Final Report

Bryan Pritsker

*Machine Learning*

***Abstract*— This project presents an analysis of patient medical data, explores correlations and patterns to discern predictive indicators for strokes, and utilizes machine learning models to predict stroke occurrences.**

1. Introduction

The aim of this final project is to analyse a patient’s underlying medical data and predict if the patient will have a stroke. Strokes are critical medical emergencies that arise when blood flow to the brain is blocked or when there is sudden bleeding in the brain. There are several health conditions that could contribute to strokes, including diabetes, obesity, high blood pressure, and smoking. The dataset for the project, obtained from the “Kaggle” website, contains these health conditions and more medical information on over 5000 patients. Additionally, it provides various other features for each patient, such as gender, occupation, and marital status. The final column of the dataset, labeled “stroke”, contains binary results indicating whether the patient experienced a stroke (with “1” indicating a stroke and “0” indicating no stroke). Throughout the project, all features in the dataset are thoroughly examined to identify correlations and patterns related to strokes.

1. Data Preprocessing: Identifying Missing Values

Before proceeding to visualize the data and develop robust machine learning models capable of predicting strokes in patients, it's crucial to thoroughly understand the provided dataset. Upon initial inspection, while reviewing data types and identifying missing values, it became apparent that the DataFrame exhibited a total of 201 missing values in the “bmi” column. Given the dataset's size of approximately 5000 values, 201 missing values represented a significant portion that could not be overlooked. Since there was no clear reason behind the missing values in the “bmi” column, it was determined that these missing values followed the MCAR (“missing completely at random”) pattern. To address this issue, we will perform data visualization with the complete data, and try to examine if there are any existing correlations between the “bmi” feature and the other independent features in the dataset (not including the output column). If exploratory data analysis determines that there are correlations, then we will impute the missing values using a prediction model, rather than just delete the missing rows all together.

1. Data Visualization

Data visualization emerged as a powerful tool in our project for two key reasons.

1. *Revealing Imbalanced Data*

Firstly, during exploratory data analysis, we uncovered a significant imbalance in the dataset while examining the output column. Specifically, one pie chart comparing the percentage of patients who experienced a stroke to those who did not revealed that 95% of all patients in the dataset did not have a stroke. Such a striking imbalance, if left unaddressed, could lead to misleading results in the subsequent machine learning phase of the project. It's crucial to note that imbalanced datasets tend to favor the majority class, since the majority class has more instances for training. Consequently, prediction models that overlook these imbalances perform poorly on minority classes and yield unreliable overall accuracy scores. To produce the most favorable and reliable results, our prediction models later in the project will need to be tuned with specific imbalanced data techniques before they can make their final predictions. The techniques will be compared at the end of the project to see which one provides the most stable and reliable results.

1. *Underlining Connections Between Input Features*

In addition to highlighting imbalanced data, exploratory data analysis also unveiled significant correlations among features in the dataset. For instance, one bar chart showed that as a patient’s “bmi” increased, his likelihood of experiencing a stroke directly increased as well. Moreover, when considering the complete “bmi” data alongside “marital status”, the correlation between this combination of features and stroke incidence became even more pronounced. While the proportion of stroke incidences continued to rise with increasing “bmi”, unmarried patients experienced notably fewer stroke cases compared to married individuals (even when “bmi” data crossed into the “obese” range). These results not only suggested that certain input features were correlated with the output column, but they also indicated that specific input features had strong correlations with one another: especially the “bmi” column and other input features. As a result, data visualization supported the idea of imputing the missing values. Moreover, it emphasized that we should fill in the missing data by using a prediction model. Unlike conventional methods that just use the mean or median to fill missing values, our approach taps into the potential relationships between known attributes and the missing data. By leveraging these connections, the prediction model can more accurately replace the missing values. The prediction model that we opted to utilize was linear regression, since the missing “bmi” values were continuous values. Linear regression is one of the most effective prediction models when dealing with continuous variables.

1. Label Encoding and One Hot encoding

Prior to employing linear regression, we first had to transform the categorical variables in the dataset into numerical values. Considering that our dataset had five categorical columns, we decided to apply label encoding to all the non-numerical features. Upon applying the label encoder however, we noticed that some of the columns that got converted now had a “meaningful order between their classes” (when originally, these columns were supposed to contain nominal variables: values with no meaningful order). Consequently, we next turned to “one hot encoding”, a method that preserves nominal variables by representing each unique class with its own binary vector. Due to its slower processing and the significant increase in data dimensionality, we limited one-hot encoding to two pivotal columns: “work\_type” and “smoking\_status”. Both columns were composed of nominal classes, and it was far more useful to preserve this relationship for the subsequent linear regression algorithm that we implemented.

1. impute missing values

After encoding the data, we finally could utilize linear regression to replace the missing “bmi” values. We began by dividing the DataFrame into two subsets: one containing rows with *missing data* and the other with *complete data*. Next, we created the matrix *X\_train* by dropping the columns “bmi” and “stroke” from *complete\_data*, and the vector *y-train* by slicing the “bmi” column from *complete\_data*. Note that it was crucial to remove the “stroke” column from *X\_train* to prevent data leakage. Data leakage arises when the prediction model gains access to future information that it is supposed to predict. By incorporating the output column into the training dataset, the model essentially learns to predict the output based on both input features and the output itself, leading to inaccurate or biased results. Thus, after correctly splitting *complete\_data* into *X\_train* and *y\_train*, we proceeded to fit these components to the linear regression model. The linear regression model identified the line of best fit by utilizing the columns in *X\_train* to minimize the sum of squared differences between the observed dependent variable and the values predicted by the model—a method known as the least squares method. The least squares method is what allowed the model to determine the optimal slope (set of weights for each column) and y-intercept that would minimize these differences (the training error). Once the best-fit line was determined, we plugged the *missing\_data* back into our predicted linear model and replaced the missing “bmi” values. Consequently, the linear regression model effectively utilized other features in our DataFrame to impute missing “bmi” values as continuous data. This result was accomplished without causing any data leakage and allowed us to use our updated data to build machine learning models.

1. logistic regression

Accordingly, the first machine learning algorithm that we exercised was logistic regression.

1. *Explanation of Algorithm*

Logistic regression was a suitable prediction algorithm for our dataset because it is a classification algorithm that works extremely well with binary outputs. Unlike linear regression which assumes that input and output values are linearly correlated, logistic regression maps values into a range between 0 and 1 by using the sigmoid function to make predictions on categorical outputs. Essentially, the sigmoid function transforms the continuous output of the linear combination into a probability value between 0 and 1 (on a graph it looks like an “S” shaped curve). The output of this function is then used to classify instances into one of the two classes: if the output is greater than 0.5, the instance is classified as belonging to Class 1; otherwise, it is classified as belonging to Class 0. Note that the weights in the linear combination part of the logistic regression model are obtained differently than in linear regression. The key difference lies in the cost function used for optimization. In logistic regression, the cost function is the log loss function, not the mean squared error (MSE) function that is used in linear regression. Thus, to minimize this log loss error, weights are obtained iteratively using gradient descent.

1. *Standardization of Numeric Features*

To ensure that the logistic regression model would perform its best, we first scaled the numerical features in the *encoded\_df* by using “Standard Scaler” normalization. Standard Scaler is a type of normalization which subtracts the mean of the column from each data point, and then divides the result by the standard deviation of the column. We applied Standard Scaler on the columns: “age”, “bmi”, and “avg\_glucose\_level”. Although it was not necessary to scale features before performing logistic regression, it is good practice to do this for most classification models (to ensure that the features in the dataset hold equal weightage in the model).

1. *Feature Selection*

Next, it was time to implement feature selection. Feature selection is very useful for logistic regression models because it reduces the dimensionality of the dataset. This in turn simplifies the model, making it easier to interpret, faster to execute, and less prone to data biases. The feature selection technique that we decided to use was a form of the wrapper method known as “sequential backward elimination”. We implemented the wrapper method over the filter method because the wrapper method considers interactions between input features and assesses their performance with respect to a machine learning algorithm. The data visualization we conducted earlier showed us that it is crucial to use the connections between different features in our prediction algorithms. During the process of sequential backward elimination, we fitted all the features to a Decision Tree model, calculated the performance of these features based on the “Roc AUC” score (with tol = 0.03), and then greedily removed the least relevant features (one at a time) that did not improve the performance of the algorithm. This process stopped once removing any extra feature would cause the Roc AUC score to drop by 0.03. We employed Roc AUC scoring over regular model accuracy because the ROC curve is designed to evaluate how efficiently the model is performing. The higher the Roc AUC score, the better the model is at distinguishing between the positive (majority) and negative (minority) classes (thus it is an extremely useful metric when the data is imbalanced). In our case, the wrapper method dropped one column, “work\_type\_0”, from the selected features, suggesting that removing this feature could improve the model. It's worth noting that the sequential backward eliminator did not drop the column “bmi”, further emphasizing the importance of this feature to the model.

1. *Logistic Regression Implementation and Imbalanced Data Techniques*

After the feature selection, the last step in our logistic regression model was to correct the imbalance in the training data. To reiterate, the main goal of our project was not to build a model that would perform well on the whole dataset. Instead, our goal was to optimize our prediction algorithm to accurately predict the patients that had a stroke, while maintaining stable results for the non-stroke patients. To accomplish this task, we utilized “random oversampling” as our imbalanced data technique. During the random oversampling procedure, the minority class training examples (stroke patients) were selected multiple times until the training dataset became balanced. Then, to make the logistic regression model specifically be concerned with stroke patients, we applied a custom weight to the stroke patients’ class (weight = 1.4). By assigning a higher weight to the minority class, the logistic regression model was informed that misclassifications of stroke patients will be more costly than misclassifications of non-stroke patients (it will more greatly affect the log-loss cost function). This adjustment helped further address the class imbalance and forced the model to pay more attention to correctly classifying instances of stroke patients. Also, to regularize the model and make sure the model avoided fitting too closely to the training data, we added a regularization parameter (C = 0.5) to prevent overfitting.

1. *Logistic Regression Results*

Before applying any imbalanced data techniques, the logistic regression model had a total accuracy of 93.8%. Although this result seemed strong, it was in fact very misleading. When viewing the confusion matrix results of the regular logistic regression model, this algorithm made 79 false positive predictions out of 80 total stroke patients in the test data. This meant that 98.75% of the time, if the model viewed a stroke patient, it incorrectly predicted that this patient did not experience a stroke. Clearly, the regular logistic regression performed terribly on the minority class, and its results were highly biased towards the majority (non-stroke patients) class. On the other hand, when reviewing the results of the logistic regression model with random oversampling, this algorithm made significant improvements in predicting stroke patients. It successfully predicted 69 out of 80 stroke patients in the test data, good enough for an 86.25% specificity score (True Negative / “True Negative + False Positive”). In addition to this, the model also maintained a 70% recall score! Hence, this model not only maintained a decent performance on the majority class, but it also came away with a consistent accuracy in identifying stroke instances. For these reasons, the random oversampled logistic regression model accomplished our initial objectives.

1. Robustness of Random Oversampled Logistic Regression Model

When comparing the logistic regression (LR) model with random oversampling to another LR model using a different method for imbalanced data, several noteworthy findings arise. The non-oversampled LR model tackled the imbalanced data by assigning a higher weight to the minority class instead of generating new instances of training samples. Specifically, a weight of 32 was assigned to the minority stroke class. Note that it is crucial to closely examine these two models, especially in our project dealing with sensitive medical data. We want to avoid fabricating training instances unless it improves the model's performance and makes it better prepared for real-world testing scenarios. Despite the adjustment made to the non-oversampled LR model, it achieved a final accuracy of 70.1%, which was 0.5% lower than the random oversampling LR model. Both models attained the same specificity score, achieving an 86.25% accuracy when identifying stroke patients. However, the recall score for the random oversampling LR model was slightly higher, accurately predicting 7 more non-stroke patients. Though these differences are modest, they demonstrate the robustness of the random oversampling LR model. It not only performed on par with the non-oversampled LR model, but it outperformed it by a few patients.

1. Robustness of LR model with replaced missing instances vs deleted missing instances

When cross examining the random oversampled LR model with replaced missing instances versus the random oversampled LR model with deleted missing instances, the discrepancies are even more striking. In the case of the latter model (with deleted rows of missing instances), this algorithm scored a 3% lower test accuracy than the LR model with imputed missing “bmi” values. That is a substantial difference. Additionally, while both models achieved the same specificity scores, the LR model with deleted missing instances got a worse recall score. It made 18 more false positive predictions on non-stroke patients, yet it viewed 30 less non-stroke patients than the LR model with replaced missing values. Consequently, this highlighted the importance of replacing the missing "bmi" values using linear regression. Not only did we select the most suitable algorithm for imputing continuous values, but we also strengthened the resilience of our LR model.

1. neural network

The last algorithm that we applied to the stroke dataset was the “neural network”. Neural networks are constructed using layers of interconnected nodes or "neurons". These layers can learn complex patterns and relationships within the data, making them effective for stroke prediction.

1. *Creation of Neural Network BEFORE Imbalanced Data Techniques*

In our neural network model, we initialized two hidden layers using PyTorch, with 10 neurons in the first layer and 5 neurons in the second. The first hidden layer applied the ReLU activation function on the output, while the second hidden layer employed the sigmoid activation function. We then trained this network of neurons through a process called “backpropagation”: adjusting the weights of the connections between neurons (using gradient descent) to minimize the difference between the predicted and actual outcomes. This approach allowed the neural network to learn from the data without the need for explicit feature selection. There was one major problem though that the neural network encountered. The network lacked diverse training data on the minority class (the stroke patients). Thus, even when we tried to apply random oversampling to transform the training data, the neural network overfit to these training instances and performed terribly on predicting stroke patients in the test data.

1. *Using SMOTE before fitting data to the Neural Network*

Therefore, to provide the neural network with more exposure to unique instances of the minority class, we decided to use an imbalanced data technique known as “SMOTE”. SMOTE proved to be a great complement for the neural network model because it randomly generated new samples of stroke patients for the network to learn. The way SMOTE generated these samples was by finding the k-nearest minority neighbors for each minority training instance. Then, SMOTE randomly created new instances by using data that existed along the lines that joined the minority sample and its selected neighbors. These new instances were generated until the training dataset became balanced between stroke and non-stroke patients. It is important to stress here that similar to our imputation of the missing “bmi” values, SMOTE was applied only to the training data after the encoded\_df was split using train\_test split. We did this to prevent data leakage and not let the neural network have access to the testing data.

1. *Neural Network Results*

Now that we had this transformed training data, we finally were able to fit this data to the neural network model. Unlike the logistic regression model, no weights needed to be applied in the neural network because the data was already balanced out using SMOTE. The neural network was trained using the “log-loss” function and the weights were back propagated over 1000 iterations. When viewing the results of the confusion matrix, the neural network scored a 72.4% testing accuracy, a slight improvement over the logistic regression model. Furthermore, the network successfully predicted 54 out of 62 stroke patients in the test data, earning an 87% specificity score (also an improvement over the logistic regression model). Where the neural network really shined was in limiting the number of false negatives that it predicted, earning an overall recall score of 71.5%. Consequently, the neural network was able to expand upon the results of the logistic regression model and improve nearly all factors on its stroke predictions.

1. conclusion

This project has shown how machine learning can be employed to accurately predict strokes using patient medical data. Through comprehensive data preprocessing, visualization, and model implementation, particularly leveraging logistic regression and neural networks, the study achieved promising results in identifying potential stroke cases while addressing imbalanced data challenges.

References

1. Analysis, Dossier. “Exploring Common Data Visualization Techniques Using Matplotlib and Seaborn.” *Medium*, Medium, 28 Apr. 2024, medium.com/@dossieranalysis/exploring-common-data-visualization-techniques-using-matplotlib-and-seaborn-961f09095679.
2. Brownlee, Jason. “8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset.” *MachineLearningMastery.Com*, 15 Aug. 2020, machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/.
3. Brownlee, Jason. “Smote for Imbalanced Classification with Python.” *MachineLearningMastery.Com*, 16 Mar. 2021, machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/.
4. “Defining a Neural Network in Pytorch¶.” *Defining a Neural Network in PyTorch - PyTorch Tutorials 2.3.0+cu121 Documentation*, pytorch.org/tutorials/recipes/recipes/defining\_a\_neural\_network.html. Accessed 8 May 2024.
5. Detective, The Data. “Preprocessing: Regression Imputation of Missing Continuous Values.” *Medium*, Towards Data Science, 4 Dec. 2019, towardsdatascience.com/preprocessing-regression-imputation-of-missing-continuous-values-f612179bafb4.
6. Fedesoriano. “Stroke Prediction Dataset.” Kaggle, 26 Jan. 2021, www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset/data.
7. guest\_blog. “10 Techniques to Solve Imbalanced Classes in Machine Learning (Updated 2024).” *Analytics Vidhya*, 17 Jan. 2024, www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/#:~:text=SMOTE%20(Synthetic%20Minority%20Oversampling%20Technique,chosen%20point%20and%20its%20neighbors.
8. “KDE Plot Visualization with Pandas and Seaborn.” *GeeksforGeeks*, GeeksforGeeks, 21 Dec. 2023, www.geeksforgeeks.org/kde-plot-visualization-with-pandas-and-seaborn/.
9. Navlani, Avinash. “Python Logistic Regression Tutorial with Sklearn & Scikit.” *DataCamp*, DataCamp, 16 Dec. 2019, www.datacamp.com/tutorial/understanding-logistic-regression-python.
10. Pykes, Kurtis. “Pytorch Tutorial: Building a Simple Neural Network from Scratch.” *DataCamp*, DataCamp, 6 July 2022, www.datacamp.com/tutorial/pytorch-tutorial-building-a-simple-neural-network-from-scratch.
11. Sowjanya, A Mary, and Owk Mrudula. “Effective Treatment of Imbalanced Datasets in Health Care Using Modified Smote Coupled with Stacked Deep Learning Algorithms.” *Applied Nanoscience*, U.S. National Library of Medicine, 2023, www.ncbi.nlm.nih.gov/pmc/articles/PMC8811587/.
12. Swalin, Alvira. “How to Handle Missing Data.” *Medium*, Towards Data Science, 19 Mar. 2018, towardsdatascience.com/how-to-handle-missing-data-8646b18db0d4.
13. Tam, Adrian. “Develop Your First Neural Network with Pytorch, Step by Step.” *MachineLearningMastery.Com*, 7 Apr. 2023, machinelearningmastery.com/develop-your-first-neural-network-with-pytorch-step-by-step/.