Project 2 Report

By Steven Oh

Preparing the Data

To properly prepare the data, it was necessary to visualize the data to fully understand what it contained and to set up a plan of action. The first step taken to visualize the data was checking the data's shape and size, along with getting the info for each of the columns. Getting this information is crucial to properly prepare the data as it shows any columns with null values and the datatype for each column. In this case, there were no null values (at least for now), and most columns (besides the `"deg-malig"`) were object types. This meant that most of the dataset needed to be converted to be able to fit a model.

After viewing the data itself, the `data['{column title}'].unique()` was used to understand the data further. This function was helpful as it identified that the missing values were set to `"?"` rather than `"null"`. For the `"node-caps"` column, I decided it was best to tie it with the `"inv-nodes"` column as they both contained information about nodes. To replace the missing data, a median approach was used. This was done with a for-loop that would iterate through the data and replace the `"?"` for each `"inv-nodes"` with `"yes"` or `"no"`. It would replace it with a `"yes"` if there were more `"yes"` values than `"no"` and vice versa, else, it would set it to `"no"` as that seemed like the safer option for this dataset. Since there was only one missing value for the `"breast-quad"` column, I manually replaced it with whatever value was most popular (`"left_low"` in this case).

Finally, the last step to prepare the data was to convert all the object datatypes to another type that is compatible with the models. All of the columns except `"tumor-size"`, `"inv-nodes"`, and `"age"`, were converted to categorical types using one-hot encoding. I decided to specifically drop the first for the `"class"`, `"node-caps"`, `"breast"`, and the `"irradiat"` columns as they only contained 2 unique values. This was using the thought process that it would allow for these columns to still be related to each other. Lastly, for the other columns, it was observed that the unique values were ranges. Therefore, I replaced the values and set them to the beginning of the range. This would allow these columns to continue to stay related to each other, increasing the accuracy of the models.

Training the Model

The 3 techniques that I used to train the model were K-Nearest Neighbor Classifier, Decision Trees, and Random Forest Classifier. The dependent variable was set to the `"class_recurrence-events"` as the goal is to accurately predict how many patients will have a recurrence of the disease. The independent variables were the rest of the data as all the columns were related or correlated well with the dependent variable.

Results

Both the K-Nearest Neighbor and Decision Tree performed fairly decently, while the Random Forest Classifier did the worst (at least in terms of accuracy). I will discuss accuracy first and then dive deeper into the classification report in the summary.

For the K-Nearest Neighbor model, the accuracy of the test was 62% and the accuracy of the train was 81%. Then `GridSearchCV` was used to improve the model, trying to find which `n_neighbor` parameter would be most optimal. 34 was found to be the best, increasing the accuracy of the test to 71%, but lowering the accuracy of the train to 69%. This does show, however, that 1st iteration of the model may have been a little overfit. I did additional testing, setting an extra parameter for the `GridSearchCV` to focus on the recall scoring, which gave an accuracy score of 69% for the test and 98% for the train (this is with 1 `n_neighbor`). This shows signs of overfitting.

For the Decision Tree model, the accuracy was 66% for the test and 98% for the train. This model showed signs of overfitting, making it a less likely contender for the recommended model for this dataset.

For the Random Forest model, `GridSearchCV` was used from the start of the model building, creating an accuracy score of 37% for the test and 40% for the train. If we were to decide on the model based on the accuracy score, this model would also be taken from the contender list.

Summary

As mentioned above, if the model selection was based on the average accuracy report, I would recommend using the K-Nearest Neighbor model for this dataset with 34 as the `n_neighbors`. However, as we can consider the difference between the recall and precision scores, I would probably recommend the Random Forest model. We should focus on recall as it is better to focus on decreasing false negatives, meaning that we would rather be wrong in terms of falsely predicting a recurrence than saying that there is not a recurrence when there was.

Ultimately, this specific model performs fantastically with recall, with a score of 96% for the recall on saying `"True"` (that there is a recurrence) for the test and a score of 100% for the train. For precision, however, the score is 31% on saying `"True"` for the test and 33% for the train. This is the reason why the average accuracy score plummets lower than most of the other models.

The most important standard model performance metric to optimize for this model is `n_estimators` as this determines how many “trees” to build for the model. Lastly, using the `scoring="recall"` parameter for the GridSearchCV is crucial to focus on the parameter that matters most in a certain dataset.

Use of ChatGPT

ChatGPT was only used to find the `value_counts()` functionality in the pandas library.