Telecom Churn Analysis

Javier Lopez

D209, Predictive Analysis

Table of Contents

Part I. Research Question

A1. Research Question

Can we accurately predict how much bandwidth a customer will use in twelve months?

A2. Goals

We aim to understand customer bandwidth usage and create a model to predict future customer bandwidth usage accurately. We also aim to understand trends in future customer bandwidth usage.

Part II. Method Justification

B1. Prediction Method

We chose to predict future bandwidth usage with a random forests model. The model will begin analyzing the data by building several trees on the training data; that is where the forests part of the algorithm name comes into play. The model constructs the trees using a randomly selected subset of the features and a subset of the training data. Then, at each decision tree node, the model will choose the best possible split based on specific splitting criteria, such as the lowest variance of the response variable. The process recursively continues until the tree reaches stopping criteria, such as the maximum depth or the minimum samples at a leaf node.

Once the model builds the decision trees, it will combine their predictions to make a single prediction for each data point, usually the average or majority vote of all predictions on the tree. Our expected output from the random forests model is a prediction of the bandwidth usage for each customer and a significantly small mean squared error (MSE), the square of the average delta between the true and predicted values.

B2. Assumption

One of the multiple assumptions of a random forests regression model is that the data has no multicollinearity. Similar to other regression models, if there is high multicollinearity among the input features, the model may not be able to determine the relative importance of each variable when predicting the target variable. To solve this assumption, we will calculate the variance inflation factor for each feature. The variance inflation factor will estimate the multicollinearity for each feature relative to the other features within the data set. We expect an outcome of less than five for each feature not to be considered high multicollinearity.

B3. Libraries and Packages

We chose Python as our programming language because of its familiarity and the number of tools available. Below is a list of all packages and libraries that we used to help us complete our research.

- Numpy - Allowed us to manipulate the data working alongside Pandas and to plot our data in conjunction with Seaborn and Matplotlib.

- Pandas - Allowed us to view our data in an organized manner.
- Seaborn - Allowed us to visualize our data in multiple ways, including heatmaps and line graphs.
- Matplotlib - Works with Seaborn to visualize data and add elements, such as legends and labels, to our visualizations.
- Sklearn - From this library, we used multiple packages, including the label encoder, standard scaler, confusion matrix, k-neighbor classifier, and gradient boosting classifier.
- Statsmodels - We used the variance-inflation-factor function from this library to measure each variable's multicollinearity.

These packages gave us the tools to preprocess the data, create and score models, and test feature importance.

Part III. Data Preparation

C1. Data Preprocessing Goal

Our primary focus during data preprocessing will be feature encoding. Random Forests only accepts numerical features in the form of integers and float-point numbers, and most of the features in our data are categorical variables. For this reason, besides checking for duplicates, missing values, outliers, and feature scaling, most of our data preprocessing will be focused on encoding categorical features using methods including Sklearn's LabelEncoder and Pandas' dummy function.

C2. Variables

We will use thirty-eight explanatory variables and one target variable, churn, for our classification model. Below is a list of the input variables classified as continuous or categorical.

Continuous	**Categorical**	**Categorical**	**Categorical**
Population	Area	OnlineBackup	Options
Children	Marital	DeviceProtection	RespectfulResponse
Age	Gender	TechSupport	CourteousExchange
Income	Techie	StreamingTV	ActiveListening
Outage_sec_perweek	Contract	StreamingMovies	Churn
Email	Port_modem	PaperlessBilling	
Contacts	Tablet	PaymentMethod	
Yearly_equip_failure	InternetService	TimelyResponse	
Tenure	Phone	TimelyFixes	
MonthlyCharge	Multiple	TimelyReplacements	
Bandwidth_GB_Year *(target)*	OnlineSecurity	Reliability	

C3. Data Preparation

C3.1. Import and Review

We began our data preparation by importing and reviewing the data using the Pandas info function. The function provides us with the number of variables, their data types, and how many non-null values are in each variable. From the output, we saw that we had 50 integer, float, and object variables; none were missing values, with a total of 10,000 observations.

```
## Import data
df = pd.read_csv('churn_clean.csv').reset_index(drop=True)
## Review shape and data types
df.info()
```

C3.2. Drop and Rename Variables

There were a few variables we considered to be unnecessary, and they included CaseOrder, CustomerID, Interaction, and UID. We also found some variables redundant based on how granular we wanted to observe location patterns. We kept the Population and Area variables but dropped the City, State, County, Zip code, Lat, Lng, and TimeZone variables.

We also dropped the Job variable, which we deemed redundant since the job would only provide insight into a customer's potential income and usage patterns. Since we already have a variable for the customer's income and yearly bandwidth usage, we did not deem it necessary to keep the Job variable, which also has a significantly high cardinality. The following step was to rename the last eight variables of survey responses to ensure they were descriptive of the data they represent.

```
## Drop granular data
df.drop([
    'CaseOrder',
    'Customer_id',
    'Interaction',
    'UID',
    'City',
    'State',
    'County',
    'Zip',
    'Lat',
    'Lng',
    'TimeZone',
    'Job'
], axis=1, inplace=True)
## Rename non-descript variables
df.rename({
    'Item1':'TimelyResponse',
    'Item2':'TimelyFixes',
    'Item3':'TimelyReplacements',
    'Item4':'Reliability',
    'Item5':'Options',
    'Item6':'RespectfulResponse',
    'Item7':'CourteousExchange',
    'Item8':'ActiveListening'
}, axis=1, inplace=True)
```

C3.3. Treat Duplicate Variables and Outliers

This step involved checking for duplicates and keeping just one of the duplicate observations, as we would only want one observation per customer. When implementing the duplicated function from Pandas, we found no duplicate values in our data. We then normalized our data before testing it for outliers using sklearn's StandardScaler function. The first step of this process was to isolate all of the string variables into another data frame, then scale the integer and float-point variables. Once our data was normalized, we removed observations with an absolute value greater than three. An observation with an absolute value greater than three is considered three standard deviations away from the means, thus, deemed an outlier. Once we detected all outliers, we evaluated the data loss, and because they comprised less than ten percent of our data, we opted to remove them.

```
## Check for duplicate values
print('Duplicate Values Found:', df.duplicated().sum())
## Isolate string variables from numeric variables
object_df = pd.DataFrame()
for col in df.columns:
    if df[col].dtype == object:
        object_df[col] = df[col]
        df = df.drop(col, axis=1)
## Scale the data
scaler = StandardScaler()
df = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
## Detect outliers
df[df.abs() > 3].dropna(how='all')
## Remove observations with absolute zscore over 3
df = df[df.abs() < 3].dropna()
## Evaluate data loss
lost = ((len(object_df) - len(df)) / len(object_df)) * 100
print('Data Lost: {}%\nData Kept: {}%'.format(lost, 100-lost))
## Drop same outlier observations from object data frame
object_df = object_df.loc[df.index]
```

C3.4. Encode Categorical Variables

As we mentioned earlier, most of our data consists of categorical variables. Before using these variables in our model, we encoded them into numeric variables using sklearn's LabelEncoder function. We began by transforming the Contract variable using a dictionary for the contract length in years. Then we looped through each variable and encoded those containing 'Yes' and 'No' into binary. We then used Pandas to get dummy variables for those remaining, Area, Marital, Gender, InternetService, and PaymentMethod.

```
## Define dictionary for contract variable
contract = {
    'Month-to-month': 0,
    'One year': 1,
    'Two Year': 2
}
## Encode the contract variable and drop it from object df
df['Contract'] = object_df['Contract'].map(contract)
object_df.drop('Contract', axis=1, inplace=True)
## Instantiate the label encoder
le = LabelEncoder()
## Loop through each variable to encode it
for col in object_df.columns:
    if 'Yes' in object_df[col].values:
        df[col] = le.fit_transform(object_df[col])
        object_df.drop(col, axis=1, inplace=True)
object_df = pd.get_dummies(object_df)
df[object_df.columns] = object_df
```

C4. Save to CSV

The clean data set used for our Random Forests model can be found in “*rf_clean.csv*.”

Part IV. Analysis

D1. Split the Data

The training data set used for our Random Forests model can be found in “*rf_train.csv*.”
The test data set used for our Random Forests model can be found in “*rf_test.csv*.”

D2. Data Analysis

The random forests regression model analyzes the data set by building decision trees and combining their predictions to make a single prediction for each data point. The model can be used to predict numerical values based on input features and can be tuned to improve performance.

D2.1. Model Testing

We began by testing our model with all input variables in the data set. We used the predictions to get the mean squared score, plot the true and predicted values, and calculate the correlation coefficient.

```
## Instantiate random forest model
rf = RandomForestRegressor(random_state=10)
```

```
## Fit the model
rf.fit(X_train, y_train)
```

```
RandomForestRegressor(random_state=10)
```

```
## Use the fitted model to make predictions
y_pred = pd.DataFrame(rf.predict(X_test), index=y_test.index, columns=y_test.columns)
```

```
## Score the quality of predictions
r2 = rf.score(X_test, y_test)
mse = mean_squared_error(y_test, y_pred)
print(f'R-Squared: {round(r2, 6)}\nMean Squared Error: {round(mse, 6)}')
```

```
R-Squared: 0.997885
Mean Squared Error: 0.00209
```

```
## Plot residuals
resid = y_scaler.inverse_transform(y_pred) - y_scaler.inverse_transform(y_test)
sns.scatterplot(data=resid)
plt.axhline(mse, c='red')
plt.title('Random Forest Regressor Residuals')
plt.xlabel('Observation')
plt.ylabel('Residual')
plt.legend([f'MSE: {round(mse, 6)}'], loc='lower right')
plt.show()
```

```
## Store true and predicted values in a data frame
res = pd.DataFrame(y_test.values, index=y_test.index, columns=['True'])
res['Predicted'] = y_pred
## Calculate the correlation coefficient
corr_coef = round(res.corr()['True']['Predicted'], 6)
```

```
## Plot the true and predicted values to view correlation
sns.scatterplot(res['True'], res['Predicted'])
plt.title('Correlation Between True and Predicted Values')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.show()
```

D2.2. Feature Importance

The random forests model provides us with the importance of the features used to predict bandwidth usage. When interpreting Gini importance values, it is important to note that the importance value of a feature is relative to that of the other features fed to the model. Tenure, InternetService_DSL, MonthlyCharge, Age, and Children were our model's five most important features. No customer survey responses appeared in the top ten feature importance values. It is important to note this, as it will help us and stakeholders to drill down into those specific features to identify underlying patterns within those variables.

```
## Store feature importance in a data frame
importance = pd.DataFrame(
    rf.feature_importances_,
    index=rf.feature_names_in_,
    columns=['gini']
).sort_values('gini', ascending=False)
```

```
## Review the top ten most important features
importance.head(20)
```

D2.3. Feature Selection

Once we understood the importance of each feature, we used step-forward feature selection to find the best set of features for our reduced model. This step recursively added one feature at a time by order of importance.

```
## Test models using each feature
scores = pd.DataFrame(columns=['r2', 'mse', 'features'])
mod = RandomForestRegressor(random_state=10)
for i in range(1, len(importance)):
    features = importance.index[:i]
    mod.fit(X_train[features], y_train)
    pred = mod.predict(X_test[features])
    scores.loc[i-1] = r2_score(y_test, pred), mean_squared_error(y_test, pred), features.values
```

```
## Print results
scores = scores.sort_values('mse').reset_index(drop=True)
print(f'Initial Model MSE: {round(mse, 6)}\nBest Model MSE: {round(scores.mse[0], 6)}\n\nFeatures:')
for feature in scores['features'][0]:
    print(feature)
print(f'\nTotal Features: {len(scores.features[0])}')
```

D2.4. Check for Multicollinearity

After we found the set of features in the model with the lowest MSE, we calculated their variance inflation factor to determine their multicollinearity.

```
## Calculate the variance inflation factor
for i in range(len(features)):
    print(f'{features[i]}: {round(vif(X[features], i), 3)}')
```

```
Tenure: 1.0
InternetService_DSL: 1.291
MonthlyCharge: 1.209
Age: 1.001
Children: 1.004
StreamingTV: 1.46
```

D2.5. Predict Future Bandwidth Usage

Our last step was to predict future bandwidth usage. We began by isolating active customers who have yet to churn and increasing their tenure by 12 months. We then used that data to predict bandwidth usage.

```
## Define X and y of active customers
X = X[X['Churn'] == 0]
y = y.loc[X.index]
```

```
## Increase tenure by 12 months
future_X = X
future_X['Tenure'] = X['Tenure'] + 12
```

```
## Instantiate random forest model
rf = RandomForestRegressor(random_state=10)
```

```
## Fit the random forest model
rf.fit(X[features], y)
```

```
RandomForestRegressor(random_state=10)
```

```
## Predict using the fitted model
y_pred = pd.DataFrame(index=y.index)
y_pred['Current Bandwidth'] = y_scaler.inverse_transform(y)
y_pred['Bandwidth +12 Months'] = y_scaler.inverse_transform(rf.predict(future_X[features]).reshape(-1,1))
```

D3. Random Forests Analysis Code

```
## Instantiate random forest model

rf = RandomForestRegressor(random_state=10)

## Fit the model

rf.fit(X_train, y_train)

## Use the fitted model to make predictions

y_pred = pd.DataFrame(rf.predict(X_test), index=y_test.index, columns=y_test.columns)

## Score the quality of predictions

r2 = rf.score(X_test, y_test)

mse = mean_squared_error(y_test, y_pred)

print(f'R-Squared: {round(r2, 6)}\nMean Squared Error: {round(mse, 6)}')

## Store true and predicted values in a data frame

res = pd.DataFrame(y_test.values, index=y_test.index, columns=['True'])

res['Predicted'] = y_pred

## Calculate the correlation coefficient

corr_coef = round(res.corr()['True']['Predicted'], 6)

Feature Selection

## Store feature importance in a data frame

importance = pd.DataFrame(

   rf.feature_importances_,

   index=rf.feature_names_in_,

   columns=['gini']

).sort_values('gini', ascending=False)

## Review the top ten most important features
```

```
importance.head(20)
## Test models using each feature
scores = pd.DataFrame(columns=['r2', 'mse', 'features'])
mod = RandomForestRegressor(random_state=10)
for i in range(1, len(importance)):
    features = importance.index[:i]
    mod.fit(X_train[features], y_train)
    pred = mod.predict(X_test[features])
    scores.loc[i-1] = r2_score(y_test, pred), mean_squared_error(y_test, pred), features.values
## Print results
scores = scores.sort_values('mse').reset_index(drop=True)
print(f'Initial Model MSE: {round(mse, 6)}\nBest Model MSE: {round(scores.mse[0], 6)}\n\nFeatures:')
for feature in scores['features'][0]:
    print(feature)
print(f'\nTotal Features: {len(scores.features[0])}')
## Define reduced features
features = scores.features[0]
Check for Multicollinearity
## Calculate the variance inflation factor
for i in range(len(features)):
    print(f'{features[i]}: {round(vif(X[features], i), 3)}')
Predict Future Usage
## Define X and y of active customers
X = X[X['Churn'] == 0]
y = y.loc[X.index]
## Increase tenure by 12 months
future_X = X
future_X['Tenure'] = X['Tenure'] + 12
```

```
## Instantiate random forest model
rf = RandomForestRegressor(random_state=10)
## Fit the random forest model
rf.fit(X[features], y)
## Predict using the fitted model
y_pred = pd.DataFrame(index=y.index)
y_pred['Current Bandwidth'] = y_scaler.inverse_transform(y)
y_pred['Bandwidth +12 Months'] = y_scaler.inverse_transform(rf.predict(future_X[features]).reshape(-1,1))
## Calculate the difference in bandwidth usage
resid = y_pred['Current Bandwidth'] - y_pred['Bandwidth +12 Months']
```

Part V. Data Summary and Implications

E1. Accuracy and MSE

We measured the accuracy of our random forest models using the R-Squared and MSE values. R-Squared measures how well the features explain the variation in the target variable. A higher R-squared value means the model explains the variance in the data well. We began with an initial R-Squared value of 99.79%, compared to our reduced model's 99.83%. To calculate the model's MSE, we got the average squared sum of the difference between the true and predicted values. The lower the MSE value, the better the model performs. Our initial model had an MSE of 0.0020, compared to our reduced model's MSE of 0.0018. Below is the comparison between the initial and reduced model performance and the features used in the reduced model to predict bandwidth usage.

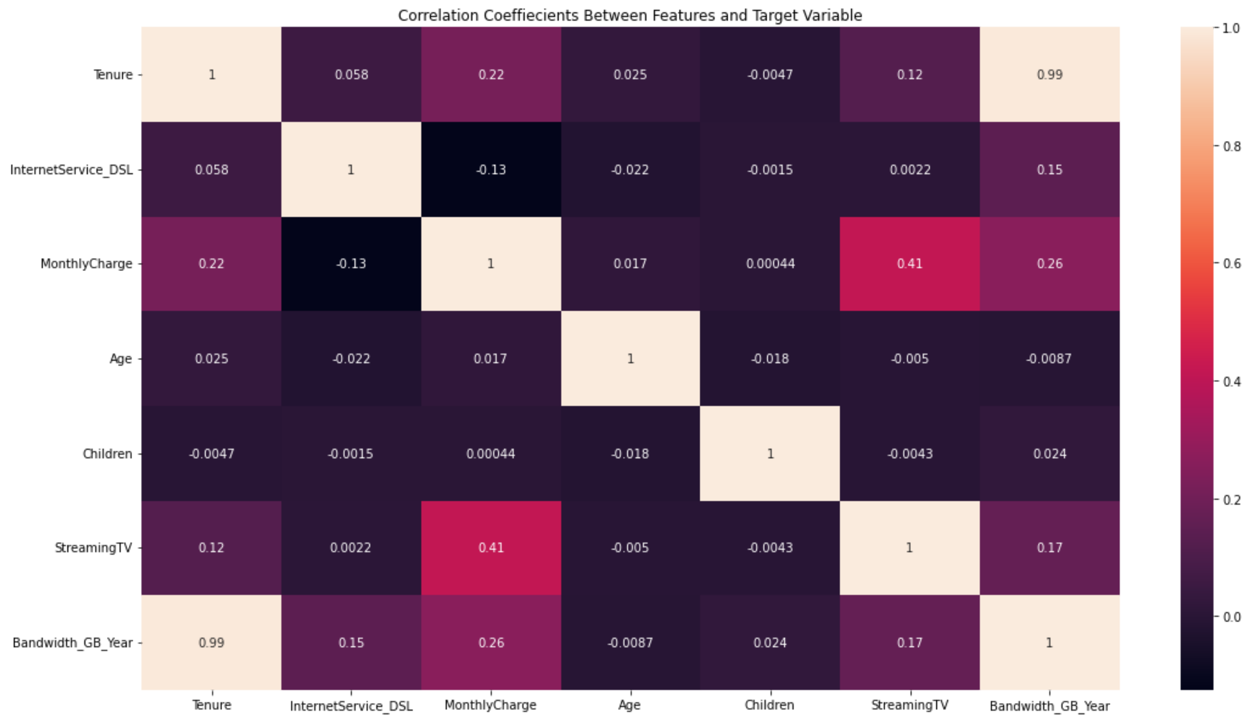
```
Initial Model R-Squared: 0.997982
Best Model R-Squared: 0.998253

Initial Model MSE: 0.002022
Best Model MSE: 0.001751

Features:
Tenure
InternetService_DSL
MonthlyCharge
Age
Children
StreamingTV

Total Features: 6
```

E2. Results and Implications

We created our random forest regression model to predict the average bandwidth usage for each customer for the following year. Through an analysis process of measuring feature importance, multicollinearity, and model performance, we created a well-performing model capable of predicting future bandwidth usage with 99.83% accuracy. Our results showed that, on average, each customer would use 1.21 GB of bandwidth more than the previous year. Looking further into the features used to predict future usage, we can see that the customer's time with the company and the type of internet they subscribe to are the two most important factors. Other important factors are the customer's monthly charge, age, the number of children in their household, and whether they subscribe to TV streaming services.

To further understand our results, we visualized a heatmap below of the reduced model's features and our target variable. The heatmap shows a strong positive relationship between tenure and annual bandwidth usage. There is also a positive correlation between monthly charges, children, customers subscribed to DSL, TV streaming, and annual bandwidth usage, meaning as one goes up, so does the other.

There is only one negative relationship, and that is between age and annual bandwidth usage. The negative relationship implies that customers use less bandwidth as they age.



E3. Limitations

The most impactful limitation in our analysis was the computing cost of tuning hyperparameters for random forest models. We can tune random forest hyperparameters to find the best values for the *n_estimators, max_depth, min_samples_split, min_samples_leaf,* and *max_features.* Iterating through the multiple possible values for each hyperparameter took significant time and resources and thus was not part of the analysis. To improve upon the model, I would recommend as the next steps focus on hyperparameter tuning.

E4. Recommendations

After analyzing current annual bandwidth usage and predicting what it would be like a year from now, I recommend that the telecom company focus on attracting younger customers and parents. The focus on younger bandwidth consumers will ensure the company has a higher likelihood of adding on streaming services like TV and movie streaming. Additionally, I recommend that the company strategize customer retention methods, as longer-term customers tend to have higher annual bandwidth usage.

Part VI. Demonstration

G. Panopto Video

The Panopto video is linked in the project submission.

H. Code Sources

We did not use third-party code to create this project.

I. Citation Sources

We did not use in-text citations in this project.