Stroke Predictions

Chathumini Abeyratna, Saadiya Allahbaksh, Kevin Karnani

June 8, 2021

Abstract

We were curious to see the effect that certain features had in the possibility of a person having a stroke and how well machine learning models predicted stroke. We decided to dive deep into the background research we found, any related work we came across, the dataset we will be using, and the machine learning models that we created to predict stroke based on features. We found that for many cases, `Age` contributed the most to strokes, followed by `BMI` and `Smoking_Status`. Conversely, factors like `Ever_Married`, `Gender`, `Hypertension`, and `Heart_Disease` had little to no effect on strokes.

Contents

1 Introduction

The World Health Organization (WHO) identifies strokes as the second leading cause of death globally [6]. A stroke happens when a person's blood supply to their brain is interrupted or reduced, causing brain cells to die within minutes. It prevents the brain tissue from getting the oxygen and nutrients that it needs and is responsible for approximately 11% of total deaths. To see what effect certain factors had on people who suffered strokes, we decided to create 3 Machine Learning Models: Logistic Regression, Support Vector Machines, and Random Forest. We did this by not only writing our own code to create the models, but we also compared the yielded metrics to those from built-in models of `Scikit-Learn`, a built-in Machine Learning Python library.

2 Dataset

The dataset that we will be using for the models were taken from Kaggle [2]. There are 5110 entries of individuals, their relevant feature details, and whether they had a stroke or not.

2.1 Columns

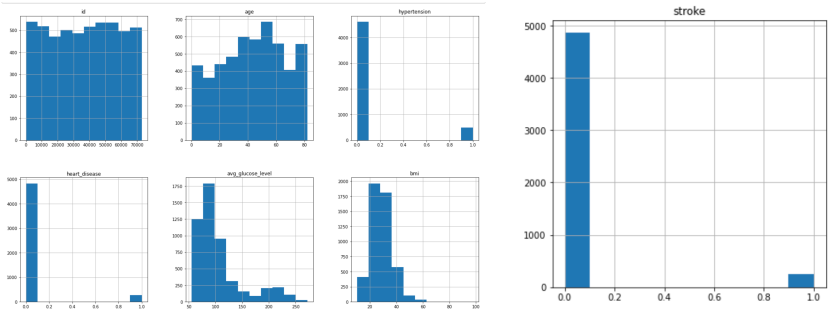
- `id`: unique identifier
- `gender`: "Male", "Female" or "Other"
- `age`: age of the patient
- `hypertension`: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
- `heart_disease`: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
- `ever_married`: "No" or "Yes"
- `work_type`: "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
- `Residence_type`: "Rural" or "Urban"
- `avg_glucose_level`: average glucose level in blood
- `bmi`: body mass index
- `smoking_status`: "formerly smoked", "never smoked", "smokes" or "Unknown"
- `stroke`: 1 if the patient had a stroke or 0 if not

3 Methodology

Since we have three members, we decided to tackle the project by assigning a machine learning model to each member. Chathumini focused on creating the Logistic Regression model, while also exploring the differences between batch and stochastic gradient descent and implementing k-fold cross validation. Saadiya worked on creating the SVM model and plotting their Precision-Recall Curves. Kevin implemented the ID3 and Random Forest models, compared their yielded metrics, and determined the most important features.

3.1 Exploratory Data Analysis and Feature Engineering

Before we started creating any models, we collectively worked on the Exploratory Data Analysis (EDA) of the stroke risk dataset. For EDA, we plotted histograms and a heatmap to visualize the data and the correlation between the features. Through EDA, we learned that there were some missing values for certain features, unnecessary columns, categorical variables, and most importantly, that there was an imbalance in the data in terms of the stroke feature. Figures 1a and 1b show how the data distribution was initially:



(a) Initial Feature Distribution (b) Initial Label Distribution

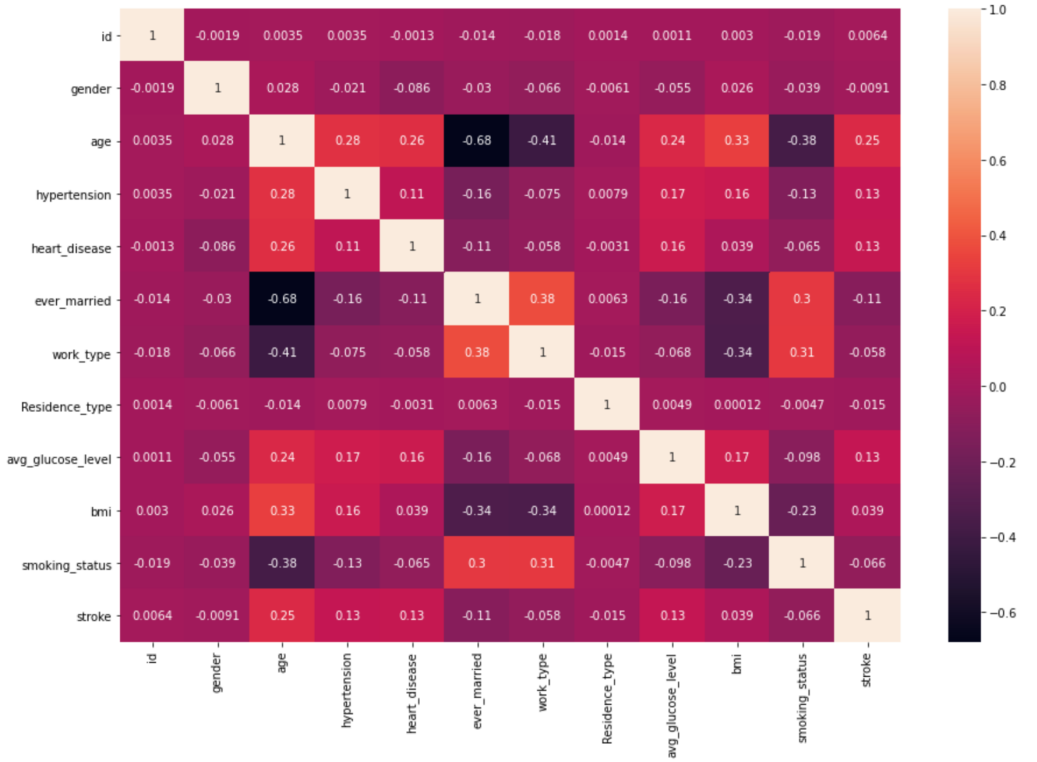
To fix these issues, we performed feature engineering [3]. We filled the missing values with that features average result and discarded any unnecessary columns. We then changed the categorical data to numerical data. To fix the imbalance issue, we used the Synthetic Minority Oversampling Technique (SMOTE), which works by selecting examples that are close in the feature spaces, draws a line between the examples in the feature space, and draws a new sample at a point along that line. We also standardized the data by computing the following equation on our features X:

$$X_{standardized} = \frac{X - \mu(X)}{\sigma(X)} \tag{1}$$

Where $\mu(X)$ is the mean of X, and $\sigma(X)$ is the standard deviation of X.

Figures 2a and 2b show feature correlations before and after: replacing N/A values, converting categorical data to discrete data, and standardizing data using Equation (1):

(a) Initial Covariance Matrix



(b) Final Covariance Matrix

3.2 Picking Models

With the feature engineering and SMOTE applied to the data, we are ready to proceed. Given the nature of the data, we are facing a supervised learning problem, more specifically a classification problem. Given the severity of the issue at hand, we wanted to compare some more basic classification algorithms, like Logistic Regression [4], and see how it would fare against different discriminative algorithms such as SVMs [5] and Decision Trees. In addition, we wanted to find out which features mattered the most in this case, which is the Random Forest algorithm was picked out of all possible Decision Tree based algorithms.

4 Methods

4.1 Logistic Regression

The Logistic Regression Model is a statistical model that is used to solve classification problems and basically models the probability of a certain class or event happening. In this case, we will be using it to predict whether a person with certain features will get a stroke or not. We will be testing it with both batch gradient descent and stochastic gradient descent. We will also explore k-folding and seeing the effects it might have on the model's results.

4.1.1 Batch Gradient Descent

For this model, I split the data 2/3 for the training set and 1/3 for the testing set. I set random initial values for theta, which are the parameters, and used a learning rate η of 0.01. N is the number of observations. This model ran until the absolute value change in the loss of data was less than 2^{-23} or after 1,500 iterations had passed.

While the model ran, it:

- Updated the parameters (theta) using batch gradient descent. Batch gradient descent computes the gradient on the whole dataset. To minimize the cost function and find the optimal solution, we need to run the gradient descent on each parameter θ_j:

$$\theta_j \leftarrow \theta_j - \eta \frac{\partial}{\partial \theta_j} J(\theta) \tag{2}$$

  Where the value of the gradient in Equation (2) is:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \Big(h_\theta(x^{(i)}) - y^{(i)} \Big) x_j^{(i)} \tag{3}$$

  In Equation (3), $h_\theta(x^{(i)})$ is the sigmoid function, or logistic function. It maps input values to a limited interval. It squeezes any number in $\mathbb{R}$ to the open interval $(0, 1)$,

so it is well suited for classification purposes. This hypothesis function approximates the probability of the actual output being equal to 1 and is equivalent to:

$$P(y = 1 \mid \theta, x) = \frac{1}{1 + e^{-\theta^T x}} \tag{4}$$

Therefore, using Equation (4), our sigmoid function is defined as:

$$h_\theta(x^{(i)}) = \frac{1}{1 + e^{-\theta^T x^{(i)}}} \tag{5}$$

Plugging Equation (5) into the gradient descent function, we get the final formula for updating the parameters using batch gradient descent:

$$\theta_j \leftarrow \theta_j - \frac{\eta}{N} \sum_{i=1}^{N} \Big(h_\theta(x^{(i)}) - y^{(i)} \Big) x_j^{(i)} \tag{6}$$

Equation (6) can then be vectorized into easier terms, for code optimizations:

$$\theta \leftarrow \theta - \frac{\eta}{N} X^T \Big(h_\theta(X) - Y \Big) \tag{7}$$

- Computed the loss of the data using the logistic regression cost function. The cost function basically summarizes how well the model is behaving - the smaller the value is, the better. Since we're dealing with a binary classification problem, we define the cost for the two cases separately:

$$J(\theta) = \begin{cases} -\ln(h_\theta(x)) & y = 1 \\ -\ln(1 - h_\theta(x)) & y = 0 \end{cases} \tag{8}$$

Equation (8) can be simplified to:

$$J(\theta) = -y^{(i)} \ln(h_\theta(x)) - (1 - y^{(i)}) \ln(1 - h_\theta(x)) \tag{9}$$

For N observations, we modify Equation (9) to:

$$J(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \Big(y^{(i)} \ln(h_\theta(x)) + (1 - y^{(i)}) \ln(1 - h_\theta(x)) \Big) \tag{10}$$

Equation (10) can then be vectorized into easier terms, for code optimizations:

$$J(\theta) = -\frac{1}{N} \Big(Y^T \ln(h_\theta(X)) + (1 - Y)^T \ln(1 - h_\theta(X)) \Big) \tag{11}$$

Like stated above, this while-loop ran until either it created ideal parameters (absolute value change in the loss of data was less than 2^{-23}) or until 1,500 iterations happened.

4.1.2 Stochastic Gradient Descent

Then, we switched out the batch gradient descent for stochastic gradient descent. Stochastic gradient descent computes the gradient using a random instance at a single time, so it's typically much faster than batch gradient descent. Algorithm (1) shows the pseudocode:

Algorithm 1 Stochastic Gradient Descent

Ensure: $m > 0$
1: **for** i in range(m) **do**
2: $\quad \theta_j \leftarrow \theta_j + \eta(\hat{y} - y^{(i)})X_j^{(i)}$
3: **end for**

4.1.3 K-Folds Cross Validation

Our last stage with logistic regression was to see how the model worked in k-fold cross validation. Cross validation is a resampling, statistical method used to estimate the skill of a model. k refers to the number of groups that a given data set will be split into. Once the data is split, each group will get a chance to be the testing set and the data will be put through both the logistic regression with batch gradient descent and stochastic gradient descent. Algorithm (2) shows the pseudocode:

Algorithm 2 K-Folds Cross Validation

Ensure: $k > 1$
Randomly shuffle the dataset
2: Split the dataset into k groups
for each unique group **do**
4: $\quad$ Take the group as the testing set
$\quad$ Use the remaining groups as the training set
6: $\quad$ Perform logistic regression with the training and testing set
$\quad$ Compute and save the accuracy, precision, recall and f1 measure in an array
8: **end for**
Display the accuracy, precision, recall and f1 measure arrays for each fold

4.2 Support Vector Machines

The Support Vector Machine (SVM) is a supervised learning model that can be used to solve classification problems. The SVM model finds a hyperplane that distinctly classifies data points in an N-dimensional space, where N is the number of features. For our dataset, 10 features are being used to predict whether a person will get a stroke or not. This model chooses the best hyperplane that separates the data into positive and negative values and is the furthest away from the closest data points in order to classify the data points. For

the binary class label, the dataset classified 0 as patients who did not have a stroke and 1 as patients who had a stroke. For the SVM model, 0 is mapped to -1 and 1 is mapped to $+1$.

The hyperplane is found by minimizing a cost function to find the optimal weights (w) and the intercept (b). The SVM model is defined as:

$$f(x) = \text{sgn}(wx + b) \tag{12}$$

A cost function measures how the model is doing in terms of finding a hyperplane that separates the negative and positive data points with the biggest margin possible while keeping the misclassification of the data as low as possible. For the SVM algorithm, we decided to minimize the cost function by using stochastic gradient descent. By expanding upon Equation (12), the formula is given below:

$$J(w) = \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{i=1}^{N} \max(0, 1 - y^{(i)}(wx^{(i)} + b)) \tag{13}$$

Where $\max(0, 1 - y^{(i)}(wx^{(i)} + b))$ is the hinge loss function. A hinge loss function is a loss function that is used for binary classification. The gradient of Equation (13) is calculated as follows:

$$\frac{\partial}{\partial w}J(w) = \frac{1}{N}\sum_{i=1}^{N} \begin{cases} w & \max(0, 1 - y^{(i)}(wx^{(i)} + b)) = 0 \\ w - Cy^{(i)}x^{(i)} & \max(0, 1 - y^{(i)}(wx^{(i)} + b)) \neq 0 \end{cases} \tag{14}$$

For the stochastic gradient descent algorithm, the maximum number of iterations is set to 2048, the learning rate is set to 0.000001 and the cost threshold is set to 0.01. The model will train the data until the model starts to converge, i.e., when there is no significant decrease in the current cost when compared to the previous cost.

4.3 Random Forest

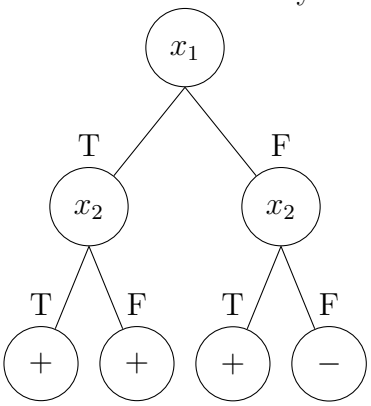
Random forest, as the "forest" part of the name implies, consists of a large number of individual Decision Trees (DTs) that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction. For some background, we picked this route because DTs, while still discriminative in nature, perform classification in a manner that is fundamentally different than other algorithms like Logistic Regression and SVMs. Each internal node tests an attribute, each branch is an attribute value, and each leaf assigns a classification. While the latter two algorithms use metric cost functions between features (Euclidean/Manhattan Distance) and sometimes a kernel function (Gaussian/Logistic Kernel), Decision Trees apply a sequence of decisions or rules that often depend on a single variable at a time.

As we run through using a Decision Tree, we start at the root node, check on a feature and value, and make a decision from there, taking us to the next level of the tree. The way we decide which features we split on is through information gain, which is based on sample entropy. Sample Entropy (Equation (15)) and Information Gain (Equation (16)) can be computed as follows:

$$H(Y) = -\sum_{i=1}^{N} P(Y = y_i) \log_2 P(Y = y_i) \tag{15}$$

$$I(Y) = H(Y) - \sum_{i=1}^{N} \frac{P(Y = y_i)}{P(Y)} H(Y_i) \tag{16}$$

The easiest way to run through our own tree would be to do this is if we deal with binarized data and construct binary DTs, for example:



However, since determining the optimal value of continuous data to split on is no easy feat, all unique values were used instead. While every tree with a branching factor greater than two can be recomposed into an equivalent binary tree, so we decided to do this to reduce recursive complexity. Now that we have a way to decide which feature to split on, we want to first pick the features that yield the highest Information Gain, or the lowest sample entropy. This is because the entropy is a measure of the "randomness" of a system, so we want to find the most deterministic features. Algorithm (3) shows us how to find a proper decision tree using the ID3 (Iterative Dichotomiser 3):

Algorithm 3 ID3

```
if No Data Left then
    return mode(Y)
end if
if All Y values are equal then
    return Y[0]
end if
if Only 1 feature in features then
    return mode(Y[feature])
end if
Find Best Feature and Delete from Features
Find Data above and below Value associated with Feature
return [ID3(below, Features), ID3(above, Features)]
```

From there, we have constructed our DT using the ID3 algorithm. From here, we need to cover the idea behind a Random Forest. Random Forests do what is called **Bagging**, otherwise known as Bootstrap Aggregation. In essence, this means that in a collection of trees, each tree structure is different when subjected to random sampling. This is a policy that takes advantage of how sensitive a decision tree is to the data it is trained on (due to DT's natural tendency of high variance) [7]. Algorithm (4) shows us how to construct a Random Forest:

Algorithm 4 Random Forest

```
for i in range(num_trees) do
    randomly sample from data
    randomly sample from features
    append ID3(data, features) to forest
end for
return forest
```

To classify the data, we then recurse through the tree using the data we wish to look at, and return the classification yielded by the leaf node we land on.

5 Evaluation

For reference, Accuracy is defined as:

$$\frac{TP + TN}{TP + FP + TN + FN} \tag{17}$$

Precision is defined as:

$$\frac{TP}{TP + FP} \tag{18}$$

Recall is defined as:

$$\frac{TP}{TP + FN} \tag{19}$$

F1-Score is defined as:

$$\frac{2(18)(19)}{(18) + (19)} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{20}$$

Where:

- TP is the count of where both predictions and true values are positive
- FP is the count of where predictions are positive but true values are negative
- TN is the count of where both predictions and true values are negative
- FN is the count of where predictions are negative but true values are positive

5.1 Logistic Regression

After running the logistic regression file multiple times, we noticed that the results for each situation between each run were very similar. So, we decided to include the results from one run to maintain space. On the left-hand side of Table 1, you can see the different experiment setting each logistic regression model created. Table 1 shows the accuracy, precision, recall and F1-measure results. The accuracy for the Logistic Regression Built in function was around 81.74%. All the models worked well in terms of accuracy, but the one that came the closest was the ones that involved k-fold cross validation. These tests almost always had an 80% or higher accuracy rate. The same could be said for precision and F1-measure. However, for recall, the models without k-fold cross validation had a more similar result to the built in function, which was around 85%.

Between Batch gradient descent and stochastic gradient descent, there wasn't that much of a difference in the results. They were either the same percentage, or off by 1 point. K-Fold Cross Validation seems to have a huge effect in making the model more precise. It is very clear in these results that it's better to do k-fold cross validation than not in order to get more accurate results. This makes sense, since it creates multiple training groups and can create a better model as a result of this.

Table 2 shows the True Positive, False Positive, True Negative and False Negative Results for the Logistic Regression Model in the different experimental settings.

Algorithm	Accuracy	Precision	Recall	F1-Score
Batch Gradient Descent	77.625	73.546	86.827	79.637
Stochastic Gradient Descent	77.999	74.397	85.899	79.736
K-Folds = 5 (BGD)	80.905	77.929	86.259	81.876
K-Folds = 5 (SGD)	80.864	77.655	86.662	81.910
K-Folds = 10 (BGD)	80.936	77.718	86.779	81.972
K-Folds = 10 (SGD)	81.440	77.968	87.695	82.523
Sci-Kit Learn	81.739	79.988	85.034	82.434

Table 1: Metrics of Different Algorithms (Averaged)

Algorithm	TP	FP	TN	FN
Batch Gradient Descent	1404	505	1087	213
Stochastic Gradient Descent	1389	478	1114	228
K-Folds = 5 (BGD)	838.6	237.6	734.2	133.6
K-Folds = 5 (SGD)	842.6	242.4	729.4	129.6
K-Folds = 10 (BGD)	421.6	121	365.1	64.3
K-Folds = 10 (SGD)	426.1	120.5	365.5	59.9

Table 2: Confusion Matrix Results (Averaged)

5.2 SVMs

Similar to the logistic regression script, for multiple runs of the support vector machine (SVM) script, SVM produced statistics that were very similar for multiple runs. Therefore, results from one run are included in Table 3. This table shows the accuracy, precision, recall, and f1-measure for the SVM algorithm and the built in SVM function. Table 4 shows the True Positive, False Positive, True Negative and False Negative Results for the SVM model.

Algorithm	Accuracy	Precision	Recall	F1-Score
SVM	78.218	74.931	84.625	79.483
Sci-Kit Learn	82.674	78.308	90.25	83.856

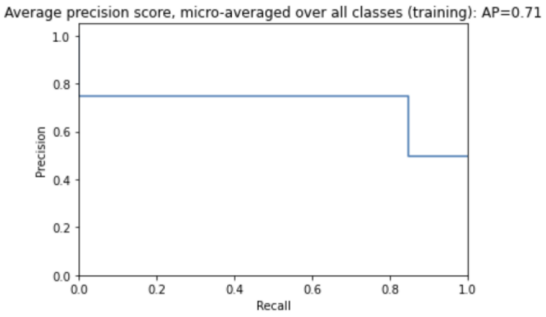
Table 3: Metrics

Algorithm	TP	FP	TN	FN
SVM	1354	453	1156	246
Sci-Kit Learn	1444	400	1209	156

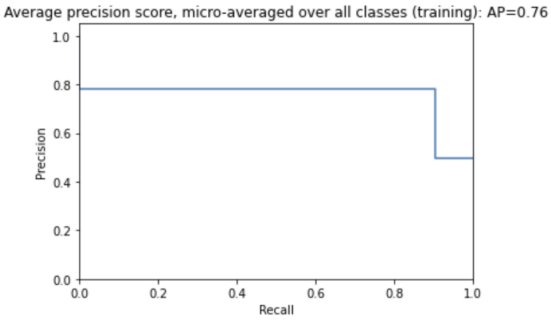
Table 4: Confusion Matrix Results

5.2.1 Precision-Recall Curves

Figures 3a and 3b plot the precision (y-axis) against the recall (x-axis).



(a) PR Curve of Our Model



(b) PR Curve of Sci-Kit Learn's Model

5.3 Random Forest

Unlike Logistic Regression and SVMs, our Random Forest model was somewhat flawed as we assumed all data was continuous rather than looking at a mix of continuous and discrete data. Due to this, not was our DT model unnecessarily expensive in both time and space complexities, but it caused our Random Forest model to be not much of an improvement over a regular DT. Table 5 and Table 6 show the results of ID3 and Random Forest:

Algorithm	**Accuracy**	**Precision**	**Recall**	**F1-Score**
ID3	70.365	65.722	86.085	74.538
Sci-Kit Learn	89.685	88.503	91.404	89.93

Table 5: ID3 Metrics

Algorithm	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Random Forest	71.206	66.651	85.776	75.014
Sci-Kit Learn	94.734	93.773	95.918	94.833

Table 6: Random Forest Metrics

6 Conclusion

6.1 Overall Analysis

From our data analysis and the graphs we plotted, people over the age of 50 are more likely to get a stroke compared to people under the age of 50. Individuals with hypertension (high blood pressure) are more likely to get a stroke. Males are more prone to a stroke compared to females.

The Logistic Regression model seemed to perform similarly, whether it was created with Batch Gradient Descent or Stochastic Gradient Descent. However, the results very clearly

show that running the model with k-folds will significantly increase the accuracy, by at least 3%. Additionally, the higher the number of folds, the greater the accuracy. For our results, it seemed like Logistic Regression with Stochastic Gradient Descent and with K-Folds = 10 had the most similar results to the built in Scikit Logistic Regression. There is a time drawback for doing higher k-folds, so if getting more accurate results by 3% is important, then it would take longer to run than Logistic Regression without k-folds.

The built-in SVC function from the Scikit Learn library had higher percentages for accuracy, precision, recall, and f-measure compared to the support vector machine algorithm. The accuracy, precision, and F-measure was better by approximately 4% for the built-in SVC function compared to the SVM algorithm and the recall was approximately 6% higher.

Lastly, Random Forest wasn't quite the success story we were hoping to have. With Sci-Kit Learn having over 20% in nearly every metric, it is clear we need work. Some of the improvements include the binarization of continuous data for complexity purposes, as well as the differentiation of continuous data from discrete data in the ID3 algorithm for better metrics. While Random Forest didn't yield the results we wanted, it was useful in helping us with determining the level of contribution of each feature. Figure 4 shows us the importance of the different features:

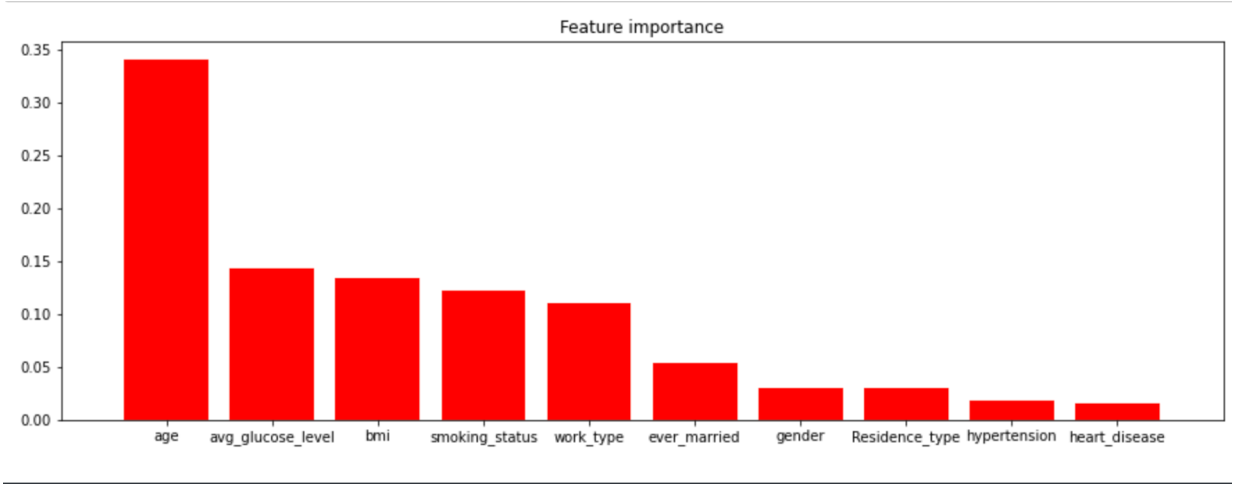


Figure 4: Feature Importance

6.2 Future Work

For future work, we would like to explore the importance of various features to help filter features that are irrelevant or redundant. This would help improve the models which are used to predict whether a patient is likely to have a stroke. In addition, we would like to experiment with finding the optimal learning rate for the stochastic gradient algorithm. Small learning rates tend to converge too slowly, which makes our model too inefficient,

whereas large learning rates causes our model to overshoot the minima and diverge. Finding the optimal learning rate will help in lowering the number of iterations in order to find a model with high accuracy.

Some extra conclusions we determined were that, apart from a lot of categories that do not make sense for this context, some of the metrics are flawed. BMI in general is not a good metric to use, so it would be better to measure weight [1]. Also, we are unsure if smoking is related to tobacco, cannabis, crack cocaine, or any other drug that can be inflammed and inhaled. Nonetheless, we see that age is clearly the biggest contributor to one's likelihood to suffer a stroke by far.

References

[1] Brock Armstrong. Is Bmi an Accurate Way to Measure Body Fat? Technical report, June 2019. URL: https://www.scientificamerican.com/article/is-bmi-an-accurate-way-to-measure-body-fat/.

[2] fedesoriano. Stroke Prediction Dataset. Technical report, Februrary 2021. URL: https://www.kaggle.com/fedesoriano/stroke-prediction-dataset.

[3] Aditya Khosla, Yu Cao, Cliff C. Lin, Hsu-Kuang Chiu, Jungling Hu, and Honglak Lee. An Integrated Machine Learning Approach to Stroke Prediction. Technical report, July 2010. URL: https://people.csail.mit.edu/khosla/papers/kdd2010.pdf.

[4] Andrew Ng and Tengyu Ma. CS229 Notes 1. pages 14–17, Fall 2020. URL: http://cs229.stanford.edu/notes2020fall/notes2020fall/cs229-notes1.pdf.

[5] Andrew Ng and Tengyu Ma. CS229 Notes 3. pages 11–17, Fall 2020. URL: http://cs229.stanford.edu/notes2020fall/notes2020fall/cs229-notes3.pdf.

[6] Poonam K. Singh. World Stroke Day. October 2019. URL: https://www.who.int/southeastasia/news/speeches/detail/world-stroke-day-2019.

[7] Tony Yiu. Understanding Random Forest. Technical report, June 2019. URL: https://towardsdatascience.com/understanding-random-forest-58381e0602d2.