Random Forest Classification – Performance Assessment

Steven Schindler

Data Mining – D209

Elleh, Festus; PhD

College of I.T., Western Governors University

Table of Contents

Part I: Research Question

A1.

The data set I am working with is the Churn Dataset which has customer demographics, services and products purchased by the customer as well as if they have discontinued service in the past month. My research question is as follows: “Can Random Forest Classification be used to predict whether a customer will churn?”

A2.

The primary goal of this analysis is to develop a Random Forest Classifier model to help the company predict customer churn, so that they might take proactive steps in preventing churn.

Part II: Method Justification

B1.

Random forest classification is good for predicting churn as it is a categorical variable. It constructs multiple decision trees at training time. Each tree then makes an individual prediction then the most popular of these predictions is taken to get a single result. The expected outcome for the random forest model would be Yes or No for each decision tree made. Then a single result is produced from the most popular choice of each decision tree. (DataCamp. (n.d.).)

B2.

An assumption of random forest classifier is that sampling is representative, the tree-based model relies heavily on the training set to build the model.(Data Science Stack Exchange. (2014).)

B3.

The pandas library is a good tool for working with data sets as it allows data to be read in a dataframe. The numpy library is efficient working with multi-dimensional arrays and matrices, this allows for easier manipulation of the data. The string and stats libraries are good for manipulating character strings and doing basic statistic calculations respectively. The warnings library is used to suppress unnecessary warnings. The variance_inflation_factor library is used for vif calculations.

Finally, the sklearn library can import many modules such as model_selection, feature_selection, ensemble, and metrics. These libraries allow for quick work to be done in random forest algorithms. Ensemble has the random forest classification function which implements the algorithm. Model_selection allows the data to be split into training and testing sets and has RandomizedSearchCV which implements randomized search cross validation. Feature_selection has SelectKBest which can select the optimal independent variables to use. Metrics has the accuracy_score and mean_squared_error funtions which calculate the model's accuracy and mean square error respectively.

Part III: Data Preparation

C1.

One goal for data preprocessing is to express categorical values numerically. One way is to use one hot encoding to create dummy variables for categorical variables with more than yes/no values. The second way is to represent "yes" as 1 and "no" as zero. These two methods will allow categorical values to be expressed numerically so the Random Forest Classifier can be used effectively.

C2.

Using SelectKBest(Jepsds. (2021).) to select features based off p-values the variables in the following table are the features selected:

Independent Variable	Data Type	Data Class
Bandwidth_GB_Year	Continuous	Quantitative
MonthlyCharge	Continuous	Quantitative
Tenure	Continuous	Quantitative
StreamingMovies_Yes	Categorical	Qualitative
StreamingMovies_No	Categorical	Qualitative
Contract_Month-to-month	Categorical	Qualitative
StreamingTV_Yes	Categorical	Qualitative
StreamingTV_No	Categorical	Qualitative
Contract_Two Year	Categorical	Qualitative
Contract_One year	Categorical	Qualitative
Multiple_No	Categorical	Qualitative
Multiple_Yes	Categorical	Qualitative
InternetService_DSL	Categorical	Qualitative
Techie_No	Categorical	Qualitative
Techie_Yes	Categorical	Qualitative
InternetService_Fiber Optic	Categorical	Qualitative
DeviceProtection_Yes	Categorical	Qualitative
DeviceProtection_No	Categorical	Qualitative
OnlineBackup_No	Categorical	Qualitative
OnlineBackup_Yes	Categorical	Qualitative
InternetService_None	Categorical	Qualitative
PaymentMethod_Electronic Check	Categorical	Qualitative
Phone_Yes	Categorical	Qualitative
Phone_No	Categorical	Qualitative

To Reduce multicollinearity, I remove Contrract_Two Year and InternetService_None, as well as the "No" response for all Yes/No variables using k-1 column removal. Therefore, my new table looks like this:

Independent Variable	Data Type	Data Class
Bandwidth_GB_Year	Continuous	Quantitative
MonthlyCharge	Continuous	Quantitative
Tenure	Continuous	Quantitative
StreamingMovies_Yes	Categorical	Qualitative
Contract_Month-to-month	Categorical	Qualitative

StreamingTV_Yes	Categorical	Qualitative
Contract_One year	Categorical	Qualitative
Multiple_Yes	Categorical	Qualitative
InternetService_DSL	Categorical	Qualitative
Techie_Yes	Categorical	Qualitative
InternetService_Fiber Optic	Categorical	Qualitative
DeviceProtection_Yes	Categorical	Qualitative
OnlineBackup_Yes	Categorical	Qualitative
PaymentMethod_Electronic Check	Categorical	Qualitative
Phone_Yes	Categorical	Qualitative

C3.

The first step was to read the csv into a pandas dataframe, then drop categorical variables with either too many values(city, state, etc.) or variables that could lead to ethical qualms when predicting churn(Gender):

```
df = pd.read_csv('churn_clean.csv')
df=df.drop(['City','State','County','Customer_id','Interaction','UID','Job','Gender'],
axis=1)
```

The next step is to check for missing values and trailing or leading spaces:

```
df.isna().sum()

#strips leading and trailing spaces
string_list = list(df.select_dtypes(include = {'object'}))
for i in string_list:
    df[i] = df[i].str.strip()
```

Then we remove outliers from the numeric variables that have a Z-score less than 3, check the min/max of all numeric variables for nonsensical outliers such as zero for population. Then remove all records where population is zero as there cannot be a zero population and it is less than 1 percent of the data:

```
# remove outliers where z score is < 3
col_num_names = list(df.select_dtypes(include = {'float64','int64'}))

for i in col_num_names:
    z = pd.DataFrame(np.abs(stats.zscore(df[i])))
    df_clean = df[(z<3).all(axis=1)]
    print(df_clean[i].name,"min is ", df_clean[i].min(),"max is
",df_clean[i].max(),"\n")

#population cannot be zero.
count = (df_clean['Population']==0).sum()
print(count)

#less than 1 percent of the data so we can remove the zeros from population
df_clean.drop(df_clean[df_clean['Population'] == 0].index, inplace = True)
```

Finally, we can add dummy variables with One Hot Encoding using the get_dummies function from the pandas library:

```
new_df = pd.get_dummies(df_clean)
```

C4.

The new data file is called clean_d209_2.csv.

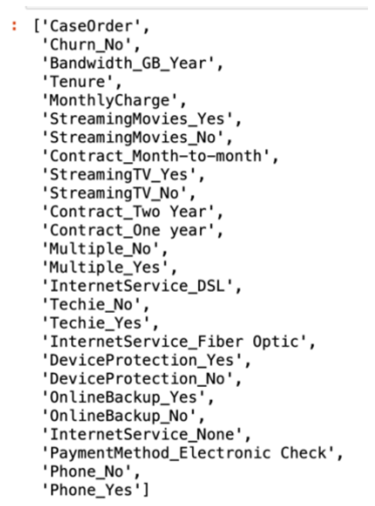
Part IV: Analysis

D1.

The files for the training sets are named X_train.csv and y_train.csv. The files for the testing sets are named X_test.csv and y_test.csv. They are all included in the submission.

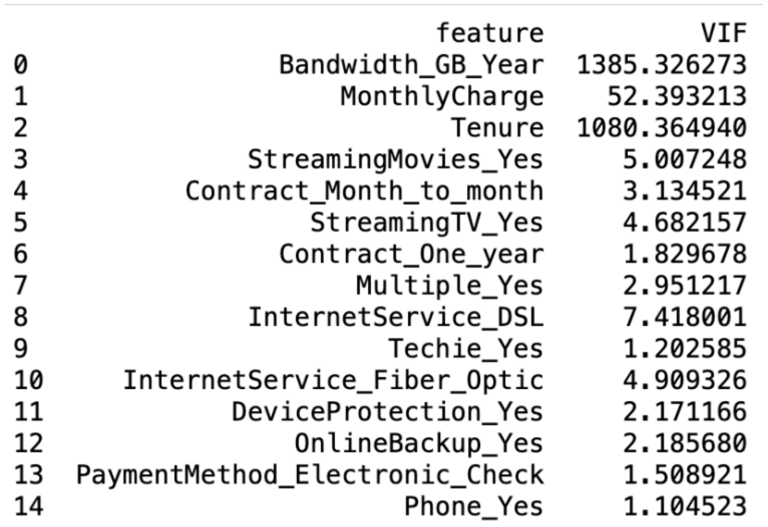
D2.

The first step was to split the data into X and y with y being the "Yes" value for churn and X being everything but the "Yes" value for churn. Then we select the best X variables that have p-values less than .05. We are left with these variables:



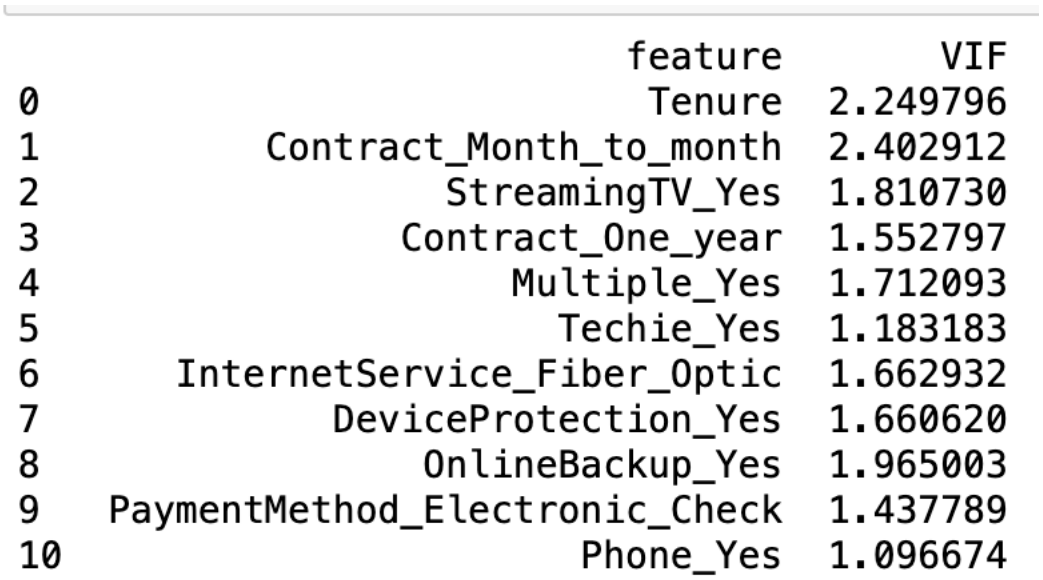
```
: ['CaseOrder',
   'Churn_No',
   'Bandwidth_GB_Year',
   'Tenure',
   'MonthlyCharge',
   'StreamingMovies_Yes',
   'StreamingMovies_No',
   'Contract_Month-to-month',
   'StreamingTV_Yes',
   'StreamingTV_No',
   'Contract_Two Year',
   'Contract_One year',
   'Multiple_No',
   'Multiple_Yes',
   'InternetService_DSL',
   'Techie_No',
   'Techie_Yes',
   'InternetService_Fiber Optic',
   'DeviceProtection_Yes',
   'DeviceProtection_No',
   'OnlineBackup_Yes',
   'OnlineBackup_No',
   'InternetService_None',
   'PaymentMethod_Electronic Check',
   'Phone_No',
   'Phone_Yes']
```

Then I created a new data frame with only these variables then checked the VIF scores for multicollinearity:



```
                          feature          VIF
0                 Bandwidth_GB_Year  1385.326273
1                     MonthlyCharge    52.393213
2                            Tenure  1080.364940
3               StreamingMovies_Yes     5.007248
4            Contract_Month_to_month     3.134521
5                  StreamingTV_Yes     4.682157
6                 Contract_One_year     1.829678
7                      Multiple_Yes     2.951217
8               InternetService_DSL     7.418001
9                       Techie_Yes     1.202585
10      InternetService_Fiber_Optic     4.909326
11            DeviceProtection_Yes     2.171166
12                OnlineBackup_Yes     2.185680
13  PaymentMethod_Electronic_Check     1.508921
14                       Phone_Yes     1.104523
```

I then removed 'Bandwidth_GB_Year', 'MonthlyCharge', 'StreamingMovies_Yes', 'InternetService_DSL' to reduce multicollinearity as these variables all have VIF > 5. I did not remove ‘Tenure’ as it is highly correlated with Bandwidth.



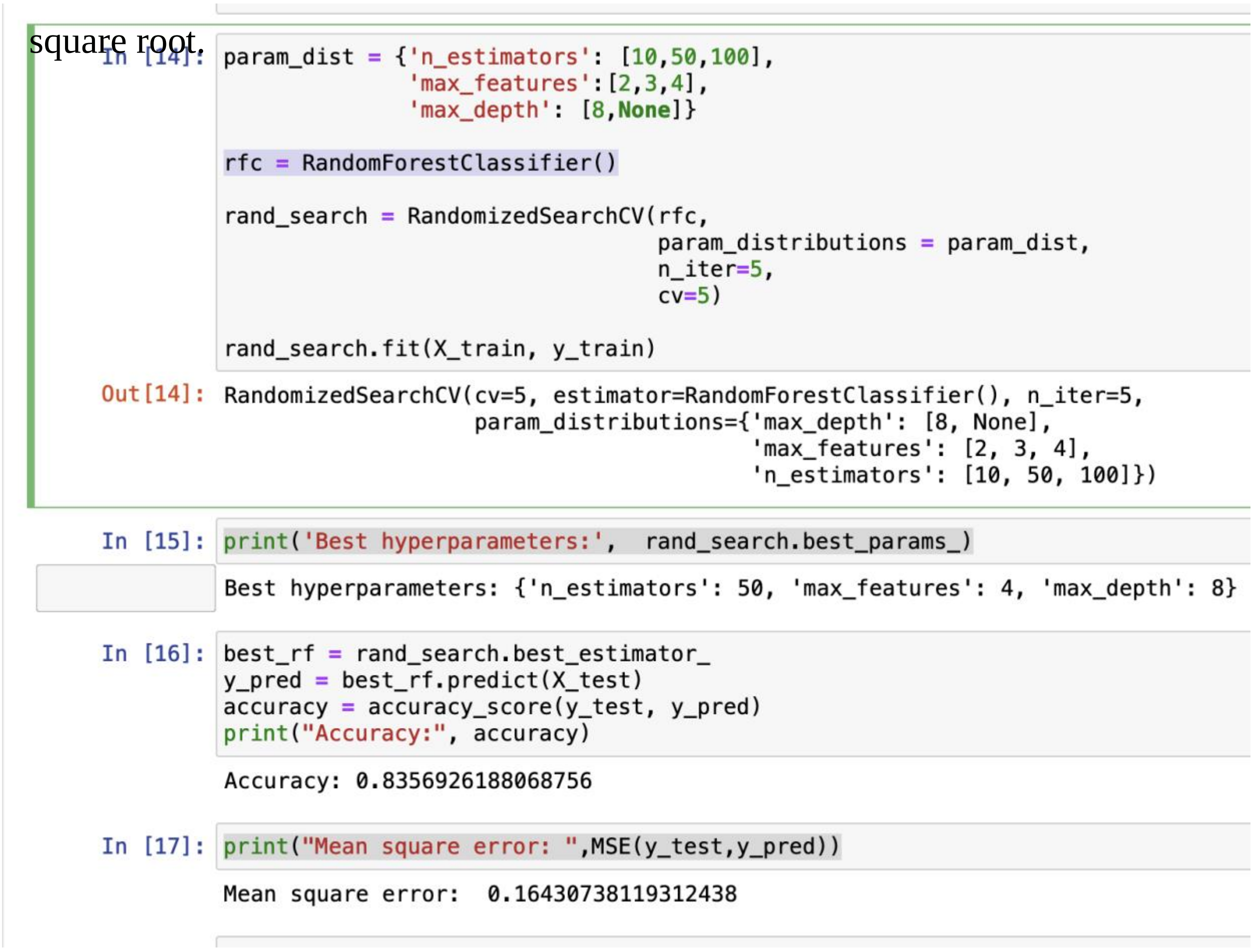
```
                          feature       VIF
0                           Tenure  2.249796
1          Contract_Month_to_month  2.402912
2                  StreamingTV_Yes  1.810730
3                Contract_One_year  1.552797
4                     Multiple_Yes  1.712093
5                       Techie_Yes  1.183183
6      InternetService_Fiber_Optic  1.662932
7             DeviceProtection_Yes  1.660620
8                 OnlineBackup_Yes  1.965003
9   PaymentMethod_Electronic_Check  1.437789
10                       Phone_Yes  1.096674
```

After removing those 4 variables the VIF’s were now all below 5. I then split the data into training and testing sets with training being 80% and testing being

20% of the data respectively. Next I create a param_dist dictionary and instantiate a random forest classifier called rfc.

param_dist = {'n_estimators': [10,50,100],
'max_features':[2,3,4],
'max_depth': [8,None]}
rfc = RandomForestClassifier()

Then I used a randomized search cross validation which will randomly search parameters within a range per hyperparameter. (DataCamp. (n.d.).) Finally, I fit the data to the training set and print the best parameters, accuracy and mean square root.



```
In [14]: param_dist = {'n_estimators': [10,50,100],
                       'max_features':[2,3,4],
                       'max_depth': [8,None]}

         rfc = RandomForestClassifier()

         rand_search = RandomizedSearchCV(rfc,
                                          param_distributions = param_dist,
                                          n_iter=5,
                                          cv=5)

         rand_search.fit(X_train, y_train)

Out[14]: RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_iter=5,
                            param_distributions={'max_depth': [8, None],
                                                 'max_features': [2, 3, 4],
                                                 'n_estimators': [10, 50, 100]})

In [15]: print('Best hyperparameters:',  rand_search.best_params_)

         Best hyperparameters: {'n_estimators': 50, 'max_features': 4, 'max_depth': 8}

In [16]: best_rf = rand_search.best_estimator_
         y_pred = best_rf.predict(X_test)
         accuracy = accuracy_score(y_test, y_pred)
         print("Accuracy:", accuracy)

         Accuracy: 0.8356926188068756

In [17]: print("Mean square error: ",MSE(y_test,y_pred))

         Mean square error:  0.16430738119312438
```

D3.

X = new_df.drop(["Churn_Yes"],1)
y = new_df["Churn_Yes"]

```
feature_names = X.columns

skbest = SelectKBest(score_func = f_classif, k='all')
X_new = skbest.fit_transform(X, y)

p_values = pd.DataFrame({'Feature': X.columns,
'p_value':skbest.pvalues_}).sort_values('p_value')
p_values[p_values['p_value'] < .05]

features_to_keep = p_values['Feature'][p_values['p_value'] < .05]
# Print the name of the selected features
features_to_keep.tolist()

new_df = pd.DataFrame().assign(
Bandwidth_GB_Year=new_df['Bandwidth_GB_Year'],
 MonthlyCharge=new_df['MonthlyCharge'],
 Tenure=new_df[ 'Tenure'],
 StreamingMovies_Yes=new_df[ 'StreamingMovies_Yes'],
 Contract_Month_to_month=new_df['Contract_Month-to-month'],
 StreamingTV_Yes=new_df[ 'StreamingTV_Yes'],
 Contract_One_year=new_df[ 'Contract_One year'],
 Multiple_Yes=new_df[ 'Multiple_Yes'],
 InternetService_DSL=new_df[ 'InternetService_DSL'],
 Techie_Yes=new_df[ 'Techie_Yes'],
 InternetService_Fiber_Optic=new_df[ 'InternetService_Fiber Optic'],
 DeviceProtection_Yes=new_df[ 'DeviceProtection_Yes'],
 OnlineBackup_Yes=new_df[ 'OnlineBackup_No'],
 PaymentMethod_Electronic_Check=new_df[ 'PaymentMethod_Electronic
Check'],
 Phone_Yes=new_df[ 'Phone_No'])

vif_data = pd.DataFrame()
vif_data["feature"] = new_df.columns

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(new_df.values, i)
                          for i in range(len(new_df.columns))]
print(vif_data)
```

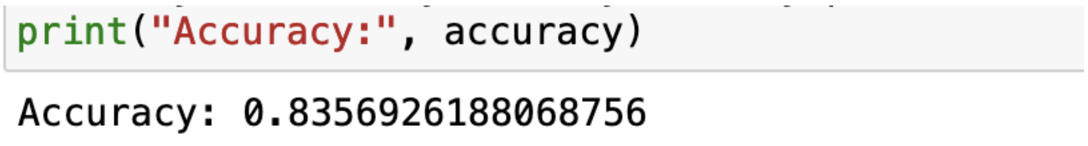
```
# remove variables with multicolinearity > 5
new_df =
new_df.drop(['Bandwidth_GB_Year','MonthlyCharge','StreamingMovies_Yes','Int
ernetService_DSL'],axis = 1)
vif_data = pd.DataFrame()
vif_data["feature"] = new_df.columns

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(new_df.values, i)
                   for i in range(len(new_df.columns))]
print(vif_data)

X = new_df
X.info()

X_train, X_test, y_train, y_test = train_test_split(X, y,
train_size = 0.8, test_size=0.2, random_state=15, stratify=y)

X_train.to_csv('X_train.csv')
X_test.to_csv('X_test.csv')
y_train.to_csv('y_train.csv')
y_test.to_csv('y_test.csv')

param_dist = {'n_estimators': [10,50,100],
          'max_features':[2,3,4],
          'max_depth': [8,None]}

rfc = RandomForestClassifier()

rand_search = RandomizedSearchCV(rfc,
                         param_distributions = param_dist,
                         n_iter=5,
                         cv=5) (DataCamp. (n.d.).)



rand_search.fit(X_train, y_train)

print('Best hyperparameters:',  rand_search.best_params_)

best_rf = rand_search.best_estimator_
```

```
y_pred = best_rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Mean square error: ",MSE(y_test,y_pred))
```

Part V: Data Summary and Implications

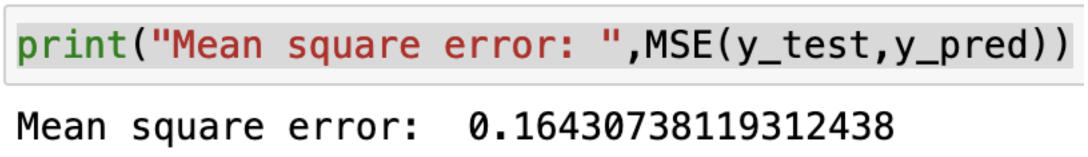
E1.

The accuracy score for my model is .835 meaning my model is correct 83.5% of the time. The mean square error or MSE is .164 the lower the value the more accurate a model is. With my MSE being .164 means that my model is wrong 16.4% of the time.



```
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8356926188068756
```



```
print("Mean square error: ",MSE(y_test,y_pred))
```

```
Mean square error:  0.16430738119312438
```

E2.

I used Random Forest Classification machine learning for this analysis, it led to an accuracy of 83.5%. I used SelectKBest to select the best independent variables to get the most accurate results. Then I used Randomized Search Cross Validation for hyperparameter tuning of the model. The model accuracy could be improved further with hyperparameter tuning on parameters that were not used such as min_sample_split or min_sample_leaf.

E3.

A limitation of my analysis is the data available is not equally distributed. The csv file had 10,000 customers but 7350 were non churn customers while 2650 were churn customers which could result in getting the less common "Yes" wrongly classified. A more balanced data set could improve model performance.

E4.

The model's accuracy is 83% while that is near 100% there is still room for error. The company could however label those who fall into the category of "Yes" for churn as a "high risk" for churning. From there marketing or other customer retention methods could be applied to these customers in an attempt to mitigate churn.

References

G.

Jepsds. (2021). Feature Selection using SelectKBest. Kaggle. Retrieved April 18, 2023, from https://www.kaggle.com/code/jepsds/feature-selection-using-selectkbest/notebook

DataCamp. (n.d.). Random Forest Classifier in Python. Retrieved April 21, 2023, from https://www.datacamp.com/tutorial/random-forests-classifier-python

H.

Data Science Stack Exchange. (2014). Assumptions/limitations of random forest models. Retrieved April 21, 2023, from https://datascience.stackexchange.com/questions/6015/assumptions-limitations-of-random-forest-models