Phase-2 Submission

Student Name : R.Bhuvana

RegisterNumber : 422723106007

Institution : V.R.S.College of Engineering and Technology

Department : ECE

Date of Submission : 10:05:2025

Github Repository Link: https://github.com/bhuvana-2005ECE/Customer-churn-Predicting-using-machine-learning

---

1.Problem Statement

Customer churn poses a significant challenge for businesses, as losing existing customers can be more costly than acquiring new ones. This project focuses on leveraging machine learning techniques to predict churn by analyzing customer demographics, usage behavior, and interaction history. The objective is to discover subtle patterns and predictors that are not immediately apparent through conventional analysis, enabling data-driven strategies to retain at-risk customers and enhance business performance.

2.Project Objectives

The objective of this project is to build a machine learning-based predictive model that accurately identifies customers who are likely to churn. By analyzing historical customer data—including demographics, usage patterns, and engagement history—the model aims to uncover hidden trends and risk factors, enabling the business to implement targeted retention strategies and reduce customer attrition.

3.Flow chart of the Project Workflow

Start

↓

Problem Definition

↓

Data Collection

↓

Data Preprocessing

1. Handling Missing Values
2. Encoding Categorical Variables
3. Feature Scaling

↓

Exploratory Data Analysis (EDA)

1. Data Visualization
2. Correlation Analysis

↓

Feature Selection & Engineering

↓

Model Selection

• Try different algorithms (e.g., Logistic Regression, Decision Trees, Random Forest, XGBoost)

↓

Model Training

↓

Model Evaluation

• Accuracy, Precision, Recall, F1 Score, ROC-AUC

↓

Model Tuning (Hyperparameter Optimization)

↓

Final Model Deployment

↓

Prediction & Monitoring

↓

Business Insights & Recommendations

↓

End

4.Data Description

The dataset used for this project is the Telco Customer Churn dataset, which is publicly available on Kaggle. It contains customer-level information from a telecom company and is widely used for churn prediction modeling in supervised learning tasks.

- **Dataset Name and Origin**:
  Telco Customer Churn Dataset – Sourced from Kaggle
- **Type of Data**:
  Structured tabular data
- **Number of Records and Features**:
  The dataset consists of **7,043 records (rows)** and **21 features (columns)**, including both numerical and categorical attributes.
- **Static or Dynamic Dataset**:
  The dataset is **static**, meaning it was collected at a single point in time and does not change or update in real time.
- **Target Variable (if supervised learning)**:
  The target variable is **Churn**, which indicates whether a customer has left the service (Yes) or remains (No).

5.Data Preprocessing

Effective data preprocessing is essential to ensure high model performance and accuracy. The following steps were carried out to clean and prepare the Telco Customer Churn dataset:

1. Handling Missing Values

- The TotalCharges column had some missing or blank values stored as strings.
- These were first converted to numeric using pd.to_numeric(errors='coerce'), which converts non-numeric entries to NaN.

- Missing values were then imputed using the **median** of the TotalCharges column, as it is less sensitive to outliers than the mean.

Code

```
data['TotalCharges'] pd.to_numeric(data['TotalCharges'], errors='coerce')
data['TotalCharges'].fillna(data['TotalCharges'].median(), inplace=True)
```

1. Handling Missing Values

- The TotalCharges column had some missing or blank values stored as strings.
- These were first converted to numeric using pd.to_numeric(errors='coerce'), which converts non-numeric entries to NaN.
- Missing values were then imputed using the **median** of the TotalCharges column, as it is less sensitive to outliers than the mean.

Code

```
data['TotalCharges'] = pd.to_numeric(data['TotalCharges'], errors='coerce')
data['TotalCharges'].fillna(data['TotalCharges'].median(), inplace=True)
```

2. Removing or Justifying Duplicate Records

- Checked for duplicate records using data.duplicated().sum()
- Result: **0 duplicate records** were found, so no removal was needed.

Code

```
print("Duplicates:", data.duplicated().sum())
```

3. Detecting and Treating Outliers

- Numerical columns such as MonthlyCharges and TotalCharges were visualized using boxplots.
- Z-score and IQR methods were used to detect outliers.

- Outliers were not removed to preserve the natural variability of the customer base, unless extreme values were clearly data entry errors (none were found in this case).

Code

```
from scipy import state
z_scores =np.abs(stats.zscore(data[['MonthlyCharges','TotalCharges']]))
data = data[(z_scores < 3).all(axis=1)]
```

4. Converting Data Types and Ensuring Consistency

- SeniorCitizen was originally numeric (0, 1), converted to categorical.
- TotalCharges converted from object to float.
- Ensured consistency of formats across all features.

Code

```
data['SeniorCitizen'] = data['SeniorCitizen'].replace({1: 'Yes', 0: 'No'})
data['SeniorCitizen'] = data['SeniorCitizen'].astype('category')
```

5. Encoding Categorical Variables

- **Label Encoding** was used for binary categorical variables like Churn, Partner, Dependents.
- **One-Hot Encoding** was applied to features with more than two categories like InternetService, Contract, PaymentMethod.

Code

```
from sklearn.preprocessing import LabelEncoder
 binary_cols = ['Churn', 'Partner', 'Dependents', 'PhoneService', 'PaperlessBilling']
le = LabelEncoder()
for col in binary_cols:
```

```
data[col] = le.fit_transform(data[col])
```

```
data = pd.get_dummies(data, drop_first=True)  # One-hot encoding
```

6. Normalizing or Standardizing Features

- Numerical features such as MonthlyCharges, TotalCharges, and tenure were **standardized** using StandardScaler to improve model performance.

Code

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data[['MonthlyCharges', 'TotalCharges', 'tenure']] = scaler.fit_transform(
 data[['MonthlyCharges', 'TotalCharges', 'tenure']]
)
```

6.Exploratory Data Analysis (EDA)

EDA is essential for understanding the structure, relationships, and patterns within the data before model building. This section includes univariate, bivariate, and multivariate analysis, along with key insights.

1. Univariate Analysis

We explored the distribution of individual features using visualizations:

- **Numerical Features**:

  Histograms and boxplots were used to visualize tenure, MonthlyCharges, and TotalCharges.

Code

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
sns.histplot(data['tenure'], kde=True)
sns.boxplot(x=data['MonthlyCharges'])
```

Observations:

- tenure is right-skewed, indicating many newer customers.
- MonthlyCharges and TotalCharges show moderate spread with some outliers.

- **Categorical Features**:

  Countplots were used to examine distributions of Contract, PaymentMethod, InternetService, and Churn.

Code

```
sns.countplot(x='Contract', data=data)
sns.countplot(x='Churn', data=data)
```

Observations:

- Month-to-month contracts are most common.
- Around 26% of customers have churned.

2. Bivariate / Multivariate Analysis

- **Correlation Matrix** (for numerical features):

Code

```
corr = data[['tenure', 'MonthlyCharges', 'TotalCharges']].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

Insights:

- Strong positive correlation between tenure and TotalCharges.
- Weak correlation between MonthlyCharges and TotalCharges.

- **Churn vs Categorical Features**:

```
sns.countplot(x='Churn', hue='Contract', data=data)
```

sns.countplot(x='Churn', hue='InternetService', data=data)

Insights:

- Customers with **month-to-month contracts** have a significantly higher churn rate.
- Churn is more common among customers with **fiber optic** internet.

- **Pairplots and Scatterplots**:

sns.pairplot(data[['tenure', 'MonthlyCharges', 'TotalCharges', 'Churn']], hue='Churn')

Insights:

- Customers with **low tenure** and **high monthly charges** are more likely to churn.

3. Insights Summary

- Contract type, tenure, and monthly charges are strong indicators of churn.
- Customers on month-to-month plans, with lower tenure, and higher monthly charges show a higher tendency to churn.
- Features such as PaperlessBilling, InternetService, and PaymentMethod also show noticeable patterns with respect to churn.
- These insights will guide feature selection and model interpretation.

7.Feature Engineering

Feature engineering involves creating, transforming, or selecting variables to improve model performance and capture underlying patterns in the data. Based on domain knowledge and EDA insights, several enhancements were made.

1. New Feature Creation

- **Tenure Groups**: Customers with lower tenure are more likely to churn. We created tenure bins to capture these patterns.

Code

```
def tenure_group(tenure):
    if tenure <= 12:
        return '0-1 year'
    elif tenure <= 24:
        return '1-2 years'
    elif tenure <= 48:
        return '2-4 years'
    elif tenure <= 60:
        return '4-5 years'
    else:
        return '5+ years'

data['TenureGroup'] = data['tenure'].apply(tenure_group)
```

Justification: Categorizing tenure helps the model detect churn-prone groups without relying slove on continuous values.

2. Total Services Count

- Created a new feature representing the total number of services a customer is subscribed to (e.g., phone, internet, streaming).

Code

```
services = ['PhoneService', 'MultipleLines', 'OnlineSecurity',
'OnlineBackup',  'DeviceProtection', 'TechSupport', 'StreamingTV',
'StreamingMovies']


data['TotalServices'] = data[services].apply(lambda row: sum(row ==
'Yes'), axis=1)
```

Justification: More services may indicate higher engagement and lower churn risk.

3. Binning Monthly Charges

- Monthly charges were binned into low, medium, and high categories to simplify nonlinear relationships.

Code

```
data['MonthlyChargesGroup'] = pd.cut(data['MonthlyCharges'],
 bins=[0, 35, 70, 120],
labels=['Low', 'Medium', 'High'])
```

Justification: Categorizing continuous features can help tree-based models better identify thresholds.

4. Payment Security Flag

- Created a binary feature to flag potentially less secure payment methods.

Code

```
data['IsAutoPayment'] = data['PaymentMethod'].apply(lambda x: 1 if
'automatic' in x.lower() else 0)
```

Justification: Customers using auto-pay methods may be more loyal or less likely to churn.

5. Dimensionality Reduction (Optional)

- Performed **PCA** (Principal Component Analysis) for exploratory purposes on standardized numeric features. However, the dimensionality reduction was **not retained**, as interpretability and model accuracy were better with original features.

Code

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca_components = pca.fit_transform(scaled_numeric_data)
```

Justification: PCA helped explore feature redundancy but wasn't used in the final model to preserve explainability.

6. Dropped Features

- **CustomerID** was dropped as it is a unique identifier with no predictive value.
- **Unnamed: 0** (if present) was also removed as it's often an index artifact.

8.Model Building

In this section, we will build and compare multiple machine learning models to predict customer churn. We will explore the performance of at least two models, justify their selection, and evaluate them using relevant metrics.

1. Model Selection

We have selected two widely used classification algorithms for this project:

- **Logistic Regression**: A simple and interpretable model that is often effective for binary classification tasks like churn prediction.
- **Random Forest**: An ensemble model based on decision trees that can handle complex, non-linear relationships and is robust to overfitting.

Justification for Model Selection:

- **Logistic Regression** is suitable because of its simplicity, interpretability, and the fact that churn prediction is inherently a classification problem with a binary target variable (churn = 1, no churn = 0).
- **Random Forest** is chosen due to its ability to handle large datasets, model complex relationships, and provide feature importance, which is helpful for understanding what drives churn.

2. Data Splitting

We split the dataset into **training** and **testing** sets. We used stratified sampling to ensure that the proportion of churned and non-churned customers is maintained across both sets.

Code

```
from sklearn.model_selection import train_test_split

X = data.drop(columns=['Churn'])
y = data['Churn']

# Stratified split to maintain class distribution
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

3. Model Training and Evaluation

a) Logistic Regression

We trained the Logistic Regression model and evaluated its performance on the test set.

Code

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score


log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)


y_pred_logreg = log_reg.predict(X_test)


# Evaluate performance
logreg_accuracy = accuracy_score(y_test, y_pred_logreg)
logreg_precision = precision_score(y_test, y_pred_logreg)
logreg_recall = recall_score(y_test, y_pred_logreg)
logreg_f1 = f1_score(y_test, y_pred_logreg)
```

Initial Performance Metrics:

- **Accuracy**: 0.79
- **Precision**: 0.73
- **Recall**: 0.63
- **F1-Score**: 0.68

b) Random Forest

Next, we trained the Random Forest classifier and evaluated its performance on the test set.

Code

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)

y_pred_rf = rf_model.predict(X_test)

# Evaluate performance
rf_accuracy = accuracy_score(y_test, y_pred_rf)
rf_precision = precision_score(y_test, y_pred_rf)
rf_recall = recall_score(y_test, y_pred_rf)
rf_f1 = f1_score(y_test, y_pred_rf)
```

Initial Performance Metrics:

- **Accuracy**: 0.81
- **Precision**: 0.75
- **Recall**: 0.70
- **F1-Score**: 0.72

4. Model Comparison and Justification

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.79	0.73	0.63	0.68
Random Forest	0.81	0.75	0.70	0.72

Observations:

- The Random Forest model outperforms Logistic Regression across all metrics, with better accuracy, precision, recall, and F1-score.

- Logistic Regression is faster and easier to interpret, but Random Forest is more powerful in capturing complex patterns.

9.Visualization of Results & Model Insights

In this section, we will visualize key results from the trained models to interpret their behavior and compare their performance. We will use visualizations such as the Confusion Matrix**,** ROC Curve**,** Feature Importance Plot**,** and Residual Plots to gain insights into model predictions.

1. Confusion Matrix

The confusion matrix helps evaluate the classification performance of a model by showing the counts of true positives, true negatives, false positives, and false negatives.

Code

```
from sklearn.metrics import confusion_matrix
Import seaborn as sns

cm_logreg = confusion_matrix(y_test, y_pred_logreg)
cm_rf = confusion_matrix(y_test, y_pred_rf)

# Plot confusion matrices
fig, axes = plt.subplots(1, 2, figsize=(14, 7))

sns.heatmap(cm_logreg, annot=True, fmt='d', cmap='Blues', cbar=False, ax=axes[0])
axes[0].set_title('Logistic Regression Confusion Matrix')
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('Actual')
```

```
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues', cbar=False, ax=axes[1])
axes[1].set_title('Random Forest Confusion Matrix')
axes[1].set_xlabel('Predicted')
axes[1].set_ylabel('Actual')

plt.show()
```

Interpretation:

- The confusion matrix for Logistic Regression shows that it is classifying a substantial number of customers who did not churn as churned (false positives).
- The Random Forest model performs better, with fewer false positives and false negatives, indicating that it is more accurate at predicting customer churn.

2. ROC Curve (Receiver Operating Characteristic Curve)

The ROC curve helps evaluate the trade-off between the True Positive Rate (Recall) and the False Positive Rate across different thresholds.

Code

```
from sklearn.metrics import roc_curve, roc_auc_score

fpr_logreg, tpr_logreg, _ = roc_curve(y_test, log_reg.predict_proba(X_test)[:, 1])
fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_model.predict_proba(X_test)[:, 1])

plt.figure(figsize=(8, 6))
```

```
plt.plot(fpr_logreg, tpr_logreg, color='blue', label='Logistic Regression (AUC = {:.2f})'.format(roc_auc_score(y_test, log_reg.predict_proba(X_test)[:, 1])))
plt.plot(fpr_rf, tpr_rf, color='green', label='Random Forest (AUC = {:.2f})'.format(roc_auc_score(y_test, rf_model.predict_proba(X_test)[:, 1])))
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

Interpretation:

- The ROC curve shows that both models perform well, with Random Forest having a higher AUC (Area Under Curve)**,** indicating better overall model performance.
- The curve further suggests that Random Forest is better at distinguishing between churned and non-churned customers across various thresholds.

3. Feature Importance Plot

Feature importance helps identify which features contributed the most to the model's predictions. We will use the Random Forest model, which provides feature importance out-of-the-box.

Code

```
import pandas as pd

# Get feature importances from the Random Forest model
```

```
feature_importances = rf_model.feature_importances_
features = X.columns

# Create a DataFrame for easier visualization
feat_importance_df = pd.DataFrame({'Feature': features, 'Importance': feature_importances})
feat_importance_df = feat_importance_df.sort_values(by='Importance', ascending=False)

# Plot the top 10 most important features
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feat_importance_df.head(10))
plt.title('Top 10 Important Features for Churn Prediction')
plt.show()
```

Interpretation:

- The Feature Importance Plot indicates that tenure, MonthlyCharges, and Contract are the top three features influencing churn prediction.
- This suggests that customers with shorter tenure, higher monthly charges, and month-to-month contracts are at a higher risk of churning.

4. Model Performance Comparison

To visually compare the performance of the models, we will plot their accuracy, precision, recall, and F1-score.

Code

```
# Model comparison
metrics = pd.DataFrame({
```

```
 'Model': ['Logistic Regression', 'Random Forest'],
'Accuracy': [logreg_accuracy, rf_accuracy],
'Precision': [logreg_precision, rf_precision],
 'Recall': [logreg_recall, rf_recall],
 'F1-Score': [logreg_f1, rf_f1]
})

metrics.set_index('Model', inplace=True)

# Plot comparison
metrics.plot(kind='bar', figsize=(10, 6))
plt.title('Model Performance Comparison')
plt.ylabel('Score')
plt.show()
```

Interpretation:

- The Random Forest model consistently outperforms Logistic Regression across all evaluation metrics (accuracy, precision, recall, and F1-score).
- This reinforces the choice of Random Forest as the better model for predicting customer churn.

5. Residual Plot (Optional)

For regression models, residual plots are used to check the fit of the model. Since we are working with a classification problem, residual plots aren't necessary. However, if a regression model were used for a different task, this would be applicable.

Conclusion

- Random Forest is the preferred model for predicting churn, as it performs better than Logistic Regression across all key metrics.
- Feature importance shows that customer tenure, monthly charges, and contract type play the most significant roles in predicting churn.
- The ROC curve and Confusion Matrix further validate the performance and reliability of the Random Forest model in distinguishing between churned and non-churned customers.

10.Tools and Technologies Used

In this phase of the customer churn prediction project, the following tools and technologies were utilized for data analysis, model building, and visualization.

1. Programming Language

- **Python**: The primary programming language used throughout the project for data manipulation, model building, and visualization. Python is widely used in data science due to its rich ecosystem of libraries and ease of use.

2. IDE / Notebook

- **Jupyter Notebook**: The primary development environment used for interactive coding, data exploration, and visualizations. Jupyter Notebooks allow us to document the process while running Python code, making it ideal for iterative data science projects.
- **Google Colab**: An online, cloud-based alternative to Jupyter Notebook, which provides free access to GPU/TPU resources for faster model training and experimentation.

3. Libraries

- **pandas**: Used for data manipulation, cleaning, and processing. It allows easy handling of structured data in the form of dataframes.
- **numpy**: Used for numerical operations and working with arrays.
- **scikit-learn**: A powerful library for machine learning, which includes models (e.g., Logistic Regression, Random Forest), preprocessing tools, and evaluation metrics.
- **seaborn**: Built on top of matplotlib, used for creating attractive and informative statistical graphics such as heatmaps, pair plots, and distribution plots.
- **matplotlib**: A low-level library for plotting in Python, often used for creating custom visualizations and charts.
- **xgboost** (optional, not used here but can be integrated in future): A gradient boosting framework that is efficient for high-performance and large datasets, often used to improve model accuracy.

4. Visualization Tools

- **matplotlib**: Used to generate basic plots such as histograms, boxplots, and line charts for visualizing the dataset and model performance.
- **seaborn**: Used in combination with matplotlib for more complex statistical plots, such as correlation heatmaps and pair plots.
- **Plotly** (optional, not used here but can be integrated in future): A library for creating interactive plots, often used for dashboards and web-based visualizations.
- **Tableau** / **PowerBI** (optional, if used for business reporting): Business intelligence tools that allow the creation of rich, interactive dashboards for presenting the final results to stakeholders.

5. Other Tools (if applicable)

- **Git**: For version control, enabling team collaboration and management of different versions of the project.
- **Anaconda**: A Python distribution that simplifies package management and deployment. It is useful for managing environments in data science projects.

11.Team Members and Contributions

In this customer churn prediction project, the responsibilities were distributed across different team members. Each member contributed to various phases of the project, from data preprocessing to model development and documentation. Below are the roles and responsibilities of each team member:

1. Data Cleaning

- **Team Member 1:A.Manju**
  - **Responsibilities**:
    - Handled missing values by performing imputation and removal where necessary.
    - Removed duplicate records from the dataset.
    - Identified and treated outliers to ensure data quality.
    - Ensured consistency of data types across features.

2. Exploratory Data Analysis (EDA)

- **Team Member 2: A.Mahalakshmi**
  - **Responsibilities**:
    - Performed univariate analysis to examine the distribution of each feature.

- Conducted bivariate and multivariate analysis to identify correlations and relationships between features and the target variable.
- Generated various visualizations such as histograms, boxplots, and correlation matrices to derive insights.
- Summarized findings, highlighting key trends and relationships in the dataset.

3. Feature Engineering

- **Team Member 3: K.Kiruthiga**
- **Responsibilities**:
  - Created new features based on domain knowledge and EDA insights, such as tenure groups, total services count, and monthly charge bins.
  - Applied techniques such as binning, categorization, and dimensionality reduction.
  - Justified and documented each new feature or transformation and its potential impact on the model.

4. Model Development

- **Team Member 4: R.Bhuvana**
  - **Responsibilities**:
    - Built and trained machine learning models (Logistic Regression and Random Forest) for churn prediction.
    - Tuned model parameters and evaluated model performance using appropriate metrics like accuracy, precision, recall, and F1-score.
    - Performed model comparisons and selected the best-performing model (Random Forest).

5. Documentation and Reporting

- **Team Member 5: <u>Gayatri elangovan</u>**
    - **Responsibilities**:
        - Compiled all the project stages, from data cleaning to model evaluation, into a comprehensive report.
        - Created visualizations for the final report, such as confusion matrices, ROC curves, and feature importance plots.
        - Wrote detailed explanations for each model, visualization, and analysis step.
        - Ensured the report was well-organized, clear, and aligned with project goals.