Kyphosis Disease Classification

Hanah Chang

1. Introduction

Our objective is to classify whether our patient had kyphosis after the spinal surgery. The dataset contains 3 predictors and 1 target variable. The dataset was downloaded from (https://www.kaggle.com/abbasit/kyphosis-dataset)

- Age: in months
- Number: the number of vertebrae involved
- Start: the number of the first (topmost) vertebra operated on.
- Kyphosis: our target variable. when level = 'present', a kyphosis was present after the surgery. level = 'absent' means otherwise.

2. Data & Libaray

Our dataset is consist of 81 observations

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
df = pd.read_csv("kyphosis.csv")
print(df.shape)
print(df.head(5))
```

```
(81, 4)
  Kyphosis  Age  Number  Start
0   absent   71       3      5
1   absent  158       3     14
2  present  128       4      5
3   absent    2       5      1
4   absent    1       4     15
```

Range for Age, Number, Start variables varies. However, since we are using Decision Tress algorithm, we do not need data scaling/normalization here.

In [3]:
```
df.describe()
```

Out[3]:

	Age	Number	Start
count	81.000000	81.000000	81.000000
mean	83.654321	4.049383	11.493827
std	58.104251	1.619423	4.883962
min	1.000000	2.000000	1.000000
25%	26.000000	3.000000	9.000000
50%	87.000000	4.000000	13.000000
75%	130.000000	5.000000	16.000000
max	206.000000	10.000000	18.000000

3. Explanatory Analysis & Data Cleaning

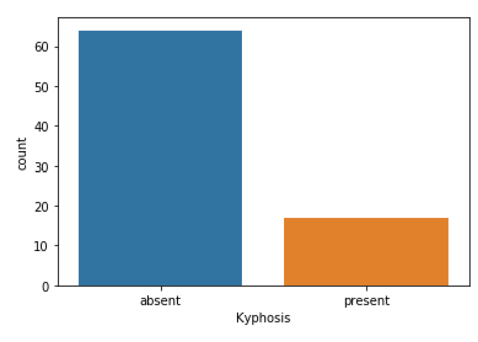
We can see that 26% of the patients (17 out of 64) developed kyphosis after spinal surgery.

In [4]:
```
df.Kyphosis.value_counts()
```

Out[4]:
```
absent     64
present    17
Name: Kyphosis, dtype: int64
```

In [5]:
```
sns.countplot(df['Kyphosis'], label = "Counts")
```

Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x27404ed7588>



We are going to transform categorical values from target variable into 0/1. That is, 'absent' turned into 0 and 'present' turned into 1 after applying LabelEncoder.

In [6]:
```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
LabelEncoder_y = LabelEncoder()
df['Kyphosis'] = LabelEncoder_y.fit_transform(df['Kyphosis'])
```

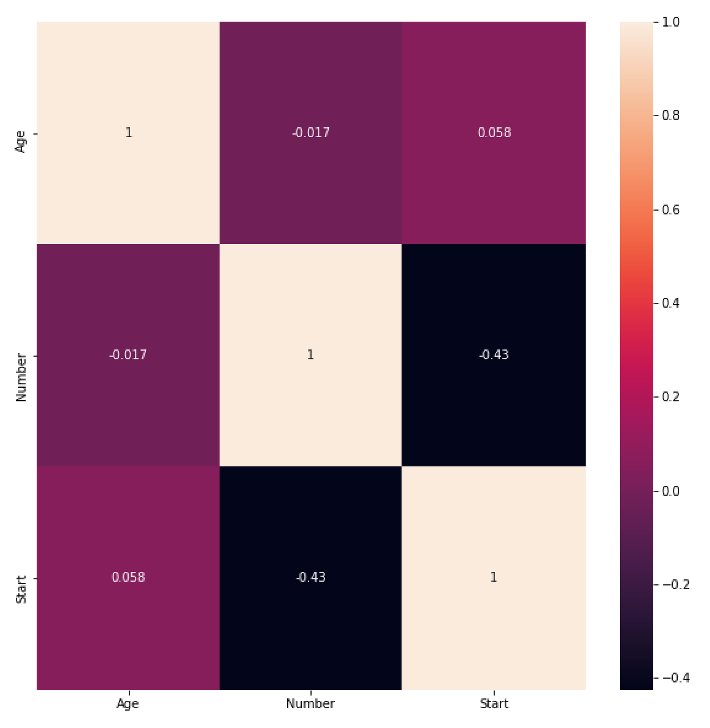
In [7]:
```
df.Kyphosis.value_counts()
```

Out[7]:
```
0    64
1    17
Name: Kyphosis, dtype: int64
```

Next we look into correlations between variables. There seem to be negative correlation between Number and Start variable. Since Decision Tree is non-parametric algorithm and it does not make assumptions on relationship between features, the trend we see won't impact on model performance.

In [8]:
```
plt.figure(figsize=(10,10))
sns.heatmap(df[['Age','Number','Start']].corr(), annot=True)
```

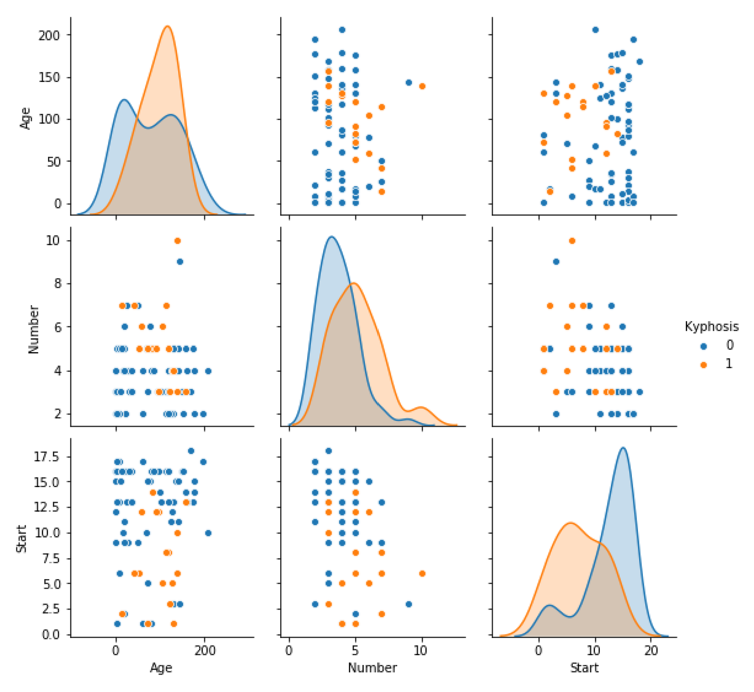
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x2740a6ec108>



From the pair chart, we can see that mean value for Kyphosis Present and Kyphosis Absent are quite similar, in case of Age and Number. Our task can be challenging.

In [10]:
```
sns.pairplot(df, hue='Kyphosis', vars = ['Age', 'Number', 'Start'])
```

Out[10]: <seaborn.axisgrid.PairGrid at 0x2740a69f548>



4. Decision Tree, Random Forest - Training / Optimizing

We are going to train our model using two algorithms. We expect the Random Forest will throw a better result since the algorithm overcomes Decision Tree's overfitting problem by taking averages of multiple predictions from multiple random decision trees.

In [11]:
```
X = df.drop(['Kyphosis'],axis=1)
y = df['Kyphosis']
```

In [12]:
```
from sklearn.model_selection import train_test_split
```

In [32]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

In [33]:
```
print("X_train", X_train.shape)
print("y_train",y_train.shape)
print("X_test",X_test.shape)
print("y_test",y_test.shape)
```

```
X_train (56, 3)
y_train (56,)
X_test (25, 3)
y_test (25,)
```

We fit two models using DecisionTreeClassifier() and RandomForestClassifier. By setting n_estimators=150, we are pruning 150 trees and the result will take majority vote.

In [34]:
```
from sklearn.tree import DecisionTreeClassifier
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train,y_train)
```

Out[34]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [35]:
```
from sklearn.ensemble import RandomForestClassifier
RandomForest = RandomForestClassifier(n_estimators=150)
RandomForest.fit(X_train, y_train)
```

Out[35]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=150,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

With 'DecisionTreeClassifier' object, we can obtain what we call feature importance (= significance of attribute), and get to know which variable has the highest impact when it comes to training. Feature importance is calculated based on how much each feature contributes to decreasing the weighted impurity.

In [36]:
```
feature_importances = pd.DataFrame(decision_tree.feature_importances_,
                                   index = X_train.columns,
                                    columns=['importance']).sort_values('importance',ascending=False)
```

We can see that Age returns the lowest impurity and entropy after splitting.

In [37]:
```
feature_importances
```

Out[37]:

	importance
Start	0.485735
Age	0.462962
Number	0.051304

5. Decision Tree, Random Forest - Evaluating

For training data set, we didn't misclassify any sample with training data, with two algorithms.

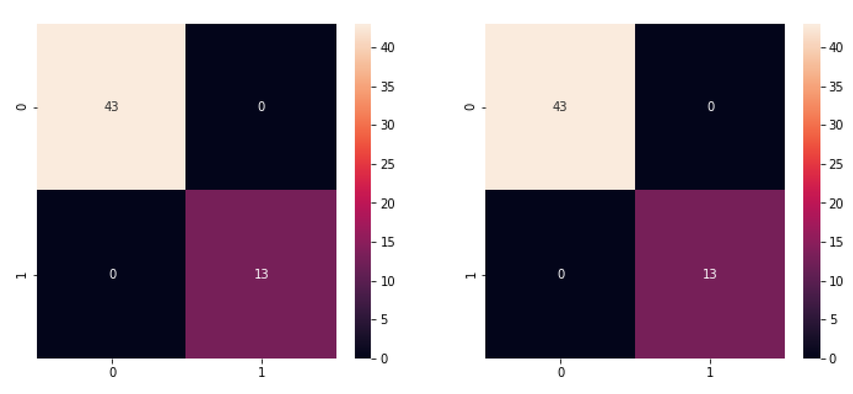
In [38]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [39]:
```
y_predict_train_tree = decision_tree.predict(X_train)
y_predict_train_forest = RandomForest.predict(X_train)
```

In [40]:
```
cm = confusion_matrix(y_train, y_predict_train_tree)
cm2 = confusion_matrix(y_train, y_predict_train_forest)
```

In [41]:
```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
sns.heatmap(cm, annot=True)
plt.subplot(1,2,2)
sns.heatmap(cm2, annot=True)
```

Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x2740bb08448>



In [43]:
```
print(classification_report(y_train, y_predict_train_tree))
print(classification_report(y_train, y_predict_train_forest))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        43
           1       1.00      1.00      1.00        13

    accuracy                           1.00        56
   macro avg       1.00      1.00      1.00        56
weighted avg       1.00      1.00      1.00        56

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        43
           1       1.00      1.00      1.00        13

    accuracy                           1.00        56
   macro avg       1.00      1.00      1.00        56
weighted avg       1.00      1.00      1.00        56
```

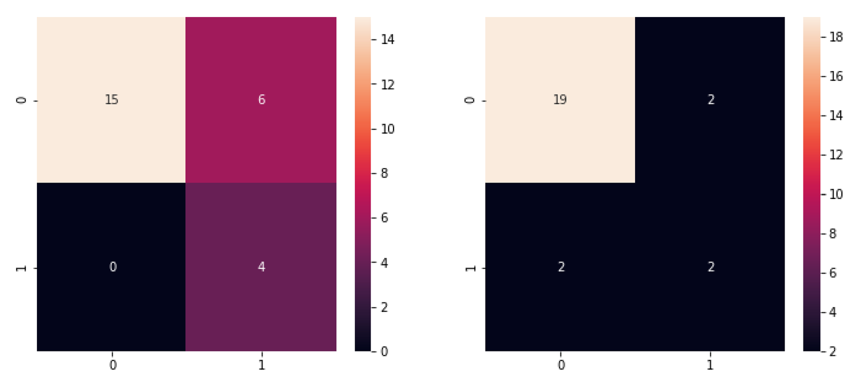
When applied both model on testing data set, Decision tree showed 79% F1 score vs. Random Forest showed 84%.

In [44]:
```
y_predict_test_tree = decision_tree.predict(X_test)
y_predict_test_forest = RandomForest.predict(X_test)
```

In [45]:
```
cm3 = confusion_matrix(y_test, y_predict_test_tree)
cm4 = confusion_matrix(y_test, y_predict_test_forest)
```

In [46]:
```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
sns.heatmap(cm3, annot=True)
plt.subplot(1,2,2)
sns.heatmap(cm4, annot=True)
```

Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x2740bc91ac8>



In [48]:
```
print(classification_report(y_test, y_predict_test_tree))
print(classification_report(y_test, y_predict_test_forest))
```

```
              precision    recall  f1-score   support

           0       1.00      0.71      0.83        21
           1       0.40      1.00      0.57         4

    accuracy                           0.76        25
   macro avg       0.70      0.86      0.70        25
weighted avg       0.90      0.76      0.79        25

              precision    recall  f1-score   support

           0       0.90      0.90      0.90        21
           1       0.50      0.50      0.50         4

    accuracy                           0.84        25
   macro avg       0.70      0.70      0.70        25
weighted avg       0.84      0.84      0.84        25
```

6. Conclusion

In general, ensemble algorithm (such as random forest) returns better result than decision tree model because ensemble algorithm overcomes the issues with single decision trees by reducing the effect of noise. Our project was another proof.