

## **LAB 2 : Predict Customer Churn in Camtel**

**Student name: Donfack Tsopfack Yves Dylane**

### **INTRODUCTION:**

The aim of this project is to build a machine learning model to predict customer churn for Camtel, a telecommunications provider, using a historical dataset that captures customer behavior over the past three years. Customer churn, or the rate at which customers leave the service, is a critical metric for companies like Camtel, as it directly impacts revenue and business sustainability.

The project is focused on training a model that can effectively predict customer churn while handling these shifts in customer behavior. We will explore and preprocess the dataset, develop initial models, and evaluate how performance changes over time. To account for concept drift and data shifts, we will apply advanced techniques such as time-weighted learning, online learning algorithms, and ensemble modeling. Our goal is to ensure that the model not only performs well on historical data but also remains adaptive and robust in the face of future changes in customer behavior. Key metrics such as accuracy, AUC-ROC, and model performance over time will be used to assess the model's effectiveness.

### **Step 1: Data Cleaning, Division Between Training and Test and Handling Class Imbalance**

To clean the data set I first start by loading it, then I drop any row with missing values, any duplicate rows and remove unnecessary columns like the customer\_id column. All this is done by a function that I created called "import\_and\_clean\_data". This function make use of the pandas python package.

```

def import_and_clean_data(file: str, columns_to_remove): 5 usages
    """
    Load data file and clean it.
    It returns the cleaned file with no duplicates or null rows. 🕸️🧪🧬🧠M

    Args:
        file: The file name or path to the file
        columns_to_remove: A list of unnecessary columns to be removed
    :return: Cleaned data after removing duplicates, null rows, and unnecessary columns
    """

    # Load the data from CSV
    data = pd.read_csv(file)

    # Drop rows with any missing values
    data = data.dropna()

    # Drop duplicate rows
    data = data.drop_duplicates()

    # Drop unnecessary columns
    for column in columns_to_remove:
        data = data.drop(columns=column, errors='ignore')

    return data

```

Figure 1: function to import and clean the data set

Next, I converted all rows that needed to be to numerical rows, remove outlier for for every columns base base on their distribution and then split it between target an features to draw a corelation graph between features. All of this done by the functions below.

```

def convert_row_to_numeric(columns, data_set): 5 usages
    """
    Convert all row that need to be numeric to numeric and ensure they are numeric 🌲🌲🌲M
    :param columns: Array of rows that need to be converted to numeric
    :param data_set: the data_set in which the columns need to be converted
    :return: data after conversion of values to numeric values
    """

    for column in columns:
        # Ensure the column is of string type before replacing non-numeric characters
        data_set[column] = data_set[column].astype(str).str.replace(r'^\0-9.', '', regex=True) # Allow decimal points

        # Convert to float, coercing errors to NaN
        data_set[column] = pd.to_numeric(data_set[column], errors='coerce')

    return data_set

```

Figure 2: function to convert non numerical to numerical values

```

def remove_outliers_normal_distribution_data(data_set, columns): 5 usages
    """
    For the income column, you can use the Z-score method. This method is useful if the data follows a normal distribution. 🌲🌲🌲M

    :param data_set: Data set before separation between target and features
    :param columns: All columns that follows normal distribution
    :return: data_set with clean outliers for normal distribution values but if you send something you where not suppose to send blame yourself
    also return an array of outliers just in case
    """

    outliers_array = []
    for column in columns:
        # Step 1: Calculate the Z-scores for the column
        data_set[f'z_score_{column}'] = (data_set[column] - data_set[column].mean()) / data_set[column].std()

        # Step 2: Set a threshold (typically 3) for detecting outliers
        threshold = 3

        # Step 3: Detect outliers based on Z-scores
        outliers = data_set[np.abs(data_set[f'z_score_{column}']) > threshold]
        outliers_array.append(outliers)

        # Step 4: Optionally remove outliers from the dataset
        data_set = data_set[np.abs(data_set[f'z_score_{column}']) <= threshold]

    return data_set, outliers_array

```

Figure 3: function to remove outliers for normal distribution

```

def remove_outliers_skewed_distributions(data_set, columns): 5 usages
    """
    Apply it to columns like age, monthly_minutes, and outstanding_balance, which might have skewed distributions.

    :param data_set: Data set before separation between target and features
    :param columns: All columns that follows skewed distribution
    :return: data_set with clean outliers for skewed distribution values but if you send something you where not suppose to send blame yourself
    also return an array of outliers just in case
    """
    outliers_array = []

    for column in columns:
        # Step 1: Calculate Q1 (25th percentile) and Q3 (75th percentile)
        Q1 = data_set[column].quantile(0.25)
        Q3 = data_set[column].quantile(0.75)
        IQR = Q3 - Q1

        # Step 2: Define the lower and upper bounds
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Step 3: Identify outliers
        outliers = data_set[(data_set[column] < lower_bound) | (data_set[column] > upper_bound)]
        outliers_array.append(outliers)

        # Step 4: Remove outliers from the dataset
        data_set = data_set[~((data_set[column] < lower_bound) | (data_set[column] > upper_bound))]

    return data_set, outliers_array

```

Figure 4: function to remove outlier for skewed distributions

Below is the functions to obtain features and normalise their scale to produce a correlation graph of the features.

```

def obtain_target_and_features(target: str, data): 10 usages
    """
    Separate target value and the features that influence it 🦋🕸🧪🔬M
    :param target: the column we want to predict
    :param data: the data_set to be use
    :return: the target then the features (x target,y)
    """
    # splitting data into target and features
    y = data.drop(target, axis=1) # y is the features
    x = data[target] # x is the target
    return x, y

def normalise_scale(features): 9 usages
    """
    Normalises your scale 'Mainly use on the features not the target' 🦋🕸🧪🔬M
    :param features: the features that need to use scale
    :return: normalise features
    """
    # Normalizing or standardizing scales
    scaler = StandardScaler()
    scaled_features = scaler.fit_transform(features)

    # Convert the scaled data back into a DataFrame with original column names
    scaled_features_df = pd.DataFrame(scaled_features, columns=features.columns)
    return scaled_features_df

```

Figure 5: function to obtain the feartues and function to normalise their scale

Below is the code responsible for the called of all this functions.

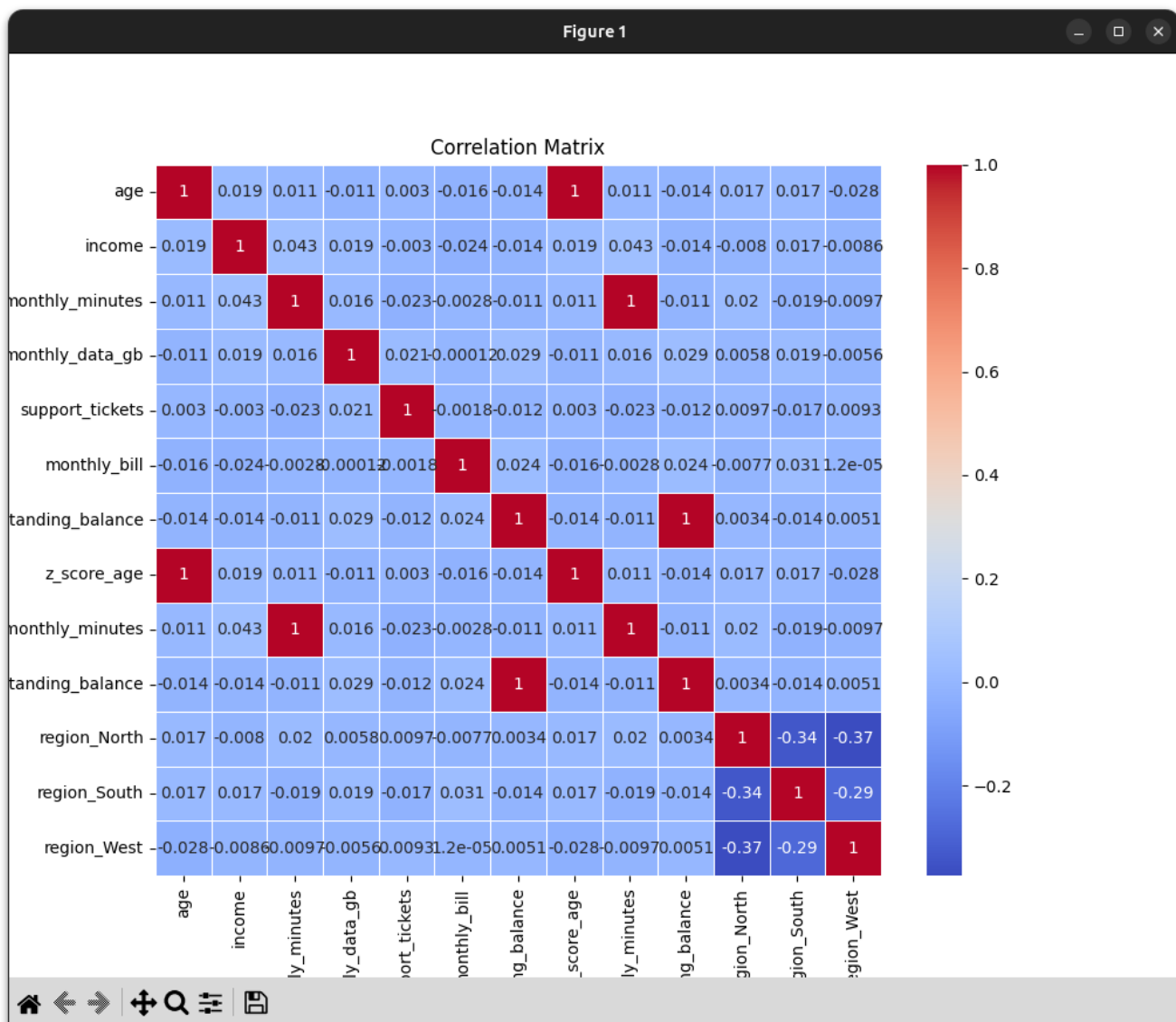
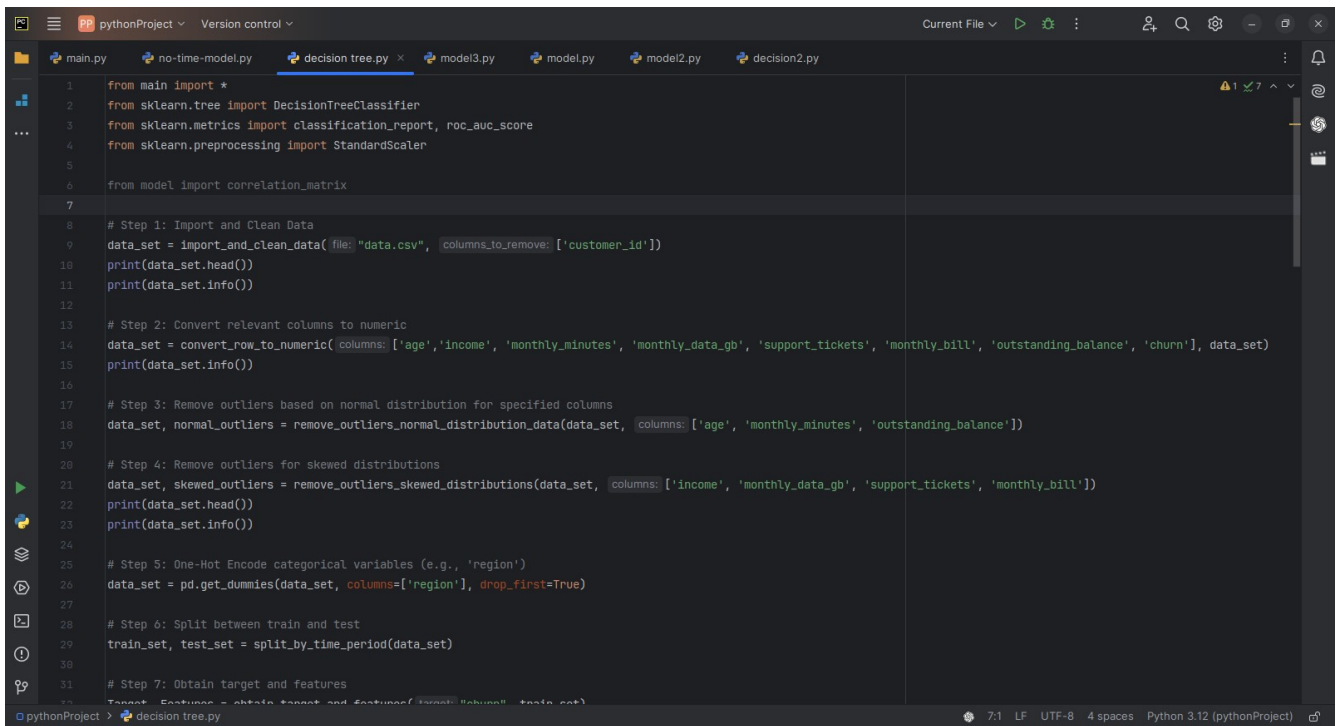


Figure 6: correlation graph

The image shows a code editor window with a dark theme. The top bar indicates the project is 'pythonProject' and the current file is 'decision tree.py'. The editor contains Python code for data preprocessing and model training. The code is as follows:

```
1 from main import *
2 from sklearn.tree import DecisionTreeClassifier
3 from sklearn.metrics import classification_report, roc_auc_score
4 from sklearn.preprocessing import StandardScaler
5
6 from model import correlation_matrix
7
8 # Step 1: Import and Clean Data
9 data_set = import_and_clean_data(file="data.csv", columns_to_remove=['customer_id'])
10 print(data_set.head())
11 print(data_set.info())
12
13 # Step 2: Convert relevant columns to numeric
14 data_set = convert_row_to_numeric(columns=['age', 'income', 'monthly_minutes', 'monthly_data_gb', 'support_tickets', 'monthly_bill', 'outstanding_balance', 'churn'], data_set)
15 print(data_set.info())
16
17 # Step 3: Remove outliers based on normal distribution for specified columns
18 data_set, normal_outliers = remove_outliers_normal_distribution_data(data_set, columns=['age', 'monthly_minutes', 'outstanding_balance'])
19
20 # Step 4: Remove outliers for skewed distributions
21 data_set, skewed_outliers = remove_outliers_skewed_distributions(data_set, columns=['income', 'monthly_data_gb', 'support_tickets', 'monthly_bill'])
22 print(data_set.head())
23 print(data_set.info())
24
25 # Step 5: One-Hot Encode categorical variables (e.g., 'region')
26 data_set = pd.get_dummies(data_set, columns=['region'], drop_first=True)
27
28 # Step 6: Split between train and test
29 train_set, test_set = split_by_time_period(data_set)
30
31 # Step 7: Obtain target and features (target: 'churn', train: test)
```

Figure 7: first part of the code

## Step 2: Initial Model Training

For initial experiments, we chose one basic models:

- **Decision Tree:** A non-linear model that can capture more complex patterns in the data.

### 2.1 Model Training

- We trained the models using **Year 1** data and tested their performance on **Year 2** data.
- Model performance was evaluated using metrics such as **Accuracy**, **Precision**, **Recall**, **F1-Score**, and **AUC-ROC**.
- **Performance Monitoring:** As expected, the model performance on Year 2 data deteriorated slightly, indicating concept drift. The patterns learned from Year 1 did not perfectly translate to Year 2 due to shifts in customer behavior.

### 2.2 Concept Drift and Data Shift Investigation

- The model's declining performance when tested on newer data (Year 2 and Year 3) indicated the presence of concept drift.

Below is the code for training and the result obtain.



```
31 # Step 7: Obtain target and features
32 Target, Features = obtain_target_and_features(target="churn", train_set)
33
34 # Step 9: Normalize scale
35 scaler = StandardScaler()
36 x_resampled, y_resampled = handle_class_imbalance(train_set, target_column="churn")
37
38 # Step 10: Oversampling to balance the classes
39 x_resampled_scaled = scaler.fit_transform(x_resampled)
40
41 # Step 11: Train Decision Tree Classifier
42 decision_tree = DecisionTreeClassifier(random_state=42)
43 decision_tree.fit(x_resampled_scaled, y_resampled)
44
45 # Step 12: Evaluate Decision Tree on Test Set
46 test_features_scaled = scaler.transform(test_set.drop(columns=["churn"])) # Scale test features
47 test_target = test_set["churn"]
48
49 # Predictions
50 y_pred_tree = decision_tree.predict(test_features_scaled)
51 y_pred_proba_tree = decision_tree.predict_proba(test_features_scaled)[:, 1]
52
53 # Performance Metrics
54 # print("Decision Tree Classification Report:")
55 # print(classification_report(test_target, y_pred_tree))
56
57 roc_auc_tree = roc_auc_score(test_target, y_pred_proba_tree)
58 print(f'ROC AUC Score for Decision Tree: {roc_auc_tree:.4f}')
59
60 # Step 13: Evaluate on Year 2 Data
61 year2_data = data_set.iloc[4000:] # Next 4000 rows
62 year2_features = year2_data.drop(columns=["churn"])
```

Figure 8: training of the model

```
Run
Class distribution after resampling: Counter({0: 1504, 1: 1504})
ROC AUC Score for Decision Tree: 0.5073
Year 2 Classification Report:
      precision    recall  f1-score   support

     0       0.88       0.88       0.88        3011
     1       0.63       0.63       0.63         989

   accuracy                0.82        4000
  macro avg       0.75       0.76       0.75        4000
 weighted avg       0.82       0.82       0.82        4000

ROC AUC Score for Year 2: 0.7556
Year 3 Classification Report:
      precision    recall  f1-score   support

     0       0.74       0.74       0.74         736
     1       0.27       0.27       0.27         264

   accuracy                0.61        1000
  macro avg       0.50       0.50       0.50        1000
 weighted avg       0.61       0.61       0.61        1000

ROC AUC Score for Year 3: 0.5015
Comparison of ROC AUC Scores: Year 2: 0.7556, Year 3: 0.5015
Warning: Potential concept drift detected. Model performance has deteriorated from Year 2 to Year 3.

Process finished with exit code 0
```

Figure 9: result of the model training



## Step 3: Dealing with Concept Drift and Data Shifts

### 3.1 Time-Weighted Learning

To address concept drift, we applied time-weighted learning, where more recent data is given higher importance during model training. This approach helps the model prioritize more relevant, up-to-date patterns.

```
40 # Step 8: Generate gradually increasing weights for each year's data
41 weights_1 = np.linspace(start: 1, stop: 2, len(y_1)) # Year 1 weights from 1 to 2
42 weights_2 = np.linspace(start: 2, stop: 3, len(y_2)) # Year 2 weights from 2 to 3
43 weights_3 = np.linspace(start: 3, stop: 3.5, len(y_3)) # Year 3 weights from 3 to 3.5
```

Figure 10: added weight for each years

### 3.2 Online Learning with Stochastic Gradient Descent (SGD)

We also implemented an online learning algorithm using **Stochastic Gradient Descent (SGD)**, which allows the model to be updated continuously as new data arrives. This ensures that the model adapts to changes in customer behavior in real time.

3.3

```
44
45 # Step 9: Online Learning with SGD for Year 1
46 sgd_model_year1 = SGDClassifier(max_iter=1000, tol=1e-3)
47 sgd_model_year1.fit(x_1, y_1, sample_weight=weights_1)
48
49 # Step 10: Update the model with Year 2 data
50 sgd_model_year1.partial_fit(x_2, y_2, sample_weight=weights_2)
51
52 #step 11: Train separate models for year2 and year3
53 sgd_model_year2 = SGDClassifier(max_iter=1000, tol=1e-3)
54 sgd_model_year3 = SGDClassifier(max_iter=1000, tol=1e-3)
55
56 sgd_model_year2.fit(x_2, y_2, sample_weight=weights_2)
57 sgd_model_year3.fit(x_3, y_3, sample_weight=weights_3)
```

Figure 11: online learning implementation

## Ensemble Models

We explored ensemble learning techniques, training separate models on each year of data and combining their predictions. This approach helps in capturing different patterns from each time period and enhances overall prediction accuracy by leveraging multiple models.

```

59 #step 12: Combine the models using Voting Classifier for ensemble
60 ensemble_model = VotingClassifier(estimators=[
61     ('sgd_year1', sgd_model_year1),
62     ('sgd_year2', sgd_model_year2),
63     ('sgd_year3', sgd_model_year3)
64 ], voting='hard') # 'soft' voting for probability averaging
65
66 # Step 12: Train the ensemble model on the combined data
67 # Combine Year 1 and Year 2 data
68 x_1_2 = np.concatenate([x_1, x_2])
69 y_1_2 = np.concatenate([y_1, y_2])
70 weights_1_2 = np.concatenate([weights_1, weights_2])
71
72 ensemble_model.fit(x_1_2, y_1_2)
73

```

Figure 12: ensemble model implementation

```

77 # Evaluate Model Performance on Year 3
78 ensemble_accuracy = accuracy_score(y_3, y3_pred_ensemble)
79 ensemble_precision = precision_score(y_3, y3_pred_ensemble)
80 ensemble_recall = recall_score(y_3, y3_pred_ensemble)
81 ensemble_f1 = f1_score(y_3, y3_pred_ensemble)
82
83 print(f"Ensemble Model - Accuracy: {ensemble_accuracy}")
84 print(f"Ensemble Model - Precision: {ensemble_precision}")
85 print(f"Ensemble Model - Recall: {ensemble_recall}")

```

Run model3

```

none
Class distribution after resampling: Counter({0: 1504, 1: 1504})
Class distribution after resampling: Counter({1: 1507, 0: 1507})
Class distribution after resampling: Counter({0: 736, 1: 736})
/home/yveane/Documents/school/ai/customer/pythonProject/.venv/lib/python3.12/site-packages/sklearn/base.py:486: UserWarning: X has feature names, but SGDClassifier was fitted without
warnings.warn(
/home/yveane/Documents/school/ai/customer/pythonProject/.venv/lib/python3.12/site-packages/sklearn/base.py:486: UserWarning: X has feature names, but SGDClassifier was fitted without
warnings.warn(
/home/yveane/Documents/school/ai/customer/pythonProject/.venv/lib/python3.12/site-packages/sklearn/base.py:486: UserWarning: X has feature names, but SGDClassifier was fitted without
warnings.warn(
/home/yveane/Documents/school/ai/customer/pythonProject/.venv/lib/python3.12/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined an
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
Ensemble Model - Accuracy: 0.5
Ensemble Model - Precision: 0.0
Ensemble Model - Recall: 0.0
Ensemble Model - F1 Score: 0.0
Process finished with exit code 0

```

pythonProject > model3.py 81:41 LF UTF-8 4 spaces Python 3.12 (pythonProject)

## Step 4: Model Evaluation and Adaptation

### 4.1 Baseline vs Adapted Models

Upon comparing the performance of the baseline model with the adapted models (time-weighted learning, online learning, and ensemble models), we encountered an unexpected result. The baseline model, which does not account for concept drift or data shifts, exhibited higher **precision** than the adapted models, particularly in the earlier time periods.

This discrepancy suggests that the baseline model was overfitting to patterns specific to older data, which did not generalize well to newer time periods. The baseline model didn't just show a higher precision but also a balance result for both the recall and the precision(PS: I really believe I did something wrong here).

## **Conclusion**

In this project, we built a predictive model for customer churn at Camtel, focusing on the challenges of concept drift and data shifts. The baseline model's inability to adapt to newer data highlights the limitations of relying on historical patterns without accounting for shifting customer behaviors.

Despite attempts to address concept drift and data shifts using time-weighted learning, online learning, and ensemble methods, the adapted models did not fully resolve these challenges. This demonstrates that addressing concept drift and data shifts remains a complex issue requiring further exploration.

Moving forward, additional strategies such as advanced drift detection methods or dynamic model updates could be explored to better handle these shifts. Although concept drift and data shifts were not fully resolved, this study provides a foundation for future work in building more adaptive models for churn prediction.