Predicting Medicare Payments

James J. Heffers

University of Michigan - Ann Arbor

heffers@umich.edu

6/15/2022

Introduction

- Claims data contains a vast amount of billing records submitted by hospitals, physicians, and other sources for a wide breadth of services, such as physician visits, in/out patient procedures, days at a skilled nursing facility, prescriptions, and much more!
- We wish to explore Medicare costs for patients with chronic conditions and in particular create a predictive model for the payments Medicare will make for Plan D beneficiaries based on their age range, sex, and what chronic conditions they have.
- This presentation just shares methods, results, and insights. All processes/code can be found in the notebooks in the accompanying GitHub repository! There is also an accompanying R Shiny dashboard to share aggregated data for costs! Links are provided at end for convenience.

Methodology

- Data is collected from the Center for Medicare and Medicaid Services webpage (CMS.gov - link to data also provided at the end).
- Perform data mining to understand what features we have, and how best to clean and prepare the data.
- Fit models and tune hyperparameters to find a strong model.
- Deploy the model as a real time inference pipeline on Microsoft Azure Platform and use it to predict costs for a new beneficiary.

Understanding the data

- Upon loading the data to our notebook, we first check where the null data is, and how much there is (there are 55 columns in the data frame, the below image only shows a few of the entries).

```
df.isnull().sum()/df.shape[0]*100
```

```
BENE_SEX_IDENT_CD         0.000000
BENE_AGE_CAT_CD           0.000000
CC_ALZHDMTA               3.490433
CC_CANCER                 3.490433
CC_CHF                    0.000000
CC_CHRNKIDN               0.000000
CC_COPD                   3.490433
CC_DEPRESSN               3.490433
CC_DIABETES               0.000000
CC_ISCHMCHT               0.000000
CC_OSTEOPRS               3.490433
CC_RA_OA                  0.000000
CC_STRKETIA               3.490433
CC_2_OR_MORE              0.000000
DUAL_STUS                 0.000000
BENE_COUNT_PA_LT_12      59.710040
AVE_MO_EN_PA_LT_12       59.710040
AVE_PA_PAY_PA_LT_12      60.228151
AVE_IP_PAY_PA_LT_12      64.323047
AVE_SNF_PAY_PA_LT_12     70.231332
AVE_OTH_PAY_PA_LT_12     70.231332
AVE_IP_ADM_PA_LT_12      64.323047
AVE_SNF_DAYS_PA_LT_12    70.231332
BENE_COUNT_PA_EQ_12       7.580784
```

Understanding the data

- We see that the "less than 12 months" columns are missing vast amounts of data. However the number of beneficiaries in those situations is small, so in light of these two things we will drop those columns from our analysis. We also see that Plan C has almost no data (missing near 90%!) so we omit those columns next.
- The documentation for the CMS PUF data discusses "suppressed" data. To limit the number of categories (rows), sometimes conditions are suppressed (left blank) so that more people can fit into the category. We see that the suppressed data is "small" (about 3.5%), and suppressed conditions are in the same rows! So we can easily drop those rows, and that still leaves us with over $21,000$ rows.

Understanding the Data

- We have two more steps. First we use the describe() method to get information about the data frame statistics. We note the maximum value for plan D is staggering compared to the mean and standard deviation, so we wish to drop some of these extreme outliers.

```
df2['AVE_PDE_CST_PD_EQ_12'].describe()
```

✓ <1 sec

```
count    16699.000000
mean      5069.836457
std       2110.611758
min       1279.000000
25%       3534.000000
50%       4608.000000
75%       6200.500000
max      26641.000000
```

Understanding the Data

- We compute $\mu \pm 3\sigma$, and set these constraints on the Plan D costs column. This also drops any NaN rows in Plan D Costs column.
- The other columns are not going to be our target, so we will impute the missing values using SimpleImputer() with the mean method.

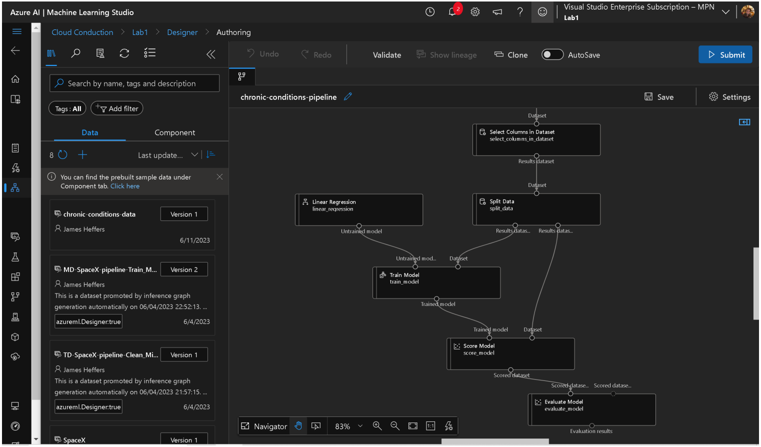
```
sup = df2['AVE_PDE_CST_PD_EQ_12'].mean() +3*df2['AVE_PDE_CST_PD_EQ_12'].std()
inf = df2['AVE_PDE_CST_PD_EQ_12'].mean() -3*df2['AVE_PDE_CST_PD_EQ_12'].std()

print(sup,inf)
```

```
11401.67173259739 -1261.9988180516084
```

Modeling

- First, as a sanity check, we run a quick designer pipeline in the Azure ML studio, using linear regression, and see what metrics are returned.



Modeling

- The MSE and RSquared are shown below, and it looks promising!

Mean_Absolute_Error	Root_Mean_Squared_Error	Relative_Squared_Error	Relative_Absolute_Error	Coefficient_of_Determ n
694.31518	913.229169	0.220509	0.44627	0.779491

Modeling

We will now revisit our notebook and run several models (e.g., Gradient Boosting, Random Forest, KNN) and tune hyperparameters. The below output shows the results. Gradient Boosting seems to be the strongest performing model. However we will do one more quick sanity check!

```python
results = pd.DataFrame([lin_reg,grad_boost,kn,dec_tree,rforest], columns=['model','R2','RMSE'])
results.sort_values('R2', ascending=False)
```

	model	R2	RMSE
1	Gradient Boosting	0.879200	683.969346
2	KNN	0.857604	742.595625
3	Decision Tree	0.840028	787.093527
4	Random Forest	0.803259	872.873524
0	Linear Reg	0.784961	912.562214

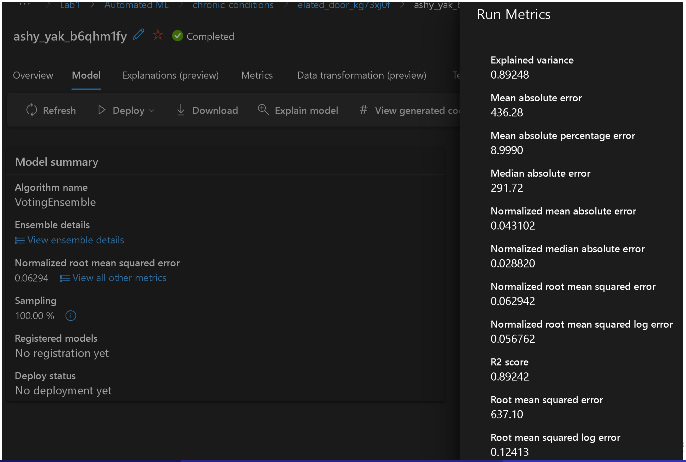
Modeling

- We feed the cleaned data set into Azure AML, and after the process runs, it will report what was found to be the best performing model along with the metrics and feature importance. We see the output is a Gradient Boosting model (AML puts the best model at the top)!

Duration	Hyperparameter	
1m 1s	algorithm : ['XGBoostRegressor', 'Li	...
1m 8s	algorithm : ['XGBoostRegressor', 'Li	...
23s	tree_method : auto	...
25s	min_data_in_leaf : 20	...

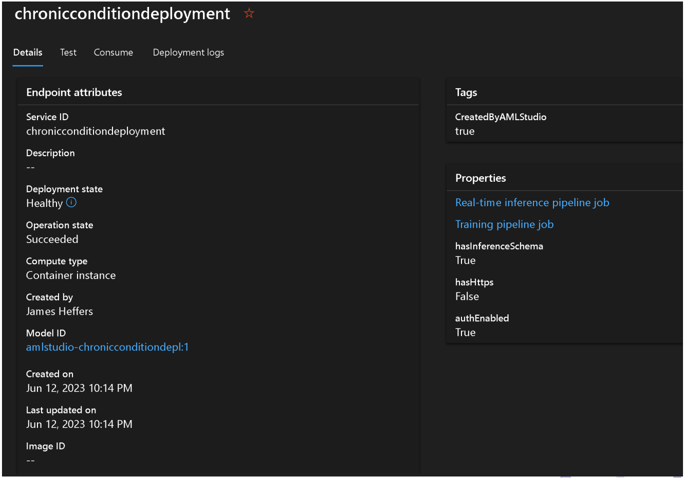
Modeling

The model provided by Azure AML also has metrics very close to what we found in our investigation. This reassures us that a Gradient Boosting model is the best performing model for this data. We now register and deploy the model.



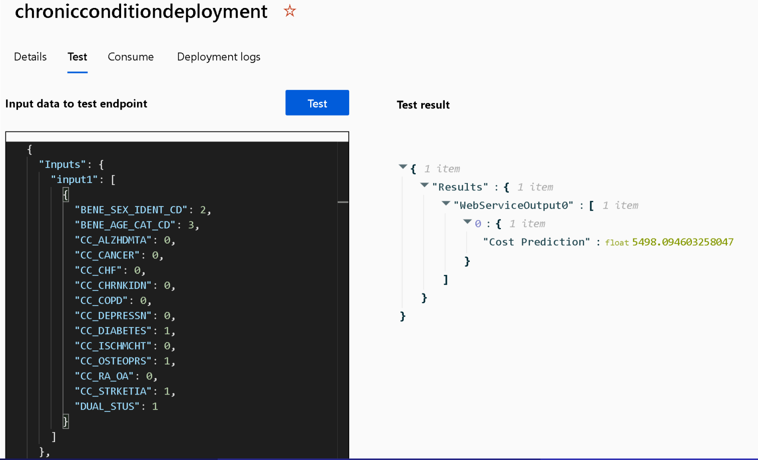
Deployment

For the purpose of this project, we will simply deploy it as a real-time inference pipeline in an Azure Container Instance (ACI). We can now feed it data on a new beneficiary and see the predicted cost!



Predicting

We now put in some test data: our test beneficiary is a female (Sex $= 2$) in the age range of 70-74 (Age $= 3$), with diabetes, osteoporosis, stroke, and is dual status eligible. We see that the prediction for the Medicare payment is roughly $\$5,500$.



The End!

Thanks for checking out my project! If you did not get this PDF from my GitHub, then the link to the corresponding repository containing all the code used is below. Additionally, you will also find the R Shiny code there, and the link to the online dashboard is provided as well!

GitHub: https://github.com/TheProfessor712/Claims-data-model

R Shiny: https://theprofessor712.shinyapps.io/RShinyDash/

Source

The data for this project was pulled from CMS.gov. The link is below!

Data Source:
https://www.cms.gov/Research-Statistics-Data-and-Systems/Downloadable-Public-Use-Files/BSAPUFS/Chronic_Conditions_PUF