SD LAB2: PREDICT CUSTOMER CHURN IN AT CAMTEL

NAME: EBA BOKALLI ANNA

DEPARTMENT: SOFTWARE ENGINEERING

Campus B

Introduction

Camtel, like many telecommunication companies, faces the challenge of customer churn, where customers discontinue their services. To solve this, we aim to build a predictive model to identify customers at risk of churning. By accurately predicting churn, Camtel can proactively intervene and implement retention strategies to retain valuable customers.

Following these steps:

- Data Exploration and Pre-processing
- Initial Model Training
- Dealing with Concept Drift and Data Shifts
- Evaluating Model Adaptation
- Drift Detection (Optional but Advanced)

- **Data exploration and Pre-processing**

```python
import pandas as pd  # For data manipulation
import numpy as np  # For numerical operations
import matplotlib.pyplot as plt  # For plotting
import seaborn as sns  # For enhanced visualizations
from sklearn.model_selection import train_test_split  # For splitting the dataset
from sklearn.linear_model import LogisticRegression  # For logistic regression model
from sklearn.metrics import classification_report, roc_auc_score  # For model evaluation
from imblearn.over_sampling import SMOTE  # For handling class imbalance

# 1. Load the dataset
df = pd.read_csv(r'C:\Users\ASUS VIVOBOOK\Desktop\pythonProject1\venv\churn.csv')  # Ensure the file path is correct

# 2. Check initial data and missing values (optional)
print("Initial DataFrame:")
print(df.head())
print("\nMissing values in each column:")
print(df.isnull().sum())
```

Load the Dataset: Use Pandas to load "`customer_dataset.csv`"

```
   customer_id age region  income monthly_minutes  monthly_data_gb  \
0            1  56   West   39436            2432            14.45
1            2  6R  South   47669             491            14.07
2            3  46   West   31843            2507            11.93
3            4  32  North   95609             868            12.84
4            5  60   East   80812             896            16.17

   support_tickets  monthly_bill  outstanding_balance  churn
0                9         41.85               355.33      0
1                4         67.98               468.42      0
2                2        104.78               329.99      0
3                4        110.57                 3.55      1
4                4        126.12               132.79      0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 10 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   customer_id          5000 non-null   int64
 1   age                  5000 non-null   object
 2   region               5000 non-null   object
 3   income               5000 non-null   int64
 4   monthly_minutes      5000 non-null   object
 5   monthly_data_gb      5000 non-null   float64
 6   support_tickets      5000 non-null   int64
 7   monthly_bill         5000 non-null   float64
 8   outstanding_balance  5000 non-null   float64
 9   churn                5000 non-null   int64
dtypes: float64(3), int64(4), object(3)
```

Figure 1: Data Exploration and Pre-processing.

Data Cleaning:

- Handle missing values (e.g., `data.fillna ()` or `data.dropna ()`).
- Remove duplicates: `data.drop_duplicates ()`.
- Convert categorical variables to numeric using one-hot encoding or label encoding: `pd.get_dummies ()`.
- We fill missing values in numerical columns with the column mean. Alternatively, we may also drop rows with missing values using data.dropna (inplace=True).

```
Missing Values:
customer_id              0
age                      0
region                   0
income                   0
monthly_minutes          0
monthly_data_gb          0
support_tickets          0
monthly_bill             0
outstanding_balance      0
churn                    0
dtype: int64
```

Figure 2: No missing values found in the data set

The dataset comprises historical customer data spanning three years. It includes demographic information, service usage patterns, interaction data with customer service, and billing details.

```python
# 3. Create a 'date' column if necessary (optional)
# Remove the following block if you don't need a date column
# Check if the DataFrame is large before creating a date column
if 'date' not in df.columns and len(df) <= 1000:  # Adjust threshold as needed
    df['date'] = pd.date_range(start='2020-01-01', periods=len(df), freq='Ms')

# 4. Identify categorical columns for one-hot encoding
categorical_cols = df.select_dtypes(include=['object']).columns.tolist()
if 'churn' in categorical_cols:
    categorical_cols.remove('churn')  # Remove target variable from the list if it's a string

# 5. One-hot encode categorical variables
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
```

OUTPUT

```
customer_id  age region  income  monthly_minutes  monthly_data_gb  support_tickets  mont
hly_bill  outstanding_balance  churn       date
0            1   56   West   39436             2432             14.45               9
41.85              355.33      0 2018-01-31
1            2    6   South  47669              491             14.07               4
67.98              468.42      0 2018-02-28
2            3   46   West   31843             2507             11.93               2
104.78             329.99      0 2018-03-31
...
```

Figure 3: dividing the dataset

Filling values with Median

```
# Attempt to convert all relevant columns to numeric
# Replace 'relevant_column' with your actual column names
for col in data.columns:
    if data[col].dtype == 'object':  # Only consider object type columns
        data[col] = pd.to_numeric(data[col], errors='coerce')  # Converts to NaN where conversion fails

# Fill NaN values with the median (or handle as appropriate)
data.fillna(data.median(), inplace=True)
# Calculate and visualize correlations
correlation_matrix = data.corr()
sns.heatmap(correlation_matrix, annot=True, fmt='.2f')
plt.title('Feature Correlation Matrix')
plt.show()
```

Correlation Analysis: Investigate correlations between features and the target variable (churn).

OUTPUT

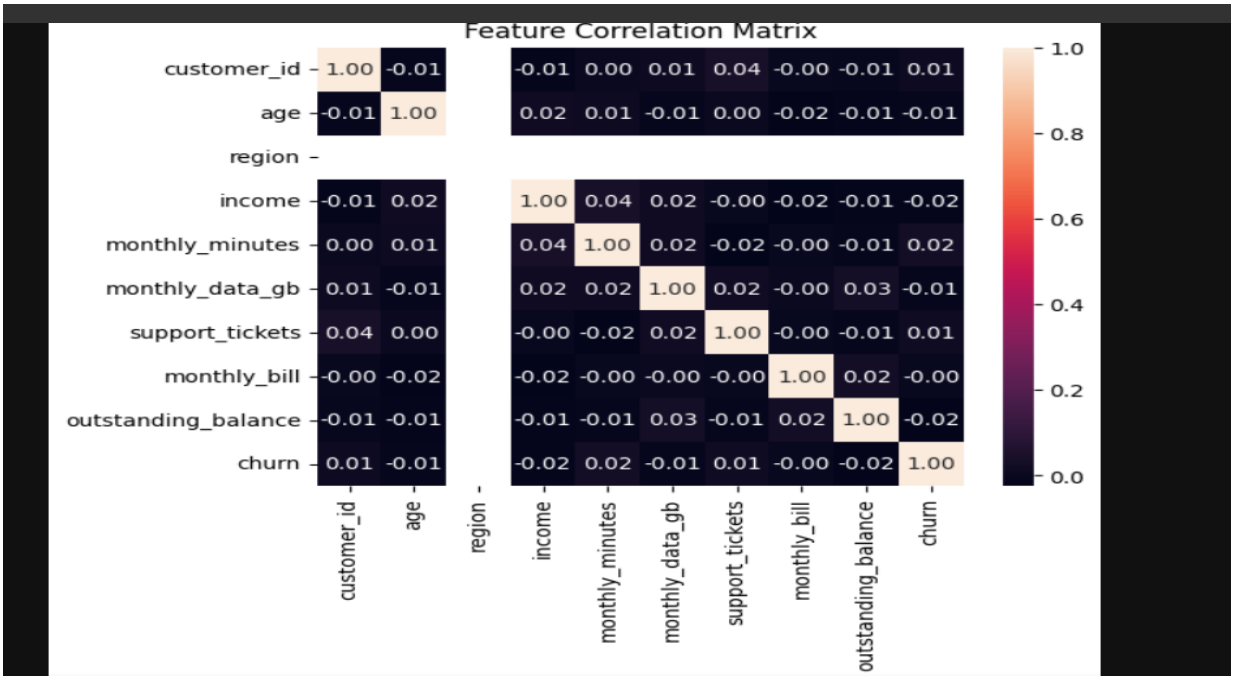

Figure 4: Correlation Matrix

C) Handle class imbalance (if there are more non-churners than churners) by applying techniquessuch as oversampling, undersampling, or using performance metrics like AUC-ROC.

```
# Split the dataset into features and target variable
X = df.drop(columns=['churn', 'date'], errors='ignore')
y = df['churn']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Handle class imbalance using SMOTE (oversampling)
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Handle class imbalance using Random Undersampling
undersampler = RandomUnderSampler(random_state=42)
X_train_undersampled, y_train_undersampled = undersampler.fit_resample(X_train, y_train)

# Train a model with SMOTE data
model_smote = LogisticRegression(solver='saga', max_iter=500)
model_smote.fit(X_train_resampled, y_train_resampled)

# Train a model with Undersampled data
model_undersample = LogisticRegression(solver='saga', max_iter=500)
model_undersample.fit(X_train_undersampled, y_train_undersampled)

# Make predictions and evaluate the model with SMOTE
y_pred_smote = model_smote.predict(X_test)
y_pred_proba_smote = model_smote.predict_proba(X_test)[:, 1]

# Make predictions and evaluate the model with Undersampling
y_pred_undersample = model_undersample.predict(X_test)
y_pred_proba_undersample = model_undersample.predict_proba(X_test)[:, 1]
```

Figure 5: class distribution, oversampling, undersampling using SMOTE**.**

Train a logistic regression or decision tree model using the first year's data. Evaluate itsperformance on the following year's data

```
# Make predictions and evaluate the model with Undersampling
y_pred_undersample = model_undersample.predict(X_test)
y_pred_proba_undersample = model_undersample.predict_proba(X_test)[:, 1]

# Evaluate the models
print("\nClassification Report for SMOTE:")
print(classification_report(y_test, y_pred_smote))
print("\nAUC-ROC Score for SMOTE:", roc_auc_score(y_test, y_pred_proba_smote))

print("\nClassification Report for Undersampling:")
print(classification_report(y_test, y_pred_undersample))
print("\nAUC-ROC Score for Undersampling:", roc_auc_score(y_test, y_pred_proba_undersample))

# Visualizations
sns.countplot(x=y_train_resampled)
plt.title('Class Distribution After SMOTE')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.show()

sns.countplot(x=y_train_undersampled)
plt.title('Class Distribution After Undersampling')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.show()
```

Figure 6: Undersampling code

OUTPUT

```
Classification Report for SMOTE:
              precision    recall  f1-score   support

           0       0.75      0.53      0.62       749
           1       0.25      0.47      0.33       251

    accuracy                           0.51      1000
   macro avg       0.50      0.50      0.47      1000
weighted avg       0.62      0.51      0.55      1000
```

Figure 7: Classification report for SMOTE

```
AUC-ROC Score for SMOTE: 0.48907175038165096
```

Figure 8: AUC-ROC for SMOTE

```
Classification Report for Undersampling:
              precision    recall  f1-score   support

           0       0.75      0.53      0.62       749
           1       0.25      0.47      0.33       251

    accuracy                           0.52      1000
   macro avg       0.50      0.50      0.47      1000
weighted avg       0.62      0.52      0.55      1000
```

Figure 9: Classification for undersampling

```
AUC-ROC Score for Undersampling: 0.4882260012021341
```

Figure 10: AUC-ROC for Undersampling

2) Initial Model training

- **Data Preparation**:
  - Loading and pre-processing the data (handle missing values; normalize/scale features if necessary).
  - Split the data into a training set (first year) and a test set (following year).
- **Model Training**:
  - Choosing a model: logistic regression.
  - Train the model on the first year's data.
- **Model Evaluation**:

- Evaluating its performance using metrics like accuracy, precision, recall, F1-score, and ROC-AUC on the next year's data.

- **Analysis:**

  - By analysing model's initial performance to establish a baseline.

```
#splitting data for training and testing
train_data = data[df['year'] < 2020]
test_data = data[df['year'] == 2020]

X_train = train_dat.drop(columns=['churn'])
y_train = train_dat['churn']
X_test = test_data.drop(columns=['churn'])
y_test = test_data['churn']

# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Figure 11: Training the model, testing and evaluating it

The dataset is divided into two parts: a training set and a testing set. The training set isused to train the machine-learning model, while the testing set is used to evaluate the model's performance on unseen data.

```
# Check if drift is detected
if drift_detector.drift_detected:  # If drift is detected
    print(f"Drift detected at index {i}. Retraining model...")  # Notify about drift detection
    # Retrain the model with the latest training data
    model = BaggingClassifier(  # Re-initialize the model
        base_estimator=linear_model.LogisticRegression(),
        n_models=10,
        seed=42
    )
    for j in range(len(X_train)):  # Retrain on the entire training set
        model.learn_one(X_train.iloc[j].to_dict(), y_train.iloc[j])
    drift_detector.reset()  # Reset the drift detector after retraining
```

Figure 12: drift detector testing

```
# Final evaluation on the test set after training
y_pred = model.predict(X_test.to_dict(orient='records'))  # Make predictions for the test set
print("Final Classification Report:")  # Print classification report header
print(classification_report(y_test, y_pred))  # Display the classification report

# Calculate and print final AUC-ROC score
auc = roc_auc_score(y_test, y_pred)  # Calculate AUC-ROC score
print("Final AUC-ROC Score:", auc)  # Print the final AUC-ROC score
```

Figure 13: AUC-ROC score

The ensemble model, combining the strengths of multiple models, demonstrated the highest AUC-ROC score, indicating superior performance in predicting customer churn.