Decision Tree Classification for Breast Cancer Data:

Analysis of Model Performance and Cost Complexity Pruning

By: Agrim Tripathi

Introduction

The task at hand is to build a classification decision tree model in python to predict the survival status (alive/dead) of breast cancer patients using the Gini impurity criterion. The dataset used was the SEER dataset that can be found on kaggle, and is also attached in the project folder.

The objective was to plot and compare the model performance by varying the complexity of the tree using cost complexity pruning.

This report will go through the whole script and its resulting plots with an overview of the various steps that were taken along the way.

Overview

Data Preprocessing: Encoded categorical variables (e.g., race, marital status) and split the data into training and test sets.

Model Building: Built a decision tree using Gini impurity and trained it on the dataset.

Cost-Complexity Pruning: Applied pruning by varying `ccp_alpha` to reduce overfitting and improve model performance.

Evaluation: Assessed model accuracy, validation loss and confusion matrix on the test set for each pruning level

The Dataset (Link)

It contains 1 header row and 4024 entries corresponding to each patient, in a .csv format. It has 16 columns namely :

```
['Age', 'Race', 'Marital Status', 'T Stage', 'N Stage', '6th Stage',
'differentiate', 'Grade', 'A Stage', 'Tumor Size', 'Estrogen Status',
'Progesterone Status', 'Regional Node Examined', 'Regional Node Positive',
'Survival Months', 'Status']
```

Out of these, ‘Status’ is our output feature which is to be predicted by a tree consisting of the 15 remaining features. Hence we split the data into x and y where x contains all the independent features (the first 15) whereas y contains only the status column.

According to the kaggle page there were no null/missing values in the set, so we moved on directly to the encoding process.

Preprocessing

The columns `['Race','Marital Status','T Stage','N Stage','6th Stage', 'A Stage','Estrogen Status','Progesterone Status']` are categorical and are hence **one-hot encoded** using the `get_dummies` function, while the columns `['Grade', 'differentiate']` are ordinal with a relative ranking amongst them and were hence remapped and subsequently **label encoded**. The `['Status']` column, being a binary categorical single-column output feature, was label encoded.

The resulting data frames were saved into x_encoded and y_encoded respectively.

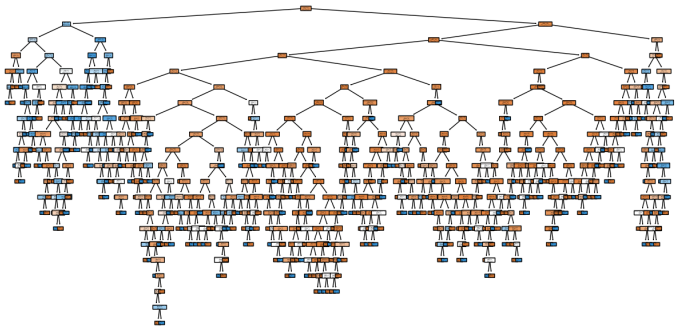
Then the `train_test_split` function was used to split the encoded data into training and testing datasets, x_train, x_test, y_train and y_test.

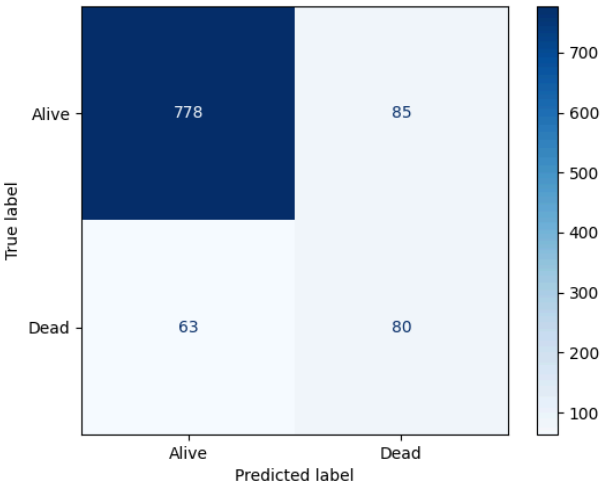
The Initial Decision Tree

```
clf_dt = DecisionTreeClassifier(random_state=42)

clf_dt = clf_dt.fit(x_train, y_train)
```

clf_dt and its confusion matrix are then plotted using matplotlib resulting in this:





This is a huge tree because it is not yet pruned and its complexity has not yet been varied. It performs much better in predicting alive patients than it does for the dead class of patients.

This is mostly due to the imbalanced source data which has much more entries in the alive class than in the dead class. Also the complexity of the tree may have led to overfitting which is solved by cost complexity pruning.

Cost Complexity Pruning (finding ccp_alpha using `cross_val_score()`)

`ccp_alpha` is a parameter of `DecisionTreeClassifier()` that decides the pruning level of the original which basically simplifies the tree by cutting off unnecessary branches with an expectation of higher accuracy scores.

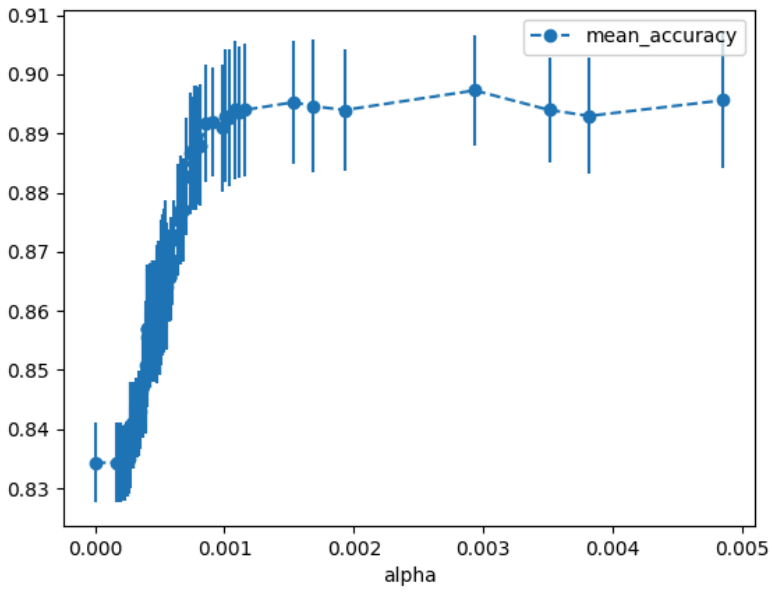
To find the exact value of our desired alpha we use

`clf_dt.cost_complexity_pruning_path(x_train,y_train)`

Which calculates the effective `ccp_alpha` values for pruning the tree. We then use cross validation to calculate the performance scores of each tree whilst varying `ccp_clpha`.

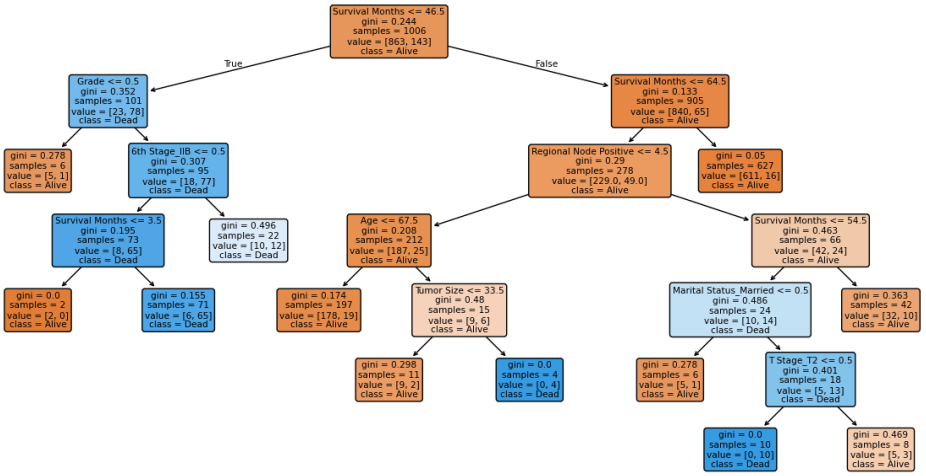
```
## ideal alpha using cross validation
alpha_loop_values = []
for ccp_alpha in ccp_alphas:
    clf_dt = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    scores = cross_val_score(clf_dt, x_train, y_train)
    alpha_loop_values.append([ccp_alpha, np.mean(scores), np.std(scores)])
```

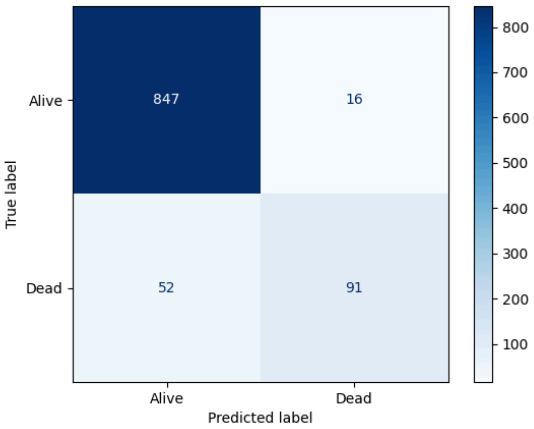
This allows us to plot an accuracy plot to see how the performance of the tree varies with different levels of pruning. The accuracy plot is below



Final Tree

Once we have the ideal `ccp_alpha` value, we can proceed to implement it to `clf_dt` and have our final tree `clf_dt_pruned`. It looks much simpler and performs better.





Suggestions to Improve the Model

1. Using a different method like the entropy criterion or the random forest method often leads to better results in such problems
2. Using resampled data that removes the class imbalance in the status column leads to much better results in predicting the dead class of patients. The confusion matrix below shows the result from a tree trained and tested on resampled data. It can be found in the project folder(resampled.py)