Predicting Patient Readmission with Logistic Regression

August 30, 2021

1 Research Question

We will build a model for predicting whether or not a patient will be readmitted to the hospital within 30 days of their release. We will look at patient demographics and health conditions to make this prediction.

2 Methodology

Logistic Regression is the method of choice here because we are trying to predict a binary outcome: either "yes" or "no" (1 or 0).

Logistic Regression relies on the following assumptions: - The dependent variable is in fact binary - All of the observations are independent - There is no Multicollinearity among explanatory variables - There are no extreme outlisers - There is a Linear Relationship Between Explanatory Variables and the Logit of the Response Variable - The sample size is sufficiently large

Source

2.1 Tools

Python in Jupyterlab was used to write the code for this analysis. There are many statistics libraries written in python that make what we are planning to do very simple: - `Numpy` and `pandas` for standard dataframe and numerical operations - `Sklearn` and `statsmodels` for building the regression models - `matplotlib`, `yellowbrick`, and `sns` for visualizations.

[1]:
```
%%capture

# Install Packages
!pip install numpy
!pip install pandas
!pip install matplotlib
!pip install sklearn
!pip install statsmodels
!pip install yellowbrick
!pip install seaborn
```

[2]:
```
# Import Packages
import pandas as pd
```

```
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
from sklearn.metrics import confusion_matrix




import random

random.seed(10)
```

3 Data Preparation

We start with a clean dataset of 10,000 observations. We're going to do some additional cleaning to optimize it for our purposes.

[3]:
```
df = pd.read_csv("data/medical_clean.csv")
df.head()
```

[3]:

	CaseOrder	Customer_id	Interaction	UID	City	State	County	Zip	Lat	Lng	…	TotalCharge	Additional_charges	Item1	Item2	Item3
0	1	C412403	8cd49b13-f45a-4b47-a2bd-173ffa932c2f	3a83ddb66e2ae73798bdf1d705dc0932	Eva	AL	Morgan	35621	34.34960	-86.72508	…	3726.702860	17939.403420	3	3	2
1	2	Z919181	d2450b70-0337-4406-bdbb-bc1037f1734c	176354c5eef714957d486009feabf195	Marianna	FL	Jackson	32446	30.84513	-85.22907	…	4193.190458	17612.998120	3	4	3
2	3	F995323	a2057123-abf5-4a2c-abad-8ffe33512562	e19a0fa00aeda885b8a436757e889bc9	Sioux Falls	SD	Minnehaha	57110	43.54321	-96.63772	…	2434.234222	17505.192460	2	4	4
3	4	A879973	1dec528d-eb34-4079-adce-0d7a40e82205	cd17d7b6d152cb6f23957346d11c3f07	New Richland	MN	Waseca	56072	43.89744	-93.51479	…	2127.830423	12993.437350	3	5	5
4	5	C544523	5885f56b-d6da-43a3-8760-83583af94266	d2f0425877b10ed6bb381f3e2579424a	West Point	VA	King William	23181	37.59894	-76.88958	…	2113.073274	3716.525786	2	1	3

```
   Item4  Item5 Item6 Item7 Item8
0      2      4     3     3     4
1      4      4     4     3     3
2      4      3     4     3     3
3      3      4     5     5     5
4      3      5     3     4     3

[5 rows x 50 columns]
```

3.1 Drop Irrelavant Data

We begin the data preparation by removing irrelevant variables. We removed the following: - CaseOrder - Customer_id - Interaction - UID - City - State - County - Zip - Lat - Lng - Interaction - TimeZone - Additional_charges - Job - Item1 - Item2 - Item3 - Item4 - Item5 - Item6 - Item7 - Item8 - TotalCharge

[4]:
```
df = df.drop(['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'City',
              'State', 'County', 'Zip', 'Lat', 'Lng', 'Interaction', 'TimeZone',
               'Additional_charges', 'Job', 'Item1', 'Item2', 'Item3', 'Item4',
              'Item5', 'Item6', 'Item7', 'Item8', 'TotalCharge'], axis=1)
```

3.2 Make all data numeric

Logistic Regression requires numeric values to work, we will modify the data with dummy variables to accomodate this.

[5]:
```
readmis = df['ReAdmis']

df = pd.get_dummies(df)

# We want ReAdmis, our target variable, to be a single column
df = df.drop(['ReAdmis_No', 'ReAdmis_Yes'], axis=1)
df = df.assign(ReAdmis=readmis)
df['ReAdmis'] = pd.Categorical(df['ReAdmis'])
df['ReAdmis'] = df['ReAdmis'].cat.codes
```

3.3 Splitting Data

We'll first divide the data into our target `y`, `ReAdmis`, and our predictors `X`.

Feature	Type
Population	numeric
Area	categorical
Children	numeric
Age	numeric
Income	numeric
Marital	categorical
Gender	categorical

Feature	Type
VitD_levels	numeric
Doc_visits	numeric
Full_meals_eaten	numeric
vitD_supp	numeric
Soft_drink	categorical
Initial_admin	categorical
HighBlood	categorical
Stroke	categorical
Complication_risk	categorical
Overweight	categorical
Arthritis	categorical
Diabetes	categorical
Hyperlipidemia	categorical
BackPain	categorical
Anxiety	categorical
Allergic_rhinitis	categorical
Reflux_esophagitis	categorical
Asthma	categorical
Services	categorical
Initial_days	numeric

All data labled "categorical" has of course been split into dummy variables.

[6]:
```
X = df.loc[:,df.columns!='ReAdmis']
y = df['ReAdmis']
```

We further split the data into training and testing data. We'll also apply a scaler.

[7]:
```
scaler = preprocessing.StandardScaler().fit(X)
scaled = scaler.transform(X)

X_train, X_test, y_train, y_test = train_test_split(scaled, y, test_size=0.2,␣
 ↪random_state=42)
```

4 Models

We'll start with an initial model using all of the predictors. Then we'll reduce the predictors and compare the reduced models to the initial one.

4.1 Initial Model

[8]:
```
initial_model = LogisticRegression(random_state=0).fit(X_train, y_train)
initial_model.predict(X_test)
initial_score = initial_model.score(X_test, y_test)
```

4.2 Data Reduction

We'll use Recursive Feature Elimination to reduce the data.

[9]:
```
selector = RFE(initial_model, n_features_to_select=10, step=3)
selector = selector.fit(X_train, y_train)

best_variables = []
sort = selector.ranking_.sort()

for i, val in enumerate(selector.ranking_):
    if val == 1:
        best_variables.append(X.columns[i-1])

print("\n\nSelected reduced features:")
for b in best_variables:
    print('- %s' % b)
```

```
Selected reduced features:
- Services_MRI
- Population
- Children
- Age
- Income
- VitD_levels
- Doc_visits
- Full_meals_eaten
- vitD_supp
- Initial_days
```

4.3 Reduced Model

Now we rebuild the model with the reduces list of variables from the previous section.

[10]:
```
X_reduced = df[best_variables]

scaler = preprocessing.StandardScaler().fit(X_reduced)
scaled = scaler.transform(X_reduced)

X_train, X_test, y_train, y_test = train_test_split(scaled, y, test_size=0.2)
reduced_model = LogisticRegression(random_state=0).fit(X_train, y_train)
reduced_model.predict(X_test)
reduced_score = reduced_model.score(X_test, y_test)
```

Using only the five selected variables we were able to create a model that is only .003% worse than the full model, showing the impact of just these 5 variables.

5 Results Summary

5.1 Model Accuracy

We will look at the scores of the initial and reduced models and compare them.

[11]:
```
print('Initial model\'s score: %.3f, ' % initial_score)
print('Reduced model\'s score: %.3f, ' % reduced_score)
print('Difference: %.3f' % (reduced_score - initial_score))

print('\nCoefficients:')
for i, c in enumerate(reduced_model.coef_[0]):
    print('- %s: %.3f' % (X_reduced.columns[i], c))
```

```
Initial model's score: 0.983,
Reduced model's score: 0.979,
Difference: -0.004

Coefficients:
- Services_MRI: 0.234
- Population: 0.110
- Children: 0.104
- Age: 0.015
- Income: -0.031
- VitD_levels: 0.064
- Doc_visits: 0.043
- Full_meals_eaten: 0.053
- vitD_supp: 0.001
- Initial_days: 14.723
```

5.2 Confusion Matrix

We will look at the confusion matrix to get a better idea of the model's accuracy.

[12]:
```
confusion_matrix(y_test, reduced_model.predict(X_test))
```

[12]:
```
array([[1248,   24],
       [  18,  710]])
```

Here is a more human-readable table:

	Predicted Positive	Predicted Negative
Actual Positive	1209	22
Actual Negative	19	750

1959/2000 Of the test observations were true positives/negatives. Or 98%

5.3 Statistical Significance

We were able to get within 4 thousandths of the accuracy of the full model with only the 10 variables we reduced to, an inconsequential change. With this information we now know that if a hospital wants to estimate the total cost of a visit they need only look at the reduced variables such as MRI services and Initial Days.

5.4 Limitations

With only 10,000 patient records to work with there is a chance this model has been biased towards a particular conclusion. For example in another set of patients a different pre-existing condition such as Anxiety or a demographic attribute such as Gender may contribute more to ReAdmission.

5.5 Recommendations

The biggest contributer to readmission according to our model is the initial length of a patient's visit. Therefore to prevent readmissions hospitals should pay extra attention to patients that have already spent considerable time in the hospital.