Predicting Customer Churn using Feed-Forward Neural Networks

Michael Murphy

1. Framing the Problem

Problem Statement:

The goal of this project is to predict customer churn for the telecommunications company Expresso. Churn prediction is crucial for telecommunications companies as it helps them identify customers who are likely to switch to a competitor or terminate their service. By accurately predicting churn, the company can take proactive measures to retain customers and minimize revenue loss.

Stakeholders and their Needs:

Telecommunications Companies: Predicting customer churn allows telecommunications companies to implement targeted retention strategies. This helps them reduce customer attrition rates, improve customer satisfaction, and enhance overall business performance.

Marketing and Sales Teams: Churn prediction enables marketing and sales teams to identify customers at a higher risk of churning. They can then design personalized offers and incentives to retain these customers, improving customer loyalty and increasing revenue.

Customer Support Teams: Predicting churn helps customer support teams proactively address customer issues and concerns, providing timely assistance and personalized solutions to prevent customer attrition.

Data Requirements:

To build an effective churn prediction model, relevant customer data is essential. This includes historical customer information, usage patterns, billing data, service preferences, customer interactions, and any additional data that can provide insights into customer behavior. The data should be representative of the customer population and include both churned and non-churned customers.

Data provided:

The data provided by Expresso, a telecommunications company, includes various features related to user behavior and characteristics for churn prediction. These features cover aspects such as user demographics, tenure, financial aspects (e.g., amount paid, revenue generated), usage patterns (e.g., recharge frequency, data usage), communication behavior (e.g., calls made to different networks), merger status, subscription history, top service package subscribed, and frequency of using the top package. The target variable, "CHURN," indicates whether a user has churned or not. The dataset aims to capture user-specific information that can be utilized to predict churn, assisting the company in understanding customer retention and implementing targeted strategies to reduce churn rates.

2. Approach

Preprocessing of Data

The Expresso data has 19 columns and around 2 million rows, the data was filtered down to 200k rows to speed up the model training process. The preprocessing steps are outlined below:

Missing Value Imputation: The customer churn dataset had missing values in various columns. For numeric columns, missing values were replaced using an IterativeImputer. This imputer models each feature with missing values as a function of other features and uses that estimate for imputation. It does so in an iterated round-robin fashion, treating every variable as an output in turn.

Bayesian Encoding: Bayesian encoding was applied on categorical columns. Each category in a categorical feature is replaced by the Bayesian estimate of the target variable. This has the effect of replacing the categorical variable with a continuous one that maintains some of the information about the target.

Label Encoding: After handling missing values, label encoding is applied to categorical columns. Label encoding converts each value in a column to a number. This encoding is needed because neural networks require their input to be numerical.

Class Imbalance Handling: The target variable ("CHURN") is binary and was imbalanced with only 18.7% of the target variable being classified as CHURN, meaning there are significantly more instances of not churned than churned customers. This imbalance can cause issues as models tend to be biased towards the majority class. To deal with this, oversampling was performed on the minority class (where "CHURN" equals 1) to match the number of samples in the majority class. This created a balanced dataset, which should lead to a less biased model.

3. Model architecture

The evaluation involved three distinctive models, with each varying in complexity and design, with the Lightning Module model delivering the highest recall. The first model was a straightforward feed-forward neural network with a single hidden layer, followed by a ReLU activation function, and concluding with an output layer. This simple architecture served as a baseline for the subsequent, more sophisticated models. The second model expanded upon the first by including an additional hidden layer, both of which were linear layers. Each hidden layer was followed by the application of a LeakyReLU activation function, a variant of the standard ReLU activation function designed to allow a small, non-zero gradient when the unit is not active, mitigating the problem of dying neurons encountered in ReLU. The best performing model used lightning module which used the following model architecture:

Base Class: The model is a subclass of the nn.Module class, which is the base class for all neural network modules in PyTorch.

Initialization: The __init__ method defines the layers of the neural network. The method takes the size of the input layer, the size of the hidden layer, and the size of the output layer as parameters.

Hidden Layers: There are two hidden layers in the neural network. Both hidden layers are linear layers (nn.Linear) with the given hidden size, and both use the LeakyReLU activation function (nn.LeakyReLU).

Output Layer: The output layer is also a linear layer (nn.Linear) with the given output size.

Forward Pass: The forward method implements the forward pass of the neural network. It takes an input tensor, passes it through the two hidden layers and the output layer, and returns the result. Each pass through a hidden layer is followed by the application of the LeakyReLU activation function.

4. Training Process

The training process for the first model begins by setting up the Cross-Entropy Loss function. The chosen optimization strategy is Stochastic Gradient Descent (SGD), set with a learning rate of 0.01. The model's training loop then processes the training data over three epochs. In each epoch, the model generates predictions based on the input data, computes the loss by contrasting the predicted output with the real labels, and then carries out a backpropagation procedure to calculate gradients. These gradients are utilized by the SGD optimizer to update the model's parameters. The training data is divided into batches using PyTorch's DataLoader, and shuffled before each epoch to prevent the model from recognizing and adapting to any order patterns in the training samples.

The training process for the second model starts by defining the Cross-Entropy Loss function. The optimization algorithm in this instance is Stochastic Gradient Descent (SGD), configured with a learning rate of 0.1. The training procedure occurs over three epochs, within the epochs the model processes the training data in batches. From there the process is the same as the first model. Below is the training process for the lightning module model, the best performing model:

Loss Function: The loss function is the cross-entropy loss (nn.CrossEntropyLoss)

Optimizer: The optimizer is Adam (optim.Adam), the learning rate is set to 0.001.

Training Loop: The training loop is defined in the training_step method. In each step of the training loop, the model takes a batch of inputs(features) and labels, computes the output of the model for the batch of inputs, computes the loss by comparing the output to the labels, and returns the loss. The self.log call logs the training loss for this step.

Validation Loop: The validation loop is defined in the validation_step method. Similar to the training loop, the model takes a batch of inputs and labels and computes the output of the model. It then computes the accuracy, recall, precision, and F1-score of the model output by comparing it to the labels and logs these metrics.

5. Evaluation results

The results from each model are as follows:

Model	**Accuracy**	**Recall**
First Model– ReLU	86.03%	90.05%
Second Model – LeakyReLU	86.09%	90.61%
Lightning Model – LeakyReLU	93.71%	94.15%

Given the three models tested, it's clear that the Lightning Model using LeakyReLU as an activation function outperforms the other two. This model achieves a notably higher recall rate of 94.15%, compared to 90.61% from the second model and 90.05% from the first. Recall is a particularly important metric in this context as it represents the ability of the model to correctly identify customers who are at risk of churning. Thus, the better the model's recall, the fewer customers the business will likely lose.

High recall rates enable more accurate identification of at-risk customers, allowing the business to implement effective customer retention strategies. These might range from personalized marketing campaigns to incentives such as discounts or loyalty programs, tailored to the specific needs and preferences of each identified customer segment.

The model can provide insights into the factors contributing to customer churn, informing not only the approach to retaining current customers but also guiding improvements to services and operations. Ultimately, the utilization of this predictive model, particularly the superior-performing Lightning Model, can be instrumental in enhancing customer satisfaction, reducing churn, and improving the overall profitability of the business.