Random Forest Background: Churn Prediction Model Tutorial

Hello, my name is Molly Miraglia and I’m self teaching myself machine learning, specifically, Random Forest. This is a **work in progress.** I’m lucky I have a background in phylogenetics in which I experienced Random Forest in academic research journals/presentations and lab discussion. As I go further into my career, I want to use machine learning for exploratory data analysis and business analytics. I have a strong background in Python, R, SQL, and Tableau and it would be easy to apply these skills to using Random Forest/other machine learning techniques and present my findings to make data driven decisions that can ultimately benefit all stakeholders. I’m using an R Markdown, as I’m most familiar with it, but also I can incorporate code from Python and SQL. I’m writing my understanding and knowledge of Random Forest, not only for others to understand, but also for myself. I feel like once you can explain something in simple terms, you understand the material well.

I also wanted to focus on a Churn Prediction model using Random Forest, as this is very applicable the type of work I want to do. I am borrowing heavily from the work of Natassha Selvaraj, as she wrote a wonderful, detailed article for 365 DataScience that gave an in-depth tutorial for a Churn Prediction model in Python. I also am commenting code and writing out my process as I go along so I can further understand the Random Forest workflow. I am also thankful to the countless Youtube and Google searches I’ve watched for better understanding of the Random Forest concepts.

## TL;DR

I’m bragging about my background. Below is my own Random forest explanation from what I’ve learned online and a Churn Prediction model from an online tutorial.

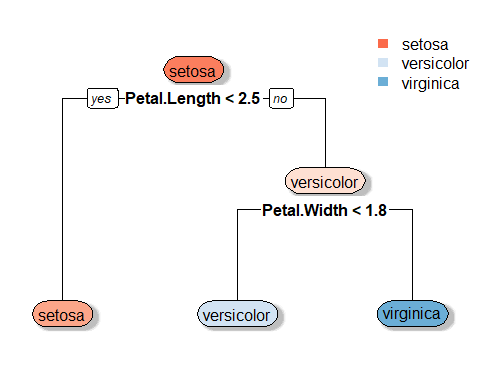
## Inspiration and Data Source:

* [365 DataScience Tutorial by Natassha Selvaraj](https://365datascience.com/tutorials/python-tutorials/how-to-build-a-customer-churn-prediction-model-in-python/)
* [Kaggle](https://www.kaggle.com/datasets/blastchar/telco-customer-churn)

## What do I know About Random Forest?

### Let’s start with Decision Trees

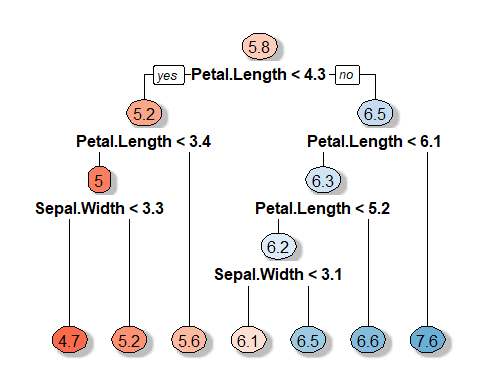
A random forest is composed of an ensemble of decision trees, which are models that look like a flow chart, but they actually help make predictions based on data. A decision tree makes an initial statement, known as the root, and then can branch off to different nodes based on a decision. There are two types of nodes, the internal nodes, which split the data into smaller subsets, and then there are leaf nodes, which is an outcome or prediction and does not further split the data. I’ll show an [example](https://www.spsanderson.com/steveondata/posts/2023-09-29/index.html) with the famous Iris (flower) dataset.



The decision tree created shows the decisions leading to accurately naming the species of Iris based on petal measurements. If we selected an Iris from the dataset used to create this decision tree, that had a petal length of 2.8 cm and a petal width of 1.9 cm, we would start at the root node of the decision tree where we would make our first decision about the petal length. Since the iris petal length is greater than 2.5 cm, we can move onto the right branch of the decision tree. This internal node then splits the data, and asks about petal width. Since the petal width is greater than 1.8, we can move to the right leaf node, which determines the species is ‘virginica’. If the petal length was less than 2.5, the data would not be needed to be split into further categorization, we could move to the left side of the branch, and get the outcome ‘setosa’.

The type of decision trees that random forest is based on is CART, which stands for Classification and Regression Trees. Classification means that the decision tree is splitting based on discrete or categorical data, and the leaf node will be a category. Categorical trees are usually found by finding the Gini Impurity or measure of Entropy. The plot above is considered a classification tree, as the leaf nodes are classifying what species the Iris is.

A regression tree means that each leaf node will be a numeric value, and is split based on a numeric measure. Regression trees are usually found using variance reduction or the mean squared error. The plot below is a regression tree which would predict the Sepal Length of a given Iris. If we selected an Iris from the dataset used to create this decision tree, that had a petal length of 4.6 cm and sepal width of 2.9 cm, we would start at the root, which is 5.8 (average sepal length of entire dataset). Since the given sepal length is NOT less than 4.3, we would move to the right branch of the tree. At the 6.5 internal node, we would move to left branch (4.6 < 6.1), and then at the 6.3 internal node, we would move to the left branch again (4.6 < 5.2). At the 6.2 internal node, we would then evaluate the sepal width (2.9 < 3.1), and move to the 6.1 leaf node.



Decision trees work well with the data used to create them (training data), but can be highly sensitive to any changes in training data. Notice in my examples, I said ‘selected an Iris from the dataset used to create [the] decision tree’. If we took an Iris from outside this dataset, it would not fit this model. The outside dataset has different distributions and characteristics, and the decision tree could not generalize the new data correctly. There is also the problem of a decision tree overfitting the training data. That means the decision tree captured noise and specific patterns from the training data and those patterns and noise would skew a decision made with new data. Decision trees are a good way to form decisions if you’re working with one dataset, however usually in research and other business intelligence roles, there will never be just one dataset. This is why we use Random Forest!

In it’s simplest form, Random Forest creates an ensemble of decision trees. New data is ran through ALL of the decision trees created and the most popular outcome is the best prediction. Now, there are a bunch of steps in between, but the goal of random forest is to improve prediction accuracy and reduce overfitting.

Will continue writing (bootstrapping, aggregation, bagging etc.) **Work in progress**

## Machine Learning Workflow

I am borrowing this [Machine Learning Workflow](https://towardsdatascience.com/random-forest-in-python-24d0893d51c0) (MLW) roadmap.

1. State the question and determine required data
2. Acquire the data in an accessible format
3. Identify and correct missing data points/anomalies as required
4. Prepare the data for the machine learning model
5. Establish a baseline model that you aim to exceed
6. Train the model on the training data
7. Make predictions on the test data
8. Compare predictions to the known test set targets and calculate performance metrics
9. If performance is not satisfactory, adjust the model, acquire more data, or try a different modeling technique
10. Interpret model and report results visually and numerically

I don’t think the tutorial will go all the way to number 8, but I will probably reference this MLW when conducting my own analyses after I’m more comfortable.

## Let’s get started with the 365Data Science Tutorial

### MLW Step 1

Following the Machine Learning Workflow, we must first state the question and determine required data. Luckily, we are working with given data from the tutorial. The main question in a Churn Prediction Model is usually, what is causing customers to Churn? We can look at the attributes of the data and depending on patterns in the data, we can try and accurately predict new things that will make a customer churn, and fix these issues.

### MLW Step 2

We can now move onto step 2 in our machine learning worfklow and can read in the dataset from [Kaggle](https://www.kaggle.com/datasets/blastchar/telco-customer-churn). This dataset is customer information of a telephone company, including demographics, customer account information, services that the customer signed up for, and the Churn status. Churn means if a customer unsubscribes/stops using a business’s product. For this dataset, the Churn status is if a customer stopped doing business *in the last month*. There are no dates for the churn, it’s just a Boolean of yes/no.

import pandas as pd  
import io  
  
df = pd.read\_csv('Customer\_Churn.csv')  
  
# Capture the output of df.info()  
buffer = io.StringIO()  
df.info(buf=buffer)  
info\_str = buffer.getvalue()  
  
# Extract column, non-null count, and data type information  
info\_lines = info\_str.split('\n')[2:-3]  
info\_data = [line.split()[:3] for line in info\_lines]  
info\_data = [[line.split()[1], line.split()[-1]] for line in info\_lines]  
  
  
# Create a DataFrame from the extracted information  
info\_df = pd.DataFrame(info\_data, columns=['Column Name', 'Data Type'])

Table 1. Churn Dataset Column Names

| **Column Name** | **Data Type** |
| --- | --- |
| customerID | object |
| gender | object |
| SeniorCitizen | int64 |
| Partner | object |
| Dependents | object |
| tenure | int64 |
| PhoneService | object |
| MultipleLines | object |
| InternetService | object |
| OnlineSecurity | object |
| OnlineBackup | object |
| DeviceProtection | object |
| TechSupport | object |
| StreamingTV | object |
| StreamingMovies | object |
| Contract | object |
| PaperlessBilling | object |
| PaymentMethod | object |
| MonthlyCharges | float64 |
| TotalCharges | object |
| Churn | object |

We can see all the column names and data types. We know there are 19 independent variables that are used to predict customer churn. Now we will see what percentage of customers of the dataset have churned.

df["Churn"].value\_counts()

Churn No 5174 Yes 1869 Name: count, dtype: int64

Now using the value\_counts…:

## The percent of churned customers in the data set is 27.0%

According to the tutorial, we are now dealing with an imbalanced classification problem, since not all customers in the dataset have churned. Feature engineering will be used later to address this problem by creating a balanced dataset before performing Random Forest.

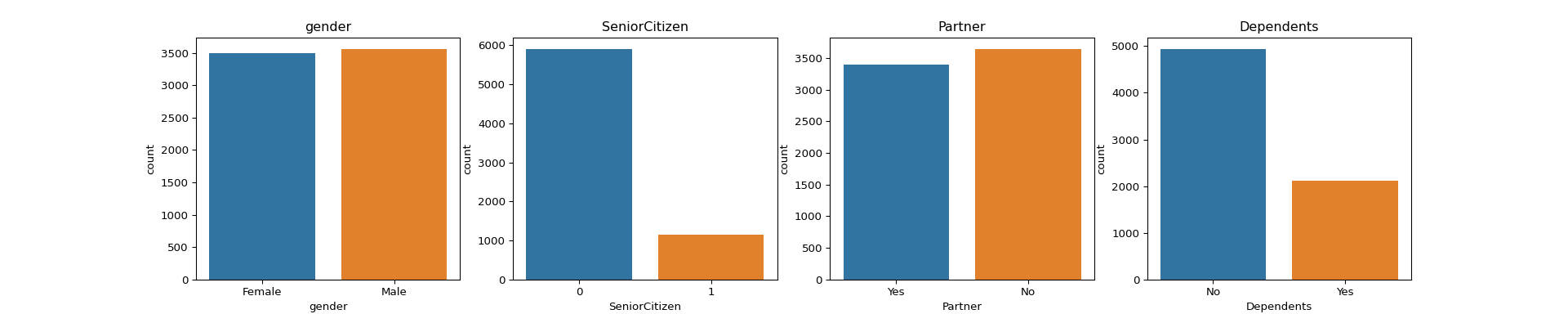
### MLW Step 3

The tutorial now moves to exploratory data analysis, and I will count this part as step 3 of the Machine Learning Workflow. I’m assuming that the data has no anomalies (since this is from Kaggle dataset has been specifically crafted just for Churn Prediction Model purposes). Below is code I could use if I was using a fancy IDE, which gives me all the exploratory analysis I need in one line using the ydata\_profiling library. For now, I will just follow the tutorial!

#from ydata\_profiling import ProfileReport  
  
#profile = ProfileReport(df, title = "Churn Profiling Report")

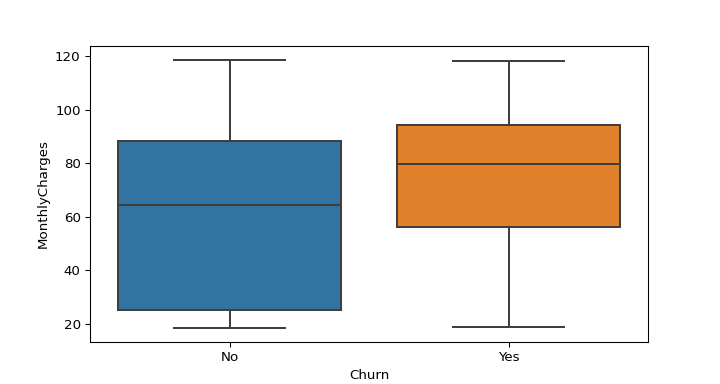
We are using exploratory analysis to gain a better understanding of the relationship between the column names/independent variables with customer churn.

import matplotlib.pyplot as plt  
import seaborn as sns  
import numpy as np  
  
cols = ['gender','SeniorCitizen',"Partner","Dependents"]  
numerical = cols  
  
plt.figure(figsize=(20,4))  
  
for i, col in enumerate(numerical):  
 ax = plt.subplot(1, len(numerical), i+1)  
 sns.countplot(x=str(col), data=df)  
 ax.set\_title(f"{col}")

 I could make more descriptive plots using R, but for the sake of the tutorial, I’ll keep the plots as is. We can see the trends in these plots show that there most customers in the dataset are younger and are mostly without dependents. We can see there is almost an equal amount of females and males, and marital status.

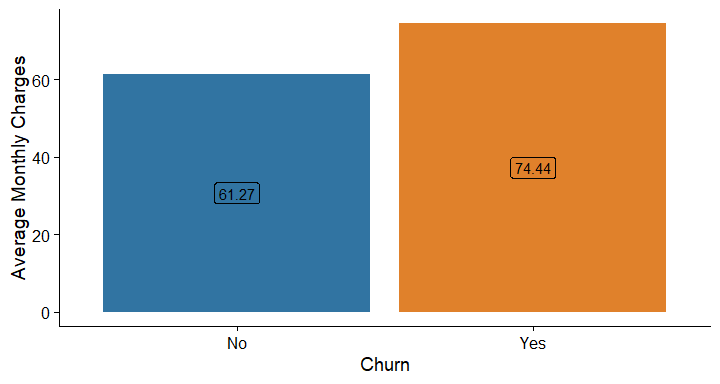
Personally for me, and the people I know, cost is a huge factor in why people unsubscribe from a product. Using a boxplot, we can see if this is reflected in the dataset.

sns.boxplot(x='Churn', y='MonthlyCharges', data=df)



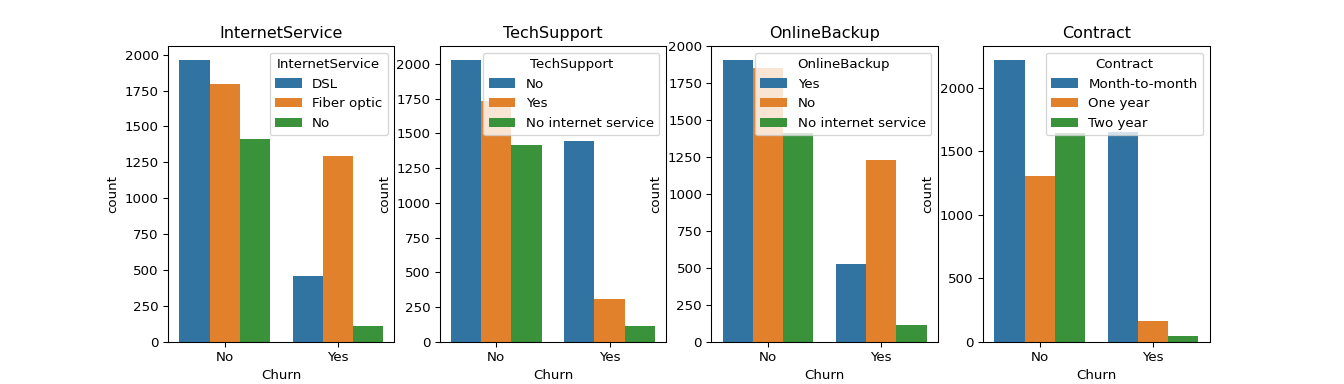
From this box plot, we can see that customers who have churned have a higher median of monthly charges than customers who have not churned. We can also do this in R to compare averages instead of median (just to be fancy lol)

df <- py$df  
  
avg\_mc <- df %>%   
 group\_by(Churn) %>%  
 summarize(`Average Monthly Charges` = round(mean(MonthlyCharges), digits = 2))  
  
avg\_mc %>% ggplot(aes(x = Churn, y = `Average Monthly Charges`, fill = Churn, label = `Average Monthly Charges`)) +  
 geom\_col() +  
 geom\_label(position = position\_stack(vjust = 0.5)) +  
 scale\_fill\_manual(values = c("#3174A2", "#E0812B")) +  
 theme\_cowplot() +  
 theme(legend.position = "none")



In the tutorial, we’re now looking at other attributes to evaluate their relationship to customer churn.

cols = ['InternetService',"TechSupport","OnlineBackup","Contract"]  
  
plt.figure(figsize=(14,4))  
  
for i, col in enumerate(cols):  
 ax = plt.subplot(1, len(cols), i+1)  
 sns.countplot(x ="Churn", hue = str(col), data = df)  
 ax.set\_title(f"{col}")



Again, I could do this in R, but for time’s sake, I will move on. We can see for Internet Service, customers with fiber optic cables churn more than others. Customers who have opted for no tech support also churn more than others. Customers who have no online backup service churn more than those who do. Customers who churn are on a month to month contract. We can see in these trends from the data visualization alone, that there are trends and reasons why a customer might churn. The tutorial recommends from this analysis, a company could offer tech support for free to prevent customers from leaving.

### MLW Step 4

We are now on step 4 of the Machine Learning Workflow of preprocessing the data for the machine learning model. According to the tutorial, there are three steps in the process; cleaning the dataset, encoding categorical variables, and oversampling.

For cleaning the data, the tutorial notices that the data type in Table 1. for TotalCharges is an object, but it should be numeric. We will now convert it to numeric.

df['TotalCharges'] = df['TotalCharges'].apply(lambda x: pd.to\_numeric(x, errors='coerce')).dropna()

Encoding categorical varaibles just means making the categorical variables into numeric format so they can be easily digested in our Random Forest model. So for example, turning “Female” into 0 and “Male” into 1. The code below uses Scikit-Learn’s label encoder.

from sklearn import preprocessing   
  
cat\_features = df.drop(['customerID','TotalCharges','MonthlyCharges','SeniorCitizen','tenure'],axis=1)  
cat\_features.head()

## gender Partner Dependents ... PaperlessBilling PaymentMethod Churn  
## 0 Female Yes No ... Yes Electronic check No  
## 1 Male No No ... No Mailed check No  
## 2 Male No No ... Yes Mailed check Yes  
## 3 Male No No ... No Bank transfer (automatic) No  
## 4 Female No No ... Yes Electronic check Yes  
##   
## [5 rows x 16 columns]

le = preprocessing.LabelEncoder()  
df\_cat = cat\_features.apply(le.fit\_transform)  
df\_cat.head()

## gender Partner Dependents ... PaperlessBilling PaymentMethod Churn  
## 0 0 1 0 ... 1 2 0  
## 1 1 0 0 ... 0 3 0  
## 2 1 0 0 ... 1 3 1  
## 3 1 0 0 ... 0 0 0  
## 4 0 0 0 ... 1 2 1  
##   
## [5 rows x 16 columns]

We’ll now merge the encoded dataframe with the original.

num\_features = df[['customerID','TotalCharges','MonthlyCharges','SeniorCitizen','tenure']]  
finaldf = pd.merge(num\_features, df\_cat, left\_index=True, right\_index=True)

The tutorial now moves onto oversampling. Since only 27% of the customers in this dataset churned, that means there is a class imbalance problem, which means if we built a model using this data without preprocessing it by oversampling, the model would only be correct 73% (100 - 27) of the time. Oversampling is increasing the representation of the minority (Churn customers) to address the imbalance. The tutorial doesn’t explicitly say this, but I believe we’re using bootstrapping to fix the imbalance. In the tutorial it states that oversampling ‘involves randomly selecting samples from the minority class and additing it to the training dataset’, so it sounds kind of similar to bootstrapping. Before oversampling, the tutorial is doing a split in the dataset for testing and training data. This way the test dataset will be a representative of the true population.

from sklearn.model\_selection import train\_test\_split  
  
finaldf = finaldf.dropna()  
finaldf = finaldf.drop(['customerID'],axis=1)  
  
X = finaldf.drop(['Churn'],axis=1)  
y = finaldf['Churn']  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)

Now that the training dataset is created, oversampling can occur.

from imblearn.over\_sampling import SMOTE  
  
oversample = SMOTE(k\_neighbors=5)  
X\_smote, y\_smote = oversample.fit\_resample(X\_train, y\_train)  
X\_train, y\_train = X\_smote, y\_smote

We can now check the number of samples in each class to see if they are equal.

y\_train.value\_counts()

## Churn  
## 0 3452  
## 1 3452  
## Name: count, dtype: int64

There are 3,452 values in each class, which indicates a balanced dataset.

### MLW Step 5

We already have established the machine learning model we’re going to use, which will be random forest!

### MLW Step 6

We can now write the code for our Random Forest, and train the model on our training dataset. It’s funny how you need all this background knowledge to just write two lines of simple code! See my comments in the code below on the random state argument for the RandomForestClassifier function.

from sklearn.ensemble import RandomForestClassifier  
  
rf = RandomForestClassifier(random\_state=42)  
# I changed the tutorial's random state from 46 to 42. Random state just ensures that no matter how many times I run this code, the results will remain the same. I changed it to 42 as that's usually the default for Scikit  
rf.fit(X\_train,y\_train)

## RandomForestClassifier(random\_state=42)

### MLW Step 7

Now that the Random Forest model is fitted to our training data, we can evaluate predictions on the test dataset.

from sklearn.metrics import accuracy\_score  
  
preds = rf.predict(X\_test)  
print(round(100 \*accuracy\_score(preds,y\_test),2))

## 77.47

So what this means, is that the model created has a 77.42% accuracy rate of correctly predicting Churn. If we fed in new data into this model, it would take all variables into consideration and predict if a customer would churn accurately 77.42% of the time.

## What Have I Learned?

I’ve learned a lot of background in Machine Learning! I also learned the steps of actually making a random forest model, and how to use it! Will write more later… **Work in Progress**