Arabel Rachman

Project 1 Report

Data Preparation

The first step of data preparation is understanding the data provided for the project. In order to do this, I gathered information about the overall shape and contents of the data. Using functions provided by the Pandas library, I was able to see that the data consists of 10 different variables and there are 386 patient data provided. Out of the 10 variables provided, the class variable will be the dependent variable being predicted using the other 9 dependent variables. Following this, I needed to clean the dataset by changing any variable data types, checking for any duplicate data, and filling in any missing data. By looking at the data types for all the variables, I found that all of them were initially set as objects except for one int64 data. In order to perform machine learning computations, I changed all of the object data types into categorical data, except for one that only consisted of “true” or “false” values, for which I changed into an int64 type, where a “true” is a 1 and a “false” is a 0. I also cleaned up any missing and duplicate data by deleting duplicates using the drop_duplicates() function and filling in any missing values by using the mode value for their respective age groups. Lastly, one-hot encoding is performed on the categorical data to make machine learning analysis easier.

Insights From Data Preparation

In order to gain more insights from the data, I visualized them using matplotlib and seaborn graphs. The first thing I noticed was that the class variable, the one being predicted, is not evenly spread in the data. Instead, it is around a 70/30 split, where the majority of the data has no-recurrence-events for their class. This means that I need to be careful when creating the train/test splits, as I need to stratify this variable in order to maintain even distribution between the data splits. Another observation I noticed was that many of the variables seemed skewed towards one side. Either one category is much less frequent than the others or one is much more frequent. Lastly, I noticed that the node-caps variable consisted of the values “yes”, “no”, “*”, and “?” where the latter two values are much less frequent than the former. This could indicate missing or unclear values, where the node-caps are unable to be determined. I could treat these values as missing data and replace them with the mode, but I chose not to as they may actually serve a purpose rather than just signifying missing data.

ML Model Training Procedure

The first step I took in training the models was to create a good train/test split for the data. I did this by calling the scikit-learn train_test_split() function, making sure to stratify the y variable and opting for a 70/30 split. For the default KNN model, I trained it by simply calling the fit()

function on the model. I opted to use 5 neighbors as this model's hyperparameter, as I found that to be the conventional number for this model. I trained the Grid Search CV Model by calling the GridSearchCV() function, making sure to use recall as its scoring metric as I believe it to be the most important metric for this problem (explained later). After this, I selected the best estimator out of the ones trained, which happened to be one with a neighbor of 1, and proceeded to test this model on the test data. Lastly, I used the SGDClassifier for the linear classification model, as this model attempts to find the linear boundary between the data to predict the class variable for each data point. I trained this model by calling the SGDClassifier() function, fitting it on the training data, and testing its scores on the test data.

Model Performance Evaluation

In order to evaluate the models, it is necessary to classify which metric is most important for the problem. For this problem, I believe recall to be the most important. This is because when diagnosing a patient, false negatives are the most important value to minimize. It's better to falsely diagnose someone instead of not diagnosing a patient who actually has the disease, as this could lead to large consequences for their health. In predicting the test data, I found the GridSearchCV model to be the best, with a recall of 0.33. Second is the default KNN model, with a recall of 0.14. The worst performing would be the linear classifier, with a recall of 0.083. While the GridSearchCV did perform the best, I did find evidence of overfitting on the model, as its metrics on training data is way too high compared to the test data which could cause issues. Further metrics on training and test data can be found in the Jupyter Notebook.

Would I Trust This Model?

While I do believe that I effectively trained these models to fit on the training data, I don't think that any of their performance is trustable. Even the best performing model, the Grid Search CV performed with a recall of 0.33. This means that the model only correctly classifies 33% of the positive values, which means that there would be a 67% chance of the model missing a diagnosis. I believe that the models used may be too rudimentary for the complexity of the data that they are working with. Both KNN and linear classification would struggle with higher dimensional data. For the KNN, finding the "distances" between neighbors would be less meaningful as dimensions become higher, while for the linear classification finding a linear boundary would be harder for higher dimensions. Despite this, I believe that I was able to effectively create ML models that were able to work on the dataset provided to me.