INTERNSHIP PROJECT REPORT

Submitted to
INLIGHNX GLOBAL PVT. LTD.
Bangalore, India

Date of Submission
15/01/26

Customer Churn Prediction System

Model: DecisionTreeClassifier, **RandomForestClassifier**, XGBClassifier

Prepared by

Nadar Deepthi Chenthil
AI & ML Intern
ITID5802

Internship Duration
15/11/25 – 15/01/26

Index

Link to GitHub with the .ipynb file containing the implementation code, all dataset .csv files and documentation:

https://github.com/Deepthi-Nadar/Customer-Churn-Prediction-System-Using-ML/blob/main

Customer Churn Prediction System

1. Introduction

Customer churn prediction is a critical business problem, especially in industries like **telecommunications, banking, SaaS, and subscription-based services**, where retaining existing customers is more cost-effective than acquiring new ones.

This project focuses on building a **Customer Churn Prediction System** using machine learning techniques. The system analyzes historical customer data and predicts whether a customer is likely to leave the service (churn) or continue.

The notebook `Customer_churn.ipynb` implements a **complete end-to-end ML pipeline**, starting from raw data loading to model saving for deployment.

Problem Definition

- **Input**: Customer demographic, service usage, and billing information
- **Output**: Churn prediction (Yes / No)

Type of Problem

- **Supervised Learning**
- **Binary Classification Problem**

Business Importance

- Helps companies identify at-risk customers
- Enables proactive retention strategies
- Reduces revenue loss

2. Libraries and Dependencies

The notebook uses the following Python libraries:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, cross_val_score
import pickle
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

- **pandas** – Data loading and manipulation
- **numpy** – Numerical operations
- **matplotlib / seaborn** – Data visualization
- **sklearn (scikit-learn)** –
    - Preprocessing (LabelEncoder, StandardScaler)
    - Model building (Logistic Regression / Random Forest / etc.)
    - Model evaluation (accuracy, confusion matrix, classification report)
- **pickle** – Saving and loading trained models and encoders

These libraries together form the backbone of the machine learning workflow.

3. Dataset Overview

The dataset contains customer information commonly found in telecom churn datasets.

Key Features

Feature	**Description**
gender	Customer gender
SeniorCitizen	Whether the customer is a senior citizen (0/1)
Partner	Whether customer has a partner
Dependents	Whether customer has dependents
tenure	Number of months the customer stayed
PhoneService	Phone service subscription
InternetService	Type of internet service
OnlineSecurity	Online security subscription
TechSupport	Tech support subscription
Contract	Contract type
MonthlyCharges	Monthly bill amount
TotalCharges	Total bill amount
Churn	Target variable

4. Data Loading

The dataset is loaded using the **pandas** library, which provides powerful data manipulation capabilities.

Steps Performed

1. Import the dataset using pd.read_csv()

```python
#importing the dataset

df=pd.read_csv("/content/WA_Fn-UseC_-Telco-Customer-Churn.csv")
```

Store the data in a pandas DataFrame

2. Perform initial inspection using:
   - df.head() – Displays first few rows

```python
df.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	St
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	No	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	No	Yes	No	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	No	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	No	Yes	Yes	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	No	No	No	

```python
df.head()
```

OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn
No	Yes	No	No	No	No	Month-to-month	Yes	Electronic check	29.85	29.85	No
Yes	No	Yes	No	No	No	One year	No	Mailed check	56.95	1889.5	No
Yes	Yes	No	No	No	No	Month-to-month	Yes	Mailed check	53.85	108.15	Yes
Yes	No	Yes	Yes	No	No	One year	No	Bank transfer (automatic)	42.30	1840.75	No
No	No	No	No	No	No	Month-to-month	Yes	Electronic check	70.70	151.65	Yes

   - df.shape – Shows number of rows and columns

```python
df.shape

(7043, 21)
```

   - df.info() – Displays data types and null values

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
```

```
4   Dependents        7043 non-null   object
5   tenure            7043 non-null   int64
6   PhoneService      7043 non-null   object
7   MultipleLines     7043 non-null   object
8   InternetService   7043 non-null   object
9   OnlineSecurity    7043 non-null   object
10  OnlineBackup      7043 non-null   object
11  DeviceProtection  7043 non-null   object
12  TechSupport       7043 non-null   object
13  StreamingTV       7043 non-null   object
14  StreamingMovies   7043 non-null   object
15  Contract          7043 non-null   object
16  PaperlessBilling  7043 non-null   object
17  PaymentMethod     7043 non-null   object
18  MonthlyCharges    7043 non-null   float64
19  TotalCharges      7043 non-null   object
20  Churn             7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

- df.describe() – Statistical summary of numerical columns

```
df.describe()
```

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges
count	7043.000000	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692	2279.734304
std	0.368612	24.559481	30.090047	2266.794470
min	0.000000	0.000000	18.250000	0.000000
25%	0.000000	9.000000	35.500000	398.550000
50%	0.000000	29.000000	70.350000	1394.550000
75%	0.000000	55.000000	89.850000	3786.600000
max	1.000000	72.000000	118.750000	8684.800000

Purpose

- To understand the structure of the data
- To identify missing or incorrect values
- To verify column names and data types

5. Data Pre-processing

Data preprocessing is one of the most important steps in any machine learning project. The quality of preprocessing directly impacts model performance.

5.1 Handling Missing Values

- Certain columns like TotalCharges may contain blank or non-numeric values
- These values are converted to numeric using pd.to_numeric()
- Rows with invalid or missing values are either:
    - Removed, or
    - Imputed based on project requirements

5.2 Dropping Irrelevant Columns

```
# dropping customerID column as this
df=df.drop(columns=["customerID"])
```

- Columns such as customerID do not contribute to prediction
- These columns are removed to avoid noise in the model

5.3 Encoding Categorical Variables

Machine learning models cannot work directly with categorical (text) data.

Steps followed:

1. Identify categorical columns (object data type)
2. Apply **Label Encoding** to convert categories into numeric values
3. Store each encoder in a dictionary for reuse

```
#Printing the unique columns values in all the columns
numerical_features=["tenure","MonthlyCharges","TotalCharges"]


for col in df.columns:
    if col not in numerical_features:
        print(col, df[col].unique())
        print("-"*50)
```

```
gender ['Female' 'Male']
--------------------------------------------------
SeniorCitizen [0 1]
--------------------------------------------------
Partner ['Yes' 'No']
--------------------------------------------------
Dependents ['No' 'Yes']
--------------------------------------------------
PhoneService ['No' 'Yes']
--------------------------------------------------
MultipleLines ['No phone service' 'No' 'Yes']
--------------------------------------------------
InternetService ['DSL' 'Fiber optic' 'No']
--------------------------------------------------
OnlineSecurity ['No' 'Yes' 'No internet service']
--------------------------------------------------
OnlineBackup ['Yes' 'No' 'No internet service']
--------------------------------------------------
DeviceProtection ['No' 'Yes' 'No internet service']
--------------------------------------------------
```

```
DeviceProtection ['No' 'Yes' 'No internet service']
--------------------------------------------------
TechSupport ['No' 'Yes' 'No internet service']
--------------------------------------------------
StreamingTV ['No' 'Yes' 'No internet service']
--------------------------------------------------
StreamingMovies ['No' 'Yes' 'No internet service']
--------------------------------------------------
Contract ['Month-to-month' 'One year' 'Two year']
--------------------------------------------------
PaperlessBilling ['Yes' 'No']
--------------------------------------------------
PaymentMethod ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
--------------------------------------------------
Churn ['No' 'Yes']
--------------------------------------------------
```

```python
encoders={} #to save the encoders in dictionary

for column in object_columns:
    label_encoder = LabelEncoder()
    df[column]=label_encoder.fit_transform(df[column])
    encoders[column]=label_encoder


# save the encoders to a pickle file
with open("encoders:pk1","wb") as f:
    pickle.dump(encoders, f)
```

```python
encoders
```

```
{'gender': LabelEncoder(),
 'Partner': LabelEncoder(),
 'Dependents': LabelEncoder(),
 'PhoneService': LabelEncoder(),
 'MultipleLines': LabelEncoder(),
 'InternetService': LabelEncoder(),
 'OnlineSecurity': LabelEncoder(),
 'OnlineBackup': LabelEncoder(),
 'DeviceProtection': LabelEncoder(),
 'TechSupport': LabelEncoder(),
 'StreamingTV': LabelEncoder(),
 'StreamingMovies': LabelEncoder(),
 'Contract': LabelEncoder(),
 'PaperlessBilling': LabelEncoder(),
 'PaymentMethod': LabelEncoder(),
 'Churn': LabelEncoder()}
```

```python
df.head()
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV
0	0	0	1	0	1	0	1	0	0	2	0	0	0
1	1	0	0	0	34	1	0	0	2	0	2	0	0
2	1	0	0	0	2	1	0	0	2	2	0	0	0
3	1	0	0	0	45	0	1	0	2	0	2	2	0
4	0	0	0	0	2	1	0	1	0	0	0	0	0

```
df.head()
```

OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn
0	2	0	0	0	0	0	1	2	29.85	29.85	0
2	0	2	0	0	0	1	0	3	56.95	1889.50	0
2	2	0	0	0	0	0	1	3	53.85	108.15	1
2	0	2	2	0	0	1	0	0	42.30	1840.75	0
0	0	0	0	0	0	0	1	2	70.70	151.65	1

Saving encoders ensures consistent transformation during future predictions.

6. Feature Scaling

Feature scaling is applied to numerical features so that all features contribute equally to model training.

Why Scaling is Required

- Prevents dominance of large-value features
- Improves convergence speed
- Enhances performance of distance-based models

Scaling Technique Used

- **StandardScaler** from scikit-learn

Formula applied:

```
Z = (X - μ) / σ
```

Scaled Features

- tenure
- MonthlyCharges
- TotalCharges

The same scaler must be reused during prediction to maintain consistency.

7. Train-Test Split

The dataset is split into:

- **Training set (e.g., 80%)**
- **Testing set (e.g., 20%)**

Training and testing the dataset

```python
X = df.drop(columns=["Churn"])
y = df["Churn"]
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
print(y_train.shape)
```

```
(5634,)
```

```python
print(y_train.value_counts())
```

```
Churn
0    4138
1    1496
Name: count, dtype: int64
```

This ensures unbiased evaluation of the model.

8. Model Building

After preprocessing, the cleaned dataset is ready for model training.

Model Selection

Churn prediction is a classification problem. Common algorithms suitable for this task include:

- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier

Training with default hyperparameters

```python
models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "XGBoost": XGBClassifier(random_state=42)
}
```

The chosen model learns patterns between customer attributes and churn behavior by minimizing classification error.

Training Process

1. Initialize the model with suitable hyperparameters
2. Fit the model using training data (`X_train`, `y_train`)
3. Store the trained model in memory for evaluation

Random Forest gives the highest accuracy compared to other models with default parameters

```
rfc = RandomForestClassifier(random_state=42)
```

```
rfc.fit(X_train_smote, y_train_smote)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
print(y_test.value_counts())
```

```
Churn
0    1036
1     373
Name: count, dtype: int64
```


```
print(y_test.value_counts())
```

```
Churn
0    1036
1     373
Name: count, dtype: int64
```


9. Model Evaluation

Model evaluation helps determine how well the trained model performs on unseen data.

9.1 Accuracy Score

- Represents the percentage of correctly classified instances
- Easy to understand but not sufficient for imbalanced datasets

9.2 Confusion Matrix

The confusion matrix provides a detailed breakdown:

Actual / Predicted	**No Churn**	**Churn**
No Churn	True Negative	False Positive
Churn	False Negative	True Positive

This helps analyze specific types of errors.

9.3 Classification Report

Provides advanced metrics:

- **Precision** – How many predicted churns were correct

- **Recall** – How many actual churns were detected
- **F1-score** – Balance between precision and recall

```
# evaluate on test data
y_test_pred = rfc.predict(X_test)

print(f"accuracy Score: {accuracy_score(y_test, y_test_pred)}\n\n")
print(f"Confusion matrix:\n {confusion_matrix(y_test, y_test_pred)}\n\n")
print("Classification Report:\n", classification_report(y_test, y_test_pred))
```

```
accuracy Score: 0.7785663591199432


Confusion matrix:
 [[878 158]
 [154 219]]


Classification Report:
               precision    recall  f1-score   support

           0       0.85      0.85      0.85      1036
           1       0.58      0.59      0.58       373

    accuracy                           0.78      1409
   macro avg       0.72      0.72      0.72      1409
weighted avg       0.78      0.78      0.78      1409
```

10. Cross-Validation

What is Cross-Validation?

Cross-validation is a technique used to evaluate model performance more reliably by training and testing the model on multiple subsets of data.

Technique Used

K-Fold Cross-Validation

- The dataset is divided into *K* equal parts (folds)
- The model is trained on K-1 folds and tested on the remaining fold
- This process is repeated K times
- Final performance is averaged across all folds

Why Cross-Validation is Important

- Reduces dependency on a single train-test split
- Detects overfitting and underfitting
- Provides more stable performance estimates

```python
cv_scores={}

#5 fold cross vaildation for each model
for model_name, model in models.items():
    print(f"Training {model_name} with default paramters")
    scores =  cross_val_score(model, X_train_smote, y_train_smote, cv=5, scoring="accuracy")
    cv_scores[model_name]=scores
    print(f"{model_name} cross-vaildation accuracy: {np.mean(scores):.2f}")
    print("-"*70)
```

```
Training Decision Tree with default paramters
Decision Tree cross-vaildation accuracy: 0.78
----------------------------------------------------------------------
Training Random Forest with default paramters
Random Forest cross-vaildation accuracy: 0.84
----------------------------------------------------------------------
Training XGBoost with default paramters
XGBoost cross-vaildation accuracy: 0.83
----------------------------------------------------------------------
```

```python
scores
```

```
array([0.70048309, 0.75649547, 0.90271903, 0.89486405, 0.90030211])
```

```python
cv_scores
```

```
{'Decision Tree': array([0.68297101, 0.71299094, 0.82175227, 0.83564955, 0.83564955]),
 'Random Forest': array([0.72524155, 0.77824773, 0.90513595, 0.89425982, 0.90090634]),
 'XGBoost': array([0.70048309, 0.75649547, 0.90271903, 0.89486405, 0.90030211])}
```

Cross-validation confirms that the Logistic Regression model generalizes well.

11. Prediction on New Input

New customer data is:

1. Converted to DataFrame
2. Encoded using saved encoders
3. Scaled using the same scaler
4. Passed to the trained model

```python
input_data = {
    'gender': 'Female',
    'SeniorCitizen': 0,
    'Partner': 'Yes',
    'Dependents': 'No',
    'tenure': 1,
    'PhoneService': 'No',
    'MultipleLines': 'No phone service',
    'InternetService': 'DSL',
    'OnlineSecurity': 'No',
    'OnlineBackup': 'Yes',
    'DeviceProtection': 'No',
    'TechSupport': 'No',
    'StreamingTV': 'No',
    'StreamingMovies': 'No',
    'Contract': 'Month-to-month',
    'PaperlessBilling': 'Yes',
    'PaymentMethod': 'Electronic check',
    'MonthlyCharges': 29.85,
    'TotalCharges': 29.85
}
```

```python
input_data_df = pd.DataFrame([input_data])

with open("encoders.pkl", "rb") as f:
  encoders = pickle.load(f)


# encode categorical featires using teh saved encoders
for column, encoder in encoders.items():
  input_data_df[column] = encoder.transform(input_data_df[column])

# make a prediction
prediction = loaded_model.predict(input_data_df)
pred_prob = loaded_model.predict_proba(input_data_df)

print(prediction)

# results
print(f"Prediction: {'Churn' if prediction[0] == 1 else 'No Churn'}")
print(f"Prediciton Probability: {pred_prob}")
```

```
[0]
Prediction: No Churn
Prediciton Probability: [[0.78 0.22]]
```

12. Project Workflow Summary

Data Collection
↓
Data Cleaning
↓
Encoding & Scaling
↓
Train-Test Split
↓
Model Training
↓
Model Evaluation
↓
Save Model & Encoders
↓
Prediction on New Data

13. Conclusion

This notebook demonstrates a **complete end-to-end customer churn prediction system**, covering:

- Data preprocessing
- Feature engineering

- Model training
- Evaluation
- Model persistence

It can be extended further by:

- Hyperparameter tuning
- Trying advanced models (XGBoost, Neural Networks)
- Deploying using Flask/FastAPI
