customer-churn-project

April 24, 2025

1 PROJECT OVERVIEW

Customer churn, or customer attrition, refers to when a customer ceases their relationship with a company or service provider. In today's highly competitive business environment, retaining customers is a critical factor for long-term success. Predicting customer churn can help organizations take proactive steps to retain customers, thus minimizing revenue loss. This project aims to build a machine learning model that can predict whether a customer will churn based on their demographic, account, and service-related data.

2 PROBLEM STATEMENT

The goal of this project is to develop a classification model that predicts whether a customer will churn. Using demographic data (such as gender, senior citizen status, and tenure), along with information about the services they use (such as internet service, phone service, and online security), we will attempt to build a model that helps the company identify customers who are at a high risk of churning. By predicting customer churn, the company can proactively design retention strategies to keep these customers, thereby improving customer satisfaction and reducing financial loss.

[22]:
```python
# Importing useful libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import␣
 ↪accuracy_score,precision_score,recall_score,f1_score,classification_report,confusion_matrix
# 1. Load Dataset
df = pd.read_csv("C:/Users/Himanshu/Downloads/customer_data.csv")
df.head()
print("\nDataset Info:")
df.info()
print("\nMissing Values:")
(df.isnull().sum())
```

```
Dataset 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
 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      7032 non-null   float64
 20  Churn             7043 non-null   object
dtypes: float64(2), int64(2), object(17)
memory usage: 1.1+ MB

Missing Values:
```

[22]:

```
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
```

```
StreamingMovies      0
Contract             0
PaperlessBilling     0
PaymentMethod        0
MonthlyCharges       0
TotalCharges        11
Churn                0
dtype: int64
```

[54]:

```
df.dropna(inplace=True)
# Feature Engineering
# Keeping tenure as a numerical feature instead of binning
# Encoding Categorical Variables
label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Feature Scaling
scaler = StandardScaler()
feature_columns = df.columns.difference(['customerID', 'Churn'])
df[feature_columns] = scaler.fit_transform(df[feature_columns])

# Splitting Data
X = df.drop(columns=['Churn'])
y = df['Churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪2,random_state=42, stratify=y)

# Handling Class Imbalance
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)
```

[26]:

```
# Model Training with XGBoost
xgb = XGBClassifier(objective='binary:logistic',␣
 ↪eval_metric='logloss',random_state=42)
param_grid = {
'n_estimators': [50, 100, 200],
'max_depth': [3, 6, 9],
'learning_rate': [0.01, 0.1, 0.2]
}
grid_search = GridSearchCV(xgb, param_grid, cv=5, scoring='f1', n_jobs=-1)
grid_search.fit(X_train, y_train)
# Best Model
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
```

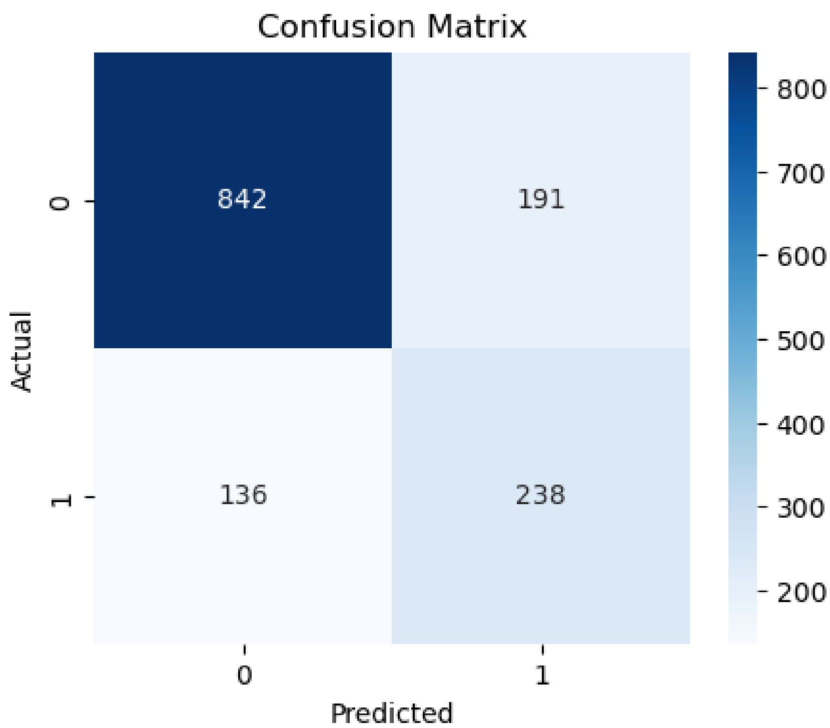
[27]:
```
# Model Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
# Confusion Matrix
plt.figure(figsize=(5,4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
3
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy: 0.767590618336887
Precision: 0.5547785547785548
Recall: 0.6363636363636364
F1 Score: 0.5927770859277709

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.82      0.84      1033
           1       0.55      0.64      0.59       374

    accuracy                           0.77      1407
   macro avg       0.71      0.73      0.72      1407
weighted avg       0.78      0.77      0.77      1407
```



[ ]:
```
# Predicting for New Data
def predict_churn(new_data):
 new_df = pd.DataFrame([new_data])
# Encoding categorical variables
for col, le in label_encoders.items():
 if col in new_df.columns:
 new_df[col] = new_df[col].apply(lambda x: le.transform([x])[0]
 if x in le.classes_ else -1)
# Ensuring numerical columns are correctly formatted
for col in feature_columns:
 if col in new_df.columns:
 new_df[col] = pd.to_numeric(new_df[col], errors='coerce')
new_df.fillna(0, inplace=True) # Handling NaN values that may arise
new_df[feature_columns] = scaler.transform(new_df[feature_columns])
return best_model.predict(new_df)[0]
```

[86]:
```
# Example Prediction
new_customer = {'gender': 'Male', 'SeniorCitizen': 0, 'Partner':␣
 ↪'Yes','Dependents': 'No', 'tenure':
12.0, 'PhoneService': 'Yes',
'MultipleLines': 'No', 'InternetService': 'Fiber optic','OnlineSecurity': 'No',␣
 ↪'OnlineBackup': 'No',
```

```
'DeviceProtection': 'No', 'TechSupport': 'No', 'StreamingTV':'Yes',␣
↪'StreamingMovies': 'No',
'Contract': 'Month-to-month', 'PaperlessBilling': 'Yes','PaymentMethod':␣
↪'Electronic check',
'MonthlyCharges': 70.35, 'TotalCharges': 850.0}
print("Predicted Churn (1 = Yes, 0 = No):", predict_churn(new_customer))
```

```
Predicted Churn (1 = Yes, 0 = No): None
```

3 4. Insights

High Risk: Month-to-month contracts, high monthly charges, and lack of add-ons.

Retention Strategy: Offer discounts or upgrades for high-risk churn profiles.

Services Impact: OnlineSecurity, TechSupport reduce churn likelihood.

4 Video Explanation:

https://drive.google.com/file/d/1F1IMVpdOxfPuhkJ5dGNQrkdqd3yZ7Fcj/view?usp=sharing

[ ]:
```

```