

D603 Task 1
By Eric Williams

B1:PROPOSAL OF QUESTION

Is it possible to predict whether or not a patient will be readmitted to the hospital based on their demographic and their medical history using the Random Forest classification method?

B2:DEFINED GOAL

The goal of this analysis is to use the Random Forest classification method to identify key factors in predicting hospital readmittance. Although this could be used to try to increase profits, the goal of this analysis is to provide patients with better care by helping predict the likelihood that they will be readmitted in the future.

C1:EXPLANATION OF CLASSIFICATION METHOD

The Random Forest classification method is a machine learning tool that will help us create a model to predict future results based on past observations. The Random Forest algorithm builds decision trees and selects random observations, which is much more powerful than an individual decision tree. It's a great tool for handling massive amounts of data and finding complex relationships. I expect that the outcome of this method should be an algorithm that can predict readmittance with a high degree of accuracy based on the input of observable variables in the patients.

C2:PACKAGES OR LIBRARIES LIST

Here are the packages I imported and why I needed them

1. Pandas and Numpy to work with the data in a dataframe and perform calculations needed for analysis
2. Sklearn for its various tools for machine learning. Those specific tools are:
 - a. Train_test_split
 - b. RandomForestClassifier
 - c. LabelEncoder
 - d. enable_halving_search_cv
 - e. Classification_report
 - f. Accuracy_score
 - g. Confusion_matrix
 - h. Roc_auc_score
 - i. Roc_curve
 - j. HalvingGridSearchCV
 - k. K-fold

D1:DATA PREPROCESSING

To prepare the data for processing, I needed to ensure the data was clean with no missing values.

```
print(data.isnull().sum())
```

CaseOrder	0
Customer_id	0
Interaction	0
UID	0
City	0
State	0
County	0
Zip	0
Lat	0
Lng	0
Population	0
Area	0
TimeZone	0
Job	0
Children	0
Age	0
Income	0
Marital	0
Gender	0
ReAdmis	0
VitD_levels	0
Doc_visits	0
Full_meals_eaten	0
vitD_supp	0
Soft_drink	0
Initial_admin	0
HighBlood	0
Stroke	0
Complication_risk	0
Overweight	0
Arthritis	0
Diabetes	0
Hyperlipidemia	0
BackPain	0
Anxiety	0
Allergic_rhinitis	0
Reflux_esophagitis	0
Asthma	0
Services	0
Initial_days	0
TotalCharge	0
Additional_charges	0
Item1	0
Item2	0
Item3	0
Item4	0
Item5	0
Item6	0
Item7	0
Item8	0

dtype: int64

I also stripped any spaces before or after the column titles to reduce the likelihood of problems with column names later on. I then inspected the data to ensure it looked properly prepared for analysis:

```
# Display the first few rows of the dataset
print(data.head())
```

```
CaseOrder Customer_id Interaction \
0 1 C412403 8cd49b13-f45a-4b47-a2bd-173ffa932c2f
1 2 Z919181 d2450b70-0337-4406-bdbb-bc1037f1734c
2 3 F995323 a2057123-abf5-4a2c-abad-8ffe33512562
3 4 A879973 1dec528d-eb34-4079-adce-0d7a40e82205
4 5 C544523 5885f56b-d6da-43a3-8760-83583af94266

UID City State County Zip \
0 3a83ddb66e2ae73798bdf1d705dc0932 Eva AL Morgan 35621
1 176354c5eef714957d486009feabf195 Marianna FL Jackson 32446
2 e19a0fa0a8da885b8a436757e889bc9 Sioux Falls SD Minnehaha 57110
3 cd17d7b6d152cb6f23957346d11c3f07 New Richland MN Waseca 56072
4 d2f0425877b10ed6bb381f3e2579424a West Point VA King William 23181

Lat Lng ... TotalCharge Additional_charges Item1 Item2 Item3 \
0 34.34960 -86.72508 ... 3726.702860 17939.403420 3 3 2
1 30.84513 -85.22907 ... 4193.190458 17612.998120 3 4 3
2 43.54321 -96.63772 ... 2434.234222 17505.192460 2 4 4
3 43.89744 -93.51479 ... 2127.830423 12993.437350 3 5 5
4 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]
```

```
] data.columns = data.columns.str.strip()
```

```
] data.describe()
```

```
]

```

	Children	Age	Income	Gender	ReAdmis	VitD_levels	Doc_visits	Full_meals_eaten	vitD_supp	Soft_drink	...	It
count	10000.000000	10000.000000	10000.000000	9786.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	...	10000.000000
mean	2.097200	53.511700	40490.495160	0.512773	0.366900	17.964262	5.012200	1.001400	0.398900	0.257500	...	3.500000
std	2.163659	20.638538	28521.153293	0.499862	0.481983	2.017231	1.045734	1.008117	0.628505	0.437279	...	1.040000
min	0.000000	18.000000	154.080000	0.000000	0.000000	9.806483	1.000000	0.000000	0.000000	0.000000	...	1.000000
25%	0.000000	36.000000	19598.775000	0.000000	0.000000	16.626439	4.000000	0.000000	0.000000	0.000000	...	3.000000
50%	1.000000	53.000000	33768.420000	1.000000	0.000000	17.951122	5.000000	1.000000	0.000000	0.000000	...	3.000000
75%	3.000000	71.000000	54296.402500	1.000000	1.000000	19.347963	6.000000	2.000000	1.000000	1.000000	...	4.000000
max	10.000000	89.000000	207249.100000	1.000000	1.000000	26.394449	9.000000	7.000000	5.000000	1.000000	...	7.000000

In addition, I dropped columns not related to demographic and medical history, and then encoded the variables because Random Forest requires numerical input. The code for this is provided in sections D2 and D3 respectively.

D2:DATASET VARIABLES

I dropped any columns not related to medical history or patient demographic. Below I have sorted the remaining variables by continuous and categorical, although some could technically be listed as either. This is because some columns are numerical but fit nicely into a category. For example, Items 1 through 8 listed below required the patients to respond to a survey in a specific range with limited options. However, I have chosen to list these as continuous because they are numerical and do not need to be encoded for analysis:

Categorical (non-numerical)

Marital

Gender

ReAdmis

Soft_drink

Initial_admin
HighBlood
Stroke
Overweight
Arthritis
Diabetes
Hyperlipidemia
BackPain
Anxiety
Allergic_rhinitis
Reflux_esophagitis
Asthma
Services
Complication_risk

Continuous (numerical)

Income
VitD_levels
Doc_visits
Initial_days
TotalCharge
Additional_charges
Full_meals_eaten
vitD_supp
Children
Age
Item1
Item2
Item3
Item4
Item5
Item6
Item7
Item8

D3: STEPS FOR ANALYSIS

As previously stated, I looked for missing data and stripped extra spaces on the column titles. Then I dropped the unneeded columns:

```
: data = data.drop(['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'City', 'State',  
                  'County', 'Zip', 'Lat', 'Lng', 'Area', 'Population', 'TimeZone',  
                  'Job'], axis=1)
```

Then I encoded the data:

```

#Encoding columns

#1. Binary Columns encoding
binary_columns = ['ReAdmis', 'HighBlood', 'Stroke', 'Overweight', 'Arthritis',
                  'Diabetes', 'Hyperlipidemia', 'BackPain', 'Anxiety',
                  'Allergic_rhinitis', 'Reflux_esophagitis', 'Asthma', 'Soft_drink']

for col in binary_columns:
    data[col] = data[col].map({'No': 0, 'Yes': 1})

#2. Special binary encoding just for Gender
data['Gender'] = data['Gender'].map({'Male': 0, 'Female': 1})

#3. One-Hot Encoding for Initial_admin and Services (nominal)
data = pd.get_dummies(data, columns=['Initial_admin', 'Services'], drop_first=True)

#4. Ordinal Encoding for Complication_risk
complication_risk_mapping = {'Low': 1, 'Medium': 2, 'High': 3}
data['Complication_risk'] = data['Complication_risk'].map(complication_risk_mapping)

#Checking the dataframe
print(data.head())
print(data.dtypes)

#Converting boolean (True/False) columns to integers
data = data.astype({'col': int for col in data.columns if data[col].dtype == 'bool'})

data.to_csv("C:/Users/18014/Desktop/data_encoded.csv", index=False)

```

The numerical columns did not need to be converted. However, I had to encode the columns so they were numerical so they could be used in the Random Forest analysis. First I converted all of the yes/no entries into 0 and 1 values. Then I did a special conversion for gender, mapping male to 0 and female to 1. Then I encoded Initial_admin and Services using one hot encoding because they were not numerical or ordinal. Then I used ordinal encoding because complication risk was inherently tiered. Lastly, I converted any true/false values into boolean 0's and 1's and exported the data frame. At this point, my data was prepared for the Random Forest algorithm so I began creating the model by splitting the data. The **D4:CLEANED DATASET** requirement will be fulfilled with the submission of the dataset.

E1:SPLITTING THE DATA

For all rubric requirements in section E, I will show the code below for each segment:

```

#Splitting the data into readmission and all other variables
X = data.drop('ReAdmis', axis=1)
y = data['ReAdmis']

#Split the data into training and testing 80/20
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=1)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=1)

#Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=1)

#Training the model and predictions
rf_model.fit(X_train, y_train)
y_val_pred = rf_model.predict(X_val)

```

E2:INITIAL MODEL CREATION

The beginning of the model creation is listed above here are the results:

```
#Accuracy and classification
print("Validation Accuracy:", accuracy_score(y_val, y_val_pred))
print("Validation Classification Report:\n", classification_report(y_val, y_val_pred))
```

Validation Accuracy: 0.9815

Validation Classification Report:

	precision	recall	f1-score	support
0	0.98	0.99	0.99	1288
1	0.98	0.97	0.97	712
accuracy			0.98	2000
macro avg	0.98	0.98	0.98	2000
weighted avg	0.98	0.98	0.98	2000

```
#Predicting probabilities for the positive class (class 1)
y_prob = rf_model.predict_proba(X_test)[: , 1]
```

```
#Calculating AUC
```

```
auc = roc_auc_score(y_test, y_prob)
print(f'AUC-ROC: {auc}')
```

AUC-ROC: 0.9978429108737342

```
#Generating a confusion matrix
```

```
y_pred = rf_model.predict(X_test)
```

```
cm = confusion_matrix(y_test, y_pred)
```

```
cm
```

```
array([[1247, 22],
       [ 28, 703]], dtype=int64)
```

E3:HYPERPARAMETER TUNING

Here are the hyperparameters I used for tuning and the justification for each:

1. **n_estimators**: The number of trees in the forest. The more trees in the forest, the higher the accuracy--but as this is a computationally heavy process, it's best to analyze when the model will begin to get diminishing returns in performance to avoid wasting time and energy.
2. **max_depth**: The maximum depth of the trees. Generally, deeper trees will allow the model to capture more complicated relationships. However, to avoid possible overfitting but still allow for capturing the in depth relationships, it's important to find a balance.
3. **min_samples_split**: The minimum number of samples required to split an internal node. Again, we want to find a balance between capturing complicated relationships and overfitting. Adjusting the minimum number of samples required to split an internal node to see which is more optimal for our model will help us create a better model.

Here is the result of the hyperparameter tuning:

Best hyperparameters found: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 200}

Below is the code I used for this process and a screenshot of the results:

```
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold

#Parameters for tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}

#Initializing Random Forest model
rf_model = RandomForestClassifier(random_state=1)

#K-Fold cross-validation
halving_search = HalvingGridSearchCV(
    estimator=rf_model,
    param_grid=param_grid,
    cv=5,
    factor=2,
    random_state=1,
    scoring='roc_auc',
    verbose=2
)

#Hyperparameter search on training set
halving_search.fit(X_train, y_train)

#Give an output of the best parameters
print("Best hyperparameters found: ", halving_search.best_params_)
optimized_rf_model = halving_search.best_estimator_
```

Best hyperparameters found: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 200}

```
: #Random Forest Classifier with the best hyperparameters
optimized_rf_model = RandomForestClassifier(
    max_depth=None,
    min_samples_split=2,
    n_estimators=200,
    random_state=1
)

#Train model and make predictions on test set
optimized_rf_model.fit(X_train, y_train)
y_pred = optimized_rf_model.predict(X_test)

#Evaluating model performance and confusion matrix
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

E4:PREDICTIONS

After running the above code with new specifications for the random forest algorithm, here are the results:

```
#Evaluating model performance and confusion matrix
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9755
Classification Report:
              precision    recall  f1-score   support

     0       0.98      0.98      0.98     1269
     1       0.97      0.96      0.97      731

   accuracy          0.98      2000
  macro avg       0.97      0.97      0.97     2000
 weighted avg     0.98      0.98      0.98     2000

Confusion Matrix:
[[1248  21]
 [ 28 703]]
```

```
#Making predictions on the test set
y_pred_prob = halving_search.best_estimator_.predict_proba(X_test)[: , 1]

#Calculating AUC-ROC
roc_auc = roc_auc_score(y_test, y_pred_prob)
print("AUC-ROC:", roc_auc)
```

AUC-ROC: 0.9986972557595998

F1:MODEL EVALUATION

Here is a side by side comparison of accuracy, precision, recall, F1 score, and AUC-ROM:

Initial model:

```
Validation Accuracy: 0.9815
Validation Classification Report:
              precision    recall  f1-score   support

     0       0.98      0.99      0.99     1288
     1       0.98      0.97      0.97      712

   accuracy          0.98      2000
  macro avg       0.98      0.98      0.98     2000
 weighted avg     0.98      0.98      0.98     2000
```

AUC-ROC: 0.9978429108737342

Optimized Model:

```
Accuracy: 0.9815
Classification Report:
              precision    recall  f1-score   support

     0       0.98      0.98      0.98     1269
     1       0.97      0.96      0.97      731

   accuracy          0.98      2000
  macro avg       0.97      0.97      0.97     2000
 weighted avg     0.98      0.98      0.98     2000
```

AUC-ROC: 0.9979302293241229

Calculated comparisons:

1. Accuracy did not notably improve, but I calculated the improvement at 0.05% since it is less than the rounding value
2. Precision, Recall, F1 score are all unchanged in the weighted average. There are some small variations in the positive and negative class, but the changes aren't significant overall
3. AUC-ROC improved by 0.0087%

To look at the actual changes in the model and avoid making conclusions that are caused by rounding, we need to observe the confusion matrices:

Initial model:

```
([[1247, 22],  
 [ 28, 703]],
```

Optimized Model:

```
[[1248 21]  
 [ 28 703]]
```

The optimization was successful in moving one datapoint from the false positives to the true positives. That's it--that is the extent of the improvement. That means in 50 misattributed datapoints, the optimization fixed one. Because the model was so accurate, the improvements are fairly negligible--but it was an improvement nonetheless.

F2: RESULTS AND IMPLICATIONS

The results of the analysis are twofold: firstly, I created a model that predicts hospital readmittance with an astoundingly accurate 98%. Given the data of a future patient, we can almost certainly predict whether or not the patient will be readmitted. This can help the hospital give better medical care and help patients manage their expectations when it comes to readmittance. The hospital could also look into what the greatest predictors of readmittance could be and use that to create proactive health care plans for patients. Secondly, this data shows that there is a nearly perfectly predictable correlation between patients and their readmittance. This is significant because it means there are specific trends among patients and that readmittance is not random.

F3: LIMITATION

There are some potential limitations to this analysis. Firstly, if this data is not representative of all patients, we would need to adjust our model in the future. It would be best to check that this model is accurate when presented with new data because a 98% accuracy might be too good to be true in the real world. It would also be helpful to introduce new data to ensure our model is not overfit. However, most of the concerns moving forward should be the quality and bias of the data because the model is very accurate.

F4:COURSE OF ACTION

Originally, I set out to create a prediction model for whether or not patients would be readmitted based on their demographic and their background. Because this model is so successful in predicting readmittance, the best thing to do is to use it to help patients. We should use this model to tell patients their likelihood of readmittance to help manage their expectations. Next, we should identify the strongest predictors of readmittance and work with patients to mitigate any behavior that could contribute to readmittance.

Sources

No sources were used except for WGU official course materials.