Josué Batista Matos Deschamps de Melo
Leandro Guimarães Miranda
Marcos Vinicius Santos Souza
Vagner Ferreira Santos

# Detectando casos de COVID a partir de imagens de Tomografia

São Paulo

2023

## 1 Introdução

Este trabalho tem como objetivo desenvolver mecanismos de detecção de infecção pelo vírus SARS-CoV-2 (vírus causador da COVID-19) através da análise de suas tomografias computadorizadas com o auxílio de Inteligência Artificial. O conjunto de dados utilizado para esse propósito foi coletado de pacientes reais em hospitais localizados no estado de São Paulo e está disponível no Kaggle.

Para isso foram necessários utilizar métodos de Processamento de Imagens e utilização de modelos de Machine Learning, onde o Processamento de Imagens é uma área que envolve técnicas e algoritmos para manipular e analisar imagens. As técnicas utilizadas em Machine Learning possuem o papel de classificação das imagens e predições através do modelo.

No final, é importante ter cuidado ao interpretar os resultados e considerar o impacto ético e social do desenvolvimento e aplicação neste trabalho, pois para a área da saúde estes dados se tornam muito mais sensíveis.

## 2 Solução Proposta

Avaliar e testar diferentes algoritmos de machine learning para encontrar uma solução automatizada e eficaz de diagnóstico de Covid, utilizando tomografias já diagnosticadas. Foram testados os algoritmos de árvore de decisão, floresta randômica, KNN, regressão logística, SVM e rede neural, a fim de analisar as métricas de acurácia, *recall*, precisão e *F1 Score* para escolher o mais eficaz e assertivo. Por se tratar da área da saúde, a taxa de assertividade tem que ser a maior possível, e por isso escolhemos o algoritmo de SVM para classificação do diagnóstico, com resultados muito similares a rede neural, o SVM foi escolhido por ser menos complexo e ter uma performance melhor em tempo de treinamento, apresentando resultados superiores aos demais. Para realizar a classificação, as imagens foram reajustadas para o mesmo tamanho e realizado o *data augmentation*, para geração de novas imagens a partir das existentes. Não foi necessário realizar o balanceamento entre as classes, pois as quantidades entre as duas classes eram quase iguais.

## 3 Análise e tratamento do conjunto dados

O conjunto de dados consiste em um total de 2.482 tomografias computadorizadas, sendo 1.252 tomografias positivas para infecção por SARS-CoV-2 (COVID-19) e 1.230 tomografias de pacientes não infectados por SARS-CoV-2. Esses dados foram coletados de pacientes reais em hospitais localizados em São Paulo, Brasil.

Utilizamos as seguintes funções para analisar as informações contidas no conjunto de dados e prepará-las para os modelos:

### 3.1 Função create_data_frame:

- Descrição: Essa função cria um DataFrame a partir dos dados do conjunto de treinamento.
- Técnica utilizada: Iteração por cada classe (COVID e non-COVID) e cada arquivo no diretório correspondente.
- O que a função faz:
  - Itera sobre as classes de COVID (`covid_class`) e, para cada classe, itera sobre os arquivos no diretório correspondente (`os.listdir(os.path.join(train_dir, sp))`).
  - Para cada arquivo, adiciona uma nova linha à lista `train_data` contendo o nome do arquivo, o ID da classe (0 para COVID e 1 para non-COVID) e a classe em si.
  - Retorna um DataFrame pandas com os dados do treinamento.

### 3.2 Função random_data_frame:

- Descrição: Esta função embaralha aleatoriamente os dados em um DataFrame.
- Técnica utilizada: Amostragem aleatória do DataFrame.
- O que a função faz:
  - Recebe um DataFrame chamado `data`.
  - Aplica a amostragem aleatória no DataFrame usando o método `sample` do pandas, com o parâmetro `frac=1` para manter todos os dados e `random_state=42` para obter resultados consistentes.
  - Reindexa o DataFrame para garantir que os índices sejam sequenciais.
  - Retorna o DataFrame embaralhado.

### 3.3 Função read_image:

- Descrição: Esta função lê uma imagem de um determinado caminho de arquivo.

- Técnica utilizada: Leitura de imagem usando OpenCV.
- O que a função faz:
  - Recebe o caminho do arquivo da imagem como entrada (`filepath`).
  - Usa a biblioteca OpenCV (`cv2`) para ler a imagem do caminho do arquivo.
  - Retorna a imagem lida.

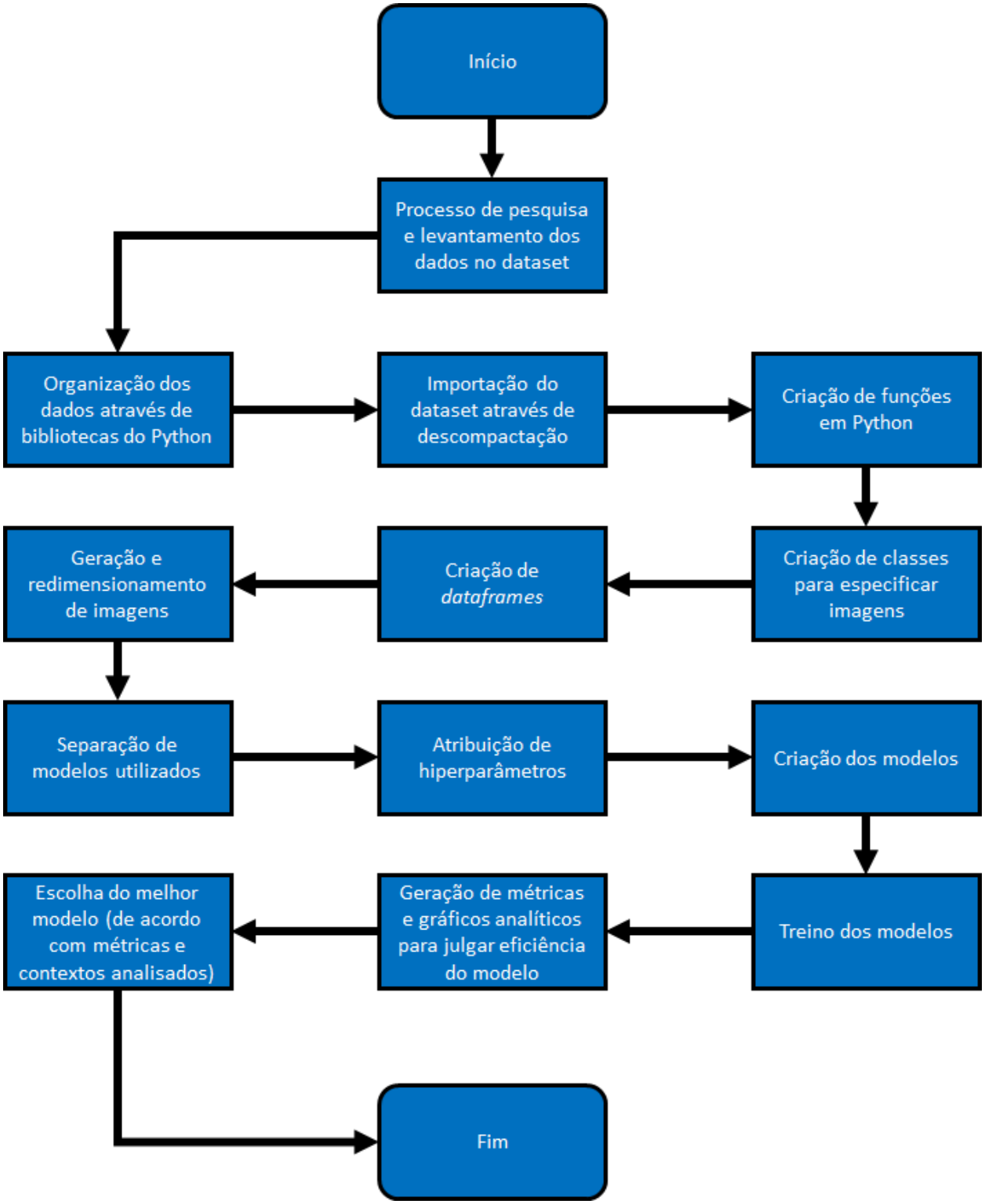
### 3.4 Função resize_image:

- Descrição: Esta função redimensiona uma imagem para um determinado tamanho.
- Técnica utilizada: Redimensionamento de imagem usando OpenCV.
- O que a função faz:
  - Recebe uma imagem (`image`) e o tamanho de destino (`image_size`) como entrada.
  - Cria uma cópia da imagem para evitar a modificação da original.
  - Usa a função `cv2.resize` do OpenCV para redimensionar a imagem para o tamanho de destino, utilizando a técnica de interpolação `cv2.INTER_AREA`.
  - Retorna a imagem redimensionada.

### 3.5 Função load_images:

- Descrição: Esta função carrega as imagens com base nos dados fornecidos.
- Técnica utilizada: Carregamento e normalização de imagens.
- O que a função faz:
  - Recebe um DataFrame chamado `data`.
  - Inicializa um array numpy chamado `X_train` com dimensões baseadas no número de amostras no DataFrame e no tamanho das imagens.
  - Itera sobre as amostras do DataFrame usando `tqdm` para exibir uma barra de progresso.
  - Para cada amostra, lê a imagem usando a função `read_image` e armazena-a em uma variável chamada `image`.
  - Se a imagem não for nula, redimensiona.

## 4. Fluxograma



# 5. Resultados

## 5.1 KNN

Entre diversos testes, foi escolhido 5 como hiper parâmetro de quantidade vizinhos, por apresentar os melhores resultados. Os tratamentos na base de dados, foram os mesmos entre todos os modelos, porém foi retirado o *data augmentation* do KNN, pois os resultados pioraram após a implementação da técnica. Foi necessário realizar uma multiplicação matricial, para transformar a matriz de 4 dimensões em uma matriz de 2 dimensões que pode ser utilizada pelo algoritmo. Com acurácia de 90%, o modelo tende a acertar um pouco mais casos positivos de Covid do que os negativos, mas possuindo um resultado muito próximo entre as duas classes. Para casos positivos de Covid, a precisão foi de 89%, o *recall* de 92% e o *F1 Score* de 90%. Para os casos negativos, o modelo obteve precisão de 92%, *recall* de 88% e *F1 Score* de 90%. O *Data Augmentation* foi descartado neste modelo, pois fez a acurácia cair para 70%.

## 5.2 Árvore de Decisão e Random Forest

Nestes modelos foi necessário realizar o “achatamento” das imagens para um vetor unidimensional, isto é, processo de conversão para cada imagem em uma representação linear, de modo que a informação espacial seja perdida e as características da imagem sejam organizadas em um único vetor. O achatamento das imagens é necessário para que os modelos de Árvore de Decisão e Random Forest possam processar as imagens como dados de entrada e realizar a classificação com base nessas informações.

Para o modelo de Árvore de Decisão foi realizado o devido treino que gerou métricas razoáveis: recall, precisão e acurácia obtiveram o valor de 80%. Mesmo com estas conquistas não foi suficiente para que se considerasse um modelo apto para o tipo de dataset.

Com isso, os dados foram treinados utilizando o modelo Random Forest, que, diferente do antecessor, gerou melhores resultados. Vale destacar que para este modelo foi utilizado um hiperparâmetro *n_estimators* definido em 100, pois trata-se justamente do número de árvores de decisão que serão criadas na floresta. Neste processo, cada árvore será treinada em uma amostra aleatória do conjunto de dados e a combinação das previsões de todas as árvores é usada para tomar a decisão final.

Logo que foram realizadas tais mudanças, o modelo atingiu valores mais interessantes: recall, precisão e acurácia obtiveram o valor de 90%. Após esta análise, o modelo de Árvore de Decisão foi descartado.

### 5.3 Rede Neural

Neste modelo, aplicamos o conceito de Redes Neurais Convolucionais (Convolutional Neural Networks - CNNs). Esses modelos têm uma alta capacidade de representação dos dados. O aprendizado de características é realizado por meio de operações de convolução, agrupamento e pooling. A etapa de classificação é composta, usualmente, por uma camada fully connected com função de ativação *PReLU* e uma camada de saída softmax.

O algoritmo começa carregando o modelo pré-treinado *DenseNet121*, que foi treinado em um grande conjunto de dados do ImageNet. Esse modelo já possui pesos otimizados para o reconhecimento de padrões em imagens. Em seguida, definimos a camada de entrada, que espera tensores de entrada com dimensões (64, 64, 3), onde 64 representa altura e largura, respectivamente, e 3 representa os três canais de cor (RGB). Somente então podemos prosseguir para o próximo passo, que envolve a aplicação de uma camada de convolução. Utilizamos uma camada de convolução 2D com 3 filtros de tamanho 3x3. Essa camada tem o objetivo de adaptar o número de canais de entrada para 3, garantindo que a imagem tenha três canais RGB.

Em seguida, conectamos à *DenseNet121*. O tensor resultante da camada de convolução 2D acima é passado como entrada para o modelo *DenseNet121*, permitindo aproveitar as características extraídas pelo modelo para melhorar o desempenho do modelo final. Na sequência, usamos a camada de pooling. Optamos por utilizar o objeto *MaxPooling*, que mostrou ter uma pequena melhora em relação ao *GlobalAveragePooling2D*, fornecendo resultados melhores. Isso ocorre porque é mais informativo usar o maior valor dentro de uma janela do que calcular a média, o que ajuda a reduzir a dimensionalidade espacial do tensor resultante do *DenseNet121*.

Ao usar a camada de achatamento (Flatten), o tensor resultante do *MaxPooling* é transformado em um vetor unidimensional, permitindo que seja alimentado em uma camada densa (totalmente conectada). Adicionamos uma camada densa com 256 neurônios, como descrito, e escolhemos a ativação *PReLU* (Parametric Rectified Linear Unit) para essa camada. A função *PReLU* permite ter parâmetros ajustáveis para controlar a inclinação da função negativa, evitando que neurônios se tornem "mortos" durante o treinamento. Isso significa que eles não ficam inativos e, assim, contribuem

para a aprendizagem. Além disso, ajuda a evitar algum tipo de viés. Embora não tenha havido melhoras perceptíveis ao comparar *PReLU*, *ReLU* e *Leaky ReLU,* escolhemos *PReLU* como uma garantia adicional na rede.

Durante o desenvolvimento e treinamento da rede densa, observamos que o aumento do número de neurônios nem sempre resultava em um aumento de desempenho da rede. Realizamos um teste com 512 neurônios, executando 100 epochs, onde um valor maior de neurônios resultou em uma perda significativa de acurácia.

Também utilizamos a técnica de *Dropout* com uma taxa de 0,3 (30%). O Dropout é uma técnica de regularização que ajuda a evitar o overfitting, desativando aleatoriamente alguns neurônios durante o treinamento.

Na camada de saída da rede, adicionamos uma camada densa de saída com 2 neurônios e função de ativação softmax. Essa camada gera as probabilidades de cada classe de saída. Neste modelo, temos 2 classes possíveis (binário).

### 5.3.1 Compilação do modelo

O modelo é compilado com a função de perda categorical_crossentropy, que é adequada para problemas de classificação com múltiplas classes. O otimizador utilizado é o Adam, com uma taxa de aprendizado de 1e-3, beta_1=0.9, beta_2=0.999 e epsilon=0.1.

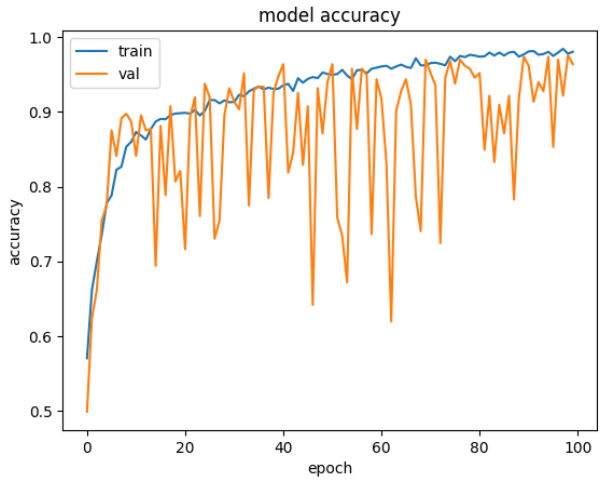
O modelo foi treinado levando em consideração a divisão dos dados em conjunto de treinamento e teste, com 80% dos dados atribuídos ao conjunto de treinamento (X_train e y_train) e 20% aos dados de teste (X_test e y_test). O treinamento foi realizado com 100 epochs e um batch size de 32. Esse tamanho de batch foi escolhido considerando o equilíbrio entre tempo e desempenho de aprendizado, em comparação com batch sizes de 16 e 64, respectivamente.

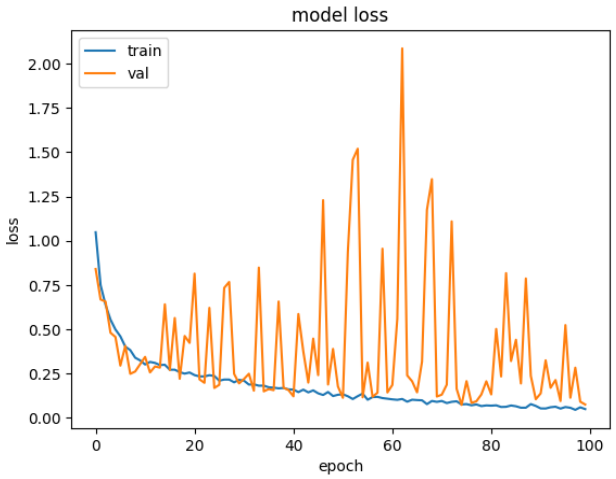
### 5.3.2 Métricas

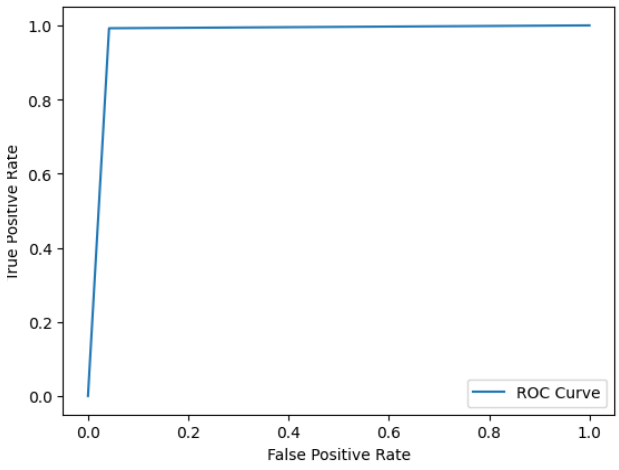
Foram escolhidas algumas métricas para avaliar o desempenho do modelo. Essas são as métricas utilizadas com os valores do modelo final já preparados. As métricas: Accuracy, F1 Score, AUC score, Recal Score e Precision obtiveram aproximadamente 97.5%.

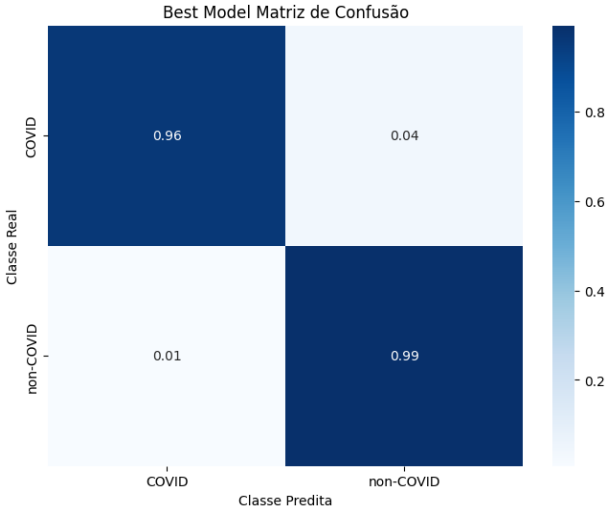
### 5.3.3 Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.99 | 0.96 | 0.97 | 238 |
| 1 | 0.96 | 0.99 | 0.98 | 259 |




## 5.4 Regressão Logística

Outro modelo testado com a base foi um regressor logístico, onde houveram muitos testes diferentes com algumas tentativas de melhoria da performance do modelo.

Primeiramente, decidimos rodar o modelo com as imagens redimensionadas para 64x64 pixels de resolução em formato de cores em tons de cinza e outro teste com as imagens coloridas.

Após os resultados, notamos que o desempenho do modelo em tons de cinza teve um desempenho superior, cerca de +3% de acurácia de teste em relação ao modelo com cores RGB. O que faz sentido porque as imagens de tomografia não possuem uma gama de cores em que o RGB traria mais informações para o dataset. A partir daqui, resolvemos adotar a premissa de que o dataset deveria continuar sendo tratado em tons de cinza.

Feito isso, aplicamos normalização nos valores dos pixels da imagem, mas desta vez sem muita variação no desempenho do modelo.

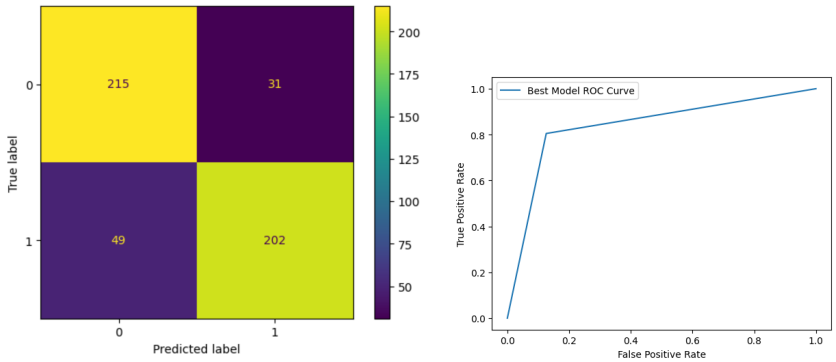
Então, decidimos adicionar um seletor de features para diminuir a quantidade de features e tentar se livrar de features que não possuem muito valor. Usando o SelectorPercentile para remover 20% das features do dataset.

Outras tentativas de melhoria do algoritmo foram aplicadas, como um seletor das features mais importantes, o uso de Data Augmentation para gerar mais dados parecidos a fim de melhorar o treinamento e também a aplicação de GridSearch para encontrar os melhores hiper-parâmetros para features do regressor logístico, como regularização, número de iterações e outros.

No fim, mesmo aplicando essas técnicas o modelo com melhor desempenho foi um dos modelos iniciais que contava apenas com a normalização das imagens com uma acurácia de teste de 84%.

Métricas do melhor modelo de regressão logística:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| non-covid | 0.81 | 0.87 | 0.84 | 246 |
| covid | 0.87 | 0.80 | 0.83 | 251 |
| accuracy | | | 0.84 | 497 |
| macro avg | 0.84 | 0.84 | 0.84 | 497 |
| weighted avg | 0.84 | 0.84 | 0.84 | 497 |




### Stochastic Gradient Descent

A mesma estratégia e passo-passo do modelo de regressão logística foi aplicada para um modelo SGD, e os resultados que obtivemos foram diferentes.

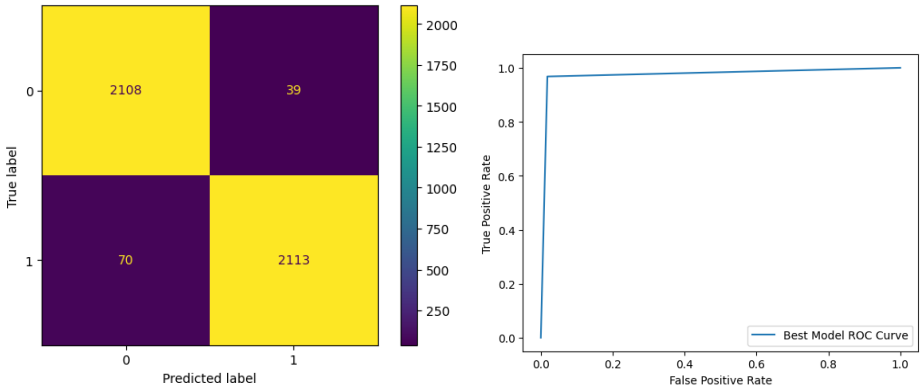
O melhor modelo obteve melhores resultados usando a escala de cores RGB, seleção de 80% das features mais importantes e normalização.

## 5.5 Suporte Vector Machine

Aqui também usando a estratégia e passo-passo do modelo de regressão logística, mas desta vez usando um modelo de SVM. Este modelo leva mais tempo na fase de treinamento dependendo da quantidade de features e o volume de dados, mas os resultados compensaram no final.

O melhor modelo obteve melhores resultados usando a escala de cores de tons de cinza e aumentando o dataset com Data Augmentation, com uma acurácia de teste de 97,4%.

| | | | | |
|---|---|---|---|---|
| non-covid | 0.97 | 0.98 | 0.97 | 2147 |
| covid | 0.98 | 0.97 | 0.97 | 2183 |
| | | | | |
| accuracy | | | 0.97 | 4330 |
| macro avg | 0.97 | 0.97 | 0.97 | 4330 |
| weighted avg | 0.97 | 0.97 | 0.97 | 4330 |




## 6. Conclusão

Após uma série de tentativas com modelos diferentes, chegamos a conclusão de que o melhor modelo é o SVM, fizemos esta escolha com base nos resultados e também no tempo de execução de treinamento. Em redes neurais, por exemplo, conseguimos resultados parecidos porém com um tempo de treinamento muito alto e com um modelo mais complexo.

## 7. Referências


1. Silva R, Silva Neto DR da. Inteligência artificial e previsão de óbito por Covid-19 no Brasil: uma análise comparativa entre os algoritmos Logistic Regression, Decision Tree e Random Forest. Saúde debate [Internet]. 2022 Dec; 46(spe8):118–29. Available from: https://doi.org/10.1590/0103-11042022E809
2. Keidar, D., Yaron, D., Goldstein, E. et al. COVID-19 classification of X-ray images using deep neural networks. Eur Radiol 31, 9654–9663 (2021). https://doi.org/10.1007/s00330-021-08050-1
3. Guhathakurata S, Kundu S, Chakraborty A, Banerjee JS. A novel approach to predict COVID-19 using support vector machine. Data Science for COVID-19. 2021:351–64. doi: 10.1016/B978-0-12-824536-1.00014-9. Epub 2021 May 21. PMCID: PMC8137961.
4. Bertozzo, Richard Junior. Aplicação de Machine Learning em dataset de consultas médicas do SUS (2019). https://repositorio.ufsc.br/handle/123456789/202663
5. Theerthagiri, Prasannavenkatesan & Jacob, I. Jeena & Ruby, A.Usha & Yendapalli, Vamsidhar. (2020). Prediction of COVID-19 Possibilities using KNN Classification Algorithm. 10.21203/rs.3.rs-70985/v2.
6. Freeman K, Geppert J, Stinton C, Todkill D, Johnson S, Clarke A et al. Use of artificial intelligence for image analysis in breast cancer screening programmes: systematic review of test accuracy BMJ 2021; 374 :n1872 doi:10.1136/bmj.n1872
7. Jayakumar, S., Sounderajah, V., Normahani, P. et al. Quality assessment standards in artificial intelligence diagnostic accuracy systematic reviews: a meta-research study. npj Digit. Med. 5, 11 (2022). https://doi.org/10.1038/s41746-021-00544-y
8. Soares, Eduardo, Angelov, Plamen, Biaso, Sarah, Higa Froes, Michele, and Kanda Abe, Daniel. "SARS-CoV-2 CT-scan dataset: A large dataset of real patients CT scans for SARS-CoV-2 identification." medRxiv (2020). doi: https://doi.org/10.1101/2020.04.24.20078584.
9. Angelov, P., & Soares, E. (2020). Towards explainable deep neural networks (xDNN). Neural Networks, 130, 185-194.