DETECTION OF COVID-19 USING DEEP LEARNING
Final Project

Oswaldo Russian

1299650

BACKGROUND SECTION

Summary
Several learning algorithms were implemented to classify chest x-ray images as COVID-19 positive and "normal" radiographies. These include Convolutional Neural Network (CNN) architectures built through keras, and built-in scikit-learn machine learning models. The data were taken from the COVID-19 Radiography Database available on kaggle. The performance of the algorithms was evaluated through the corresponding confusion matrices.

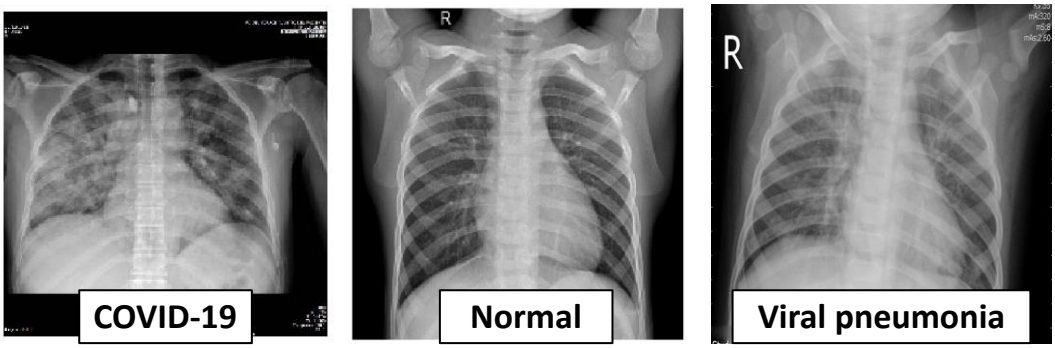
Introduction
In this class, we have learned about the power of deep learning techniques for image classification. For this project, I have tried to harness some of these tools towards the classification of chest radiography images in order to identify COVID-19 positive cases, and to distinguish COVID-19 positive cases from other respiratory ailments based only on the interpretation of such images.

The code for this project has been developed using Python. Machine learning algorithms were taken from the scikit-learn library. Convolutional neural networks have been built using keras.

METHOD SECTION

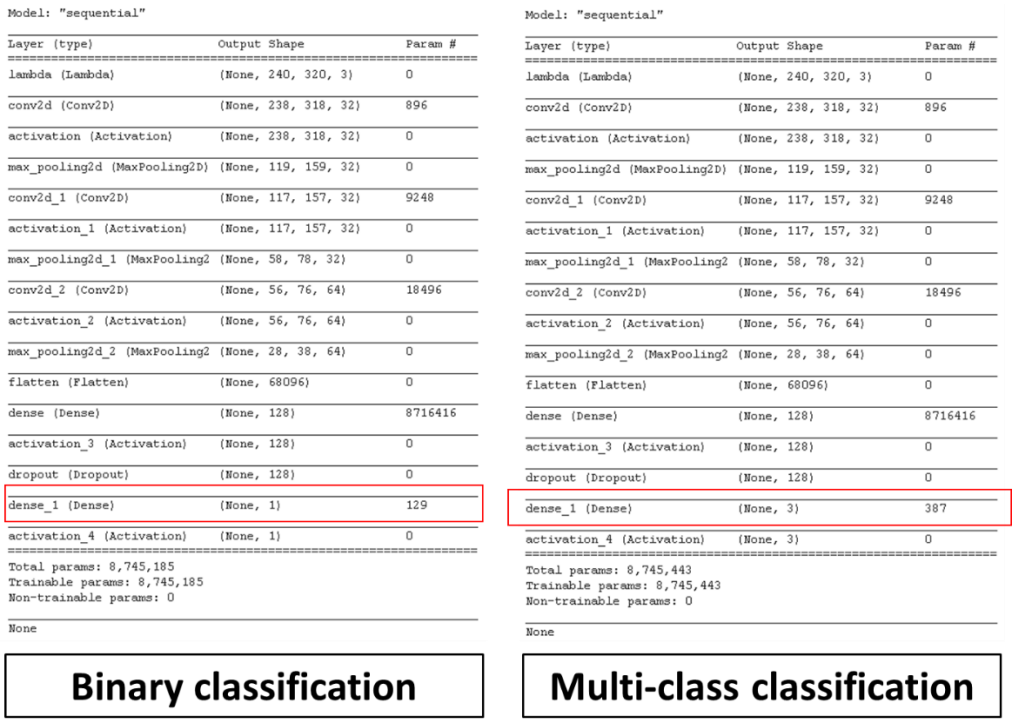
Datasets
The data for this project was taken from the publicly available COVID-19 Radiography Database. These data were provided by Tawsifur Rahman at the University of Dhaka.

The data include 219 COVID-19 positive images, 1341 normal images and 1345 viral pneumonia images. From these totals, 80% of the images were selected for training and 20% were used for testing. A sample of these images is shown below:



Models

Two convolutional neural network architectures (summarized below) were evaluated.



Model: "sequential"

Layer (type)	Output Shape	Param #
lambda (Lambda)	(None, 240, 320, 3)	0
conv2d (Conv2D)	(None, 238, 318, 32)	896
activation (Activation)	(None, 238, 318, 32)	0
max_pooling2d (MaxPooling2D)	(None, 119, 159, 32)	0
conv2d_1 (Conv2D)	(None, 117, 157, 32)	9248
activation_1 (Activation)	(None, 117, 157, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 58, 78, 32)	0
conv2d_2 (Conv2D)	(None, 56, 76, 64)	18496
activation_2 (Activation)	(None, 56, 76, 64)	0
max_pooling2d_2 (MaxPooling2	(None, 28, 38, 64)	0
flatten (Flatten)	(None, 68096)	0
dense (Dense)	(None, 128)	8716416
activation_3 (Activation)	(None, 128)	0
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129
activation_4 (Activation)	(None, 1)	0

Total params: 8,745,185
Trainable params: 8,745,185
Non-trainable params: 0

None

Binary classification

Model: "sequential"

Layer (type)	Output Shape	Param #
lambda (Lambda)	(None, 240, 320, 3)	0
conv2d (Conv2D)	(None, 238, 318, 32)	896
activation (Activation)	(None, 238, 318, 32)	0
max_pooling2d (MaxPooling2D)	(None, 119, 159, 32)	0
conv2d_1 (Conv2D)	(None, 117, 157, 32)	9248
activation_1 (Activation)	(None, 117, 157, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 58, 78, 32)	0
conv2d_2 (Conv2D)	(None, 56, 76, 64)	18496
activation_2 (Activation)	(None, 56, 76, 64)	0
max_pooling2d_2 (MaxPooling2	(None, 28, 38, 64)	0
flatten (Flatten)	(None, 68096)	0
dense (Dense)	(None, 128)	8716416
activation_3 (Activation)	(None, 128)	0
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 3)	387
activation_4 (Activation)	(None, 3)	0

Total params: 8,745,443
Trainable params: 8,745,443
Non-trainable params: 0

None

Multi-class classification

The only difference between these architectures is introduced in the final layer (prior to activation). Because of the selected loss function (sparse categorical cross entropy), the size of the output layer had to be equal to the number of classes, which is the reason for the change. For binary classification, binary cross entropy loss was used.

The following machine learning models were also implemented:

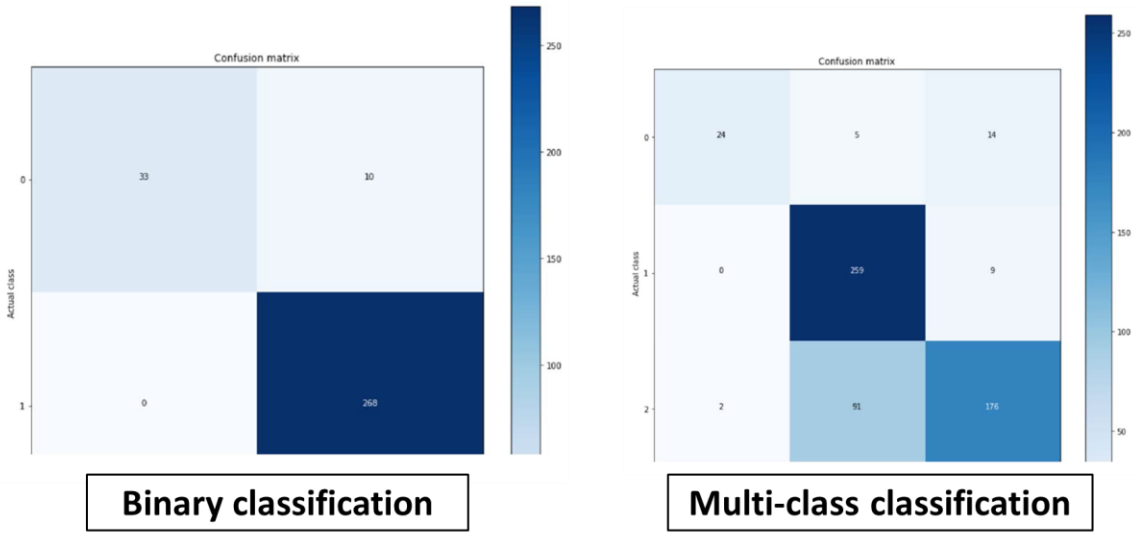
- K-Neighbors Classifier
- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- Support Vector Classifier
- Multilayer Perceptron

EXPERIMENT SECTION

A binary classification experiment was conducted to evaluate the performance of the noted models. In binary classification experiments discussed in this report, the two classes considered are: (0) COVID-19 positive chest X-rays, and (1) Normal chest X-rays. In multi-class classification experiments, an additional class is considered as (2) Viral pneumonia chest X-rays. This additional class was included to illustrate the effect of introducing noise on the predictions.

Deep Learning Models

The confusion matrices obtained through the noted CNN architectures for binary and multi-class classification are shown below. In both cases, only ten epochs were considered.

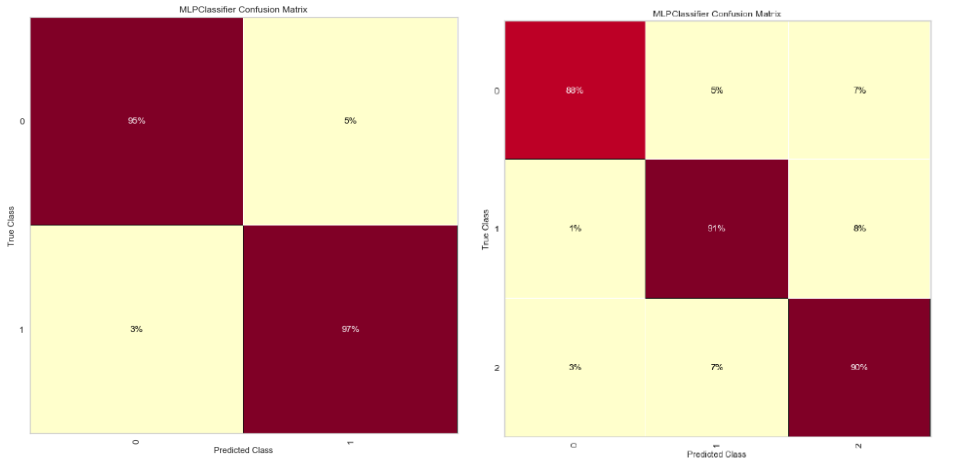


In the case of binary classification, images of normal (healthy) chest X-rays are correctly classified as such for every observation. That is, no false-positives were observed. However, of 43 COVID-19 positive testing images, ten are misclassified as normal.

In the case of multi-class classification, the confusion matrix reflects the effect of the added noise introduced by considering a separate class of respiratory ailments. Similarly to the case of binary classification, a fairly significant number of COVID-19 positive observations were misclassified as either Normal (5) or Viral Pneumonia (14). Better performance was obtained for normal images, with none being classified as COVID-19 positive, and only 9 of 258 being classified as Viral Pneumonia. An interesting result was observed in the number of Viral Pneumonia images misclassified as Normal (91 of 279), compared to only 2 of 279 misclassified as COVID-19 images. This is interpreted as Normal and Viral Pneumonia X-rays being more similar (in terms of the features that the CNNs capture) to each other than they are to COVID-19 positive X-rays. This insight is also supported by noting that no Normal images were misclassified as COVID-19 positive in either binary or multi-class classification schemes.

Machine Learning Models

For both binary classification and multi-class classification problems, the Multilayer Perceptron algorithm exhibited the best results. The default parameters were used in the Multilayer Perceptron implementation for binary classification, while the parameters were optimized for multi-class classification using GridSearchCV. For this method, the confusion matrices are shown below. It is seen here that there is a false-positive rate of 5%, while 3% of COVID-19 positive cases were not detected (they were classified as Normal). For multi-class classification, we notice that the MLP performs better than our CNN architecture in identifying Viral Pneumonia images.



Binary classification **Multi-class classification**

CONCLUSIONS AND FUTURE WORK

For binary classification, we summarize the accuracies of the tested methods below:

Model	Accuracy score
Logistic Regression	0.981
Decision Tree	0.923
Random Forest	0.968
Support Vector Classifier	0.965
Multi-layer Perceptron	0.974
Convolutional Neural Network	0.967

For binary classification, our CNN matches up well with built-in machine learning models at only ten epochs. With increased epochs or increased data quantities, the deep learning algorithm should outperform the machine learning models. It is to be examined, however, why the deep learning algorithm (CNN) was more sensitive to noise than the machine learning models. Although, it is noted that the machine learning models for multi-class classification were subject to parameter optimization.

REFERENCES

1. M.E.H. Chowdhury, T. Rahman, A. Khandakar, R. Mazhar, M.A. Kadir, Z.B. Mahbub, K.R. Islam, M.S. Khan, A. Iqbal, N. Al-Emadi, M.B.I. Reaz, M. T. Islam, "Can AI help in screening Viral and COVID-19 pneumonia?". IEEE Access, Vol. 8, 2020, pp. 132665 - 132676.
2. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python". JMLR 12, 2011, pp. 2825-2830.
3. Chollet, F., & others. (2015). Keras. GitHub. Retrieved from https://github.com/fchollet/keras

CODE DESCRIPTION AND CODE

Descriptions and summaries of the various codes used for this project are presented in the first page of the output files. These are shown in the following pages. The only exception is the binary classification code using machine learning models, as it follows a very similar format as the code for multi-class classification code using machine learning models, which is thoroughly described. This is considered appropriate because machine learning models are not the priority of this project.

DEEP LEARNING – BINARY

CODE SUMMARY: in this code we are following a similar format as we did for the case of machine learning models. We start a bit differently, though, by setting up colab (which we didn't do for ML) and importing relevant libraries. Then, we define our training parameters. After that we load and visualize our data for each class. We use the get_data function in ourder to build our training and testing arrays. We then define our CNN and proceed to fit the model to our data. After that, we plot performance metrics with particular emphasis on the confusion matrix.

```
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)
import os
os.chdir("/content/gdrive/My Drive/Colab Notebooks/Final_Project")
```

```
Mounted at /content/gdrive
```

```
#Basic
import numpy as np
import cv2
import scipy
import os
from keras.utils import np_utils
import random

#Pre-processing
import glob
from keras.preprocessing.image import ImageDataGenerator
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from time import time

#Model
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, Lambda
from keras.layers import Dense
from keras.utils import np_utils

#Visualizers
%matplotlib inline
import matplotlib.pyplot as plt
```

```
epochs = 10
```

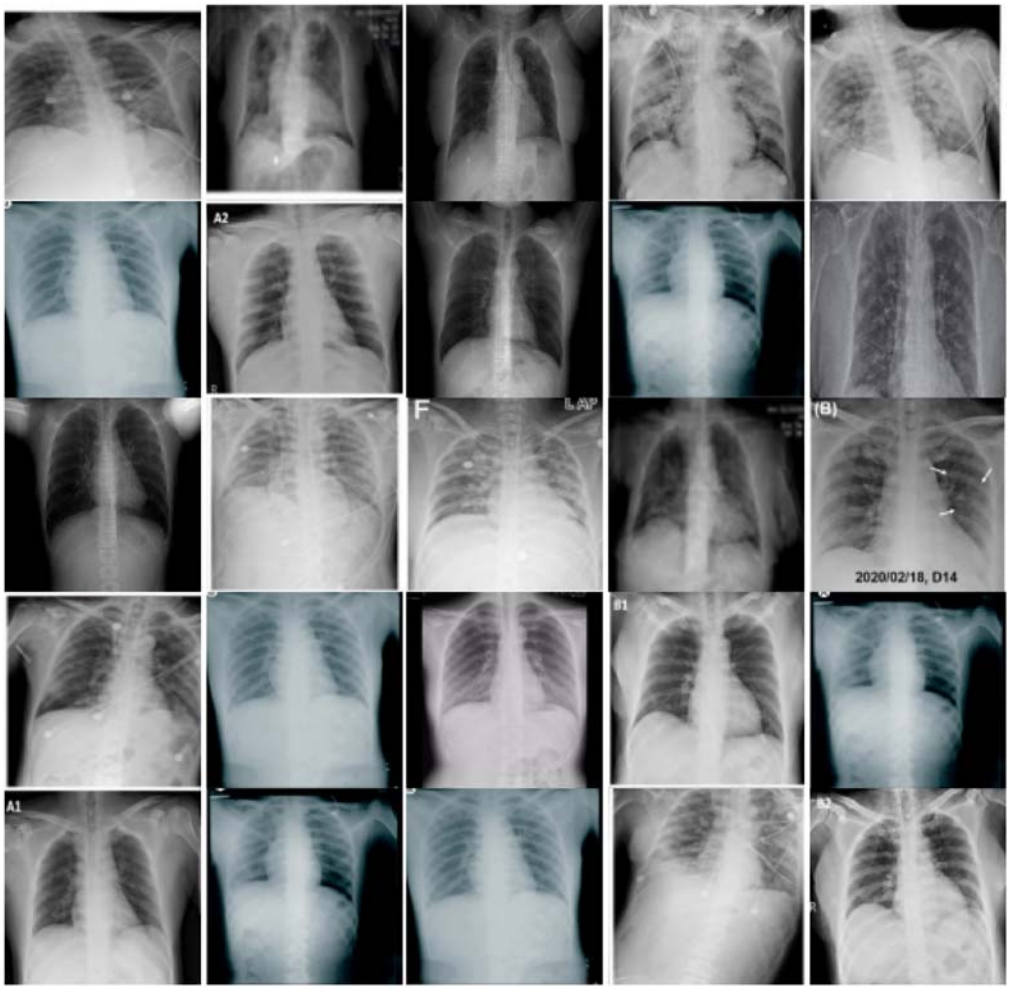
```
BASE_DIR = "/content/gdrive/My Drive/Colab Notebooks/Final_Project/"
batch_size = 32


imagePatches = glob.glob(BASE_DIR +'dataset/Train/Class1/*.png', recursive=True)
for filename in imagePatches[0:10]:
    print(filename)
```

```
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (125).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (112).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (106).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (109).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (124).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (101).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (120).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (107).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (102).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (1).png
```

```
image_name = "/content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19(144).png" #Image to be used as que
def plotImage(image_location):
    image = cv2.imread(image_name)
    image = cv2.resize(image, (224,224))
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)); plt.axis('off')
    return
plotImage(image_name)
```

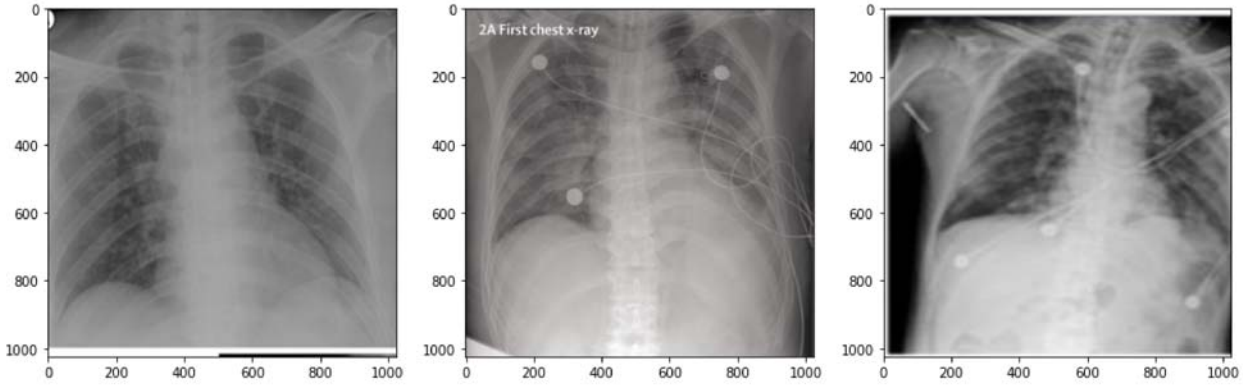
```
# Plot Multiple Images
bunchOfImages = imagePatches
i_ = 0
```

```
    plt.rcParams['figure.figsize'] = (10.0, 10.0)
    plt.subplots_adjust(wspace=0, hspace=0)
    for l in bunchOfImages[:25]:
        im = cv2.imread(l)
        im = cv2.resize(im, (224, 224))
        plt.subplot(5, 5, i_+1) #.set_title(l)
        plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB)); plt.axis('off')
        i_ += 1
```




```
    def randomImages(a):
        r = random.sample(a, 4)
        plt.figure(figsize=(16,16))
```

```
        plt.subplot(131)
        plt.imshow(cv2.imread(r[0]))
        plt.subplot(132)
        plt.imshow(cv2.imread(r[1]))
        plt.subplot(133)
        plt.imshow(cv2.imread(r[2]));
    randomImages(imagePatches)
```

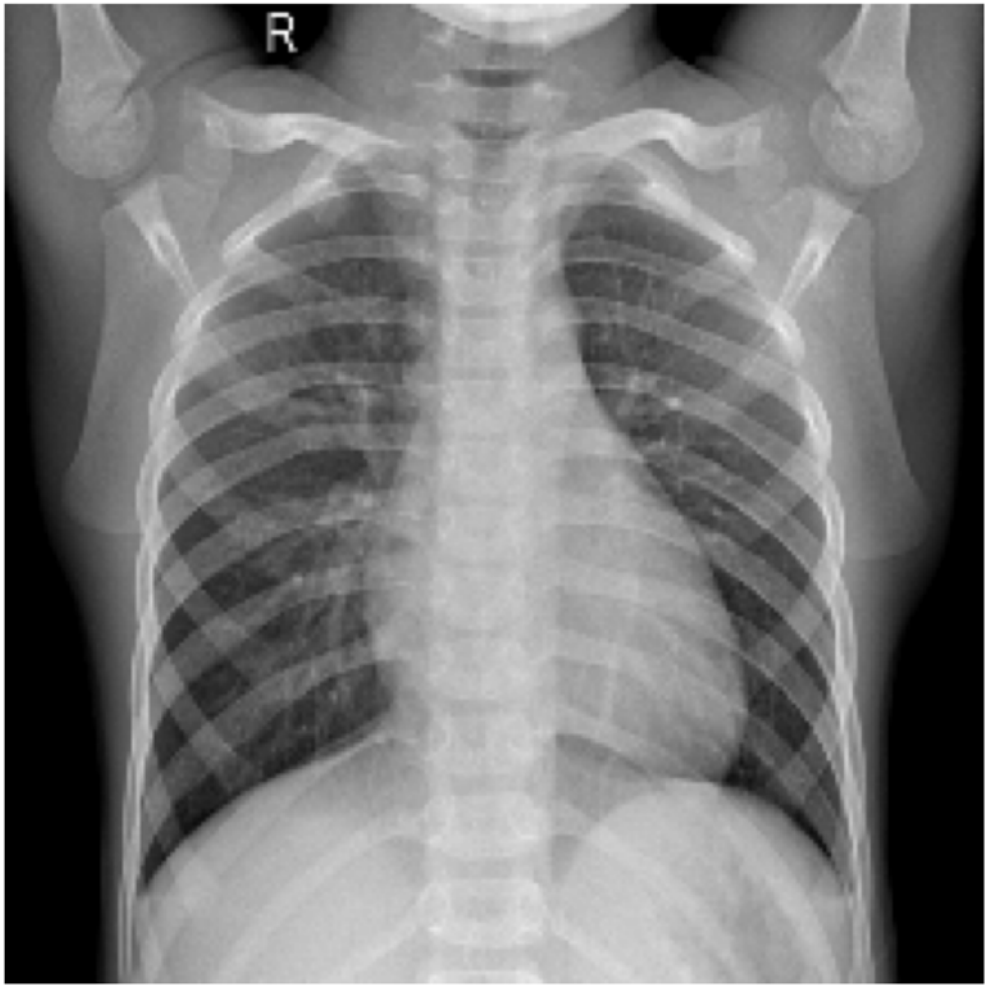



```
imagePatches2 = glob.glob(BASE_DIR +'dataset/Train/Class2/*.png', recursive=True)
for filename in imagePatches2[0:10]:
    print(filename)
```

```
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (108).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (1066).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (1071).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (1067).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (1063).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (107).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (1069).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (1062).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (1070).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (1068).png
```

```
image_name2 = "/content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (132).png" #Image to be used as que
def plotImage(image_location):
    image = cv2.imread(image_name2)
    image = cv2.resize(image, (224,224))
```

```
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)); plt.axis('off')
    return
plotImage(image_name2)
```

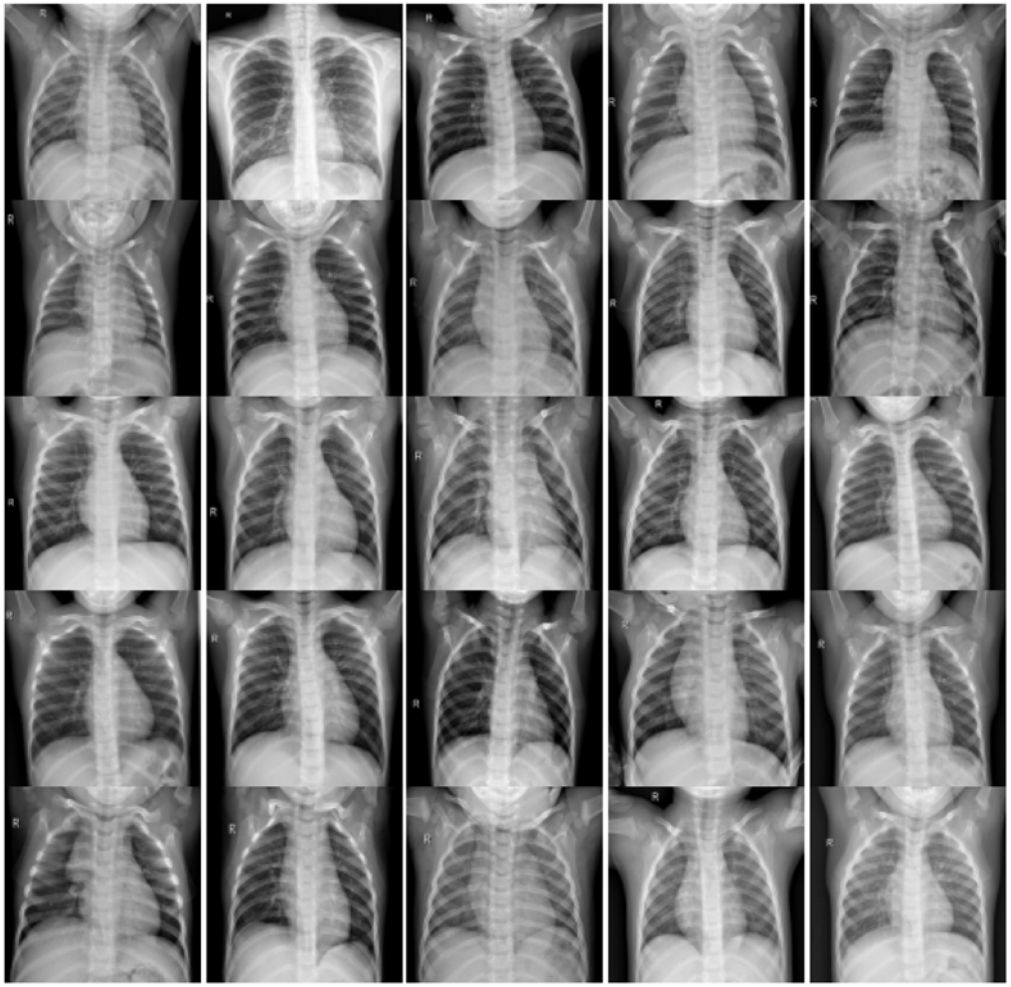



```
image_name2 = "/content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (132).png"
image = cv2.imread(image_name2)
print(image.shape)
```

```
(1024, 1024, 3)
```

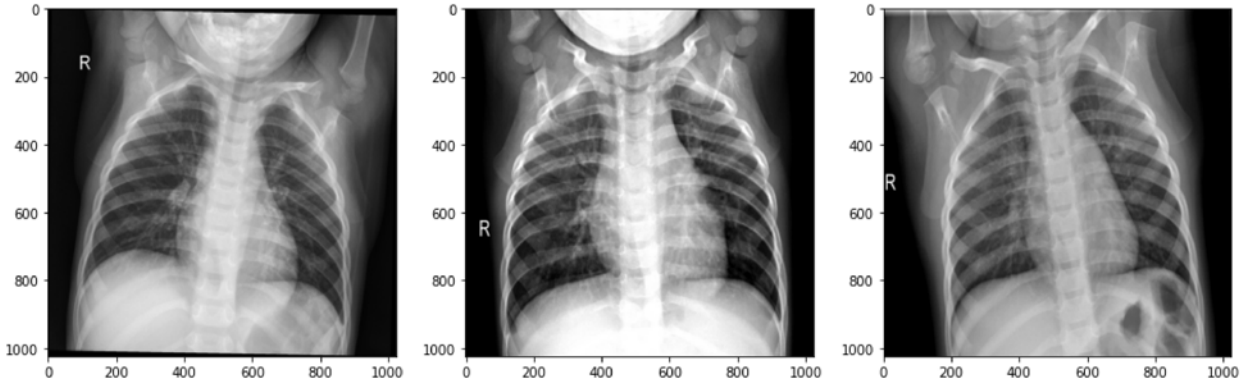
```
# Plot Multiple Images
```

```
bunchOfImages2 = imagePathes2
i_ = 0
plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.subplots_adjust(wspace=0, hspace=0)
for l in bunchOfImages2[:25]:
    im = cv2.imread(l)
    im = cv2.resize(im, (224, 224))
    plt.subplot(5, 5, i_+1) #.set_title(l)
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB)); plt.axis('off')
    i_ += 1
```



```
def randomImages(a):
    r = random.sample(a, 4)
```

```
        plt.figure(figsize=(16,16))
        plt.subplot(131)
        plt.imshow(cv2.imread(r[0]))
        plt.subplot(132)
        plt.imshow(cv2.imread(r[1]))
        plt.subplot(133)
        plt.imshow(cv2.imread(r[2]));
    randomImages(imagePatches2)
```




```
def get_data(folder):
    """
    Load the data and labels from the given folder.
    """
    X = []
    y = []

    for rayx_type in os.listdir(folder):
        if not rayx_type.startswith('Class3'):
            if rayx_type in ['Class1']:
                label = '0'
            else:
                label = '1'
            for image_filename in os.listdir(folder + rayx_type):
                img_file = cv2.imread(folder + rayx_type + '/' + image_filename)
                if img_file is not None:
                    image = cv2.resize(img_file, (320, 240))
```

```
                    image = cv2.resize(img_file, (320, 240))
                    img_arr = np.asarray(image)
                    X.append(img_arr)
                    y.append(label)
        X = np.asarray(X)
        y = np.asarray(y)
        return X,y


X_train, y_train = get_data(BASE_DIR + 'dataset/Train/')
X_test, y_test = get_data(BASE_DIR + 'dataset/Test/')

encoder = LabelEncoder()
encoder.fit(y_train)
y_train = encoder.transform(y_train)
y_test = encoder.transform(y_test)


def get_model():
    model = Sequential()
    model.add(Lambda(lambda x: x * 1./255., input_shape=(240, 320, 3), output_shape=(240, 320, 3)))
    model.add(Conv2D(32, (3, 3), input_shape=(240, 320, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(32, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(64, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
    model.add(Dense(128))
    model.add(Activation('relu'))
    model.add(Dropout(0.7))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))

    model.compile(loss='binary_crossentropy',
                  optimizer='rmsprop',
                  metrics=['accuracy'])

    return model
```

```
    return model

model = get_model()
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda (Lambda)              (None, 240, 320, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 238, 318, 32)      896
_________________________________________________________________
activation (Activation)      (None, 238, 318, 32)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 119, 159, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 117, 157, 32)      9248
_________________________________________________________________
activation_1 (Activation)    (None, 117, 157, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 58, 78, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 56, 76, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 56, 76, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 38, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 68096)             0
_________________________________________________________________
dense (Dense)                (None, 128)               8716416
_________________________________________________________________
activation_3 (Activation)    (None, 128)               0
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
_________________________________________________________________
activation_4 (Activation)    (None, 1)                 0
=================================================================
Total params: 8,745,185
Trainable params: 8,745,185
Non-trainable params: 0
_________________________________________________________________
None
```

Double-click (or enter) to edit

```
model = get_model()

# fits the model on batches
history = model.fit(
    X_train,
    y_train,
    validation_split=0.2,
    epochs=epochs,
    shuffle=True,
    batch_size=batch_size)
```

```
Epoch 1/10
32/32 [==============================] - 92s 3s/step - loss: 1.7012 - accuracy: 0.7493 - val_loss: 0.0540 - val_accuracy: 0.980(
Epoch 2/10
32/32 [==============================] - 90s 3s/step - loss: 0.2119 - accuracy: 0.9178 - val_loss: 0.1926 - val_accuracy: 0.916(
Epoch 3/10
32/32 [==============================] - 89s 3s/step - loss: 0.2116 - accuracy: 0.9083 - val_loss: 0.0272 - val_accuracy: 0.992(
Epoch 4/10
32/32 [==============================] - 89s 3s/step - loss: 0.4107 - accuracy: 0.9225 - val_loss: 0.0937 - val_accuracy: 0.968(
Epoch 5/10
32/32 [==============================] - 89s 3s/step - loss: 0.0936 - accuracy: 0.9715 - val_loss: 0.0642 - val_accuracy: 0.984(
Epoch 6/10
32/32 [==============================] - 89s 3s/step - loss: 0.0926 - accuracy: 0.9617 - val_loss: 0.0134 - val_accuracy: 0.996(
Epoch 7/10
32/32 [==============================] - 89s 3s/step - loss: 0.0567 - accuracy: 0.9794 - val_loss: 0.0468 - val_accuracy: 0.984(
Epoch 8/10
32/32 [==============================] - 89s 3s/step - loss: 0.0584 - accuracy: 0.9808 - val_loss: 0.0926 - val_accuracy: 0.976(
Epoch 9/10
32/32 [==============================] - 89s 3s/step - loss: 0.0702 - accuracy: 0.9807 - val_loss: 0.0284 - val_accuracy: 0.996(
Epoch 10/10
32/32 [==============================] - 90s 3s/step - loss: 0.0290 - accuracy: 0.9941 - val_loss: 0.0315 - val_accuracy: 0.996(
```

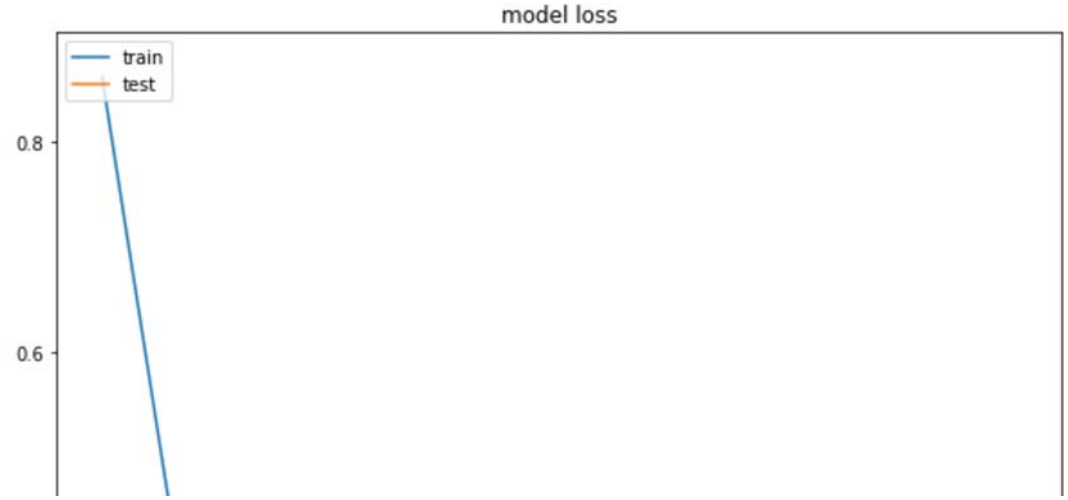
```
from sklearn.metrics import accuracy_score

print('Predicting on test data')
y_pred = np.rint(model.predict(X_test))

print(accuracy_score(y_test, y_pred))
```

```
Predicting on test data
```

```
0.9678456591639871
```

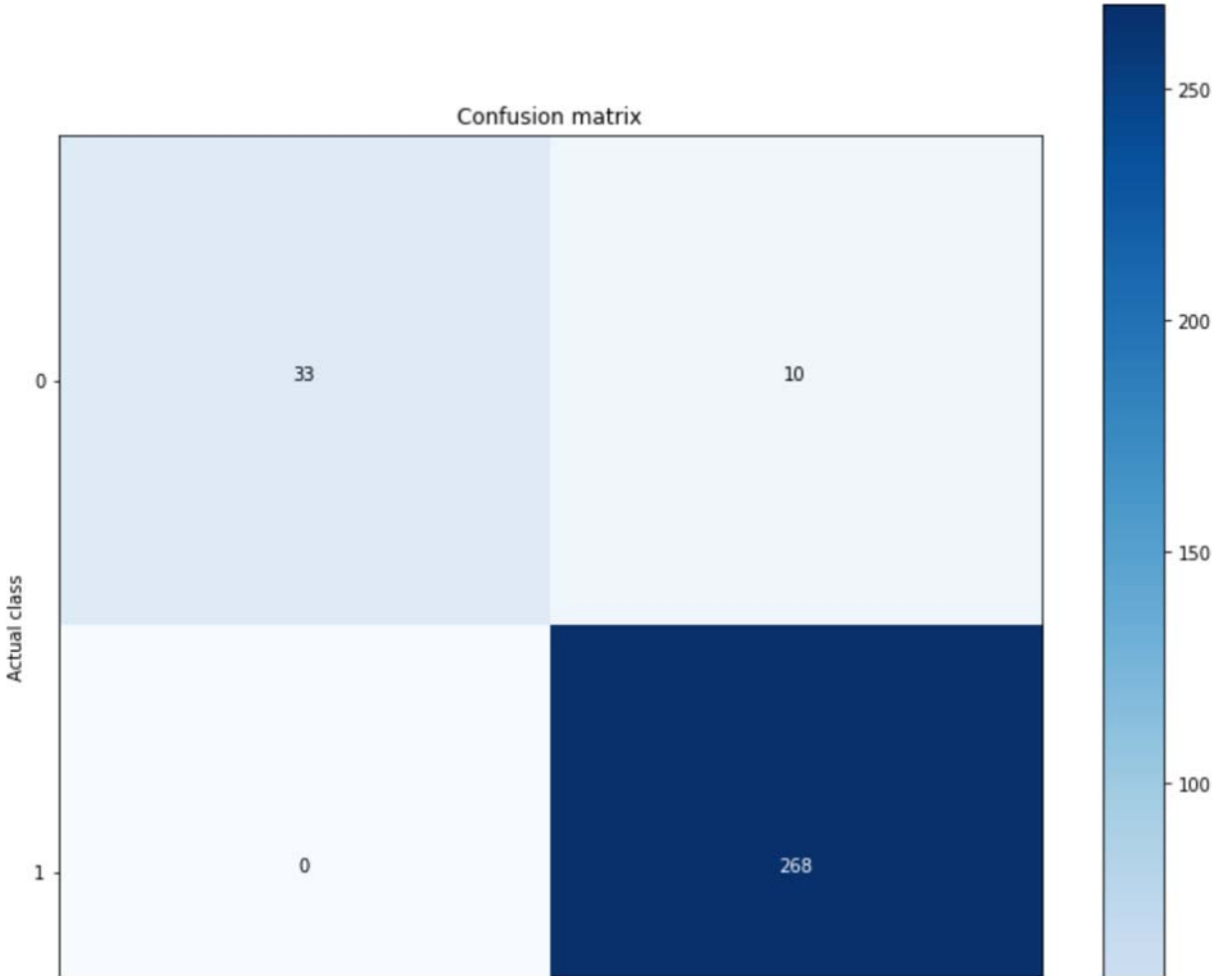
```
def plot_learning_curve(history):
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('./accuracy_curve.png')
    plt.clf()
    # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('./loss_curve.png')

plot_learning_curve(history)
from sklearn.metrics import confusion_matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[ 33  10]
 [  0 268]]
```




```
import matplotlib.pyplot as plt
#Note, this code is taken from the SKLEARN website.
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
```

```
    plt.tight_layout()
    plt.ylabel('Actual class')
    plt.xlabel('Predicted class')


from collections import Counter
from sklearn.metrics import confusion_matrix
import itertools
# compute the confusion matrix
confusion_mtx = confusion_matrix(y_test, y_pred)
# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(2))
```


Confusion matrix
Actual class
0
1
33
10
0
268
250
200
150
100

DEEP LEARNING – MULTI-CLASS

CODE SUMMARY: in this code we are following a similar format as we did for the case of machine learning models. We start a bit differently, though, by setting up colab (which we didn't do for ML) and importing relevant libraries. Then, we define our training parameters. After that we load and visualize our data for each class. We use the get_data function in ourder to build our training and testing arrays. We then define our CNN and proceed to fit the model to our data. After that, we plot performance metrics with particular emphasis on the confusion matrix. In order to build the confusion matrix, we had to do some additional of the predictions vector, *y_pred* (compared to the binary classification case).

```
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)
import os
os.chdir("/content/gdrive/My Drive/Colab Notebooks/Final_Project")
```

```
Mounted at /content/gdrive
```

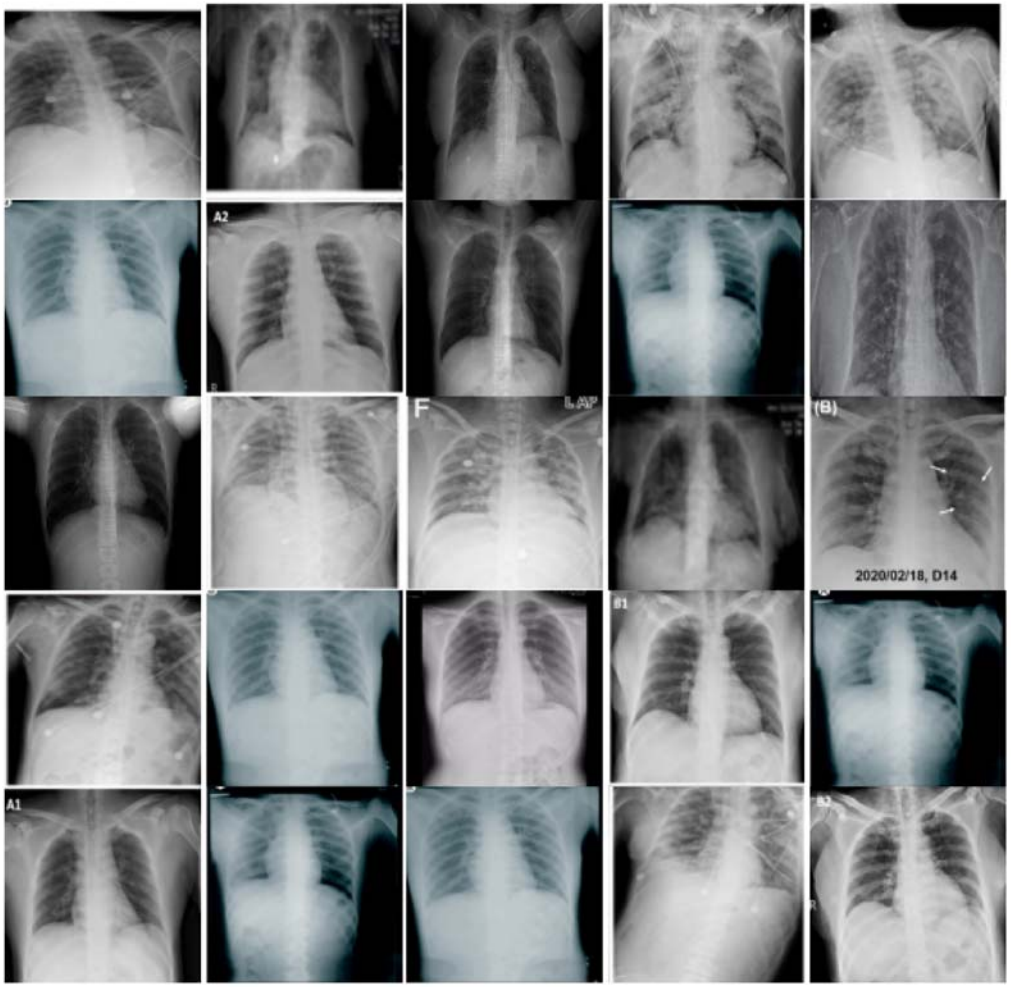
```
#Basic
import numpy as np
import cv2
import scipy
import os
from keras.utils import np_utils
import random

#Pre-processing
import glob
from keras.preprocessing.image import ImageDataGenerator
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from time import time

#Model
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, Lambda
from keras.layers import Dense
from keras.utils import np_utils

#Visualizers
%matplotlib inline
import matplotlib.pyplot as plt
```

```
epochs = 10
BASE_DIR = "/content/gdrive/My Drive/Colab Notebooks/Final_Project/"
batch_size = 32


imagePatches = glob.glob(BASE_DIR +'dataset/Train/Class1/*.png', recursive=True)
for filename in imagePatches[0:10]:
    print(filename)
```

```
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (125).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (112).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (106).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (109).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (124).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (101).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (120).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (107).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (102).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19 (1).png
```

```
image_name = "/content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19(144).png" #Image to be used as que
def plotImage(image_location):
    image = cv2.imread(image_name)
    image = cv2.resize(image, (224,224))

    print(image.shape)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)); plt.axis('off')
    return
plotImage(image_name)
```

```
# Plot Multiple Images
bunchOfImages = imagePatches
i_ = 0
plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.subplots_adjust(wspace=0, hspace=0)
for l in bunchOfImages[:25]:
    im = cv2.imread(l)
    im = cv2.resize(im, (224, 224))
    plt.subplot(5, 5, i_+1) #.set_title(l)
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB)); plt.axis('off')
    i_ += 1
```



```
imagePatches2 = glob.glob(BASE_DIR +'dataset/Train/Class2/*.png', recursive=True)
for filename in imagePatches2[0:10]:
    print(filename)
```

```
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (108).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (1066).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (1071).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (1067).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (1063).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (107).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (1069).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (1062).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (1070).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (1068).png
```

```
image_name2 = "/content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (132).png" #Image to be used as que
def plotImage(image_location):
    image = cv2.imread(image_name2)
    image = cv2.resize(image, (224,224))
    print(image.shape)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)); plt.axis('off')
    return
plotImage(image_name2)
```

```
(224, 224, 3)
```

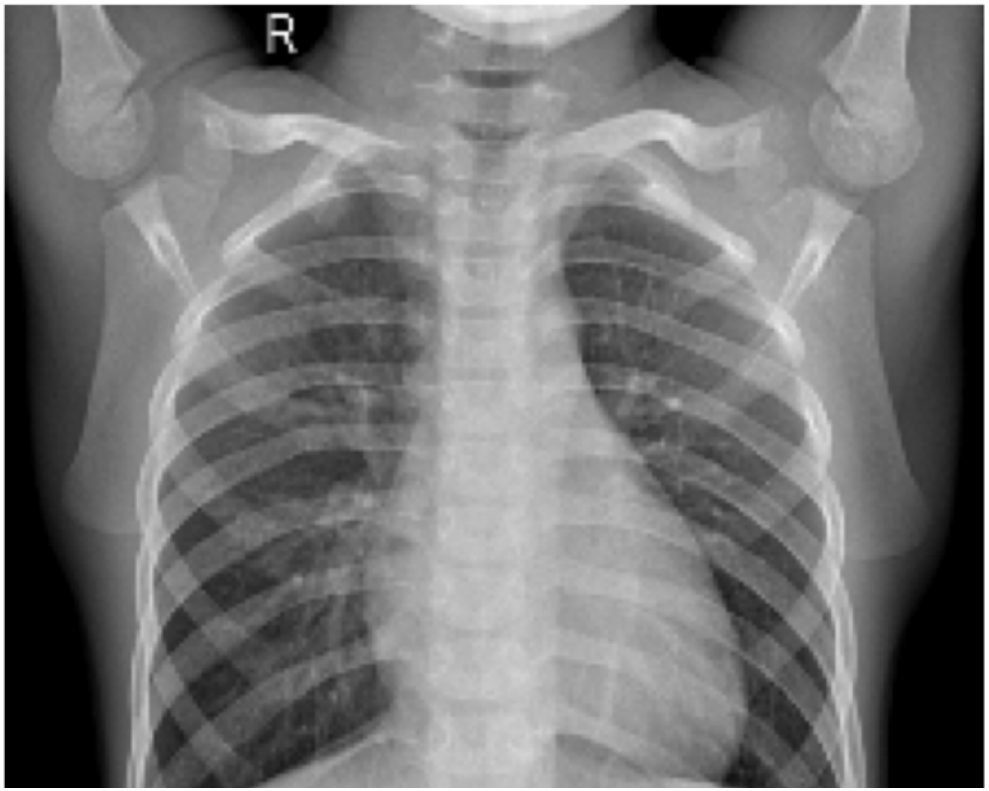

```
# Plot Multiple Images
bunchOfImages2 = imagePatches2
i_ = 0
plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.subplots_adjust(wspace=0, hspace=0)
for l in bunchOfImages2[:25]:
    im = cv2.imread(l)
    im = cv2.resize(im, (224, 224))
    plt.subplot(5, 5, i_+1) #.set_title(l)
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB)); plt.axis('off')
    i_ += 1
```

```
imagePatches3 = glob.glob(BASE_DIR +'dataset/Train/Class3/*.png', recursive=True)
for filename in imagePatches3[0:10]:
    print(filename)
```

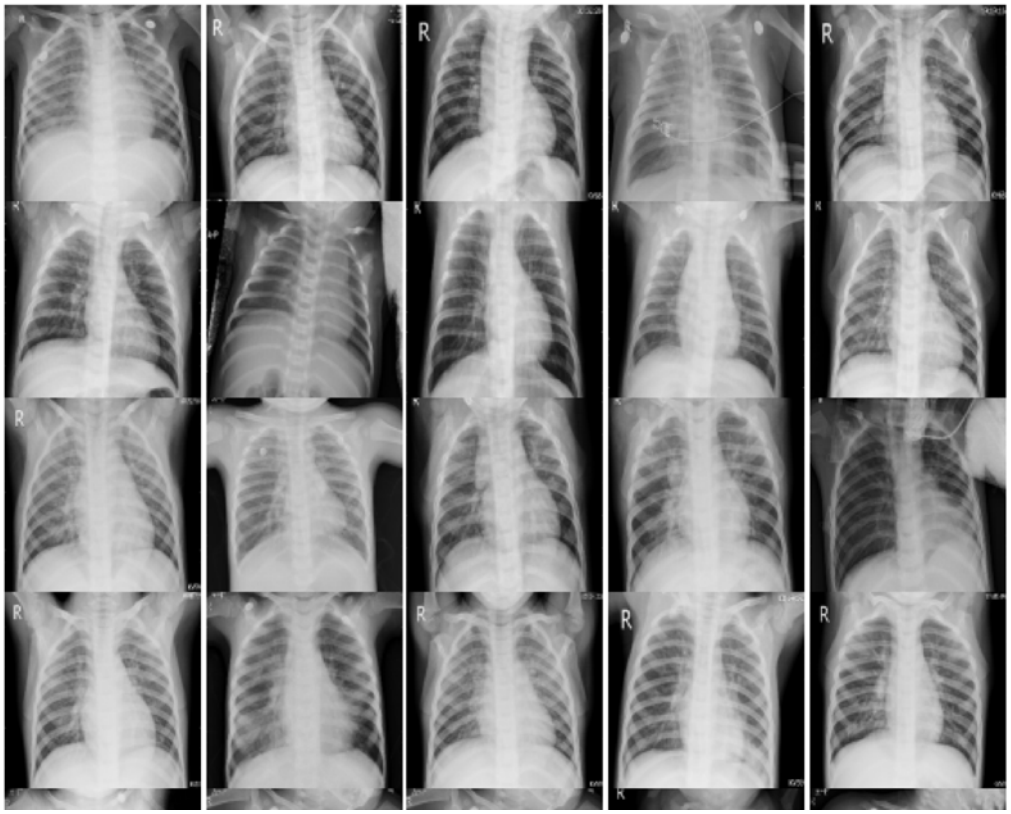
```
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class3/Viral Pneumonia (1061).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class3/Viral Pneumonia (1063).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class3/Viral Pneumonia (1060).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class3/Viral Pneumonia (1052).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class3/Viral Pneumonia (1059).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class3/Viral Pneumonia (1071).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class3/Viral Pneumonia (1065).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class3/Viral Pneumonia (1064).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class3/Viral Pneumonia (106).png
 /content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class3/Viral Pneumonia (1069).png
```

```
image_name3 = "/content/gdrive/My Drive/Colab Notebooks/Final_Project/dataset/Train/Class3/Viral Pneumonia (1).png" #Image to be used
def plotImage(image_location):
    image = cv2.imread(image_name2)
    image = cv2.resize(image, (224,224))
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)); plt.axis('off')
    print(image.shape)
    return
plotImage(image_name2)
```

```
(224, 224, 3)
```




```
# Plot Multiple Images
bunchOfImages3 = imagePatches3
i_ = 0
plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.subplots_adjust(wspace=0, hspace=0)
for l in bunchOfImages3[:25]:
    im = cv2.imread(l)
    im = cv2.resize(im, (224, 224))
    plt.subplot(5, 5, i_+1) #.set_title(l)
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB)); plt.axis('off')
    i_ += 1
```



```
def get_data(folder):
    """
    Load the data and labels from the given folder.
    """
    X = []
    y = []

    for rayx_type in os.listdir(folder):
        if not rayx_type.startswith('.'):
            if rayx_type in ['Class1']:
                label = '0'
            else:
                if rayx_type in ['Class2']:
                    label = '1'
                else:
                    label ='2'
            for image_filename in os.listdir(folder + rayx_type):
                img_file = cv2.imread(folder + rayx_type + '/' + image_filename)
```

```
                if img_file is not None:
                    # Downsample the image to 224, 224, 3
                    #img_file = scipy.misc.imresize(arr=img_file, size=(120, 160, 3))
                    #img_arr = np.asarray(img_file)
                    image = cv2.resize(img_file, (320, 240))
                    img_arr = np.asarray(image)
                    X.append(img_arr)
                    y.append(label)
        X = np.asarray(X)
        y = np.asarray(y)
        return X,y


from keras.utils import to_categorical

X_train, y_train = get_data(BASE_DIR + 'dataset/Train/')
X_test, y_test = get_data(BASE_DIR + 'dataset/Test/')

encoder = LabelEncoder()
encoder.fit(y_train)
y_train = encoder.transform(y_train)
y_test = encoder.transform(y_test)


def get_model():
    model = Sequential()
    model.add(Lambda(lambda x: x * 1./255., input_shape=(240, 320, 3), output_shape=(240, 320, 3)))
    model.add(Conv2D(32, (3, 3), input_shape=(240, 320, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(32, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(64, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
    model.add(Dense(128))
    model.add(Activation('relu'))
    model.add(Dropout(0.7))
    model.add(Dense(3))
```

```
    model.add(Activation('sigmoid'))

    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer='sgd',
                  metrics=['accuracy'])

    return model


model = get_model()
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda (Lambda)              (None, 240, 320, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 238, 318, 32)      896
_________________________________________________________________
activation (Activation)      (None, 238, 318, 32)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 119, 159, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 117, 157, 32)      9248
_________________________________________________________________
activation_1 (Activation)    (None, 117, 157, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 58, 78, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 56, 76, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 56, 76, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 38, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 68096)             0
_________________________________________________________________
dense (Dense)                (None, 128)               8716416
_________________________________________________________________
activation_3 (Activation)    (None, 128)               0
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 387
_________________________________________________________________
activation_4 (Activation)    (None, 3)                 0
```

```
=================================================================
Total params: 8,745,443
Trainable params: 8,745,443
Non-trainable params: 0
_________________________________________________________________
None
```

```
model = get_model()

# fits the model on batches
history = model.fit(
    X_train,
    y_train,
    validation_split=0.2,
    epochs=epochs,
    shuffle=True,
    batch_size=batch_size)
```

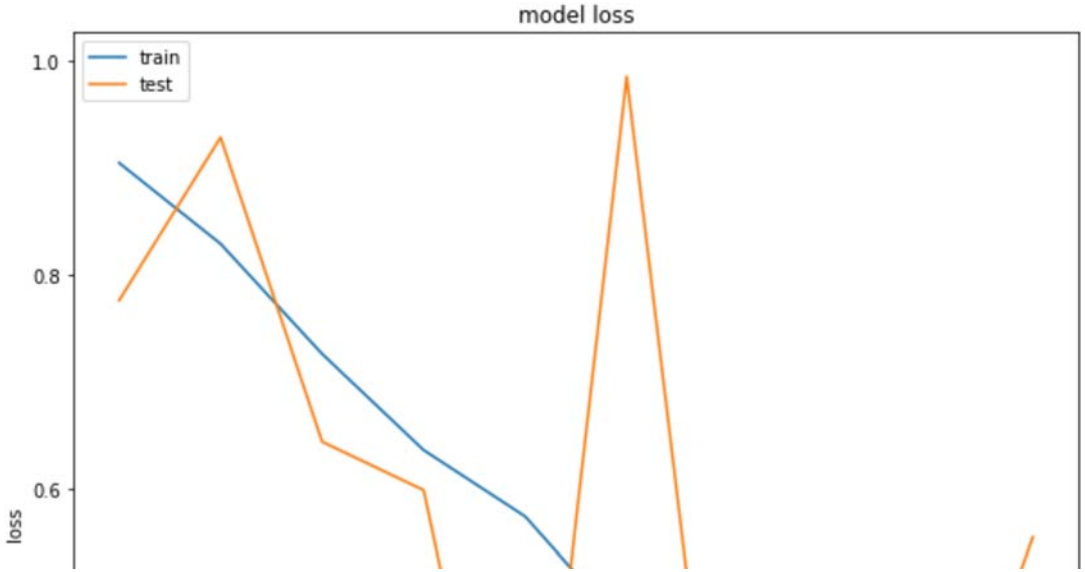
```
Epoch 1/10
59/59 [==============================] - 166s 3s/step - loss: 0.9197 - accuracy: 0.5319 - val_loss: 0.7757 - val_accuracy: 0.238
Epoch 2/10
59/59 [==============================] - 168s 3s/step - loss: 0.8568 - accuracy: 0.5735 - val_loss: 0.9283 - val_accuracy: 0.055
Epoch 3/10
59/59 [==============================] - 166s 3s/step - loss: 0.7481 - accuracy: 0.6223 - val_loss: 0.6435 - val_accuracy: 0.808
Epoch 4/10
59/59 [==============================] - 166s 3s/step - loss: 0.6696 - accuracy: 0.6843 - val_loss: 0.5983 - val_accuracy: 0.759
Epoch 5/10
59/59 [==============================] - 168s 3s/step - loss: 0.6120 - accuracy: 0.7537 - val_loss: 0.1534 - val_accuracy: 0.980
Epoch 6/10
59/59 [==============================] - 166s 3s/step - loss: 0.5060 - accuracy: 0.7943 - val_loss: 0.9852 - val_accuracy: 0.494
Epoch 7/10
59/59 [==============================] - 166s 3s/step - loss: 0.4404 - accuracy: 0.8290 - val_loss: 0.1977 - val_accuracy: 0.965
Epoch 8/10
59/59 [==============================] - 168s 3s/step - loss: 0.4034 - accuracy: 0.8488 - val_loss: 0.2060 - val_accuracy: 0.937
Epoch 9/10
59/59 [==============================] - 169s 3s/step - loss: 0.4160 - accuracy: 0.8444 - val_loss: 0.2907 - val_accuracy: 0.914
Epoch 10/10
59/59 [==============================] - 168s 3s/step - loss: 0.3461 - accuracy: 0.8622 - val_loss: 0.5543 - val_accuracy: 0.791
```

```
from sklearn.metrics import accuracy_score

print('Predicting on test data')
y_pred = np.argmax(model.predict(X_test), axis=1)
print(accuracy_score(y_test, y_pred))
```

```
Predicting on test data
0.7913793103448276
```

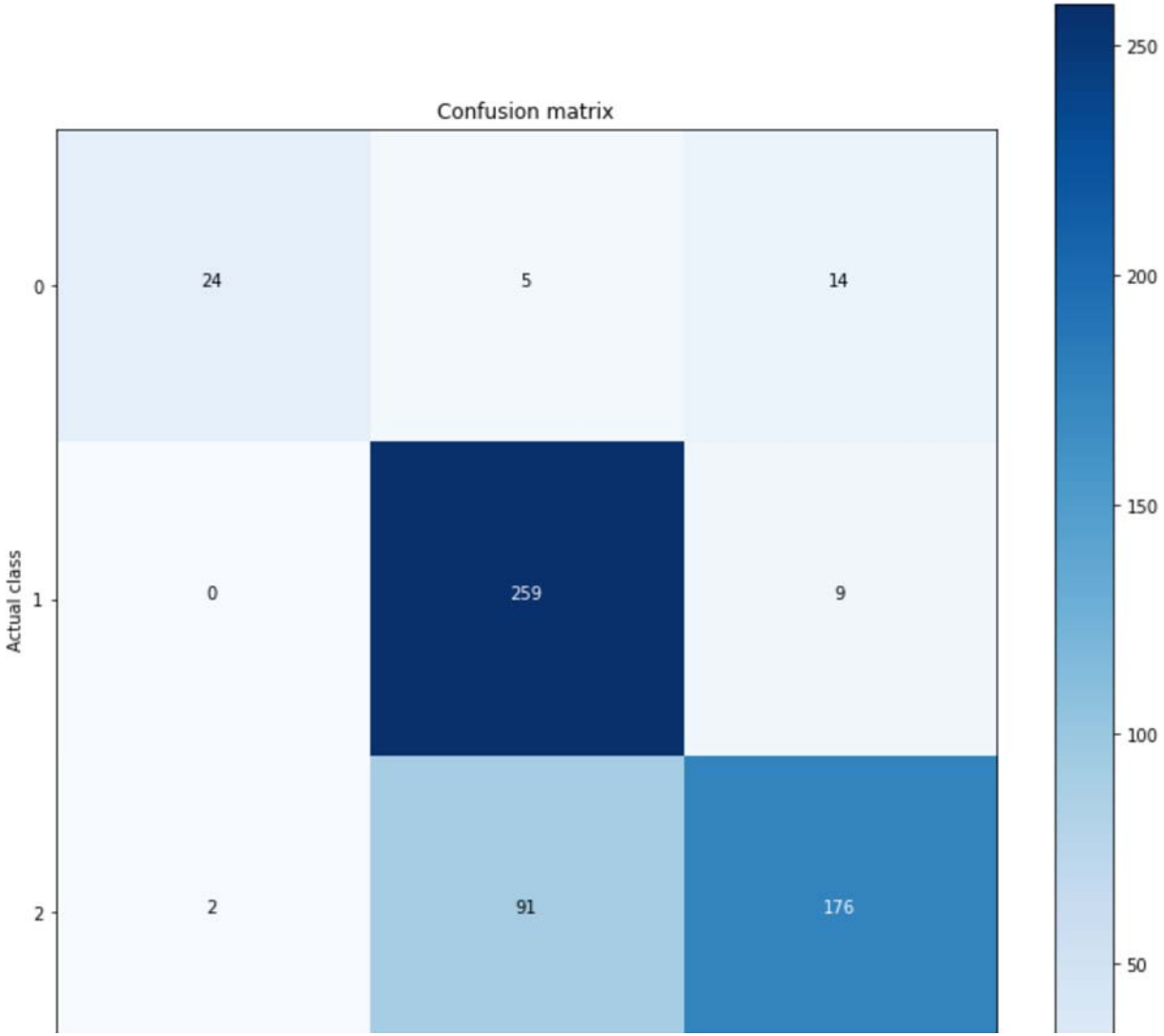
```
def plot_learning_curve(history):
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('./accuracy_curve.png')
    plt.clf()
    # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('./loss_curve.png')

plot_learning_curve(history)
from sklearn.metrics import confusion_matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[ 24   5  14]
 [  0 259   9]
 [  2  91 176]]
```




```
import matplotlib.pyplot as plt
#Note, this code is taken from the SKLEARN website.
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
```

```
                horizontalalignment="center",
                color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Actual class')
    plt.xlabel('Predicted class')


from collections import Counter
from sklearn.metrics import confusion_matrix
import itertools
# compute the confusion matrix
confusion_mtx = confusion_matrix(y_test, y_pred)
# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(3))
```


Confusion matrix
Actual class
0
1
2
24
5
14
0
259
9
2
91
176
250
200
150
100
50

MACHINE LEARNING – BINARY

In [1]:

```
#Basic
import numpy as np
import cv2
import scipy
import os
from keras.utils import np_utils
import random

#Pre-processing
import glob
from keras.preprocessing.image import ImageDataGenerator
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from time import time

#Metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc

#Classifiers
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier

#Visualizers
%matplotlib inline
import matplotlib.pyplot as plt
from yellowbrick.classifier import ConfusionMatrix
```

```
Using TensorFlow backend.
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:516: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depreca
ted; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:517: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depreca
ted; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:518: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depreca
ted; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:519: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depreca
ted; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:520: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depreca
ted; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depreca
ted; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:541: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:542: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:543: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:544: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:545: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:550: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:144: FutureWarning: The sklearn.metrics.classification module is  deprecated in version
0.22 and will be removed in version 0.24. The corresponding classes / functions should instead be imported from sklearn.metrics. Anything that cannot be imported f
rom sklearn.metrics is now part of the private API.
  warnings.warn(message, FutureWarning)
```

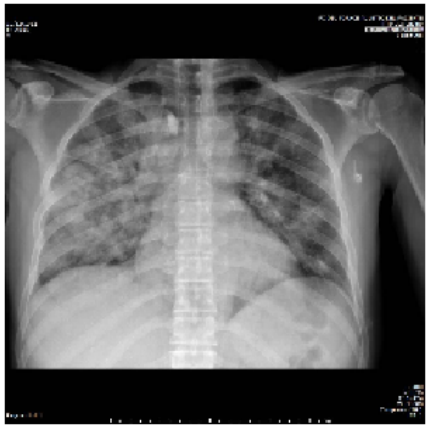
In [2]:
```
#from google.colab import drive
#drive.mount('/content/gdrive', force_remount=True)
#import os
#os.chdir("/content/gdrive/My Drive/Colab Notebooks/Final_Project")
```

In [3]:
```
BASE_DIR = 'C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/'
```

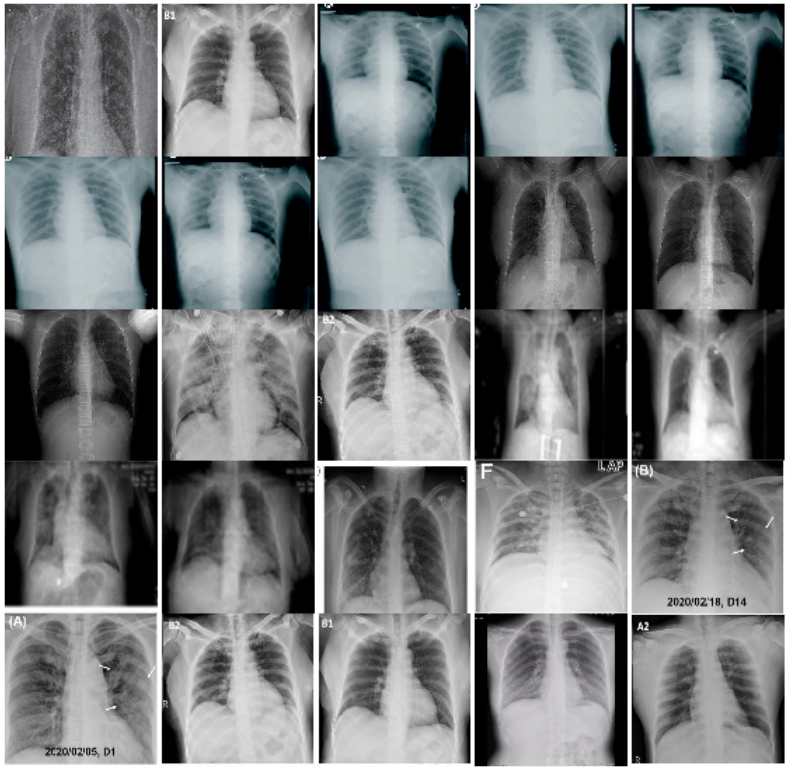
In [4]:
```
imagePatches = glob.glob(BASE_DIR +'dataset/Train/Class1/*.png', recursive=True)
for filename in imagePatches[0:10]:
    print(filename)
```

```
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (1).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (10).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (100).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (101).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (102).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (103).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (104).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (105).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (106).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (107).png
```

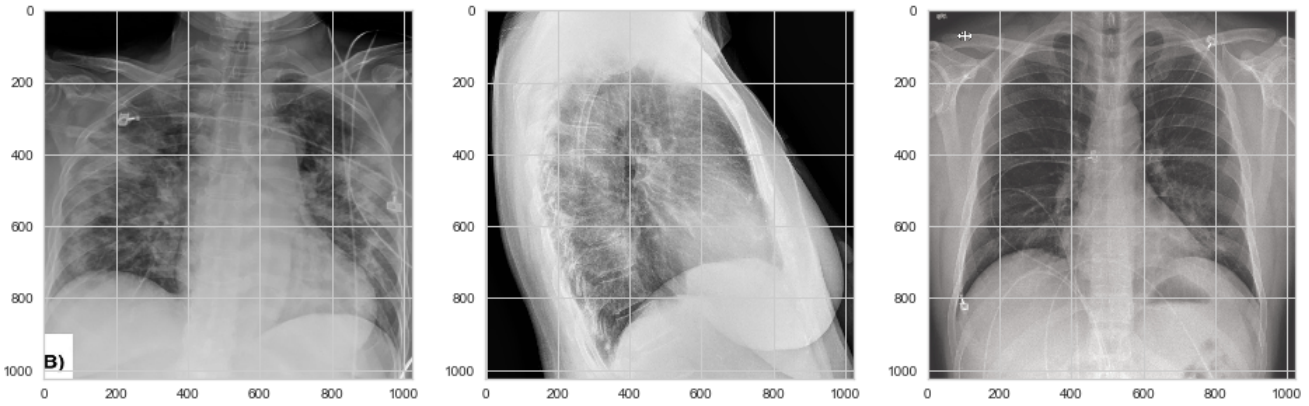
In [5]:
```
image_name = "C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1/COVID-19(144).png" #Image to be used as query
def plotImage(image_location):
    image = cv2.imread(image_name)
    image = cv2.resize(image, (224,224))
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)); plt.axis('off')
    return
plotImage(image_name)
```



In [6]:
```
# Plot Multiple Images
bunchOfImages = imagePatches
i_ = 0
plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.subplots_adjust(wspace=0, hspace=0)
for l in bunchOfImages[:25]:
    im = cv2.imread(l)
    im = cv2.resize(im, (224, 224))
    plt.subplot(5, 5, i_+1) #.set_title(l)
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB)); plt.axis('off')
    i_ += 1
```



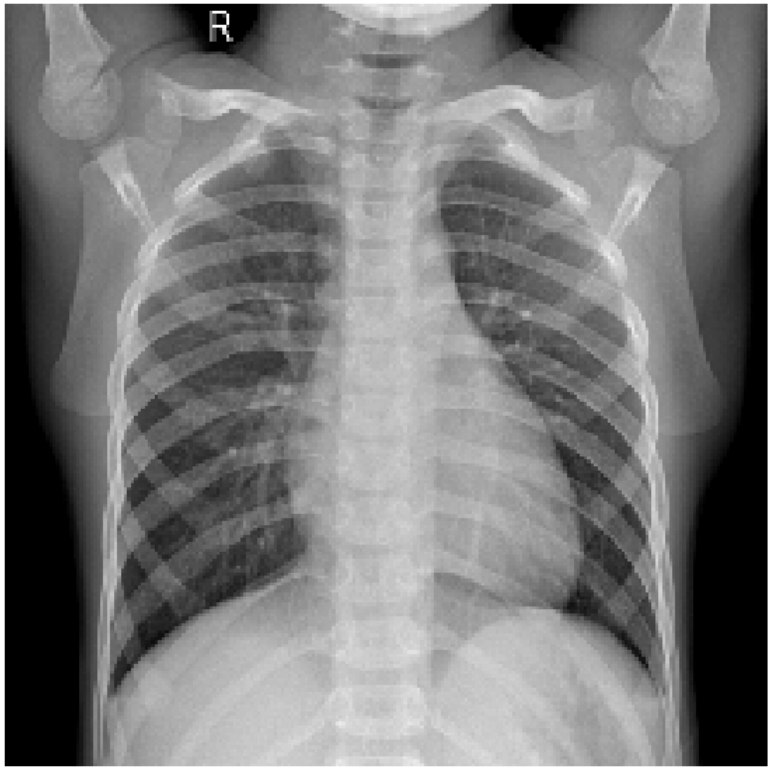
In [7]:
```
def randomImages(a):
    r = random.sample(a, 4)
    plt.figure(figsize=(16,16))
    plt.subplot(131)
    plt.imshow(cv2.imread(r[0]))
    plt.subplot(132)
    plt.imshow(cv2.imread(r[1]))
    plt.subplot(133)
    plt.imshow(cv2.imread(r[2]));
randomImages(imagePatches)
```



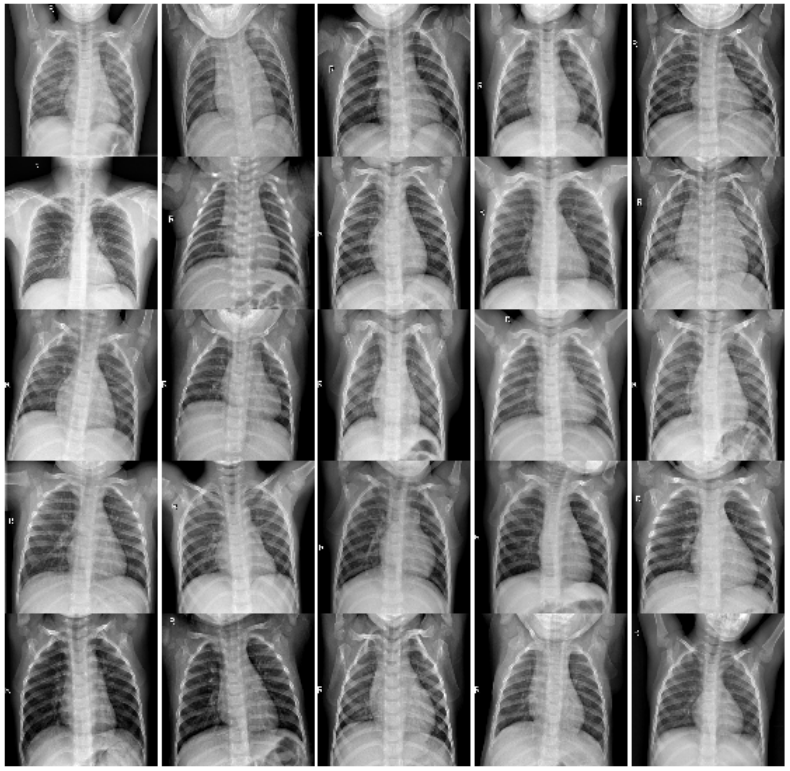
In [8]:
```
imagePatches2 = glob.glob(BASE_DIR +'dataset/Train/Class2/*.png', recursive=True)
for filename in imagePatches2[0:10]:
    print(filename)
```

```
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (1).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (10).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (100).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (1000).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (1001).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (1002).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (1003).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (1004).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (1005).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (1006).png
```

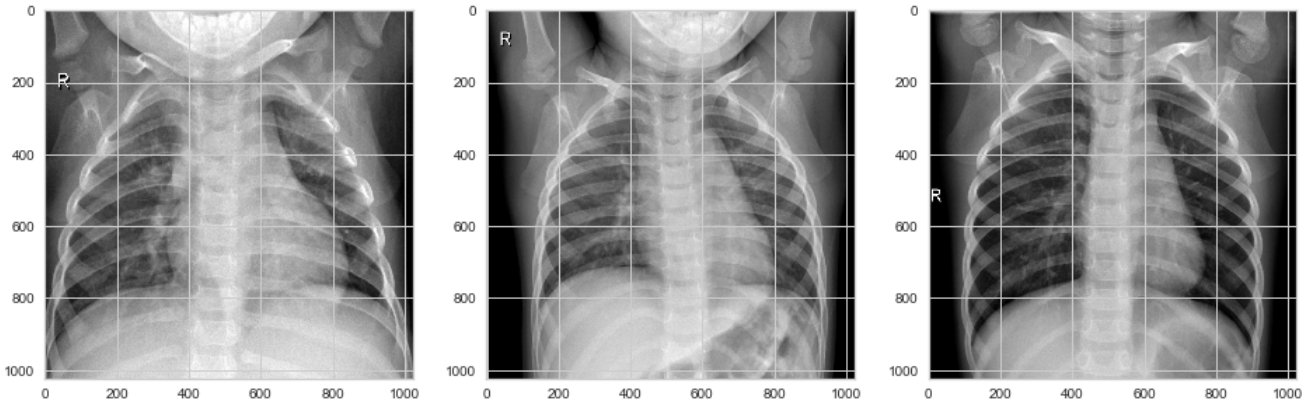
In [9]:
```
image_name2 = "C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2/NORMAL (132).png" #Image to be used as query
def plotImage(image_location):
    image = cv2.imread(image_name2)
    image = cv2.resize(image, (224,224))
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)); plt.axis('off')
    return
plotImage(image_name2)
```



In [10]:
```
# Plot Multiple Images
bunchOfImages2 = imagePatches2
i_ = 0
plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.subplots_adjust(wspace=0, hspace=0)
for l in bunchOfImages2[:25]:
    im = cv2.imread(l)
    im = cv2.resize(im, (224, 224))
    plt.subplot(5, 5, i_+1) #.set_title(l)
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB)); plt.axis('off')
    i_ += 1
```



In [11]:
```
def randomImages(a):
    r = random.sample(a, 4)
    plt.figure(figsize=(16,16))
    plt.subplot(131)
    plt.imshow(cv2.imread(r[0]))
    plt.subplot(132)
    plt.imshow(cv2.imread(r[1]))
    plt.subplot(133)
    plt.imshow(cv2.imread(r[2]));
randomImages(imagePatches2)
```



In [12]:
```
def get_data(folder):
    """
    Load the data and labels from the given folder.
    """
    X = []
    y = []

    for rayx_type in os.listdir(folder):
        if not rayx_type.startswith('Class3'):
            if rayx_type in ['Class1']:
                label = '0'
            else:
                if rayx_type in ['Class2']:
                    label = '1'
            for image_filename in os.listdir(folder + rayx_type):
                img_file = cv2.imread(folder + rayx_type + '/' + image_filename, 0)
                if img_file is not None:
                    # Downsample the image to 224, 224, 3
                    image = cv2.resize(img_file, (224, 224))
                    img_arr = np.asarray(image)
                    X.append(img_arr)
                    y.append(label)
    X = np.asarray(X)
    y = np.asarray(y)
    return X,y
```

In [13]:
```
classes = ["0", "1"]
```

In [14]:
```
X_train, y_train = get_data(BASE_DIR + 'dataset/Train/')
X_test, y_test = get_data(BASE_DIR + 'dataset/Test/')
```

In [15]:
```
#Covert 3D data arrays to 2D

nsamples, nx, ny = X_train.shape
nsamples2, nx2, ny2 = X_test.shape
X_train_new = X_train.reshape((nsamples,nx*ny))
X_test = X_test.reshape((nsamples2,nx2*ny2))

encoder = LabelEncoder()
encoder.fit(y_train)
y_train = encoder.transform(y_train)
y_test = encoder.transform(y_test)
```

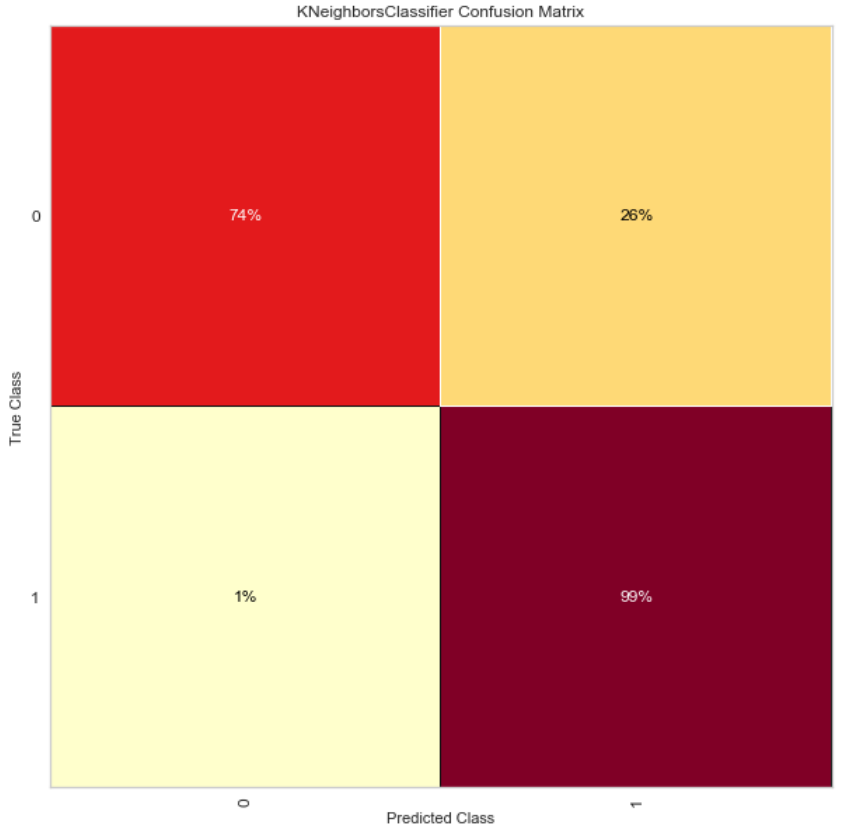
In [16]:
```
# select classifiers
classifiers=[
KNeighborsClassifier(),
LogisticRegression(random_state=0, multi_class='multinomial', solver='newton-cg'),
DecisionTreeClassifier(random_state=0),
RandomForestClassifier(),
SVC(),
MLPClassifier()]
```

In [17]:
```
#Write function for class-centric metrics

#Confusion matrix
def CM_viz():
    def CM(model,classes):
        visualizer = ConfusionMatrix(model, classes=classes,percent=True)
        visualizer.fit(X_train_new, y_train)  # Fit the visualizer and the model
        visualizer.score(X_test, y_test)  # Evaluate the model on the test data
        return visualizer.poof()
    for name in classifiers:
        ax = plt.subplot(1,1,1)
        CM(name,classes)
```

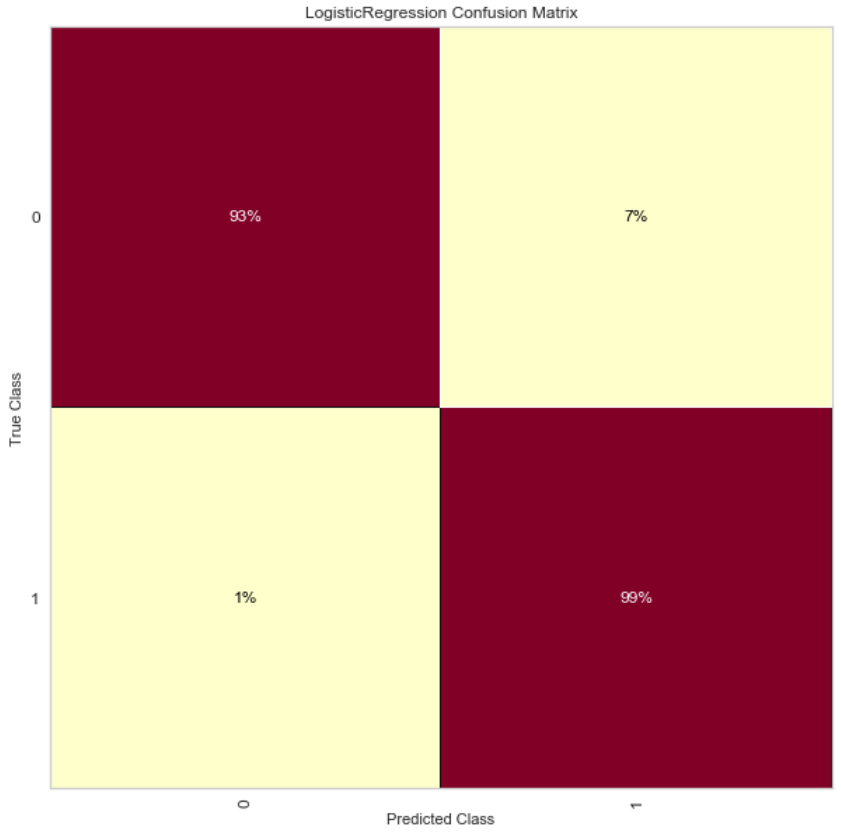
In [18]:
```
# Write function for aggregate metrics
def classifier_metrics():
    def metrics(model):
        #     model=model_name()
        model.fit(X_train_new, y_train)  # Fit the visualizer and the model
        y_pred = model.predict(X_test)
        try:
            y_prob = model.predict_proba(X_test)
            log_metric = log_loss(y_test,y_prob)
        except:
            y_prob = "Not probablistic"
            log_metric = 0
        else:
            y_pred = model.predict(X_test)

        score = accuracy_score(y_test, y_pred)
        print('accuracy_score: {0:.3f}'.format(score))

    for name in classifiers:
        print (str(name))
        metrics(name)
        print()
        print ("----------------------------------------------------")
```

In [19]:
```
#deploy visualization
visualization =[CM_viz()]
```

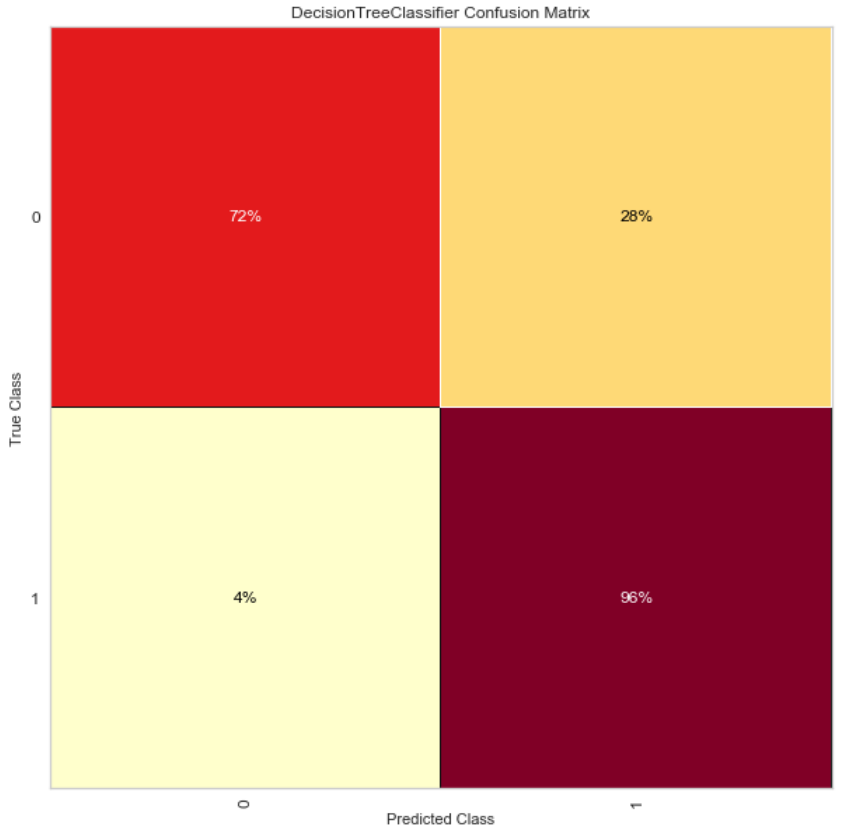
```
C:\Users\Oswaldo\anaconda3\lib\site-packages\sklearn\base.py:197: FutureWarning: From version 0.24, get_params will raise an AttributeError if a parameter cannot b
e retrieved as an instance attribute. Previously it would return None.
  FutureWarning)
```



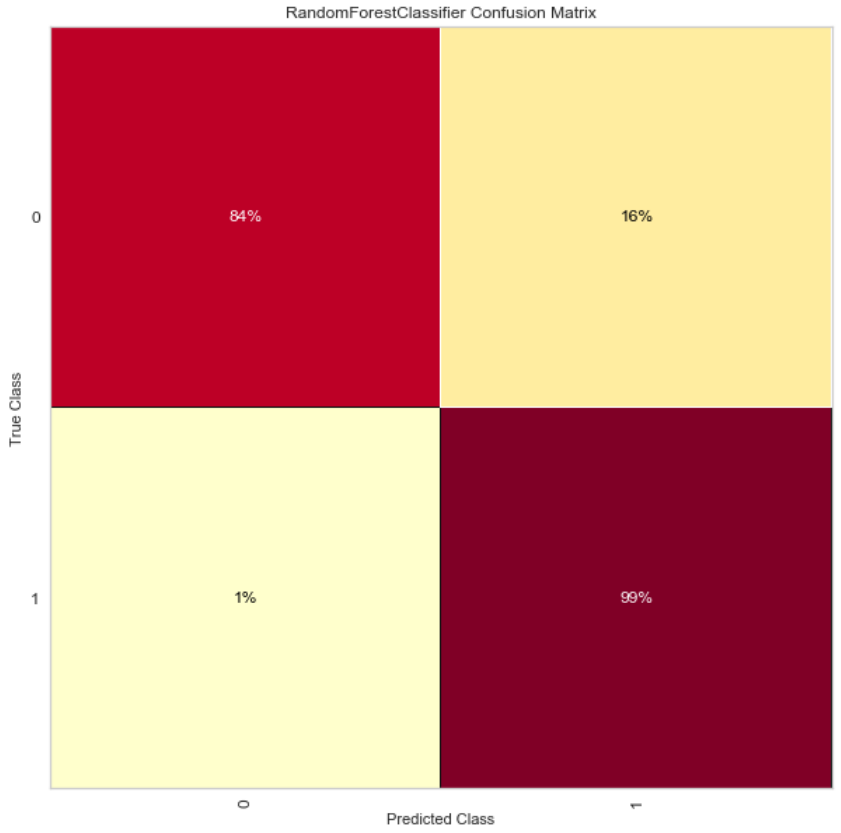

```
C:\Users\Oswaldo\anaconda3\lib\site-packages\sklearn\base.py:197: FutureWarning: From version 0.24, get_params will raise an AttributeError if a parameter cannot b
e retrieved as an instance attribute. Previously it would return None.
  FutureWarning)
```



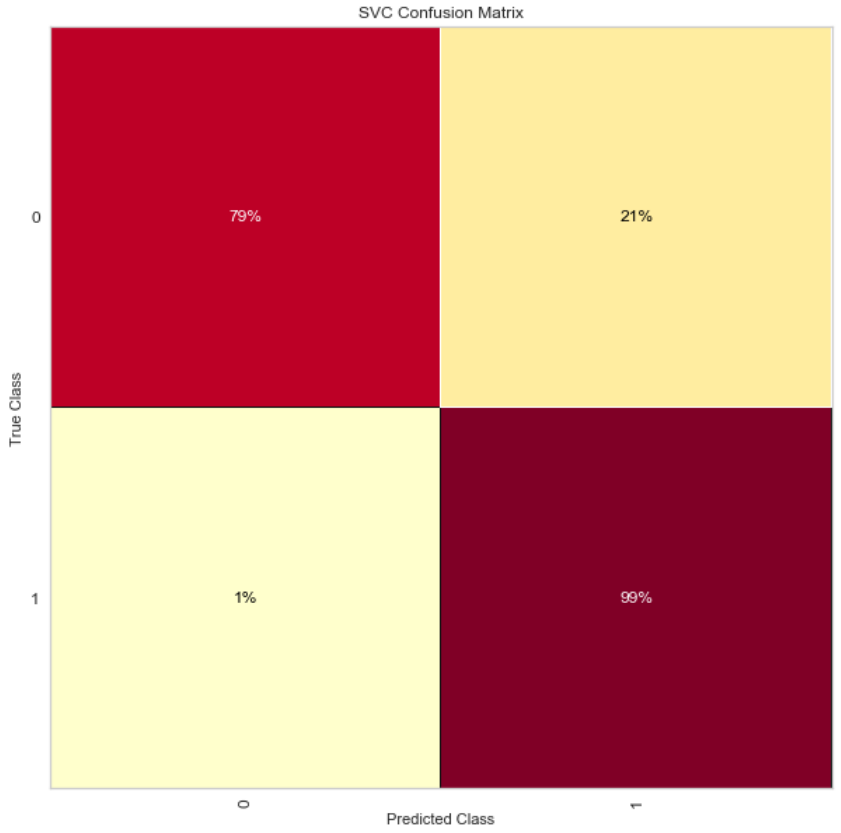

```
C:\Users\Oswaldo\anaconda3\lib\site-packages\sklearn\base.py:197: FutureWarning: From version 0.24, get_params will raise an AttributeError if a parameter cannot b
e retrieved as an instance attribute. Previously it would return None.
  FutureWarning)
```



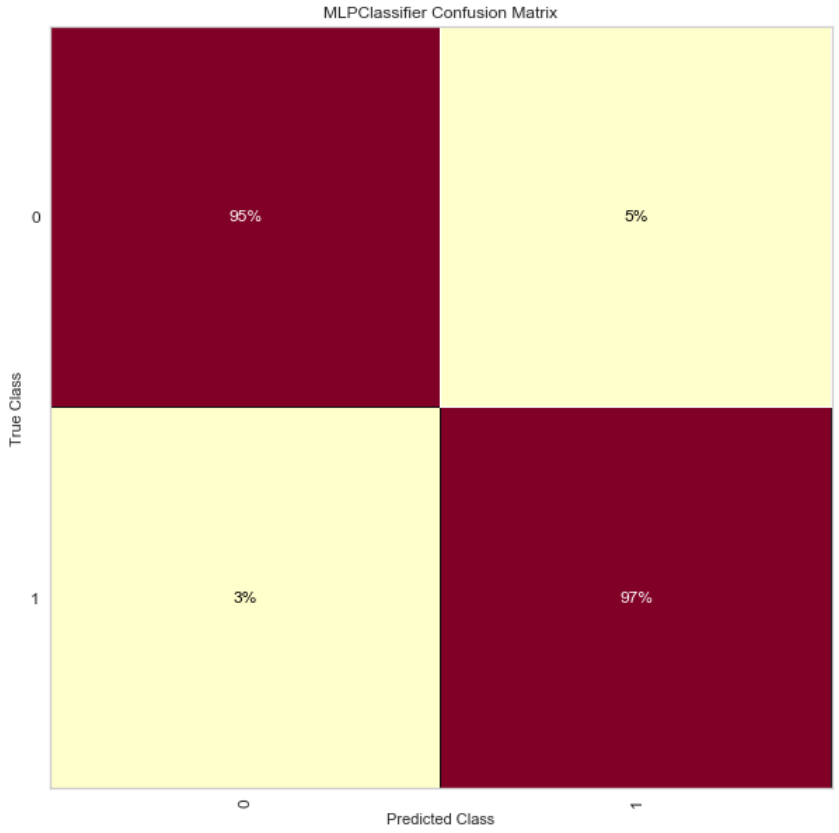

```
C:\Users\Oswaldo\anaconda3\lib\site-packages\sklearn\base.py:197: FutureWarning: From version 0.24, get_params will raise an AttributeError if a parameter cannot b
e retrieved as an instance attribute. Previously it would return None.
  FutureWarning)
```




```
C:\Users\Oswaldo\anaconda3\lib\site-packages\sklearn\base.py:197: FutureWarning: From version 0.24, get_params will raise an AttributeError if a parameter cannot b
e retrieved as an instance attribute. Previously it would return None.
  FutureWarning)
```




```
C:\Users\Oswaldo\anaconda3\lib\site-packages\sklearn\base.py:197: FutureWarning: From version 0.24, get_params will raise an AttributeError if a parameter cannot b
e retrieved as an instance attribute. Previously it would return None.
  FutureWarning)
```


MLPClassifier Confusion Matrix
0
1
True Class
95%
5%
3%
97%
0
1
Predicted Class

In [20]:
```
#Deploy aggregate metrics
classifier_metrics()
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
accuracy_score: 0.958

------------------------------------------------------
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='multinomial', n_jobs=None, penalty='l2',
                   random_state=0, solver='newton-cg', tol=0.0001, verbose=0,
                   warm_start=False)
accuracy_score: 0.981

------------------------------------------------------
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=0, splitter='best')
accuracy_score: 0.923

------------------------------------------------------
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
accuracy_score: 0.968

------------------------------------------------------
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
accuracy_score: 0.965

------------------------------------------------------
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=200,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=None, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
accuracy_score: 0.974

------------------------------------------------------
```

MACHINE LEARNING – MULTI-CLASS

Multi-label X-Ray Image Classification

In this notebook, x-ray photographs of COVID-19, Healthy, and Viral Pneumonia patients are used to train various Deep Learning classifiers to predict each of these conditions based on only such x-ray photographs.

The data for this project was taken from the publicly available COVID-19 Radiography Database (https://www.kaggle.com/tawsifurrahman/covid19-radiography-database (https://www.kaggle.com/tawsifurrahman/covid19-radiography-database)). These data were provided by Tawsifur Rahman at the University of Dhaka.

The data include 219 COVID-19 positive images, 1341 normal images and 1345 viral pneumonia images. From these totals, 80% of the images were selected for training and 20% were used for testing.

Classifiers from the sklearn library have been adopted. A non-exhaustive parameter optimization scheme was conducted for each classifier using GridSearchCV (also from sklearn). The best version of each classifier was trained and tested. To analyze the performance, several Yellowbrick visualization tools were employed. Additionally, several sklearn metrics were included to quantitatively characterize the performance of each model.

Load the relevant libraries

In [1]:

```
#Basic
import numpy as np
import cv2
import scipy
import os
from keras.utils import np_utils
import random
import seaborn as sns

#Pre-processing
import glob
from keras.preprocessing.image import ImageDataGenerator
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from time import time

#Parameter optimization
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import train_test_split

#Metrics
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import hamming_loss
from sklearn.metrics import log_loss
from sklearn.metrics import zero_one_loss
from sklearn.metrics import matthews_corrcoef

from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report
from sklearn.metrics import adjusted_rand_score
from sklearn.model_selection import cross_val_score

#Classifiers
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier

#Visualizers
%matplotlib inline
import matplotlib.pyplot as plt
from yellowbrick.classifier import ConfusionMatrix
from yellowbrick.classifier import ClassificationReport
from yellowbrick.classifier import ClassPredictionError
from yellowbrick.classifier import ConfusionMatrix
from yellowbrick.classifier import ROCAUC
from yellowbrick.classifier import PrecisionRecallCurve

#shush the warnings
import warnings
warnings.filterwarnings('ignore')
```

```
Using TensorFlow backend.
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:516: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depreca
ted; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:517: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depreca
ted; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:518: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depreca
ted; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:519: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depreca
ted; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:520: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depreca
ted; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depreca
ted; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:541: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:542: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:543: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:544: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:545: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:550: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
C:\Users\Oswaldo\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:144: FutureWarning: The sklearn.metrics.classification module is  deprecated in version
0.22 and will be removed in version 0.24. The corresponding classes / functions should instead be imported from sklearn.metrics. Anything that cannot be imported f
rom sklearn.metrics is now part of the private API.
  warnings.warn(message, FutureWarning)
```

Establish a base directory

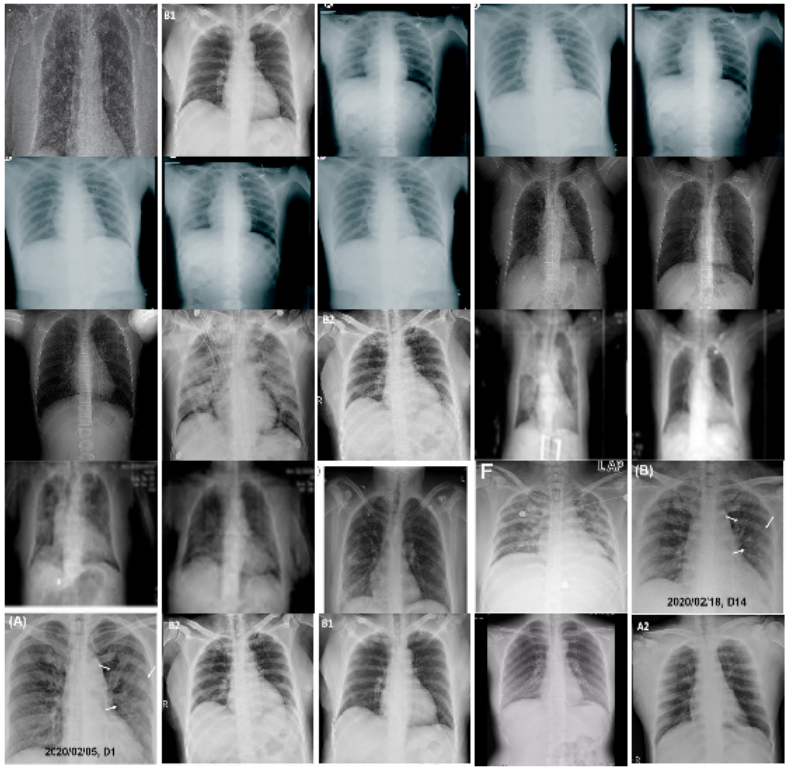
In [2]:
```
BASE_DIR = 'C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/'
```

Import and plot COVID-19 positive images

In [3]:
```
import glob
imagePatches = glob.glob(BASE_DIR +'dataset/Train/Class1/*.png', recursive=True)
for filename in imagePatches[0:10]:
    print(filename)
```

```
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (1).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (10).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (100).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (101).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (102).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (103).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (104).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (105).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (106).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class1\COVID-19 (107).png
```

In [4]:
```
# Plot Multiple Images
bunchOfImages = imagePatches
i_ = 0
plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.subplots_adjust(wspace=0, hspace=0)
for l in bunchOfImages[:25]:
    im = cv2.imread(l)
    im = cv2.resize(im, (224, 224))
    plt.subplot(5, 5, i_+1) #.set_title(l)
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB)); plt.axis('off')
    i_ += 1
```

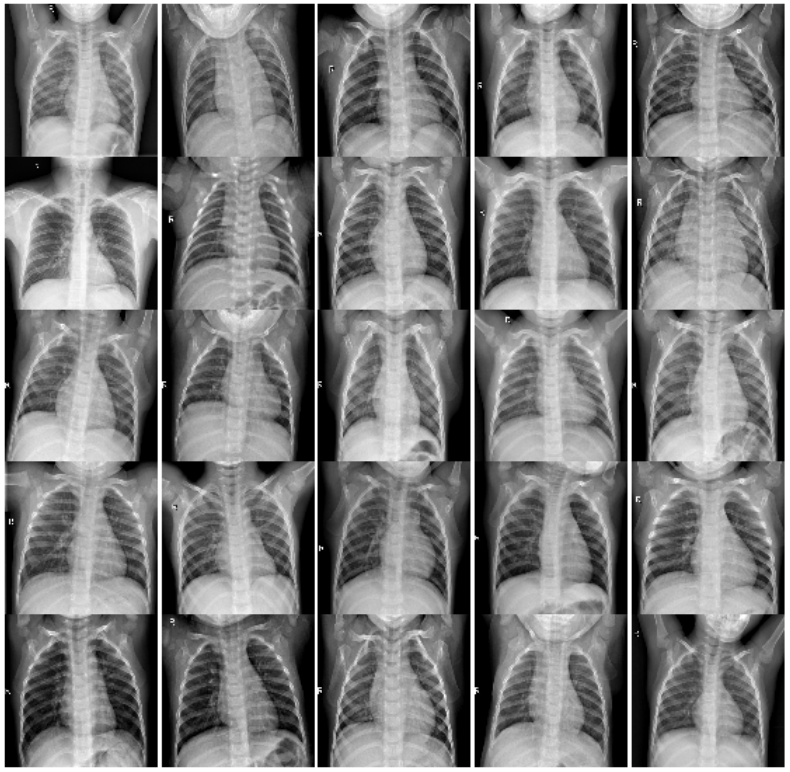


Import and plot Healthy images

In [5]:
```
imagePatches2 = glob.glob(BASE_DIR +'dataset/Train/Class2/*.png', recursive=True)
for filename in imagePatches2[0:10]:
    print(filename)
```

```
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (1).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (10).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (100).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (1000).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (1001).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (1002).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (1003).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (1004).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (1005).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class2\NORMAL (1006).png
```

In [6]:
```
# Plot Multiple Images
bunchOfImages2 = imagePatches2
i_ = 0
plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.subplots_adjust(wspace=0, hspace=0)
for l in bunchOfImages2[:25]:
    im = cv2.imread(l)
    im = cv2.resize(im, (224, 224))
    plt.subplot(5, 5, i_+1) #.set_title(l)
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB)); plt.axis('off')
    i_ += 1
```

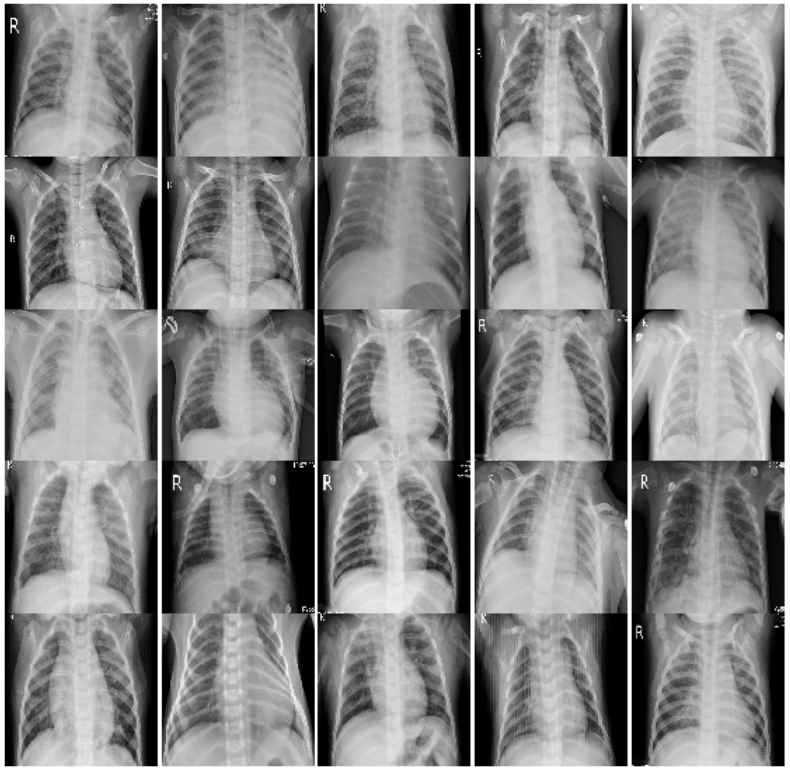


Import and plot Viral Pneumonia positive images

In [7]:
```
imagePatches3 = glob.glob(BASE_DIR +'dataset/Train/Class3/*.png', recursive=True)
for filename in imagePatches3[0:10]:
    print(filename)
```

```
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class3\Viral Pneumonia (1).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class3\Viral Pneumonia (10).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class3\Viral Pneumonia (100).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class3\Viral Pneumonia (1000).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class3\Viral Pneumonia (1001).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class3\Viral Pneumonia (1002).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class3\Viral Pneumonia (1003).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class3\Viral Pneumonia (1004).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class3\Viral Pneumonia (1005).png
C:/Users/Oswaldo/Google Drive/Colab Notebooks/Final_Project/dataset/Train/Class3\Viral Pneumonia (1006).png
```

In [8]:
```
# Plot Multiple Images
bunchOfImages3 = imagePatches3
i_ = 0
plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.subplots_adjust(wspace=0, hspace=0)
for l in bunchOfImages3[:25]:
    im = cv2.imread(l)
    im = cv2.resize(im, (224, 224))
    plt.subplot(5, 5, i_+1) #.set_title(l)
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB)); plt.axis('off')
    i_ += 1
```



Load data and assign labels

In [9]:
```
def get_data(folder):
    """
    Load the data and labels from the given folder.
    """
    X = []
    y = []

    for rayx_type in os.listdir(folder):
        if not rayx_type.startswith('.'):
            if rayx_type in ['Class1']:
                label = '0'
            else:
                if rayx_type in ['Class2']:
                    label = '1'
                else:
                    label ='2'
            for image_filename in os.listdir(folder + rayx_type):
                img_file = cv2.imread(folder + rayx_type + '/' + image_filename, 0)
                if img_file is not None:
                    # Downsample the image to 224, 224, 3
                    image = cv2.resize(img_file, (224, 224))
                    img_arr = np.asarray(image)
                    X.append(img_arr)
                    y.append(label)
    X = np.asarray(X)
    y = np.asarray(y)
    return X,y
```

In [10]:
```
#Define classes
classes = ["0","1","2"]
```

In [11]:
```
X_train, y_train = get_data(BASE_DIR + 'dataset/Train/')
X_test, y_test = get_data(BASE_DIR + 'dataset/Test/')
```

In [12]:
```
#Reshaping the 3D arrays of data to 2D

nsamples, nx, ny = X_train.shape
nsamples2, nx2, ny2 = X_test.shape
X_train_new = X_train.reshape((nsamples,nx*ny))
X_test = X_test.reshape((nsamples2,nx2*ny2))

encoder = LabelEncoder()
encoder.fit(y_train)
y_train = encoder.transform(y_train)
y_test = encoder.transform(y_test)
```

Here we introduce our matrices for parameter optimization of each classifier

Parameter optimization is automated using GridSearchCV

In [13]:
```
#Define the K-Nearest Neighbors parameter values to be tested for optimization

n_neighbors_range = np.linspace(3, 10, num = 8, dtype=int)

#Implement GridSearchCV for K-Nearest Neighbors

param_grid_KNN = dict(n_neighbors=n_neighbors_range)
grid_KNN = GridSearchCV(KNeighborsClassifier(), param_grid=param_grid_KNN, cv=5)
grid_KNN.fit(X_train_new, y_train)

print("The best KNN parameters are %s with a score of %0.2f"
      % (grid_KNN.best_params_, grid_KNN.best_score_))

#Store the best parameter values for use in K-Nearest Neighbors Model

n_neighbors_selected=grid_KNN.best_params_['n_neighbors']
```

```
The best KNN parameters are {'n_neighbors': 8} with a score of 0.89
```

In [14]:
```
#Define the Logistic Regression parameter values to be tested for optimization

C_LR_range = np.linspace(1, 10, num = 11, dtype=float)

#Implement GridSearchCV for Logistic Regression

param_grid_LR = dict(C=C_LR_range)
grid_LR = GridSearchCV(LogisticRegression(), param_grid=param_grid_LR, cv=5)
grid_LR.fit(X_train_new, y_train)

print("The best LR parameters are %s with a score of %0.2f"
      % (grid_LR.best_params_, grid_LR.best_score_))

#Store the best parameter values for use in Logistic Regression Model

C_LR_selected=grid_LR.best_params_['C']
```

```
The best LR parameters are {'C': 1.0} with a score of 0.92
```

In [15]:
```
#Define the Decision Tree parameter values to be tested for optimization

min_samples_leaf_range = np.linspace(1, 5, num = 6, dtype=int)

#Implement GridSearchCV for Decision Tree

param_grid_DT = dict(min_samples_leaf=min_samples_leaf_range)
grid_DT = GridSearchCV(DecisionTreeClassifier(), param_grid=param_grid_DT, cv=5)
grid_DT.fit(X_train_new, y_train)

print("The best DT parameters are %s with a score of %0.2f"
      % (grid_DT.best_params_, grid_DT.best_score_))

#Store the best parameter values for use in Decision Tree Model

min_samples_leaf_selected=grid_DT.best_params_['min_samples_leaf']
```

```
The best DT parameters are {'min_samples_leaf': 3} with a score of 0.79
```

In [16]:
```
#Define the Random Forest parameter values to be tested for optimization

n_estimators_range = np.linspace(1, 5, num = 6, dtype=int)

#Implement GridSearchCV for Random Forest

param_grid_RF = dict(n_estimators=n_estimators_range)
grid_RF = GridSearchCV(RandomForestClassifier(), param_grid=param_grid_RF, cv=5)
grid_RF.fit(X_train_new, y_train)

print("The best RF parameters are %s with a score of %0.2f"
      % (grid_RF.best_params_, grid_RF.best_score_))

#Store the best parameter values for use in Random Forest Model

n_estimators_selected=grid_RF.best_params_['n_estimators']
```

```
The best RF parameters are {'n_estimators': 5} with a score of 0.85
```

In [17]:
```
#Define the Support Vector Classifier parameter values to be tested for optimization

C_SVC_range = np.linspace(1, 100, num = 5, dtype=float)

gamma_SVC_range = np.linspace(1, 5, num = 5, dtype=float)

#Implement GridSearchCV for Support Vector Classifier

param_grid_SVC = dict(C=C_SVC_range, gamma=gamma_SVC_range)
grid_SVC = GridSearchCV(SVC(), param_grid=param_grid_SVC, cv=5)
grid_SVC.fit(X_train_new, y_train)

print("The best SVC parameters are %s with a score of %0.2f"
      % (grid_SVC.best_params_, grid_SVC.best_score_))

#Store the best parameter values for use in Support Vector Classifier Model

C_SVC_selected=grid_SVC.best_params_['C']
gamma_SVC_selected=grid_SVC.best_params_['gamma']
```

```
The best SVC parameters are {'C': 1.0, 'gamma': 1.0} with a score of 0.46
```

Make a bundle of classifiers

In [18]:
```
# select classifiers
classifiers=[
KNeighborsClassifier(n_neighbors = n_neighbors_selected),
LogisticRegression(C = C_LR_selected, random_state=0, multi_class='multinomial', solver='newton-cg'),
DecisionTreeClassifier(min_samples_leaf = min_samples_leaf_selected, random_state=0),
RandomForestClassifier(n_estimators=n_estimators_selected),
SVC(C = C_SVC_selected, gamma = gamma_SVC_selected),
MLPClassifier()]
```

Define functions for the visualization tools

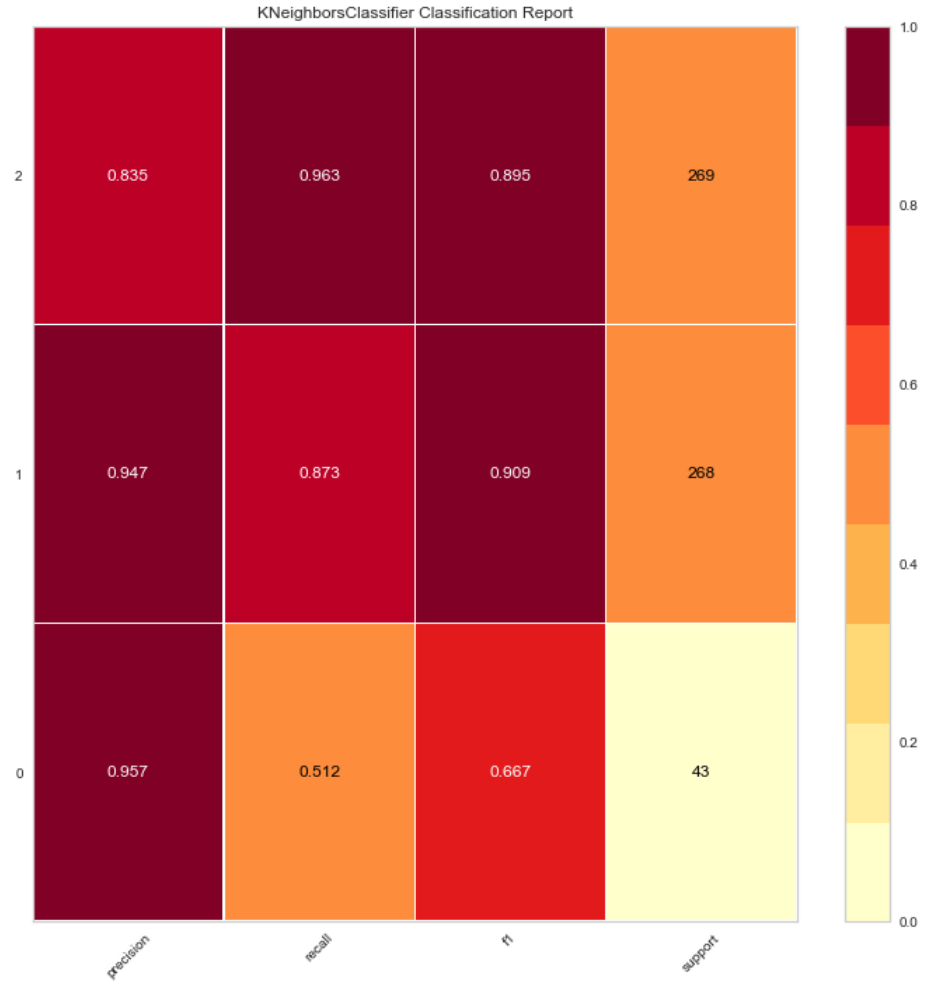
In [19]:
```
#Write function for class-centric metrics
# Classification report
def CR_viz():
    def Class_report(model,classes):
        visualizer = ClassificationReport(model, classes=classes, support=True)
        visualizer.fit(X_train_new, y_train)  # Fit the visualizer and the model
        visualizer.score(X_test, y_test)  # Evaluate the model on the test data
        return visualizer.poof()
    for name in classifiers:
        ax = plt.subplot(1,1,1)
        Class_report(name,classes)

#Class Prediction Error
def CPE_viz():
    def CPE(model,classes):
        visualizer = ClassPredictionError(model, classes=classes)
        visualizer.fit(X_train_new, y_train)  # Fit the visualizer and the model
        visualizer.score(X_test, y_test)  # Evaluate the model on the test data
        return visualizer.poof()
    for name in classifiers:
        ax = plt.subplot(1,1,1)
        CPE(name,classes)

#Confusion matrix
def CM_viz():
    def CM(model,classes):
        visualizer = ConfusionMatrix(model, classes=classes,percent=True)
        visualizer.fit(X_train_new, y_train)  # Fit the visualizer and the model
        visualizer.score(X_test, y_test)  # Evaluate the model on the test data
        return visualizer.poof()
    for name in classifiers:
        ax = plt.subplot(1,1,1)
        CM(name,classes)

#ROC-AUC
def ROC_viz():
    def ROC(model,classes):
        visualizer = ROCAUC(model, classes=classes)
        visualizer.fit(X_train_new, y_train)  # Fit the visualizer and the model
        visualizer.score(X_test, y_test)  # Evaluate the model on the test data
        return visualizer.poof()
    for name in classifiers:
        ax = plt.subplot(1,1,1)
        ROC(name,classes)

#Precision Recall Curve
def PRC_viz():
    def PRC(model,classes):
        visualizer = PrecisionRecallCurve(model,classes=classes, per_class=True, iso_f1_curves=False,
    fill_area=False, micro=False)
        visualizer.fit(X_train_new, y_train)  # Fit the visualizer and the model
        visualizer.score(X_test, y_test)  # Evaluate the model on the test data
        return visualizer.poof()
    for name in classifiers:
        ax = plt.subplot(1,1,1)
        PRC(name,classes)
```

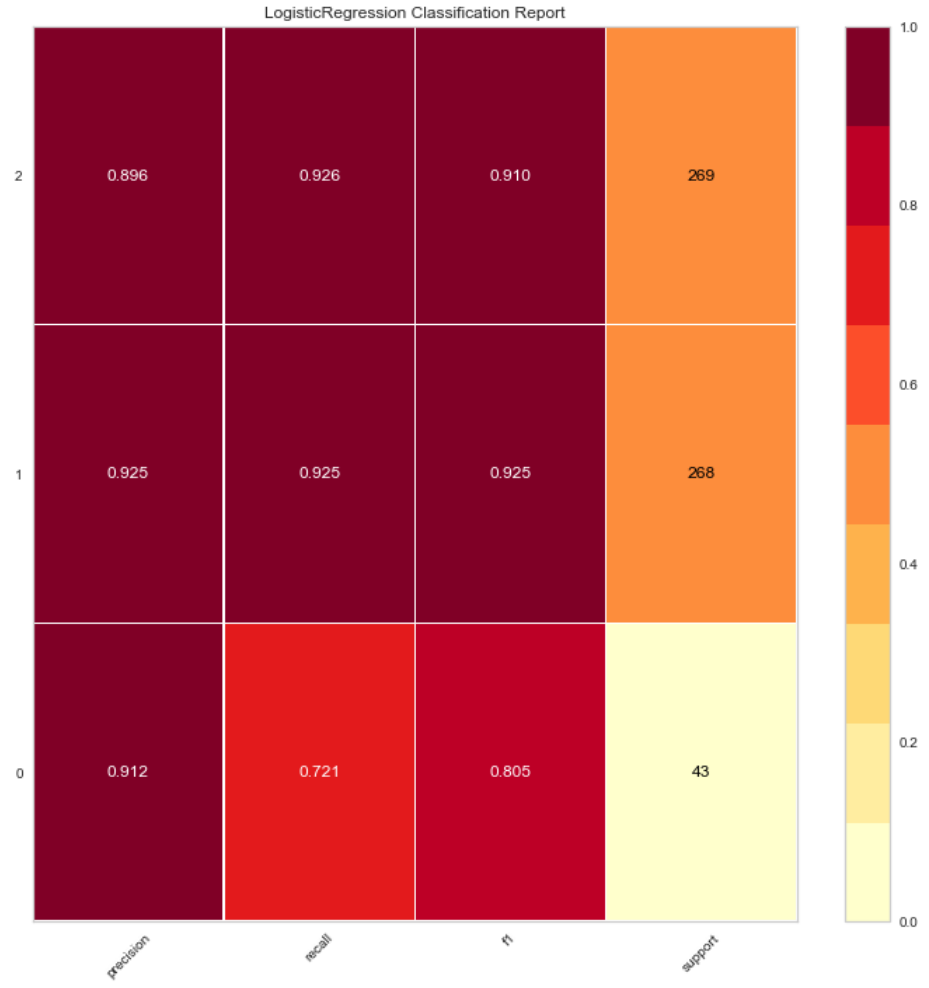
Define function for the aggregate metrics

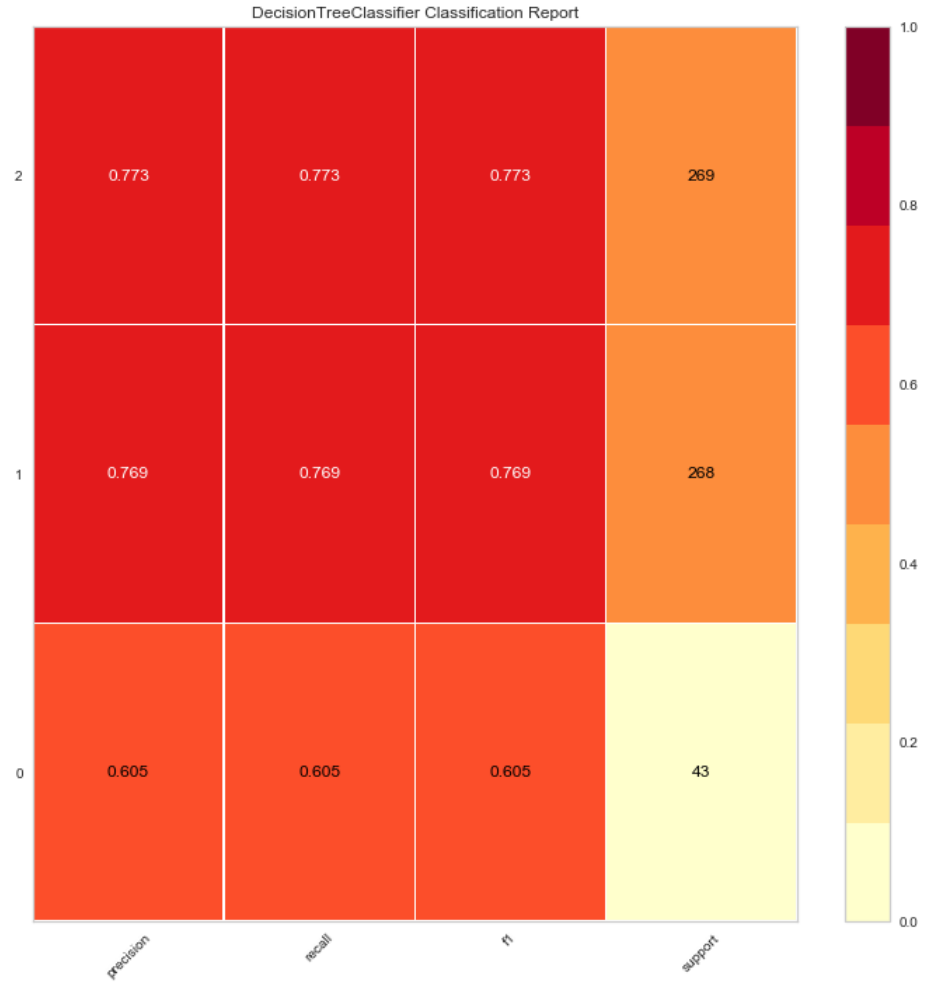
In [20]:
```
# Write function for aggregate metrics
def classifier_metrics():
    def metrics(model):
        #     model=model_name()
        model.fit(X_train_new, y_train)  # Fit the visualizer and the model
        y_pred = model.predict(X_test)
        try:
            y_prob = model.predict_proba(X_test)
            log_metric = log_loss(y_test,y_prob)
        except:
            y_prob = "Not probablistic"
            log_metric = 0
        else:
            y_pred = model.predict(X_test)
        acc_score = accuracy_score(y_test, y_pred)
        c_k_s=cohen_kappa_score(y_test,y_pred)
        zero_met=zero_one_loss(y_test,y_pred)
        hl=hamming_loss(y_test,y_pred)
        mc=matthews_corrcoef(y_test,y_pred)
        print('cohen_kappa_score: {0:.3f}'.format(c_k_s))
        print('log_loss: {0:.3f}'.format(log_metric))
        print('zero_one_loss: {0:.3f}'.format(zero_met))
        print('hemming_loss: {0:.3f}'.format(hl))
        print('matthews_corrcoef: {0:.3f}'.format(mc))
    for name in classifiers:
        print (str(name))
        metrics(name)



        print()
        print ("--------------------------------------------------------")
```

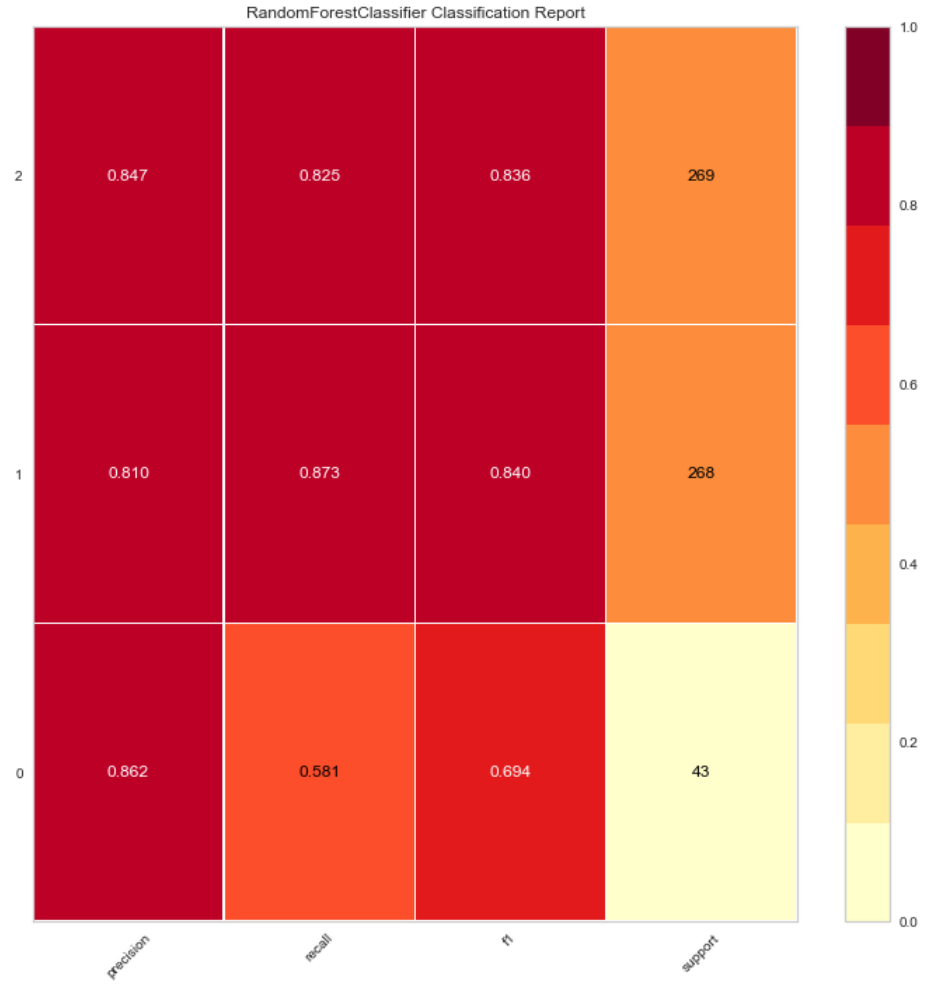
Visualize the results

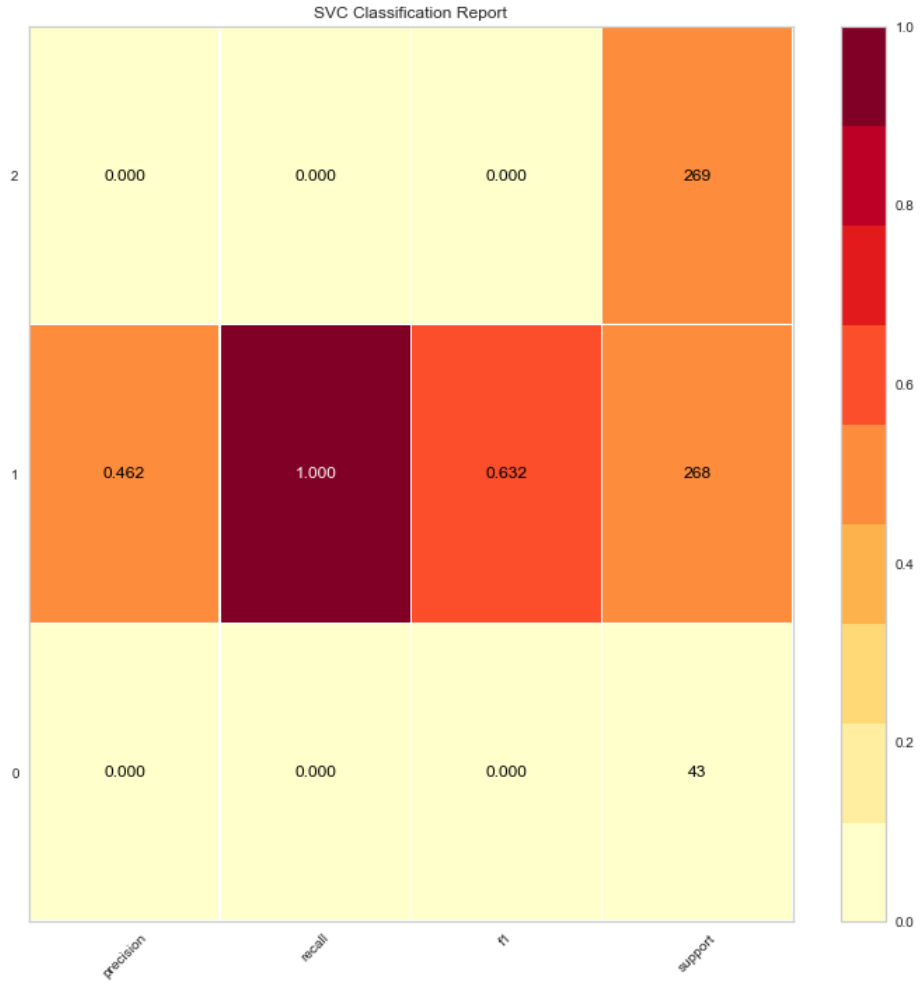
In [21]:
```
#Report the visualization
visualization =[CR_viz(),CPE_viz(),CM_viz(),ROC_viz(),PRC_viz()]
```

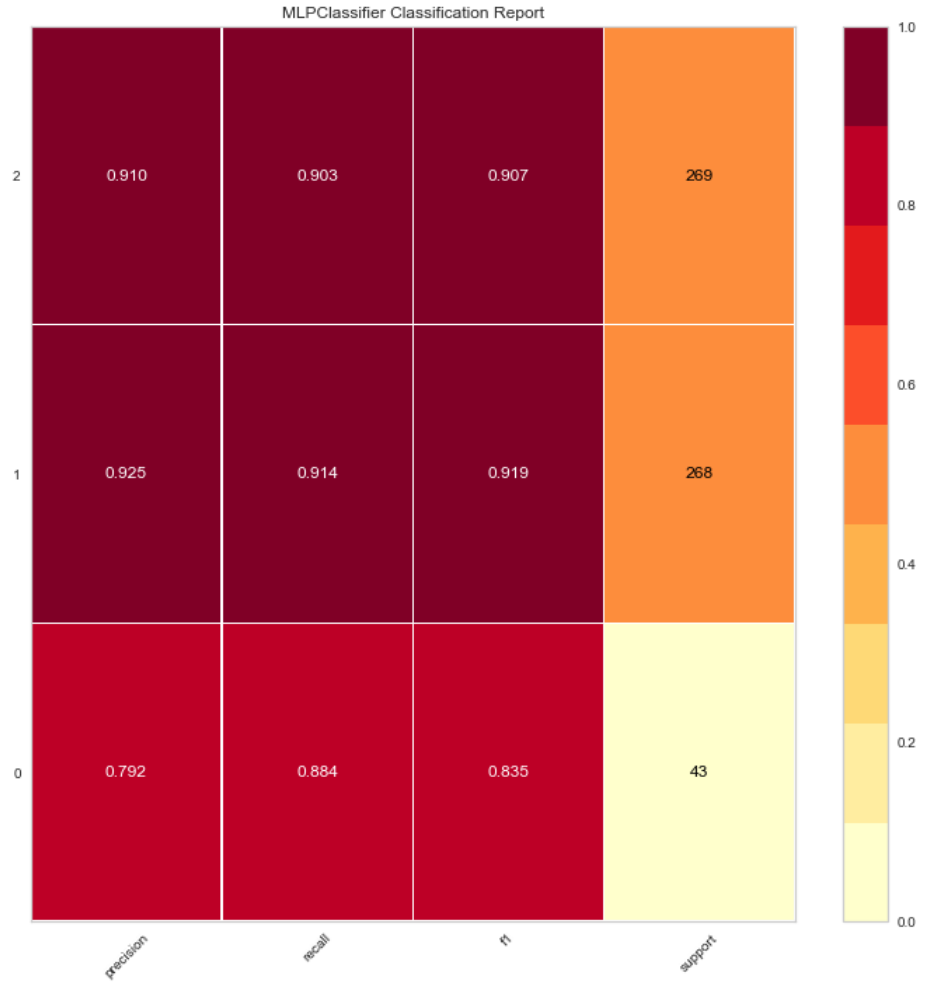

KNeighborsClassifier Classification Report
2
1
0
0.835
0.963
0.895
269
0.947
0.873
0.909
268
0.957
0.512
0.667
43
precision
recall
f1
support
1.0
0.8
0.6
0.4
0.2
0.0

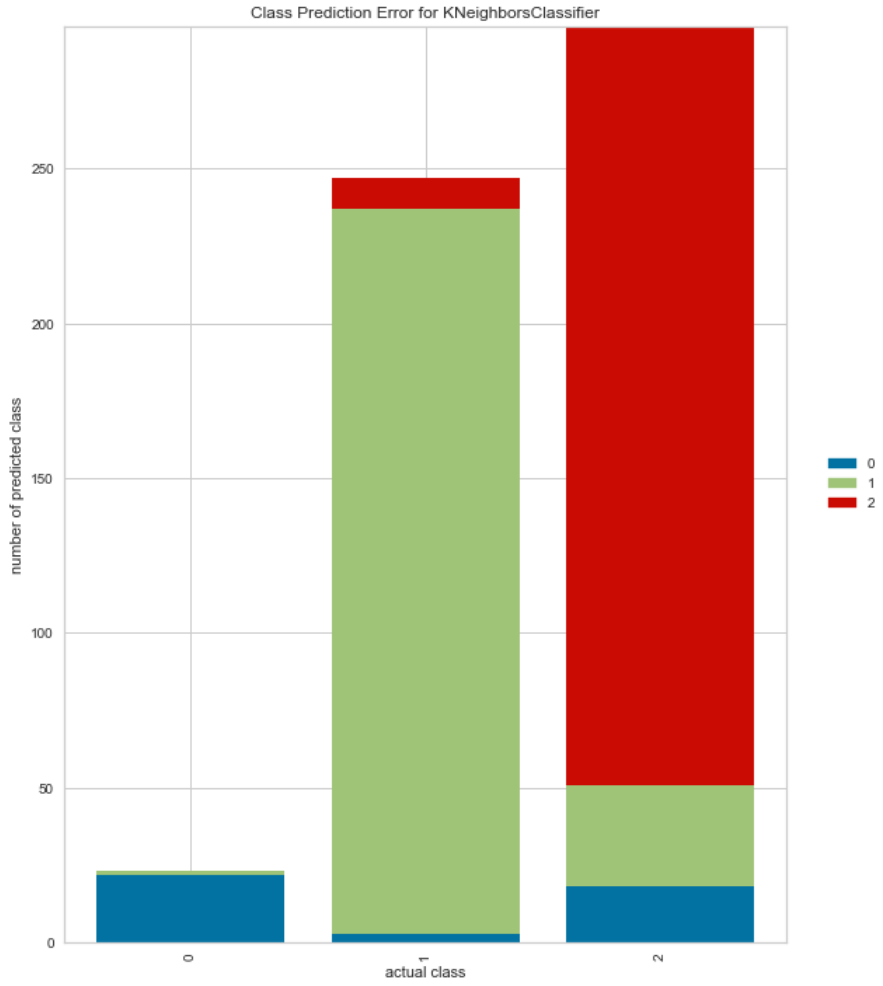

LogisticRegression Classification Report
2
1
0
0.896
0.926
0.910
269
0.925
0.925
0.925
268
0.912
0.721
0.805
43
precision
recall
f1
support
1.0
0.8
0.6
0.4
0.2
0.0

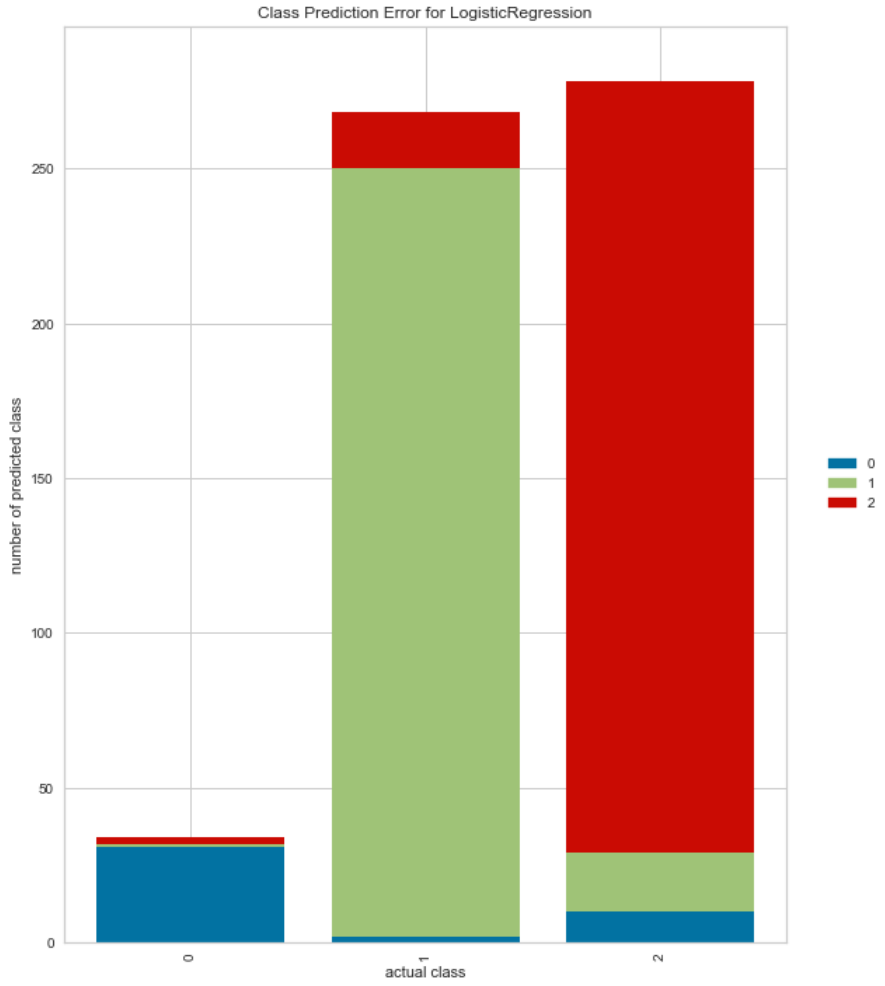

DecisionTreeClassifier Classification Report
2
1
0
0.773
0.773
0.773
269
0.769
0.769
0.769
268
0.605
0.605
0.605
43
precision
recall
f1
support
1.0
0.8
0.6
0.4
0.2
0.0

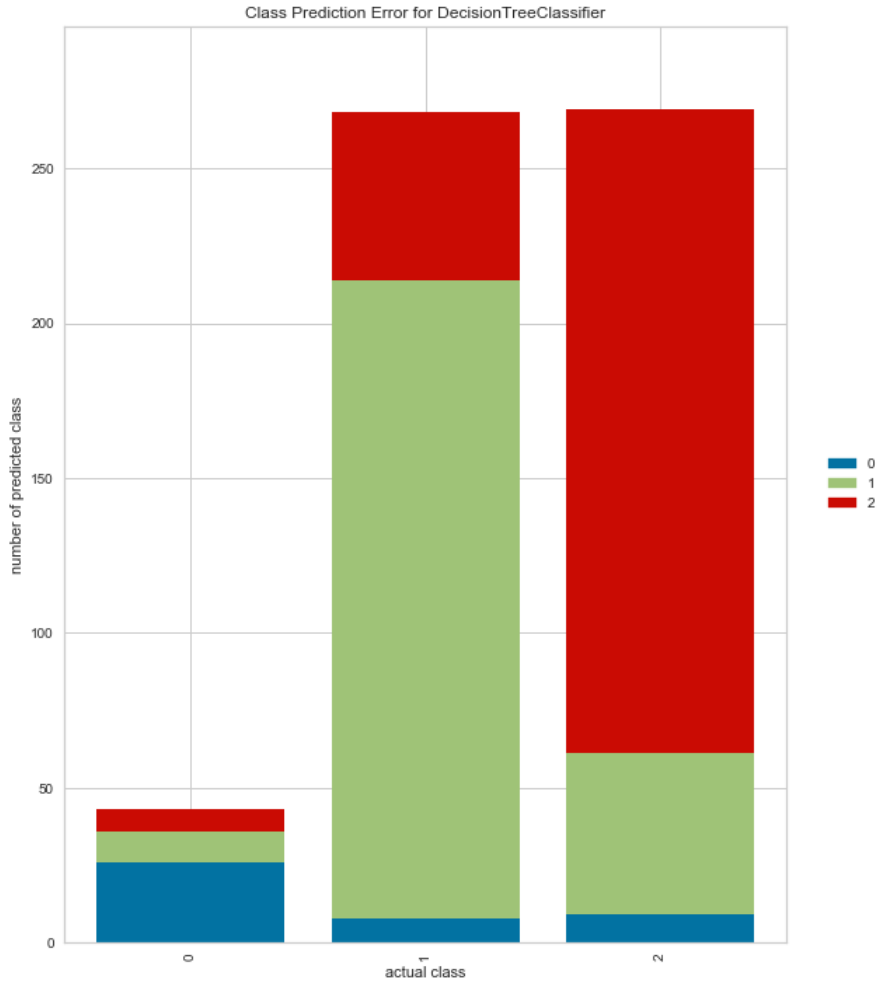

RandomForestClassifier Classification Report
2
1
0
0.847
0.825
0.836
269
0.810
0.873
0.840
268
0.862
0.581
0.694
43
precision
recall
f1
support
1.0
0.8
0.6
0.4
0.2
0.0

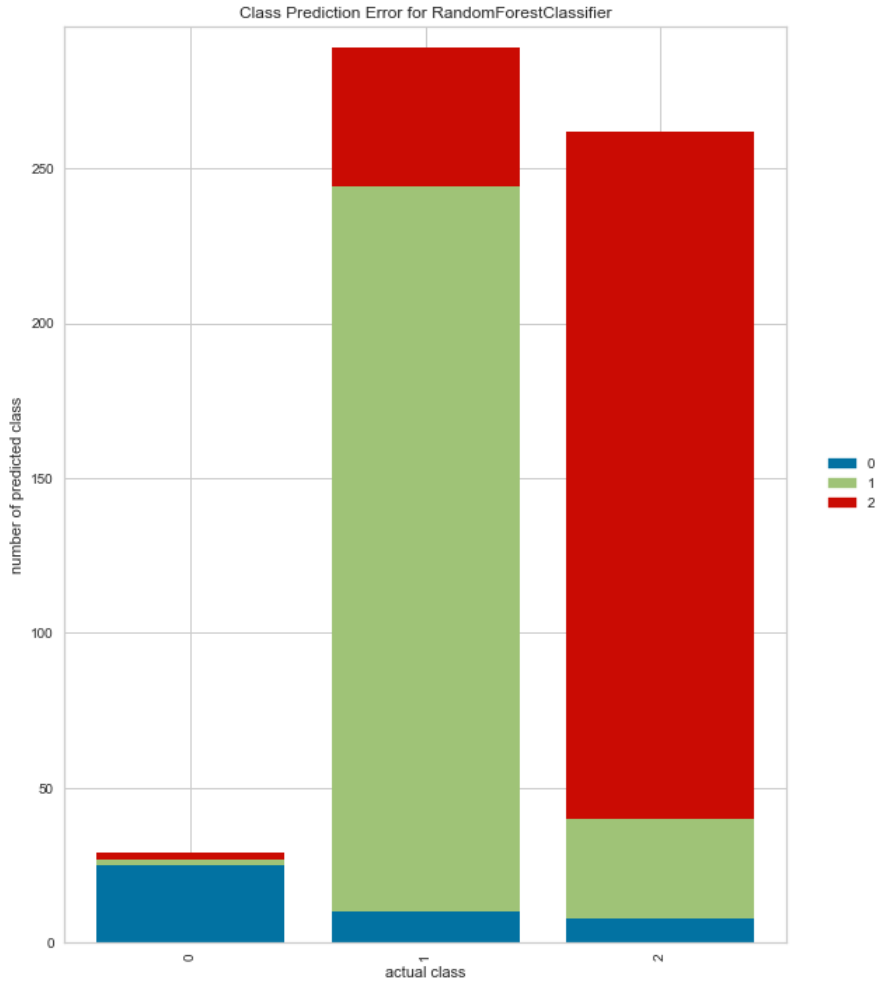

SVC Classification Report
2
1
0
0.000
0.000
0.000
269
0.462
1.000
0.632
268
0.000
0.000
0.000
43
precision
recall
f1
support
1.0
0.8
0.6
0.4
0.2
0.0

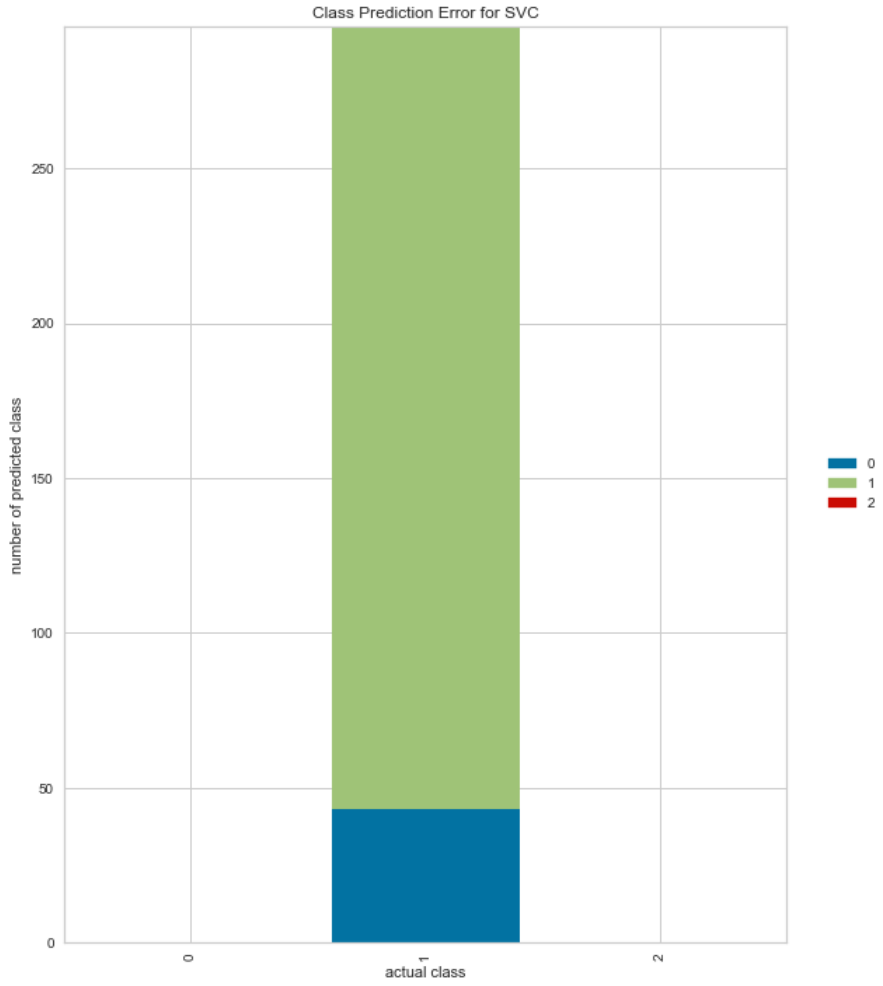

MLPClassifier Classification Report
2
1
0
0.910
0.903
0.907
269
0.925
0.914
0.919
268
0.792
0.884
0.835
43
precision
recall
f1
support
1.0
0.8
0.6
0.4
0.2
0.0

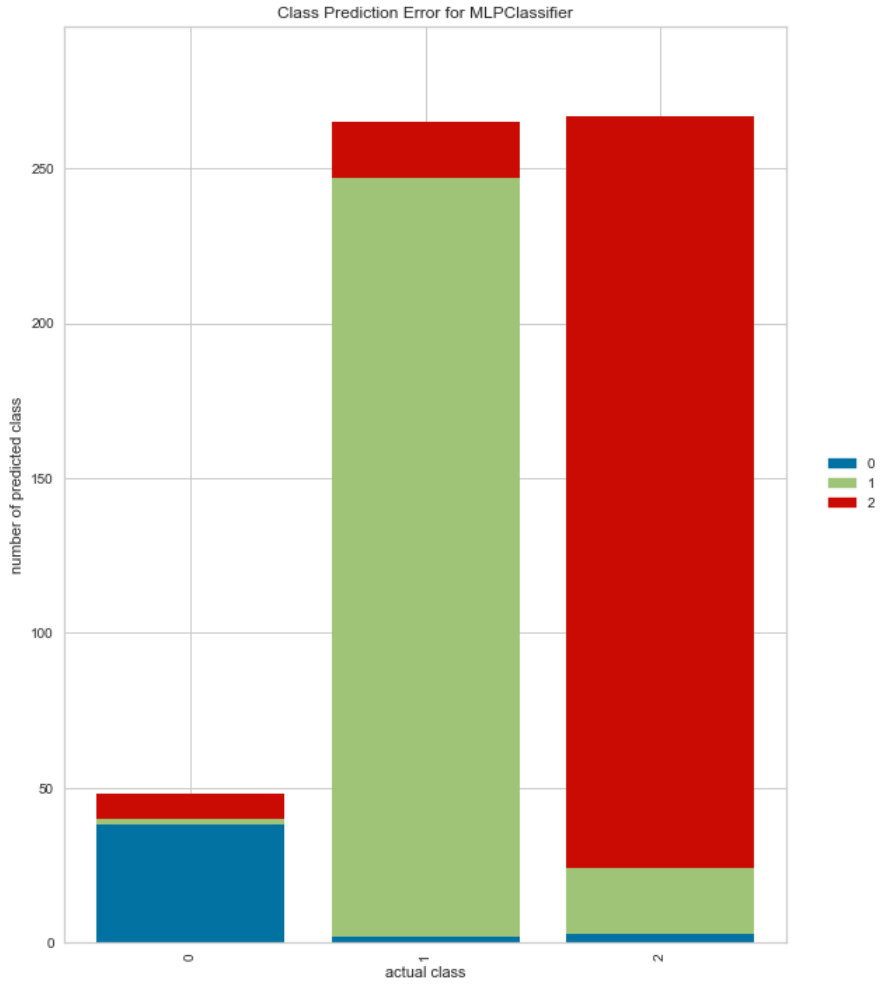

Class Prediction Error for KNeighborsClassifier
number of predicted class
actual class
0
1
2
0
50
100
150
200
250

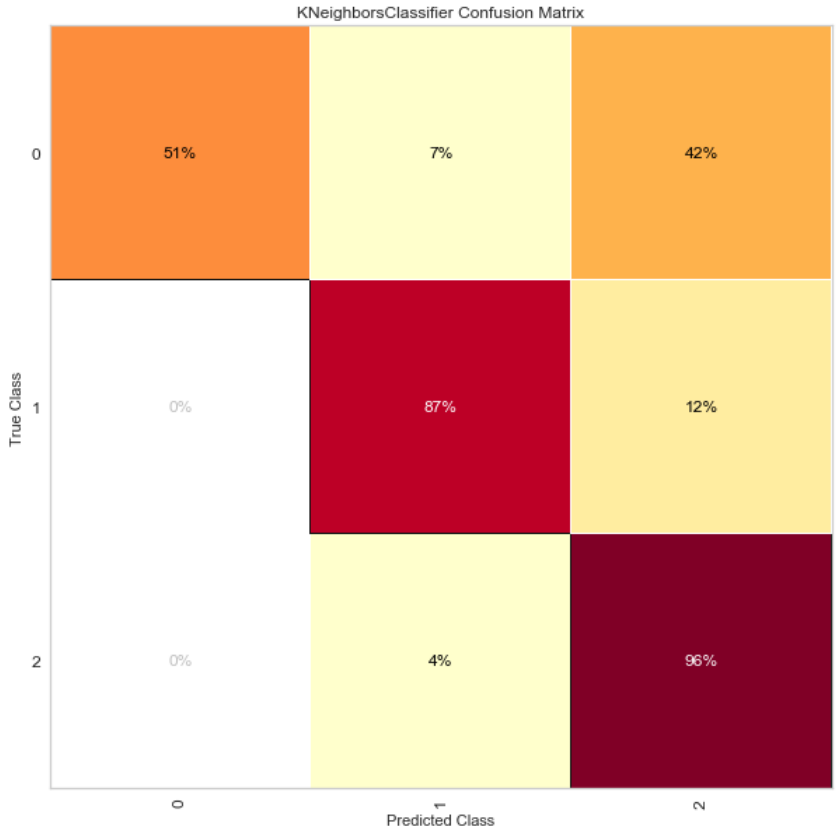

Class Prediction Error for LogisticRegression
number of predicted class
actual class
0
1
2
0
50
100
150
200
250

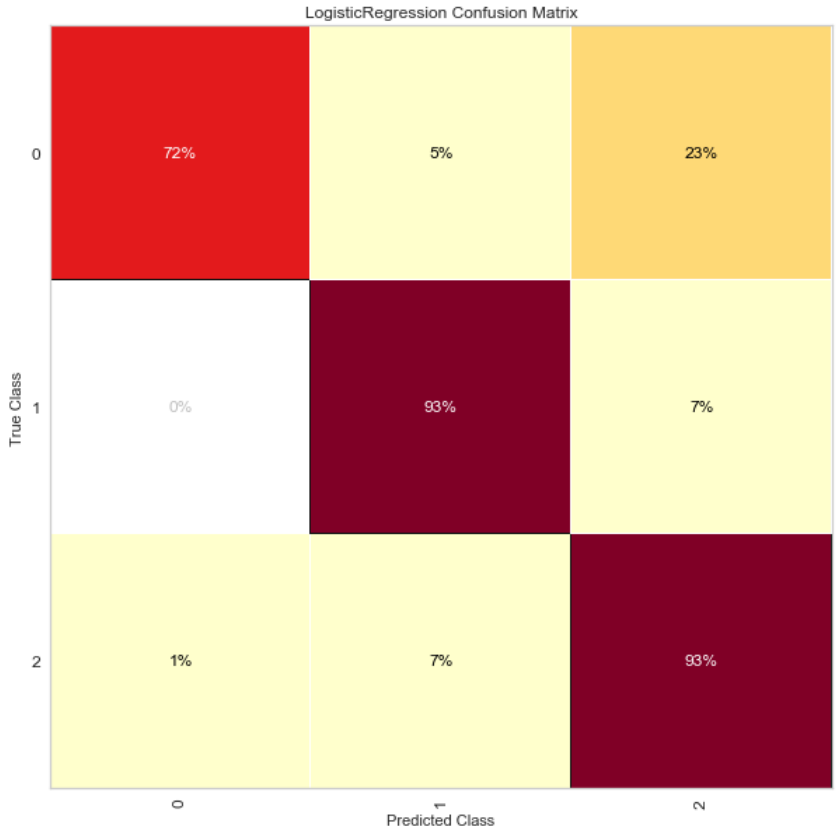

Class Prediction Error for DecisionTreeClassifier
number of predicted class
actual class
0
1
2
0
50
100
150
200
250
0
1
2

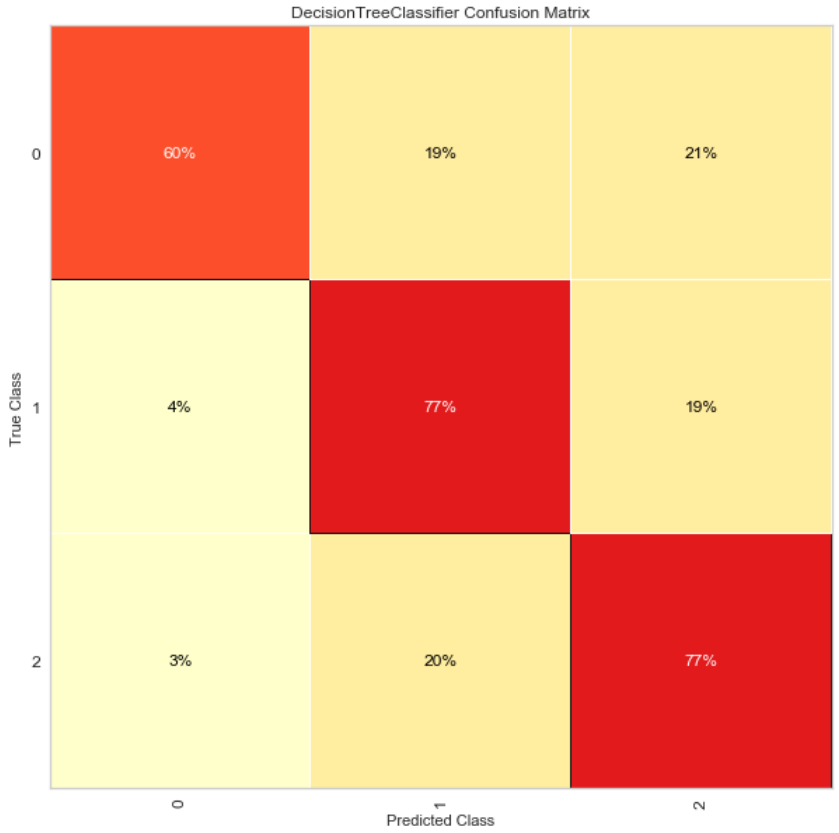

Class Prediction Error for RandomForestClassifier
number of predicted class
0
50
100
150
200
250
0
1
2
actual class
0
1
2

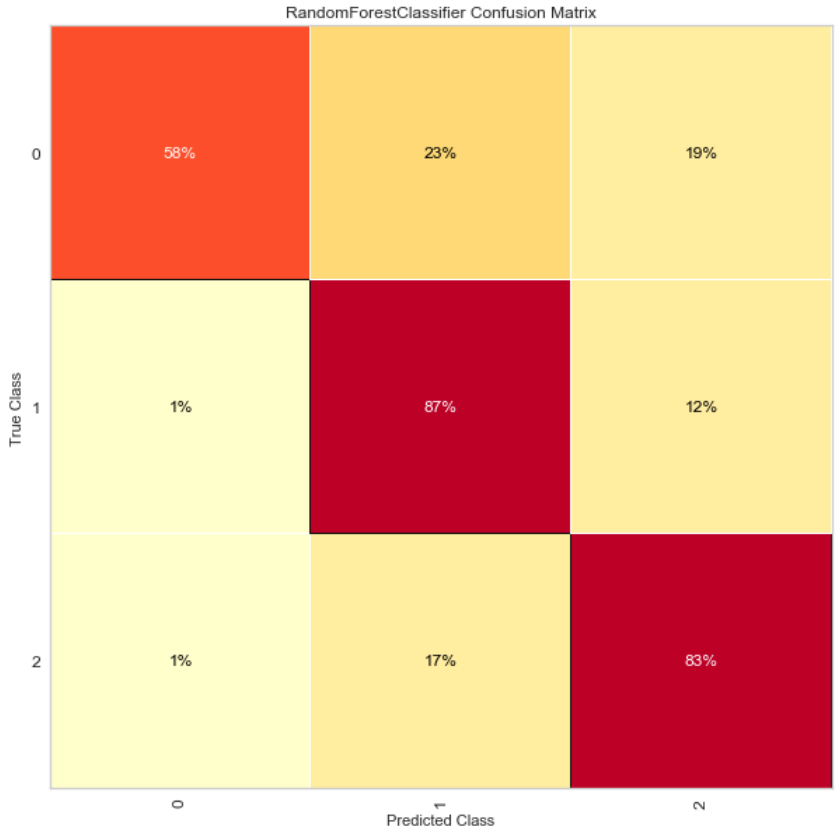

Class Prediction Error for SVC
number of predicted class
actual class
0
1
2
0
50
100
150
200
250
0
1
2

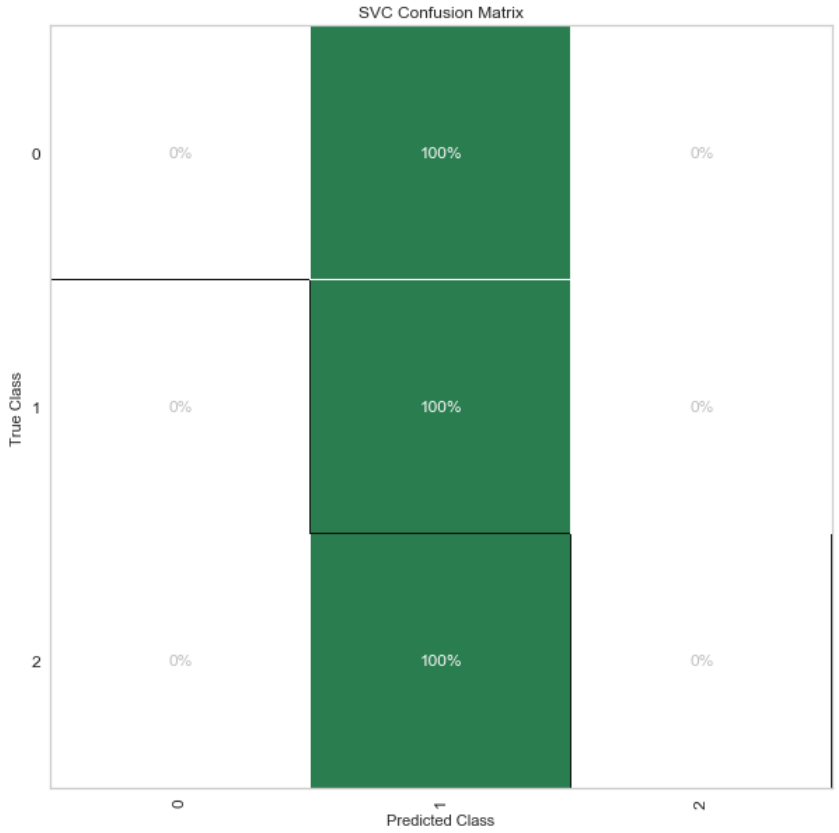

Class Prediction Error for MLPClassifier
number of predicted class
actual class
0
1
2
0
50
100
150
200
250

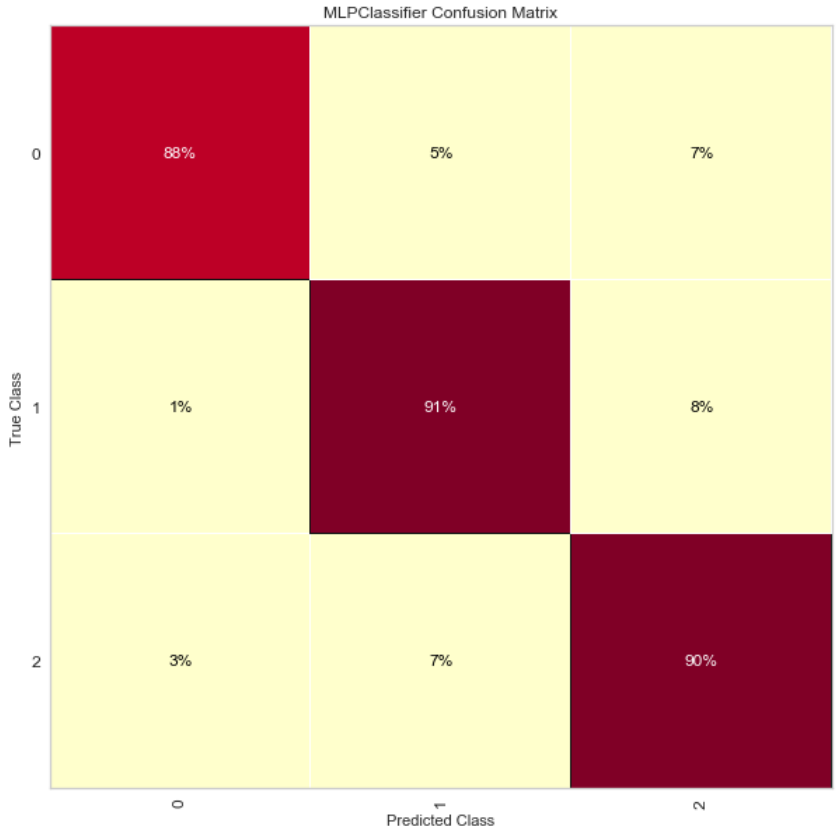

KNeighborsClassifier Confusion Matrix
True Class
Predicted Class
0
1
2
51%
7%
42%
0%
87%
12%
0%
4%
96%

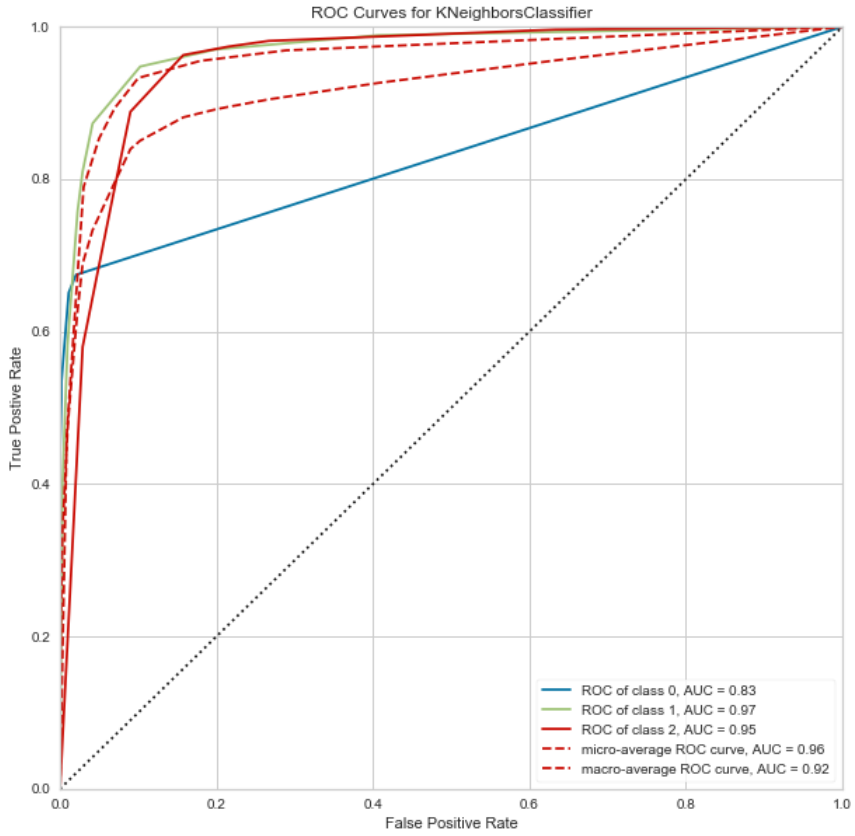

LogisticRegression Confusion Matrix
True Class
Predicted Class
0
1
2
72%
5%
23%
0%
93%
7%
1%
7%
93%

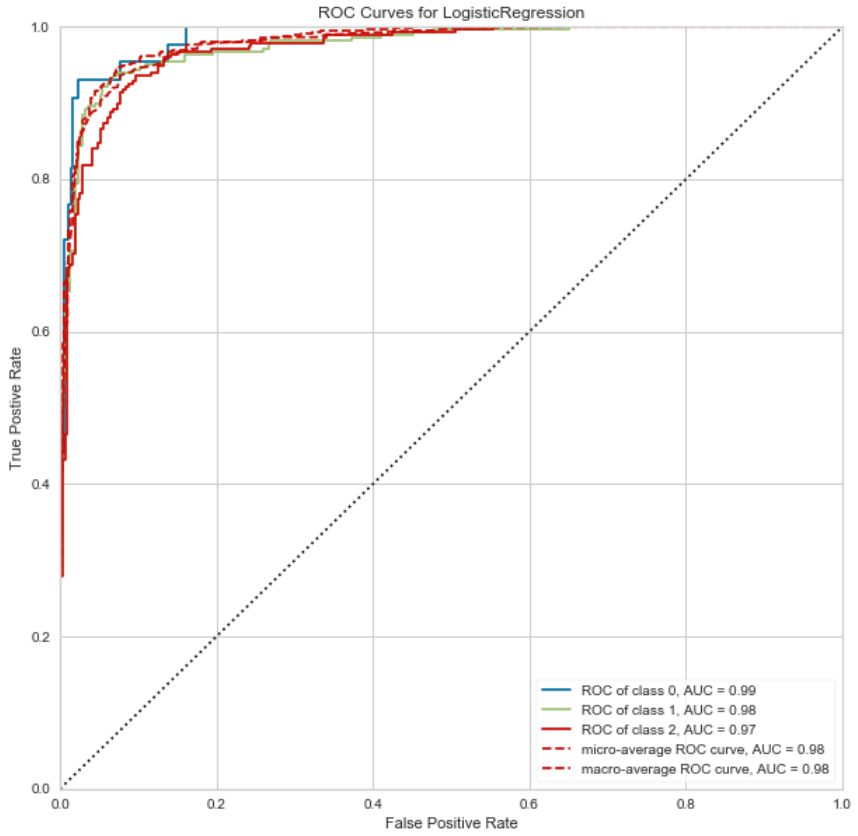

DecisionTreeClassifier Confusion Matrix
True Class
Predicted Class
0
1
2
60%
19%
21%
4%
77%
19%
3%
20%
77%

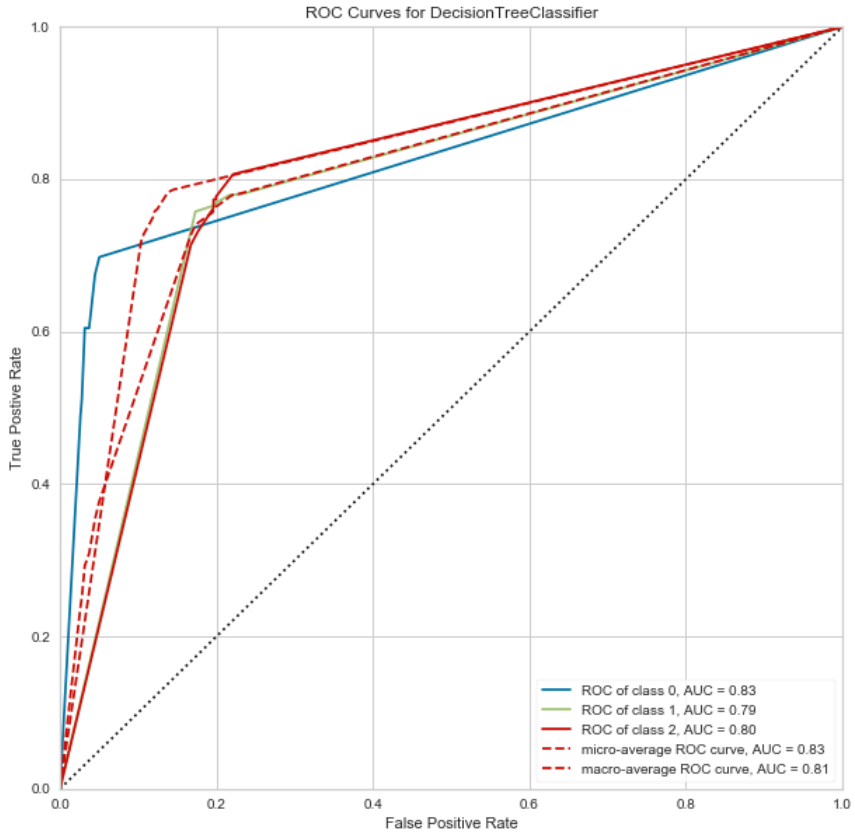

RandomForestClassifier Confusion Matrix
True Class
Predicted Class
0
1
2
58%
23%
19%
1%
87%
12%
1%
17%
83%

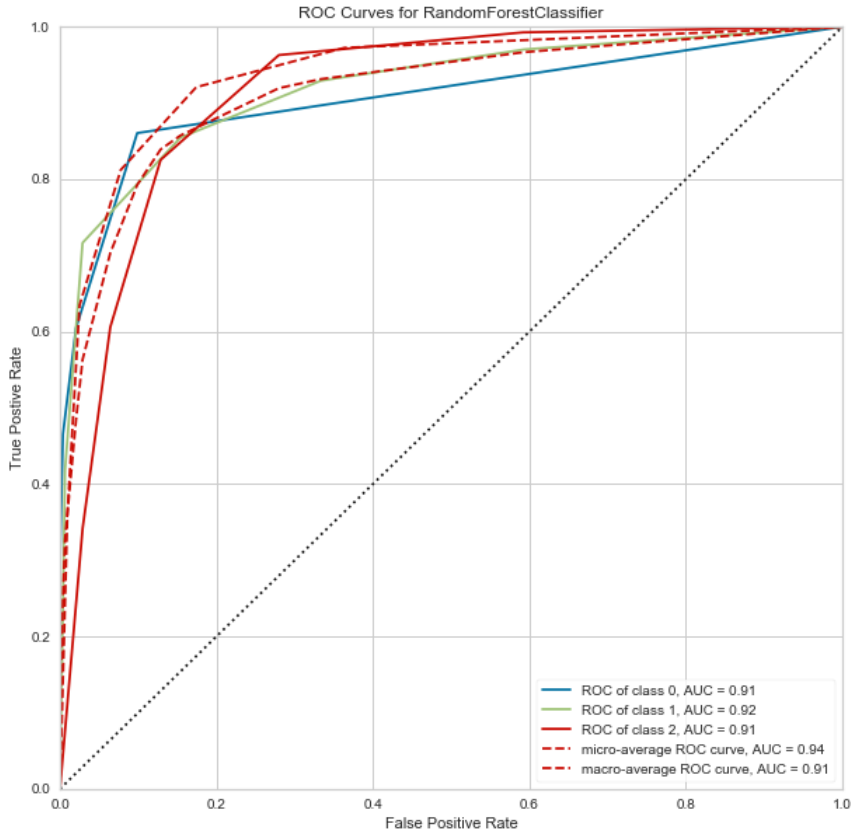

SVC Confusion Matrix
0
1
2
True Class
0%
100%
0%
0%
100%
0%
0%
100%
0%
0
1
2
Predicted Class

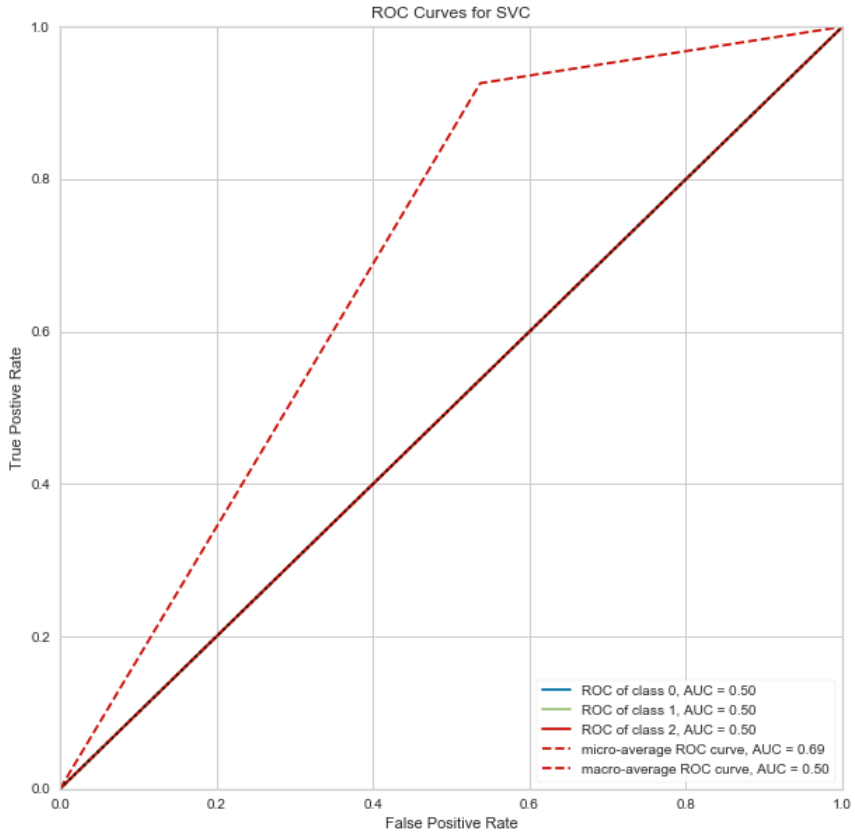

MLPClassifier Confusion Matrix
True Class
Predicted Class
0
1
2
88%
5%
7%
1%
91%
8%
3%
7%
90%

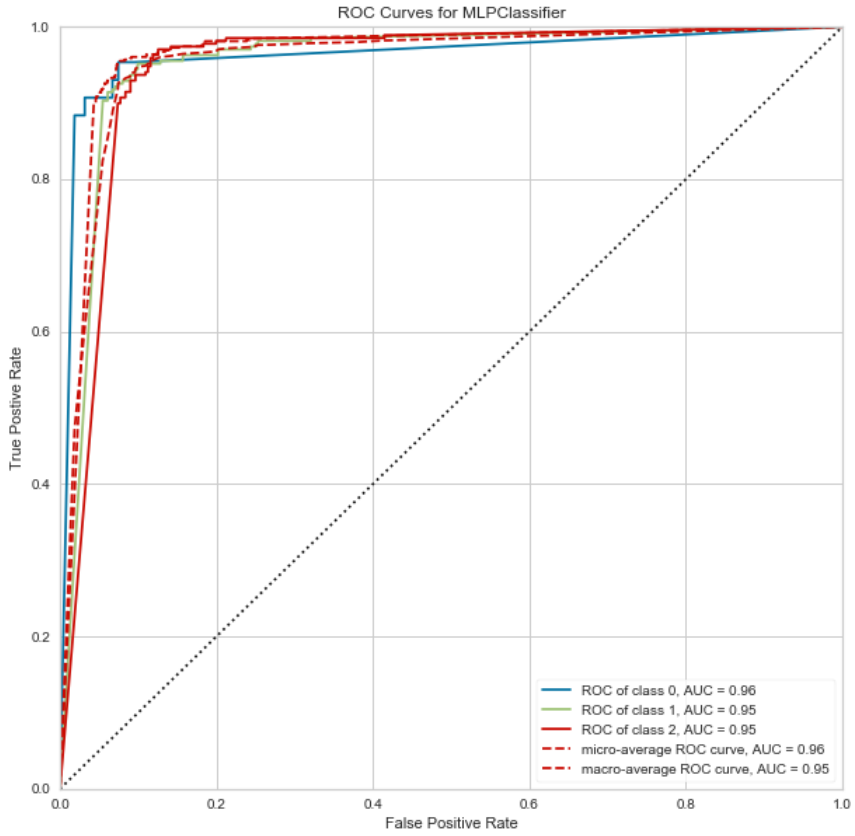

ROC Curves for KNeighborsClassifier
True Postive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC of class 0, AUC = 0.83
ROC of class 1, AUC = 0.97
ROC of class 2, AUC = 0.95
micro-average ROC curve, AUC = 0.96
macro-average ROC curve, AUC = 0.92

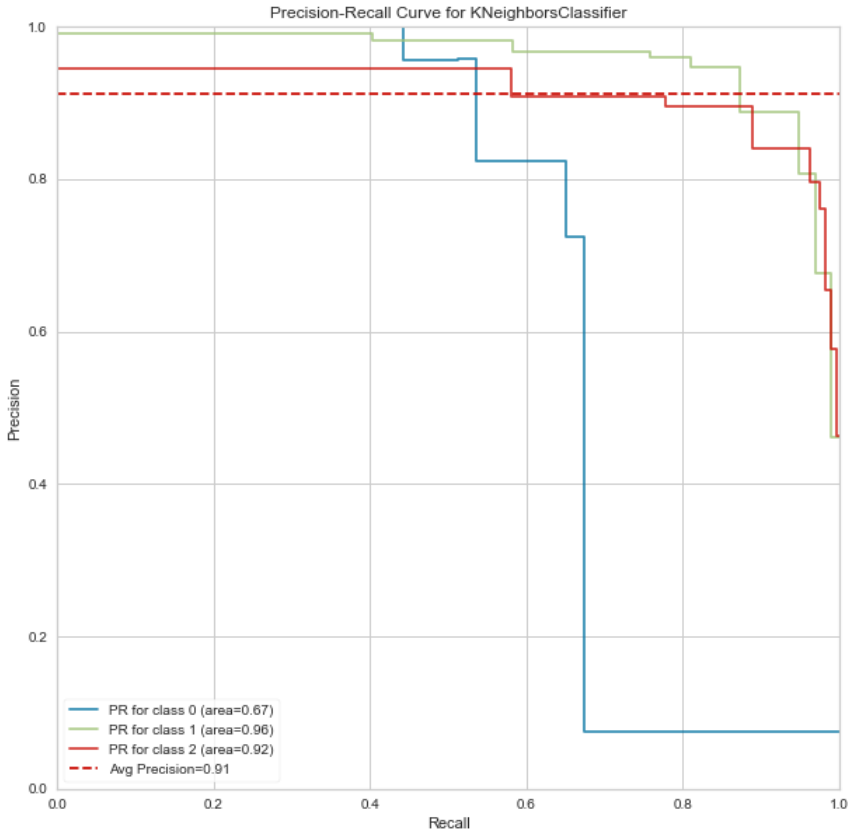

ROC Curves for LogisticRegression
True Postive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC of class 0, AUC = 0.99
ROC of class 1, AUC = 0.98
ROC of class 2, AUC = 0.97
micro-average ROC curve, AUC = 0.98
macro-average ROC curve, AUC = 0.98

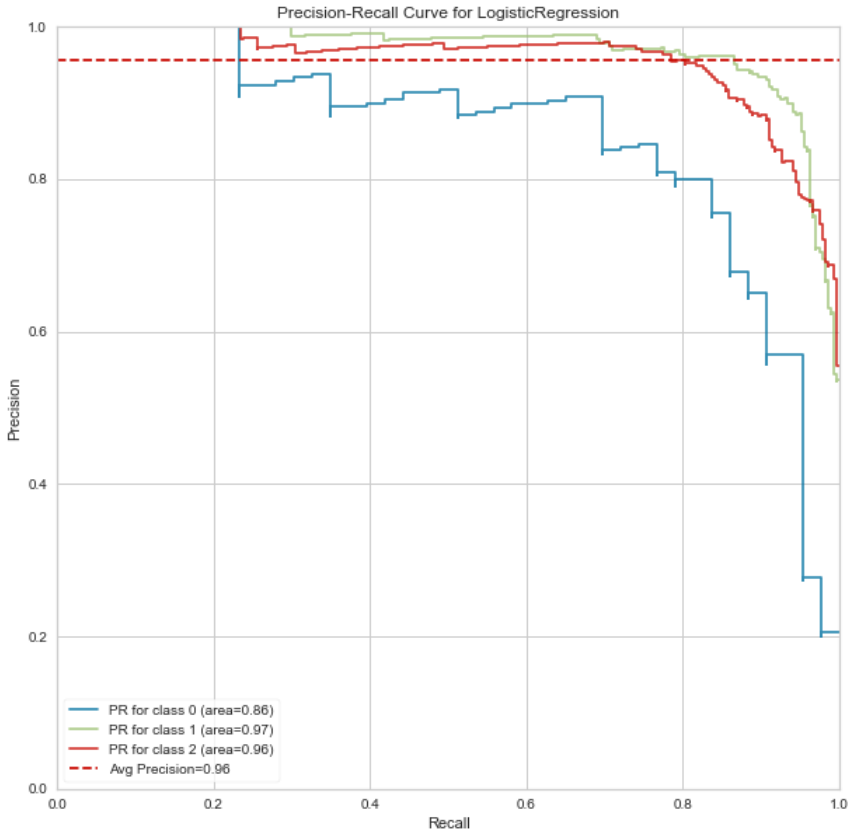

ROC Curves for DecisionTreeClassifier
True Postive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC of class 0, AUC = 0.83
ROC of class 1, AUC = 0.79
ROC of class 2, AUC = 0.80
micro-average ROC curve, AUC = 0.83
macro-average ROC curve, AUC = 0.81

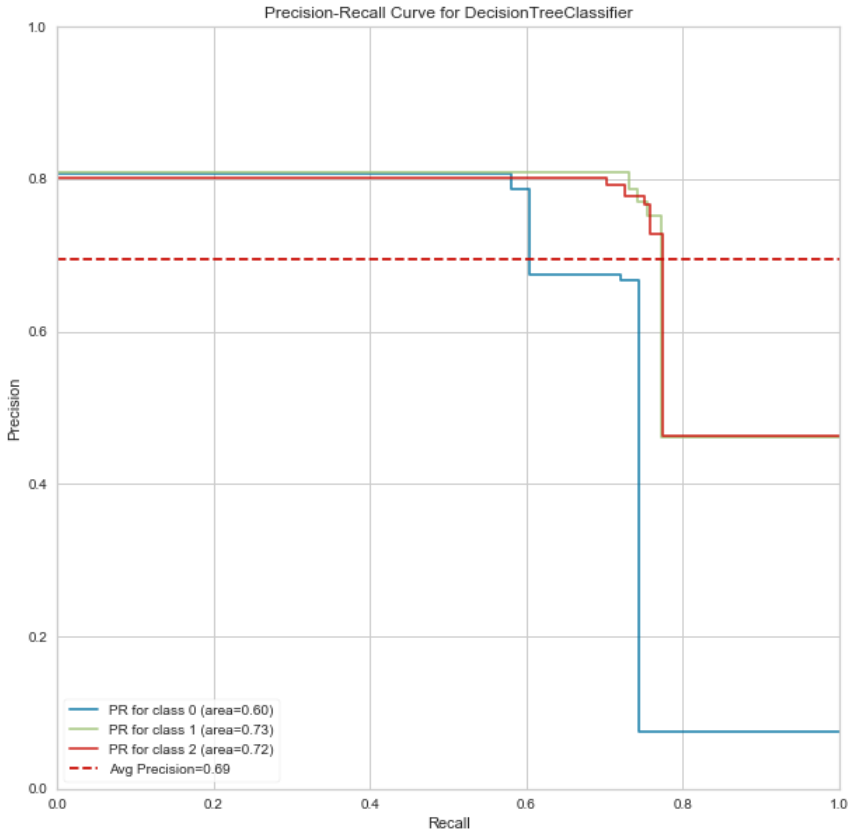

ROC Curves for RandomForestClassifier
True Postive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC of class 0, AUC = 0.91
ROC of class 1, AUC = 0.92
ROC of class 2, AUC = 0.91
micro-average ROC curve, AUC = 0.94
macro-average ROC curve, AUC = 0.91

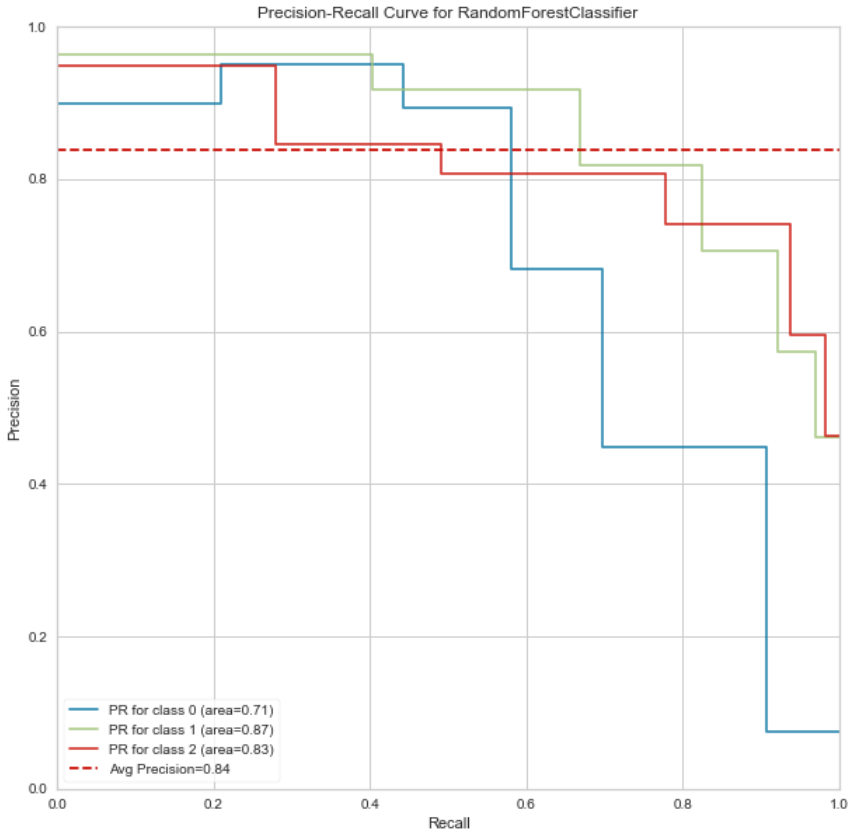

ROC Curves for SVC
True Postive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC of class 0, AUC = 0.50
ROC of class 1, AUC = 0.50
ROC of class 2, AUC = 0.50
micro-average ROC curve, AUC = 0.69
macro-average ROC curve, AUC = 0.50

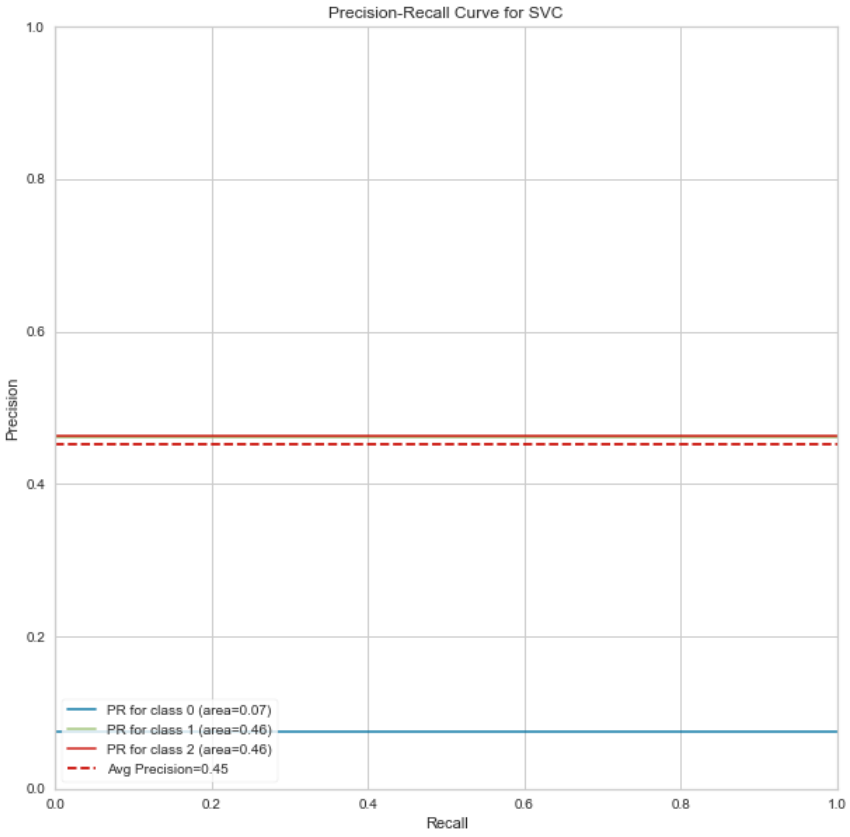

ROC Curves for MLPClassifier
True Postive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC of class 0, AUC = 0.96
ROC of class 1, AUC = 0.95
ROC of class 2, AUC = 0.95
micro-average ROC curve, AUC = 0.96
macro-average ROC curve, AUC = 0.95

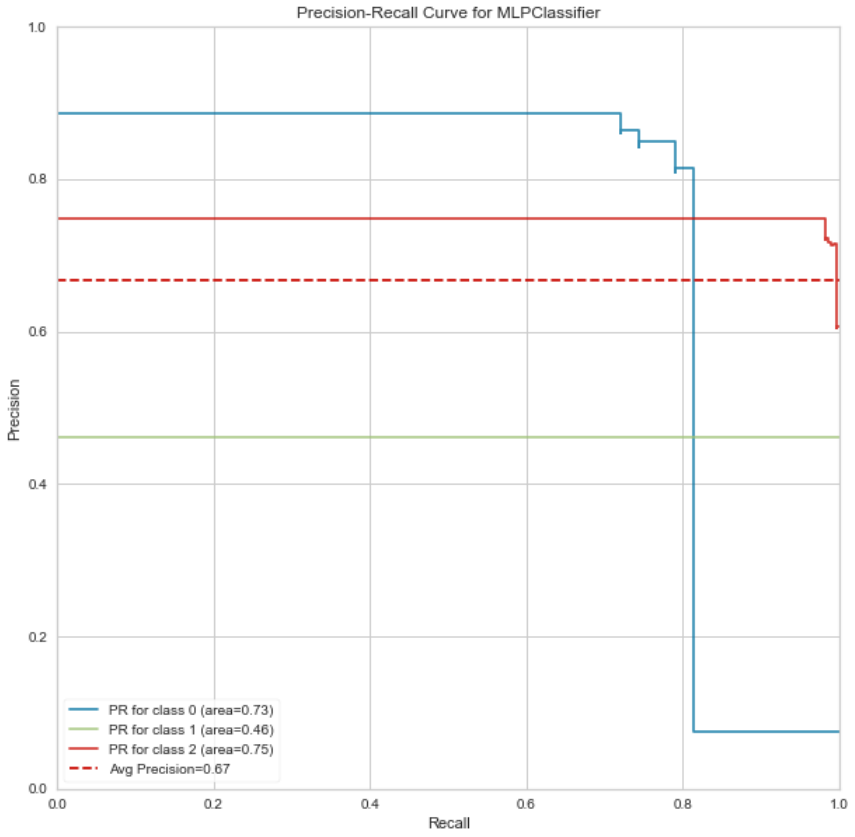

Precision-Recall Curve for KNeighborsClassifier
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
PR for class 0 (area=0.67)
PR for class 1 (area=0.96)
PR for class 2 (area=0.92)
Avg Precision=0.91


Precision-Recall Curve for LogisticRegression
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
PR for class 0 (area=0.86)
PR for class 1 (area=0.97)
PR for class 2 (area=0.96)
Avg Precision=0.96


Precision-Recall Curve for DecisionTreeClassifier
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
PR for class 0 (area=0.60)
PR for class 1 (area=0.73)
PR for class 2 (area=0.72)
Avg Precision=0.69


Precision-Recall Curve for RandomForestClassifier
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
PR for class 0 (area=0.71)
PR for class 1 (area=0.87)
PR for class 2 (area=0.83)
Avg Precision=0.84


Precision-Recall Curve for SVC
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
PR for class 0 (area=0.07)
PR for class 1 (area=0.46)
PR for class 2 (area=0.46)
Avg Precision=0.45



Report the metrics

In [22]:
```
#Report the aggregate metrics
classifier_metrics()
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=8, p=2,
                     weights='uniform')
cohen_kappa_score: 0.797
log_loss: 1.293
zero_one_loss: 0.112
hemming_loss: 0.112
matthews_corrcoef: 0.803

-----------------------------------------------------
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='multinomial', n_jobs=None, penalty='l2',
                   random_state=0, solver='newton-cg', tol=0.0001, verbose=0,
                   warm_start=False)
cohen_kappa_score: 0.840
log_loss: 0.911
zero_one_loss: 0.090
hemming_loss: 0.090
matthews_corrcoef: 0.840

-----------------------------------------------------
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=3, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=0, splitter='best')
cohen_kappa_score: 0.573
log_loss: 7.432
zero_one_loss: 0.241
hemming_loss: 0.241
matthews_corrcoef: 0.573

-----------------------------------------------------
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=5,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
cohen_kappa_score: 0.643
log_loss: 1.428
zero_one_loss: 0.200
hemming_loss: 0.200
matthews_corrcoef: 0.644

-----------------------------------------------------
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=1.0, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
cohen_kappa_score: 0.000
log_loss: 0.000
zero_one_loss: 0.538
hemming_loss: 0.538
matthews_corrcoef: 0.000

-----------------------------------------------------
```

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=200,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=None, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
cohen_kappa_score: 0.824
log_loss: 3.255
zero_one_loss: 0.098
hemming_loss: 0.098
matthews_corrcoef: 0.828

-----------------------------------------------------
```