Individual Report from Aaron Rock

Group 2

Stroke Prediction

The goal of our project was to implement and build a well-balanced model that performs well to predict stroke in patients around the world. For my portion of the work I spent most of my time working with the neural network models Keras and MLP Classifier from sklearn. Prior to these models I worked on performing exploratory data analysis on a third of the variables. Our data set looked as such when read in as CSV to data frame:

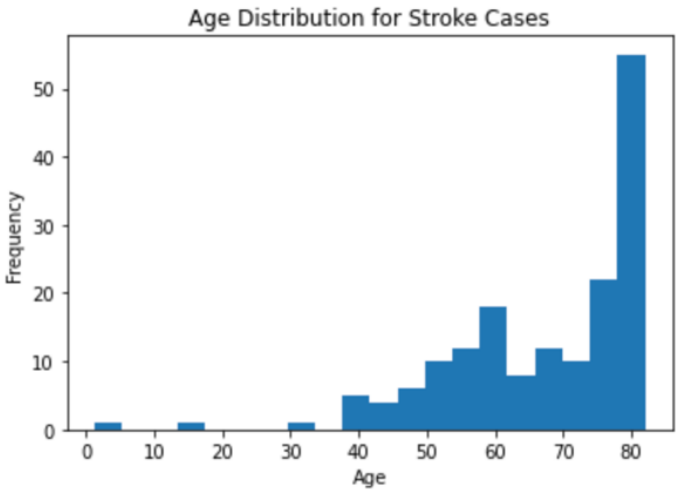
- 5,110 observations (rows) with 12 attributes (columns).
- 1. ID: Patient identifier
- 2. Gender: Patient's gender (Male, Female, or Other)
- 3. Age: Patient's age (quantitative variable)
- 4. Hypertension: Presence of hypertension (0 = no, 1 = yes; categorical variable)
- 5. Heart Disease: Presence of heart disease (0 = no, 1 = yes; categorical variable)
- 6. Ever Married: Marital status (categorical variable)
- 7. Work Type: Patient's occupation (children, government job, never worked, private, self-employed; categorical variable)
- 8. Residence Type: Patient's residence (urban, rural; categorical variable)
- 9. Avg. Glucose Level: Average glucose level in the blood (quantitative variable)

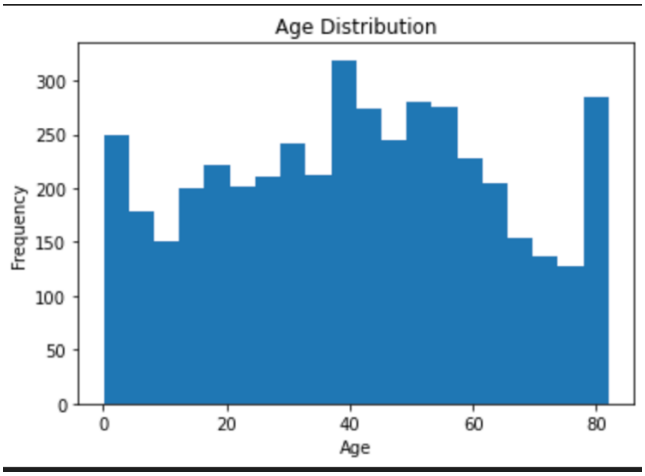
- 10. BMI: Body Mass Index (quantitative variable)

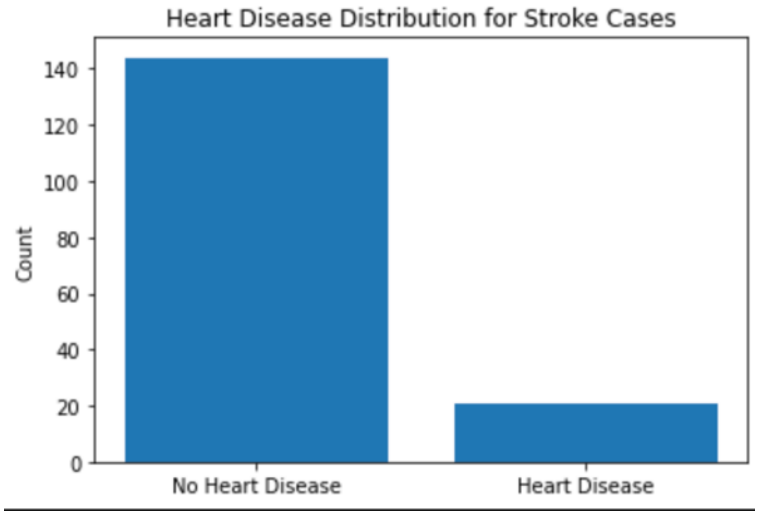
- 11. Smoking Status: Smoking habits (formerly smoked, never smoked, smokes; categorical variable)

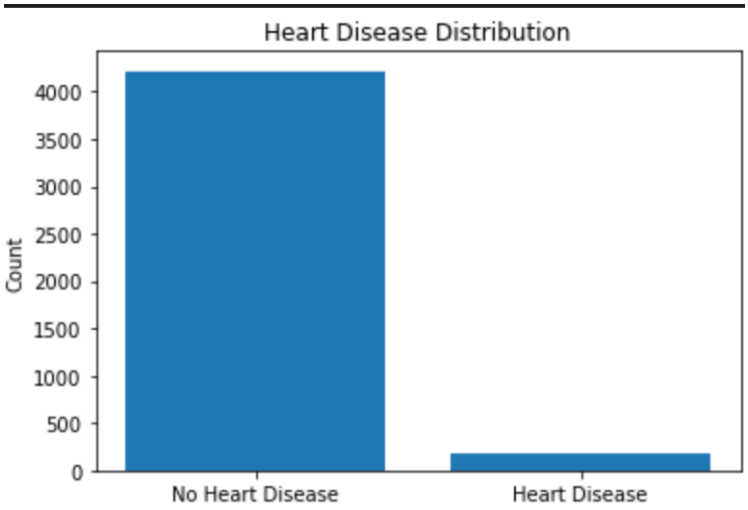
- 12. Stroke: Stroke history (0 = no stroke risk, 1 = stroke risk; target variable)

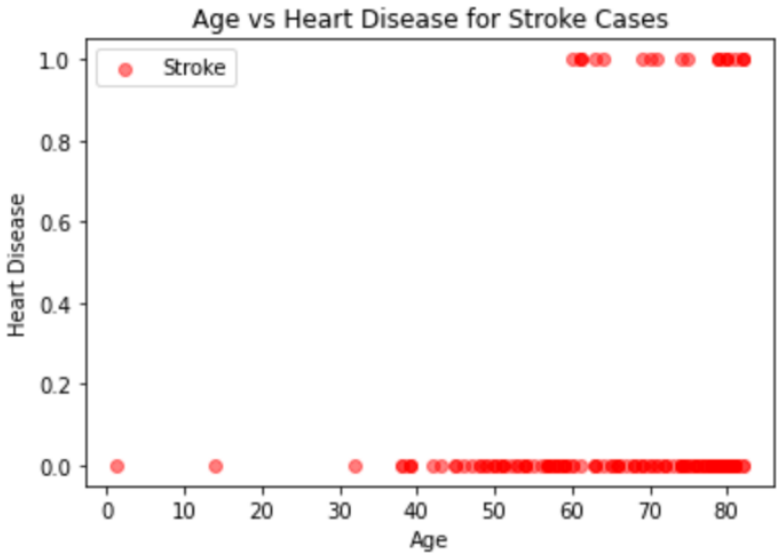
The variables I were assigned were age, heart disease, and ever married. I plotted the following plots from these variables:

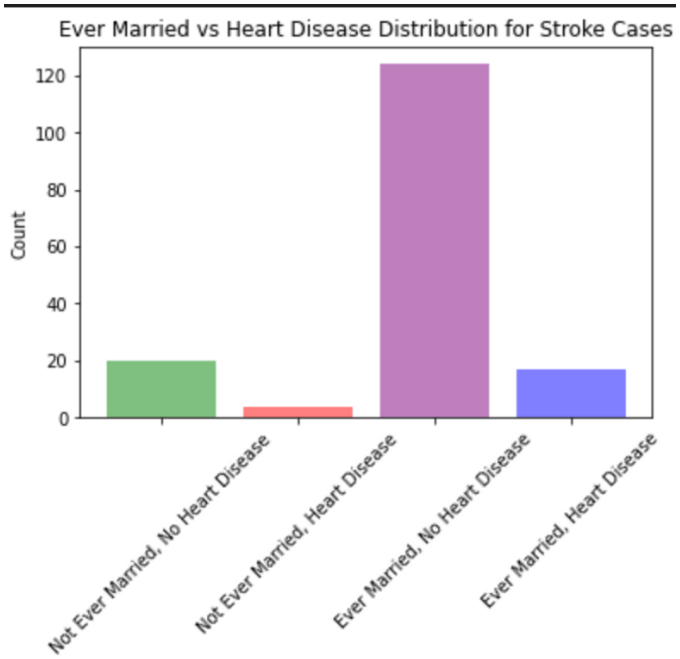





Heart Disease Distribution for Stroke Cases
Count
140
120
100
80
60
40
20
0
No Heart Disease
Heart Disease


Heart Disease Distribution
Count
4000
3500
3000
2500
2000
1500
1000
500
0
No Heart Disease
Heart Disease





Our group had a mistake with the implementation of SMOTE, so I also took time and attempted multiple under and over sampling techniques to help with model metrics. The techniques are as such as Random Under Sampling, Near miss version 1 and 2, and SMOTE. Also, tried SMOTETomek to try a hybrid of over and under sampling.

The Multi-Layer Perceptron Classifier from the sklearn is where most of my time was spent. After, correcting the use of SMOTE on the training data to help balance the training data I built 3 different MLP Classifier using a few different techniques including GridSearchCV, PCA, and RFE for my own individual model. The first MLP Classifier is a simple implementation that is fed a parameter space uses GridSearchCV to find the optimal parameters and then uses those params to predict stroke. For GridSearchCV I am setting the parameter scoring to evaluate based off of the recall score due to the imbalance issues and the fact we want to have better metric for over predicting with false positives rather than missing true negative. The parameter space looks like such:

```
parameter_space = {
    'hidden_layer_sizes': [(20,5),(5,),(10,),(50,5)],
    'activation': ['logistic', 'tanh', 'relu'],
    'solver': ['sgd', 'adam', 'lbfgs'],
    'alpha': [0.9, 0.99, 0.1, 0.0001, 0.05, 0.5],
    'learning_rate': ['constant','adaptive', 'learning'],
}
```

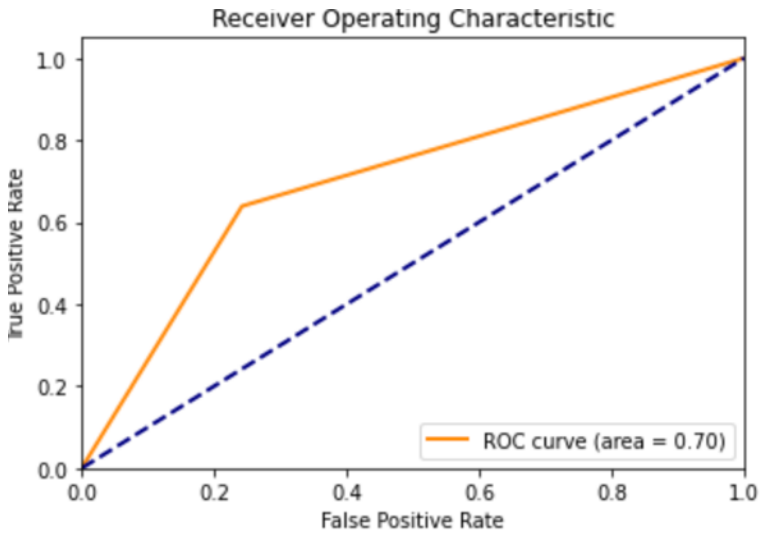
With the winning parameters being:

Best parameters found:

{'activation': 'relu', 'alpha': 0.5, 'hidden_layer_sizes': (80,), 'learning_rate': 'constant', 'solver': 'adam'}.

The model produced these results:

```
              precision    recall  f1-score   support

           0       0.98      0.76      0.85       843
           1       0.10      0.64      0.17        36

    accuracy                           0.75       879
   macro avg       0.54      0.70      0.51       879
weighted avg       0.94      0.75      0.83       879

Accuracy: 0.753
Precision: 0.101
Recall: 0.639
F1-score: 0.175
```

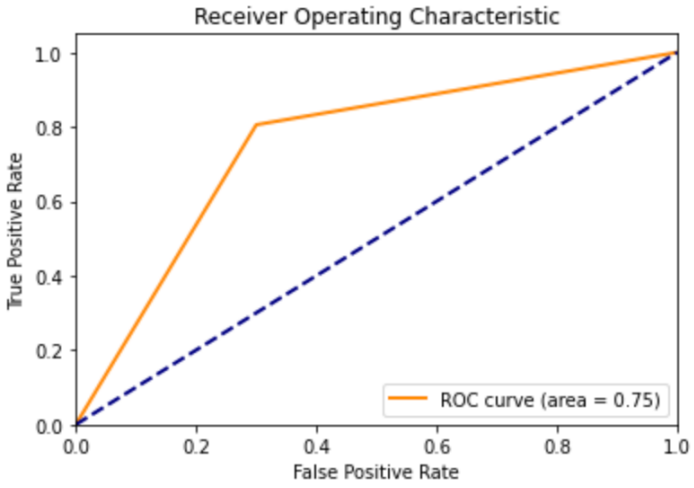


For the next MLP Classifier I wanted to relook at RFE and let it reduce the features that work best for the models performance for recall and accuracy. I am using sklearn ensemble to utilize Random Forest Classifier as an estimator to the RFECV feature selection. After getting the selected x train and x test, I am once again passing these variables to MLP Classifier to let GridSearchCV find optimal hyper parameters for the MLP model. It is worth noting that I started using a negative 1 for n jobs to utilize all processors on my computer as it started becoming

computationally heavy and why in code some parameter spaces are commented out once the best parameters are found. This model using RFE with MLP Classifier produced:

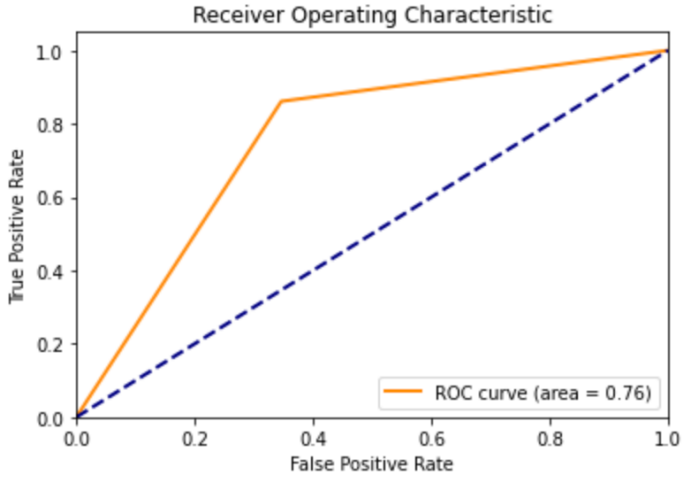
	precision	recall	f1-score	support
0	0.99	0.70	0.82	843
1	0.10	0.81	0.18	36
accuracy			0.70	879
macro avg	0.55	0.75	0.50	879
weighted avg	0.95	0.70	0.79	879

Accuracy: 0.704
Precision: 0.103
Recall: 0.806
F1-score: 0.182

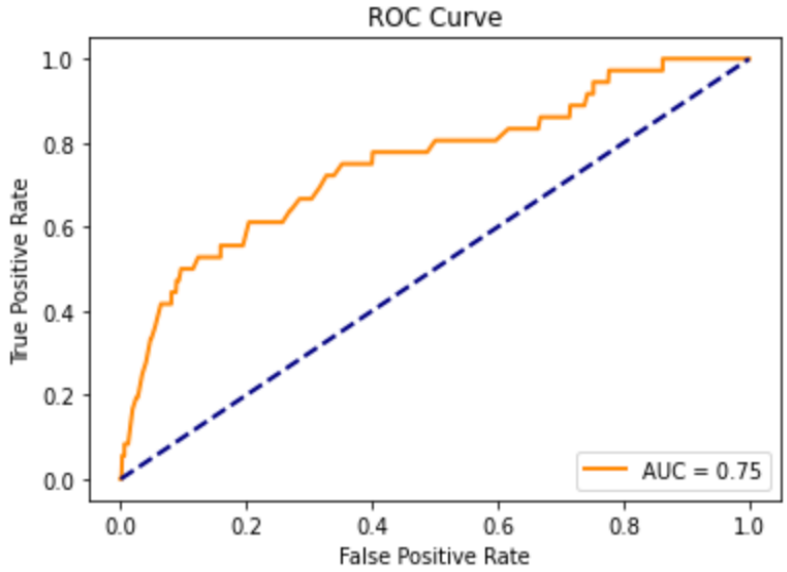


This was the best model of the group as again we were looking for high recall with respect to accuracy. The last MLP Classifier I want to explore was backed by implementing principal component analysis. To build this specific computation I used the sklearn pipeline to build a two step process of implementing PCA with MLP. This again used GridSearchCV to find all optimal parameters that were in the parameter space. For this model the metrics were as so:

```
              precision    recall  f1-score   support

           0       0.99      0.65      0.79       843
           1       0.10      0.86      0.17        36

    accuracy                           0.66       879
   macro avg       0.54      0.76      0.48       879
weighted avg       0.95      0.66      0.76       879

Accuracy: 0.662
Precision: 0.096
Recall: 0.861
F1-score: 0.173
```



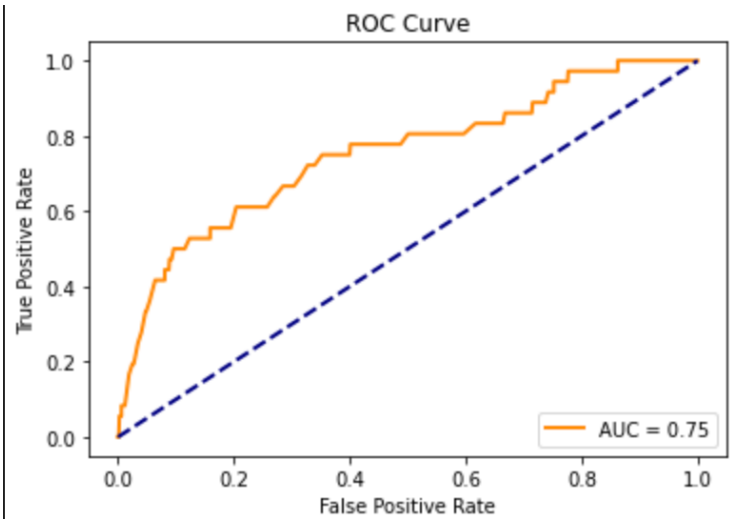
Along with MLP Classifier I built and optimized the Keras model. I experimented by starting simple and introduce more complexity as recall and accuracy improve. Both my implementations of Keras were 2 layer neural networks with a Droput layer with rate of 0.2 between the layers. Both models used the relu activation function in the first layer and sigmoid in the output layer. The first Keras model had 30 neurons in the first layer and 1 neuron in the outer layer. The results were as such:



	precision	recall	f1-score	support
0	0.98	0.74	0.84	843
1	0.10	0.64	0.17	36
accuracy			0.74	879
macro avg	0.54	0.69	0.51	879
weighted avg	0.94	0.74	0.82	879

Accuracy: 0.738
Precision: 0.092
Recall: 0.611
F1-score: 0.161

For this model I used MSE as the loss function and adam as my optimizer. The second implementation of the Keras model had 256 neurons in the first layer and 1 neuron in the outer layer. For this attempt I used adam optimizer again but lowered the rate from 0.001 to 0.0001 and used binary entropy as the loss function. The results are as followed:



```
              precision    recall  f1-score   support

           0       0.98      0.74      0.84       843
           1       0.10      0.64      0.17        36

    accuracy                           0.74       879
   macro avg       0.54      0.69      0.51       879
weighted avg       0.94      0.74      0.82       879

Accuracy: 0.737
Precision: 0.095
Recall: 0.639
F1-score: 0.166
```

This was my contribution to the final project. The following are the resources I used to help my portion and experiment with the models above.

Resources:

https://towardsdatascience.com/5-smote-techniques-for-oversampling-your-imbalance-data-b8155bdbe2b5

https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

https://nicolo-albanese.medium.com/tips-on-principal-component-analysis-7116265971ad

https://medium.com/dataman-in-ai/sampling-techniques-for-extremely-imbalanced-data-part-i-under-sampling-a8dbc3d8d6d8

https://scikit-learn.org/stable/tutorial/statistical_inference/putting_together.html

https://towardsdatascience.com/powerful-feature-selection-with-recursive-feature-elimination-rfe-of-sklearn-23efb2cdb54e