COE 379L - Project 2

James Grant Robinett (jgr2722)

Introduction

Located in this folder are two files, this PDF “Project 2 Writeup JGR2722” and the code for parts 1 and 2, “part1_part2.ipynb”. The dataset used in this project was downloaded from the class github repository using this link: https://raw.githubusercontent.com/joestubbs/coe379L-sp24/master/datasets/unit02/project2.data, and was saved as “breast_cancer.csv”, which I will leave included just for convenience when running my code.

Data Pre-Processing

The first step I took after reading the ‘breast_cancer.csv’ data into a pandas dataframe was to look at the structure of the dataframe. This was first done using `cancer.shape()` and `cancer.size()`, which returned that the dataset had a structure of *286* rows and *10* columns, with a total cell count of *2860*. Before checking data types, I wanted to see if there were any null values or duplicates. I found that there were no null values, but there were a few duplicates, which were then dropped from the dataframe using `cancer = cancer.drop_duplicates()`.

Next, I wanted to look at the type of each data column, which I found using `cancer.info()`. I did this to see what variables I would like to numerical (int or float) values before using it to fit a model. From this, I decided that the *age, tumor-size and inv-nodes* columns needed to be converted to numerical values. The data for these columns were saved as a data range (example: age 30-49), so I first had to split the data and find the median between these values. To properly split this data, I did use help from chatGPT to figure out how to split around the ‘-’. This was done to make it easier to process the data numerically when applied to fitting the model.

With these values converted to numerical data (float64), I then did a last check for any strange values using the `unique()` function. This let me see for the remaining columns that there were again some question marks contained in my columns, specifically the *node-caps* and *breast-quad* columns. I printed out all of the rows with ‘?’ values included, and found that a total of 9 rows contained question marks. The technique I used to fix this was to simply drop these rows, since 9 rows seemed small enough that it wouldn’t affect the dataset as a whole. After that, I checked for any more question marks, saw none, then moved on to univariate and bivariate analysis.

Lastly, I decided on which columns I wanted to use OneHotEncoding on. I settled on encoding the *menopause, node-caps, deg-malig, breast, breast-quad,* and *irradiat* columns. Why I did this is because when looking at the unique values, I noticed that these data columns contained either a binary value, or 1-4 possible values. I decided that this was a small enough range of possible values that it would make sense to encode these variables, making it easier for my machine learning models to process. Once finished with encoding these columns, I checked the size of the new encoded dataframe, which resulted in a column size increase of *21* columns, and *263* rows. Checking one more time for duplicates or null values, I could now move on to part 2 of the project!

Multi-Model Fitting

With the data processed and prepared for analysis, we can now start to look at how to fit multiple models to predict breast cancer recurrence. I decided that the response variable (y) was `cancer['class']`, and everything else was used for the explanatory (X) variable. With X and Y defined, we could now move to creating a train test split using the Scikit-Learn `train_test_split` module. I decided on a 80/20 split, with 80% of my data being used to train the models and 20% being used to test the models' performance. Additionally, when looking at the univariate distribution of my class column data, I decided to use `stratify=y` which ensures that the training and test datasets have the same proportion of class labels as the input dataset. This was important to do to keep an even distribution of labels.

With the training and testing datasets created, I then moved on to automating multiple classification models being trained on this dataset. Using the baseline code given during class, I created a `class MultiModelClassifier(BaseEstimator):` which takes in an estimator we want to use on our dataset and can process multiple estimators when used in a pipeline. Speaking of pipelines, I then created a simple pipeline to scale the data and apply which model we want to look at to the dataset. This was done using just a few lines of code, looking like

```
# Make a Pipeline with multi model classifier
pipeline = Pipeline([
    ('scale', StandardScaler()),
    ('mmc', MultiModelClassifier()),
])
```

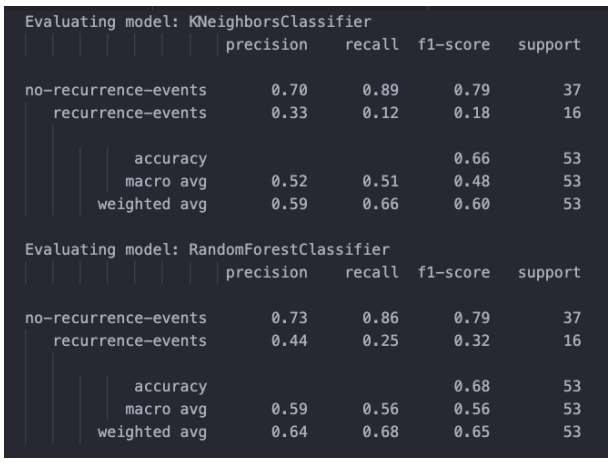
Lastly, I set up a parameter grid to search through during the grid search for each model I wanted to look at. These were all necessary steps for me to go through, as it allowed me to easily search through multiple models and parameters with these classification models all at once.

With the modular model process defined, I was able to start my `GridSearchCV` through this pipeline and parameters. I set it up so it goes through the models in a list, grabs their parameters, performs grid search, and then saves the best model to a local dictionary. I wanted to be able to have model persistence, but I was unable to figure that out in a reasonable time. In total, four classification models were selected for testing, `[KNeighborsClassifier(), RandomForestClassifier(), DecisionTreeClassifier(), LogisticRegression()]`.The best parameters were found using the grid search, and were printed like so:

```
Best parameters for KNeighborsClassifier: {'mmc__model': KNeighborsClassifier(), 'mmc__model__n_neighbors': 7}
Best parameters for RandomForestClassifier: {'mmc__model': RandomForestClassifier(), 'mmc__model__n_estimators': 95}
Best parameters for DecisionTreeClassifier: {'mmc__model': DecisionTreeClassifier(), 'mmc__model__max_depth': 2}
Best parameters for LogisticRegression: {'mmc__model': LogisticRegression(), 'mmc__model__C': 0.1}
```

Model Testing and Metrics

With the correct parameters tuned for each of the four models, we can now evaluate how well the model does on the testing data. I looped through the saved *best_models* variable and used `y_pred = model.predict(X_test)` to predict on the test data for each model. With the response variable predicted, I could then look at the classification report using Scikit Learn's classification_report, `print(classification_report(y_test, y_pred))`. This classification report is extremely useful, because it allows us to see the precision, recall, f1-score and accuracy metrics for each model. Located below are the metrics for all four models:



```
Evaluating model: KNeighborsClassifier
                    precision    recall  f1-score   support

no-recurrence-events       0.70      0.89      0.79        37
   recurrence-events       0.33      0.12      0.18        16

            accuracy                           0.66        53
           macro avg       0.52      0.51      0.48        53
        weighted avg       0.59      0.66      0.60        53

Evaluating model: RandomForestClassifier
                    precision    recall  f1-score   support

no-recurrence-events       0.73      0.86      0.79        37
   recurrence-events       0.44      0.25      0.32        16

            accuracy                           0.68        53
           macro avg       0.59      0.56      0.56        53
        weighted avg       0.64      0.68      0.65        53
```

```
Evaluating model: DecisionTreeClassifier
                    precision    recall  f1-score   support

no-recurrence-events       0.71      0.86      0.78        37
   recurrence-events       0.38      0.19      0.25        16

            accuracy                           0.66        53
           macro avg       0.54      0.53      0.52        53
        weighted avg       0.61      0.66      0.62        53

Evaluating model: LogisticRegression
                    precision    recall  f1-score   support

no-recurrence-events       0.69      0.84      0.76        37
   recurrence-events       0.25      0.12      0.17        16

            accuracy                           0.62        53
           macro avg       0.47      0.48      0.46        53
        weighted avg       0.56      0.62      0.58        53
```

In analyzing the performance of the classification models (KNeighborsClassifier, RandomForestClassifier, DecisionTreeClassifier, and LogisticRegression), the classification report provides a good look at how each model predicts breast cancer recurrence. The models perform decently with varying degrees of success in identifying the recurrence of cancer. KNeighborsClassifier (0.66), RandomForestClassifier (0.68) and DecisionTreeClassifier (0.66) have a higher accuracy compared to LogisticRegression (0.62). However, accuracy alone can be misleading in the context of medical diagnostics, where the cost of a false negative could be critical.

The model(s) I would recommend for this dataset would be either the RandomForestClassifier or the DecisionTreeClassifier, due to their ability to capture the complex relationships within the dataset. They offer a balance between knowing feature importance and being able to adapt to different data distributions. RandomForestClassifier, in particular, benefits from multiple decision trees to improve predictive performance and reduce overfitting tendencies when compared to a single decision tree. Most importantly, these models have higher recall values compared to the other models. This is important because recall is the most important metric we need to look at in the context of this data, and having a greater recall metric should be the most looked at feature of the model.

In the context of diagnosing cancer recurrence, the recall metric is the most important. A high recall indicates that a model is capable of identifying most of the positive cases, which is crucial in a medical setting where failing to detect recurrence (a false negative) can have serious, life-threatening implications for patients. Optimizing the model for recall ensures that fewer cases of recurrence go unnoticed. False positives are more acceptable as they would lead to further testing rather than missing out on potential treatment opportunities. However, we do need to be careful about high recall and low precision, as this would lead to too many false positives being made. In that context, RandomForestClassifier would beat out DecisionTreeClassifier by a little, with a precision of 0.44 vs 0.38 for 'recurrence-events'.

In conclusion, while each model provides valuable insights, the RandomForestClassifier stands out as the best option due to its higher accuracy and better recall rates. This model strikes a crucial balance between reducing false negatives and providing reliable predictions for healthcare providers.