One Year Life Expectancy Post Thoracic Surgery

OVERVIEW:

Lung cancer is the most common form of cancer world-wide, and the most common cause of cancer death. Radical surgical resection, with or without adjuvant treatment, is still a Prerequisite for cure. In spite of different additional modes of treatment, survival is still poor.It is important to have knowledge of peri- and postoperative mortality (life expectancy) and morbidity (health complications), and also of risk factors prior to surgery, to be able to improve the quality of operative procedures and identify patients running the highest risk.This helps to optimize the patient's condition, medication and respiratory status before surgery. Furthermore, the operative risks must be considered in relation to the long-term results in order to identify patients who will clearly benefit from surgery.

- DATA COLLECTION
- IBM CLOUD COUNT
- MODEL BUILDING
- APPLICATION BUILDING

PROBLEM:

1:Patients who receive thoracic surgery for lung cancer do so with the expectation that their lives will be prolonged for a sufficient amount of time afterwards.
2: The problem to solve is whether there is a way to determine postoperative 1 year survival of lung cancer patients utilizing the patient

attributes in the data set.

Proposed solution:

The aim of the project is to examine the operative mortality (life expectancy) and morbidity (Health issues) after lung cancer surgery and to identify factors associated with an adverse Outcome. IBM Watson AutoAI Machine Learning Service is developed to predict the post operative life expectancy of lung cancer patients using the computational methods. These methods were used specifically to predict whether a lung cancer patient will survive one year after he or she has had thoracic surgery. The results of each of the techniques were then measured and compared based on accuracy and performance.. The model is deployed on IBM cloud to get scoring end point which can be used as API in mobile app or web app building. We are developing a web application which is built using node red service. We make use of the scoring end point to give user input values to the deployed model. The model prediction is then showcased on User Interface.

Purpose:

Through this repository we can:

1. Apply the fundamental concepts of machine learning from an available dataset

2. Evaluate and interpret my results and justify my interpretation based on observed data set
3. Create notebooks that serve as computational records and document my thought process.

The analysis is divided into four sections, saved in juypter notebooks in this repository

1. Identifying the problem and Data Sources
2. Exploratory Data Analysis
3. Pre-Processing the Data
4. Build model

Data Set :

➢ Original from UCI Machine Learning Repository

○ Collected retrospectively at Wroclaw Thoracic Surgery Centre for patients who underwent major lung resections for primary lung cancer in the years 2007-2011 ○ 470 instances and no missing values

➢ This report consists of 454 patient data. ○ Excluding outliers from FEV1 and Age columns

Descriptions of Attributes :

Attribute Description Weakness Weakness, prior to surgery (T = 1, F = 0)
Tumor_Size T in clinical TNM - size of the original tumor, 1 (smallest) to 4 (largest) Diabetes_Mellitus Type 2 diabetes mellitus (T = 1, F = 0)
MI_6mo Myocardial Infarction (Heart Attack) up to 6 months prior (T = 1, F = 0)
PAD Peripheral arterial diseases (T = 1, F = 0)
Smoking Smoking (T = 1, F = 0)
Asthma Asthma (T = 1, F = 0)
Age Age at surgery Death_1yr 1 year survival period - (T) value if died (T = 1, F = 0)

Hypothesis Testing:

➢ Null Hypothesis: The 1 year death and live patients have the same mean, tested for each attribute.

➢ Test Statistic: Mean difference between death and live patients

➢ Significance level: 0.05

Results of Hypothesis Testing:

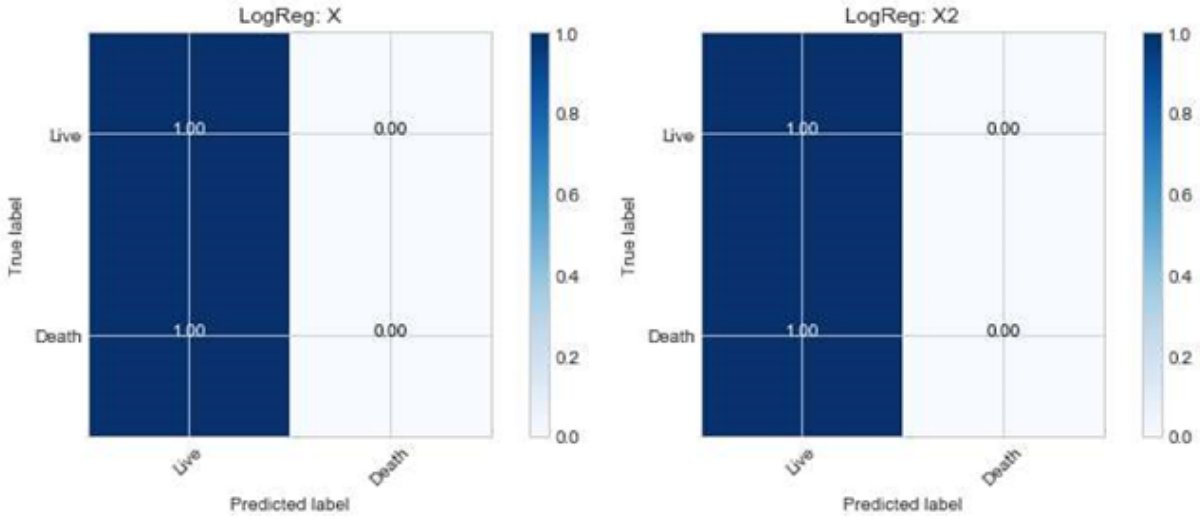
Attribute P value FVC 0.1706 FEV1 0.0588
Performance 0.0300 Pain 0.0964
Haemoptysis 0.0623
Dyspnoea 0.0242
Cough 0.0320
Weakness 0.0606
Tumor_Size 0.0003
Diabetes_Mellitus 0.0209
MI_6mo 0.7264
PAD 0.3498
Smoking 0.0581
Asthma 0.7178
Age 0.2714

Machine Learning (Supervised Classification)

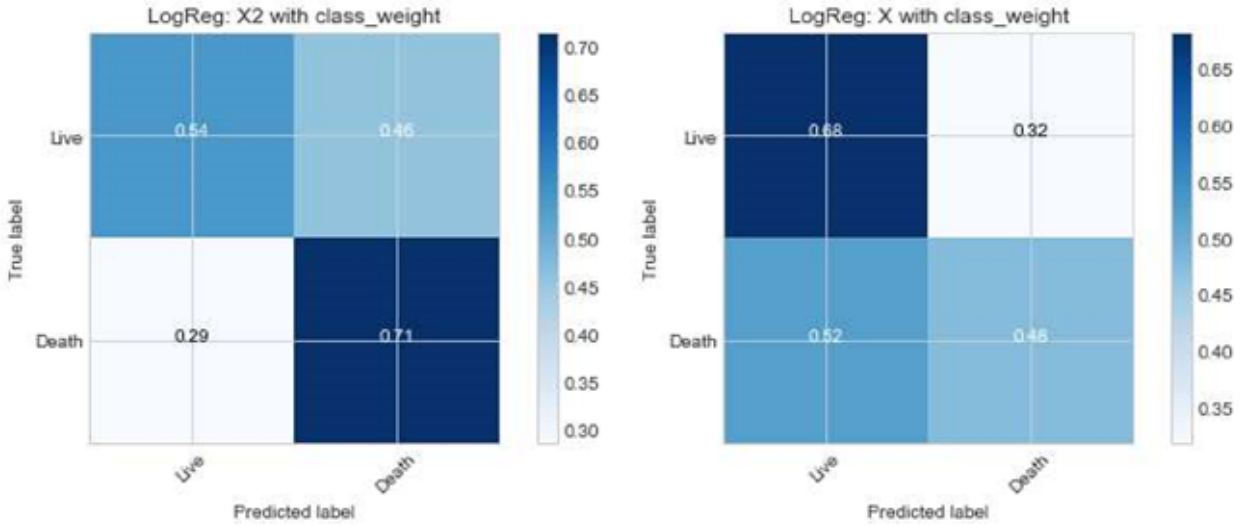
I focused on utilizing Logistic Regression and Random Forest Classifier for this supervised classification problem. From EDA and hypothesis testing to gather p values, I realized which attributes are significant in identifying the mean difference of those who lived and died in the 1 year period after surgery. I wanted focus the test on two different X data sets. The first data set drops the target variable, Death_1yr, and also the two attributes that shows little representation in the data itself, MI_6mo and Asthma. This data is referred to as X. The other data set only includes the attributes of significance concluded from the hypothesis testing in the EDA section: Performance, Dyspnoea, Cough, Tumor_Size, Diabetes_Mellitus. This data set is referred to as X2.

Since the data set is imbalanced and mostly live patients (85%), just predicting all live patients will give a high accuracy score ~85%. So for the model, accuracy will not be a good score method and instead I will look at average precision score, which summarizes the precision-recall curve. Also for the imbalance, there are couple options including downsampling, upsampling, or adjusting class weights to balance the classes. Since downsampling will create a small data set to work with and upsampling may complicate the data further, I will focus on adjusting the class weights.

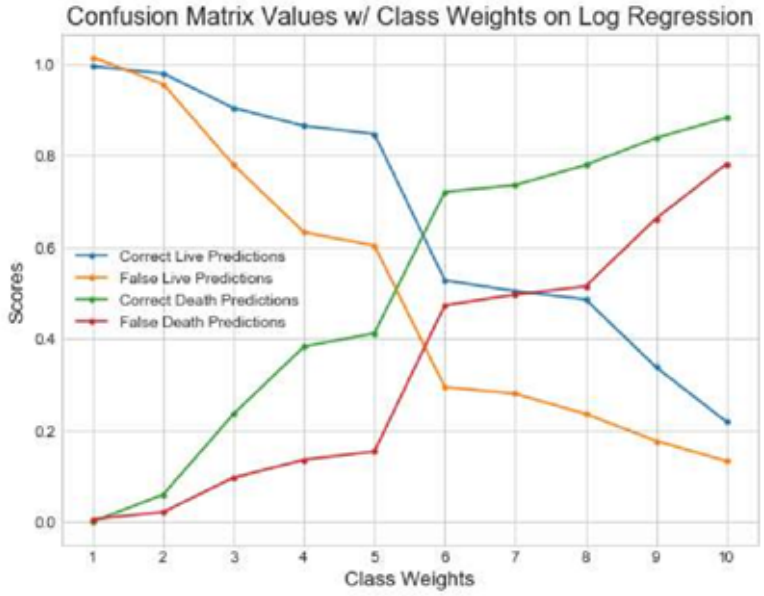
Logistic Regression



Since the data set is imbalanced with only 15% patient death, the results of the model without any class weight to offset this imbalance favors the live column in the confusion matrix. As you can see above, the model predicts mostly all live patients to maximize the accuracy score to 85%, the size of the live patient data, in both the X and X2 data sets.



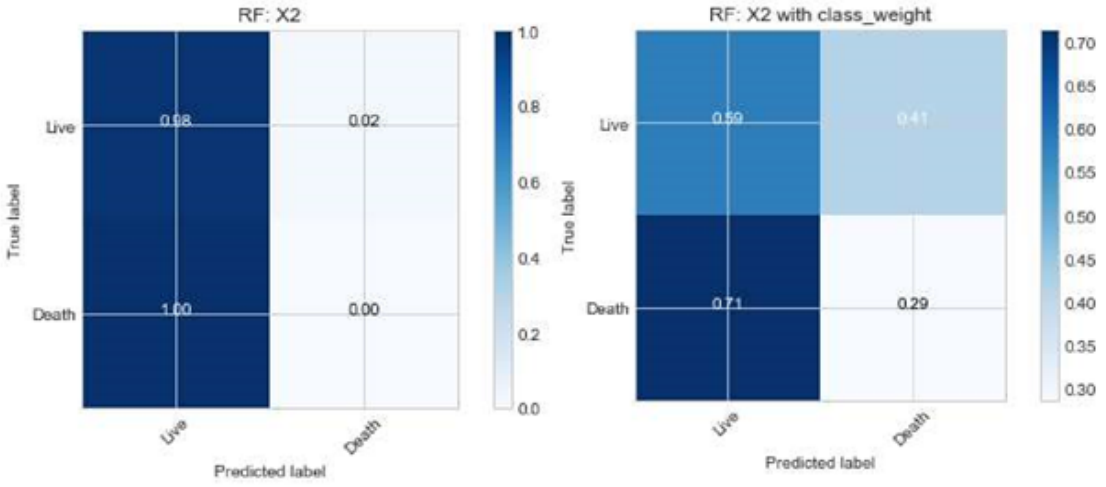
With the class weight parameter, the death prediction rate increases at the cost of live patient prediction, and also the accuracy. In order to see the effectiveness of the model for my purpose, the confusion matrix or classification report can be used to assess the death predictions. Also, the average precision score is a good summary of the precision-recall curve, which is useful in this case.



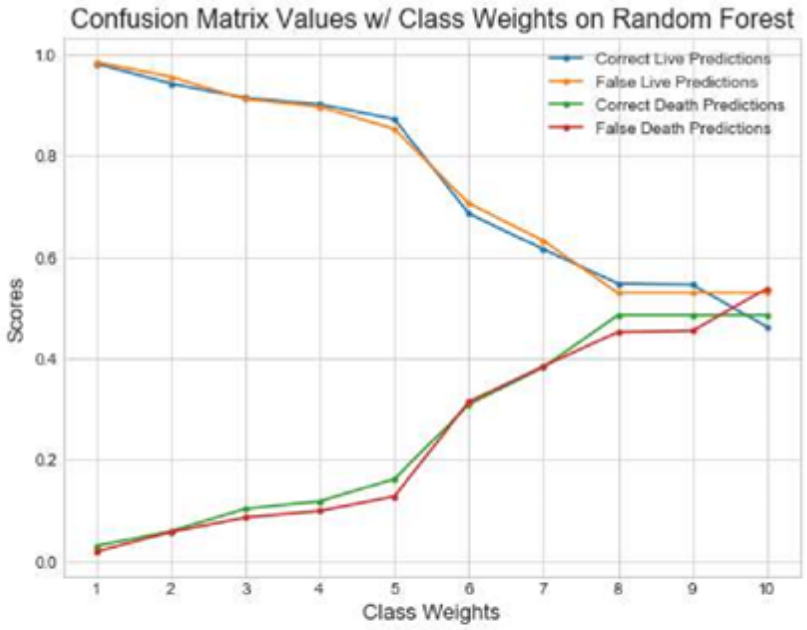
The class weight argument is set to 'balanced' to equalize the death to live ratio, which is 15 to 85. This argument can be altered with any ratio value and the effects can be seen in the graphs above. Although the correct death predictions increased with more class weight on the deaths, the false death predictions increased as well with decrease in correct live predictions. The influence of class weights can be seen in the graph above. Interesting to note that the score dips dramatically around the 5.67 value, which is the equalizing point for the ratio 15 to 85.

Random Forest Classifier

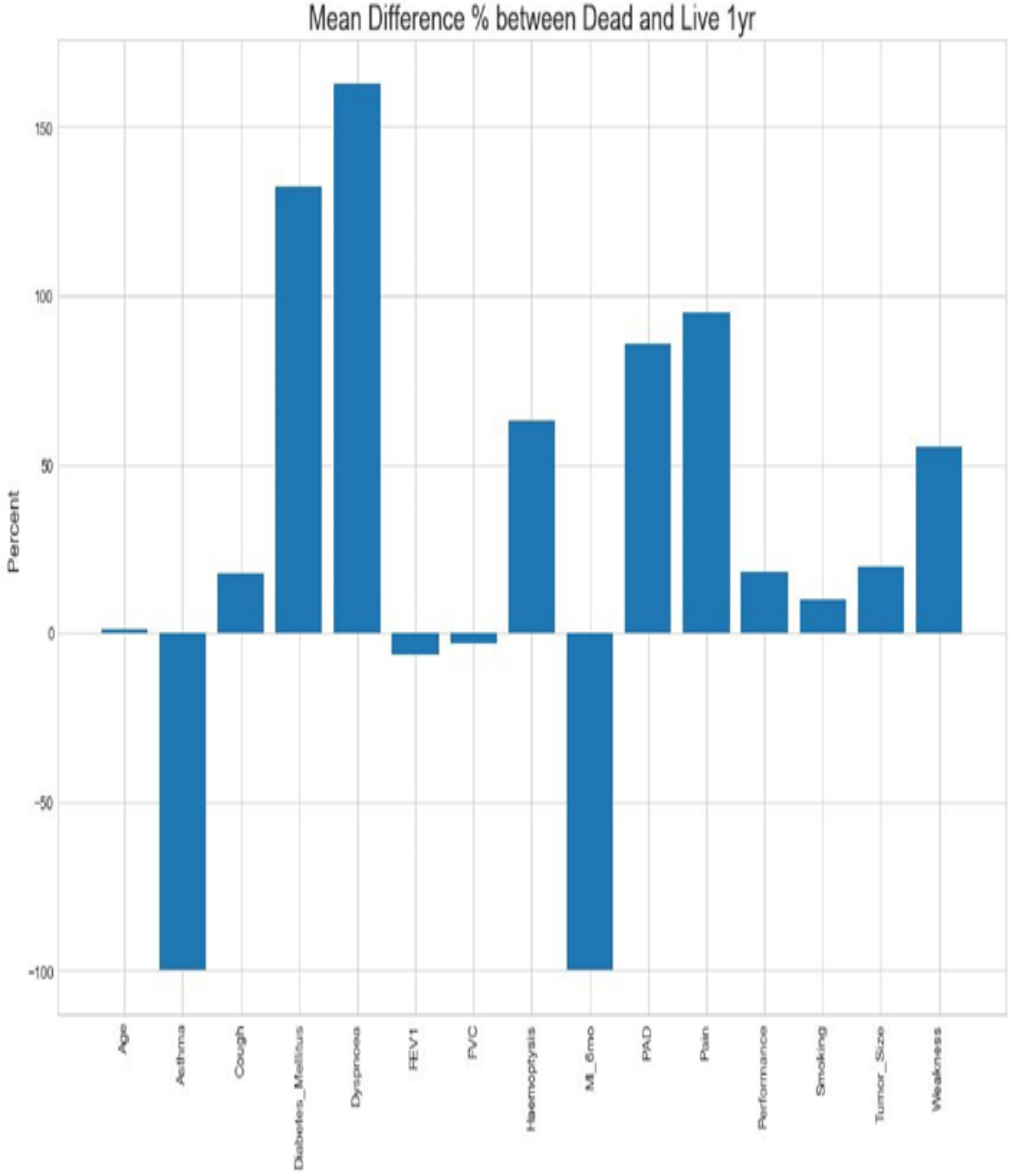
For the random forest classifier, I focused on utilizing the X2 data set to see performance comparative to the log regression model. Similar to the log regression, the random forest classifier heavily favors the live patient prediction without any class weight hyperparameters.

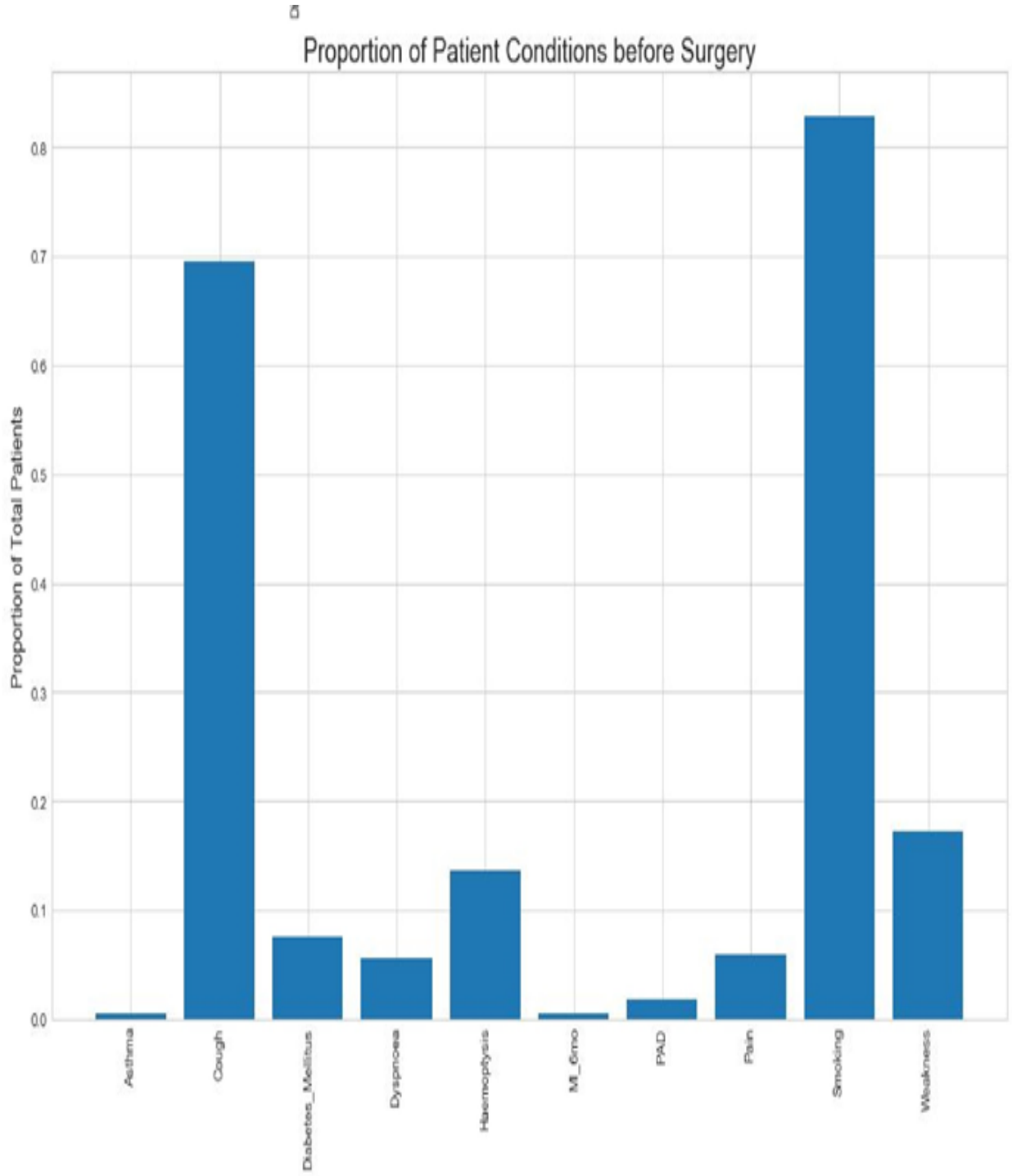


Similar to the Logistic Regression models, the Random Forest predicts deaths better with a class weight parameter to balance the data. The plot reveals the cost of correct live predictions and benefit of correct death predictions with differing class weights. It is interesting to note the different pattern this model takes compared to the log regression graphs above. Based solely on average precision, the log regression produces better results.



However, with hyperparameter tuning, the random forest classifier delivers higher average precision scores compared to what the log regression model did in any class weight value. It is interesting to note that GridSearchCV model notes the best parameter as having no class weight argument with the tested parameters in the report. So, there probably are combinations of hyperparameters that perform better than the models highlighted above with


Mean Difference % between Dead and Live 1yr
Percent
150
100
50
0
-50
-100
Age
Asthma
Cough
Diabetes_Mellitus
Dyspnoea
FEV1
FVC
Haemoptysis
MI_6mo
PAD
Pain
Performance
Smoking
Tumor_Size
Weakness


Proportion of Patient Conditions before Surgery
Proportion of Total Patients
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Asthma
Cough
Diabetes_Mellitus
Dyspnoea
Haemoptysis
MI_6mo
PAD
Pain
Smoking
Weakness

Source code:

```
%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style("whitegrid")
import types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share the notebook.
client_ca664f4bd18b4ee7be1a254407d5a9cb = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='LnDsssRcQkVEuzSKFi8xiJ_iPDfljTBahv39piIi9d8c',
    ibm_auth_endpoint="https://iam.cloud.ibm.com/oidc/token",
    config=Config(signature_version='oauth'),
    endpoint_url='https://s3.eu-geo.objectstorage.service.networklayer.com')

body = client_ca664f4bd18b4ee7be1a254407d5a9cb.get_object(Bucket='oneyearlifeexpectancypostthoracic-donotdelete-pr-ydkg5pb794dfn6',Key='Dataset.csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

Dataset = pd.read_csv(body)
Dataset.head()
Dataset.describe()
```

```
# Stats for live and death after 1 yr patients
live = Dataset[Dataset['Death_1yr'] == 0]
death = Dataset[Dataset['Death_1yr'] == 1]

cond = ['FVC', 'FEV1', 'Performance', 'Pain', 'Haemoptysis', 'Dyspnoea', 'Cough',
'Weakness',\
      'Tumor_Size', 'Diabetes_Mellitus', 'MI_6mo', 'PAD', 'Smoking', 'Asthma', 'Age']

l = [np.mean(live[c]) for c in cond]
d = [np.mean(death[c]) for c in cond]

ld = pd.DataFrame(data={'Attribute': cond, 'Live 1yr Mean': l, 'Death 1yr Mean': d})
ld = ld.set_index('Attribute')

print('Death: {:d}, Live: {:d}'.format(len(death), len(live)))
print("1 year death: {:.2f}% out of 454
patients".format(np.mean(Dataset.Death_1yr)*100))
ld
# Percentage difference in means of live vs death patients
d = np.array(d)
l = np.array(l)

p_diff = (d-l)/l*100

fig, axes = plt.subplots(2,1,figsize=(12,18))

axes[0].bar(cond, p_diff)
axes[0].set_title('Mean Difference % between Dead and Live 1yr', fontsize=18)
axes[0].set_xticks(cond)
axes[0].set_xticklabels(cond, rotation=90)
axes[0].set_ylabel('Percent', fontsize=13)
```

 # Count plot of true/false condition columns

```
tf_col = ['Pain', 'Haemoptysis', 'Dyspnoea', 'Cough', 'Weakness', 'Diabetes_Mellitus',
'MI_6mo', 'PAD', 'Smoking', 'Asthma']
tf_sum = [Dataset[col].sum()/454 for col in tf_col]
```

```
axes[1].bar(tf_col, tf_sum)
axes[1].set_xticks(tf_col)
axes[1].set_xticklabels(tf_col, rotation=90)
axes[1].set_ylabel('Proportion of Total Patients', fontsize=13)
axes[1].set_title('Proportion of Patient Conditions before Surgery', fontsize=18)

plt.tight_layout()

plt.show()
```
 # Count plots of Diagnosis, Tumor_Size, Performance with difference of live and death data

```
fig, axes = plt.subplots(3,1,figsize=(10,15))

sns.countplot(x='Diagnosis', hue='Death_1yr', data=Dataset, palette='Blues_d',
ax=axes[0]).set_title('Diagnosis', fontsize=18)
sns.countplot(x='Tumor_Size', hue='Death_1yr', data=Dataset, palette='Blues_d',
ax=axes[1]).set_title('Tumor_Size', fontsize=18)
sns.countplot(x='Performance', hue='Death_1yr', data=Dataset, palette='Blues_d',
ax=axes[2]).set_title('Performance', fontsize=18)

plt.tight_layout()
def permutation_sample(data1, data2):
    """Generate a permutation sample from two data sets."""
    data = np.concatenate((data1, data2))
    permuted_data = np.random.permutation(data)

    perm_sample_1 = permuted_data[:len(data1)]
    perm_sample_2 = permuted_data[len(data1):]

    return perm_sample_1, perm_sample_2

def draw_perm_reps(data_1, data_2, func, size=1):
    """Generate multiple permutation replicates."""
    perm_replicates = np.empty(size)
```

```
    for i in range(size):
        perm_sample_1, perm_sample_2 = permutation_sample(data_1, data_2)
        perm_replicates[i] = func(perm_sample_1, perm_sample_2)

    return perm_replicates
def diff_of_means(data_1, data_2):
    """Difference in means of two arrays."""
    diff = np.mean(data_1) - np.mean(data_2)
    return diff
# Hypothesis testing with Permutations of data
condition = ['FVC', 'FEV1', 'Performance', 'Pain', 'Haemoptysis', 'Dyspnoea', 'Cough',
'Weakness',\
            'Tumor_Size', 'Diabetes_Mellitus', 'MI_6mo', 'PAD', 'Smoking', 'Asthma', 'Age']
p_val = []

for c in condition:
    empirical_diff_means = diff_of_means(death[c], live[c])
    perm_replicates = draw_perm_reps(death[c], live[c], diff_of_means, size=10000)
    if empirical_diff_means > 0:
        p = np.sum(perm_replicates >= empirical_diff_means) / len(perm_replicates)
        p_val.append(p)
    else:
        p = np.sum(perm_replicates <= empirical_diff_means) / len(perm_replicates)
        p_val.append(p)

print(list(zip(condition, p_val)))

# Grid Search Hyperparameters for Random Forest
rfc = RandomForestClassifier(random_state=1111)

param_grid = {
          "n_estimators" : [5, 6, 7, 8, 9],
          "max_depth" : [7, 8, 9, 10],
          "min_samples_leaf" : [7, 8, 9, 10],
          "class_weight": [None, 'balanced', {0: 1, 1: 5}]}

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5,
```

```
scoring='average_precision')
CV_rfc.fit(X2, y)
print(CV_rfc.best_params_)
print('Best average precision score: {:.4f}'.format(CV_rfc.best_score_))
```

Conclusion :

Since the data set is imbalanced in logistic regression with only 15% patient death, the results of the model without any class weight to offset this imbalance favors the live column in the confusion matrix. As you can see above, the model predicts mostly all live patients to maximize the accuracy score to 85%, the size of the live patient data, in both the X and X2 data sets.

With the class weight parameter, the death prediction rate increases at the cost of live patient prediction, and also the accuracy. In order to see the effectiveness of the model for my purpose, the confusion matrix or classification report can be used to assess the death predictions. Also, the average precision score is a good summary of the precision-recall curve, which is useful in this case.

The class weight argument is set to 'balanced' to equalize the death to live ratio, which is 15 to 85. This argument can be altered with any ratio value and the effects can be seen in the graphs above. Although the correct death predictions increased with more class weight on the deaths, the false death predictions increased as well with decrease in correct live predictions. The influence of class weights can be seen in the graph above. Interesting to note that the score dips dramatically around the 5.67 value, which is the equalizing point for the ratio 15 to 85.

Similar to the Logistic Regression models, the Random Forest predicts

deaths better with a class weight parameter to balance the data. The plot reveals the cost of correct live predictions and benefit of correct death predictions with differing class weights. It is interesting to note the different pattern this model takes compared to the log regression graphs above. Based solely on average precision, the log regression produces better results.

However, with hyperparameter tuning, the random forest classifier above delivers higher average precision scores compared to what the log regression model did in any class weight value. It is interesting to note that the GridSearchCV above notes the best parameter as having no class weight argument. So, there probably are combinations of hyperparameters that perform better than the models highlighted above with class weights. For future optimization, more hyperparameter tuning can be done with a more in-depth parameter grid or utilizing RandomSearch if needed.