1, *random\_state*=42

**Justificativa**

É o modelo mais básico, e assim pode servir como um bom ponto de partida para entender como os outros modelos podem tratar o dataset.

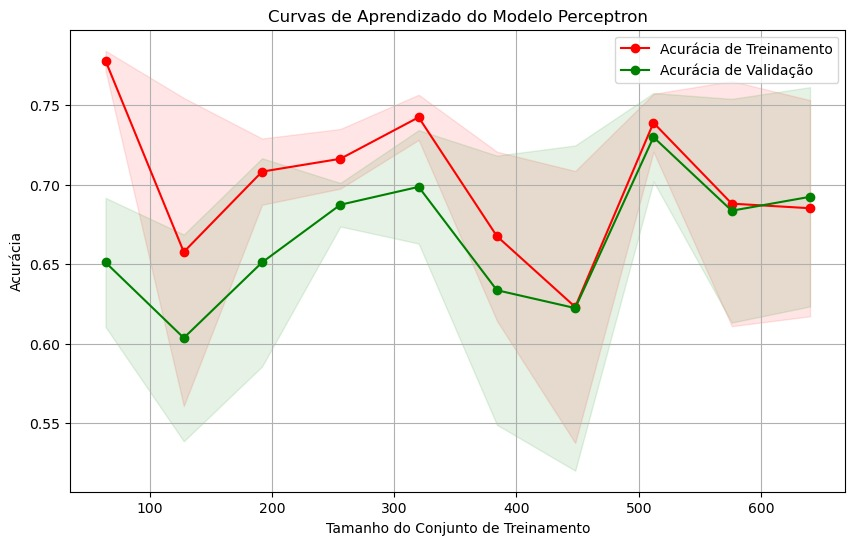
**Estratégia**

Nesse modelo testamos alguns valores e chegamos no mais próximo do ideal usando os parâmetros de max\_iter=10 e eta0=0.1(taxa de aprendizado), tivemos um acurácia próximo de 0.70 e a curva de aprendizado de treinamento e validação se acompanharam bem, mostrando estabilidade, porém o modelo apresentou uma taxa de erro maior no outcome 0.

**Modelo Resultante**

Para visualização do modelo resultante, podemos rodar o código Python disponível no [link](https://drive.google.com/file/d/1pWH9eJ5s4TjfccUFu20ZYnHZ1aWhPMBs/view?usp=drive_link).

**Desempenho**

****

**Código fonte**

O código fonte está disponível através do [link](https://drive.google.com/file/d/1XwH-G2jps6y9tROOPUGy_tjQ5933UxNc/view?usp=drive_link) e o dataset através do [link](https://drive.google.com/file/d/1XnYjOIpNgF28FnJmI3zlNuErvjqxFQtM/view?usp=drive_link). Para executar o código fonte, podemos utilizar o Google Colab.

### RBN

**Parâmetros**

Os parâmetros utilizados foram

*StandardScaler*(),

*KMeans*(*n\_clusters*=3, *random\_state*=42),

*MLPClassifier*(*hidden\_layer\_sizes*=(10,), *activation*='logistic', *max\_iter*=1000, *random\_state*=42)

**Justificativa**

Esse modelo é capaz de modelar relações não-lineares complexas, flexibilidade para lidar com o dataset em específico.

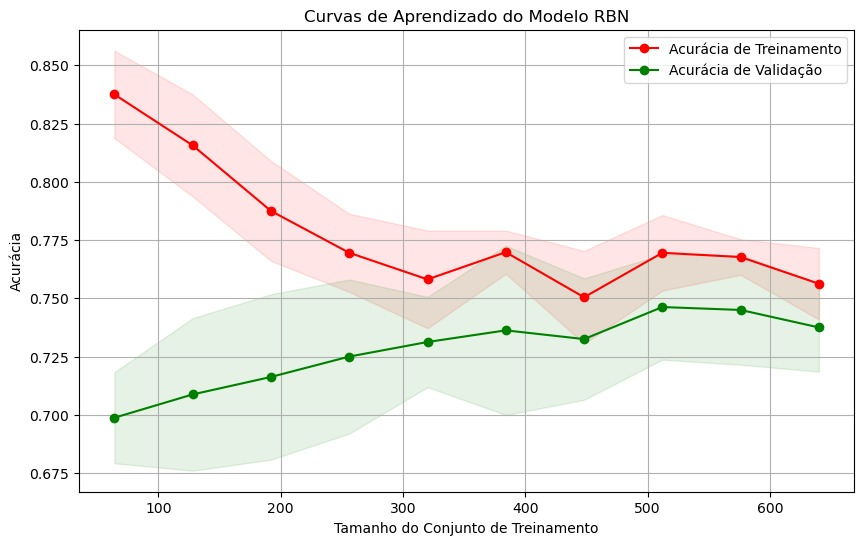
**Estratégia**

Nesse modelo fizemos um pipeline com scala, kmeans e MLPClassifier, no KMeans usamos 3 clusters e no MLPClassificer usamos hidden\_layer\_sizes=(100,50,40), activation=’tanh’, max\_iter=200, verbose=1, alpha=0.001 e solver=’adam’, tivemos uma acurácia de 0.71 e a curva de aprendizado no inicio dos dados aparenta não estar generalizando bem pois a acurácia do treinamento segue maior, mas após 300 dados, as curvas começam a ficarem bem próximas.

**Modelo Resultante**

Para visualização do modelo resultante, podemos rodar o código Python disponível no [link](https://drive.google.com/file/d/1bZ1_99O8VT_f9KQdMWM50DWAH7RH3gZx/view?usp=drive_link).

**Desempenho**



**Código fonte**

O código fonte está disponível através do [link](https://drive.google.com/file/d/1b3Zq8ShQ62h6LXwQIiPvjdTGWjcCx_NI/view?usp=drive_link) e o dataset através do [link](https://drive.google.com/file/d/1XnYjOIpNgF28FnJmI3zlNuErvjqxFQtM/view?usp=drive_link). Para executar o código fonte, podemos utilizar o Google Colab.

### DFF

**Parâmetros**

Os parâmetros utilizados foram

*Dense*(150,*activation*='relu', *name*='a1', *input\_shape*=(*X\_trainOver.shape*[1],)),

*Dense*(100,*activation*='relu',*name*='a2'),

*Dense*(50,*activation*='relu',*name*='a3'),

*Dense*(25,*activation*='relu',*name*='a4'),

*Dense*(1, *activation*='sigmoid',*name*='a5')

**Justificativa**

Se baseia similar ao MLP, mas com várias camadas ocultas permitindo que o modelo capture e aprenda padrões mais complexos e não-lineares nos dados. O que torna muito útil já que possuímos um dataset robusto o bastante para utilizar o modelo com total eficácia e maximizar a capacidade preditiva.

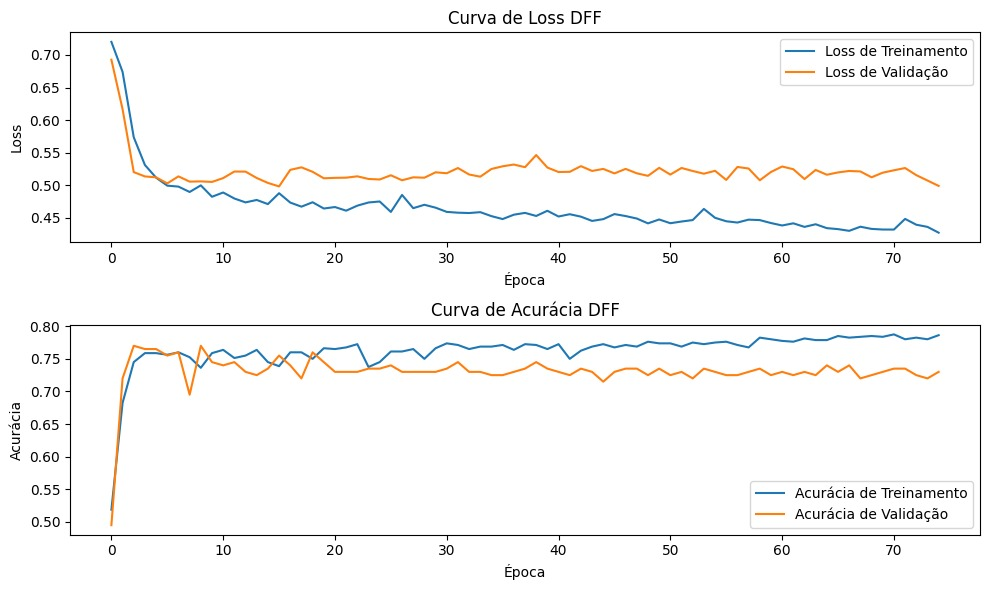
**Estratégia**

Nesse modelo usamos 5 camadas, a primeira com 150 neurônios, a segunda tem 100 neurônios, a terceira 50 neurônios, a quarta 25 neurônios e a última 1 neurônio, todas com ativação sigmoid, otimizador foi adam, a acurácia ficou em 0.74, fizemos dois gráficos para verificar se tinha overfitting, o primeiro era da curva de perda do treinamento se comparado a validação e os dois ficaram com os valores bem próximos, o segundo gráfico é da acurácia de treinamento e validação, foi visto que as duas curvas se mantiveram estáveis e juntas.

**Modelo Resultante**

Para visualização do modelo resultante, podemos rodar o código Python disponível no [link](https://drive.google.com/file/d/1igclx8Ktp-crWdGqGtY10nQKxr7gkgxY/view?usp=drive_link).

**Desempenho**



**Código fonte**

O código fonte está disponível através do [link](https://drive.google.com/file/d/13glRNgZqBJkCU9NTT6VRaFMyg_mjO4zn/view?usp=drive_link) e o dataset através do [link](https://drive.google.com/file/d/1XnYjOIpNgF28FnJmI3zlNuErvjqxFQtM/view?usp=drive_link). Para executar o código fonte, podemos utilizar o Google Colab.

## Análise comparativa entre modelos

### Modelos com melhor desempenho

- Modelo FFN (**Rede Neural**)

- Modelo Extra tree classifier (**Árvore de decisão**)

### Modelos com pior desempenho

- Modelo Perceptron (**Rede Neural**)

- Modelo Decision tree Regressor (**Árvore de decisão**)

## Recomendação de modelo

### Rede Neural - Modelo FFN

Recomendamos o uso do modelo de rede neural FFN, devido apresentar o melhor desempenho, obtendo a maior **acurácia**. As curvas de aprendizado de treinamento e teste demonstraram forte **correlação**, indicando a boa capacidade do modelo de se ajustar aos dados. Além disso, a diferença entre as perdas de treinamento e validação se manteve **constante** ao longo do treinamento, o que reforça a confiabilidade do modelo e sua capacidade de prever com precisão dados não vistos.