Comparative Analysis of Decision Tree, SVM, KNN, and Random Forest Machine Learning Algorithms against a Keras Neural Network in Predicting Stroke

Machine Learning Spring 2023 Final Report

Brandon Thong and Krista Barone

Table of Contents

Abstract …………..………. 1
Introduction……………... 2-3
Results……………………… 4-8
Discussion…………….….. 9-10
References………….….… 11
Appendix………………….. 11

Research Question: Can machine learning algorithms reliably predict the likelihood that a patient will have a stroke based on known risk factors?

Hypothesis: A neural network will predict the likelihood that a patient suffers a stroke more accurately than decision tree, random forest, K-nearest neighbors, or support vector machine models.

Abstract

The advent of machine learning has enabled data scientists to analyze large, complex datasets with enhanced capability to identify patterns with higher levels of accuracy, increased scalability, and greater freedom to tailor models than traditional statistical methods. Though machine learning algorithms such as decision trees, random forests, support vector machines (SVM), and K-nearest neighbors (KNN) are capable of performing classification, regression, and prediction tasks, neural networks offer improvements over these algorithms with their ability to model data that are nonlinear and increasingly complex. Here, we compare the ability of four machine learning models to a Keras neural network to predict stroke outcomes (stroke/no stroke) across a cohort of 5110 patients. SVM performed the most poorly by a relatively large margin, followed by the Keras neural network and KNN models. The decision tree model performed the next best and the random forest model performed exceedingly well with an F1 score of 0.99. The SVM model's poor performance highlights the limitations that the imbalance in the stroke dataset posed, even after oversampling the minority class data points. In future work, increasing the size of the data set, utilizing alternate under/oversampling techniques, class-weighing, feature selection, and application of L1 and/or L2 regularization techniques may further improve the performance of all of these models.

Introduction

Machine learning is a subset of artificial intelligence that utilizes algorithms and datasets with the intent of mimicking human learning and improving accuracy gradually over multiple iterations without explicit programming instructions (1,2). Machine learning is quickly becoming of paramount interest in the field of data science where well-established statistical methods and algorithms can be harnessed to improve the capabilities of machine learning. Additionally, machine learning can leverage information procured from large datasets to identify patterns and make predictions about specific outcomes as a result of the patterns observed. Due to its ability to conduct predictive analyses on large datasets, machine learning is becoming ever more popular in biological sciences to extract insights in fields such as drug discovery, genomics, and medical informatics.

Machine learning methodologies are typically divided into four categories: supervised, unsupervised, semi-supervised, and reinforcement learning. In supervised learning, data are trained on data where both the input and corresponding output are labeled. The model then analyzes the relationship between the input and output, receiving feedback on correctness and learning to make predictions on new data based on given features. Unsupervised learning requires the model to train on a set of input features without providing corresponding output features. The model must then predict outputs based on recognizing patterns in data by relying on clustering similar examples together (3). In semi-supervised learning, the algorithm is trained on both labeled and unlabeled data, where some inputs are given with corresponding outputs and others are not. Unlabeled data helps the model to better generalize new data, while labeled data provides feedback on the correctness of the output. In reinforcement learning, an agent learns to interact with its environment by taking actions and receiving rewards or penalties based on its actions (3).

Decision trees, random forests, support vector machines and neural networks are machine learning models that operate under supervised learning algorithms. Decision trees are tree-like structures that consisting of root nodes, the node which begins the tree, decision nodes, which are nodes made after a decision splits the root node, and leaf nodes, which are terminal nodes where a decision is made after the tree can no longer be split further. In machine learning, decision trees are used for both classification and regression tasks and can be used for predictive tasks by traversing the tree based on the values of the input features and selecting the corresponding output value at the leaf node. A random forest is a supervised machine learning algorithm that consists of multiple decision trees, reducing overfitting by creating a forest that uses bagging and randomness when constructing trees (3,4).

Support vector machines (SVM) are a supervised deep learning model that relies on mapping data to a feature space even in cases where data are not linearly separable. These models can be used to perform classification and regression tasks. K-nearest neighbor (KNN) models are supervised learning algorithms that classify data based on their similarity to neighbors (4). Neural networks, such as Keras, are modeled after neurons in the brain, where systems of interconnected nodes transmit data through a series of layers to generate an output. The Keras neural network is particularly popular because it runs atop established machine learning frameworks like TensorFlow (5). Each of these models can perform regression, classification, and prediction tasks that can be useful in a multitude of scenarios, including the prediction of the likelihood that an event, such as illness, will occur.

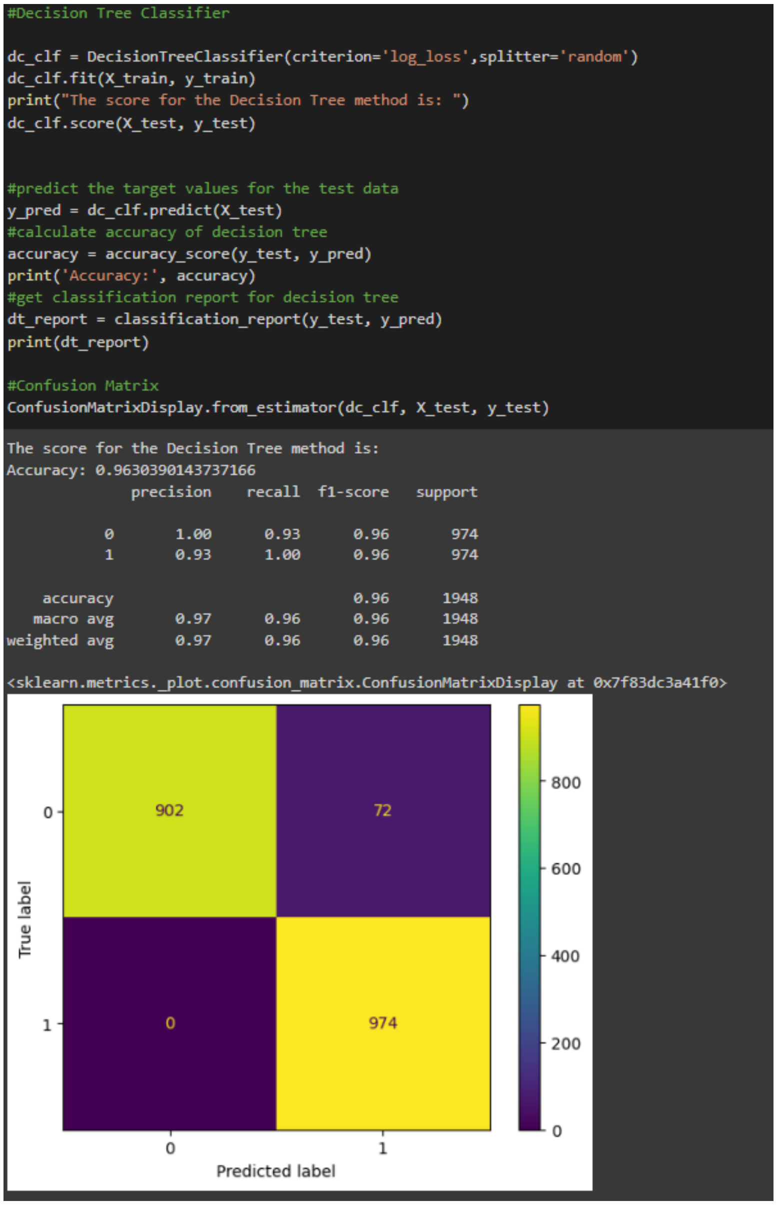
Stroke, or cerebrovascular accident (CVA), is a leading cause of serious long-term disability. Nearly 15 million people worldwide suffer from stroke annually. Each year, approximately 795,000 individuals experience stroke in the United States alone (6). Stroke is associated with a series of well-documented risk factors including age, gender, race, and ethnicity, and smoking status. Additionally, concurrent health conditions such as high blood pressure, high cholesterol, diabetes, and obesity are known to contribute to the likelihood that an individual will suffer a stroke (7).

In this project, we aim to determine whether it is possible to predict stroke outcomes using supervised learning algorithms. If successful, this may allow us to use our models on new patients to make helpful risk predictions for their future health. To address this aim, we will compare four supervised machine learning models, decision tree, random forest, support vector machine, and K nearest neighbors to a Keras neural network to predict the likelihood of stroke in a dataset consisting of 5110 patients with 11 attributes. These attributes include age, gender, hypertension, heart disease, marital status, work type, residence type, average blood glucose level, and body mass index (BMI). We hypothesize that the Keras neural network will predict the likelihood of stroke with greater accuracy than other models due to the neural network's ability to interpret more complex patterns with greater efficacy.

Results:

A decision tree model predicted the likelihood of stroke with an accuracy score of 0.9630390143737166 and f1 score of 0.96. Hyperparameter tuning indicated optimal parameters were criterion='log_loss' and splitter='random'.

Classification report and confusion matrix shown below.

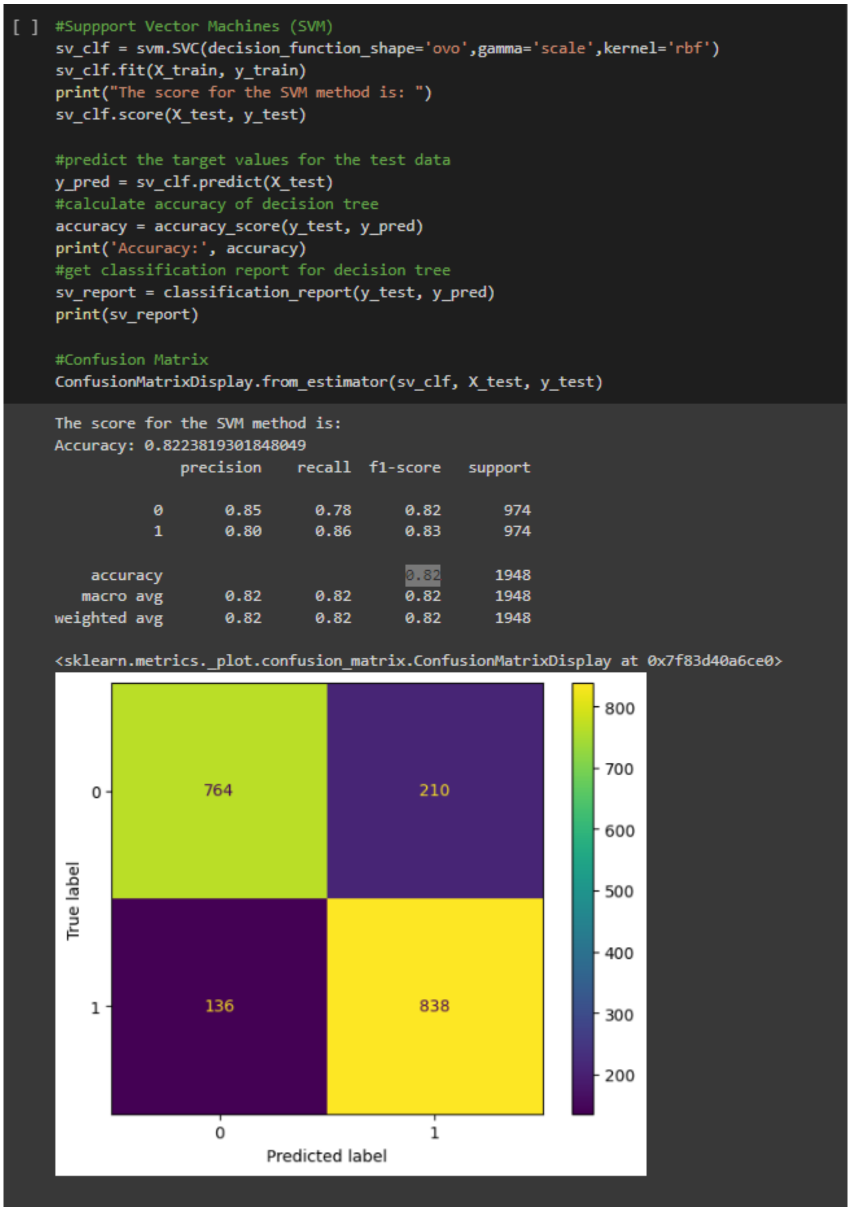


```
#Decision Tree Classifier

dc_clf = DecisionTreeClassifier(criterion='log_loss',splitter='random')
dc_clf.fit(X_train, y_train)
print("The score for the Decision Tree method is: ")
dc_clf.score(X_test, y_test)


#predict the target values for the test data
y_pred = dc_clf.predict(X_test)
#calculate accuracy of decision tree
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
#get classification report for decision tree
dt_report = classification_report(y_test, y_pred)
print(dt_report)

#Confusion Matrix
ConfusionMatrixDisplay.from_estimator(dc_clf, X_test, y_test)
```

```
The score for the Decision Tree method is:
Accuracy: 0.9630390143737166
              precision    recall  f1-score   support

           0       1.00      0.93      0.96       974
           1       0.93      1.00      0.96       974

    accuracy                           0.96      1948
   macro avg       0.97      0.96      0.96      1948
weighted avg       0.97      0.96      0.96      1948

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f83dc3a41f0>
```

A support vector machine predicted the likelihood of stroke with an accuracy of 0.8223819301848049 with an f1 score of 0.82. Hyperparameter tuning showed two results with the same rank test score, so the first one was chosen in the table. The parameters chosen were decision_function_shape='ovo', gamma='scale, and kernel='rbf'

Classification report and confusion matrix shown below.

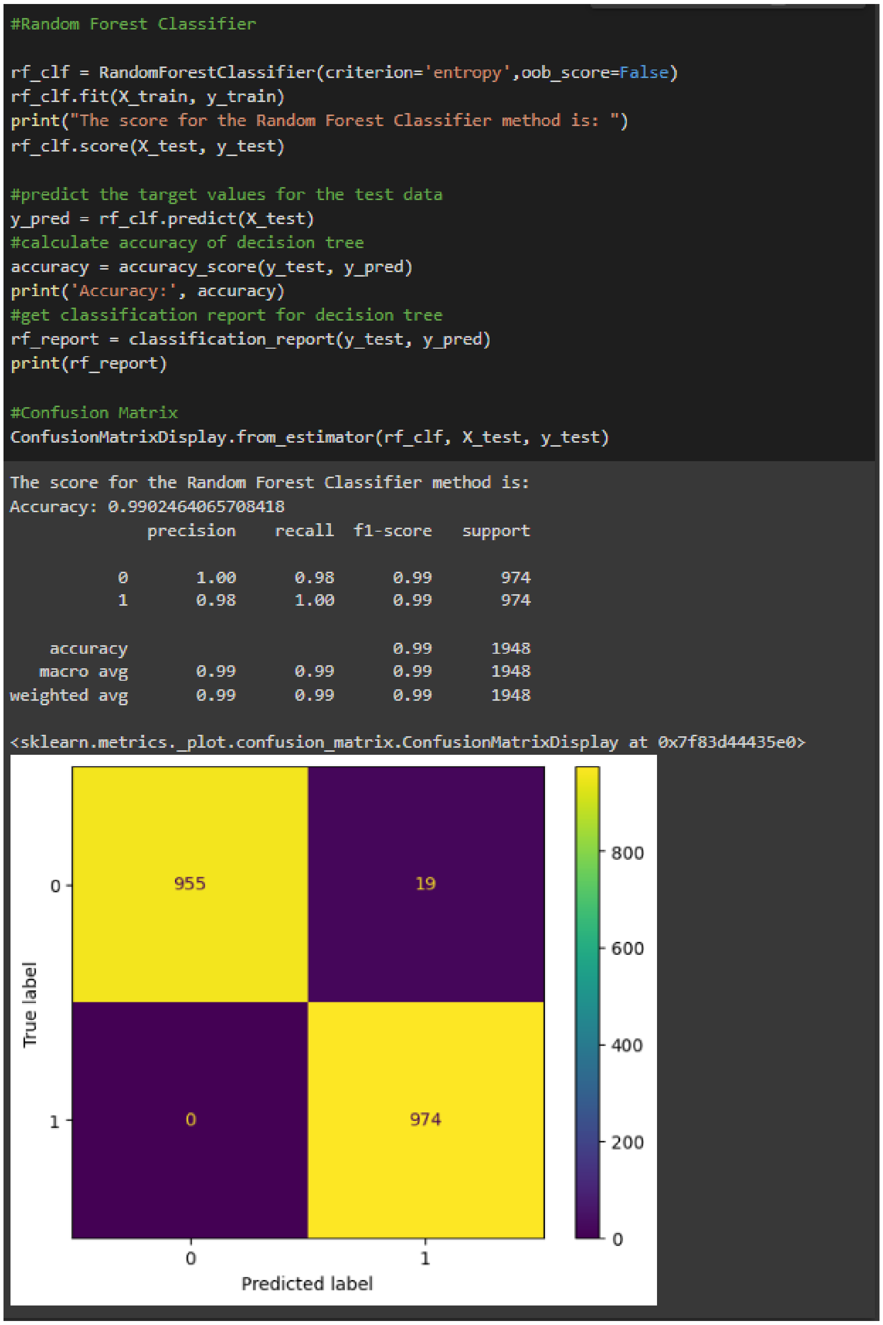


```
[ ] #Suppport Vector Machines (SVM)
    sv_clf = svm.SVC(decision_function_shape='ovo',gamma='scale',kernel='rbf')
    sv_clf.fit(X_train, y_train)
    print("The score for the SVM method is: ")
    sv_clf.score(X_test, y_test)

    #predict the target values for the test data
    y_pred = sv_clf.predict(X_test)
    #calculate accuracy of decision tree
    accuracy = accuracy_score(y_test, y_pred)
    print('Accuracy:', accuracy)
    #get classification report for decision tree
    sv_report = classification_report(y_test, y_pred)
    print(sv_report)

    #Confusion Matrix
    ConfusionMatrixDisplay.from_estimator(sv_clf, X_test, y_test)
```

```
The score for the SVM method is: 
Accuracy: 0.8223819301848049
              precision    recall  f1-score   support

           0       0.85      0.78      0.82       974
           1       0.80      0.86      0.83       974

    accuracy                           0.82      1948
   macro avg       0.82      0.82      0.82      1948
weighted avg       0.82      0.82      0.82      1948

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f83d40a6ce0>
```

A random forest computed the likelihood of stroke with an accuracy score of 0.9902464065708418 and f1 score of 0.99. Hyperparameter tuning indicated optimal parameters were criterion='entropy' and oob_score=False.

Classification report and confusion matrix shown below.

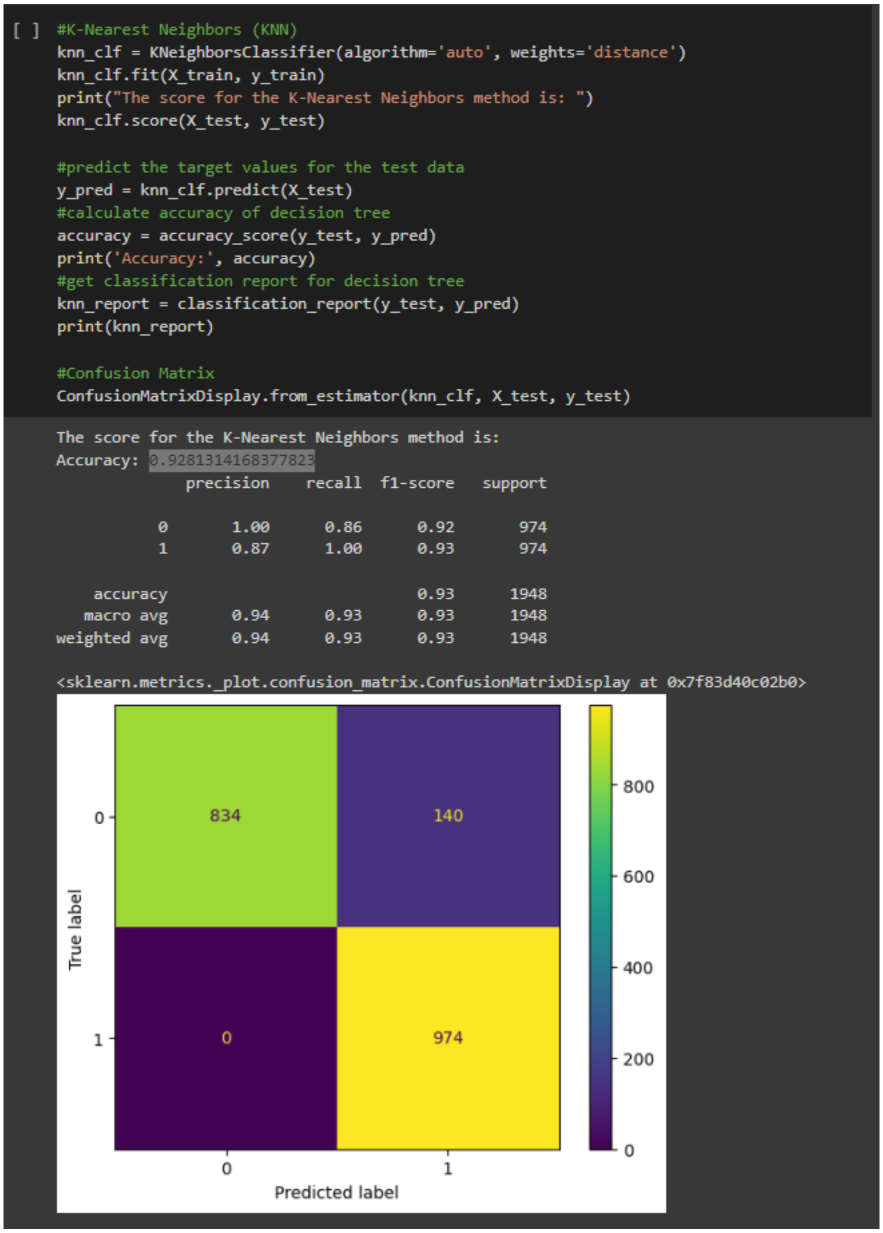


```
#Random Forest Classifier

rf_clf = RandomForestClassifier(criterion='entropy',oob_score=False)
rf_clf.fit(X_train, y_train)
print("The score for the Random Forest Classifier method is: ")
rf_clf.score(X_test, y_test)

#predict the target values for the test data
y_pred = rf_clf.predict(X_test)
#calculate accuracy of decision tree
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
#get classification report for decision tree
rf_report = classification_report(y_test, y_pred)
print(rf_report)

#Confusion Matrix
ConfusionMatrixDisplay.from_estimator(rf_clf, X_test, y_test)
```

```
The score for the Random Forest Classifier method is:
Accuracy: 0.9902464065708418
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       974
           1       0.98      1.00      0.99       974

    accuracy                           0.99      1948
   macro avg       0.99      0.99      0.99      1948
weighted avg       0.99      0.99      0.99      1948

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f83d44435e0>
```

A K-nearest neighbors model computed the likelihood of stroke with an accuracy score of 0.9281314168377823 with an f1 score of 0.93. Hyperparameter tuning provided several settings with the same rank test score, so we chose the first one. Optimal tuning settings were algorithm=’auto’ and weights=’distance’.

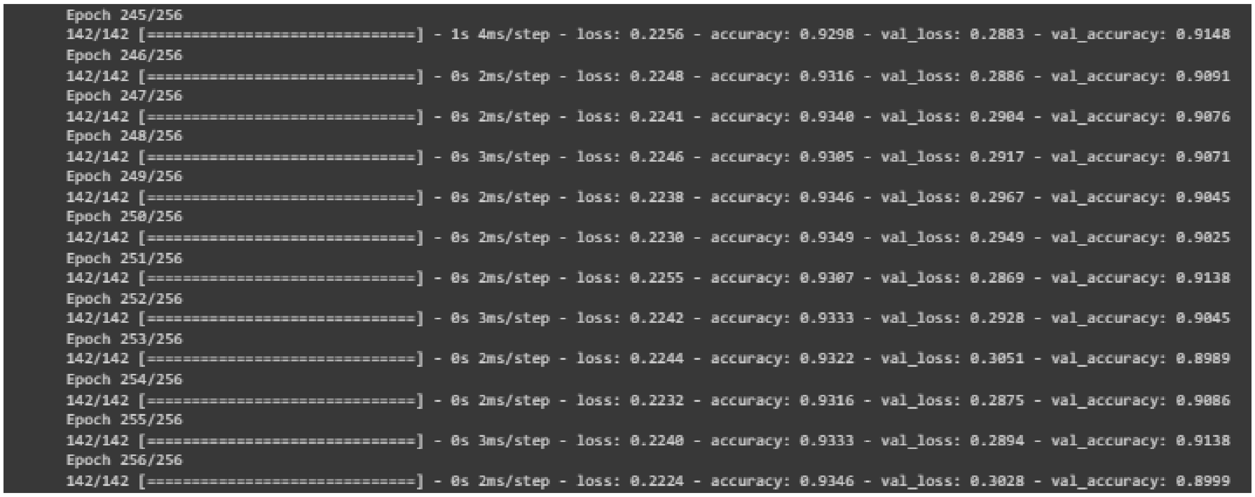
Classification report and confusion matrix shown below.



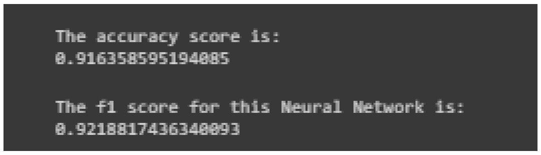
```
[ ]  #K-Nearest Neighbors (KNN)
     knn_clf = KNeighborsClassifier(algorithm='auto', weights='distance')
     knn_clf.fit(X_train, y_train)
     print("The score for the K-Nearest Neighbors method is: ")
     knn_clf.score(X_test, y_test)

     #predict the target values for the test data
     y_pred = knn_clf.predict(X_test)
     #calculate accuracy of decision tree
     accuracy = accuracy_score(y_test, y_pred)
     print('Accuracy:', accuracy)
     #get classification report for decision tree
     knn_report = classification_report(y_test, y_pred)
     print(knn_report)

     #Confusion Matrix
     ConfusionMatrixDisplay.from_estimator(knn_clf, X_test, y_test)
```

```
The score for the K-Nearest Neighbors method is: 
Accuracy: 0.9281314168377823
              precision    recall  f1-score   support

           0       1.00      0.86      0.92       974
           1       0.87      1.00      0.93       974

    accuracy                           0.93      1948
   macro avg       0.94      0.93      0.93      1948
weighted avg       0.94      0.93      0.93      1948

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f83d40c02b0>
```

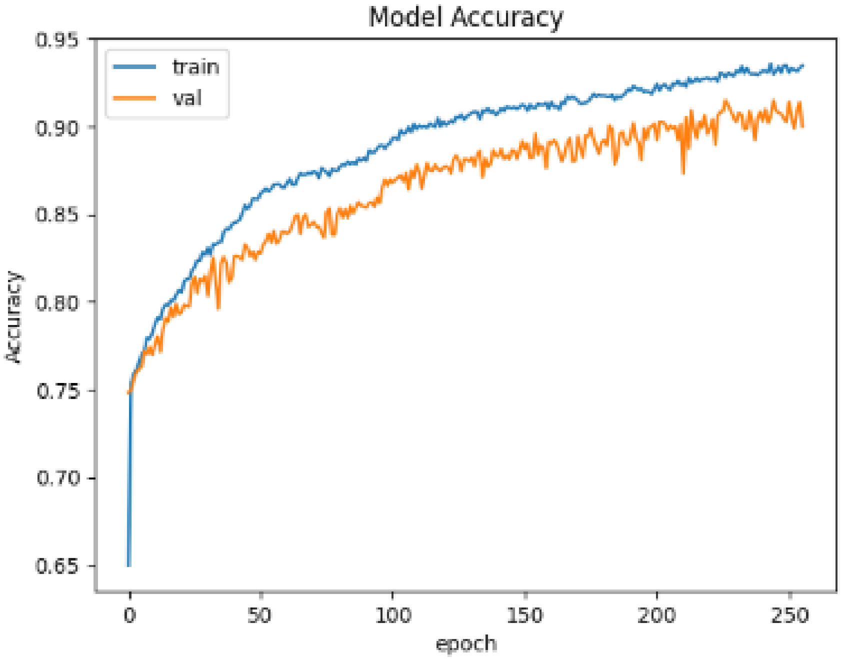
A Keras Neural Network model computed the likelihood of stroke with an accuracy score of 0.916358595194085 and an F1 score of 0.9218817436340093. A graph of the model accuracy on the training and testing data is below, displaying an improvement on prevention of overfitting that was seen during the presentation. Furthermore, the losses and accuracies for the test and validation data can be seen for each epoch in the neural network that had two hidden layers of 15 neurons each, following general rules of thumb for constructing keras neural networks.



```
Epoch 245/256
142/142 [==============================] - 1s 4ms/step - loss: 0.2256 - accuracy: 0.9298 - val_loss: 0.2883 - val_accuracy: 0.9148
Epoch 246/256
142/142 [==============================] - 0s 2ms/step - loss: 0.2248 - accuracy: 0.9316 - val_loss: 0.2886 - val_accuracy: 0.9091
Epoch 247/256
142/142 [==============================] - 0s 2ms/step - loss: 0.2241 - accuracy: 0.9340 - val_loss: 0.2904 - val_accuracy: 0.9076
Epoch 248/256
142/142 [==============================] - 0s 3ms/step - loss: 0.2246 - accuracy: 0.9305 - val_loss: 0.2917 - val_accuracy: 0.9071
Epoch 249/256
142/142 [==============================] - 0s 2ms/step - loss: 0.2238 - accuracy: 0.9346 - val_loss: 0.2967 - val_accuracy: 0.9045
Epoch 250/256
142/142 [==============================] - 0s 2ms/step - loss: 0.2230 - accuracy: 0.9349 - val_loss: 0.2949 - val_accuracy: 0.9025
Epoch 251/256
142/142 [==============================] - 0s 2ms/step - loss: 0.2255 - accuracy: 0.9307 - val_loss: 0.2869 - val_accuracy: 0.9138
Epoch 252/256
142/142 [==============================] - 0s 3ms/step - loss: 0.2242 - accuracy: 0.9333 - val_loss: 0.2928 - val_accuracy: 0.9045
Epoch 253/256
142/142 [==============================] - 0s 2ms/step - loss: 0.2244 - accuracy: 0.9322 - val_loss: 0.3051 - val_accuracy: 0.8989
Epoch 254/256
142/142 [==============================] - 0s 2ms/step - loss: 0.2232 - accuracy: 0.9316 - val_loss: 0.2875 - val_accuracy: 0.9086
Epoch 255/256
142/142 [==============================] - 0s 3ms/step - loss: 0.2240 - accuracy: 0.9333 - val_loss: 0.2894 - val_accuracy: 0.9138
Epoch 256/256
142/142 [==============================] - 0s 2ms/step - loss: 0.2224 - accuracy: 0.9346 - val_loss: 0.3028 - val_accuracy: 0.8999
```



```
The accuracy score is:
0.916358595194085

The f1 score for this Neural Network is:
0.9218817436340093
```



Discussion:

Neural networks, despite their ability to handle complex data, were not the most effective at predicting stroke outcomes. That being said, Decision trees, SVM, random forests, K-nearest neighbors and Keras neural network models were all efficacious in predicting stroke or no stroke classifications well, but to varying degrees. All of these models initially struggled to predict the minority class (stroke) data category, likely due to the imbalance in the dataset, where approximately only 5 percent of the original dataset was classified as stroke outcomes. However, this issue was addressed by resampling the minority class to match the size of the majority class to allow better model training and obtain much better classification of the minority class. Furthermore, additional processing was performed on the dataset to maximize model performance, such as removing non-predictive data points, utilizing pandas' dummies function to convert the categorical variables into numbers without ordering, normalizing non-categorical values with StandardScaler, and stratifying the "train test split" on the stroke data to ensure equal ratios of the stroke classes in the training and testing data (using a 70/30 split).

Hyperparameter tuning was performed on each of the first four models, with the best performing metrics being selected to run the model and obtain the accuracy and F1 scores shown in the figures above. The neural network was tuned by testing adjusted numbers of neurons in the hidden layers as well as the optimizer type, and by following generally accepted rules of thumb to come to the conclusion that best results came from using layers with 15 neurons, 'binary_crossentropy' for the loss function, and 'rmsprop' for the optimizer over 256 epochs. Additionally, the amount of overfitting has been reduced from the model shown in the presentation, with the model slightly overfitting by ~3% when compared to validation data which can be seen in the 'Model Accuracy' figure shown above.

Based on these scoring metrics, the performance of these models from worst to best (with F1 scores) is as follows: SVM(0.82), Neural Network(0.92), KNN(0.93), Decision Tree(0.96), and finally Random Forest(0.99).

After comparing the results from each of our models, it is clear that SVM performed the worst, around 0.1 lower than the next worst performing model in both F1 and accuracy score. This is likely a result of SVM models' generally known poor performance on imbalanced problems (8), as imbalanced support vector ratios cause data points near decision boundaries of hyperplanes to be classified poorly. The rest of the models performed significantly better, with both the K-nearest neighbors and Neural Network models performing very similarly after hyperparameter tuning and the Random Forest model performing remarkably well.

The limitations of these models can be best seen with the SVM and neural network models, as SVM struggled to perform at the same level as the others, even after hyperparameter tuning, and the neural network model was the model type we were least familiar with at the start of the project. Improved performance of these models may be achieved with further preprocessing to avoid the caveats of SVMs as well as deeper understanding of the impact of neural network layer parameters.

In future work, dataset imbalance could be further addressed by increasing the size of the dataset to introduce more training data and seeing the effects of other oversampling techniques to balance the minority class to that of the majority class. Class-weighting may also be a helpful method of addressing the problematic imbalance issue by increasing the weight of the minority class and lowering that of the majority class. It may also be beneficial to perform feature selection methods to rank importance of features in our predictions and better understand which data points are most influential in our models. Lastly, utilizing regularization models such as Lasso and Ridge regression could further improve the effectiveness of the models.

In conclusion, the Keras neural network did not outperform all of the other machine learning methods, but did perform similarly to several other models and much better than the worst one, SVM. From the performance metrics, although the keras neural network did not predict stroke the best, it did perform reliably and competently when compared to the other models. Most notably, our SVM model performed the worst by a fairly large margin, and our Random Forest model performed exceedingly well. Future work will primarily focus on increasing the size of the data set, testing other under/oversampling techniques, class-weighting, feature selection, and applying regularization techniques to further maximize the performance of the models discussed.

References:

Data Source: https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset

1. Alpaydin, E. (2014). *Introduction to Machine Learning*. Cambridge, MA: MIT Press. ISBN: 978-0-262-02818-9
2. Raschka, S. (2015). *Python Machine Learning*. Packt Publishing - ebooks Account. ISBN: 1783555130
3. https://ischoolonline.berkeley.edu/blog/what-is-machine-learning/
4. https://www.ibm.com/topics/machine-learning
5. https://keras.io/
6. Tsao CW, Aday AW, Almarzooq ZI, Alonso A, Beaton AZ, Bittencourt MS, et al. Heart Disease and Stroke Statistics—2022 Update: A Report From the American Heart Association. Circulation. 2022;145(8):e153–e639.
7. National Institute of Neurological Disorders and Stroke. (2012). Stroke: Hope through research. Retrieved August 7, 2012, from https://www.ninds.nih.gov/Disorders/All-Disorders/Stroke-Information-Page

8. Lemnaru, C., Potolea, R. (2012). Imbalanced Classification Problems: Systematic Study, Issues and Best Practices.

Appendix:

- The code used to execute this project can be viewed at the following Google Colab link as well as a pdf attached to this submission named “ML_FINAL_GOOGLECOLAB”

- https://colab.research.google.com/drive/1AnZ7bd6qkqVsiqOxIFBmYD5otLNuRN1W?usp=sharing