CUSTOMER CHURN PREDICTION

Abstract:

Customer churn analysis is the process of predicting customers who tend to cancel the service (subscription) they receive for various reasons, especially in sectors such as telecommunications, finance and insurance, and determining the necessary operational steps to prevent this cancellation.

During churn prediction, you're also:

- Identifying at-risk customers,
- Identifying customer pain points,
- Identifying strategy/methods to lower churn and increase customer retention.

Introduction:

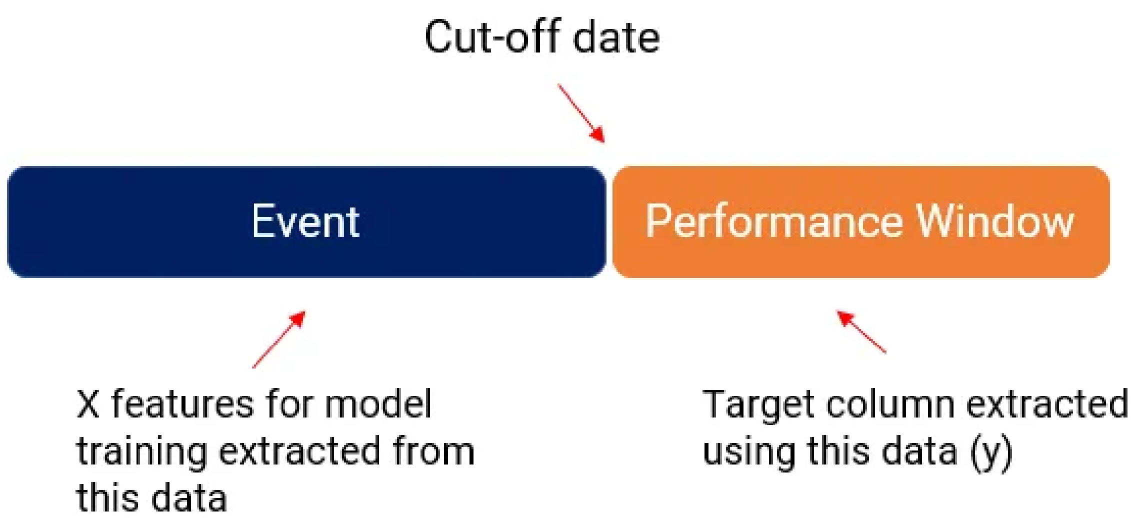
Customer retention is one of the primary KPI for companies with a subscription-based business model. Competition is tough particularly in the SaaS market where customers are free to choose from plenty of providers. One bad experience and customer may just move to the competitor resulting in customer churn.

Proposed system for churn prediction:

The proposed system in this research uses a combination of random forest, gravitational search algorithms, and differential evolution algorithms to predict customer churn.

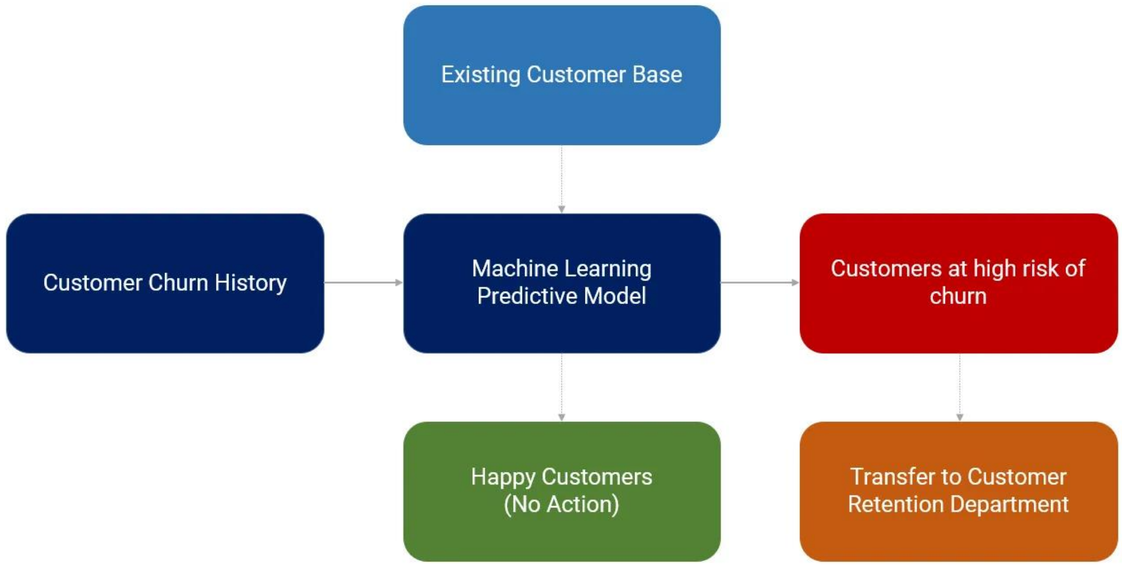
Customer Churn machine learning model used in practice:

There are two broad concepts to understand here:



create customer churn dataset

Customer Churn Model Workflow:

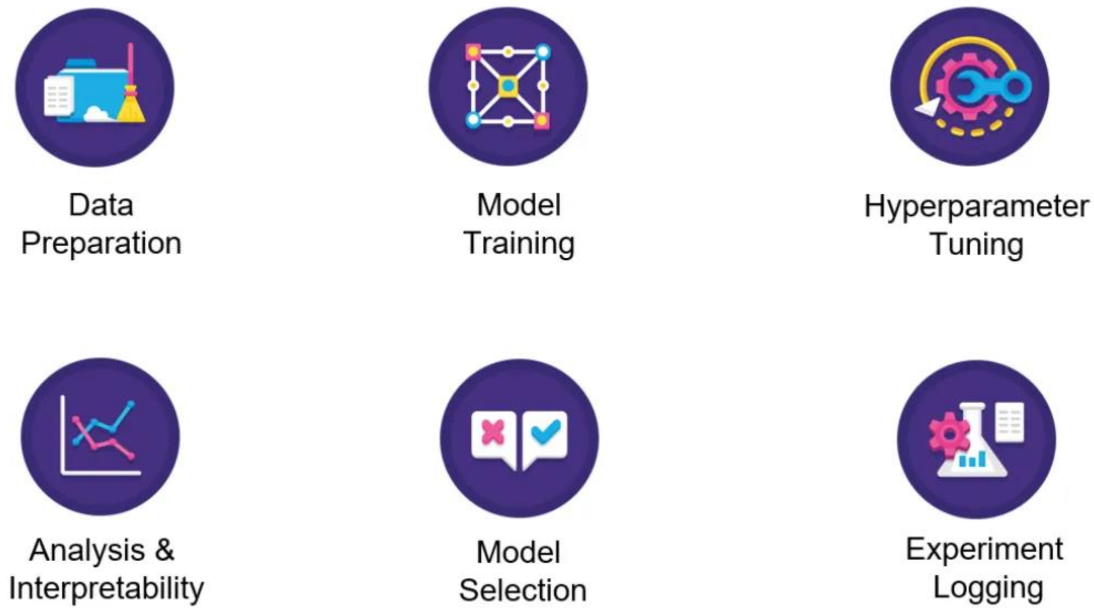


Customer Churn Model Workflow

Let's get started with the practical example

PyCaret:

PyCaret is an open-source, low-code machine learning library and end-to-end model management tool built-in Python for automating machine learning workflows. PyCaret is known for its ease of use, simplicity, and ability to quickly and efficiently build and deploy end-to-end machine learning pipelines.



Features of PyCaret

Install PyCaret

install pycaret

pip install pycaret

Dataset:

There are 17 categorical features:

CustomerID: Customer ID unique for each customer

gender: Whether the customer is a male or a female

SeniorCitizen: Whether the customer is a senior citizen or not (1, 0)

Partner: Whether the customer has a partner or not (Yes, No)

Dependent: Whether the customer has dependents or not (Yes, No)

PhoneService: Whether the customer has a phone service or not (Yes, No)

MultipeLines: Whether the customer has multiple lines or not (Yes, No, No phone service)

InternetService: Customer's internet service provider (DSL, Fiber optic, No)

OnlineSecurity: Whether the customer has online security or not (Yes, No, No internet service)

OnlineBackup: Whether the customer has an online backup or not (Yes, No, No internet service)

DeviceProtection: Whether the customer has device protection or not (Yes, No, No internet service)

TechSupport: Whether the customer has tech support or not (Yes, No, No internet service)

StreamingTV: Whether the customer has streaming TV or not (Yes, No, No internet service)

StreamingMovies: Whether the customer has streaming movies or not (Yes, No, No internet service)

Contract: The contract term of the customer (Month-to-month, One year, Two years)

PaperlessBilling: The contract term of the customer (Month-to-month, One year, Two years)

PaymentMethod: The customer's payment method (Electronic check, Mailed check, Bank transfer (automatic), Credit card (automatic))

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

MonthlyCharges: The amount charged to the customer monthly

TotalCharges: The total amount charged to the customer

Churn: Whether the customer churned or not (Yes or No)

Exploratory Data Analysis

check data types

data.dtypes

```
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
dtype: object
```

Data types

Notice that TotalCharges is of an object type instead of float64. Upon investigation, I figured out there are some blank spaces in this column which has caused Python to force the data type as object

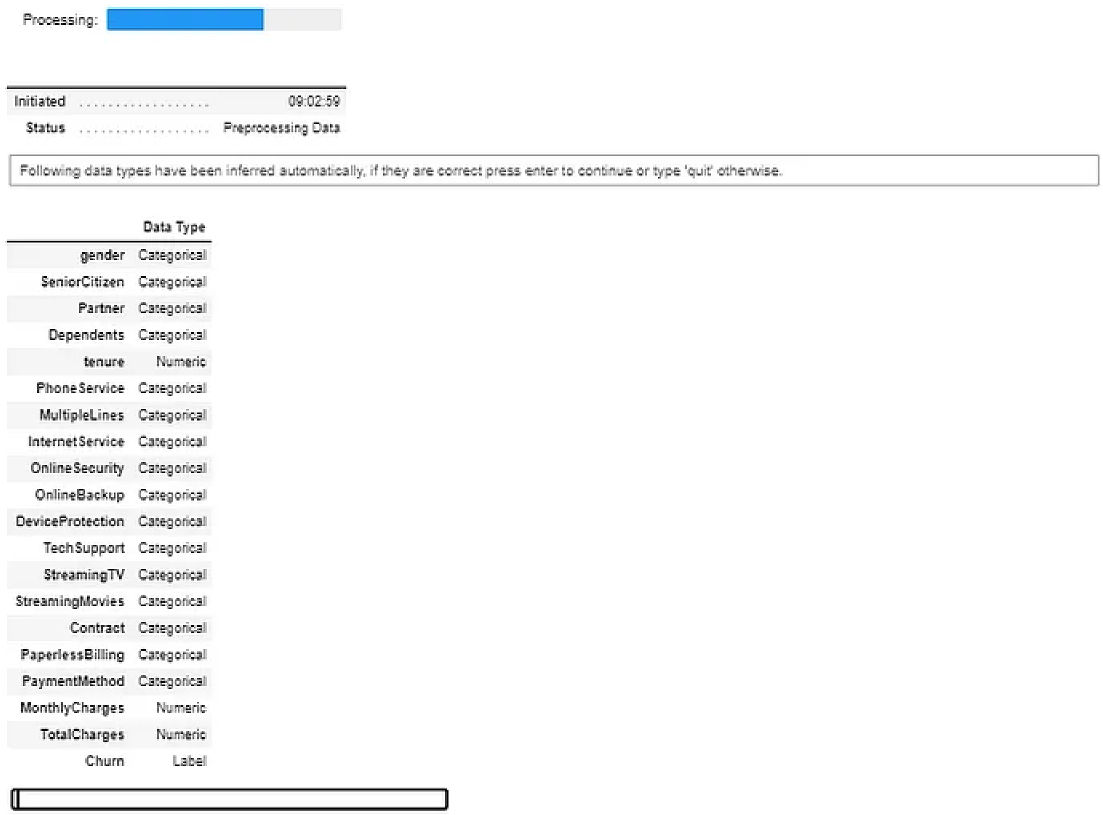
Data Preparation:

This function takes care of all the data preparation required prior to training models. Besides performing some basic default processing tasks, PyCaret also offers a wide array of pre-processing features

init setup

from pycaret.classification import *

s = setup(data, target = 'Churn', ignore_features = ['customerID'])



Processing:

Initiated	09:02:59
Status	Preprocessing Data

Following data types have been inferred automatically, if they are correct press enter to continue or type 'quit' otherwise.

	Data Type
gender	Categorical
SeniorCitizen	Categorical
Partner	Categorical
Dependents	Categorical
tenure	Numeric
PhoneService	Categorical
MultipleLines	Categorical
InternetService	Categorical
OnlineSecurity	Categorical
OnlineBackup	Categorical
DeviceProtection	Categorical
TechSupport	Categorical
StreamingTV	Categorical
StreamingMovies	Categorical
Contract	Categorical
PaperlessBilling	Categorical
PaymentMethod	Categorical
MonthlyCharges	Numeric
TotalCharges	Numeric
Churn	Label

setup function in pycaret.classification

Whenever you initialize the setup function in PyCaret, it profiles the dataset and infers the data types for all input features. In this case, you can see except for tenure MonthlyCharges and TotalCharges , everything else is categorical, which is correct, you can now press enter to continue. If data types are not inferred correctly (which can happen sometimes), you can use numeric_feature and categorical_feature to overwrite the data types.

Also, notice that I have passed ignore_features = ['customerID'] in the setup function so that it is not considered when training the models. The good thing about this is PyCaret will not remove the column from the dataset, it will just ignore it behind the scene for model training.

	Description	Value
0	session_id	598
1	Target	Churn
2	Target Type	Binary
3	Label Encoded	No: 0, Yes: 1
4	Original Data	(7043, 21)
5	Missing Values	True
6	Numeric Features	3
7	Categorical Features	16
8	Ordinal Features	False
9	High Cardinality Features	False
10	High Cardinality Method	None
11	Transformed Train Set	(4930, 33)
12	Transformed Test Set	(2113, 33)
13	Shuffle Train-Test	True
14	Stratify Train-Test	False

Output from setup — truncated for display

Model Training & Selection:

Now that data preparation is done, let's start the training process by using compare_models functionality. This function trains all the algorithms available in the model library and evaluates multiple performance metrics using cross-validation.

compare all models

best_model = compare_models(sort='AUC')

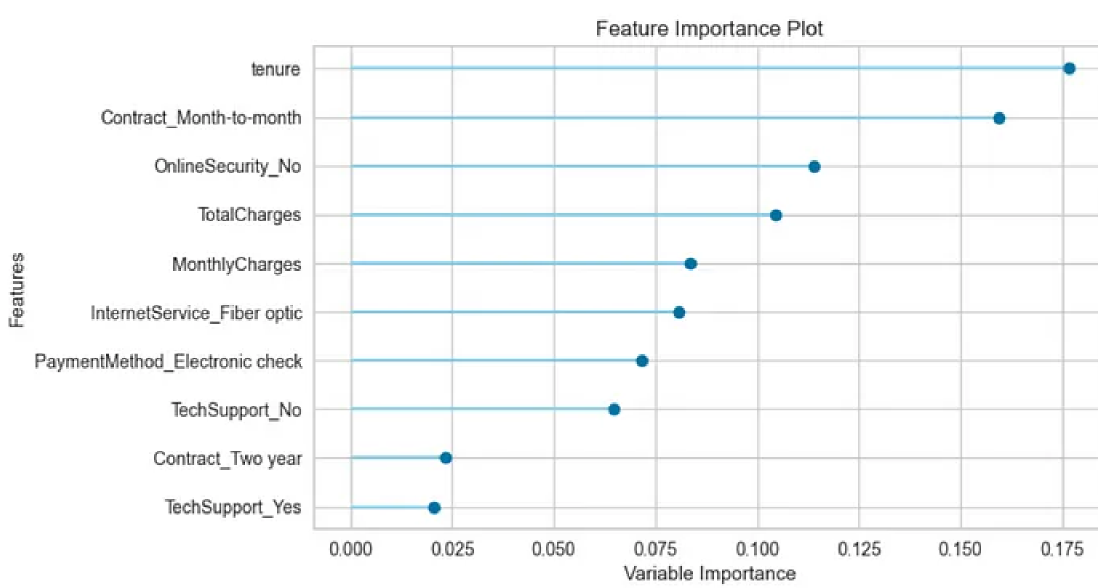
	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
gbc	Gradient Boosting Classifier	0.8002	0.8472	0.5086	0.6475	0.5690	0.4416	0.4474	0.1800
lr	Logistic Regression	0.8061	0.8431	0.5281	0.6594	0.5857	0.4613	0.4666	0.0280
ada	Ada Boost Classifier	0.7992	0.8409	0.5078	0.6449	0.5678	0.4395	0.4450	0.0640
catboost	CatBoost Classifier	0.7943	0.8376	0.5047	0.6312	0.5605	0.4285	0.4333	6.9900
lda	Linear Discriminant Analysis	0.7992	0.8368	0.5366	0.6356	0.5811	0.4505	0.4537	0.0110
lightgbm	Light Gradient Boosting Machine	0.7907	0.8323	0.5148	0.6176	0.5613	0.4254	0.4286	0.0480
nb	Naive Bayes	0.7347	0.8286	0.7777	0.4945	0.6042	0.4195	0.4441	0.0070
xgboost	Extreme Gradient Boosting	0.7878	0.8215	0.4985	0.6141	0.5492	0.4127	0.4171	0.3830
rf	Random Forest Classifier	0.7925	0.8193	0.4844	0.6321	0.5477	0.4163	0.4229	0.2420
et	Extra Trees Classifier	0.7730	0.7881	0.4610	0.5817	0.5133	0.3681	0.3729	0.2550
knn	K Neighbors Classifier	0.7606	0.7492	0.4298	0.5521	0.4827	0.3302	0.3350	0.0740
dt	Decision Tree Classifier	0.7323	0.6565	0.4953	0.4865	0.4903	0.3090	0.3093	0.0100
qda	Quadratic Discriminant Analysis	0.5655	0.6069	0.6929	0.3433	0.4522	0.1631	0.1941	0.0080
svm	SVM - Linear Kernel	0.7347	0.0000	0.4884	0.5776	0.4828	0.3257	0.3543	0.0180
ridge	Ridge Classifier	0.8041	0.0000	0.5039	0.6636	0.5718	0.4480	0.4558	0.0060

Output from compare_models Hyperparameter

Tuning

Feature Importance Plot

plot_model(tuned_gbc, plot = 'feature')



Adding Custom Metric in PyCaret:

Thanks to PyCaret, it is extremely easy to achieve this using add_metric function.

```
# create a custom function

def calculate_profit(y, y_pred):

    tp = np.where((y_pred==1) & (y==1), (5000-1000), 0)

    fp = np.where((y_pred==1) & (y==0), -1000, 0)

    return np.sum([tp,fp])

# add metric to PyCaret

add_metric('profit', 'Profit', calculate_profit)
```

Now let’s run compare_models and see the magic.

```
# compare all models

best_model =
compare_models(sort='Profit')
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	Profit	TT (Sec)
nb	Naive Bayes	0.7347	0.8286	0.7777	0.4945	0.6042	0.4195	0.4441	296500.0000	0.0100
lr	Logistic Regression	0.8061	0.8431	0.5281	0.6594	0.5857	0.4613	0.4666	235700.0000	0.9350
lda	Linear Discriminant Analysis	0.7992	0.8368	0.5366	0.6356	0.5811	0.4505	0.4537	235600.0000	0.0150
ridge	Ridge Classifier	0.8041	0.0000	0.5039	0.6636	0.5718	0.4480	0.4558	225400.0000	0.0120
gbc	Gradient Boosting Classifier	0.8002	0.8472	0.5086	0.6475	0.5690	0.4416	0.4474	225300.0000	0.2860
ada	Ada Boost Classifier	0.7992	0.8409	0.5078	0.6449	0.5678	0.4395	0.4450	224500.0000	0.1100
lightgbm	Light Gradient Boosting Machine	0.7907	0.8323	0.5148	0.6176	0.5613	0.4254	0.4286	223000.0000	0.0660
catboost	CatBoost Classifier	0.7943	0.8376	0.5047	0.6312	0.5605	0.4285	0.4333	220900.0000	8.6560
xgboost	Extreme Gradient Boosting	0.7878	0.8215	0.4985	0.6141	0.5492	0.4127	0.4171	215300.0000	0.5820
rf	Random Forest Classifier	0.7925	0.8193	0.4844	0.6321	0.5477	0.4163	0.4229	212200.0000	0.3000
et	Extra Trees Classifier	0.7730	0.7881	0.4610	0.5817	0.5133	0.3681	0.3729	193600.0000	0.2910
dt	Decision Tree Classifier	0.7323	0.6565	0.4953	0.4865	0.4903	0.3090	0.3093	186700.0000	0.0180
svm	SVM - Linear Kernel	0.7347	0.0000	0.4884	0.5776	0.4828	0.3257	0.3543	185200.0000	0.0290
qda	Quadratic Discriminant Analysis	0.5655	0.6069	0.6929	0.3433	0.4522	0.1631	0.1941	180400.0000	0.0130
knn	K Neighbors Classifier	0.7606	0.7492	0.4298	0.5521	0.4827	0.3302	0.3350	175500.0000	0.0780

Output from compare_models

Notice that a new column Profit is added this time and surprisingly Naive Bayes which is a pretty bad model in terms of AUC is the best model when it comes to profit.

Conclusion:

Using PyCaret provides many more solutions and functionalities, all in less time and effort! If you have any other examples or techniques you're curious about, drop them in the comments and we'll try to create one.