Dharma Naidu
Vikash Singh

Classification of Brain Hemorrhage

Methods:

Python’s Scikit Learn library, one of the most comprehensive open source machine learning libraries available, was used to write scripts to build and validate the performance of multivariable prediction models. The data used in this machine learning pipeline comprised of 50,000 instances extracted from MRI (magnetic resonance imaging), each representing a pixel from 62 unique patients. The parameters which ultimately translated to the feature space for various machine learning algorithms consisted of information about blood flow, and quantitative measures derived from the MRI. The binary label which each classifier attempts to predict is whether or not each pixel bled. Rather than building a classifier at the patient level, machine learning was done at the pixel level, which could potentially link to the patient level for hemorrhage predictions at the patient level. Several different algorithms were compared in these experiments, specifically Logistic Regression, Gaussian Naive Bayes, Random Forest, and Deep Neural Networks. Each of these algorithms has its own advantages and disadvantages, and each model built using one of these algorithms was designed to exploit a specific advantage of the model. We paid particularly close attention to the use of neural networks, as neural networks are strong general-purpose machine learning problem solvers. These networks represent another family of classifiers that have recently received attention both in and out of the medical field for impressive performances in a wide variety of predictive modeling challenges. The Keras Python package using the Theano backend was used to create neural networks, allowing for the ability to experiment with the architecture of the network, (depth, nodes per layer), as well as the activation functions used in each layer. The neural network was the only algorithm that specifically required parameter tuning, which involved experimentation with the depth of the network and activation functions combined with an intuition about certain architectures. Training the neural network required a tuning process as each neural network architecture could be differently suited to address different classification problems and data sets. Originally a simple 3 layer architecture (10/5/1) was used, and different activation functions in each layer were tested to begin the tuning process. Using the sigmoid activation yielded the best results, primarily due to the instability (AUC in some folds being <.5) while using the tanh and rectified linear (relu) activation functions. Furthermore networks were tested in which activation functions were mixed between the layers, specifically using mostly sigmoid layers, while mixing in layers with rectified linear activation functions. Mathematically, using such layers could be problematic as the output from the rectified linear function could overpower the sigmoid function since the target space is not compressed for large positive numbers, however the optimal network found contained six layers, with the second utilizing a relu activation. From there, experimentation was done to tune the neural network manually. Although a computerized grid search algorithm could be used to automate the process of finding the optimal parameters, this was not able to be done in a time efficient manner given the size of the dataset. While tuning the neural network, there was a balance in finding the depth of the network to provide the optimal AUC since it is known that when using sigmoid activation functions within a deep neural network, the gradient vanishes through multiple iterations, which impacts the backpropagation algorithm and gradient descent optimization process in a negative way (Hochreiter, 1998).

Evaluation:

The first set of scripts written using each of the algorithms employed a stratified 10-fold cross validation approach, in which the entire data set was repeatedly randomly split into ten folds, trained on nine of them, and tested on the final fold. After analyzing the seemingly stellar performance of these models, it was addressed that using this validation mechanism was flawed in that pixels corresponding to a particular patient could appear in both the training and test sets, inducing some bias. To address this bias, a new ten fold cross validation method was custom implemented from scratch to randomly sample the minimum number of patients (and all corresponding pixels) to reach at least 5000 instances for the test set, and store all the remaining instances corresponding to completely different patients in the training set, thereby ensuring no patient overlap could occur in the test and training set. In order to evaluate the performance of the multivariable prediction models, a Receiver Operator Characteristics curve (ROC) was plotted (false positive rate vs true positive rate), and area under the curve (AUC) was used as a measure of merit for these models. Although ten different models were created in this validation process, a single ROC curve and AUC was generated for each algorithm by concatenating the probabilistic predictions and their corresponding labels across all the folds, generating two global lists containing prediction information for all ten models created. The AUC was initially compared across all algorithms, both in the flawed validation experiments and the custom validated experiments. For the neural networks trained with custom validation, AUCs were measured for the different architectures being examined to understand the optimization of the neural network architecture.

Results:

Model Type	AUC Without Patient Splitting	AUC With Patient Splitting
Logistic Regression	0.925610051063	0.663202112522
Gaussian Naive Bayes	0.743488909174	0.739214684458
Random Forest	0.98923585522	0.843985730228
Best Neural Network	0.93080618173	0.866178680636

Table 1: AUCs of Multiple Models

Activation Function	AUC
split (sigmoid, relu)	0.892789867271
sigmoid	0.843268416341
tanh	0.67932306661
Rectified Linear (relu)	0.59614529024

Table 2: AUCs of Neural Network based on Activation Function

Neural Network Type	AUC with Patient Splitting
4 layer	0.84788478635
6 layer	0.866178680636
9 layer	0.799608862626

Table 3: AUCs of Neural Networks by Number of Layers

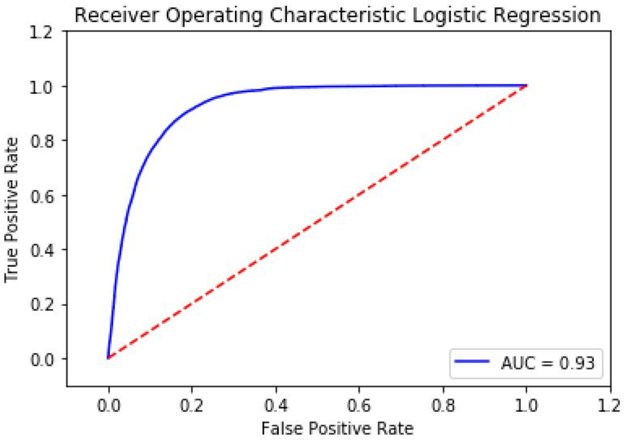

Figure 1: ROC Graph for Logistic Regression without Custom Splits

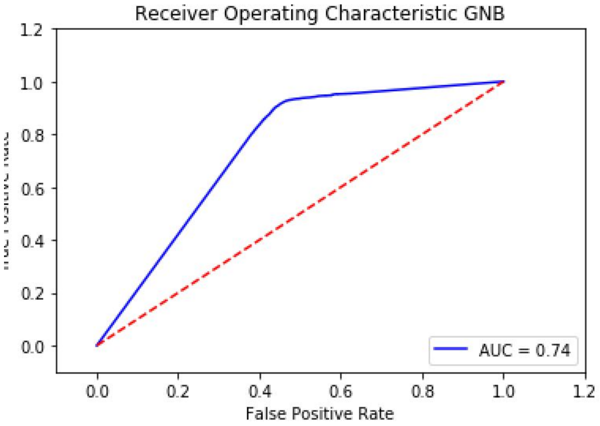

Figure 2: ROC graph for Gaussian Naive Bayes without Custom Splits

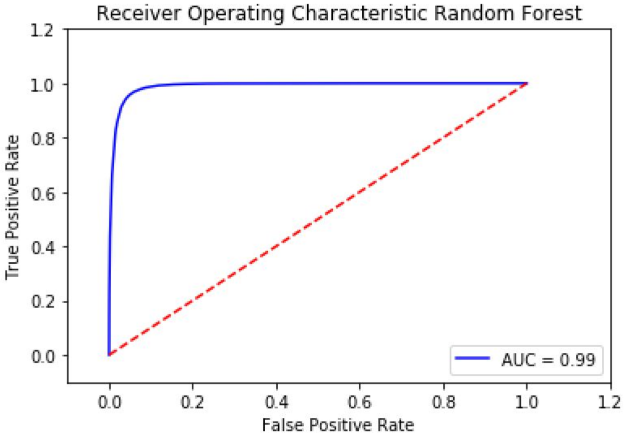

Figure 3: ROC Graph for Random Forest Without Custom Split

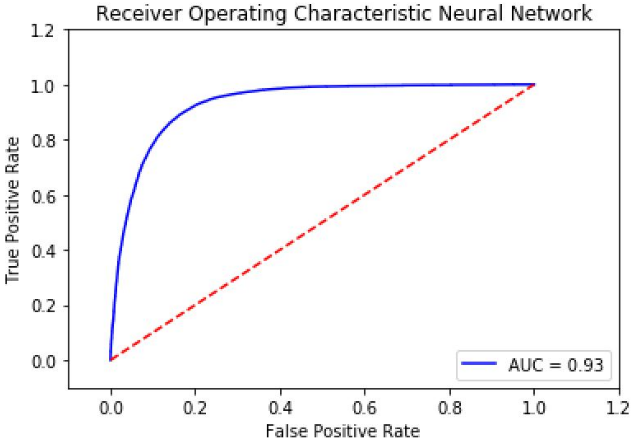

Figure 4: ROC Graph for Neural Network without Custom Splits

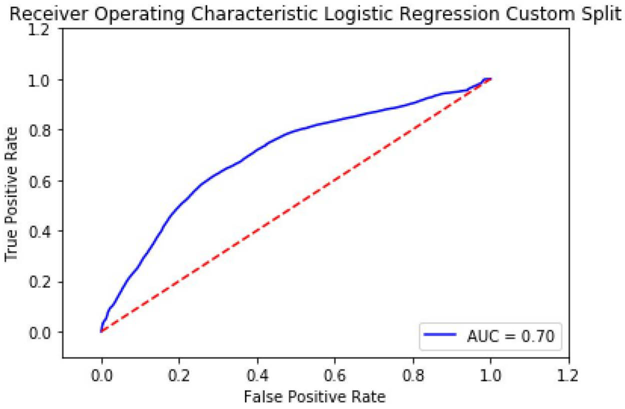

Figure 5: ROC Graph for Logistic Regression with Custom Splits

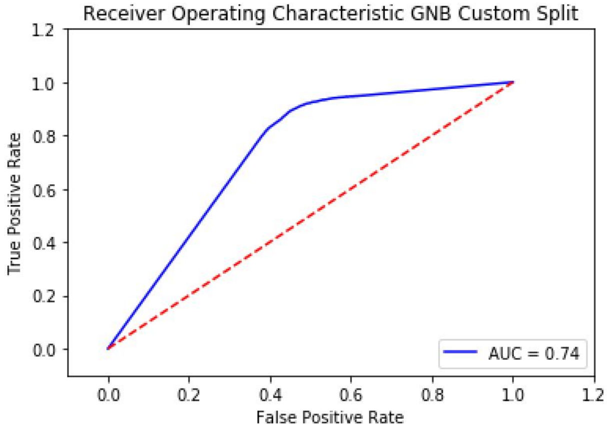

Figure 6: ROC Graph for Gaussian Naive Bayes with Custom Splits

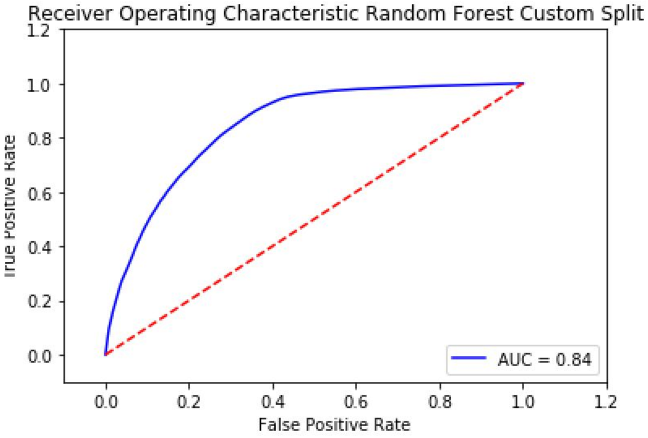

Figure 7: ROC Graph for Random Forest with Custom Split

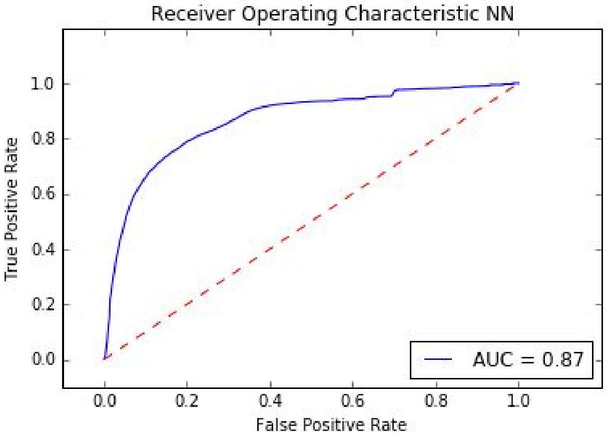

Figure 8: ROC Graph for Neural Network with Custom Split

Discussion:

As one of the primary motivations for this project was to examine the performance of neural networks, particularly deep neural networks to this bleeding classification task at the pixel level, and compare it to several other widely used algorithms. Initially when the custom splitting was not implemented and the cross validation procedure was vulnerable to the patient overlap bias, the random forest yielded an AUC of ~.989, which seemed too good to be true (Table 1, Figure 3). When the custom validation was implemented and run to prevent patient overlap, the performance of all models declined. The optimal neural network finally found after tuning yielded an AUC of ~.87, demonstrating a superior performance compared to all other tested algorithms (Table 1, Figure 8). After finding an activation function which yielded stable results across folds, the process of tuning of deep neural network involved discovering a balance in the depth of the neural network such that the negative impact of vanishing gradient was outweighed by increased abstraction through the layers (Table 2, 3). Dropout (ignoring random neurons in feedforward process) within our deep neural network was also applied in first couple layers in order to prevent the network from overfitting and serve as a mechanism for regularization (Srivasta et. al,2014). Although random forest again did perform reasonably well, it did not nearly reach its stellar performance prior to patient splitting, leading to the conclusion that it was overfitting originally, perhaps based on a link between patients. Knowledge of how a random forest works further elucidates this as a possibility since branching within the trees can conditionally be done on parameters that have heavy overlap between the patients. While the experiments yielded to a neural network with good performance at the pixel level of classifying bleeding, one discussed goal was to translate the performance of such a classifier to the patient level for direct clinical use. One mechanism which would be great to explore in the future would be to calculate the predicted number of bleeding pixels per patient based on the deep neural network built in this project and concatenate that as a parameter to another machine learning algorithm (perhaps another neural network) containing the clinical information about each patient as well the binary label of whether that patient has a hemorrhage or not. Further measures could absolutely be taken to continue to build the optimal classifier and could potentially include transformations of the feature space using techniques such as PCA, and also creative feature engineering based on domain knowledge of the physiological basis of hemorrhage. This approach would be an interesting way of embedding machine learning at the pixel level into machine learning at the patient level.

References:

Amato F, López A, Peña-Méndez E , Vaňhara P, Hampl A, Havel J. Artificial neural networks in medical diagnosis. *J Appl Biomed*. 2013;11:47-58

Hochreiter S. The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. International Journal of Uncertainty Fuzziness and Knowledge-Based Systems. 1998;6(2):107-116

Srivasta N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research.* 2014;15:1929-1958