## **Design Document: Machine Learning Pipeline for Customer Churn Prediction**

### **Overview**

This document outlines the design of a machine learning pipeline for predicting customer churn using telecom data. The pipeline includes modules for data preprocessing, visualization, outlier handling, model training, and evaluation. Each component is designed to enhance maintainability, readability, and reusability of the codebase.

### **Components**

## **STEP 1 - Preprocessing**

### **Module: step1.py**

#### **Class: Preprocessor**

**Responsibilities:**

* Load data from a CSV file.
* Encode categorical variables into numerical values.

**Attributes:**

* file\_path (str): Path to the CSV file.

**Methods:**

1. **\_\_init\_\_(self, file\_path)**
   * Initializes the Preprocessor with a CSV file path.
   * Loads the data from the specified CSV file.
2. **get\_basic\_info(self)**
   * Prints basic information about the dataset.
   * Displays the first few rows, shape, and data types of the dataset.
3. **get\_unique\_values(self, columns)**
   * Prints unique values and their counts for specified columns.
4. **remove\_outliers(self, column, method='iqr', threshold=None)**
   * Removes outliers from a specified column using Interquartile Range (IQR) or threshold method.
5. **dummify\_columns(self, columns)**
   * Converts specified categorical columns into dummy/indicator variables.
6. **handle\_missing\_values(self, strategy='mean', columns=None)**
   * Handles missing values in the dataset using specified strategies ('mean', 'median', 'mode', 'drop').
7. **scale\_features(self, columns, method='standard')**
   * Scales specified numerical features using StandardScaler or MinMaxScaler.
8. **label\_encode\_columns(self, columns)**
   * Label encodes specified categorical columns using LabelEncoder.
9. **drop\_columns(self, columns)**
   * Drops specified columns from the dataset.
10. **save\_processed\_data(self, file\_path)**
    * Saves the processed data to a CSV file.
11. **get\_numerical\_columns(self)**
    * Returns a list of numerical columns in the dataset.
12. **get\_categorical\_columns(self)**
    * Returns a list of categorical columns in the dataset.
13. **get\_data(self)**
    * Returns the processed DataFrame.

### **STEP 2 - Visualization**

#### ***Module: step2.py***

#### ***Class: Visualizer***

**Responsibilities:**

* Generate various types of plots for data exploration.

**Attributes:**

* data (DataFrame): Data to visualize.

**Methods:**

1. \_\_init\_\_(self, data)
   * Initializes with the data.
2. scatter\_plot(self, x, y, hue=None)
   * Generates a scatter plot of two variables.
   * **Parameters:**
     + x (str): Column name for x-axis.
     + y (str): Column name for y-axis.
     + hue (str, optional): Column name for color encoding.
3. bar\_plot(self, x, y)
   * Generates a bar plot of two variables.
   * **Parameters:**
     + x (str): Column name for x-axis.
     + y (str): Column name for y-axis.
4. hist\_plot(self, x)
   * Generates a histogram of a variable.
   * **Parameters:**
     + x (str): Column name for the histogram.
5. box\_plot(self, x, y)
   * Generates a box plot of two variables.
   * **Parameters:**
     + x (str): Column name for x-axis.
     + y (str): Column name for y-axis.
6. heatmap(self, annot=True, cmap='viridis')
   * Generates a heatmap of the correlation matrix.
   * **Parameters:**
     + annot (bool, optional): Boolean flag to annotate the heatmap (default: True).
     + cmap (str, optional): Colormap to use for the heatmap (default: 'viridis').
7. pair\_plot(self, hue=None)
   * Generates a pair plot of the dataset.
   * **Parameters:**
     + hue (str, optional): Column name for color encoding.
8. violin\_plot(self, x, y, hue=None)
   * Generates a violin plot of two variables.
   * **Parameters:**
     + x (str): Column name for x-axis.
     + y (str): Column name for y-axis.
     + hue (str, optional): Column name for color encoding.
9. count\_plot\_categorical(self, cat\_var)
   * Displays count plots for all categorical variables.
   * **Parameters:**
     + cat\_var (list): List of categorical variable names.
10. density\_plot\_numerical(self, num\_var)
    * Displays density plots for all numerical features.
    * **Parameters:**
      + num\_var (list): List of numerical variable names.
11. bivariate\_analysis(self, num\_var, target\_var)
    * Performs bivariate analysis using FacetGrid and box plots.
    * **Parameters:**
      + num\_var (list): List of numerical variable names.
      + target\_var (str): Target variable name.
12. class\_imbalance\_plot(self, target\_var)
    * Shows the class imbalance using a count plot.
    * **Parameters:**
      + target\_var (str): Target variable name.

### **STEP 3 - Modeling**

#### ***Module: step3.py***

#### ***Class: ModelTrainer***

**Responsibilities:**

* Split data into training and test sets.
* Train a regression model.
* Evaluate the trained model.

**Attributes:**

* data (DataFrame): Data to process.
* target\_column (str): Target variable.

**Methods:**

1. \_\_init\_\_(self, data, target\_column, drop\_cols=[])
   * Initializes with data and target column.
   * **Parameters:**
     + data (DataFrame): Input data containing predictors and target.
     + target\_column (str): Name of the target variable.
     + drop\_cols (list, optional): Columns to drop from the input data (default: []).
2. split\_data(self, test\_size=0.3, random\_state=42)
   * Splits data into training and test sets.
   * **Parameters:**
     + test\_size (float, optional): Proportion of the dataset to include in the test split (default: 0.3).
     + random\_state (int, optional): Seed used by the random number generator (default: 42).
3. train\_model(self, model\_name)
   * Trains a regression model specified by model\_name.
   * **Parameters:**
     + model\_name (str): Name of the model to train ('SVC', 'RandomForest', 'XGB', 'LogisticRegression').
4. predict(self, model\_name, X)
   * Predicts the target variable using the trained model.
   * **Parameters:**
     + model\_name (str): Name of the model used for prediction.
     + X (DataFrame): Input features for prediction.
5. evaluate\_model(self, model\_name)
   * Evaluates the trained model and returns accuracy and classification report.
   * **Parameters:**
     + model\_name (str): Name of the model to evaluate ('SVC', 'RandomForest', 'XGB', 'LogisticRegression').
   * **Returns:**
     + accuracy (float): Accuracy score of the model on the test data.
     + report (str): Classification report of the model's performance.
6. get\_train\_test\_data(self)
   * Returns the split training and test data.
   * **Returns:**
     + X\_train (DataFrame): Features for training.
     + X\_test (DataFrame): Features for testing.
     + y\_train (Series): Target variable for training.
     + y\_test (Series): Target variable for testing.

**Models:**

* SVC: Support Vector Classifier with RBF kernel.
* RandomForest: Random Forest Classifier.
* XGB: XGBoost Classifier with specified hyperparameters.
* LogisticRegression: Logistic Regression Classifier.

### **STEP 4 - Evaluation**

#### **Module: step4.py**

#### **Class: Evaluator**

**Responsibilities:**

* Calculate various evaluation metrics for model performance.
* Plot ROC curves and residual plots for model evaluation.

**Attributes:**

* scores (DataFrame): DataFrame to store evaluation metrics for different models.
* roc\_curves (dict): Dictionary to store ROC curves and AUC values.

**Methods:**

1. \_\_init\_\_(self)
   * Initializes the Evaluator with an empty scores DataFrame and an empty roc\_curves dictionary.
2. evaluate\_model(self, model\_name, y\_train, train\_pred, y\_test, test\_pred)
   * Evaluates a model using various metrics on both training and test sets.
   * **Parameters:**
     + model\_name (str): Name of the evaluated model.
     + y\_train (Series): Actual values of the target variable in the training set.
     + train\_pred (Series): Predicted values of the target variable in the training set.
     + y\_test (Series): Actual values of the target variable in the test set.
     + test\_pred (Series): Predicted values of the target variable in the test set.
   * **Prints:**
     + Accuracy score, classification report, confusion matrix, and Cohen's Kappa score for the model.
3. evaluate\_model\_with\_auc(self, model\_name, y\_train, train\_pred, y\_test, test\_pred, y\_pred\_prob)
   * Extends evaluate\_model by including ROC curve evaluation and AUC calculation.
   * **Parameters:**
     + Same as evaluate\_model, plus y\_pred\_prob (Series): Predicted probabilities of the target variable in the test set.
   * **Prints:**
     + AUC score for the ROC curve of the model.
4. plot\_roc\_curves(self)
   * Plots ROC curves for all evaluated models.
   * **Displays:**
     + ROC curves with AUC scores for each model.
5. plot\_residuals(self, y\_train, y\_train\_pred)
   * Plots residuals (difference between actual and predicted values) for the training set.
   * **Parameters:**
     + y\_train (Series): Actual values of the target variable in the training set.
     + y\_train\_pred (Series): Predicted values of the target variable in the training set.
   * **Displays:**
     + Scatter plot of predicted values versus residuals.
6. get\_scores(self)
   * Returns the scores DataFrame containing all evaluation metrics for the models.

**Dependencies:**

* numpy (as np): Numerical operations.
* pandas (as pd): Data manipulation and storage.
* seaborn and matplotlib.pyplot (as plt): Visualization tools.
* scipy.stats and statsmodels.api: Statistical functions and models.
* sklearn.metrics: Evaluation metrics such as ROC curve, AUC, accuracy score, classification report, confusion matrix, and Cohen's Kappa score.

### **Script: main.py**

**Responsibilities:**

* Orchestrates the entire data pipeline from preprocessing to evaluation.

**Steps:**

#### ***Preprocessing:***

1. **Load data.**
2. **Encode categorical columns.**

#### ***Visualization:***

1. **Generate scatter plot, bar plot, and histogram.**
2. **Display count plots and density plots for categorical and numerical variables.**
3. **Plot box plots for numerical variables categorized by churn status.**
4. **Show class imbalance using a count plot.**

#### ***Handling Outliers:***

1. **Fit regression model to identify influential points.**
2. **Remove influential points.**
3. **Get cleaned data.**

#### ***Model Training:***

1. **Split data into training and test sets.**
2. **Initialize and train multiple classification models:**
   * Support Vector Classifier (SVC)
   * Random Forest Classifier (RFC)
   * XGBoost Classifier (XGB)
   * Logistic Regression (LogReg)

#### ***Evaluation:***

1. **Calculate and print evaluation metrics:**
   * Accuracy
   * Classification report
   * Confusion matrix
   * Cohen's Kappa score
   * Area Under the Curve (AUC) for ROC curve

Conclusion

This design document outlines a robust machine learning pipeline for predicting customer churn using telecom data. Each component, from data preprocessing to model evaluation, is meticulously designed to enhance the maintainability, readability, and reusability of the codebase. By following this structured approach, the pipeline ensures comprehensive handling of data transformations, insightful visualization, rigorous outlier handling, effective model training, and thorough model evaluation.

The preprocessing module, encapsulated in `step1.py`, efficiently handles data loading, categorical encoding, outlier removal, and feature scaling. It sets a solid foundation for preparing the data for modeling by ensuring data integrity and consistency.

Visualization capabilities, implemented in `step2.py`, offer diverse plots such as scatter plots, histograms, and bar plots, providing intuitive insights into data distributions and relationships. These visualizations aid in understanding feature distributions, detecting patterns, and exploring potential correlations.

Model training and evaluation, managed in `step3.py` and `step4.py`, respectively, introduce a variety of classifiers including Support Vector Machines, Random Forests, XGBoost, and Logistic Regression. The pipeline splits the data, trains multiple models, evaluates their performance using key metrics like accuracy, classification reports, ROC curves, and AUC scores. This systematic evaluation ensures robust performance assessment and model selection.

The `main.py` script orchestrates the entire pipeline, seamlessly integrating preprocessing, visualization, model training, and evaluation steps. It serves as a cohesive framework to automate the end-to-end process of churn prediction, making it adaptable for different datasets and scalable for future enhancements.

In conclusion, this machine learning pipeline not only predicts customer churn effectively but also promotes best practices in software design and data science, ensuring reliability and efficacy in churn prediction applications.