Predicting customer churn using machine learning to uncover hidden patterns

Student Name: Harini.S

Register Number: [513523106302]

Institution: [AMCET]

Department: [ECE]

Date of Submission: [26-04-2025]

- **Problem Statement**

Customer retention is a critical metric for business success, especially in highly competitive markets. Losing customers (churn) directly impacts revenue and long-term growth. However, many organizations struggle to identify early warning signs of churn due to the complex and often hidden patterns in customer behavior.

- **Objectives of the Project**

The objective of this project is to develop a robust machine learning model that can accurately predict customer churn by analyzing historical and behavioral customer data. The model aims to uncover hidden patterns and key factors influencing churn, enabling businesses to identify high-risk customers early and implement data-driven retention strategies to reduce churn rates and improve customer lifetime value.

- **Scope of the Project**

1. Data Collection & IntegrationCollect historical customer data, including demographics, transactions, usage behavior, and customer support

interactions.Integrate data from multiple sources such as CRM systems, billing logs, and user activity databases.

2. Data Preprocessing & Feature Engineering Clean and preprocess the data by handling missing values, encoding categorical variables, and normalizing features.Create new features that may better capture customer behavior and trends (e.g., customer tenure, engagement scores).

3. Model Development

Train and evaluate multiple machine learning algorithms (e.g., Logistic Regression, Random Forest, XGBoost).

Use cross-validation and hyperparameter tuning to optimize model performance.

4. Pattern Discovery & Interpretability

Use feature importance techniques (e.g., SHAP, LIME) to interpret model predictions.

Identify hidden behavioral patterns and key drivers of customer churn.

5. Evaluation & Metrics

Evaluate the model using appropriate metrics such as accuracy, precision, recall, F1-score, and ROC-AUC.Compare model performances to select the most suitable one for deployment.

6. Deployment (Optional based on project scope)

Deploy the model as a service or integrate it into existing business systems.Set up monitoring to track model performance over time.

7. Business Insights & Recommendations

Translate model findings into actionable insights.Provide strategies for targeted customer retention based on churn predictions.

- **Data Sources**

- 1. Customer Demographics

- Source: CRM systems or user profilesData Includes: Age, gender, location, income, occupation, sign-up date

2. Transactional Data

Source: Billing systems, e-commerce platforms, point-of-sale systems

Data Includes: Purchase history, transaction frequency, total spend, subscription type, payment method, refunds

3. Usage Behavior

Source: Application/server logs, user activity trackersData Includes: Login frequency, time spent on platform, feature usage patterns, last active date

4. Customer Support Interactions

Source: Helpdesk software (e.g., Zendesk, Freshdesk), call center logsData Includes: Number of tickets, resolution time, sentiment of support interactions, number of complaints

5. Marketing and Communication Logs

Source: Email platforms, SMS logs, CRMData Includes: Email open/click rates, promotional response, engagement with campaigns

6. Feedback and Surveys

Source: Customer feedback tools, NPS surveys, in-app feedbackData Includes: Satisfaction scores, comments, Net Promoter Score (NPS)

7. External Data (Optional)

Source: Public datasets or purchased third-party dataData Includes: Industry trends, social media sentiment, economic indicators

- **High-Level Methodology**

1. Problem Definition

Clearly define the business problem: predict which customers are likely to churn.

Identify goals such as reducing churn rate and improving customer retention strategies.

2. Data Collection

Gather data from various sources: CRM, transaction logs, usage metrics, customer support systems, etc.

3. Data Preprocessing

Clean the data (handle missing values, remove duplicates, correct inconsistencies).

Encode categorical variables and normalize numerical data.

Split the dataset into training, validation, and test sets.

4. Feature Engineering

Create new features from raw data (e.g., customer tenure, average transaction value).

Select relevant features using correlation analysis or feature importance techniques.

5. Model Selection

Choose suitable machine learning models (e.g., Logistic Regression, Random Forest, XGBoost, Neural Networks).

Use cross-validation to ensure the model generalizes well to unseen data.

6. Model Training and Evaluation

Train models on the training dataset.

Evaluate performance using metrics like accuracy, precision, recall, F1-score, and ROC-AUC.

7. Interpretability & Pattern Discovery

Use tools like SHAP or LIME to interpret model predictions and uncover hidden churn patterns.

Identify key factors influencing customer churn.

8. Model Deployment (Optional)

Deploy the model into a production environment for real-time churn prediction.

Set up pipelines for continuous data input and prediction.

9. Business Insights & Actionable Recommendations

Present findings to stakeholders with visualizations and clear insights.

Recommend retention strategies based on model output (e.g., targeted campaigns, personalized offers**).**

Data Collection – *1. dentify Data Requirements*
Determine the types of data needed to predict churn, such as:
Customer demographics (age, gender, location)
Account and subscription details (start date, plan type, auto-renew status)

Transaction history (purchase frequency, amount spent)
Usage behavior (login frequency, activity level, feature usage)
Customer support interactions (number of tickets, complaint frequency)
Marketing engagement (email opens, offer clicks, survey responses)

2. Data Sources
Collect data from:
CRM systems (e.g., Salesforce, Zoho)
Internal databases (e.g., MySQL, PostgreSQL)
Web/app analytics tools (e.g., Google Analytics, Mixpanel)
Support systems (e.g., Zendesk, Freshdesk)
Transaction systems (e.g., billing software, POS systems)
Marketing tools (e.g., Mailchimp, HubSpot)
Surveys & Feedback platforms (e.g., Typeform, SurveyMonkey)

Data Cleaning – *1. Handle Missing Values*
Identify missing data using .isnull() or .info() (in Python/pandas).
Strategies:
Remove rows/columns with too many missing values.
Impute values:
Numerical: Mean, median, or mode
Categorical: Most frequent or a new category (e.g., 'Unknown')
Predictive imputation: Use models to estimate missing values.

2. Remove Duplicates
Detect and drop duplicates using:
df.drop_duplicates(inplace=True)

Exploratory Data Analysis (EDA) – *1. Understand the Dataset*
Shape of the data:
df.shape, df.columns, df.info()
Data types & missing values:
df.isnull().sum(), df.describe()

2. Target Variable Analysis (Churn)
Distribution of churned vs. non-churned customers:
df['churn'].value_counts(normalize=True).plot(kind='bar')
Check class imbalance – important for model selection & evaluation strategy.

Feature Engineering – *1. Temporal Features*
Customer tenure: Days/months since account creation
df['tenure_days'] = (today - df['signup_date']).dt.days
Recency: Days since last activity or transaction
Churn window: Time since last purchase/login before churn

2. Aggregated Behavioral Features
Average purchase value
Total number of transactions
Frequency of logins or usage
Average session duration
Time between transaction

Model Building –1. Temporal Features

Customer tenure: Days/months since account creation
df['tenure_days'] = (today - df['signup_date']).dt.days
Recency: Days since last activity or transaction
Churn window: Time since last purchase/login before churn
2. Aggregated Behavioral Features
Average purchase value
Total number of transactions
Frequency of logins or usage
Average session duration

Time between transactions

Model Evaluation –1. Evaluation Metrics
Because churn is a binary classification problem (Churn = 1, Not Churn = 0), here are the key metrics:
Accuracy
Proportion of correct predictions.
Not ideal alone if there's class imbalance.
from sklearn.metrics import accacy_score
accuracy_score(y_test, y_pred)
Precision
Of all predicted churns, how many were correct?
High precision = few false positives.
from sklearn.metrics import precision_score
precision_score(y_test, y_pred)
Recall (Sensitivity)
Of all actual churns, how many did we catch?
High recall = few false negatives.
from sklearn.metrics import recall_score
recall_score(y_test, y_pred)
F1 Score
Harmonic mean of precision and recall; balances both.
from sklearn.metrics import f1_score
f1_score(y_test, y_pred)

ROC AUC (Receiver Operating Characteristic - Area Under Curve)
Measures model's ability to distinguish between churn vs. no-churn across all thresholds.from sklearn.metrics import roc_auc_score
roc_auc_score(y_test, y_proba)

Confusion Matrix
Shows true positives, true negatives, false positives, and false negatives.from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)

Classification Report

Complete summary of all key metrics.from sklearn.metrics import classification_reportprint(classification_report(y_test, y_pred))

2. Cross-Validation
Evaluate performance across multiple data splits to reduce overfitting:from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X, y, cv=5, scoring='f1')print("Average F1 score:", scores.mean())

Visualization & Interpretation – *1. Correlation Heatmap*
Visualize how features relate to each other and to churn:
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Feature Correlation Heatmap')

2. Distribution of Churn

Compare churn vs. non-churn across key features:
sns.histplot(data=df, x='monthly_spend', hue='churn', kde=True)
Use boxplots for numeric features:
sns.boxplot(x='churn', y='tenure_days', data=df)

Deployment – 1. Prepare the Model for Deployment

Serialize the model:

import joblib

joblib.dump(model, 'churn_model.pkl')

Save any preprocessing pipelines (e.g., encoders, scalers).

2. Choose a Deployment Method

Batch prediction: Run the model at scheduled intervals on customer data.

Real-time API: Expose the model via a REST API.

3. Deploy Using a Web Framework (API Example)

Use Flask or FastAPI for a lightweight REST API:

from fastapi import FastAPI

import

- **Existing system**

1. Manual or Rule-Based Churn Detection

Churn is identified after it occurs (reactive, not proactive).

Businesses rely on fixed rules like:

"If a customer hasn’t logged in for 30 days, mark them as likely to churn."

"If a user downgrades or contacts support multiple times, tag them as at risk."

These rules are not data-driven and often miss subtle patterns.

2. Basic Retention Strategies

Generic marketing emails or offers are sent to all customers periodically.

No personalization or targeted retention efforts based on churn risk.

3. Lack of Behavioral Insight

Customer behavior data (e.g., login frequency, support tickets, usage patterns) is collected but not fully analyzed.

Decisions are often made on intuition or simple metrics like transaction history.

4. Limited Reporting

Dashboards may show historical churn rates, but do not provide:Predictive insightsReal-time alertsReasons behind churn

5. No Automation or Integration

Teams handle retention manually—contacting customers, offering discounts, etc.

No system in place to automatically flag high-risk users or suggest actions.

6. Missed Opportunities

High-value customers may churn unnoticed.

Resource allocation is not optimized (e.g., spending time on customers who wouldn't churn anyway).

- **Proposed System**

1. Predictive Churn Detection

Leverages machine learning algorithms to predict churn before it happens.

Models trained on historical customer data to uncover hidden patterns and behaviors associated with churn.

2. Personalized Retention Actions

Customers are scored based on their churn risk (e.g., 0–1 probability).High-risk customers can be targeted with:

Personalized offers

Loyalty rewards

Outreach by support/success teams

3. Real-Time or Scheduled Predictions

Predictions can be made:

In real time via an API (e.g., for SaaS platforms)

In batch mode for periodic reports (e.g., daily or weekly churn lists)

4. Feature-Driven Insights

Key churn drivers are identified (e.g., low usage, frequent support complaints).

Business teams can act based on clear indicators.

5. Automated Pipeline

Includes data ingestion, preprocessing, prediction, and reporting.

Minimal manual intervention required.

6. Integrated Dashboards & Alerts

Dashboards show:

Overall churn risk distribution

Segment-wise churn trends (e.g., by plan type or region)

Alerts notify teams when churn risk spikes in a segment.

7. Scalable & Extendable

Can be integrated with CRM, marketing tools, and data lakes.

Scales with data volume and business growth.

8. Performance Monitoring

Model performance (accuracy, AUC, recall) is tracked over time.

Retraining is triggered when performance drops

- **Tools and Technologies**

1. Data Collection & Storage

Databases: MySQL, PostgreSQL, MongoDB

Cloud Storage: AWS S3, Google Cloud Storage, Azure Blob Storage

Data Warehouses: Snowflake, Google BigQuery, Amazon Redshift

ETL Tools: Apache Airflow, Talend, AWS Glue, Informatica

2. Programming Languages

Python (most widely used for data science and ML)R (for statistical analysis)SQL (for querying databases)

3. Data Analysis & Preprocessing

Python Libraries:

pandas, numpy – data manipulation

scikit-learn – preprocessing, modeling

matplotlib, seaborn, plotly – data visualization

4. Machine Learning & Modeling

scikit-learn – classic ML models (Logistic Regression, Random Forest, etc.)

XGBoost, LightGBM, CatBoost – gradient boosting models

TensorFlow, Keras, PyTorch – deep learning frameworks (optional)

MLflow – model tracking and versioning

5. Model Evaluation & Explainability

SHAP, LIME – model interpretability

Yellowbrick – visual ML diagnostics

6. Model Deployment

Flask / FastAPI – build REST APIs for predictions

Docker – containerization

Kubernetes – orchestration for scaling

Cloud Platforms:

AWS (SageMaker, Lambda)

Google Cloud (Vertex AI, Cloud Run)

Microsoft Azure (ML Studio, Functions)

7. Monitoring & Logging

Prometheus, Grafana – performance monitoring

ELK Stack – logs analysis (Elasticsearch, Logstash, Kibana)

8. Visualization & Reporting

Tableau, Power BI, Looker – dashboards

Streamlit, Dash – custom web apps for interactive visualizations

Programming Language – *Primary Programming Language: Python*

Why Python?

1. Ease of Learning: Python is beginner-friendly, with a simple syntax and a large community, which makes it easy to learn and apply.

2. Extensive Libraries: Python has a wide variety of libraries and frameworks that are specifically built for data science, machine learning, and data visualization, making it a versatile language for the entire workflow.

3. Data Science Libraries: Python offers a rich set of libraries for data manipulation, machine learning, and visualization:

pandas: For data manipulation and analysis

NumPy: For numerical operations

scikit-learn: For traditional machine learning models and preprocessing

XGBoost, LightGBM, CatBoost: For gradient boosting models

TensorFlow, Keras, PyTorch: For deep learning models (if needed)

matplotlib, seaborn, plotly: For data visualization

4. Integration: Python integrates well with databases, web applications, and cloud platforms, making it ideal for building end-to-end solutions.

5. Deployment: With tools like Flask and FastAPI, Python makes it easy.

Notebook/IDE *– or developing and running a Customer Churn Prediction project, you'll want to use an Integrated Development Environment (IDE) or Jupyter Notebook for an interactive development experience. Both options are popular in the data science and machine learning community, depending on your preference and workflow. Here's a breakdown:*

1. Jupyter Notebook / JupyterLab

Jupyter Notebooks are an essential tool for data science projects, offering an interactive way to write and test code, visualize data, and document the process. It's ideal for projects where you need to combine code, visualizations, and markdown explanations.

Why Jupyter Notebook?

Interactive Coding: You can write and execute code in cells, making it easy to test individual parts of your workflow.

Visualizations: Easily plot charts and graphs inline using libraries like matplotlib, seaborn, and plotly.

Documentation: Supports markdown for adding explanations, creating clear and documented workflows.

Widely Used in Data Science: Many datasets, models, and tutorials are shared in the form of Jupyter Notebooks.

Flexible: Can run Python, R, and other languages via plugins or kernels.

How to Use Jupyter Notebook:

1. Install Jupyter via Anaconda or pip:
With Anaconda:conda install jupyter
With pip:pip install notebook

2. Start a Jupyter Notebook:
Open a terminal and type jupyter notebook to launch the Jupyter interface in your browser.
3. Working in Notebooks:
Write code, run it, visualize data, and document your steps all in one place.
Great for prototyping, experimenting with models, and sharing results.
JupyterLab:
An enhanced version of Jupyter Notebook with a more modern interface, offering better project management and file navigation.
2. IDEs for Python
If you prefer an IDE (especially for larger projects or if you're focusin more on deployment), here are some great options for Python:
a) Visual Studio Code (VSCode)
Why VSCode?
Lightweight and fast, with powerful extensions.
Support for Python, Jupyter Notebooks, Git, and more.
IntelliSense (code completion and suggestions).
Integrated terminal for running scripts and commands.
Excellent support for extensions like Python, Pylint, and Jupyter.
Setup:

1. Install [VS]

Libraries – *1. Data Collection and Manipulation Libraries*

These libraries help you handle and preprocess data effectively.

pandas:

Used for data manipulation and analysis.

Essential for loading, cleaning, and transforming data.

Example:

```
import pandas as pd
df = pd.read_csv('customer_data.csv')
```

NumPy:

Provides support for large, multi-dimensional arrays and matrices.

Used for numerical operations.

Example:

```
import numpy as np
array = np.array([1, 2, 3])
```

SQLAlchemy:

Used to interact with relational databases (e.g., MySQL, PostgreSQL) in Python.

Example:

```
from sqlalchemy import create_engine
engine =
create_engine('postgresql://username:password@localhost/mydatabase')
df = pd.read_sql('SELECT * FROM customers', engine)
```

2. Data Visualization Libraries

These libraries help vi

alize trends, relationships, and distributions in the data.

matplotlib:

A fundamental library for creating static, animated, and interactive plots.

Example

```
import matplotlib.pyplot as plt
df['churn'].value_counts().plot(kind='bar')
plt.show()
```

seaborn:

Built on top of matplotlib, it provides a high-level interface for drawing attractive statistical graphics.

Example:
import seaborn as sns
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plotly:
For creating interactive plots and dashboards.
Example:
import plotly.express as px
fig = px.scatter(df, x='age', y='tenure', color='churn')
fig.show()
3. Machine Learning Libraries
These are essential for training, tuning, and evaluating models.
scikit-learn:
The go-to library for classical machine learning algorithms (e.g., Logistic Regression, Decision Trees, Random Forest).
Includes tools for model selection, feature selection, and evaluation.
Example:
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model

Optional Tools for Deployment – 1. Web Frameworks for API Deployment

a) Flask

Description: Flask is a lightweight web framework in Python used for building web applications and APIs. It's easy to set up, flexible, and great for small-to-medium-sized projects.

Use Case: Exposing a machine learning model as a RESTful API to make predictions.

Pros:

Lightweight and simple to use.

Easy to integrate with Python code and libraries.

Well-suited for small-scale applications.

Example:

```
from flask import Flask, request, jsonify
import joblib
app = Flask(_name_)
# Load your model
model = joblib.load('churn_model.pkl')
@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json(force=True)
    prediction = model.predict([data['features']])
    return jsonify({'prediction': prediction.tolist()})
if _name_ == '_main_':
    app.run(debug=True)
```

b) FastAPI

Description: FastAPI is a modern

Team Members and Roles

1. SANTHOSH KUMAR. K - TEAM LEADER / MANAGER

2. HARI GANESH. B - DATA SCIENTIST

3. HARINI. S - SOFTWARE DEVELOPER

4. SANTHOSH. G - BUSINESS ANALYST