Revisiting Deep Artificial Neural Networks for the Diagnostic of

Caries Using Socioeconomic and Nutritional Features

as Determinants: Data from NHANES 2013–2014

Jong Won Lee, Juntian Wang

12/10/2023

Intro to Machine Learning

**Introduction**

The current project aims to recreate the Zanella-Calzada et al. (2018) using *NHANES 2013–2014* to investigate how demographic and socioeconomic factors, and nutrition states can predict one’s physical wellness, using cavities as an observable variable. This was because oral health is often the first type of health service that gets neglected under economic pressure.

For this project, we decided to use Python 3.10 based virtual IDE (Google Colab) and used SQLite and Python to perform data cleaning for the analysis. For model fitting, Random Forest model was fitted and tuned using R’s randomForest packages to fit the model and Caret package was used to perform hyperparameter tuning on the number of nodes. The Random Forest model was completed in a local machine using R Studio. Deep Neural Network (Multi-Level Perceptron) was fitted using the Keras package from Tensorflow via Python 3.10.

We chose NHANES dataset from 2013 to 2014 from Center of Disease Control (CDC) because the dataset was easily accessible as CDC uploaded this dataset to the popular data archive (i.e., Kaggle) in csv format. This suggests that there are technical resources readily available, including but not limited to hyperparameter tuning, fits of the various models and their performance, and various suggestions for the neural network layer structures.

**Method**

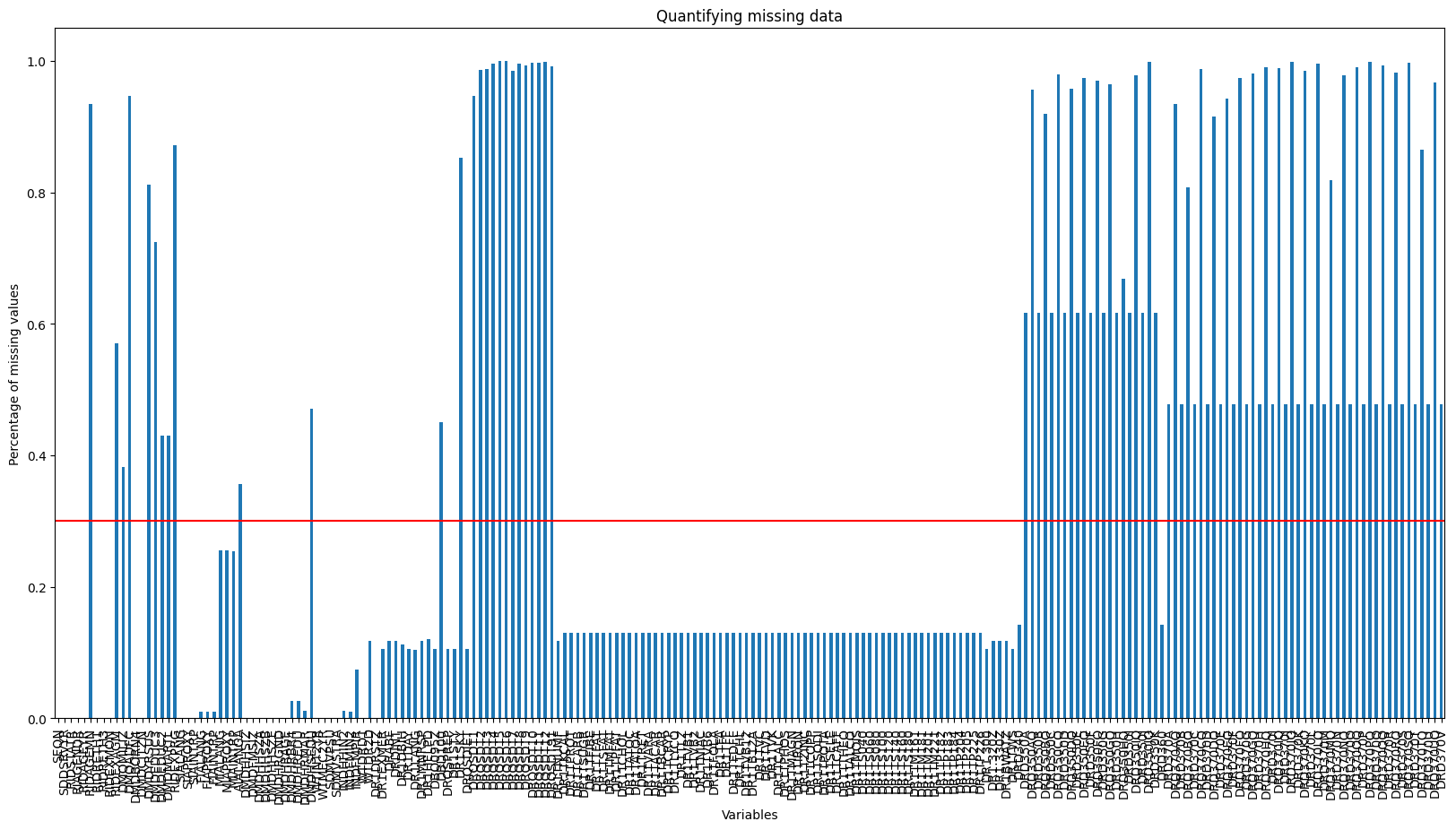
Initially, we uploaded the dataset into GitHub (link:<https://raw.githubusercontent>[.com/  
jwlee96/ML\_NHANES13\_14/main/](https://raw.githubusercontent.com/%0bjwlee96/ML_NHANES13_14/main/)) to access the dataset virtually. As there are multiple datasets (e.g., demographics, diet, examination, labs, medications, and questionnaire), we used SQLite to store original copy in the database format (.db) to query across datasets. The datasets that the Zanella-Calzada et al. (2018) used was Demographic and Dietary dataset from NHANES 2013-2014. SEQN variables were used as a primary key since these were unique patient identifiers. After merging both Demographic and Dietary dataset using SEQN, there were 9,813 subjects in total.

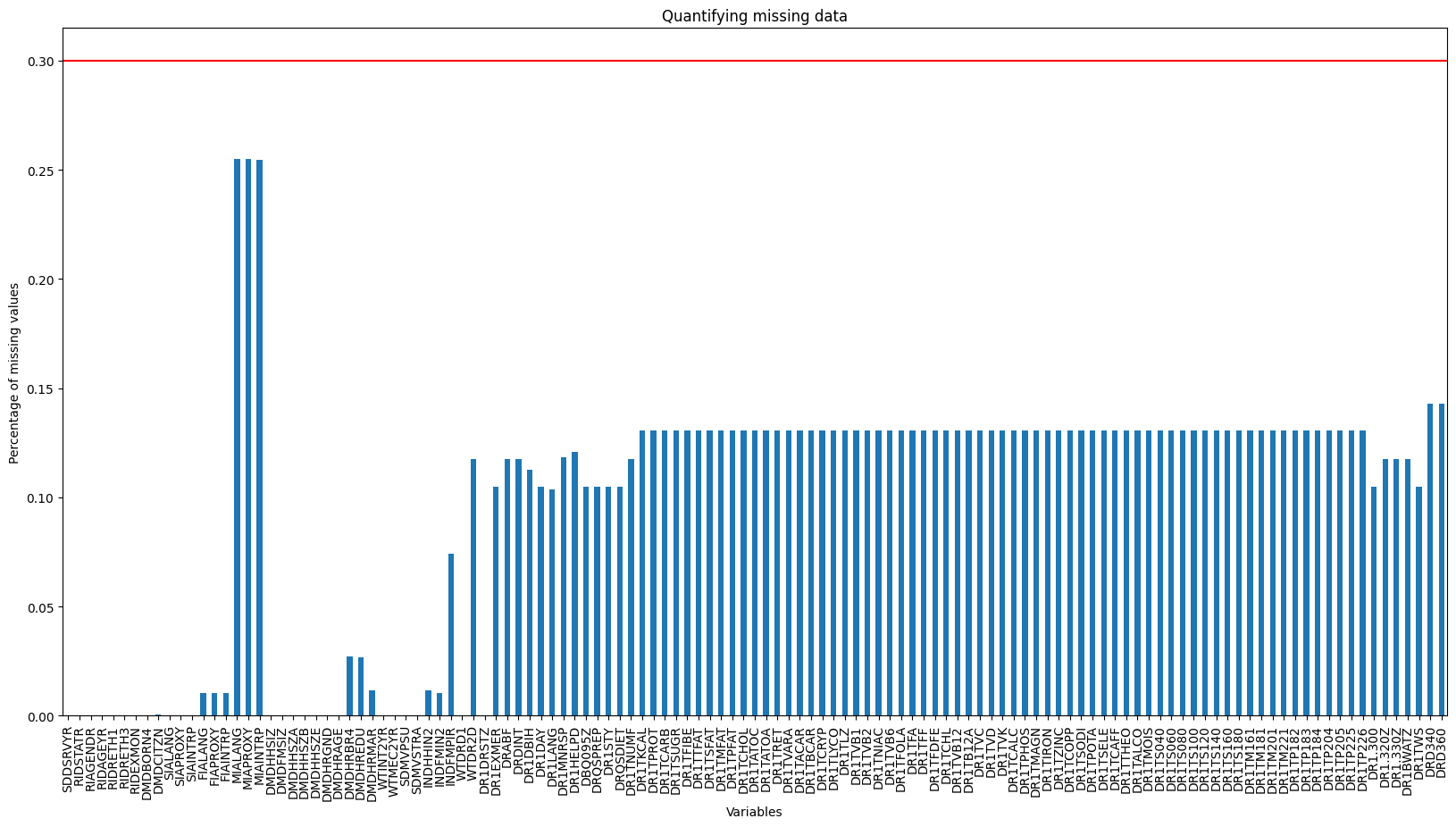
**Dependent Variable** As the acquisition process of the dependent variable “Cavities” (labels from this point beyond) were not explicitly mentioned beside that it is a binary classifier from the Diagnostics dataset. Therefore, we decided to extract all the teeth condition data (i.e., Variable with CTC string) and observed that the value D and S indicated healthy teeth, U means teeth that never developed, Q and M for missing due to non-dental disease related condition, Y for missing due to unknown issue, and K and Z where it is just a surface condition where it cannot be fully diagnosed as cavities that requires treatment. If all existing dental records of each patient’s teeth does fall under one of these categories, then they are considered as 1: healthy dental hygiene. If not, then it is 0:unhealthy dental hygiene, as remaining categories are various type of damages due to dental disease (i.e., cavities). There were 6495 healthy cases and 3318 unhealthy cases after this classification.

*Data Cleaning Process*

**Data Cleaning** We started with 213 features extracted from the Demographic and Dietary dataset of NHANES 2013-2014, and as the Zanella-Calzada et al. (2018) suggested, we dropped all the features that had more than 30% of their data missing, which eliminated 90 features (reference Figure 1). Secondly, for the missing data points we imputed using the average as well, using Simple Imputer Python packages. After the data cleaning process, we ended up with 123 features with 9813 observations, instead of 104 features as the original authors suggested. As we are planning to fit the dataset into an artificial neural network, we performed Z-score standardization on the features to eliminate the possible error caused by scale unit difference amongst features.

**Data Cleaning process visualization**Comment: Missing data percentages are the bars.





Random Training/Test dataset separation was performed with scikit-learn Python packages’ train-test separation function with 70:30 train to test ratio. After this separation, there were 6,869 subjects in the training dataset and 2,944 subjects in the test dataset.

**Random Forest Model**

As this is a binary classification task, we decided to fit the Random Forest model as a non-neural network machine learning model, so that we can compare and contrast it to the neural network model. Out of all the machine learning models, we selected the Random Forest model due to the fact that we have a lot of features (123 features) and the possible high dimensional nature of the dataset, as this dataset was collected from various individuals with different ethnic/socioeconomic backgrounds. As the Random Forest model is a tree based ensemble model, they will be able to process unique characteristics of each feature effectively to create the predictive model. For the Random Forest model, we initially attempted to use Python 3.10, but ended up using R because package support and processing speed was far better in R than Python. With R, we used randomForest packages to carry out Random Forest model fitting and Caret package for hyperparameter tuning.

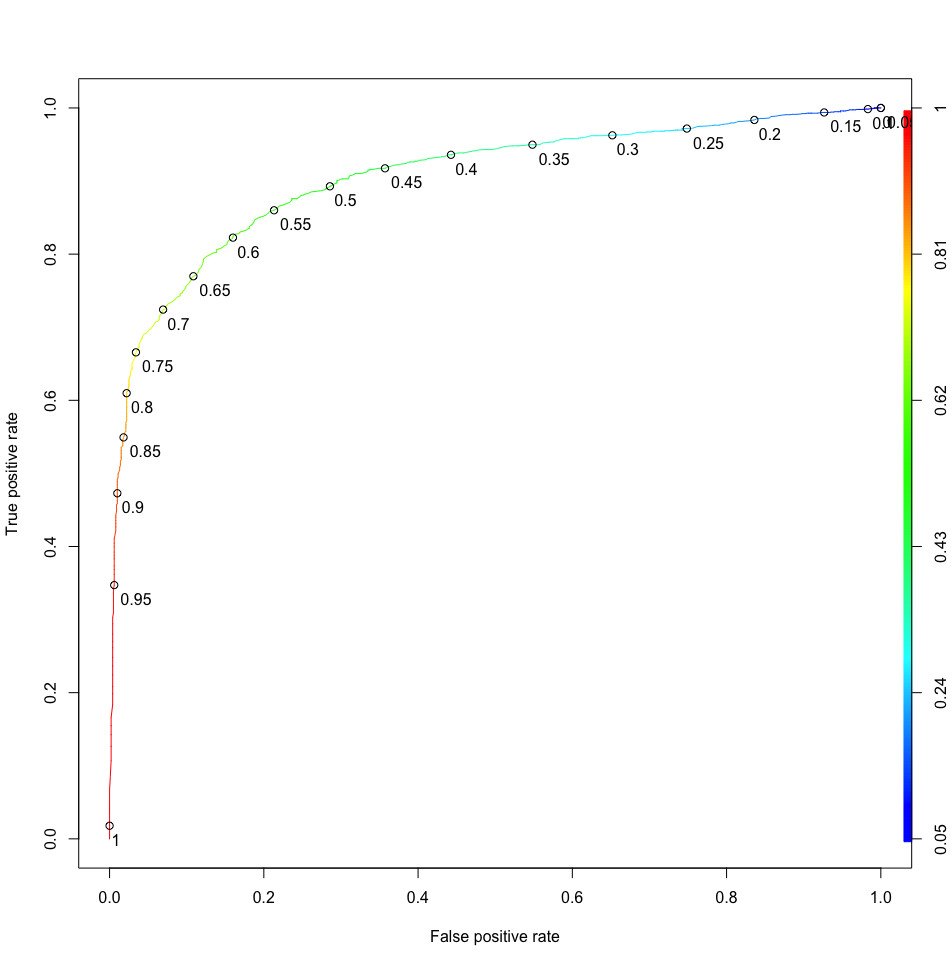
*-Initial Model Fitting*

Random Forest Classifier function in R packages in R uses default setting of number of trees being 500, M\_try of , where p refers to the number of features in the dataset, meaning it will be 12 as there are 123 features. For the maximum possible tree depth (max depth) and required node size, we used Python's GridSearchCV result of 10 max depth, and 5 node size. The initial model AUC score from the ROC curve was 0.91, using a best probability cutoff of 0.53 from the ROC curve, with the model’s prediction accuracy of 0.84. Using best probability cutoff, we created a confusion matrix below.

**Confusion Matrix of initial model**

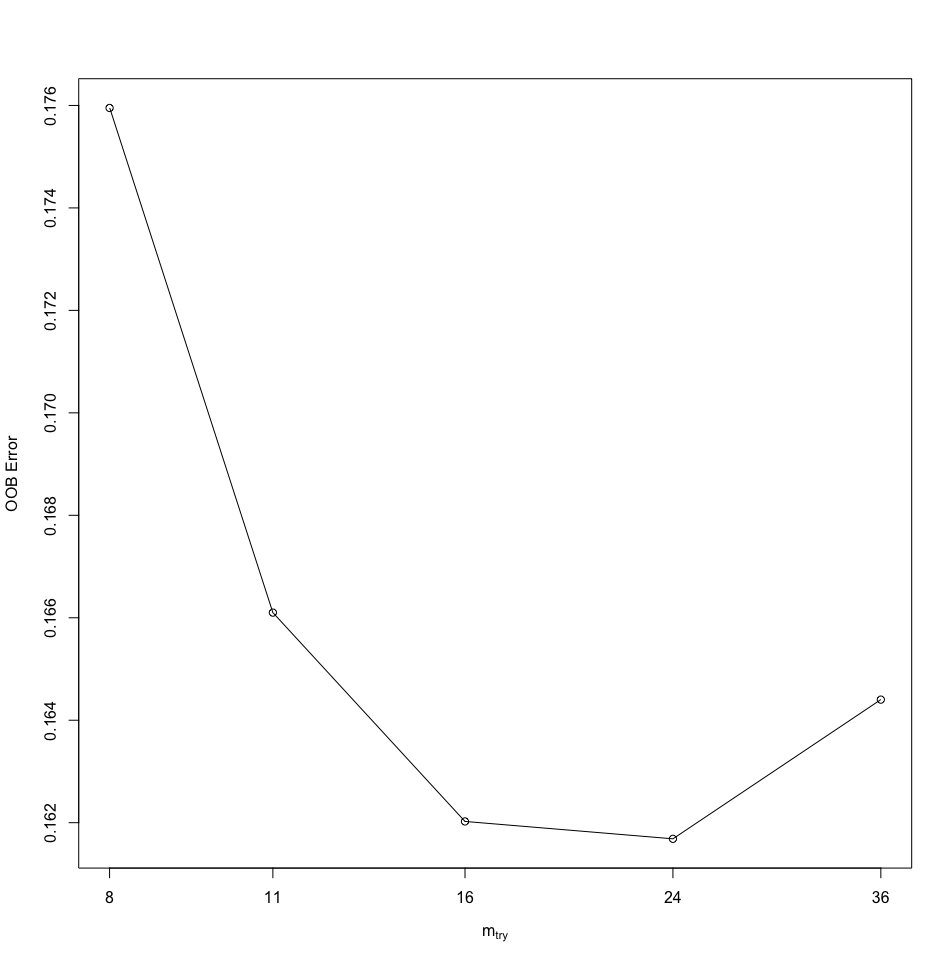
|  | Predicted Positive | Predicted Negative |
| --- | --- | --- |
| True Positive | 1680 | 644 |
| True Negative | 494 | 4051 |

**ROC curve of initial model**



*-Hyperparameter Tuning*

*Purpose of tuning Mtry* M\_try parameter in the R’s randomForest package as a maximum possible number of features that each decision tree can get, which will be used to build an ensemble model. Generally, the advantages of using a larger value for the M\_try value is that there are more chances for each decision tree to have important predictors, which will make the Random Forest model converge more efficiently. However, this comes with the performance cost, as we will have to use a lot more computing power to generate each decision tree. As we want to maximize our performance with the least amount of computing power, we decided to tune for the M\_try. To find the most efficient M\_try variables, we used R’s Caret packages to perform hyperparameter tuning. Typically, to perform hyperparameter tuning with Random Forest model, we have two choices; we can use GridSearchCV packages in Python, and Caret packages in R. While we did attempt to use GridSearchCV packages, with our current level of knowledge, the choice of input we have to tune for the M\_try was limited. In GridSearchCV, this was *max\_features* value and this value can only take squared root, natural log, and total number of parameters. So, in our case, we could only test 2, 11, and 123 for the M\_try if we used Python’s GridSearchCV. However, with R’s Caret package, we could manually try all possible values of M\_try, which will allow us to tune M\_try in a more refined manner.



Above, we plotted the out-of-bag error (OOB) error versus various number of M\_try. Here, the OOB error is the average error for each calculated using predictions from trees that do not contain in their respective bootstrapped sample. The result of the tuning parameter shows that OOB error attains minimum when M\_try value was 24.

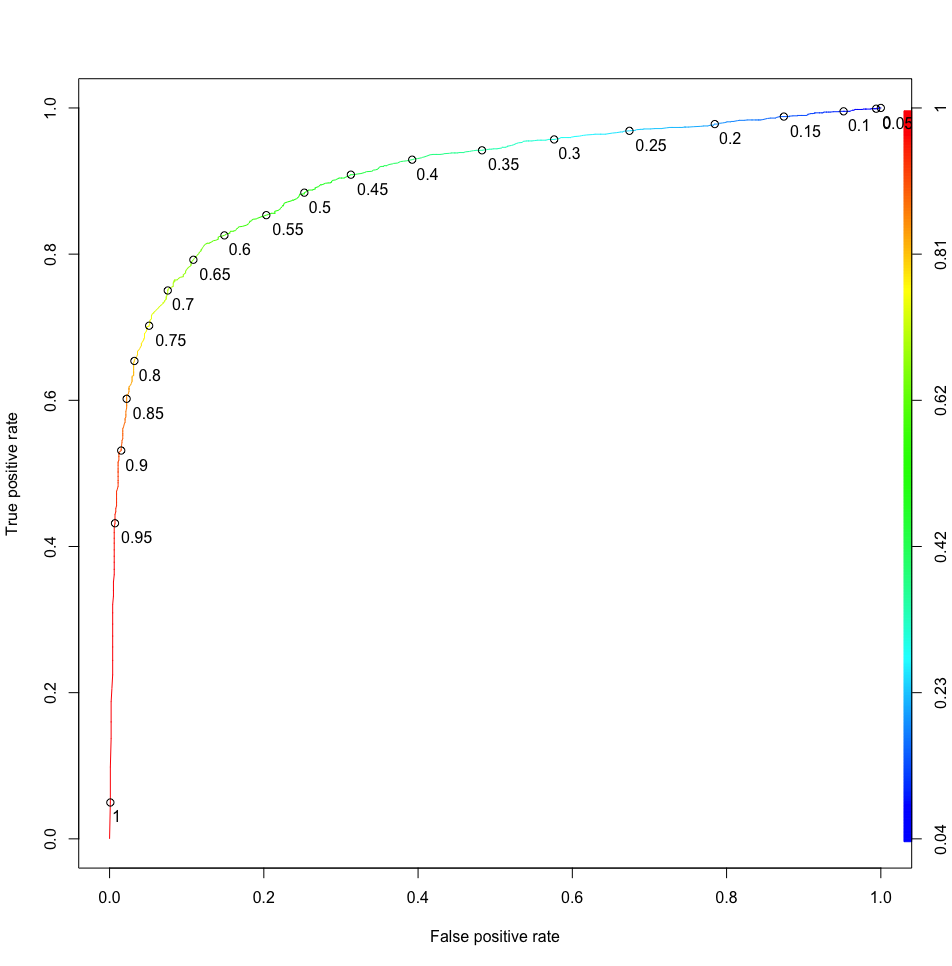
*-Final Model*

We then fit the random forest model again with M\_try of 24, using this as a final tuned model. The output shows that the current model had an AUC score of 0.91 and with the best probability cutoff being 0.51 from the ROC curve, the model's prediction accuracy was 0.84. Confusion matrix below was generated using the best probability cutoff of 0.51. For our final model, the processing time of fitting the random forest model is 16 minutes.

**Confusion matrix of best model**

|  | Predicted Positive | Predicted Negative |
| --- | --- | --- |
| True Positive | 750 | 244 |
| True Negative | 237 | 1713 |

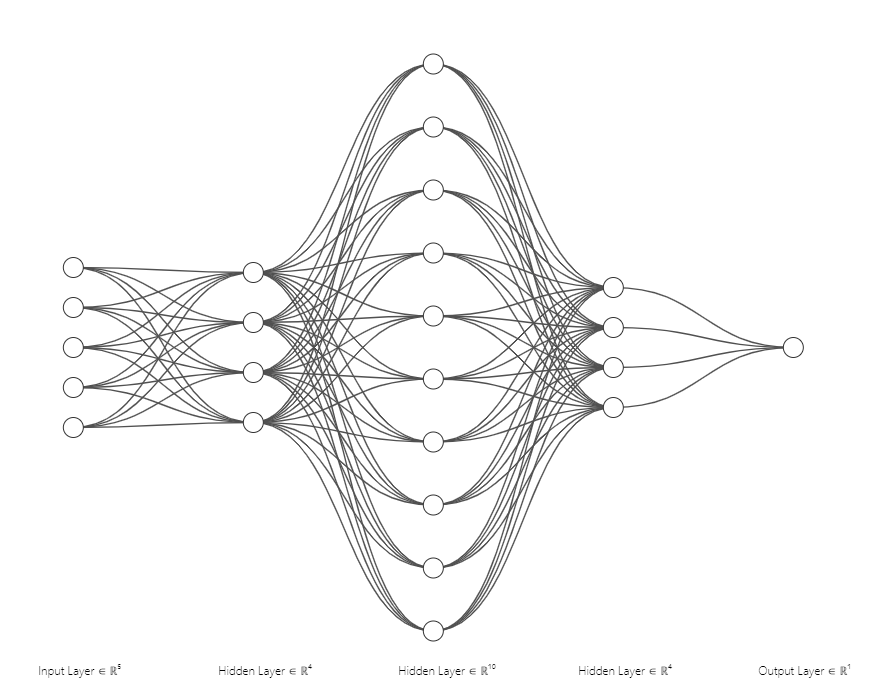
**ROC curve of best model:**



**Artificial Neural Network - Multi Level Perceptron**

Zanella-Calzada et al. (2018) attempted to fit Artificial Neural Network (ANN), specifically, Multi-Level Perceptron (MLP) to create a predictive model for the oral health classification task, using various socioeconomic and nutrition-state associated features.

While the details of why they specifically chose MLP model is unclear, we suspect that their model choice was because the NHANES dataset is with a substantial number of features in the tabular format to do binary classification tasks. Typically, hidden layer designs for the MLP are often incrementally decreasing in the number of nodes (neurons) as they proceed to the deeper layer, but Zanella-Calzada et al (2018) suggested a rather unconventional hidden layer design, similar to the Figure below. As it is difficult to create a visually comprehensible diagram due to the sheer size of layers, we created a 4 layer model that is conceptually identical but with a different number of neurons. First layer should have 123 neurons, second layers should have 100 neurons, third layers should have 1000 neurons, fourth layer should have 100 neurons, while Output layer should still be the same as 1 neuron.

**(Source: <https://alexlenail.me/NN-SVG/index.html>)

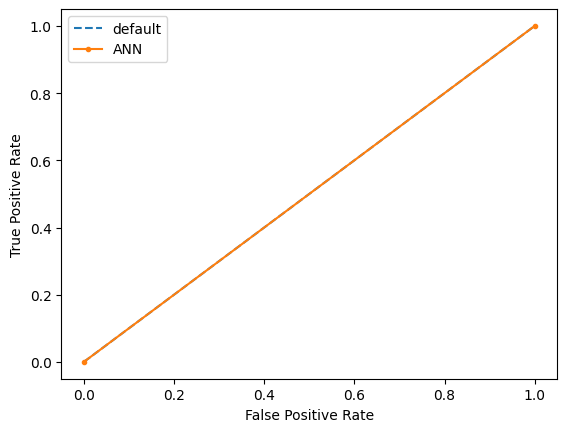
| **Input Layer** | **Dense 1** | **Dense 2** | **Dense 3** | **Dense 4** | **Output Layer** |
| --- | --- | --- | --- | --- | --- |
| 123 features  123 neurons | 123 features  100 neurons | 100 features 1000 neurons | 1000 features  100 neurons | 100 features  100 neurons | 100 features  2 neurons |

Such hidden layer design choices were not explained in the paper beside this being the result of the tuning process. However, such a design seems to have the benefit of doing feature expansion at the first Dense layer to second Dense layer where we fit 100 features into 1000 neurons. Considering that our data have high dimensional nature as the dataset was collected across multiple subjects from various backgrounds (e.g, demographics and diets), it is difficult to assume that each one of these features may come from one harmonious population. Therefore, using designs with feature expansion allows neural networks to investigate each feature in a more in-depth manner, by creating more features (1000 features) from the original features and help us train a better prediction model. However, the feature expansion comes with the risk of overfitting as we are fitting in the same dataset repetitively across the same neurons to be co-adapting together. To prevent this, each of these Dense layers were accompanied by a Dropout layer, which randomly drops a predetermined number of neurons, thus controlling the Learning Rate when going through each Batch of the original data. In the paper, the author suggested the Learning Rate to be 0.5 for arbitrary reasons, but we later tuned the Learning Rate to see which Learning Rate yielded the best result.

*Model Fitting* To fit the MLP model, we used Python 3.10’s Keras package’s Sequential model with Dense and Dropout layer function to create the neural network and optimized the neural network with Adam Optimizer, with Batch size of 64 and Epochs size of 100, while using binary cross entropy as a loss function and rectified linear unit (ReLU) for the activation function for each Dense layer. Finally, for the output layer, we attempted two different methods; once with the *softmax* function as the original author suggested, and *sigmoid* function as we thought it to be more appropriate choices as this is a binary classification task, while mathematically they should function exactly the same as *sigmoid* is just a special case of *softmax*.

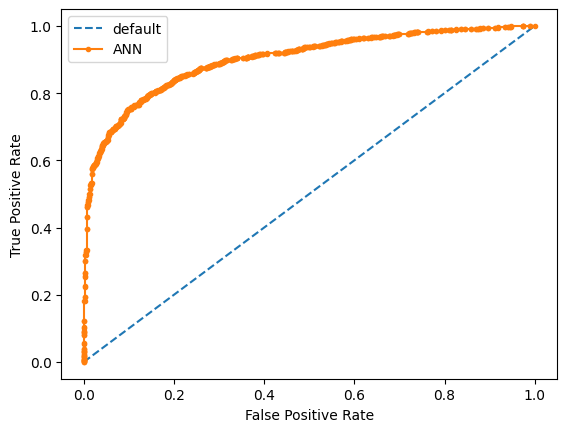
*Initial attempt* With the *softmax* function in the output layer, the models’ predictive (Test) accuracy was 0.66, with the test prediction error (loss) of 0.45. While this result is consistent with the original paper’s report of 0.68 prediction accuracy, both confusion matrix and ROC curve showed that the model is not performing well, as their prediction always resulted in one value exclusively. The confusion matrix below was created with the optimal cutoff points of 0.5. Also, the ROC curve showed that the current model’s prediction performance was exactly identical to the random classifier, indicating that the model cannot predict effectively The AUC score for the current ROC curve was 0.5.

|  | Predicted Positive | Predicted Negative |
| --- | --- | --- |
| True Positive | 0 | 994 |
| True Negative | 0 | 1950 |



*Second attempt* In this attempt, we only replaced the output layer’s activation function from *Softmax* to *Sigmoid* as *Sigmoid* is a popular choice of activation function for the binary classification task. Initially, we were not expecting to observe any differences as both functions should yield the same result theoretically. However, the model’s performance improved and the previous issue with the prediction was resolved. Using the ROC curve, we found the optimal cutoff points that can retain both sensitivity and specificity, which was 0.64 in this case. Using 0.64 as a cutoff point, we got the confusion matrix below, and the best possible accuracy for the current model was 0.81. As we can see in the confusion matrix, the model was successfully able to predict both classes of the dependent variable (1 and 0). The AUC score of the ROC curve of the current model was 0.89.

|  | Predicted Positive | Predicted Negative |
| --- | --- | --- |
| True Positive | 807 | 187 |
| True Negative | 368 | 1582 |

**

*Hyperparameter (Learning Rate) tuning*

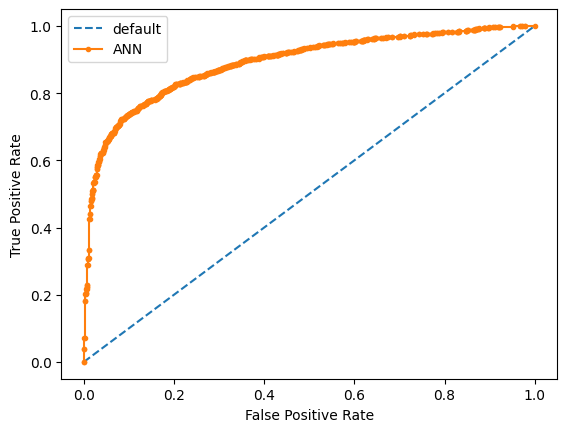
In the original paper, the author performed hyperparameter tuning for the Epoch size and as we already decided to use Batch Size of 64 for the purpose of reducing the computational load, we tuned the Learning Rate to see if this benefits the current MLP’s prediction performance. To test how change in Learning Rate affected the prediction performance of the MLP model, we aimed to find the best Learning Rate value with the lowest Test Loss and highest test accuracy, which is provided by Keras. From our result, we observed the Learning Rate of 0.6 provided the smallest Test Loss with highest test accuracy, so we fitted our final model with Learning Rates of 0.6.

*-Learning Rate tuning result*

| **Learning Rates** | **Test Loss Percentage** | **Test Accuracy** |
| --- | --- | --- |
| 0.1 | 0.79 | 0.78 |
| 0.2 | 0.43 | 0.82 |
| 0.3 | 0.41 | 0.81 |
| 0.4 | 0.40 | 0.82 |
| 0.5 | 0.47 | 0.80 |
| 0.6 | 0.37 | 0.83 |
| 0.7 | 0.38 | 0.83 |
| 0.8 | 0.39 | 0.81 |
| 0.9 | 0.54 | 0.62 |

*Final Model* As we are proceeding with author’s original four-level MLP model design with same Epoch and Batch Size, but with *Sigmoid* activation function at the Output layer and with Learning Rate of 0.6. The AUC score of the ROC curve, confusion matrix, test accuracy, and cutoffs based on ROC curve did not change from the model with Learning Rate of 0.5. The running time for the final tuned MLP model was 1 minute 28 seconds.

|  | Predicted Positive | Predicted Negative |
| --- | --- | --- |
| True Positive | 807 | 187 |
| True Negative | 368 | 1582 |



**Conclusion**

For our current project, we aimed to create a machine learning based model that requires the least amount of data cleaning and requires researchers to manually investigate the characteristics of each variable. As the NHANES dataset is a tabular high dimensional dataset, and since we are attempting to create a predictive model that can predict the binary outcome of oral health, Random Forest model and MLP model were used to create a predictive model for the binary classification task. To compare the performance of our two different machine learning models, we used AUC score from ROC curve, prediction accuracy (test accuracy) with best possible cutoff points using ROC curve, and also looked at the processing time of each model.

We observed that the AUC score of the ROC curve in the Random Forest model was 0.91, while the MLP model was 0.89. Also, using the optimal cutoff points, the best possible accuracy in the Random Forest model was slightly better (0.84) than the MLP model (0.81). In other words, the Random Forest model showed better predictive performance than the MLP model overall. However, the Random Forest model with repeated cross validation took 16 minutes for the model to be converged, trained, and to perform prediction, while MLP only took 1 minute and 28 seconds. This processing speed difference may not be a fair comparison as the Random Forest model was performed in R using a local machine while MLP was performed using Python in a virtual environment with a dedicated GPU (Google Colab). Also note that both machine learning models usd cross-validation like resampling methods (i.e., 10 fold cross validation in Random Forest and Epoch learning in MLP) on the training dataset to create the best prediction model, and this is a computation intensive job, and local machine will perform worse than virtual machine with dedicated GPU, since local machine’s computational resources are used in other background tasks.

However, this amount of large processing time difference may also reference potential differences between two different machine learning models. For instance, the Random Forest model requires establishing hierarchical structures across each variable, and creating multiple decision trees to create an ensemble environment requires a lot of computational resources, especially with 123 features. And the 10 fold cross validation process being carried out on this will require strong computing power. On the contrary, MLP model perceives these features in 1 dimensionally in input layer, and the furthest depth that such consideration for hierarchical structure across features only goes up to the predetermined layer (4 in our case), and achieves Random Forest equivalent accuracy much faster as Dropout layers in each Dense layer helps us to achieve ensemble state faster than Random Forest, with shallower but wider hierarchical structures.

Current paper has methodological limitation as the Zanella-Calzada et al (2018)’s processed dataset does not matches with our processed dataset; although these are minor differences (i.e., 104 features vs 123 features), if there were strong predictor included/excluded in the dataset we used compare to the dataset the original authors used, some of the hyperparameter tuning result (e.g., appropriate Epoch size) they performed in MLP may not be matching, although we implemented their hyperparameter tuning result in our final model.

**Citations**

Zanella-Calzada, L. A., Galván-Tejada, C. E., Chávez-Lamas, N. M., Rivas-Gutierrez, J., Magallanes-Quintanar, R., Celaya-Padilla, J. M., Galván-Tejada, J. I., & Gamboa-Rosales, H. (2018). Deep Artificial Neural Networks for the Diagnostic of Caries Using Socioeconomic and Nutritional Features as Determinants: Data from NHANES 2013⁻2014. Bioengineering (Basel, Switzerland), 5(2), 47. <https://doi.org/10.3390/bioengineering5020047>

**Appendix**

Python Code and R code can be found in the following link:  
[jwlee96/ML\_NHANES13\_14 (github.com)](https://github.com/jwlee96/ML_NHANES13_14)