Disclaimer: Some links have been removed, this is a copy of the file saved for the University of Wisconsin - Madison Informatic Skunkworks group as a part of their records. Please keep in mind that this was an end-of-semester summary written by me with no intention of this being used for anything, especially publication.

Renal AMLs Project - Spring 2022

[Link removed] : Overall folder with all presentations, code, and data

Quick Summary

We tried a lot of different approaches using various models for classification and regression, but were unable to find a model that gave us compelling results. Classification did not consistently do better than naive classifiers (this is very bad) and regression was also unimpressive.

Dataset

Data Processing Notebook (link removed)
Data Visualization Notebook: This was used to create figures and get statistics for the different datasets. (link removed)
Data - Google Drive Folder (link removed)

Overview

The data for this project consisted of patient, clinical, and extracted data. The raw dataset contained 653 rows and 706 columns. The rows contained information from 368 AMLs, 287 of which had two scans and 79 with only a single scan.

Data Preprocessing

[Dataset Preprocessing Sheet - detailed explanation of how each column was handled] (link removed)

Each feature in the dataset was analyzed to determine whether it should be kept, removed, or one-hot encoded. Features were removed if they were not medically relevant, too many patients did not have information for that feature, the data could not be used by a simple machine learning model (e.g. strings containing clinical notes), or the feature contained information that was duplicated in another column. Additionally, the features used to generate the column that was to be predicted by the model were removed to prevent data leakage. Categorical data represented by a single feature were one-hot encoded, i.e. split into multiple features such that the models would not infer an order from the categorical feature. After processing the dataset, 399 of the original 706 features remain.

Additionally, the rows were analyzed. If a row was deemed to be missing too much data, it was removed from the dataset. To calculate the change between scans for a tumor, a first scan and a last scan are needed. Therefore, any AML that only had one scan, was removed.

Calculating the value to be predicted by the value (Y)

The y value to be predicted was calculated as the difference between “1D longest dimension (cm) last imaging” and “1D longest dimension (cm) 1st imaging” divided by “date last imaging- diagnosis dx”. This “rate”, referred to as “Unidimensional Rate” in the code, was then used for regression analysis. A column featuring the change, without accounting for the time between the scans (ie, the previous formula without dividing by time), was generated for regression analysis as well. This column was referred to as “Unidimensional Change” in the code.

These previous two columns were then used to analyze the AMLs as a classification problem. Our motivation for trying a classification approach was that even if we could not create a model to successfully predict the growth of a tumor, we may be able to create a model that could distinguish between “no growth” or “little growth” vs “some amount of growth” or “high growth. For both the “Unidimensional Rate” and “Unidimensional Change”, thresholds were set to convert the numerical values to either a 0 or 1. The “Unidimensional Rate” was converted to a binary feature with thresholds of .1/year and .2/year. For example, with the .1/year threshold, an AML with a rate of change less than .1/year would be converted to zero, while an AML with a rate of change greater than .1/year would be converted to one. “Unidimensional Change” was converted to a binary feature with a threshold of .5.

Thresholds were also used in regression analysis to try to predict the growth of tumors over a certain threshold. The motivation for this being that tumors with a small change in size may simply have different measurements due to external factors, such as human error, and not actual growth in the tumor. Therefore, we thought it may be valuable to see if a model would be able to predict growth in tumors that were more certain to have actually grown. We performed regression analysis for “Unidimensional Change” above .5 and “Unidimensional rate”above .2/yr and .1/yr. As a result of these thresholds, the amount of data available for analysis decreased, which likely had a negative effect on model performance.

Methods

The following methods were used for each of the previously mentioned datasets that were generated. A neural network and XGBoost were used for both classification and regression. GPR was used exclusively for regression. SVM was used exclusively for classification. The naive classifiers were used as a baseline for comparison of classification models.

Neural Network

Neural Network Jupyter Notebook (link removed)

The neural network was created using the PyTorch framework. The model was constructed using the sequential container with four linear layers each followed by a ReLU activation excluding the final linear layer leading to the output. The data was loaded from the csv file using pandas before being converted into a tensor to be used by the model.

For regression, the model was trained over 32 epochs with the Adam optimizer with a loss rate of 1e-4 and MSELoss, or Mean Squared Error Loss, as the loss criterion. After training, the model was used for prediction with Root Mean Squared Error as the error metric collected. In addition, a parity plot was created with the predicted value for y and Ground Truth as x.

For classification, the model was trained over 16 epochs with the Adam optimizer with a loss rate of 1e-4 and CrossEntropyLoss as the loss criterion. After training, the model was used for prediction with Accuracy, F1 score, and Geometric Mean score metrics collected. Additionally, a confusion matrix is generated using Scikit learn's confusion matrix function.

Gaussian Process Regressor (GPR)

GPR Model Notebook (link removed)

A GPR model was used on the regression datasets. The model was created using the Scikit Learn framework with Scikit learn's radial basis function as the kernel. The following parameters were used when creating the model: n_restarts_optimizer=10, alpha=0.8, and normalize_y=True. The metrics were averaged across 100 train/test splits using Scikit learn's train_test_split function with a random seed corresponding to the iteration of the split from 0 to 100 exclusive. From this model, the metrics Mean Squared Error and R^2 were calculated. Additionally, a parity plot was created with the predicted value for y and Ground Truth as x.

XGBoost

XGBoost Model Notebook (link removed)

The XGBoost model was used for classification and regression. The model was constructed using the XGBoost python package. The XGBoostClassifier and XGBoostRegressor were used for classification and regression respectively. The metrics were averaged across 100 train/test splits using Scikit learn's train_test_split function with a random seed corresponding to the iteration of the split from 0 to 100 exclusive. For the XGBoostClassifier, F1 score, accuray, and Geometric Mean were calculated. For the XGBoostRegressor, the metrics Mean Squared Error and R^2 were calculated. Scikit learn's GridSearchCV function was used to tune the parameters of the model.

For regression, the parameters tuned were "max_depth", "eta", "n_estimators", and "colsample_bytree". The values for the given parameters were the best parameters found using Scikit Learn's GridSearchCV function. For the rate of change regression dataset, these parameters were set to: max_depth = 4, colsample_bytree=0.6, eta=0.01, and n_estimators=50. For the absolute change regression dataset, these parameters were set to: max_depth = 10, colsample_bytree=0.1, eta=0.01, and n_estimators=100. Additionally, a parity plot was created with the predicted value for y and Ground Truth as x.

For classificaion, the parameters tuned were “max_depth”, “eta”, “n_estimators”, and “colsample_bytree”. For the rate of change classification dataset, these parameters were set to: max_depth = 7, colsample_bytree=0.5, eta=0.01, and n_estimators=50. For the absolute change classification dataset, these parameters were set to: max_depth = 8, colsample_bytree=0.0.6, eta=0.01, and n_estimators=100. Additionally, a confusion matrix is generated using Scikit learn’s confusion matrix function.

Support Vector Machine (SVM) - Support Vector Classification (SVC)

SVM Model Notebook (link removed)

The Support Vector Classifier was used on the classification datasets. The model was created using Scikit Learn’s SVC framework with the ‘kernel’ parameter set to ‘linear’. The metrics were averaged across 100 train/test splits using Scikit learn’s train_test_split function with a random seed corresponding to the iteration of the split from 0 to 100 exclusive. From this model, the metrics Accuracy, F1 score, and Geometric mean were calculated. Additionally, confusion matrices were generated. A histogram of the permutation scores, with lines for the classification p-score and luck, was created using Scikit learn’s permutation_test_score function.

Naive Classifiers

Naive Classifier Jupyter Notebook (link removed)

Naive Classifiers are classifiers which simply adhere to a guessing strategy, and do not assess the features at a given point, when predicting. These models are a good baseline to compare to more sophisticated models, which have been trained on the training set. We use a majority classifier, minority classifier, random guesser, and stratified guesser for our benchmark Naive Classifiers.

The majority classifier predicts the majority (most common) class for every data point. Conversely, the minority classifier predicts the minority (least common) class for every data point. The random guesser chooses a random class for each prediction with each class having an equal probability of being selected. The stratified guesser randomly selects a class from the dataset with class probabilities determined by the frequency of the given class in the dataset.

The native classifiers are implemented using Scikit-learn's DummyClassifier. The majority and stratified classifiers utilize the “strategy” parameter with values "most_frequent" and "stratified" respectively. For the minority classifier, the DummyClassifier's strategy is set to "constant" with the value set to the least common class. The least common class is found using the least common y value found with pandas’ “value_counts” and “argmin” functions. For the random guesser, the DummyClassifier’s strategy is set to “uniform”.

Analysis

Through all of the analysis, we failed to find a way to process the data such that a model found meaningful results. Our results showed that we failed to find a promising approach to

predicting the growth of the Renal AMLs. However, as with all Machine Learning tasks, this does not mean that other techniques or obtaining more data may not result in better results.

For regression, the models consistently showed poor performance. When performing the regression only on data points above the thresholds, the amount of available data dropped, making performance even worse.

For classification, the models failed to consistently perform better than the naive classifiers for any of the datasets when considering all of our metrics. This means that the trained models performed worse than simply guessing.

Tables and Figures

Tables taken from the following presentations. Confusion Matrices and permutation test histograms can also be found in the following presentation
2022-04-08 Meeting
2022-04-15 Meeting
2022-04-22 Meeting → most up to date results, may have been some dataset tweaks between 4-15 and 4-29

Classification

Change

We had not created the naive classifiers at the time this data was gathered, but this is still poor performance.

Model Name	F1 Score	Accuracy	Gmean
Neural Network	0.449	0.325	0.256
SVM	0.444	0.75	0.556
XGBoost	0.285	0.75	0.408

```
Total number of data points: 196
Number in negative class: 152
Number in positive class: 44
```

Rate - .1/yr

Model Name	F1 Score	Accuracy	Gmean
SVM	0.355	0.682	0.539
XGBoost	0.305	0.751	0.452

Naive Classifier	F1 Score	Accuracy	Gmean
Majority	0.0	0.766	0.0
Minority	0.375	0.234	0.0
Random	0.296	0.486	0.475
Stratified	0.223	0.623	0.394

```
Total number of data points: 196
Number in negative class: 149
Number in positive class: 47
```

Rate .2/yr

Model Name	F1 Score	Accuracy	Gmean
SVM	0.261	0.822	0.457
XGBoost	0.223	0.873	0.328

Naive Classifier	F1 Score	Accuracy	Gmean
Majority	0.0	0.88625	0.01
Minority	0.201	0.114	0.0
Random	0.191	0.511	0.482

Stratified	0.119	0.791	0.242

```
Total number of data points: 196
Number in negative class: 173
Number in positive class: 23
```

Regression

All Data (No Thresholds)

I can't find a single table with all of the information in it, but looking at 2022-04-08 meeting notes performance looks poor if we look at the parity plots. The x being what the values should be while the y is what is being predicted. A perfect model would have all of the points on the red line, where x = y.

Rate - .1/yr

Model Name	r^2	MSE
GPR	-1.248	0.331
XGBoost	-1.249	0.370

```
Total number of data points: 47
Minimum: 0.1053 Maximum: 4.6429
Standard Deviation: 0.6495
```

Rate - .2/yr

Model Name	r^2	MSE
GPR	-16.599	0.882
XGBoost	-247.616	2.022

```
Total number of data points: 23
Minimum: 0.2341 Maximum: 4.6429
```

Standard Deviation: 0.8827