C3_W1_Assignment

August 1, 2024

1 Estimating Treatment Effect Using Machine Learning

Welcome to the first assignment of **AI for Medical Treatment**!

You will be using different methods to evaluate the results of a randomized control trial (RCT).

You will learn: - How to analyze data from a randomized control trial using both: - traditional statistical methods - and the more recent machine learning techniques - Interpreting Multivariate Models - Quantifying treatment effect - Calculating baseline risk - Calculating predicted risk reduction - Evaluating Treatment Effect Models - Comparing predicted and empirical risk reductions - Computing C-statistic-for-benefit - Interpreting ML models for Treatment Effect Estimation - Implement T-learner

1.1 Table of Contents

- Section **??**
  - Section **??**
  - Section **??**
  - Section **??**

1.2 Packages

We'll first import all the packages that we need for this assignment.

- `pandas` is what we'll use to manipulate our data
- `numpy` is a library for mathematical and scientific operations
- `matplotlib` is a plotting library
- `sklearn` contains a lot of efficient tools for machine learning and statistical modeling
- `random` allows us to generate random numbers in python
- `lifelines` is an open-source library that implements c-statistic
- `itertools` will help us with hyperparameters searching

1.3 Import Packages

Run the next cell to import all the necessary packages, dependencies and custom util functions.

```
In [9]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import sklearn
        import random
        import lifelines
        import itertools
        from public_tests import *

        plt.rcParams['figure.figsize'] = [10, 7]
```

1. Dataset ### 1.1 Why RCT?

In this assignment, we'll be examining data from an RCT, measuring the effect of a particular drug combination on colon cancer. Specifically, we'll be looking the effect of Levamisole and Fluorouracil on patients who have had surgery to remove their colon cancer. After surgery, the curability of the patient depends on the remaining residual cancer. In this study, it was found that this particular drug combination had a clear beneficial effect, when compared with Chemotherapy. ### 1.2 Data Processing In this first section, we will load in the dataset and calculate basic statistics. Run the next cell to load the dataset. We also do some preprocessing to convert categorical features to one-hot representations.

```
In [10]: data = pd.read_csv("data/levamisole_data.csv", index_col=0)
```

Let's look at our data to familiarize ourselves with the various fields.

```
In [11]: print(f"Data Dimensions: {data.shape}")
         data.head()
```

```
Data Dimensions: (607, 14)
```

Out[11]:

	sex	age	obstruct	perfor	adhere	nodes	node4	outcome	TRTMT	\
1	1	43	0	0	0	5.0	1	1	True	
2	1	63	0	0	0	1.0	0	0	True	
3	0	71	0	0	1	7.0	1	1	False	
4	0	66	1	0	0	6.0	1	1	True	
5	1	69	0	0	0	22.0	1	1	False	

	differ_2.0	differ_3.0	extent_2	extent_3	extent_4
1	1	0	0	1	0
2	1	0	0	1	0
3	1	0	1	0	0
4	1	0	0	1	0
5	1	0	0	1	0

Below is a description of all the fields (one-hot means a different field for each level): `- sex (binary): 1 if Male, 0 otherwise - age (int): age of patient at start of the study - obstruct (binary): obstruction of colon by tumor - perfor (binary): perforation of colon - adhere (binary): adherence to nearby organs - nodes (int): number of lymphnodes with detectable cancer - node4 (binary): more than 4 positive lymph nodes - outcome (binary): 1 if died within 5 years - TRTMT (binary): treated with levamisole + fluoroucil - differ (one-hot): differentiation of tumor - extent (one-hot): extent of local spread`

In particular pay attention to the `TRTMT` and `outcome` columns. Our primary endpoint for our analysis will be the 5-year survival rate, which is captured in the `outcome` variable.

Exercise 1 - proportion_treated

Since this is an RCT, the treatment column is randomized. Let's warm up by finding what the treatment probability is.

$$p_{treatment} = \frac{n_{treatment}}{n}$$

- $n_{treatment}$ is the number of patients where `TRTMT = True`
- n is the total number of patients.

```
In [12]: # UNQ_C1 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
         def proportion_treated(df):
             """
             Compute proportion of trial participants who have been treated

             Args:
                 df (dataframe): dataframe containing trial results. Column
                                 'TRTMT' is 1 if patient was treated, 0 otherwise.

             Returns:
                 result (float64): proportion of patients who were treated
             """
```

```
    ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###

    proportion = df['TRTMT'].sum()/len(df['TRTMT'])

    ### END CODE HERE ###

    return proportion
```

```
In [13]: ### test cell ex1: you cannot edit this cell
         proportion_treated_test(proportion_treated)
```

```
Test Case 1:

Example df:

   outcome  TRTMT
0        0      0
1        1      1
2        1      1
3        1      1
Proportion of patient treated: 0.75

Test Case 2:

Example df:

   outcome  TRTMT
0        0      0
1        1      0
2        0      0
3        0      0
Proportion of patient treated: 0.0

Test Case 3:

Example df:

   outcome  TRTMT
0        0      0
1        1      1
2        1      0
3        1      0
Proportion of patient treated: 0.25

 All tests passed.
```

Expected Output:

```
Test Case 1:

Example df:
    outcome  TRTMT
0        0      0
1        1      1
2        1      1
3        1      1
Proportion of patient treated: 0.75

Test Case 2:

Example df:
    outcome  TRTMT
0        0      0
1        1      0
2        0      0
3        0      0
Proportion of patient treated: 0.0

Test Case 3:

Example df:
    outcome  TRTMT
0        0      0
1        1      1
2        1      0
3        1      0
Proportion of patient treated: 0.25

 All tests passed.
```

Next let's run it on our trial data.

```
In [14]: p = proportion_treated(data)
         print(f"Proportion Treated: {p} ~ {int(p*100)}%")

Proportion Treated: 0.49093904448105435 ~ 49%
```

Exercise 2 - event_rate

Next, we can get a preliminary sense of the results by computing the empirical 5-year death probability for the treated arm versus the control arm.

The probability of dying for patients who received the treatment is:

$$p_{\text{treatment, death}} = \frac{n_{\text{treatment,death}}}{n_{\text{treatment}}}$$

- $n_{\text{treatment,death}}$ is the number of patients who received the treatment and died.
- $n_{\text{treatment}}$ is the number of patients who received treatment.

The probability of dying for patients in the control group (who did not received treatment) is:

$$p_{\text{control, death}} = \frac{n_{\text{control,death}}}{n_{\text{control}}}$$

- $n_{\text{control,death}}$ is the number of patients in the control group (did not receive the treatment) who died. - n_{control} is the number of patients in the control group (did not receive treatment).

```
In [15]: # UNQ_C2 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
         def event_rate(df):
             '''
             Compute empirical rate of death within 5 years
             for treated and untreated groups.

             Args:
                 df (dataframe): dataframe containing trial results.
                                 'TRTMT' column is 1 if patient was treated, 0 otherwise.
                                   'outcome' column is 1 if patient died within 5 years, 0 o

             Returns:
                 treated_prob (float64): empirical probability of death given treatment
                 untreated_prob (float64): empirical probability of death given control
             '''

             treated_prob = 0.0
             control_prob = 0.0

             ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###

             treated_prob = len(df[(df.TRTMT==True) & (df.outcome==1)])/len(df[df.TRTMT==True])
             control_prob = len(df[(df.TRTMT==False) & (df.outcome==1)])/len(df[df.TRTMT==Fals

             ### END CODE HERE ###

             return treated_prob, control_prob

In [16]: ### test cell ex2: you cannot edit this cell
         event_rate_test(event_rate)

Test Case 1:

Example df:
    outcome  TRTMT
0        0      1
1        1      1
2        1      1
3        0      1
4        1      0
5        1      0
6        1      0
```

```
7        0      0

Treated 5-year death rate: 0.5
Control 5-year death rate: 0.75

Error: Data-type mismatch. in variable 0, Got <class 'float'>  but expected type <class 'numpy
Error: Data-type mismatch. in variable 1, Got <class 'float'>  but expected type <class 'numpy
 2  Tests passed 1  Tests failed


        ---------------------------------------------------------------------------

        AssertionError                            Traceback (most recent call last)

        <ipython-input-16-fb106dbcb840> in <module>()
          1 ### test cell ex2: you cannot edit this cell
    ----> 2 event_rate_test(event_rate)


        ~/work/W1A1/public_tests.py in event_rate_test(target)
        102     ]
        103
    --> 104     multiple_test(test_cases, target)
        105
        106


        ~/work/W1A1/test_utils.py in multiple_test(test_cases, target)
        121         print('\033[92m', success," Tests passed")
        122         print('\033[91m', len(test_cases) - success, " Tests failed")
    --> 123         raise AssertionError("Not all tests were passed for {}. Check your equatio


        AssertionError: Not all tests were passed for event_rate. Check your equations and avo
```

Expected Output:

```
Test Case 1:

Example df:
    outcome  TRTMT
0        0      1
1        1      1
2        1      1
3        0      1
4        1      0
5        1      0
```

```
6        1      0
7        0      0

Treated 5-year death rate: 0.5
Control 5-year death rate: 0.75

 All tests passed.
```

Now let's try the function on the real data.

```
In [17]: treated_prob, control_prob = event_rate(data)

         print(f"Death rate for treated patients: {treated_prob:.4f} ~ {int(treated_prob*100)}%
         print(f"Death rate for untreated patients: {control_prob:.4f} ~ {int(control_prob*100)
```

```
Death rate for treated patients: 0.3725 ~ 37%
Death rate for untreated patients: 0.4822 ~ 48%
```

On average, it seemed like treatment had a positive effect.

Sanity checks It's important to compute these basic summary statistics as a sanity check for more complex models later on. If they strongly disagree with these robust summaries and there isn't a good reason, then there might be a bug.

1.3 Train Test Split

We'll now try to quantify the impact more precisely using statistical models. Before we get started fitting models to analyze the data, let's split it using the `train_test_split` function from `sklearn`. While a hold-out test set isn't required for logistic regression, it will be useful for comparing its performance to the ML models later on.

```
In [18]: # As usual, split into dev and test set
         from sklearn.model_selection import train_test_split
         np.random.seed(18)
         random.seed(1)

         data = data.dropna(axis=0)
         y = data.outcome
         # notice we are dropping a column here. Now our total columns will be 1 less than bef
         X = data.drop('outcome', axis=1)
         X_dev, X_test, y_dev, y_test = train_test_split(X, y, test_size = 0.25, random_state=

In [19]: print(f"dev set shape: {X_dev.shape}")
         print(f"test set shape: {X_test.shape}")
```

```
dev set shape: (455, 13)
test set shape: (152, 13)
```