Predicting Poverty Status Using Medical Condition Data

Nathan Kim(805182816), Kamakshi Sirpal(305864760), and Daniel Wang(105867284)

Abstract—**When asked to consider the major problems facing the country, the affordability of health care tops the American public's list. Last year, the census announced that median household income in 2020 decreased 2.9% between 2019 and 2020, and the official poverty rate increased 1.0 percentage point. How much of this decrease can be explained by the health conditions of the population? This paper focuses on the prediction of poverty status based on the reported diagnosis of various diseases. The authors examine the predictive power of linear regression, logistic regression, support vector machines, neural networks and random forest. The authors find that all models have comparable predictive powers and recommend a logistic regression given its relatively cheap computational cost.**

I. Introduction

The goal of our analysis was to evaluate whether poverty level of the U.S. population can be predicted using health data on disease diagnosis as provided by the International Statistical Classification of Diseases and Related Health Problems (ICD). We were interested in exploring the extent to which prevalence of various diseases in the United States can help predict the poverty level of individuals. This approach has been followed by other researchers in recent times (Kallestal et al, 2019) with the use of data mining techniques in Nicaragua, a developing country. We wanted to test to what extent a similar analysis can be applied to a developed country like the U.S. with a more advanced healthcare system. The general observation is that the causes of poor health globally are rooted in political, social and economic injustice. Poverty is both a cause and consequence of poor health. The prevalence of diseases among the U.S. population would impact the quality of human capital and hence overall output and wages. We decided to explore this pertinent question through the use of basic and complex machine learning models to try to obtain the best possible model for our dataset of roughly 16,000 observations. We used models such as linear regression, logistic regression, support vector machines, neural networks and random forests to perform a thorough analysis of this prediction.

II. Data Summary and Challenges

The data we analyzed was the Medical Expenditure Panel Survey (MEPS), which contain medical conditions of a representative sample of individuals. The data also includes income data at the individual level. The data we analyzed was from the MEPS conducted in 2019. The medical conditions data consists of medical diseases that survey respondents report to have been diagnosed with. Reported conditions were recorded as text and standardized by medical professionals into disease categories in the International Statistical Classification of Diseases and Related Health Problems (ICD). The ICD is a handbook of disease classifications created by the World Health Organization.

In the 2019 dataset, we find that there are 394 unique ICD codes reported, an incredibly large number of features to consider in our algorithms. To minimize the number of features, we aggregated the ICD codes into 20 broader disease categories. These categories are defined in the Appendix. The diagnoses data were then converted into dummy variables as respondents reported as being diagnosed or not.

III. Linear Regression

Upon preprocessing our data, we decided to employ our first model. We started our analysis by using a linear regression model. This is a widely used model when it comes to healthcare data since it enables the identification and characterization of relationships among multiple factors. It is an attractive model because the representation is so simple. It also made sense to use this as our first model and check how it performs with respect to predicting the continuous variable poverty level "POVLEV19".

We used the sklearn linear regression model for our purpose. Upon loading the dataset in Python, there were some unnecessary columns in the beginning which I eliminated to be used for our analysis. I then subset the data to only the categories of illnesses (A, B, ... T, Z) which each represent the collapsed ICD codes like infectious diseases, blood related diseases and non-illness factors like contact with health services. Our final array of input variables includes all categorical variables while our prediction label is continuous. We then split the data into training and testing sets, with our test size being set as 30% of the total dataset.

I used the .fit() function on the training data to obtain the values of the coefficients. It was interesting to note that almost half of the coefficients are positive while the remaining half are negative. This indicates that on the most preliminary level we see different health indicators impacting poverty level in different ways and there is no universal relationship. I obtained the training and testing set scores for the regression model, this gives us the coefficient of determination or R squared of the regressors. I found these values to be significantly low as below:

Training and testing set scores (R squared)

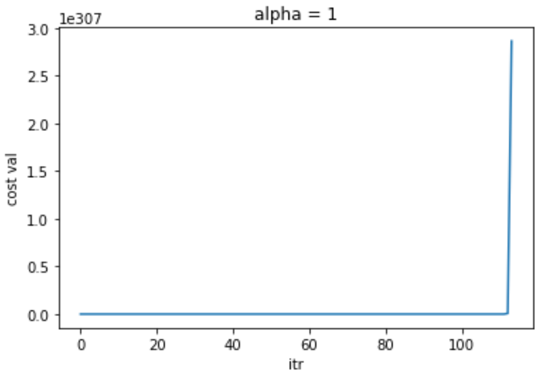
Training set score	0.02
Test set score	0.01

These results indicate that the model has a problem of underfitting since not only the test set score but the training set score is also very low. I investigated this further and evaluated the model on several metrics such as mean squared error, mean absolute error and root mean squared error. I found the below results:

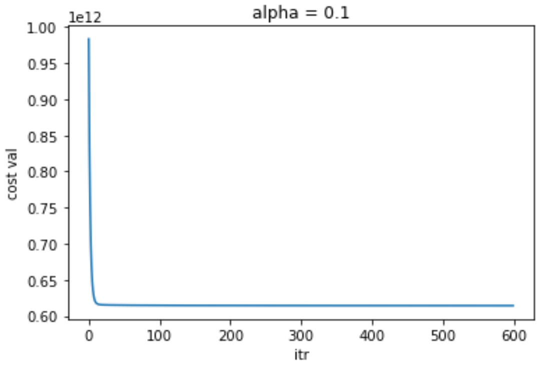
Metric evaluation

Mean squared error $\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$	134698.24011
Mean absolute error $\frac{1}{N}\sum_{i=1}^{N}\lvert y_i - \hat{y}\rvert$	265.06062
Root mean squared error $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2}$	367.01258

The model did not perform well on these metrics either and we observed very high error values. I then ran the gradient descent algorithm for optimization using the given coefficient values to check if it converges. I found that at an alpha value of 1 and maximum iterations of 500, the gradient descent function was exploding as shown below:



I tried a few more combinations of alpha and maximum iterations and found that there was convergence in the function when I used an alpha of 0.1 and maximum iterations of 600. This was the least computationally expensive set of the two hyperparameters to achieve convergence as shown below:

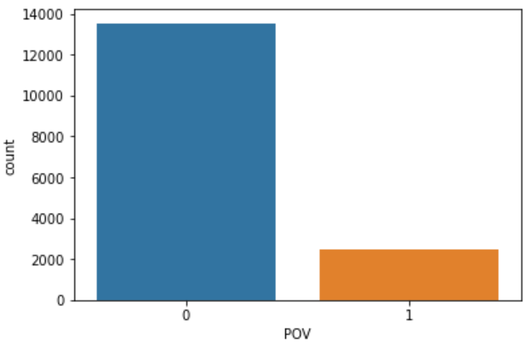


However, since the performance of the model was not very good we decided to move on to other models rather than trying to address the issue of underfitting in the linear regression model. Most types of model diagnoses for multiple linear regression models focus on resolving the issue of overfitting by introducing regularization terms but in this case it would not have been helpful. Scope for further improvement when trying to predict a continuous variable using categorical inputs is the use of multi-way ANOVA models.

IV. Logistic Regression

The second model I implemented was the logistic regression model. We used the binary classifier for poverty level as the outcome variable rather than the continuous variable for poverty level used earlier. I found that the majority of the poverty level data was classified as "0" which indicates above poverty level and "1" which indicates below poverty level. This is because we grouped multiple categories into the "0" category to obtain a binary classifier for our model. This data aligns well with the 13.4% poverty rate in America. Essentially we are trying to gauge how the health inputs can classify an individual as "poor" or "not poor" and not exactly predict their poverty level.

I visualized the imbalanced data as well as the breakdown by categories A and B which indicated that the individuals were not reported to be diagnosed with infectious and blood related diseases were most often the ones who were categorized as "0" or above poverty level. After the visualization, I split the data into training and testing sets once again selecting the test size as 30% of the data. I ran the logistic regression model and obtained the accuracy score. The test set score was 0.85 which was a marked improvement from the linear regression model. I also checked for cross validation scores with 10 folds which were found to be close to 0.843.



I then decided to fine tune the model hyperparameters to see if we can improve upon the achieved accuracy. When I tried different solvers, I achieved the same accuracy each time (0.85). I tried the solvers ‘liblinear’, ‘newton-cg’, ‘lbfgs’, ‘sag’ and ‘saga’. I was also concerned about the overfitting problem in my model since the test set score was higher than the average expected in logistic regression models (0.7). Hence I applied different regularization techniques to see how it would impact my model’s performance. The results are as below:

Penalty	*Accuracy Score*
l2	0.85072
None	0.85030
l1	0.85051

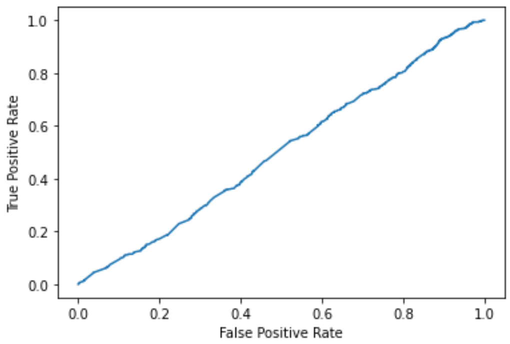
I saw some minute changes in accuracy score for different types of regularization with l2 regularization giving the best results. From a practical standpoint, L1 tends to shrink coefficients to zero whereas L2 tends to shrink coefficients evenly. L1 is therefore useful for feature selection, as we can drop any variables associated with coefficients that go to zero. L2, on the other hand, is useful when you have collinear/codependent features. In this case it is possible that the performance of L2 regularization is better for this reason.

The third hyperparameter I used for fine tuning was the value of C. C controls the penalty strength and it can be effective to adjust it and check the results. For the L2 regularization I adjusted the values of C as 1, 100, 10, 0.1 and 0.01. I observed the below results:

Penalty Strength (C)	*Accuracy Score*
1	0.85072
100	0.85030
10	0.85051
0.1	0.85072
0.01	0.85072

I observed the highest accuracy for lower levels of penalty strength (1, 0.1, 0.01) but reducing the value of C did not yield better results. Higher penalty strength of 100 and 10 did not provide as high accuracy scores. Thus I concluded that the best model for logistic regression was the one with regularisation of L2 and penalty strength of 1. The accuracy could further be improved by performing a grid search to tune the hyperparameters. However I did not see much scope for improvement in the logistic regression model.

I also wanted to plot the ROC curve to show the performance of the model at all thresholds. I used the sklearn decision function to obtain the y scores. I then computed the ROC values and plotted the true and false positive rates on the graph as below:



Ideally, the curve should be more L shaped with its edge at the top left for us to conclude that the model is a good classifier. In this case looking at solely the ROC curve as a metric the performance is not very good. The third model I explored for this project was Support Vector Machines (SVM). SVM tries to finds the “best” margin (distance between the line and the support vectors) that separates the classes and this reduces the risk of error on the data, while logistic regression does not, instead it can have different decision boundaries with different weights that are near the optimal point.

V. SVM

This is a supervised learning model with associated learning algorithms that analyse data for classification analysis. I used the svm function on sklearn and created a classifier using a linear kernel to begin with. I then trained the model using training sets of the X and Y data. Upon predicting the response for the test dataset, I calculated the accuracy. I found the accuracy to be 0.8503 which was lower than the best accuracy provided by our logistic regression model. However the precision and recall rates while low were still better than the logistic regression model. The precision score was 0.3333 while the recall rate was 0.0027.

The low recall rate means that our classifier has a high number of false negatives which can be an outcome of imbalanced class or untuned model hyperparameters. The data is definitely imbalanced which could be the explanation for this. However, to account for the second reason I tuned the hyperparameters of the model to try to obtain higher accuracy. I changed the kernel a few times to try ‘rbf’, ‘poly’, ‘sigmoid’ and ‘precomputed’. The accuracy remained exactly the same except for when I used rbf when it reduced slightly.

I also changed the regularisation parameter C for which I tried the values 0.1, 10, 0.01 and 100. I obtained the same accuracy score for all. I further tried to adjust the gamma (kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’) values. Once again the accuracy did not change for my model and I concluded that the best model had an accuracy of 0.8507 for the linear kernel. To evaluate whether our model could perform better we went on to try neural networks and random forests for classification.

VI. Neural Network analysis

To start, we will first discuss the machine learning system design used to build, train, and test the proposed neural network models. First, the data was split into training and testing sets. We used 70% of the data for training and hyperparameter tuning/grid search and 30% of the data as the final test set. Next, we built four different neural network models with 1, 2, 3, and 4 hidden layers, respectively. Using the training set, we performed cross validation/grid search to tune the hyperparameters of the models. We first performed a grid search by only tuning the learning rate. Then, we performed another grid search where we tuned both the learning rate and the L1 regularization coefficient simultaneously. So, technically, 8 different models were considered. We then trained each model using their respective optimal hyperparameter(s) on the entire training set. Finally, we evaluated the tuned and trained models on the test set and used the accuracy rate to determine the best performing model. We then provided more evaluation metrics on the optimal neural network model which we could compare to the other machine learning models.

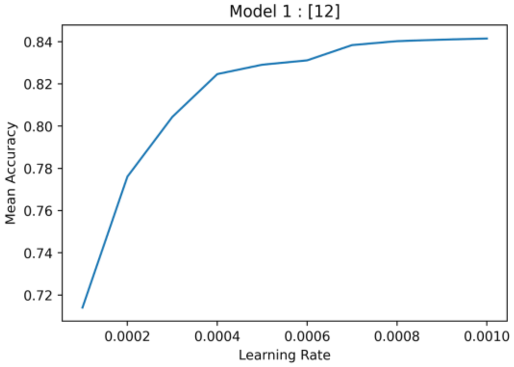
To solve our classification problem, four main neural network models were considered. The models were built using the Python module *TensorFlow*. All of the candidate neural networks were feed forward (sequential) models. Due to their popularity and success, we decided to use the rectified linear unit (ReLU) activation function in the hidden layers for all of the models. The final output layer uses a sigmoid activation function since the target variable is a binary variable. Moreover, since all of our features are also binary variables, we did not specify any preprocessing methods in the neural networks. All of the models were then trained/fit using the Adam optimization method and the binary cross entropy loss function. Below is a table that specifies the four models we considered.

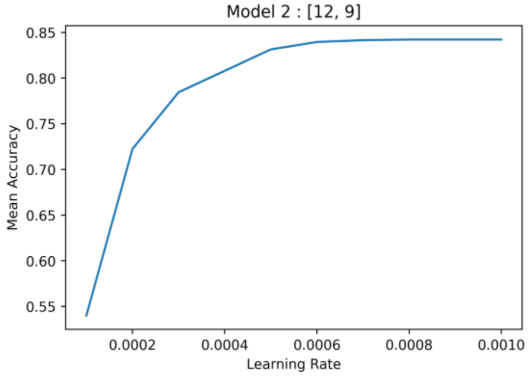
	Number of Hidden Layers	**Neurons in Each Hidden Layer**
Model 1	1	[12]
Model 2	2	[12, 9]
Model 3	3	[12, 9, 6]
Model 4	4	[12, 9, 6, 3]

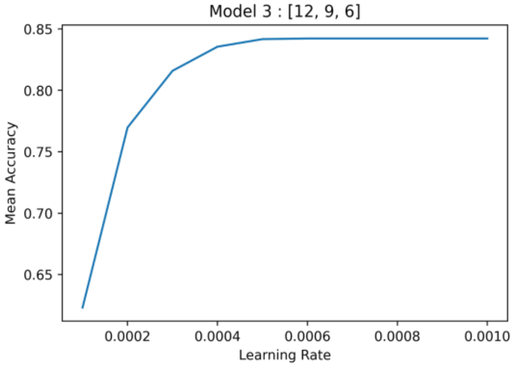
As you can see above, Model 1 had only one hidden layer with 12 neurons. Model 3 had 3 hidden layers with 12, 9, and 6 neurons, respectively. Again, each model had a final output layer with a sigmoid activation function.

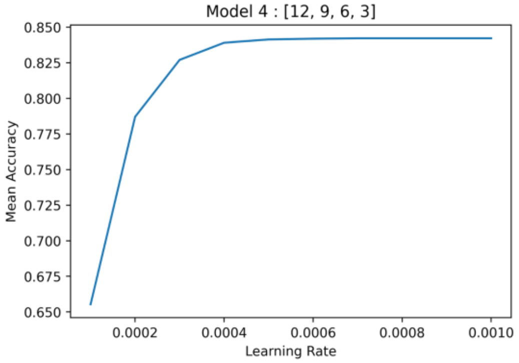
For each model, we performed a grid search using cross validation to tune the hyperparameters which included the learning rate and the L1 regularization coefficient. For the grid search process, we used the Adam optimizer, a batch size of 512, and 5 epochs to speed up the runtime of the code.

First, we only tested 10 different learning rates which included [0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008, 0.0009, 0.001]. We chose these learning rates because there was not much variation in the performance of the models beyond a learning rate of 0.001. For each model, we performed 5-fold cross validation using each learning rate. At each learning rate for each model, we calculated the mean accuracy of the model's predictions over the 5 folds. Below, we plot how the mean accuracy changes as the learning rate increases.



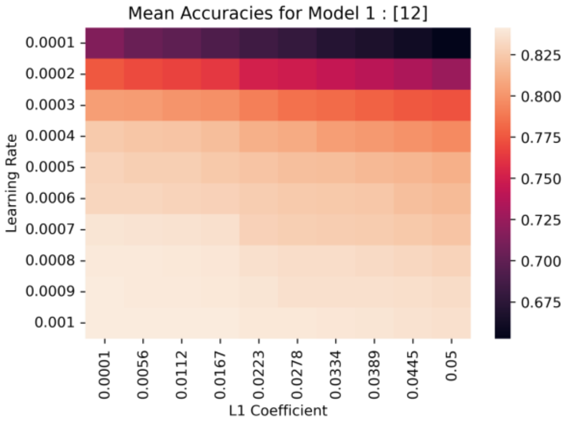




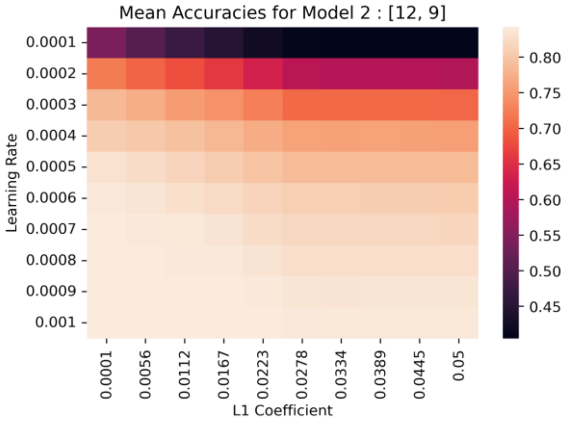


The above figures depict the grid search for the learning rate for each model. We can see that as the learning rate increases, the mean accuracy starts to flatten out. In addition, as the neural network becomes more complex (as the amount of hidden layers and neurons increase), the mean accuracy flattens out more quickly.

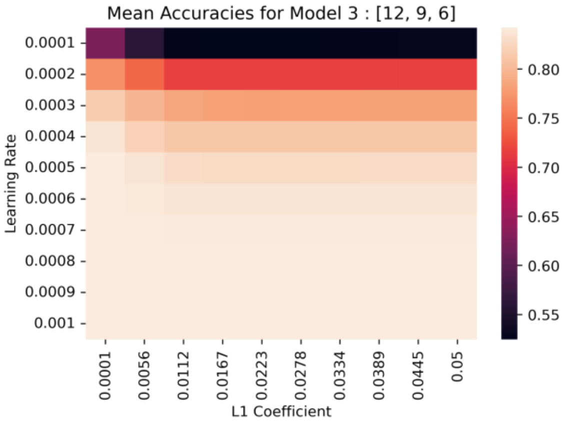
Next, to determine if our model was potentially overfitting, we decided to add an L1 regularization coefficient to each hidden layer. For each model, we performed 5-fold cross validation for all combinations of the learning rates previously specified [0.0001, 0.0002, … , 0.001] and the L1 regularization coefficients which included [0.0001, 0.0056, 0.0112, 0.0167, 0.0223, 0.0278, 0.0334, 0.0389, 0.0445, 0.05]. Therefore, in this case, we had a total of 100 pairs of hyperparameters to consider. For each pair and for each model, we calculated the mean accuracy of the model's predictions over the 5 folds. Below we have heat maps that show how the mean accuracy changes as the hyperparameters change.



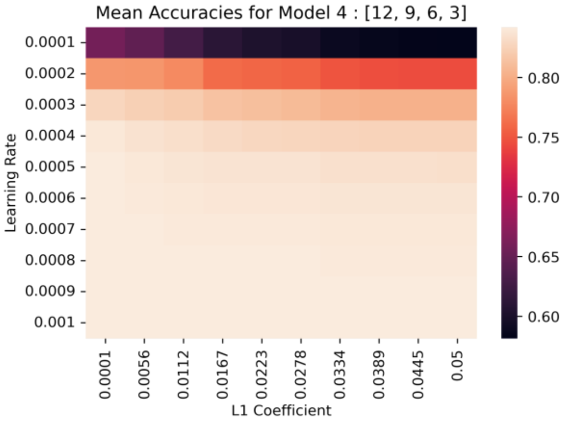
Mean Accuracies for Model 1 : [12]



Mean Accuracies for Model 2 : [12, 9]



Mean Accuracies for Model 3 : [12, 9, 6]



Mean Accuracies for Model 4 : [12, 9, 6, 3]

The above figures depict the grid search for both the optimal learning rate and the L1 regularization coefficient for each model. The hue of the heat maps represent the mean accuracy for each combination of hyperparameters. For all models, we can observe that as the learning rate increases and the L1 coefficient decreases, the mean accuracy increases. As our neural network models become more complex, the mean accuracies for each combination of hyperparameters vary less. We can see this because the color scale in the heat maps for models 3 and 4 vary less than for models 1 and 2.

After performing both types of grid searches, we used the optimal hyperparameters(s) for each model to train the models on the entire training set, and then evaluated their accuracies over the test set. When fitting the models with the optimal hyperparameters(s), we used a batch size of 512 and 30 epochs.

Below are the results of this process. The tables below show the optimal hyperparameter(s) and the test accuracies for each model.

Model	**Optimal Learning Rate**	**Test Accuracy w/o L1 Regularization**
1	0.001	0.854466
2	0.0008	0.854258
3	0.0006	0.854466

4	0.0007	0.854466

We can see that when using the optimal learning rate for each model, the accuracies are the same with the exception of Model 2 which performs slightly worse than the rest.

Model	Optimal Learning Rate	Optimal L1 Coefficient	Test Accuracy w/ L1 Regularization
1	0.001	0.0001	0.854466
2	0.0008	0.0001	0.854466
3	0.0006	0.0001	0.854466
4	0.0007	0.0001	0.854466

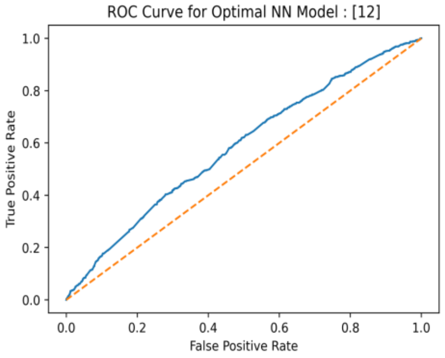
Here, we can see that when using both the optimal learning rate and regularization coefficient, we get the same accuracy rates as the models without regularization. It appears that adding an L1 regularization coefficient to each hidden layer did not improve the models.

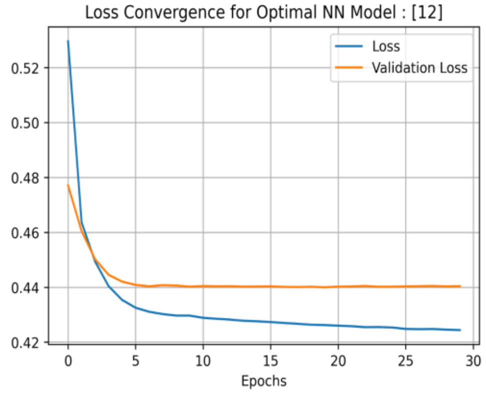
After reviewing the results of the training and testing process above, we decided to use Model 1 with no regularization term as our ‘optimal’ model. Model 1 was the simplest model which may generalize better when dealing with new data from the population. Furthermore, since adding a regularization term to the hidden layers did not make a difference, we decided that Model 1 with no regularization was the best model.

Below are more results for the optimal neural network model which include the confusion matrix, ROC curve, and AUC value.

	Predicted= 0	Predicted= 1
Actual = 0	4104	0
Actual = 1	699	0

From the confusion matrix above, we can see that out of the entire predictions on the test set, 4104 predictions were true negatives, 0 were false positives, 699 were false negatives, and 0 were true positives. These results make sense because our data was highly imbalanced. For this reason, the model has trouble with detecting positive samples. In fact, the model could not correctly identify any positive samples at all in the test set. This problem is concerning as our model may not be able to predict positive labels when provided with new data. Therefore, this issue is a major limitation of this neural network model. From the confusion matrix, we can compute the precision and recall rates for each class. For the negative class (not in poverty), the precision rate is 0.8545 and the recall rate is 1.0. For the positive class (in poverty), the precision rate is undefined and the recall rate is 1.0.





Above is the ROC curve for the optimal neural network model. We can see that the optimal neural network model does better than just randomly guessing the labels. The AUC from the ROC curve above is 0.5846. Underneath is the loss convergence plot that we obtained from fitting the optimal model to the entire training set. We can observe how the training loss converges around the 20th epoch and the validation loss converges around the 5th epoch. Both the training and validation losses are decreasing which indicate that we most likely do not have an overfitting problem which may also explain why the L1 regularization coefficient did not improve the models. We may use the accuracy, ROC curve, AUC, and the other confusion matrix metrics to compare this optimal neural network model to the other machine learning models. However, since our data is highly imbalanced, the ROC curve or AUC may not be the best way to compare models. Therefore, we should rely on other measures such as the accuracy, precision, and recall rates.

VII. Random Forest Analysis

We believed that a random forest algorithm would be a useful model to explore given the high dimensionality and unbalanced nature of the data. Furthermore, the random forest is able to calculate feature importance which may allow us to subset the data and rule out unnecessary features to improve predictability. In testing the random forest, we use sklearn’s RandomForestClassifier. We will tune the model using six hyperparameters. These are discussed below.

Number of trees in a random forest(1); this determines the number of decision trees to use in the algorithm. *The function to measure the quality of a split(2)*; this determines how decision trees in the forest optimize sample splits at each node. Gini and entropy are the two quality split functions we consider. *Number of features to consider at each split(3)*. At each node split, a decision tree will decide which feature(s) to consider in splitting the sample data. In deciding which features to select, each tree in the forest will select from a subset of the total feature list. This is done to reduce correlation between the trees. We will also set the *minimum number of samples required to split each node(4)* and the *minimum number of samples required to split each node leaf(5)*. Here node leaf refers to terminal nodes in the decision tree. Finally, we will also the *maximum number of levels in a tree(6)* which determines the max depth of each decision tree.

First we use sklearn’s RandomizedSearchCV to perform a randomized grid search to narrow the range of possibilities for each of the hyperparameters. The RandomizedSearchCV function also allows us to perform cross-validation on each set of hyperparameters tested. Table 1 displays the hyperparameter grid, or the universe of hyperparameters that the randomized grid search is sampling from.

Table 1

Hyperparameter	**Possible Values**
# of trees	[200, 400, 600, 800, 1000]
Split quality function	[Gini, Entropy]
# of features considered at each split	[$\sqrt{20}$, $\log_2(20)$] (20 is the total number of features)
Minimum samples required at each split	[2, 5, 10]
Minimum samples required at each leaf node	[1, 2, 4]
Maximum levels in a tree	[10, 32, 55, 77, 100]

Fifty sets of hyperparameters randomly sampled from the grid were modeled by the grid search function. These results were cross-validated across five folds. The hyperparameters of the model with the best mean accuracy are outlined in Table 2.

Table 2

Hyperparameter	**Possible Values**
# of trees	200
Split quality function	Gini
# of features considered at each split	$\log_2(20)$
Minimum samples required at each split	2
Minimum samples required at each leaf node	10
Maximum levels in a tree	10

This initial randomized search grid gives us an idea of what combination of hyperparameter values will be best. To further narrow the range of hyperparameter values, we perform another randomized search grid. This time the grid of hyperparameter values are more narrow and surround the values output in the first randomized search grid. This grid of narrower values is detailed in Table 3.

Table 3

Hyperparameter	**Possible Values**
# of trees	[10, 132, 255, 377, 500]
Split quality function	[Gini]
# of features considered at each split	[$\log_2(20)$] (20 is the total number of features)
Minimum samples required at each split	[5, 10]
Minimum samples required at each leaf node	[1, 2, 3]
Maximum levels in a tree	[1, 3, 4, 7, 10]

The hyperparameters with the best mean accuracy of this second randomized search grid are identified in Table 4.

Table 4

Hyperparameter	**Possible Values**
# of trees	377
Split quality function	Gini
# of features considered at each split	$\log_2(20)$
Minimum samples required at each split	5
Minimum samples required at each leaf node	2
Maximum levels in a tree	5

Now the range of hyperparameter values has been dramatically narrowed, we are able to run a straight grid search without expending excessive computational resources. Similarly, we perform a grid search using a narrower grid of values around those in Table 4. Grid search values are outlined in Table 5.

Table 5

Hyperparameter	**Possible Values**
# of trees	[300, 400]
Split quality function	Gini
# of features considered at each split	[4, 5]
Minimum samples required at each split	[4, 5, 6]
Minimum samples required at each leaf node	[2, 3]
Maximum levels in a tree	[3, 5, 7]

The optimal hyperparameters set by the final grid search are:

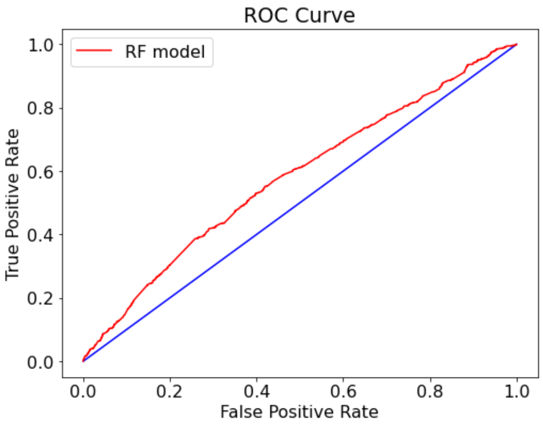
Table 6

Hyperparameter	Possible Values
# of trees	300
Split quality function	Gini
# of features considered at each split	4
Minimum samples required at each split	4
Minimum samples required at each leaf node	2
Maximum levels in a tree	3

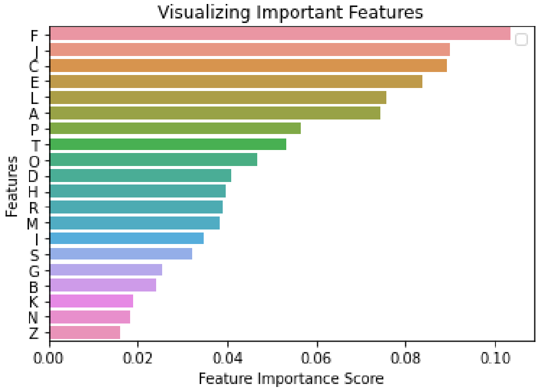
In evaluating our random forest algorithm, we find similar results to those of the neural network. The accuracy score of the model is a relatively high 0.84. But a closer look reveals the limitations of the model. Like the neural network, looking at the confusion matrix, we see that the random forest also fails to identify any of the true positive samples.

	Predicted= 0	**Predicted= 1**
Actual = 0	4020	0
Actual = 1	783	0

We can see that the model is decent at predicting whether a person is *not* in poverty, with a precision rate of 0.84 and recall rate of 1. On the other hand, the precision rate and recall rate of a person in poverty are 0 and undefined, respectively. We see this reflected in the ROC curve as well.



Random forests also allow for feature selection, where narrowing the number of features considered may improve model performance. Given more time, further analysis could be done by subsetting our features to those above a certain threshold and rerunning all algorithms again to see if there is a performance improvement.



VIII. Conclusion

To summarize, we initially approached our machine learning problem through two different learning frameworks: supervised continuous and supervised discrete. As mentioned previously, the linear regression model to predict a continuous poverty variable is unsuitable for our dataset because all of our features are binary variables. Therefore, we focused on taking a supervised discrete approach by predicting a binary poverty variable. We considered four main models which include logistic regression, SVM, random forest, and feed forward neural network. After training/cross-validating and testing each model, we find that all three models result in similar accuracy rates on the test set. When comparing other metrics such as AUC and precision/recall rates, we find low to zero scores for predicting poverty status. Since our data was highly imbalanced, our neural network models and random forest models were unable to correctly predict the true positive labels.

Using the above evaluation metrics did not provide much distinction between the models. Therefore, we believe logistic regression is the best candidate due to its simplicity relative to the other models. Moreover, the optimization process for logistic regression requires less computational resources. Since utilizing more complex models did not seem to improve performance on the test set, the logistic regression should suffice for this classification problem.

For future work, we would like to consider different classification algorithms such as XGBoost and other ensemble machine learning models. Ensemble models have proven to be very successful and perform exceptionally well in Kaggle competitions (2). Furthermore, utilizing random forest and neural networks with regularization allowed us to get some sense of feature importance in our dataset. However, we may want to consider implementing feature selection before modeling. We could use tools such as multiple correspondence analysis (MCA) to reduce the dimensionality of our data. Doing so could potentially make our models more robust and generalize better to the population. In addition, having fewer features/parameters could decrease the runtime for model training, especially the random forest and neural network model.

Appendix

ICD Category Code	Possible Values
A	Certain infectious diseases
B	Certain parasitic diseases
C	Neoplasms
D	Diseases of the blood and blood-forming organs
E	Endocrine, nutritional and metabolic diseases
F	Mental, Behavioral and Neurodevelopmental disorders
G	Diseases of the nervous system
H	Diseases of the eye, adnexa, ear, and mastoid process
I	Diseases of the circulatory system
J	Diseases of the respiratory system
K	Diseases of the digestive system
L	Diseases of the skin and subcutaneous tissue
M	Diseases of the musculoskeletal system and connective tissue
N	Diseases of the genitourinary system
O	Pregnancy, childbirth, and the puerperium
P	Certain conditions originating in the perinatal period
R	Symptoms, signs and abnormal clinical lab findings, not elsewhere classified
S	Injury to a region of the body and certain other consequences of external causes
T	Burns and corrosions to a region of the body and certain other consequences of external causes
Z	Factors influencing health status and contact with health services

References

[1] Kallestal et al, 2019, https://pubmed.ncbi.nlm.nih.gov/31665013/

[2] https://www.toptal.com/machine-learning/ensemble-methods-kaggle-machine-learn#:~:text=Ensemble%20methods%20are%20some%20of,the%20potency%20of%20aggregated%20intelligence.

[3] https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

[4] https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf

[5] https://towardsdatascience.com/improving-random-forest-in-python-part-1-893916666cd

[6] https://www.icd10data.com/ICD10CM/Codes