COMS 4771 Machine Learning (Spring 2020)
Class Project (Kaggle Competition)

Joseph High - jph2185@columbia.edu

May 10, 2020

Methodology

For this project, the primary methodology used to diagnose COVID-19 patients, based on X-ray images, was XGBoost. While this is probably an unconventional approach for an image multiclass classification problem, it was initially believed that XGBoost would have performed better than alternative methods. Moreover, I wanted to gain additional insight and experience into the XGBoost methodology given the acclaim that the algorithm has received over past 4 years. It has become the go-to approach for classification problems in several industries and Kaggle competitions.

Theory

"Extreme Gradient Boosting" (XGBoost) is an implementation of the gradient boosting framework. More specifically, XGBoost is a supervised learning algorithm which uses an additive strategy (boosting) rather than a parallel (bagging) ensemble decision tree. Ensemble methods are learning models that achieve performance by combining the predictions/scores of multiple different learners. This method in particular starts with a rough prediction and then iteratively constructs updated decision trees, which are added to the previous decision tree classifer (hence, the name *additive*). Each updated tree in the series emphasizes data that was mis-modeled in the previous iteration. A high-level overview of the algorithm's underlying mathematics is provided below.[1]

The final prediction of the i^{th} instance from an additive model is

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in F$$

where K denotes the number of trees, x_i denotes the predictor variable for the i^{th} instance, and F denotes the space of functions containing all of the decision trees (also known as Classification and Regression Trees (CART)).

[1] Chen, T.(2016), *XGBoost: A Scalable Tree Boosting System.*

For a set of hyperparameters, λ, the objective function for the XGBoost classifier is defined by

$$obj(\lambda) = L(\lambda) + \Omega(\lambda)$$

where L is the loss function and Ω is the regularization term. The loss function and regularization term or chosen based on the problem. Because the goal was to develop a multiclass classifier, the softmax function (or normalized exponentional function) was used as the loss function.

Alternative classification models were trained the image data, including variations of the VGG16 model provided in Keras. XGBoost was ultimately chosen due to its superior performance and interpretability.

Alternative Approaches

The pre-trained VGG16 model provided in the Keras API was also trained on the data using both the Adam optimizer and the SGD (Stochastic Gradient Descent) optimizer. However, the model did not perform as well as XGBoost.

I chose XGBoost over VGG16 (and other CNN methods) for the reasons listed above.

Pre-processing

In order for the algorithm to be able read and process the training data, the dataset had to first be converted into a numpy array data type. In doing so, the images within the dataset were re-shaped, flattened, and scaled to the same size (or normalized).

Three different pixel sizes were evaluated to determine what would work best. In particular, $224 \times 224 \times 3$, $229 \times 229 \times 3$, and $255 \times 255 \times 3$ were evaluated. It was found that $224 \times 224 \times 3$ resulted in the highest accuracy for XGBoost.

Hyperparameter Tuning

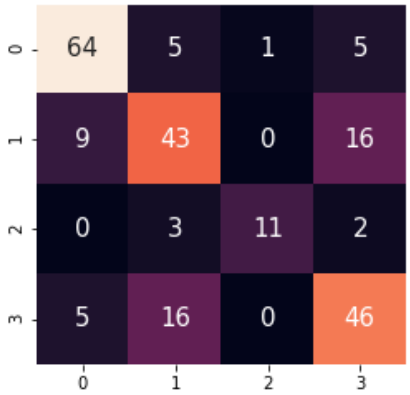
Random search was used to compute optimal hyper-parameters using the built-in function `scikit-learn.model_selection.RandomSearchCV`. This particular method randomly samples hyper-parameters from a pre-defined interval and subsequently validates model performance on a holdout dataset. Compared to the grid search approach, the chances of finding the optimal parameter are comparatively higher in random search because of the random search pattern where the model might end up being trained on the optimized parameters without any aliasing.[2]

[2] Source: https://analyticsindiamag.com/why-is-random-search-better-than-grid-search-for-machine-learning/

This approach was performed using stratified five-fold cross validation, where 100 randomly sampled combinations of 9 hyperparameters were tested on each fold, resulting in a total of 500 model specifications. Early stopping was applied to further prevent overfitting.

Error Analysis

Figure 1: Confusion Matrix



Interpretability

While the interpretability of Decision Tree methods are somewhat lost when using XGBoost, it is still more interpretable than, say, a convolutional neural network. That said, other than the pixels, it is not clear as to what features in particular the model is using to classify images.

Issues & Lessons Learned

I experienced extremely long run-times for XGBoost, while VGG16 ran relatively quickly. Hyperparameter optimization for XGBoost required more than 2 days to run. It would have helped if I narrowed down the parameter distribution (i.e., the feasible region) in advance in order to speed up the tuning process.

Another issue I faced was pre-processing the image data. I had quite a bit of difficulty getting the data into the right format for every step.