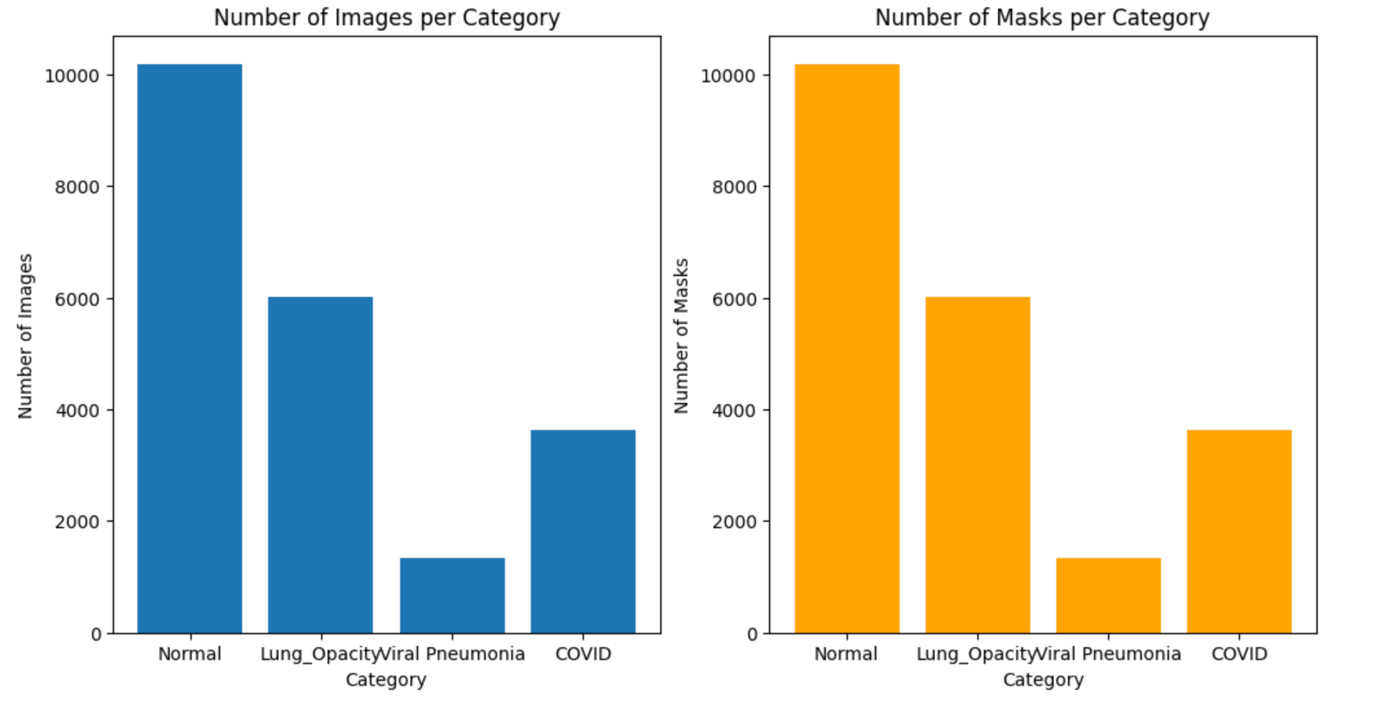
**COVID – 19 X-rays Analysis**

**Preprocessing and Modelling Report**

This report will focus on different methodologies of image preprocessing and applying different models for machine and deep learning.

1. **Preprocessing**

The used dataset, containing X-rays (named images) and masks, is not balanced, as shown in Figure 1.1.



**Figure 1.1.** Visualization of number of images (left, blue) and masks (right, orange) per category.

This is important since most of the Machine Learning algorithms are based on the assumption that the dataset is balanced. Balanced dataset is the dataset with data equally distributed among all of its classes. Training a model on an imbalanced dataset is surely possible. However, the learning becomes biased towards the majority classes.

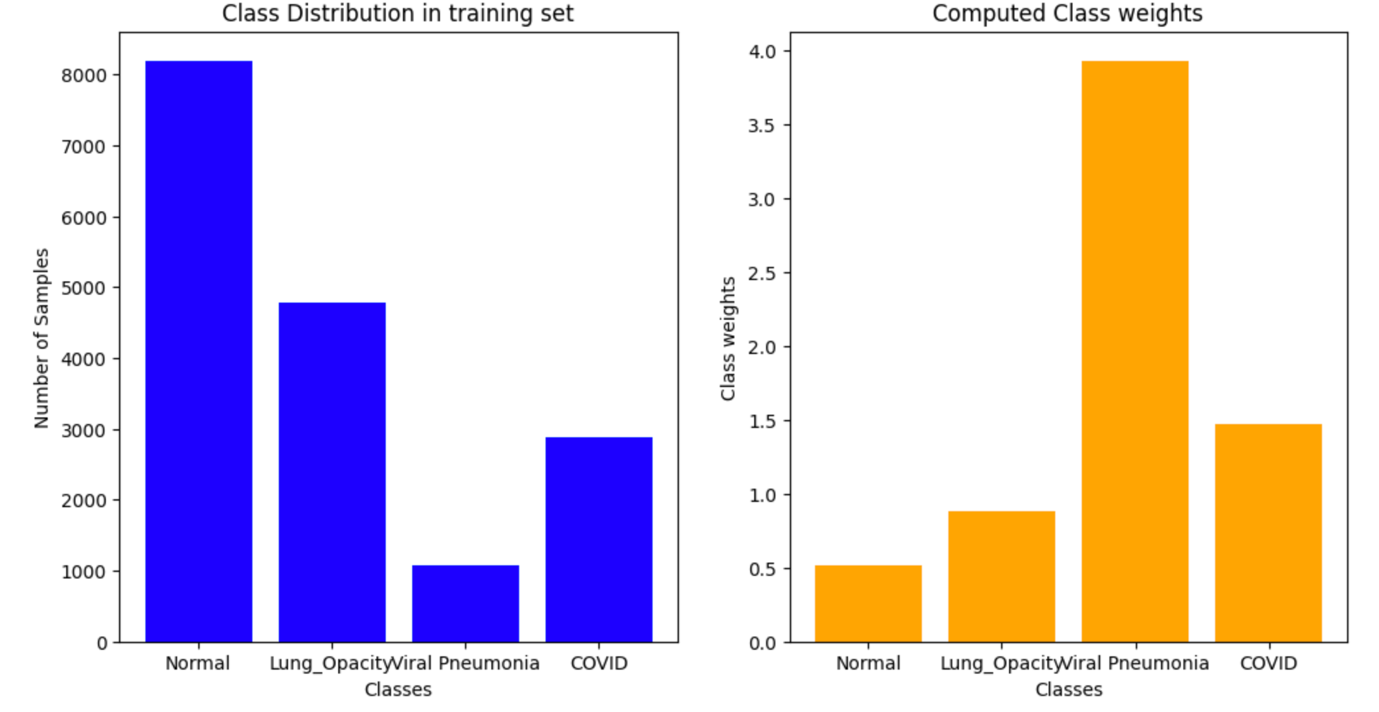
The imbalanced dataset containing images could be handled with Sample Weighting in Loss Function. The method requires to weight the loss computed for different samples differently, depending whether they belong to the majority or the minority classes. In the simple words, more priority is given to the category having less samples.

Other possible techniques to handle imbalanced dataset are oversampling and undersampling, and generating synthetic samples, encoding the labels, and splitting the data.

In the first part of this analysis, we were focused on the statistical weighting of our dataset. Statistical or class weighting is straightforward to implements. It does not alter the number of samples, unlike oversampling techniques such as Synthetic Minority Oversampling Technique (SMOTE) or undersampling that can artificially and randomly inflate or reduce dataset size. Statistical weighting reduces the risk of overfitting, since it does not create duplicate samples or synthetic data. This cannot be said for oversampling with SMOTE. It is important to underline that statistical weighting is compatible with most algorithms, classical such as Linear Regression, Random Forest, or Boosting, but also with Convolutional Neural Networks (CNN).

The class weights were implemented on the model previously split with the test size of 20%. From the total of 21165 images, 16932 images were in the training set, and 4233 in the testing set. Before splitting the model, first we checked whether all the images are grayscale. Since this was not the case with all images in the category ***Viral Pneumonia***, the code for converting them into grayscale was run. Then, images were converted into np.arrays (X and y). By flattening the images, 2D images were transformed into 1D arrays.

Figure 1.2. visualize the connection class distribution in training set and computed class weights.



**Figure 1.2.** The distribution of images in training set (left, blue) and computed class weights (right, orange)

From Figure 1.2., we can conclude that the most of priority is given to the category named Viral Pneumonia, since the number of images in this category is the lowest. On the other side, the least priority is given to the category Normal, since its number of images is the highest.

Furthermore, before starting with Machine Learning, the standard scaler was imported. Standardizing the images ensures uniform scale, or in the other words, that each pixel value has a similar range. It helps also mitigating the effect of different lighting conditions and shadows, making the model more robust to variation in image capture conditions. This may be particularly important for X-ray datasets. Standard scaler also reduces the influence of outliers (extreme pixels values). The outliers can distort the learning process.

1. **Modelling**

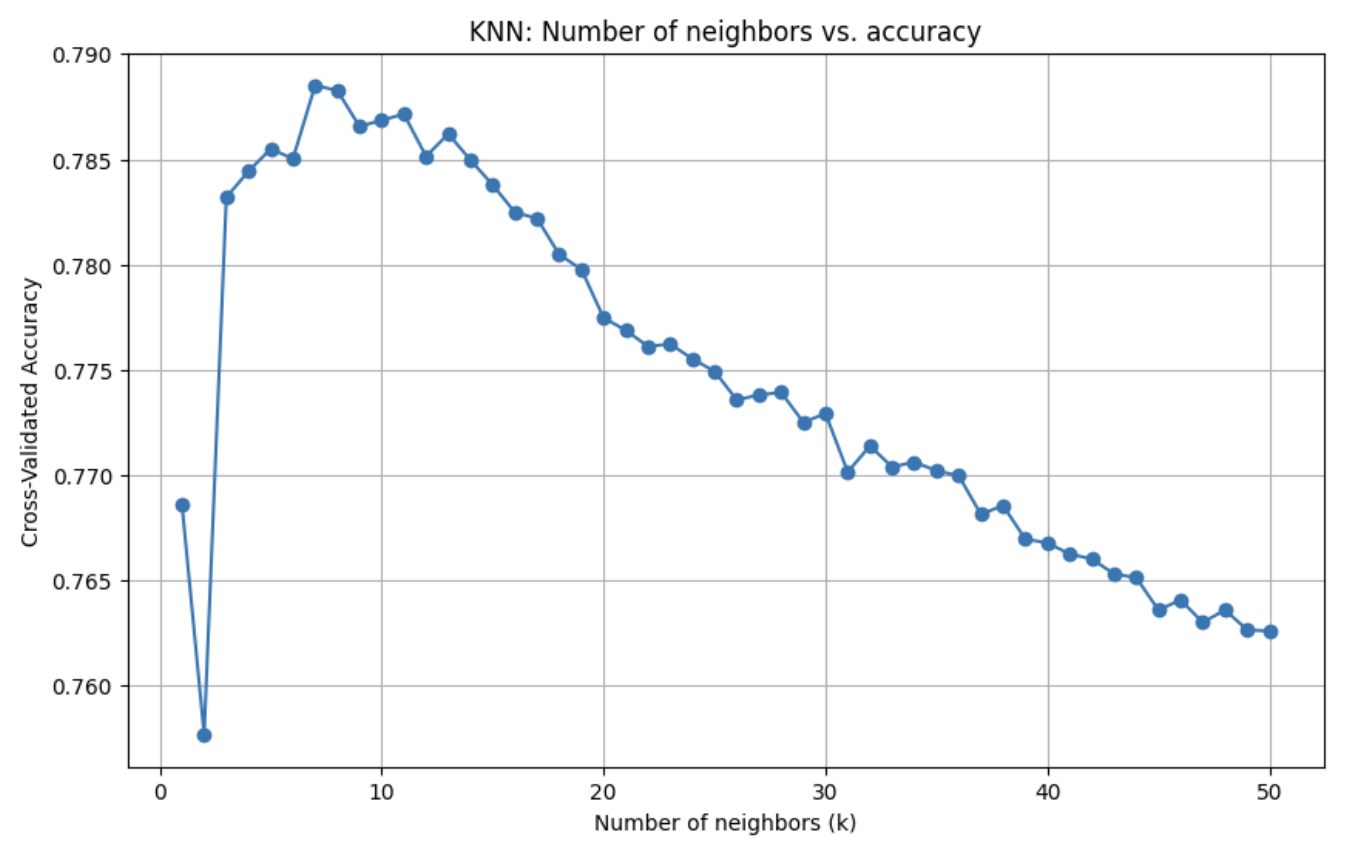
Even though we work with images, with the aim to go into deep learning, running classical models is important for the sake of learning, and showing why deep learning techniques, such as CNN, are more suitable.

The classical models done are Linear Regression, Support Vector Machine, Random Forest, KNeighbors Classifier (KNN), Decision Tree, Bagging, AdaBoost, XGBoost, and Voting. The accuracies and total training times for each model are summarized in Table 2.1.

**Table 2.1.** Accuracies and Total training time for each Classification model. Models with highest accuracies are in blue.

|  |  |  |
| --- | --- | --- |
| **Model** | **Accuracy** | **Training time (min)** |
| Logistic Regression | 0.74 | 24.30 |
| Support Vector Machines | 0.76 | 204.14 |
| **Random Forest** | **0.84** | **7.50** |
| KNN | 0.78 | 5.89 |
| Decision Tree | 0.70 | 28.06 |
| **Bagging** | **0.83** | **520.35** |
| AdaBoosting | 0.71 | 30.17 |
| **XGBoost** | **0.88** | **229.65** |
| **Voting** | **0.84** | **762.48** |

Before training KNN model, the number of neighbors was optimized, Figure 2.1., and it was obtained that the biggest accuracy is when the number of neighbors is 7.



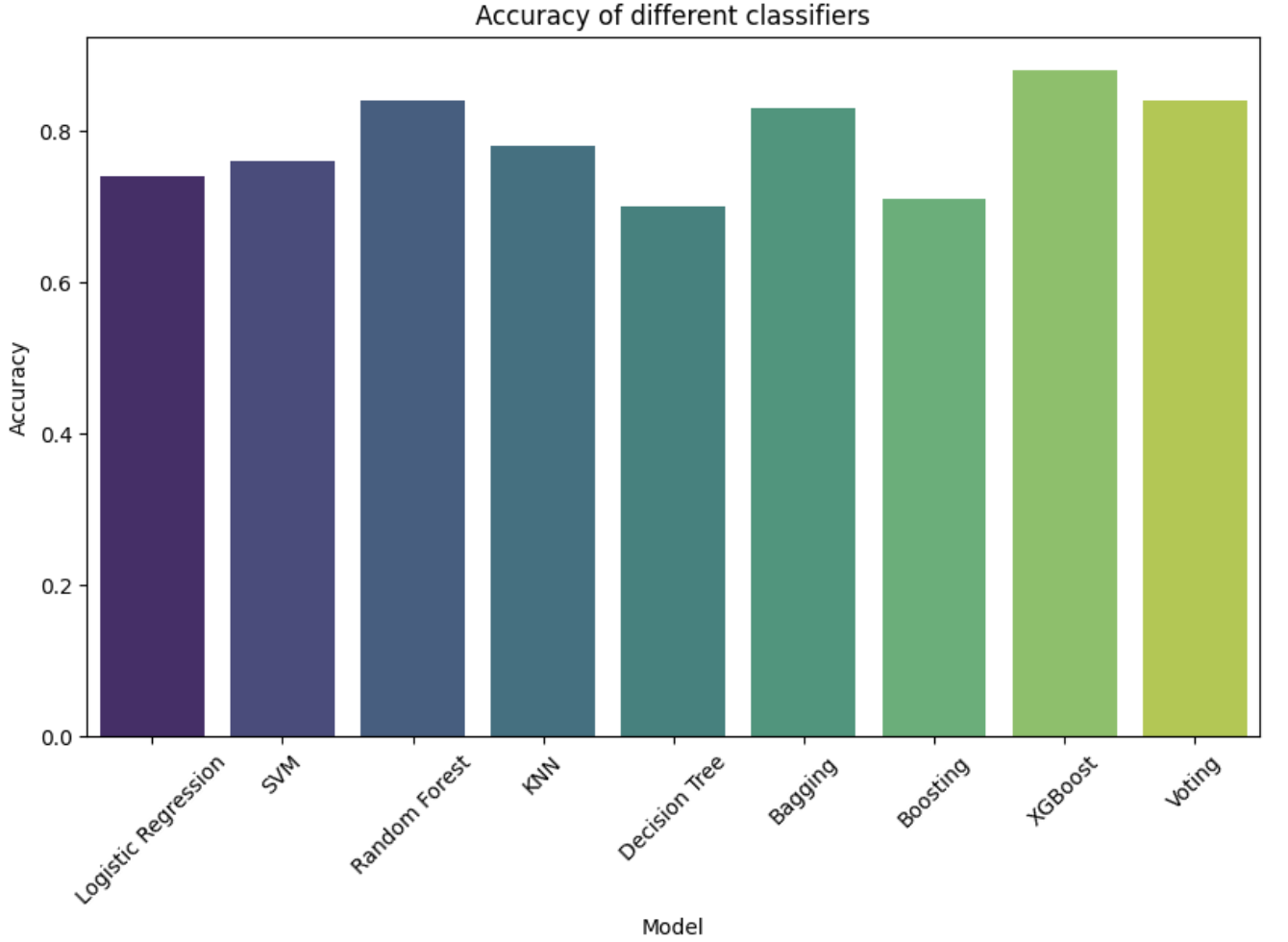
**Figure 2.1.** KNN model optimization; Number of neighbors vs. accuracy

Voting Classifier was run with the followed estimators: Random Forest Classifier, Bagging Classifier and XGB Classifier, since these three gave the best accuracies. Despite high accuracy, total training time for Voting Classifier was the highest. It is important to note that training time depends on the computer performances and will be different depending on the computer. However, all the classification models were run on the same computer, and comparing training times of each model makes sense. This may be especially important when making decision with which model to continue to work.

It is also important to emphasize that we run Stacking Classification Model as well. However, since the training time took more than 2 days, the training was stopped and the results were not obtained.

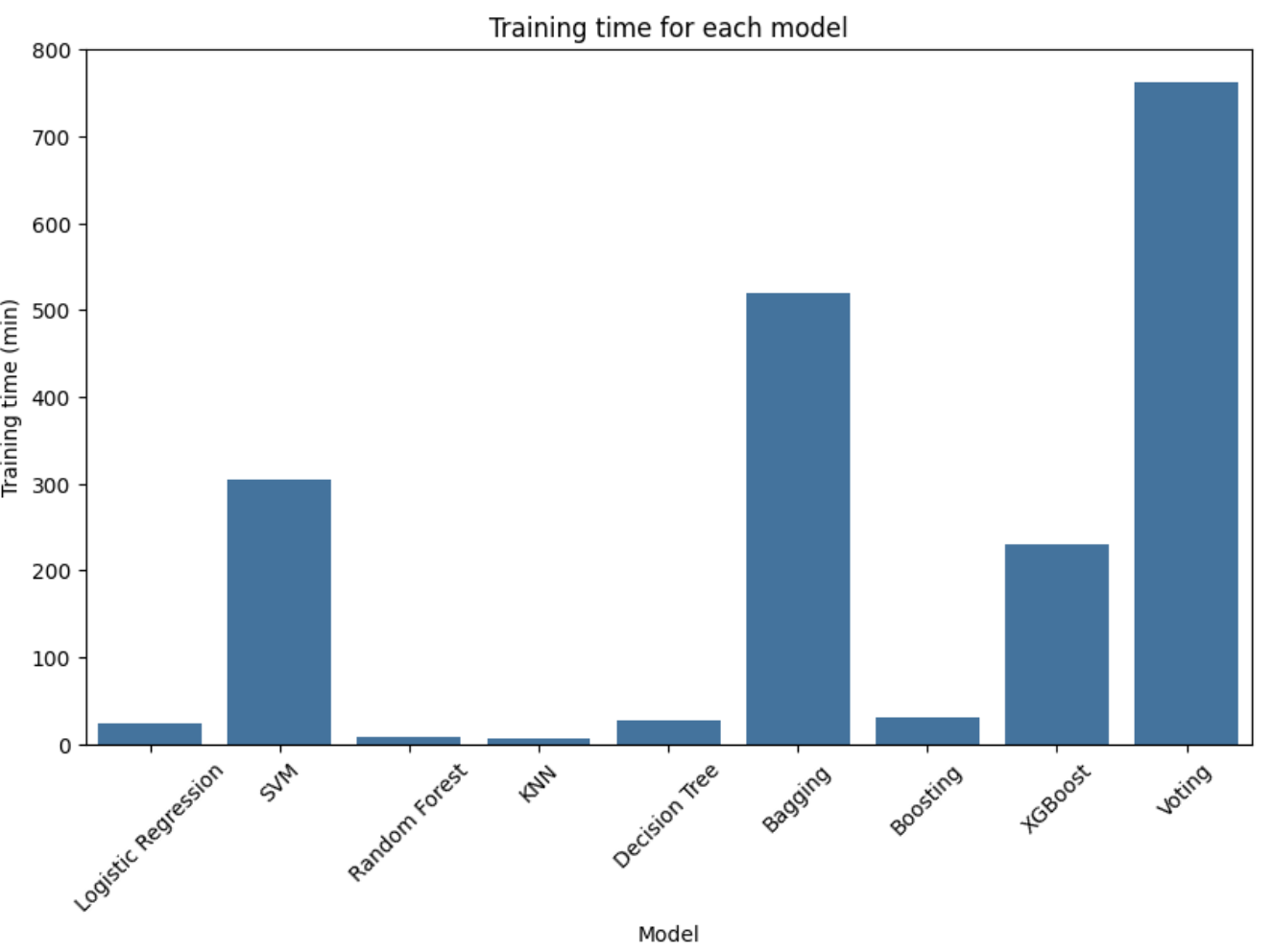
For the better presentation, accuracies and training times are visualized and presented in Figures 2.2. and 2.3., respectively.

Along with accuracies and training times, we have calculated Precision, Recall, and F1-Score for each category and each model. The results are given in Figures 2.4., 2.5., and 2.6.

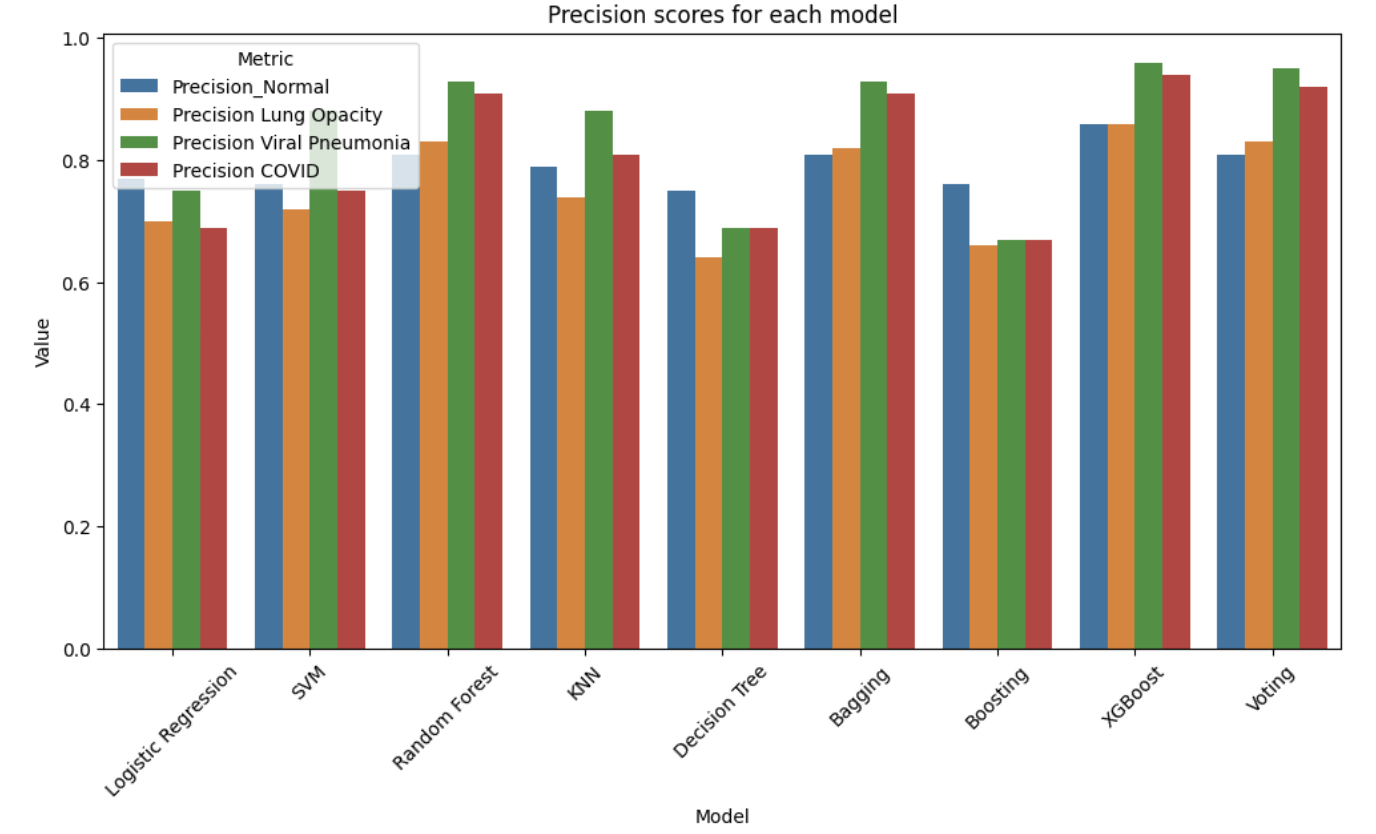


**Figure 2.2.** Accuracies of different Classification models

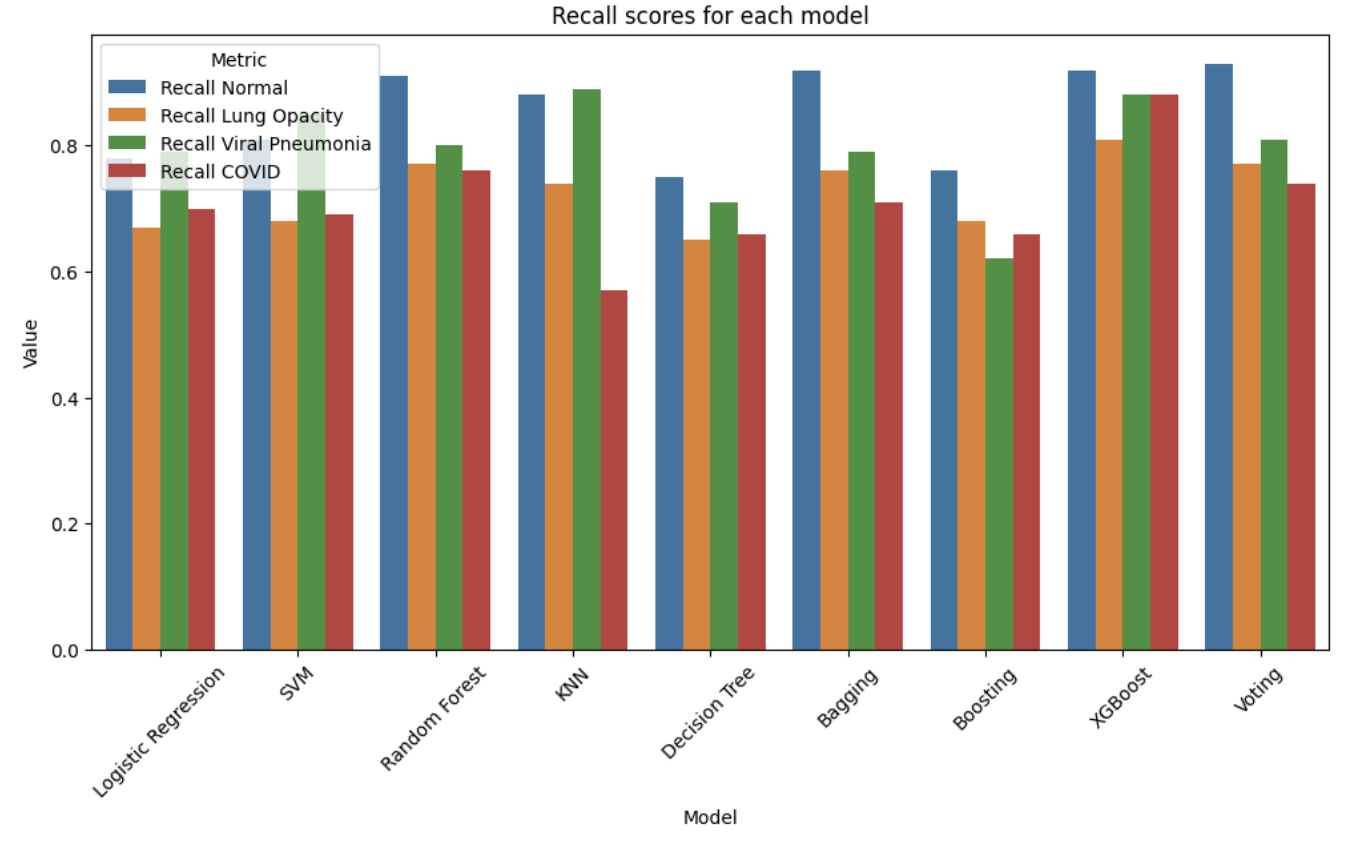
As we saw in the table, the biggest accuracies have Random Forest, Bagging, XGBoost, and Voting classifiers. Giving that the training time for voting classifier was the biggest, Figure 2.3., we will not consider this model in future optimizations. The best training time had Logistic regression, Random Forest, KNN, Decision Tree, and AdaBoost. Since Random Forest had one of the best accuracies and optimal training time, we will consider it for future optimization. Also, since Bagging and XGBoost have high accuracies and acceptable training time, we will also consider these two models for future optimizations.



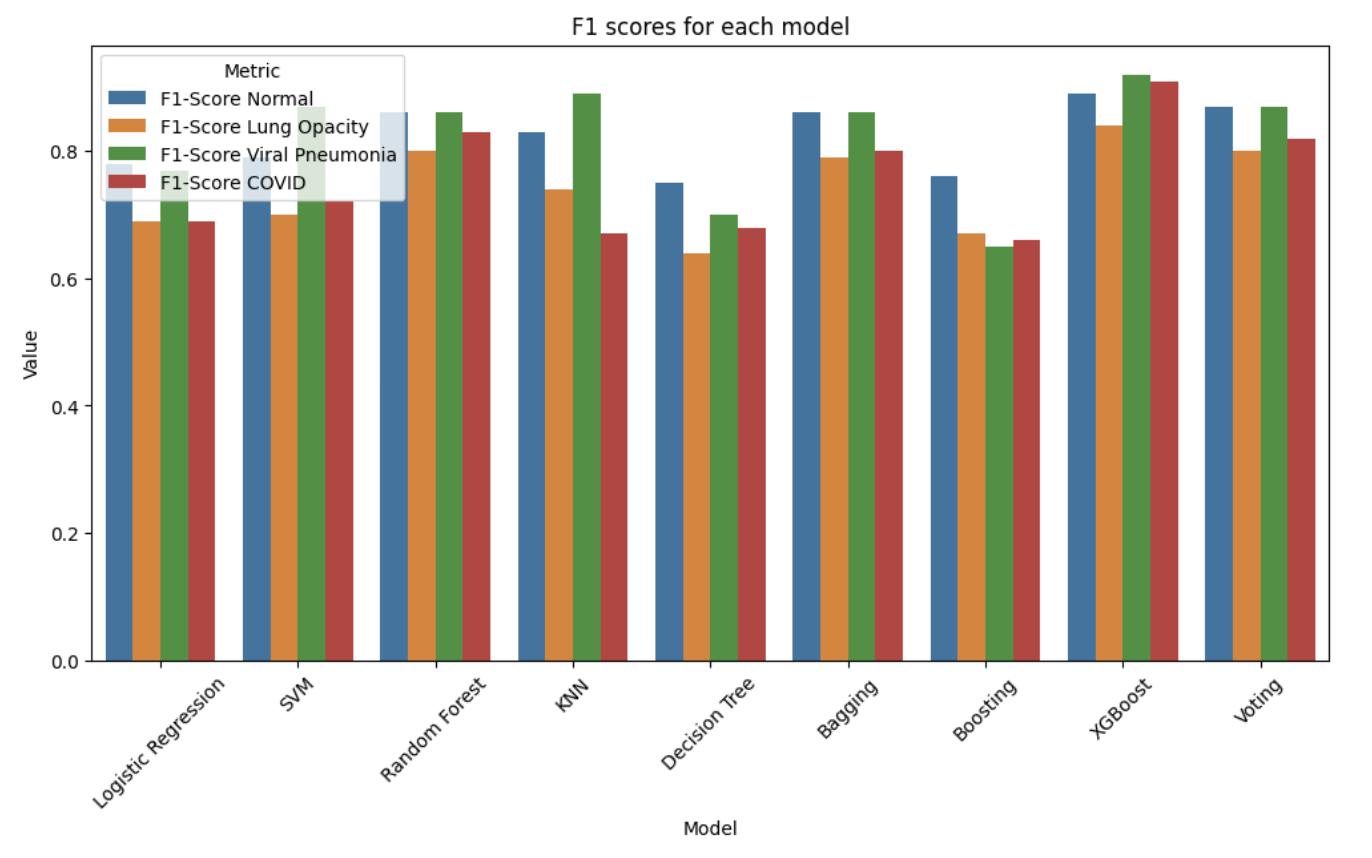
**Figure 2.3.** Total training time for each model



**Figure 2.4.** Precision Scores for each category and each model



**Figure 2.5.** Recall Scores for each category and each model



**Figure 2.6.** F1 scores for each category and each model

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  | |

**Figure 2.7.** Confusion Matrices for different Classification models

