ECS 271 - Final Project Report

Kasaraneni, Namrata
nkasaraneni@ucdavis.edu

Punj, Siddhartha
sipunj@ucdavis.edu

Sen, Baris
bsen@ucdavis.edu

March 14th, 2022

1 Introduction

In Machine Learning, a data scientist's understanding of learning models, as well as their ability to choose good models and parameters for a specific dataset, is as powerful as the models themselves. We explored how different classifiers perform on multi-class datasets, using classifiers that we implemented ourselves from the ground up. We designed and fine-tuned these models to classify fetal health data. The goal of our project is to compare and contrast different classifiers for determining which is the best for correctly classifying the fetal health of a child. This is in order to help preemptively identify fetuses at risk and lower child and maternal mortality.

The Fetal Health Classification dataset we used contains data taken from cardiotocograms (CTGs), which read the responses of ultrasound pulses sent through the uterus [2]. The data contains 2126 records that were classified by expert obstetricians into three classes: normal, suspect, and pathological. CTGs are a cost efficient way to assess the health of a fetus, and in countries where access to expert obstetricians cannot be guaranteed, having a simple pre-trained model identify at-risk fetuses using data from CTGs is a feasible way to save lives.

For this multi-class problem, we implemented three learning models: a simple Fully Connected Neural Network as a baseline, a Random Forest [4], Gradient Boosting [5].

2 Dataset

An important fact about the dataset is that the target classes are imbalanced. Figure 5 in Appendix shows the histogram of the target labels. We see that the data points belonging to pathological cases are rare. This makes the problem especially hard because a simple classifier that classifies all data points as normal would have a high accuracy. Thus, in our analysis, we use metrics such as recall and precision and ROC curves to evaluate our models. We interpret meaning of high recall and high precision in the specific case of this dataset.

Furthermore, we generated a new version of the dataset by oversampling the data belonging to the minority labels (only in the training set). This makes the data more balanced and we hypothesize that it would help the models to put more importance to classifying the minority labels correctly. In Section 5, we compare this version of the dataset with the original version.

3 Methods

3.1 Random Forest

Both the Random Forest and the Decision Tree were implemented in Python. Decision Trees are the building blocks of Random Forests. These Decision Trees found the best feature boundaries for branching continuous data by minimizing the Gini Impurity. The Decision Tree leaves were modified to store the percentage of the data subset that belonged to each class, so that the leaf could return a probability of an input belonging to a certain class.

For Decision Trees used in a Random Forest, at every node the Decision Tree considers a random subset of features instead of looking at all the features. Random Forests input randomly sampled datasets to train each Decision Tree. Data points are classified by taking a vote amongst all the trees that compose the Random Forest.

3.2 Gradient Boosting

Gradient starts with a single leaf containing the predicted output of a data point. In gradient boosting classification, this first guess utilizes the logit equation 2 in Appendix with regards to the number of positive and negative samples, and then it is converted into a probability using the logistic function.

Then, it builds a tree that is based on improving errors made by the last tree. These errors are known as pseudo residuals. Each training point maps to a leaf on the tree made, an output is calculated for that point by taking the sum of residuals divided by the previous probability for that training point in Equation 1.

$$\frac{\sum \mathit{Residuals}}{\sum[\mathit{PreviousProbability} * (1 - \mathit{PreviousProbability})} \tag{1}$$

This is converted back to a probability using the logit function again. Finally, we add the previous probability to the learning rate multiplied by the above output, for each training point, to make a probability prediction. Then, we repeat this process for all boosting rounds.

In particular, since there is no easy way to convert the leaf outputs to probabilities, we used the concept of one vs rest to create 3 different binary classifiers, and essentially took the softmax of each data point in the test set.

3.3 Neural Network

We use a fully connected neural network consisting of linear layers and non-linear activation functions. Even though we started using batch normalization [6] layers after each linear layer, we removed these layers from the network after observing an increase in the final loss value relative to the models that does not have batch normalization. To make the hyperparameter tuning easier, we implement the model and the main training loop of the network configurable. We use the PyTorch [7] framework for our implementation.

Because all features in our dataset are numerical values, we do not need to preprocess features before feeding them into the model. If we had for example categorical features, we might have needed to convert them to one-hot encoded features or feature embeddings.

4 Hyperparameter Tuning

All of the methods we used for the project have hyperparameters. We used the hyperparameter optimization framework `Optuna` [1] for tuning the hyperparameters of the models. One advantage of the `Optuna` library is that it uses more advanced algorithms on how to sample hyperparameter values than random search or grid search. In our script, it uses the TPE (Tree-structured Parzen Estimator) [3] algorithm for sampling hyperparameters and Median pruner for pruning the unsuccessful branches in hyperparamter search. These algorithms help the optimizer to focus on the ranges of hyperparameters that are more likely to result in a better accuracy.

We split our data to 60% training, 20% validation, and 20% test sets. For hyperparameter tuning, we trained our models on the training set and calculated accuracy on the validation set.

4.1 Random Forest

Random Forests have three hyperparameters: the number of trees in the forest, the number of features considered at every branch of a Decision Tree, and the depth limit of a Decision Tree. The best values for the hyperparameters are shown in Table 2. This model achieved a 91% accuracy on the validation data.

4.2 Gradient Boosting

For tuning a gradient boosting model, tuning the right hyperparameters is essential in order to get categories classified correctly- pushing training points slowly in the right direction. We considered max depth of the tree considered, number of leaves in the tree, learning rate, and number of boosting rounds (essentially epochs). Reference Table 3 for the best hyperparameters in each binary classification model.

4.3 Neural Network

One of the major challenges of neural networks is the large number hyperparameters. For our fully connected neural network, we performed tuning on the following hyperparameters: batch size, learning rate, weight decay coefficient, optimizer method, activation function, and number of layers. Table 4 shows the range of hyperparameters and the best performing configuration.

5 Evaluation

Figures 1 and 2 show the confusion matrices of each model. We see that the oversampled data results in significantly better recall and precision scores in the pathological and suspect labels. When using the original dataset, it is in all models more likely to classify a suspect case as normal, where with the balanced dataset, the suspect cases are classified significantly more accurately. We see that oversampling also improves the performance in the pathological class.

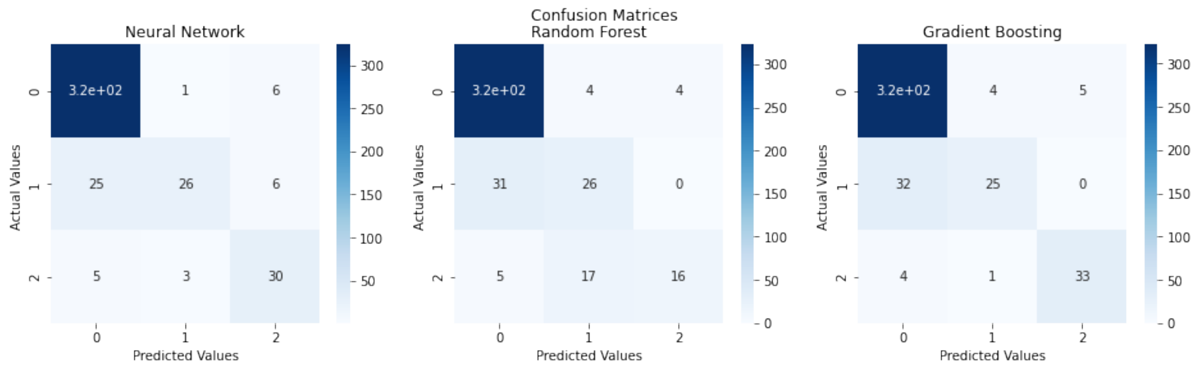

Figure 1: Confusion Matrices

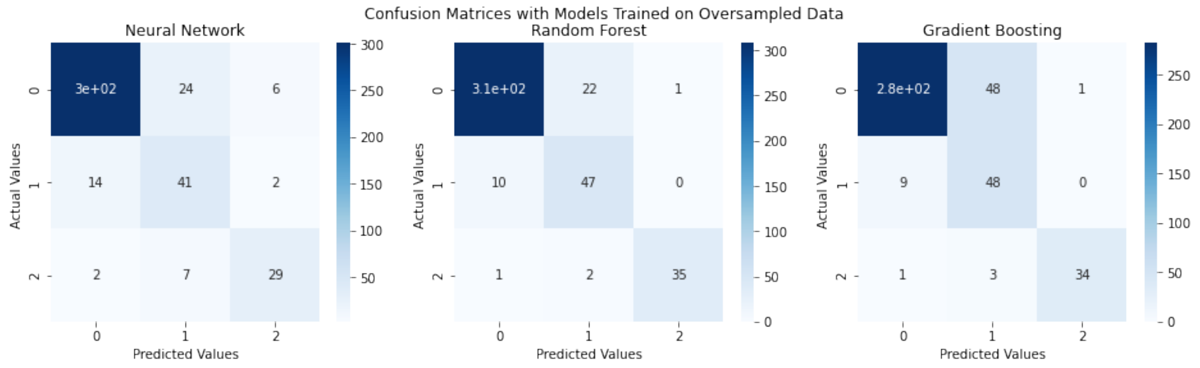

Figure 2: Here we can see how the confusion matrices changed after the models were trained on a dataset that included oversampling of the minority classes.

Table 1 shows the evaluation metrics of the models. We see that the neural network model and the gradient boosting model have the highest accuracy. However, because the dataset is an imbalanced, the accuracy can a be misleading metric. A classifier that classifies all cases as "normal" would have about 75% accuracy. Thus, we concentrate more on the precision and recall metrics in the "pathological" class.

Because we are dealing with medical data, an ideal model will identify 100% of pathological fetuses as pathological. In Figure 3 and 4, we plotted ROC curves for all three models classifying the pathological subset of the data. We see that using the balanced version of the dataset results in better ROC curves in all of the models. We find that the Random Forest is the best classifier for having 100% recall, with the lowest rate of false positives at only 2%. In comparison, Gradient Boosting and

Table 1: Model Metrics

Model	Dataset	Accuracy (%)	F1 Score	Precision (%)	Recall (%)
NN	Regular	89.20	0.75	71.43	78.95
	Oversampled	87.1	0.77	78.4	76.3
RF	Regular	85.69	0.55	80.0	42.11
	Oversampled	**91.5**	**0.95**	**97.2**	**92.1**
GB	Regular	89.20	0.86	86.84	86.84
	Oversampled	85.4	0.93	97.1	89.5

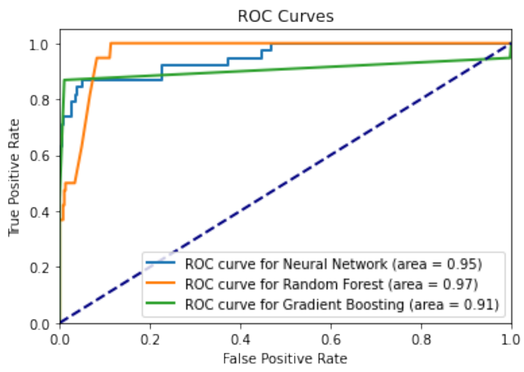

Figure 3: Receiver Operating Characteristic (ROC) curves of the models for the pathological subset of the data

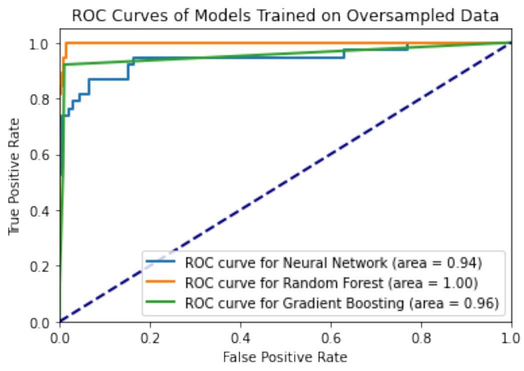

Figure 4: ROC curves of the models for the pathological subset of the data with models trained on oversampled minority labels

Neural Networks cannot guarantee 100% recall for pathological cases with a reasonable amount of false positives.

6 Conclusion

In Machine Learning, our priority is not always to optimize the accuracy of models. With a medical problem such as this one, the most important attribute of a model is its ability to always correctly identify pathological cases, while minimizing the number of false positives. Before oversampling the data, Random Forest had the worst metrics of the three models in all categories. After oversampling, Random Forest had the highest scoring metrics for accuracy and F1 scores. In both cases, however, Random Forests were the ideal model for identifying 100% of pathological cases while minimizing false positives (Figures 3 and 4). Thus, we see that there are multiple ways to interpret the performance of a machine learning model and the importance of metrics can vary depending on the specific use case.

Our results make us question our inherent assumption that this problem is multi-class. Because the three classes (healthy, suspect, and pathological) have an inherent relation to one another, and are not all equidistant, regression models might have performed better on this dataset. The question can be reformulated as a regression problem where the models would predict a value between 0 and 1 for determining how likely it is that the test case belongs to a pathological case. The results would be interpreted using different thresholds to decide on the actual class of the cases.

7 Contributions

Namrata Kasaraneni built and tuned the Decision Trees and Random Forest, and identified the Fetal Health dataset. Siddartha Punj built and tuned the Gradient Boost classifier, and modified it to fit the multiclass scenario. Baris Sen analyzed the Fetal Health dataset, built and tuned the neural network, and created an interface to compare the models. All three group members wrote the final report together. Our codebase can be found at: https://github.com/GSWarriors/ECS271-project

References

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery; Data Mining*, 2019.

[2] D. Ayres-de Campos, J. Bernardes, A. Garrido, J. Marques-de Sá, and L. Pereira-Leite. Sisporto 2.0: A program for automated analysis of cardiotocograms. *The Journal of Maternal-Fetal Medicine*, 9(5):311–318, 2000.

[3] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[5] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[6] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

A Appendix

A.1 Equations

$$logit(p) = log(\frac{p}{1-p}) \tag{2}$$

A.2 Data

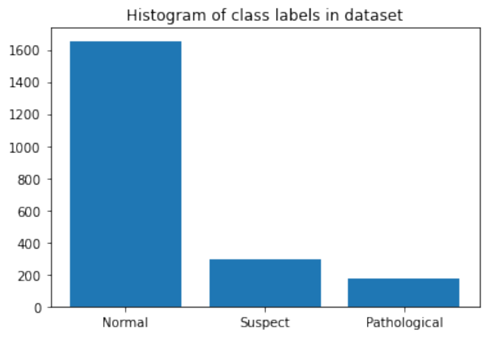

Figure 5: Histogram of labels in the dataset

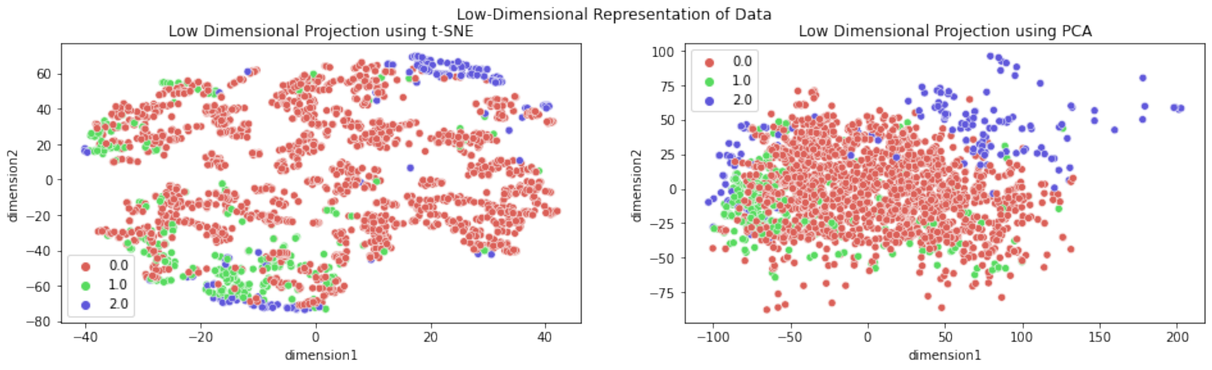

Figure 6: Plotting the dataset on a 2D plot using dimensionality reduction.
We use the common dimensionality reduction techniques to visualize the data points in 2D such as PCA and t-SNE. Even though we see some clusters, the 2D representations cannot separate those clusters from each other.

A.3 Hyperparameter Tuning

Hyperparameter	Sample Distribution	Range	Optimal Value
Number of Trees	Uniform	[50, 150]	143
Number of Features	Uniform	[2, 21]	9
Depth Limit	Uniform	[1, 40]	11

Table 2: Random Forest Hyperparameters

Hyperparameter	Sample Distribution	Range	Optimal Value
Depth Limit	Uniform	[3, 4]	3, 3, 3
Learning Rate	Log-Uniform	[1e-6, 1]	0.01, 0.9, 0.4698
Number of Leaves	Uniform	[3, 8]	8, 4, 6
Number of Boosting Rounds	Uniform	[1, 5]	5, 1, 4

Table 3: Gradient Boosting Hyperparameters (each classifier starting from class 1 vs rest and going in numerical order)

Hyperparameter	Sample Distribution	Range	Optimal Value
Batch Size	Categorical	[1, 4, 16, 64, 256]	4
Learning Rate	Log-Uniform	[1e-4, 1e-1]	0.0003
Weight Decay	Log-Uniform	[1e-5, 1e-3]	1e-5
Optimizer Method	Categorical	[Adam, SGD, Adagrad, RMSprop]	Adam
Activation Function	Categorical	[PReLU, ReLU, LeakyReLU, Tanh, Sigmoid]	PReLU
Number of Layers	Categorical	[3, 5, 7]	7

Table 4: Neural Network Hyperparameters