[Churn Prediction with Machine Learning](https://towardsdatascience.com/churn-prediction-with-machine-learning-ca955d52bd8c" \l ":~:text=Churn%20prediction%20is%20a%20common%20use%20case%20in%20machine%20learning%20domain.&text=It%20is%20very%20critical%20for,customers%20from%20leaving%20the%20company.)

A step-by-step explanation of a machine learning project.

[[](https://sonery.medium.com/?source=post_page-----ca955d52bd8c--------------------------------)](https://sonery.medium.com/?source=post_page-----ca955d52bd8c--------------------------------)

[Soner Yıldırım](https://sonery.medium.com/?source=post_page-----ca955d52bd8c--------------------------------)

Follow

[Apr 13](https://towardsdatascience.com/churn-prediction-with-machine-learning-ca955d52bd8c?source=post_page-----ca955d52bd8c--------------------------------) · 12 min read

Churn prediction is a common use case in machine learning domain. If you are not familiar with the term, churn means “leaving the company”. It is very critical for a business to have an idea about why and when customers are likely to churn. Having a robust and accurate churn prediction model helps businesses to take actions to prevent customers from leaving the company.



Photo by [Chris Liverani](https://unsplash.com/@chrisliverani?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText) on [Unsplash](https://unsplash.com/s/photos/customer?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText)

In this project, I will use “Telco Customer Churn” dataset which is available on Kaggle.

There are 20 features (independent variables) and 1 target (dependent) variable for 7043 customers. Target variable indicates if a customer has left the company (i.e. churn=yes) within the last month. Since the target variable has two states (yes/no or 1/0), this is a binary classification problem.

The variables are: 'customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'.

At first glance, only customerID seems irrelevant to customer churn. Other variables may or may not have an effect on customer churn. We will figure out.

The project is structured as follows:

1. Exploratory Data Analysis
2. Data Preprocessing
3. Model Creation and Evaluation
4. Improving the Model

**1. Exploratory Data Analysis**

Let’s start with importing the required libraries and then the dataset:

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
%matplotlib inline#Dataset  
df = pd.read\_csv("Telco-Customer-Churn.csv")  
df.shape  
(7043, 21)

I always look for missing values and try to handle them. The dataset we are using is pre-cleaned so I think there is no missing value. Let’s check just to make sure:

df.isna().sum().sum()  
0

There is no missing value in the data set so we can jump to explore it. We can start with the target variable:

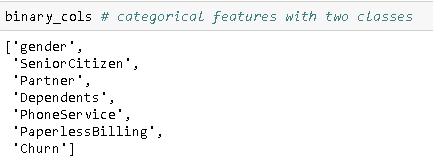
df.Churn.value\_counts()  
No 5174  
Yes 1869

Target variable has imbalanced class distribution. Positive class (Churn=Yes) is much less than negative class (churn=No). Imbalanced class distributions influence the performance of a machine learning model negatively. We will use upsampling or downsampling to overcome this issue.

It is always beneficial to explore the features (independent variables) before trying to build a model. Let’s first discover the features that only have two values.

columns = df.columns  
binary\_cols = []for col in columns:  
 if df[col].value\_counts().shape[0] == 2:  
 binary\_cols.append(col)

Image for post



The remaining categorical variables have more than two values (or classes).

Image for post

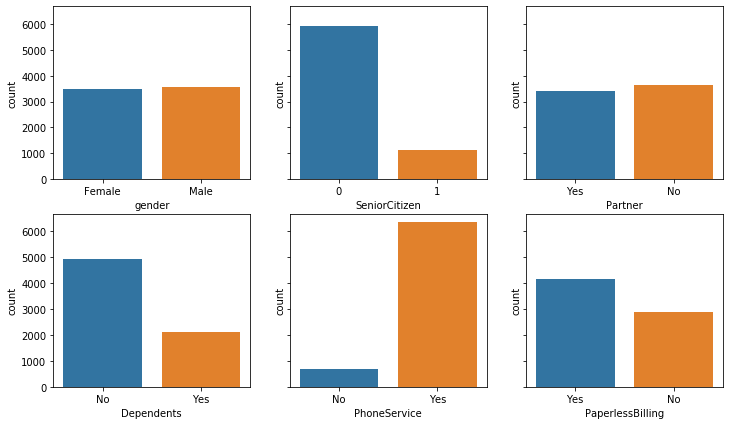


**Binary Categorical Features**

Let’s check the class distribution of binary features.

fig, axes = plt.subplots(2, 3, figsize=(12, 7), sharey=True)sns.countplot("gender", data=df, ax=axes[0,0])  
sns.countplot("SeniorCitizen", data=df, ax=axes[0,1])  
sns.countplot("Partner", data=df, ax=axes[0,2])  
sns.countplot("Dependents", data=df, ax=axes[1,0])  
sns.countplot("PhoneService", data=df, ax=axes[1,1])  
sns.countplot("PaperlessBilling", data=df, ax=axes[1,2])

Image for post



There is a high imbalance in SeniorCitizen and PhoneService variables. Most of the customers are not senior and similarly, most customers have a phone service.

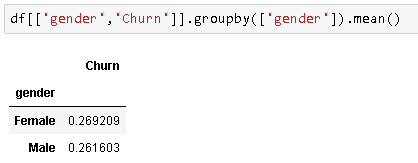
It is better to check how the target variable (churn) changes according to the binary features. To be able to make calculations, we need to change the values of target variable. “Yes” will be 1 and “No” will be 0.

churn\_numeric = {'Yes':1, 'No':0}  
df.Churn.replace(churn\_numeric, inplace=True)

Let’s see if churn rate is different for males and females:

df[['gender','Churn']].groupby(['gender']).mean()

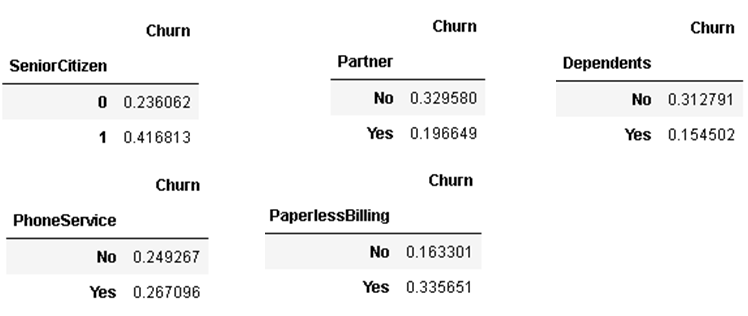
Image for post



Average churn rate for males and females are approximately the same which indicates gender variable does not bring a valuable prediction power to a model. Therefore, I will not use gender variable in the machine learning model.

Similarly, we can check other binary categorical features in terms of churn rate:

Image for post

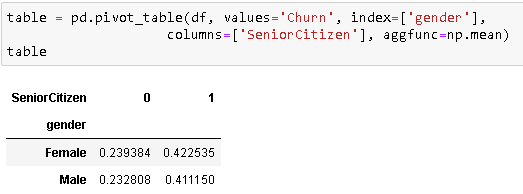


The other binary features have an effect on the target variable. The phone service may also be skipped if you think 2% difference can be ignored. I have decided to use this feature in the model.

We can also use pandas pivot\_table function to check the relationship between features and target variable.

table = pd.pivot\_table(df, values='Churn', index=['gender'],  
 columns=['SeniorCitizen'], aggfunc=np.mean)  
table

Image for post



**Other Categorical Features**

It is time to explore other categorical features. We also have continuous features such as tenure, monthly charges and total charges which I will discuss in the next part.

**Internet Service**

There are 6 variables that come with internet service which are StreamingTV, StreamingMovies, OnlineSecurity, OnlineBackup, DeviceProtection and TechSupport. There variables come into play if customer has internet service.

sns.countplot("InternetService", data=df)

Image for post

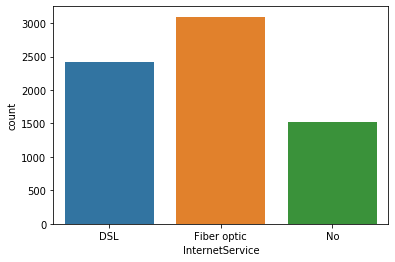
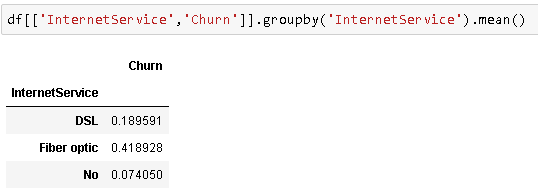
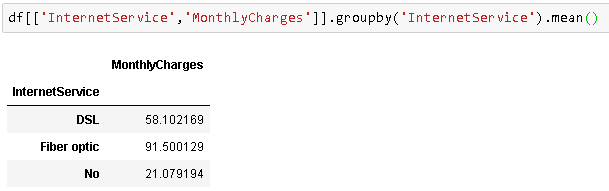


Image for post



Internet service variable is definitely important in predicting churn rate. As you can see, customers with fiber optic internet service are much likely to churn than other customers although there is not a big difference in the number of customers with DSL and fiber optic. This company may have some problems with fiber optic connection. However, it is not a good way to make assumptions based on only one variable. Let’s also check the monthly charges.

Image for post



Fiber optic service is much more expensive than DSL which may be one of the reasons why customers churn.

We can now check the distributions of internet service related variables:

fig, axes = plt.subplots(2, 3, figsize=(12, 7), sharey=True)sns.countplot("StreamingTV", data=df, ax=axes[0,0])  
sns.countplot("StreamingMovies", data=df, ax=axes[0,1])  
sns.countplot("OnlineSecurity", data=df, ax=axes[0,2])  
sns.countplot("OnlineBackup", data=df, ax=axes[1,0])  
sns.countplot("DeviceProtection", data=df, ax=axes[1,1])  
sns.countplot("TechSupport", data=df, ax=axes[1,2])

Image for post

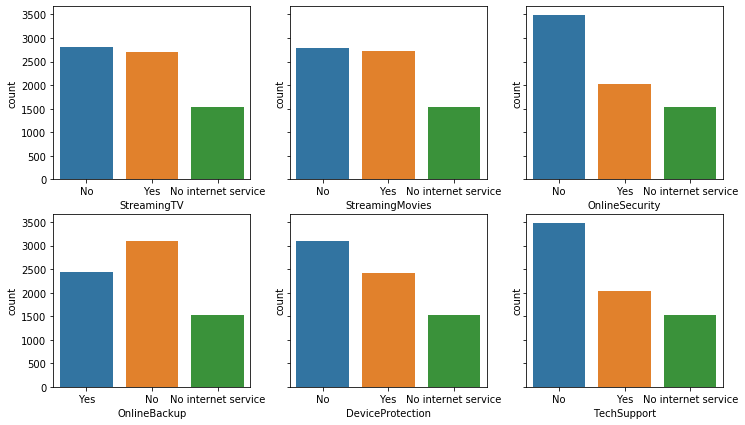
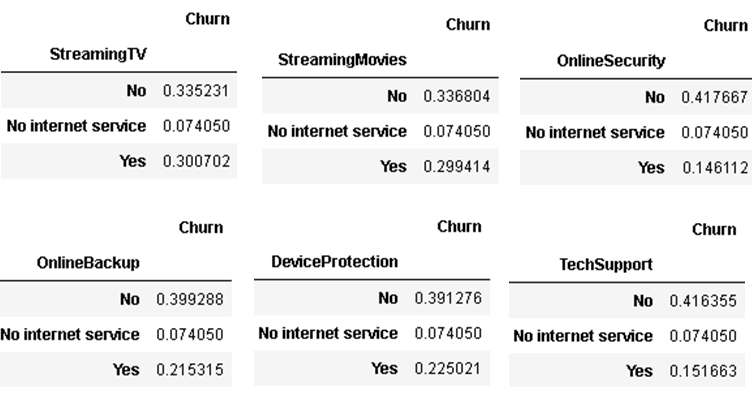


Image for post



All internet service related features seem to have different churn rates for their classes because churn rate changes according to customers having these services. The difference on StreamingTV and StreamingMovies are not much but they can still bring value to the model. You may decide not to include these two features.

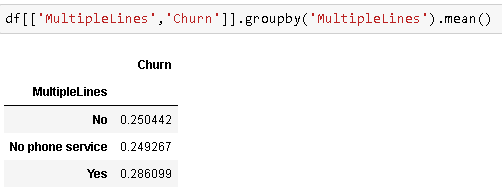
**Phone Service**

df.PhoneService.value\_counts()  
Yes 6361  
No 682df.MultipleLines.value\_counts()  
No 3390  
Yes 2971  
No phone service 682

If a customer does not have a phone service, he/she cannot have multiple lines. MultipleLines column includes more specific data compared to PhoneService column. So I will not include PhoneService column as I can understand the number of people who have phone service from MultipleLines column. MultipleLines column takes the PhoneService column one step further.

Let’s also check if having multiple lines changes the churn rate:

Image for post



It is similar to StreamingTV and StreamingMovies variables so it is up to you to take advantage of these variables in the model. I will include them in the model.

**Contract and Payment Method**

plt.figure(figsize=(10,6))  
sns.countplot("Contract", data=df)

Image for post

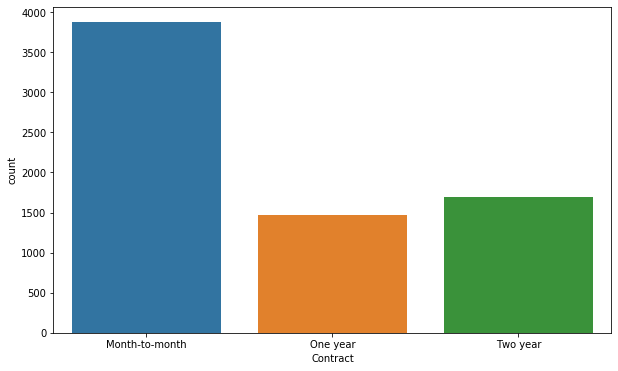
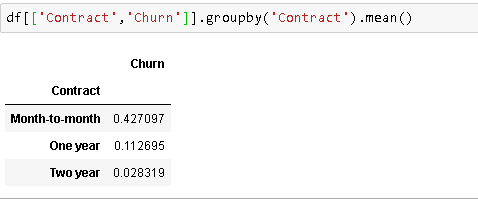


Image for post



It seems like, as expected, customers with short-term contract are more likely to churn. This clearly explains the motivation for companies to have long-term relationship with their customers.

plt.figure(figsize=(10,6))  
sns.countplot("PaymentMethod", data=df)

Image for post

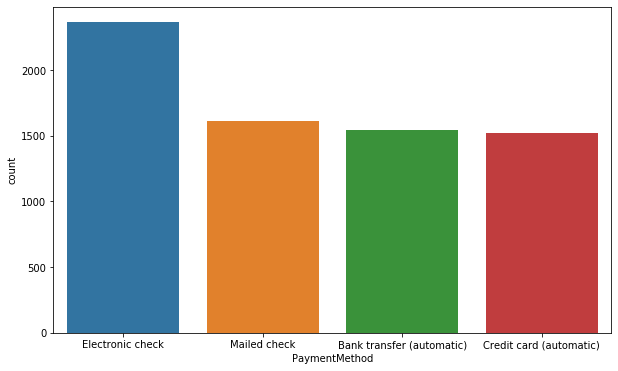
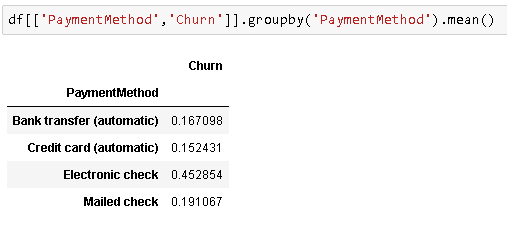


Image for post



Customers who pay with electronic check are more likely to churn and this kind of payment is more common than other payment types. Therefore, this segment may be further investigated if customers paying with electronic checks have any other thing in common.

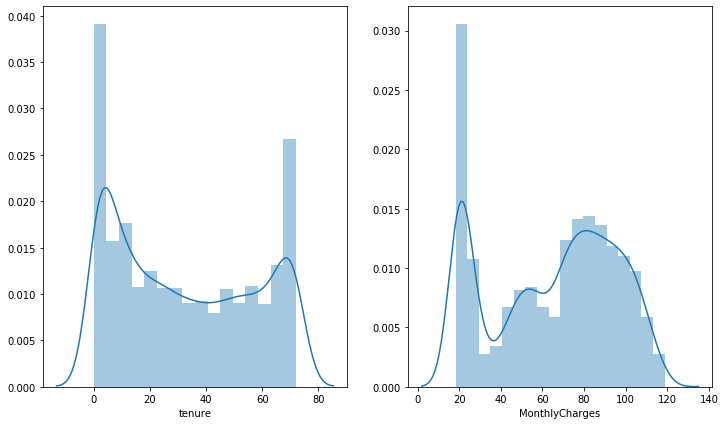
**Continuous Features**

The continuous features are tenure, monthly charges and total charges. The amount in total charges column is proportional to tenure (months) multiplied by monthly charges. So it is unnecessary to include total charges in the model. Adding unnecassary features will increase the model complexity. It is better to have a simpler model when possible. Complex models tend to overfit and not generalize well to new, previously unseen observations. Since the goal of a machine learning model is to predict or explain new observations, overfitting is a crucial issue.

Let’s also have a look at the distribution of continuous features.

fig, axes = plt.subplots(1,2, figsize=(12, 7))sns.distplot(df["tenure"], ax=axes[0])  
sns.distplot(df["MonthlyCharges"], ax=axes[1])

Image for post



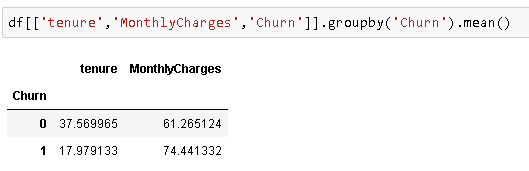
***Tenure****: Number of months the customer has stayed with the company.*

According to the distribution of tenure variable, most of the customers are either pretty new or have stayed for a long time with the company. Our goal should be finding a way to keep those customers with a tenure of up to a few months.

A similar trend is seen on MonthlyCharges. There seems to be a gap between low rates and high rates.

Let’s check how churn rate changes according to tenure and MonthlyCharges:

Image for post

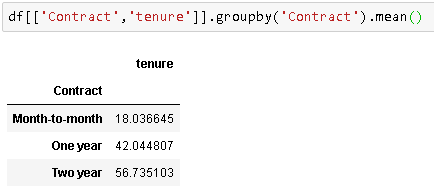


It is clear that people who have been a customer for a long time tend to stay with the company. The average tenure in months for people who left the company is 20 months less than the average for people who stay.

It seems like monthly charges also have an effect on churn rate.

Contract and tenure features may be correlated because customer with long term contract are likely to stay longer with the company. Let’s figure out.

Image for post



As expected, contract and tenure are highly correlated. Customers with long contracts have been a customer for longer time than customers with short-term contracts. I think contract will add little to no value to tenure feature so I will not use contract feature in the model.

After exploring the variables, I have decided not to use following variable because they add little or no informative power to the model:

* Customer ID
* Gender
* PhoneService
* Contract
* TotalCharges

df.drop([‘customerID’,’gender’,’PhoneService’,’Contract’,’TotalCharges’], axis=1, inplace=True)

**2. Data Preprocessing**

Categorical features need to be converted to numbers so that they can be included in calculations done by a machine learning model. The categorical variables in our data set are not ordinal (i.e. there is no order in them). For example, “DSL” internet service is not superior to “Fiber optic” internet service. An example for an ordinal categorical variable would be ratings from 1 to 5 or a variable with categories “bad”, “average” and “good”.

When we encode the categorical variables, a number will be assigned to each category. The category with higher numbers will be considered more important or effect the model more. Therefore, we need to do encode the variables in a way that each category will be represented by a column and the value in that column will be 0 or 1.

We also need to scale continuous variables. Otherwise, variables with higher values will be given more importance which effects the accuracy of the model.

from sklearn.preprocessing import LabelEncoder, OneHotEncoder  
from sklearn.preprocessing import MinMaxScaler

Encoding categorical variables:

cat\_features = ['SeniorCitizen', 'Partner', 'Dependents',  
'MultipleLines', 'InternetService','OnlineSecurity' 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',  
'StreamingMovies', 'PaperlessBilling', 'PaymentMethod']X = pd.get\_dummies(df, columns=cat\_features, drop\_first=True)

Scaling continuous variables:

sc = MinMaxScaler()  
a = sc.fit\_transform(df[['tenure']])  
b = sc.fit\_transform(df[['MonthlyCharges']])X['tenure'] = a  
X['MonthlyCharges'] = b

Let’s check the new dimension of the dataset:

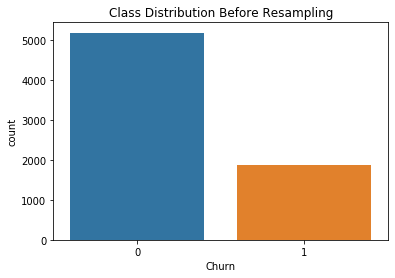
X.shape  
(7043, 26)

**Resampling**

As we briefly discussed in the beginning, target variables with imbalanced class distribution is not desired for machine learning models. I will use upsampling which means increasing the number of samples of the class with less samples by randomly selecting rows from it.

sns.countplot('Churn', data=df).set\_title('Class Distribution Before Resampling')

Image for post



Separating positive class (churn=yes) and negative class (churn=no):

X\_no = X[X.Churn == 0]  
X\_yes = X[X.Churn == 1]

Upsampling the positive class:

X\_yes\_upsampled = X\_yes.sample(n=len(X\_no), replace=True, random\_state=42)print(len(X\_yes\_upsampled))  
5174

Combining positive and negative class and checking class distribution:

X\_upsampled = X\_no.append(X\_yes\_upsampled).reset\_index(drop=True)sns.countplot('Churn', data=X\_upsampled).set\_title('Class Distribution After Resampling')

Image for post



**3. Model Creation and Evaluation**

We need to divide the dataset into training and test subsets so that we are able to measure the performance of our model on new, previously unseen examples.

from sklearn.model\_selection import train\_test\_splitX = X\_upsampled.drop(['Churn'], axis=1) #features (independent variables)y = X\_upsampled['Churn'] #target (dependent variable)

Dividing dataset into train and test subsets:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state=42)

**Ridge Classifier**

I have decided to use ridge classifier as a base model. Then I will try a model that I think will perform better.

from sklearn.linear\_model import RidgeClassifier  
from sklearn.metrics import accuracy\_score

Creating a ridge classifier object and training it:

clf\_ridge = RidgeClassifier() #create a ridge classifier object  
clf\_ridge.fit(X\_train, y\_train) #train the model

Making predictions on training set and evaluating:

pred = clf\_ridge.predict(X\_train)accuracy\_score(y\_train, pred)  
0.7574293307562213

Making predictions on test set and evaluating:

pred\_test = clf\_ridge.predict(X\_test)accuracy\_score(y\_test, pred\_test)  
0.7608695652173914

The model achieved 75% accuracy on training set and 76% accuracy on test set. The model is not overfitting because accuracies on training and test sets are pretty close. However, 75% accuracy is not very good so we will try to get a better accuracy using a different model.

**Random Forest**

from sklearn.ensemble import RandomForestClassifier

Creating a random forest object and training it:

clf\_forest = RandomForestClassifier(n\_estimators=100, max\_depth=10)clf\_forest.fit(X\_train, y\_train)

There are two parameters we need to mention here.

* n\_estimators: The number of trees in the forest.
* max\_depth: The maximum depth of the tree.

These parameters have critical roles in the accuracy of model and also preventing the model from overfitting. In general, if we use deep trees (max\_depth is very high), the model may end up overfitting.

Making predictions on training set and evaluating:

pred = clf\_forest.predict(X\_train)accuracy\_score(y\_train, pred)  
0.8860835950712732

Making predictions on test set and evaluating:

pred\_test = clf\_forest.predict(X\_test)accuracy\_score(y\_test, pred\_test)  
0.842512077294686

The accuracy on training set is 4% higher than the accuracy on test set which indicates a slight overfitting. We can decrease the depth of a tree in the forest because as trees get deeper, they tend to be more specific which results in not generalizing well. However, reducing tree depth may also decrease the accuracy. So we need to be careful when optimizing the parameters. We can also increase the number of trees in the forest which will help the model to be more generalized and thus reduce overfitting. Parameter tuning is a very critical part in almost every project.

Another way is to do cross-validation which allows to use every sample in training and test set.

**4. Improving the Model**

**GridSearchCV** provides an easy way for parameter tuning. We can do cross-validation and try different parameters using GridSearchCV.

from sklearn.model\_selection import GridSearchCV

Creating a GridSearchCV object:

parameters = {'n\_estimators':[150,200,250,300], 'max\_depth':[15,20,25]}forest = RandomForestClassifier()clf = GridSearchCV(estimator=forest, param\_grid=parameters, n\_jobs=-1, cv=5)

cv = 5 means having a 5-fold cross validation. So dataset is divided into 5 subset. At each iteration, 4 subsets are used in training and the other subset is used as test set. When 5 iteration completed, the model used all samples as both training and test samples.

n\_jobs parameter is used to select how many processors to use. -1 means using all processors.

clf.fit(X, y)

Let’s check the best parameters and overall accuracy:

clf.best\_params\_  
{'max\_depth': 20, 'n\_estimators': 150}clf.best\_score\_  
0.8999806725937379

*best\_score\_ : Mean cross-validated score of the best\_estimator.*

We have achieved an overall accuracy of almost 90%. This is the mean cross-validated score of the best\_estimator. In the previous random forest, the mean score was approximately 86% (88% on training and 84% on test). Using GridSearchCV, we improved the model accuracy by 4%.

We can always try to improve the model. The fuel of machine learning models is data so if we can collect more data, it is always helpful in improving the model. We can also try a wider range of parameters in GridSearchCV because a little adjustment in a parameter may slighlty increase the model.

Finally, we can try more robust or advanced models. Please keep in mind that there will be a trade-off when making such kind of decisions. Advanced models may increase the accuracy but they require more data and more computing power. So it comes down to business decision.

You can download the complete notebook of this project from this [repo](https://github.com/SonerYldrm/Churn-Prediction).

Thank you for reading. Please let me know if you have any feedback.