İstanbul Bilgi University

CMPE 407 Project

by

Anas Alhwid

118200014

Kareem Alshawaf

118200002

Ragid Hallak

118200012

Sadiq Tijjani

119200113

Introduction:

In this project, we aim to combine 2 datasets; ChestX-ray8 and ChestX-ray8 for Covid-19. Then we try to predict each image and which class it belongs to. We have used NIH dataset and combined it with the covid, pneumonia and no finding dataset. We will have a more diverse dataset to classify into different classes. The method used for this project is classification since it is the most suitable for classifying images. It is a predictive modeling problem where a class label is predicted for a given example of input data. We have used several learning models to find the best and the most accurate results.

Dataset:

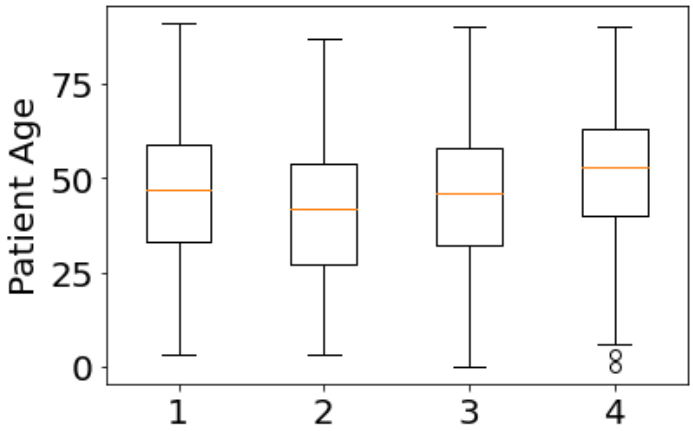
The NIH dataset is a dataset that consists of around 100.000 de-identified images of chest x-rays. By combining it with covid, pneumonia and no findings dataset, we will be able to train and help identify each image and where it belongs to. For visualization, we have used several techniques. Firstly, we have printed the first few records of the datasets to have some idea about the columns and its data.

	Image Index	Finding Labels	Follow-up #	Patient ID	Patient Age	Patient Gender
0	00000001_000.png	Cardiomegaly	0	1	57	M
1	00000001_001.png	Cardiomegaly|Emphysema	1	1	58	M
2	00000001_002.png	Cardiomegaly|Effusion	2	1	58	M
3	00000002_000.png	No Finding	0	2	80	M
4	00000003_001.png	Hernia	0	3	74	F

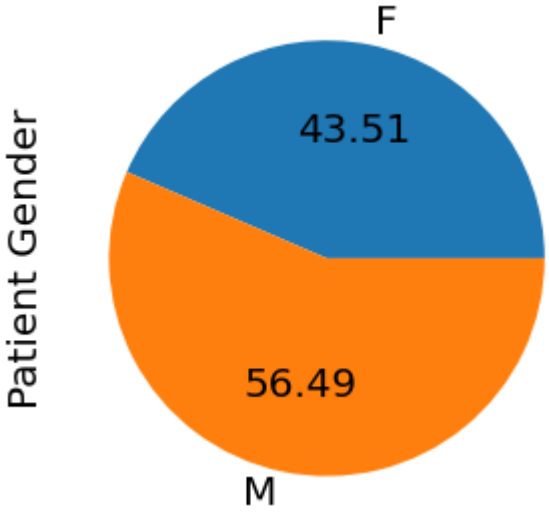
Following that, we used df.describe to gather information about the data's count, mean, frequency, etc. In totality we found that the NIH dataset has 112120 entry features with 11 columns and 3 different data types, namely object, float64 and int64.

	Image Index	Finding Labels	Follow-up #	Patient ID	Patient Age	Patient Gender	View Position
count	112120	112120	112120.000000	112120.000000	112120.000000	112120	112120
unique	112120	836	NaN	NaN	NaN	2	2
top	00000001_000.png	No Finding	NaN	NaN	NaN	M	PA
freq	1	60361	NaN	NaN	NaN	63340	67310
mean	NaN	NaN	8.573751	14346.381743	46.626365	NaN	NaN
std	NaN	NaN	15.406320	8403.876972	16.602680	NaN	NaN
min	NaN	NaN	0.000000	1.000000	0.000000	NaN	NaN
25%	NaN	NaN	0.000000	7310.750000	34.000000	NaN	NaN
50%	NaN	NaN	3.000000	13993.000000	49.000000	NaN	NaN
75%	NaN	NaN	10.000000	20673.000000	59.000000	NaN	NaN
max	NaN	NaN	183.000000	30805.000000	95.000000	NaN	NaN

Afterwards, we have included a box plot to visualize the age range between the diseases Cardiomegaly ,Pneumonia, Infiltration and Plural Thickening, for this particular subset of data we found that the median patient age was in and around 50 years old, which corresponds with the value of 46 given as the mean value of the patient age column in the data.describe(). Furthermore we made a simple pie chart to check the percentage of gender.



left to right(cardiomegaly, pneumonia, infiltration, pleural thickening)



Preprocessing:

To prepare the dataset for feeding, we first had to resize all images to the same size. This step is necessary when feeding the model so it can achieve more accurate results and be used across different learning models. Afterwards, we added the images into a numpy array and we made sure that all labels are correctly assigned, then added to the new numpy array.

Since the data is huge, a normalization was needed to intensities range from zero to one. After normalizing the data and pushing the labels into the array, we used LabelEncoder from Sklearn library and to_categorical from Tensorflow Keras library to convert the values in a categorical

column into numerical values. Now the data is numerical and almost ready to be fed, what was left was binarizing the labels to prepare them for machine learning algorithms,this step will transform our human readable labels into a vector that two hot encodes the classes in the image.

The data now is ready to be fed to the models. Since we are aiming to get the best results, we used Keras Pre-trained Models to flatten and split the data. We have tried various models such as DenseNet121, DenseNet169, VGG19, VGG16, and ResNet50. A pre-trained model is basically a model that has training on a large dataset, in our case, these models have been trained on large scale image classification tasks.

DenseNet (Dense Convolutional Networks) works by connecting each layer to all other layers in a feed-forward way. In other words, it utilizes the dense connections between the layers and also connects all of their feature-maps with each other directly. We used DenseNet121 and DenseNet169 pretrained models in our program, the difference is that the first uses [6,12,24,16] layers while the latter uses [6,12,32,32] layers.

VGG is a pre-trained model that is used for "Very Deep Convolutional Networks For Large-Scale Image Recognition". VGG models take a 224x244 pixel RGB image only to keep the input image size consistent across different data. It has three fully-connected layers and all of its layers uses ReLU. Furthermore, VGG uses small convolutional layers, which are used to transform input images to get its features. It uses small convolutional layers to preserve the spatial resolution after the convolution. In our program, we used VGG19 and VGG16, the main difference is the amount of layers.

The last model we experimented with was ResNet50, which is a convolutional neural network with a depth of 50 layers. The pre-trained model takes the output from a previous later to a later later in order to mitigate the vanishing gradient problem, which makes the network harder to train.

Experiments & Results:

To find the best results, we have used many different learning models such as SVC, XGBClassfiier, BernoulliNB, Random Forest Classifier, and more. To test the accuracy, we used the accuracy score by Sklearn Metrics library as it is accurate and widely popular. The accuracy score by Sklearn takes a set of labels that was predicted and it checks if there is an exact match for the corresponding set of labels in y_valid. Furthermore, we also used The Mean Squared Error from Sklearn Metrics library to measure the average squared difference between the estimated values and true value.

The first conducted experiment is with the DenseNet169 model, with 80% of the dataset as training set and 20% as test set. After flattening each image into a vector of features, we

standardized the dataset because they might behave badly if the individual features do not look like standard normally distributed data. The standardization was done by using StandardScaler from Sklearn library.

SVC (Support Vector Classifier) is used to fit the provided data and then it returns the “best fit” hyperplane that categorizes or divides the data. When we applied the model on our data with RBF kernel and random state =1, the accuracy score of SVC was **0.9165714285714286**, and the mean square error was **0.08342857142857144.**

XGB Classfier (Extreme Gradient Boosting) is based on the decision tree (GDBT) library, it is scalable, distributed, and gradient boosted. XGB offers parallel tree boosting and creates multiple decision trees. Furthermore, it adds weak trees together to generate a collectible strong tree (or model). XGB Classifier performs scanning across gradient values and uses the partial sums to evaluate the quality of splits at every possible split in the training set.

When we applied XGBClassifier into our models with the following parameters: learning_rate=0.69, n_estimators=100, random_state=0, seed=0, gamma=0, the accuracy score was **0.9188571428571428,** and the mean square error score was **0.08114285714285714.**

BernoulliNB is a Naive Bayes classifier that is based on the Bernoulli distribution. As all classifiers, it only accepts binary values as it works with the following formula: P(X=1)=p or P(X=0)=1-p. When we applied BernoulliNB into our model with alpha=10 parameter, the accuracy score was **0.6697142857142857,** and the mean square error score was **0.3302857142857143.**

Random Forest constructs a set of decision trees in parallel from a randomly selected subset from the training set, and then combines all of the trees to reach a single result. The final result or prediction is based on all decision trees predictions. Random Forest Classifier model minimizes the variance and overfitting. When we applied the model with random state =1 and max depth = 18 parameters, the accuracy score was **0.9165714285714286**, and the mean square error score was **0.08342857142857144.**

Cross Validation

We have also conducted a cross validation experiment with more learning models. We have created a function that tests the cross validations using the following models: SVM, Logistic Regression, KNN, Decision Tree Classifier, MLPC Classifier, BernoulliNB, Random Forest Classifier, and finally XGBClassifier. The function getBest gets the cross validation score of the training set, and then applies 10 fold cross validation with different parameters and classifiers and returns the best results.

```
def getBest(classifier, hyperparameters):
    optimum=(0, 0)
    for i in hyperparameters:
        if classifier=='SVM':
            cf = SVC(kernel=i,random_state=1)
        if classifier=='LR':
            cf = LogisticRegression(random_state=1, max_iter=10000, C=i)
        if classifier=='KNN':
            cf = KNeighborsClassifier(n_neighbors=i)
        if classifier=='DT':
            cf = DecisionTreeClassifier(random_state=1, max_depth=i)
        if classifier=='NN':
            cf = MLPClassifier(learning_rate_init=i, random_state=1, max_iter=10000)
        if classifier=='NB':
            cf = BernoulliNB(alpha=i)
        if classifier=='RF':
            cf = RandomForestClassifier(random_state=1, max_depth=i)
        if classifier=='XGB':
            cf = XGBClassifier(learning_rate=0.44, n_estimators=i, random_state=1)


        cvscore = cross_val_score(cf, X_train, Y_train, cv=10).mean()
        if cvscore>optimum[1]:
            optimum=(i, cvscore)
    return optimum
```

The downside of this function is that it takes a lot of time, however, it provides the best results. Below are the results of all the mentioned classifiers.

```
i, cvscore =getBest('SVM', ['linear','poly','rbf'])
print(f'Optimum SVM (has kernel = {i}) and gives a cv score of: {cvscore}')

i, cvscore =getBest('LR', [0.001, 0.01, 0.1, 1, 10,20,50])
print(f'Optimum Logistic Regression (has C = {i}) and gives a cv score of: {cvscore}')

i, cvscore =getBest('KNN', [i for i in range(1,20)])
print(f'Optimum K-Nearest-Neighbors (has K = {i})  and gives a cv score of: {cvscore}')

i, cvscore =getBest('DT',[i for i in range(1, 20)])
print(f'Optimum Decision Tree (has max_depth = {i})  and gives a cv score of: {cvscore}')

i, cvscore =getBest('NN',[0.0001, 0.001, 0.01, 0.1, 1, 2, 3,4,5,10 ])
print(f'Optimum Neural Network (has learning_rate_init = {i})  and gives a cv score of: {cvscore}')

i, cvscore =getBest('NB',[0.0001, 0.001, 0.01, 0.1, 1, 10, 15, 20])
print(f'Optimum Bernoulli Naive Bayes (has alpha = {i})  and gives a cv score of: {cvscore}')

i, cvscore =getBest('RF',[i for i in range(1,20)])
print(f'Optimum Random Forest (has max_depth = {i})  and gives a cv score of: {cvscore}')

i, cvscore =getBest('XGB',[100, 300, 500, 700])
print(f'Optimum XGB Classifier (has number of estimators = {i})  and gives a cv score of: {cvscore}')
```

```
Optimum SVM (has kernel = poly) and gives a cv score of: 0.9273876381498158
Optimum Logistic Regression (has C = 0.001) and gives a cv score of: 0.9273876381498158
Optimum K-Nearest-Neighbors (has K = 10)  and gives a cv score of: 0.9273876381498158
Optimum Decision Tree (has max_depth = 1)  and gives a cv score of: 0.9268162095783874
Optimum Neural Network (has learning_rate_init = 0.0001)  and gives a cv score of: 0.9210994678673762
Optimum Bernoulli Naive Bayes (has alpha = 0.0001)  and gives a cv score of: 0.6846803110929185
Optimum Random Forest (has max_depth = 1)  and gives a cv score of: 0.9273876381498158
Optimum XGB Classifier (has number of estimators = 100)  and gives a cv score of: 0.9265288579615227
```

	CV Score	Accuracy Score	Mean Square Error Score
SVC	**0.9273876381498158**	**0.9165714285714286**	**0.08342857142857144**
Bernoulli Naive Bayes	**0.6846803110929185**	**0.6697142857142857**	**0.3302857142857143**
Random Forest	**0.9273876381498158**	**0.9165714285714286**	**0.08342857142857144**
XGB Classifier	**0.9265288579615227**	**0.9188571428571428**	**0.08114285714285714**

From the results above, XGB Classifier performed the best since it has a relatively good cross-validation score, a good accuracy score and the MSE was the closest to zero , this could be because of it's Gradient Boosting capability, meaning it goes through a cycle: first, it tests the existing models on a test set, then it adds a model to make predictions better and then it tests this new model plus the existing models on a test set again, and this cycle repeats till the solution converges, this makes XGB optimal for large datasets..
On the other hand, Naive Bayes model performed the poorest with a low accuracy score and its MSE is the furthest from zero, this could be because this classifier assumes that all features in the dataset are independent , therefore the probabilities will be incorrect if this assumption is not correct.

Conclusion

All in all, we have combined 2 chestx-ray8 datasets and trained them using several classification models. We have used several techniques for visualization of the data and ran an experiment to check which model performed the best. XGB classifier performed the best with accuracy score of **0.9188571428571428** and MSE of **0.08114285714285714.**
In order to increase performance it may be useful obviously to increase the number of images used in the training, so the whole combined NIH dataset and Covid dataset could've been used without any filtering of the NIH dataset like we did in our project, of course this may lead to an exponential increase in the time taken to train the models.