The Medication Therapy Management (MTM) Study

Thomas Arnold, PhD

For the IBM Machine Learning Course 3

The following study provides a test of which classifier produces the best accuracy when trying to identify which patients received Medication Therapy Management. It appeared the Random Forest Classifier was the most accurate.

Contents

Overview

This study is an attempt to better understand the predictors of Medication Therapy Management (MTM). MTM is a treatment provided to patients who have complex prescription needs. With MTM, a pharmacist is assigned to review the patient's medications and make recommendations.

As part of my work as a senior population health data scientist, I had been assigned to evaluate the efficacy of MTM. I used propensity score matching to find a matched sample that was similar to the patients who had been assigned to a pharmacist for MTM. In comparing the MTM sample with the matched sample, it seemed that the MTM patients were doing better.

The variables I had used for matching to the MTM patients were a combination of 1) variables that I thought were important, and 2) variables that seemed to be significantly related to MTM status.

I wanted to know more about these variables and their relationship to MTM status. Since the data and application seemed to be applicable to a machine learning application, I decided to use it. The net effect is that this kills two birds with one stone. I learn about my work data and get my IBM ML homework done.

Main objective of this analysis

My main objective was to develop a better understanding of the relationship between the variables I had used in the MTM evaluation study. In the MTM study, I had used propensity score matching to find a matched sample that was similar to the MTM patients. The matched sample was matched to MTM patients using 18 variables. The main objective of this analysis is to learn more about the relationship between these 18 variables. While the initial goal is primarily related to interpretation, further analysis might also help predict which patients might benefit from MTM.

Description of the data

The data used in these analyses was pulled from the EPIC Electronic Health Records (EHR) system. There were 2,025 patients who had MTM in one year. I wanted to match these patients with similar patients. To do the matching, I first chose a number of variables that I thought might be relevant. For example, I added the number of medications and number of different providers who had seen the patient in the past year. I then used the variables in a bigger analytics data set to pull predictors of MTM status. The resulting 18 variables were used in matching the MTM patients to similar patients using the R Matchit package.

The Matchit variables are listed below, along with brief descriptions.

Matchit Variables

Variable	Range	Description
Age	0-107	Age in years
Male	0-1	1 if Sex = Male
Married	0-5	1 = < 18, 2=Singl. 3=Marrd., Partnr, 4=Seprtd. Divrcd., 5=Widwd
M1YPmnts	0-$643,575	Payments for medical care in previous year
AvgMedicationsTestYear	0-65	Average medications in the test year
AvgProvidersTestYear	0-20	Average providers in the test year
HCCCount	0-19	Total Number of HCCs
HasPCP	0-1	1 = Has a Primary Care Provider Assigned in Epic
BTM5D	0-1	BETOS M5D - Specialist - other
CCS0257	0-1	Condition Coding System (CCS) - Other aftercare
BTM1A	0-1	BETOS M1A - Office visits - new
BPSTDays	0-4,696	Total days between first and last office visit
A1CMax	0-21	Maximum A1C in test year
BMIMax	0-100	Maximum BMI in test year
BPSMax	122-290	Maximum Systolic blood pressure in test year
BPDMax	76-195	Maximum diastolic blood pressure in test year
CHLMax	0-240	Maximum cholesterol in test year
PHQMax	0-27	Maximum PHQ9 in test year

Feature Engineering

Since the data had been used previously, it was fairly clean already. The "Married" variable was categorical, so that was converted to a set of dummy variables. I used the MinMax Scaler to convert the variables to variables ranging from 0-1. The resulting dataset is shown below.

MinMax Scaled Matchit Variables

Variable	count	mean	std	min	25%	50%	75%	max
Age	407,356	.391	.232	.000	.187	.393	.589	1.000
Male	407,356	.465	.499	.000	.000	.000	1.000	1.000
M1YPmnts	407,356	.004	.016	.000	.000	.001	.002	1.000
AvgMedicationsTestYear	407,356	.013	.027	.000	.000	.004	.014	1.000
AvgProvidersTestYear	407,356	.016	.025	.000	.000	.008	.021	1.000
HCCCount	407,356	.024	.057	.000	.000	.000	.000	1.000
HasPCP	407,356	.803	.397	.000	1.000	1.000	1.000	1.000
BTM5D	407,356	.074	.262	.000	.000	.000	.000	1.000
CCS0257	407,356	.202	.402	.000	.000	.000	.000	1.000
BTM1A	407,356	.119	.324	.000	.000	.000	.000	1.000
BPSTDays	407,356	.436	.311	.000	.157	.412	.706	1.000
A1CMax	407,356	.064	.135	.000	.000	.000	.000	1.000
BMIMax	407,356	.279	.113	.000	.219	.283	.341	1.000
BPSMax	407,356	.455	.139	.000	.421	.476	.531	1.000
BPDMax	407,356	.410	.125	.000	.390	.431	.472	1.000
CHLMax	407,356	.190	.218	.000	.000	.000	.402	1.000
PHQMax	407,356	.123	.216	.000	.000	.000	.148	1.000
Under18	407,356	.218	.413	.000	.000	.000	.000	1.000
Single	407,356	.442	.497	.000	.000	.000	1.000	1.000
Separated	407,356	.053	.224	.000	.000	.000	.000	1.000
Widowed	407,356	.045	.208	.000	.000	.000	.000	1.000
MTMPatient	407,356	.005	.070	.000	.000	.000	.000	1.000

Descriptive Statistics

The MTM group had higher average values for all of the measures except “Under 18”.

Descriptive Statistics

	MTM			Non-MTM			
Variable	**N**	**Mean**	**StdDev**	**N**	**Mean**	**StdDev**	**Difference**
Age	2,025	.553	.152	405,331	.390	.232	.163
Male	2,025	.468	.499	405,331	.465	.499	.003
M1YPmnts	2,025	.016	.032	405,331	.004	.016	.012
AvgMedicationsTestYear	2,025	.068	.073	405,331	.013	.026	.055
AvgProvidersTestYear	2,025	.075	.063	405,331	.016	.025	.059
HCCCount	2,025	.137	.129	405,331	.024	.056	.113
HasPCP	2,025	.948	.222	405,331	.803	.398	.145
BTM5D	2,025	.700	.458	405,331	.071	.257	.629
CCS0257	2,025	.874	.332	405,331	.199	.399	.675
BTM1A	2,025	.364	.481	405,331	.118	.323	.246
BPSTDays	2,025	.693	.270	405,331	.435	.311	.258
A1CMax	2,025	.313	.215	405,331	.062	.134	.251
BMIMax	2,025	.366	.097	405,331	.278	.113	.088
BPSMax	2,025	.561	.072	405,331	.454	.139	.107
BPDMax	2,025	.488	.058	405,331	.410	.125	.078
CHLMax	2,025	.380	.173	405,331	.189	.218	.191
PHQMax	2,025	.321	.297	405,331	.122	.215	.199
Under18	2,025	.007	.083	405,331	.219	.413	-.212
Single	2,025	.569	.495	405,331	.441	.497	.128
Separated	2,025	.114	.317	405,331	.053	.223	.061
Widowed	2,025	.090	.287	405,331	.045	.208	.045

Balancing the Data

As you can see from the Counts in the descriptive statistics, the outcome variable MTMPatient was highly unbalanced with only .5% of all patients having an MTMPatient value of 1 (This is 2,025 patients out of 407,356). As was noted in the last section of the IBM ML Class3 presentations, unbalanced data is not optimal for analysis.

To balance the data, I used a combination of upsampling the 2,025 MTM patients with replacement to 5,000 records, and downsampling the rest of the patient records without replacement to 5,000 records. This created a dataframe with 10,000 records, 5,000 MTM and 5,000 non-MTM.

Python Code for Balancing

```
from sklearn.utils import resample

# Separate majority and minority classes
df_majority = MTMStudydf[MTMStudydf.MTMPatient == 0]
df_minority = MTMStudydf[MTMStudydf.MTMPatient == 1]

# Upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace = True,
                                 n_samples = 5000,
                                 random_state = 123)

# Downsample minority class
df_majority_downsampled = resample(df_majority,
                                   replace = False,
                                   n_samples = 5000,
                                   random_state = 123)

MTMBalanceddf = pd.concat([df_minority_upsampled, df_majority_downsampled])
round(MTMBalanceddf.describe().T,3)
```

The balancing code created a dataframe with a 50/50 split on MTMPatient.and 10,000 records. See below.

Balanced Dataframe

	count	mean	std	min	25%	50%	75%	max
Age	10,000	.473	.212	.000	.318	.514	.636	.991
Male	10,000	.465	.499	.000	.000	.000	1.000	1.000
M1YPmnts	10,000	.010	.027	.000	.000	.002	.006	.440
AvgMedicationsTestYear	10,000	.040	.062	.000	.003	.018	.050	.739
AvgProvidersTestYear	10,000	.045	.056	.000	.008	.025	.062	.533
HCCCount	10,000	.080	.115	.000	.000	.053	.105	1.000
HasPCP	10,000	.875	.331	.000	1.000	1.000	1.000	1.000
BTM5D	10,000	.381	.486	.000	.000	.000	1.000	1.000
CCS0257	10,000	.532	.499	.000	.000	1.000	1.000	1.000
BTM1A	10,000	.242	.428	.000	.000	.000	.000	1.000
BPSTDays	10,000	.563	.318	.000	.319	.590	.877	.988
A1CMax	10,000	.191	.219	.000	.000	.000	.362	.947
BMIMax	10,000	.322	.113	.000	.263	.320	.385	.998
BPSMax	10,000	.508	.122	.000	.462	.517	.579	.890
BPDMax	10,000	.449	.104	.000	.421	.462	.503	.790
CHLMax	10,000	.284	.219	.000	.000	.362	.448	1.000
PHQMax	10,000	.221	.276	.000	.000	.111	.370	1.000
Under18	10,000	.110	.313	.000	.000	.000	.000	1.000
Single	10,000	.507	.500	.000	.000	1.000	1.000	1.000
Separated	10,000	.082	.274	.000	.000	.000	.000	1.000
Widowed	10,000	.068	.253	.000	.000	.000	.000	1.000
MTMPatient	10,000	.500	.500	.000	.000	.500	1.000	1.000

Resetting the Index

I found that the index in the balanced dataset was causing problems with variable selection in the train/test split because it contained duplicate MTMPatient records. I reset the index to fix this. I had to remove the [index] column that was created.

Python code for Resetting the Index

```
MTMBalanceddf.reset_index(inplace=True)
MTMBalanceddf = MTMBalanceddf.drop(['index'], axis=1)
```

Correlation Matrix and Heatmap

I created a correlation matrix and heatmap to observe how the variables were related.

Python Code for Correlation Matrix and Heatmap

```
# Make a list of the column names
study_columns = MTMStudydf.columns

# Create a correlation matrix and copy to clipboard for transfer to Excel
MTMCorrelationdf = MTMBalanceddf[study_columns].corr()
MTMCorrelationdf.to_clipboard(excel = True, index = True)=
print('Paste the data in Excel')

# Plot the heatmap
fig, ax = plt.subplots(figsize = (15,10))
sns.heatmap(MTMCorrelationdf, cmap=colors)
```

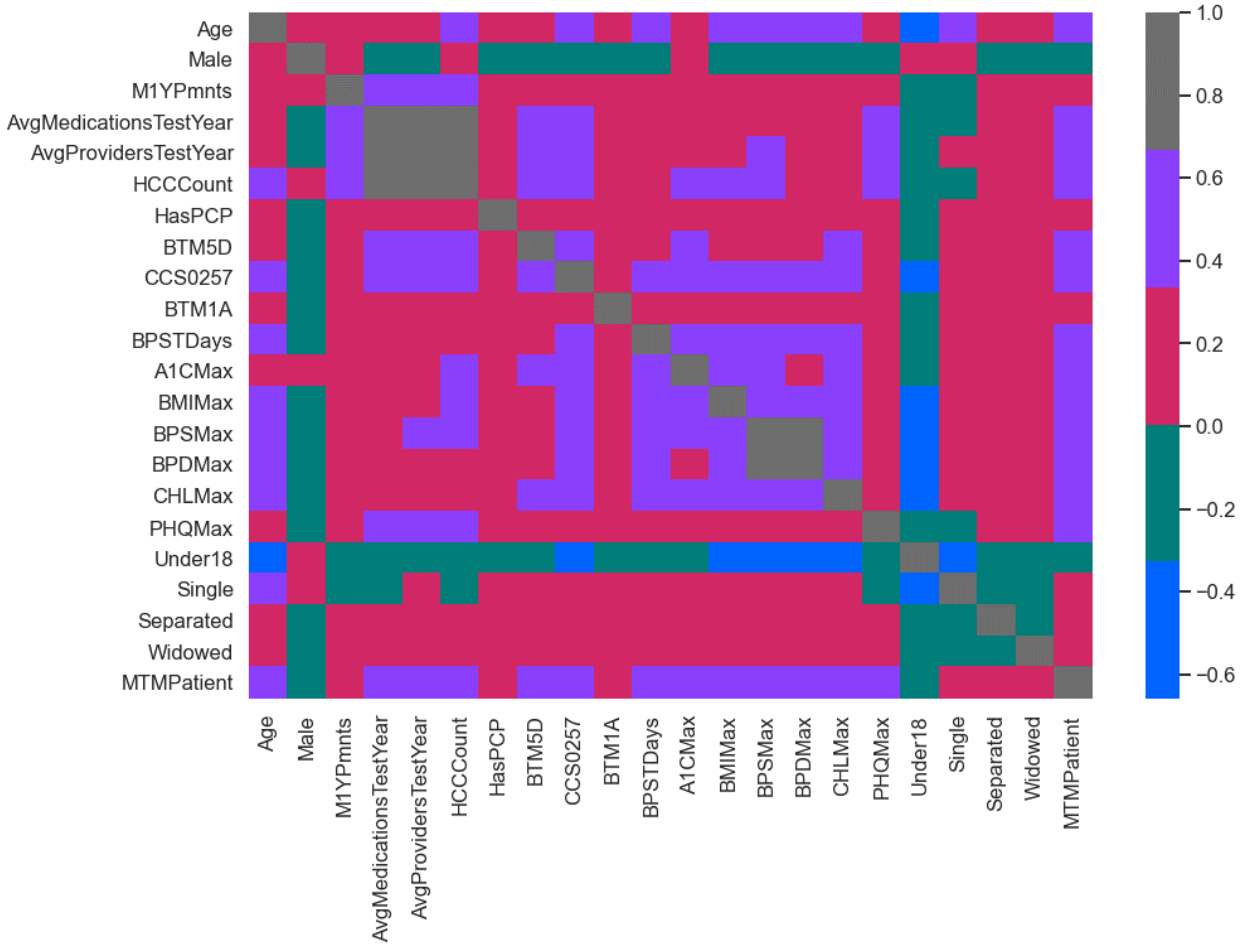
The correlation matrix and heatmap output are shown below. The variables AvgMedicationsTestYear, AvgProvidersTestYear, and HCCCount had some of the highest correlations. An HCC is a Centers for Medicare and Medicaid (CMS) measure of chronic conditions. It is not surprizing that this would be correlated with medications and providers.

Note the BPDMax and BPSMax are also highly correlated. BPD is the abbreviation used for diastolic blood pressure and BPS is systolic blood pressure. Therefore, it is not surprising that the Max values of these two blood pressure measures are highly correlated.

Correlation Matrix

#	Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	Age	1	.01	.17	**.26**	**.28**	**.35**	.18	.29	.43	.16	.34	.34	.34	.59	**.42**	.51	.12	-.66	.36	.10	.31	.37
2	Male	.01	1	.04	**-.04**	**-.06**	**.03**	-.07	-.03	.00	-.04	-.09	.03	-.06	.00	**.00**	-.07	-.14	.03	.06	-.03	-.16	-.01
3	M1YPmnts	.17	.04	1	**.53**	**.51**	**.55**	.07	.23	.30	.19	.11	.15	.15	.20	**.17**	.10	.16	-.11	.00	.05	.09	.23
4	AvgMedicationsTestYear	**.26**	**-.04**	**.53**	**1**	**.92**	**.69**	.14	.36	.44	.29	.25	.29	.29	.34	**.29**	.24	.36	-.20	-.01	.13	.11	.44
5	AvgProvidersTestYear	**.28**	**-.06**	**.51**	**.92**	**1**	**.71**	.18	.44	.51	.33	.31	.32	.33	.38	**.33**	.29	.45	-.23	.01	.13	.11	.52
6	HCCCount	**.35**	**.03**	**.55**	**.69**	**.71**	**1**	.15	.41	.52	.26	.31	.40	.36	.39	**.30**	.29	.36	-.22	.00	.12	.15	.49
7	HasPCP	.18	-.07	.07	.14	.18	.15	1	.18	.20	.12	.19	.15	.18	.16	**.12**	.24	.14	-.03	.07	.04	.08	.21
8	BTM5D	.29	-.03	.23	.36	.44	.41	.18	1	.54	.27	.32	.39	.26	.31	**.24**	.34	.30	-.20	.09	.07	.09	.63
9	CCS0257	.43	.00	.30	.44	.51	.52	.20	.54	1	.31	.38	.50	.36	.44	**.36**	.41	.33	-.33	.13	.10	.12	.67
10	BTM1A	.16	-.04	.19	.29	.33	.26	.12	.27	.31	1	.14	.08	.15	.18	**.17**	.13	.23	-.14	.04	.09	.04	.30
11	BPSTDays	.34	-.09	.11	.25	.31	.31	.19	.32	.38	.14	1	.37	.39	.49	**.45**	.55	.31	-.31	.17	.07	.08	.40
12	A1CMax	.34	.03	.15	.29	.32	.40	.15	.39	.50	.08	.37	1	.39	.40	**.31**	.51	.22	-.29	.12	.06	.09	.56
13	BMIMax	.34	-.06	.15	.29	.33	.36	.18	.26	.36	.15	.39	.39	1	.53	**.50**	.40	.31	-.41	.15	.13	.05	.37
14	BPSMax	.59	.00	.20	.34	.38	.39	.16	.31	.44	.18	.49	.40	.53	1	**.87**	.48	.30	-.56	.21	.11	.15	.43
15	BPDMax	**.42**	**.00**	**.17**	**.29**	**.33**	**.30**	**.12**	**.24**	**.36**	**.17**	**.45**	**.31**	**.50**	**.87**	1	.42	.32	-.54	.18	.11	.06	.37
16	CHLMax	.51	-.07	.10	.24	.29	.29	.24	.34	.41	.13	.55	.51	.40	.48	.42	1	.30	-.42	.24	.10	.12	.43
17	PHQMax	.12	-.14	.16	.36	.45	.36	.14	.30	.33	.23	.31	.22	.31	.30	.32	.30	1	-.23	-.05	.19	.02	.36
18	Under18	-.66	.03	-.11	-.20	-.23	-.22	-.03	-.20	-.33	-.14	-.31	-.29	-.41	-.56	-.54	-.42	-.23	1	-.36	-.11	-.10	-.32
19	Single	.36	.06	.00	-.01	.01	.00	.07	.09	.13	.04	.17	.12	.15	.21	.18	.24	-.05	-.36	1	-.30	-.28	.13
20	Separated	.10	-.03	.05	.13	.13	.12	.04	.07	.10	.09	.07	.06	.13	.11	.11	.10	.19	-.11	-.30	1	-.08	.09
21	Widowed	.31	-.16	.09	.11	.11	.15	.08	.09	.12	.04	.08	.09	.05	.15	.06	.12	.02	-.10	-.28	-.08	1	.10
22	MTMPatient	.37	-.01	.23	.44	.52	.49	.21	.63	.67	.30	.40	.56	.37	.43	.37	.43	.36	-.32	.13	.09	.10	1

Correlation Heatmap



StatsModel Logistic Regression Output

The first step in the evaluation was to obtain the regression output. Since the statsmodel regression package provided p values, I used the statsmodel Logit regression output to look at feature importance. Recall that the MTMPatient variable was dichotomous, requiring Logit regression.

StatsModel Logit Python Code

```
# Set up column lists
target = 'MTMPatient'
feature_columns = [x for x in MTMStudydf.columns if x != target]

# Run statsmodel using Logit
import statsmodels.api as sm
X_trainsm = sm.add_constant(MTMBalanceddf[feature_columns])
est = sm.Logit(MTMBalanceddf[target], X_trainsm)
smfit = est.fit()
smfitdf =
pd.DataFrame(pd.read_html(smfit.summary().tables[1].as_html(),header=0,index_col=0)[0])

# Copy statsmodel output to Excel via the clipboard
smfitdf.to_clipboard(excel = True, index = True)
print('Copy the data to Excel')
```

The following output table for the model predicting MTMPatient is sorted by t value. BTM5D (Saw a specialist), CCS0257 (Received other aftercare), A1CMax (Blood sugar test indicating probability of diabetes), and AvgProvidersTestYear were the best predictors of being seen by a pharmacist.

StatsModel Logit Regression Output

Variables	B	SE	z	p
BTM5D	2.289	.084	27.381	.000
A1CMax	4.445	.204	21.750	.000
CCS0257	1.144	.080	14.297	.000
AvgProvidersTestYear	26.974	2.316	11.647	.000
BTM1A	.571	.089	6.431	.000
HCCCount	2.420	.572	4.233	.000
HasPCP	.502	.125	4.021	.000
BPDMax	3.048	.826	3.691	.000
Single	.281	.096	2.926	.003
BPSTDays	.362	.139	2.600	.009
Widowed	.421	.171	2.467	.014
Male	.163	.074	2.199	.028
PHQMax	.308	.158	1.946	.052
BMIMax	.277	.400	.692	.489
Separated	.093	.153	.605	.545
BPSMax	-.210	.747	-.282	.778
CHLMax	-.185	.222	-.831	.406
AvgMedicationsTestYear	-3.686	1.968	-1.873	.061
Age	-1.483	.306	-4.843	.000
Under18	-1.171	.222	-5.276	.000
M1YPmnts	-13.419	1.827	-7.344	.000
Constant	-4.761	.322	-14.762	.000

A Test of Classifiers

I tried the various classifiers used in the course to see which provided better results. The code for these tests along with outputs are provided below.

- KNN
- Random Forest
- Extra Trees
- Logistic Regression
- Gradient Boosting
- Voting Classifier

Step 1: Getting a Train/Test split

The first step was getting a train/test split. The code used was as follows.

Python Code for Train/Test Split

```
# Load relevant modules
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, f1_score

# Set up X and y variables
y, X = MTMBalanceddf['MTMPatient'], MTMBalanceddf.drop(columns='MTMPatient')

# Split the data into training and test samples
X_train, X_test, y_train, y_test = train_test_split(Xknn, y, test_size=0.4, random_state=42)
```

Step 2: Setting Up a Results Output Dataframe

I set up an Output dataframe so that I could compare results. I created a "LoadStats" function to load the models. Example is for KNN model with K=7.

```
StatsColumnNames = ["Model", "Notes", "MTMPatient", "Precision", "Recall", "F1 Score", "TP",
"FP"]
CummulativeStatsdf = pd.DataFrame(columns = StatsColumnNames)

def LoadStats (Statsdf, Model, Notes, Modeldf, Confusiondf):
        Statsdf.loc[len(Statsdf.index)] = [Model, Notes, 0, Modeldf.iloc[0,0],
                Modeldf.iloc[0,1], Modeldf.iloc[0,2],
                Confusiondf.iloc[0,0], Confusiondf.iloc[0,1]]
        Statsdf.loc[len(Statsdf.index)] = [Model, Notes, 1, Modeldf.iloc[1,0],
                Modeldf.iloc[1,1], Modeldf.iloc[1,2],
                Confusiondf.iloc[1,1], Confusiondf.iloc[1,0]]

# Example
StatsOutdf = pd.DataFrame(classification_report(y_test, y_pred, output_dict=True)).transpose()
Confusiondf = pd.DataFrame(confusion_matrix(y_test, y_pred))
LoadStats(CummulativeStatsdf, 'KNN', "Neighbors = 7", StatsOutdf,Confusiondf)
```

K Nearest Neighbors

I tried a number of K Nearest Neighbors models with K = 1, K = 5, etc. I then used a loop to find the best F1 Score with the lowest Error Rate. Form this, I determined that K=7 produced the best result.

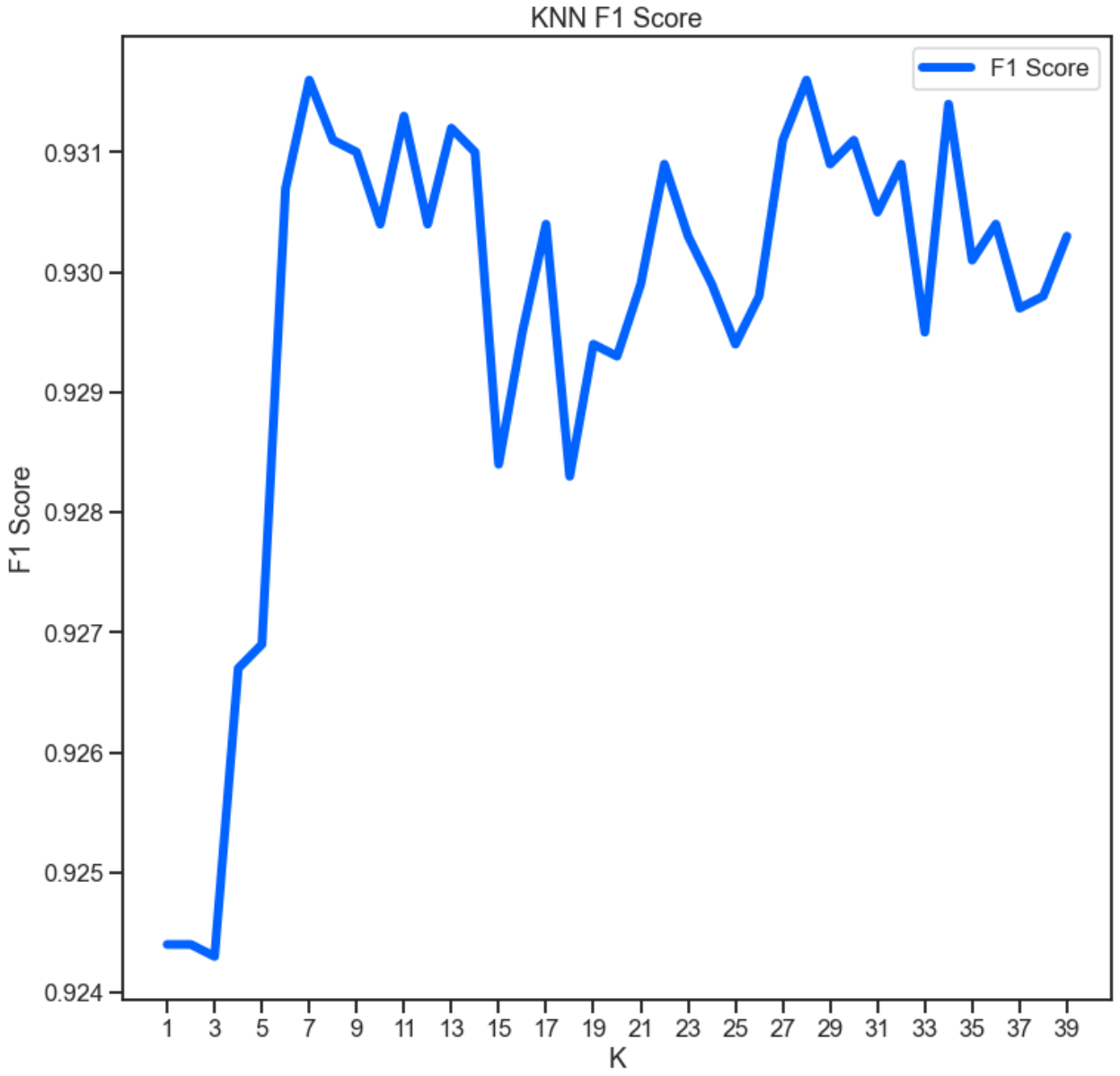
Looking for the Best KNN Model

The following code was used to test KNN models.

```
max_k = 40
f1_scores = list()
error_rates = list() # 1-accuracy

for k in range(1, max_k):
      knn = KNeighborsClassifier(n_neighbors=k, weights='distance')
   knn = knn.fit(X_train, y_train)
      y_pred = knn.predict(X_test)
   f1 = f1_score(y_pred, y_test)
   f1_scores.append((k, round(f1_score(y_test, y_pred), 4)))
   error = 1-round(accuracy_score(y_test, y_pred), 4)
   error_rates.append((k, error))

f1_results = pd.DataFrame(f1_scores, columns=['K', 'F1 Score'])
error_results = pd.DataFrame(error_rates, columns=['K', 'Error Rate'])
results = f1_results
results["Error Rate"] = error_results.iloc[:,1]
results
```

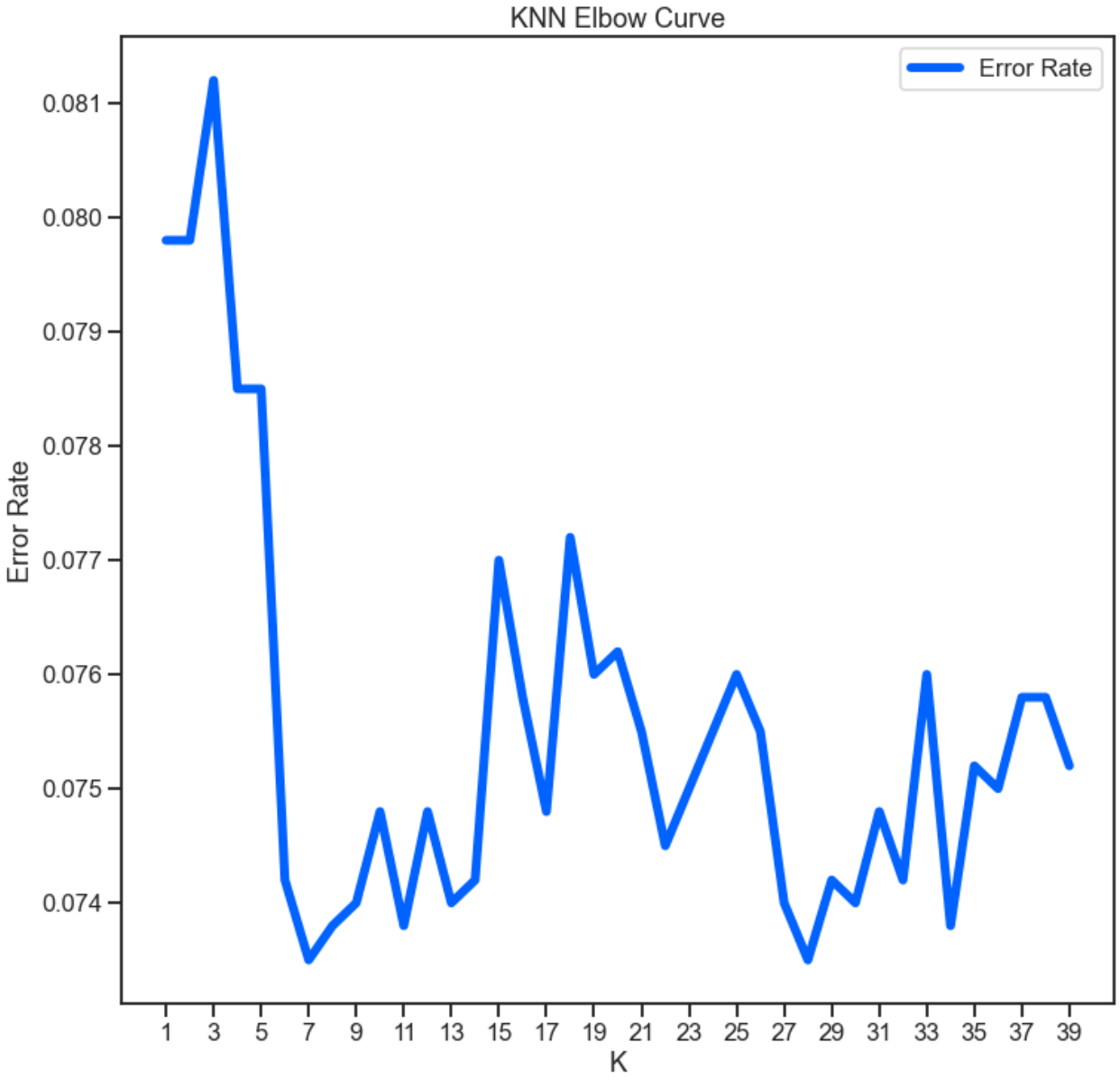
Plotting the F1 Score by Number of neighbors

```
# Plot F1 results
sns.set_context('talk')
sns.set_style('ticks')

plt.figure(dpi=300)
ax = f1_results.set_index('K').plot(color=colors[0], figsize=(12, 12), linewidth=6)
ax.set(xlabel='K', ylabel='F1 Score')
ax.set_xticks(range(1, max_k, 2));
plt.title('KNN F1 Score')
plt.savefig('knn_f1.png')
```



Plotting the Error Rate by nearest neighbors

```
# Plot Accuracy (Error Rate) results
sns.set_context('talk')
sns.set_style('ticks')

plt.figure(dpi=300)
ax = error_results.set_index('K').plot(color=colors[0], figsize=(12, 12), linewidth=6)
ax.set(xlabel='K', ylabel='Error Rate')
ax.set_xticks(range(1, max_k, 2))
plt.title('KNN Elbow Curve')
plt.savefig('knn_elbow.png')
```



Running KNN Model with K = 7

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, f1_score

knn = KNeighborsClassifier(n_neighbors=7, weights='distance')
knn = knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
# Preciision, recall, f-score from the multi-class support function
print(classification_report(y_test, y_pred))
print('Accuracy score: ', round(accuracy_score(y_test, y_pred), 2))
print('F1 Score: ', round(f1_score(y_test, y_pred), 2))

# Copy the output to dataframes
StatsOutdf = pd.DataFrame(classification_report(y_test, y_pred, output_dict=True)).transpose()
Confusiondf = pd.DataFrame(confusion_matrix(y_test, y_pred))

# Run the LoadStats Function
LoadStats(CummulativeStatsdf, 'KNN', "Neighbors = 7", StatsOutdf, Confusiondf)
CummulativeStatsdf

# Copy the output to Excel
CummulativeStatsdf.to_clipboard(excel = True, index = True)
print('Paste the data in Excel')
```

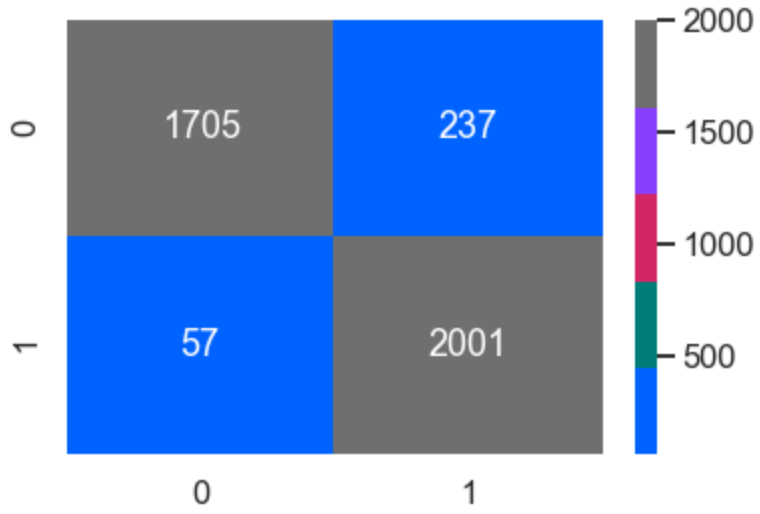
Model Output for KNN, K = 7

Model	Notes	MTMPatient	Precision	Recall	F1 Score	TP	FP
KNN	Neighbors = 7	0	.968	.878	.921	1,705	237
KNN	Neighbors = 7	1	.894	.972	.932	2,001	57

Plotting the KNN (K = 7) Confusion Matrix

```
from sklearn.metrics import confusion_matrix

sns.set_context('talk')
cm = confusion_matrix(y_test, y_pred)
ax = sns.heatmap(cm, annot=True, fmt='d', cmap=colors)
```

KNN (K=7) Confusion Matrix



Random Forest Classifiers

The nesxt step was to look at the various randome forest classifiers. Thise included 1) Random Trees, 2) Extra Trees,

Turn Off Warnings

```
# Suppress warnings about too few trees from the early models
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=RuntimeWarning)
```

Random Trees and Out Of Bag Error

The next step was to look at Random Trees and the Out Of Bag (OOB) error.

Run the Random Forest Classifier with Different Numbers of Trees

```
from sklearn.ensemble import RandomForestClassifier

# Initialize the random forest estimator
# Note that the number of trees is not setup here
RF = RandomForestClassifier(oob_score=True,
                            random_state=42,
                            warm_start=True,
                            n_jobs=-1)

oob_list = list()

# Iterate through all of the possibilities for
# number of trees
for n_trees in [15, 20, 30, 40, 50, 100, 150, 200, 300, 400]:

    # Use this to set the number of trees
    RF.set_params(n_estimators=n_trees)

    # Fit the model
    RF.fit(X_train, y_train)

    # Get the oob error
    oob_error = 1 - RF.oob_score_

    # Store it
    oob_list.append(pd.Series({'n_trees': n_trees, 'oob': oob_error}))

rf_oob_df = pd.concat(oob_list, axis=1).T.set_index('n_trees')
```

Extra Trees and the Out Of Bag Error

The next step was to look at the Extra Trees Classifier.

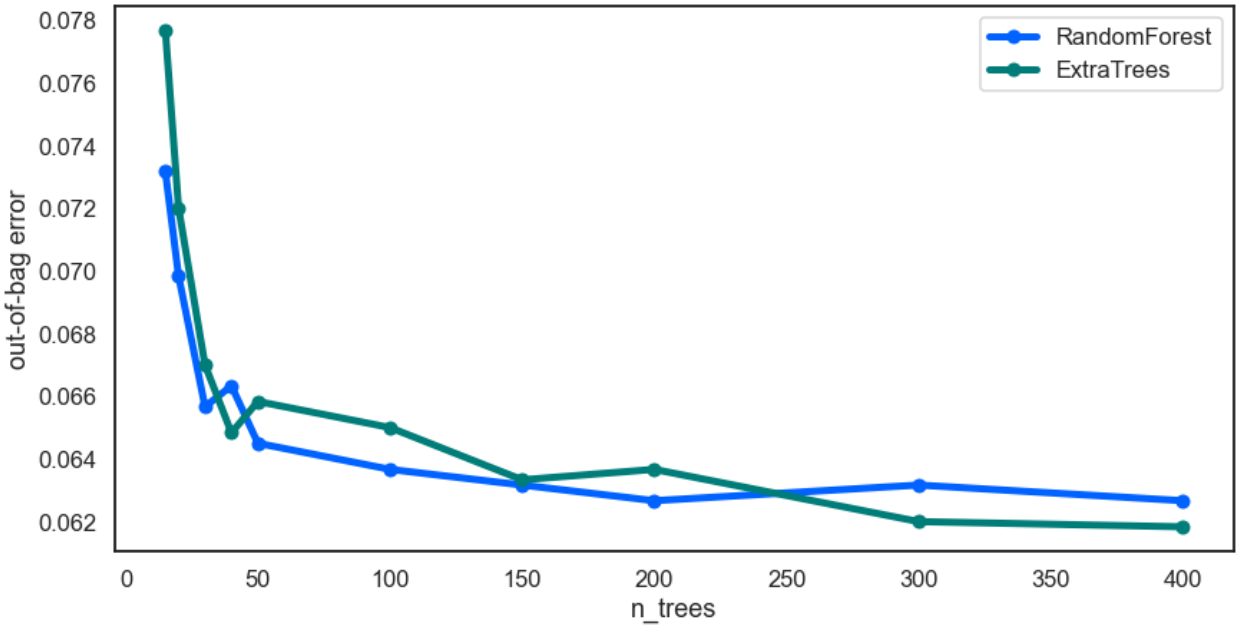
Run the Extra Trees Classifier with Different Numbers of Trees

```
from sklearn.ensemble import ExtraTreesClassifier

# Initialize the random forest estimator
# Note that the number of trees is not setup here
EF = ExtraTreesClassifier(oob_score=True,
                          random_state=42,
                          warm_start=True,
                          bootstrap=True,
                          n_jobs=-1)

oob_list = list()

# Iterate through all of the possibilities for
# number of trees
for n_trees in [15, 20, 30, 40, 50, 100, 150, 200, 300, 400]:

   # Use this to set the number of trees
   EF.set_params(n_estimators=n_trees)
   EF.fit(X_train, y_train)

   # oob error
   oob_error = 1 - EF.oob_score_
   oob_list.append(pd.Series({'n_trees': n_trees, 'oob': oob_error}))

et_oob_df = pd.concat(oob_list, axis=1).T.set_index('n_trees')
```

Plot the Out Of Bag Errors for Random Forest and Extra Trees

```
# Create the Out of Bag Dataframe
oob_df = pd.concat([rf_oob_df.rename(columns={'oob':'RandomForest'}),
                    et_oob_df.rename(columns={'oob':'ExtraTrees'})], axis=1)
oob_df

# Plot the OOB comparison
sns.set_context('talk')
sns.set_style('white')

ax = oob_df.plot(marker='o', figsize=(14, 7), linewidth=5)
ax.set(ylabel='out-of-bag error');
```

Out Of Bag Error Comparison



Create the Random Forest Model (Trees = 100)

```
# Random forest with 100 estimators
model = RF.set_params(n_estimators=100)

y_pred = model.predict(X_test)
from sklearn.metrics import classification_report, accuracy_score, precision_score, recall_score
from sklearn.metrics import f1_score, roc_auc_score

cr = classification_report(y_test, y_pred)
print(cr)

score_df = pd.DataFrame({'accuracy': accuracy_score(y_test, y_pred),
                         'precision': precision_score(y_test, y_pred),
                         'recall': recall_score(y_test, y_pred),
                         'f1': f1_score(y_test, y_pred),
                         'auc': roc_auc_score(y_test, y_pred)},
                         index=pd.Index([0]))
```

Random Forest Output

Model	Notes	MTMPatient	Precision	Recall	F1 Score	TP	FP
Random Forest	Trees = 100	0	.975	.913	.943	1,773	169
Random Forest	Trees = 100	1	.923	.978	.949	2,012	46

Create the Extra Trees Model (Trees = 100)

```
# Extra Trees with 100 estimators

model = EF.set_params(n_estimators=100)

y_pred = model.predict(X_test)
from sklearn.metrics import classification_report, accuracy_score, precision_score, recall_score
from sklearn.metrics import f1_score, roc_auc_score

cr = classification_report(y_test, y_pred)
print(cr)

score_df = pd.DataFrame({'accuracy': accuracy_score(y_test, y_pred),
                         'precision': precision_score(y_test, y_pred),
                         'recall': recall_score(y_test, y_pred),
                         'f1': f1_score(y_test, y_pred),
                         'auc': roc_auc_score(y_test, y_pred)},
                         index=pd.Index([0]))
```

Extra Trees Output

Model	Notes	MTMPatient	Precision	Recall	F1 Score	TP	FP
Extra Trees	Trees = 100	0	.972	.910	.940	1,767	175
Extra Trees	Trees = 100	1	.920	.975	.947	2,007	51

Create the Logistic Regression Model (Penalty = 12)

```
from sklearn.linear_model import LogisticRegression

# L2 regularized logistic regression
LR_L2 = LogisticRegression(penalty='l2', max_iter=500, solver='saga').fit(X_train, y_train)
y_pred = LR_L2.predict(X_test)
print(classification_report(y_pred, y_test))
```

Logistic Regression Output

Model	Notes	MTMPatient	Precision	Recall	F1 Score	TP	FP
Logistic Regression	Penalty = 12	0	.878	.890	.884	1,728	214
Logistic Regression	Penalty = 12	1	.895	.884	.889	1,819	239

Create the Gradient Boosting Model

```
from sklearn.model_selection import GridSearchCV

# The parameters to be fit
param_grid = {'n_estimators': tree_list,
        'learning_rate': [0.1, 0.01, 0.001, 0.0001],
        'subsample': [1.0, 0.5],
        'max_features': [1, 2, 3, 4]}

# The grid search object
GV_GBC = GridSearchCV(GradientBoostingClassifier(random_state=42),
            param_grid=param_grid,
            scoring='accuracy',
            n_jobs=-1)

# Do the grid search
GV_GBC = GV_GBC.fit(X_train, y_train)

# Describe the best model
GV_GBC.best_estimator_

        Output: GradientBoostingClassifier(max_features=4, n_estimators=400,
        random_state=42, subsample=0.5)

# Get the report
from sklearn.metrics import classification_report

y_pred = GV_GBC.predict(X_test)
print(classification_report(y_pred, y_test))
```

Gradient Boosting Output

Model	Notes	MTMPatient	Precision	Recall	F1 Score	TP	FP
Gradient Boosting	Estimators = 400	0	.935	.906	.921	1,760	182
Gradient Boosting	Estimators = 400	1	.914	.941	.927	1,936	122

Voting Classifier

The last step involved looking at the voting classifier, using the logistic regression and gradient boosting models as inputs.

Voting Classifier Python Code

```
from sklearn.ensemble import VotingClassifier

# The combined model--logistic regression and gradient boosted trees
estimators = [('LR_L2', LR_L2), ('GBC', GV_GBC)]

# Though it wasn't done here, it is often desirable to train
# this model using an additional hold-out data set and/or with cross validation
VC = VotingClassifier(estimators, voting='soft')
VC = VC.fit(X_train, y_train)

y_pred = VC.predict(X_test)
print(classification_report(y_test, y_pred))
```

Voting Classifier Output

Model	Notes	MTMPatient	Precision	Recall	F1 Score	TP	FP
Voting Classifier	LR and GBC	0	.909	.901	.905	1,750	192
Voting Classifier	LR and GBC	1	.907	.915	.911	1,883	175

Key Findings

After trying several different types of classification models, it seemed that the Random Forest classifier outperformed all of the rest. The Random Forest had the highest scores and the greatest number of True Positives (2,012) for predicting which patients had an MTM visit. If I were to put this model into production, it would seem that the Random Forest classifier would provide the best accuaracy.

Model Comparisons

Model	Notes	MTMPatient	Precision	Recall	F1 Score	TP	FP
KNN	Neighbors = 7	0	.968	.878	.921	1,705	237
KNN	Neighbors = 7	1	.894	.972	.932	2,001	57
Random Forest	**Trees = 100**	**0**	**.975**	**.913**	**.943**	**1,773**	**169**
Random Forest	**Trees = 100**	**1**	**.923**	**.978**	**.949**	**2,012**	**46**
Extra Trees	Trees = 100	0	.972	.910	.940	1,767	175
Extra Trees	Trees = 100	1	.920	.975	.947	2,007	51
Logistic Regression	Penalty = 12	0	.878	.890	.884	1,728	214
Logistic Regression	Penalty = 12	1	.895	.884	.889	1,819	239
Gradient Boosting	Estimators = 400	0	.935	.906	.921	1,760	182
Gradient Boosting	Estimators = 400	1	.914	.941	.927	1,936	122
Voting Classifier	LR and GBC	0	.909	.901	.905	1,750	192
Voting Classifier	LR and GBC	1	.907	.915	.911	1,883	175

Note that the descriptive statistics such as the mean differences, correlations, and statsmodel Logit output provided much more detail with regard to the feature importance. It was possible to get the Feature Importance from the Random Forest model. See below.

It appeared that the Random Forest model output was slightly different from the statsmodel Logit output. In terms of predictive power, the Random Forest seemed to be superior.

Possible Flaws

One possible flaws of the model is the balancing process used. Only about 5% of the patients not in the MTM study were used. This is a small percentage. The calculations could be replicated using a larger sample. The choice of 10,000 rows was made to speed model output.

Suggestions for the Future

The model used was very basic. Interaction terms could be added with the polynomial features package. Other data might be used. About 750 total variables were available. There are ways to weed through these to determine if some of those variables might help with model accuracy.

Export Random Forest Feature Importance

Since the Random Forest was the best predictor, I wanted to see the feature importance. The following code provided the values.

Code to Extract Feature Importance

```
Importancedf = pd.DataFrame(RF.feature_importances_)
MTMRowsdf = pd.DataFrame(MTMBalanceddf[feature_columns].columns)
MTMRowsdf.columns = ['Features']
MTMRowsdf["Importance"] = Importancedf.loc[:, 0]
MTMRowsdf.sort_values(by='Importance', inplace=True, ascending=False)

MTMRowsdf.to_clipboard(excel = True, index = True)
print('Paste the data in Excel')

MTMRowsdf
```

Random Forest Feature Importance

Features	Importance
BTM5D	.147
CCS0257	.130
AvgProvidersTestYear	.113
AvgMedicationsTestYear	.108
A1CMax	.086
HCCCount	.080
M1YPmnts	.051
BPSMax	.042
CHLMax	.040
BPSTDays	.040
Age	.038
BMIMax	.035
BPDMax	.032
PHQMax	.029
BTM1A	.008
Male	.006
Single	.005
Under18	.003
Separated	.003
Widowed	.003
HasPCP	.002

Conclusion

Various Python methods were used to determine whether it would be possible to identify patients who are currently receiving pharmacist services though Medication Therapy Management (MTM). It appeared that many of the classifiers would work. The Random Forest classifier worked best. If this model were put into production, a Random Forest Classifier would probably be the optimal choice.