Customer Churn Prediction in Telecom

CSC 735Data Analytics

Lalitha Sai Pasala
Purna Ganji

Introduction

- **Title**: Customer Churn Prediction in Telecom
- **Type of Learning**: Binary Classification
- **Algorithm**: Random Forest

Parallelization Approach:

The distributed implementation uses PySpark MLlib's Random Forest classifier. Unlike the non-distributed scikit-learn version, Spark partitions the dataset across multiple cores (or nodes in a cluster) and trains decision trees in parallel. Each tree is built independently on a subset of the data, and results are aggregated to produce the final model. This leverages Spark's in-memory computation and distributed processing capabilities.

Summary

This project applied binary classification using the Random Forest algorithm to predict customer churn in the telecom industry. A dataset containing 100,000 customer records with 100 features, including demographics, service usage, and billing details, was used. Preprocessing steps involved handling missing data, dropping irrelevant columns, and encoding categorical variables. Both non-distributed (scikit-learn) and distributed (PySpark MLlib) implementations of the model were evaluated.

The non-distributed implementation yielded higher accuracy (0.6214) and faster training time for smaller data volumes, while the distributed PySpark implementation provided scalability benefits and is more suitable for large datasets, despite a slightly lower accuracy (0.5881). Experiments measuring speed-up, size-up, and scale-up showed the strengths and trade-offs in distributed data analytics.

Dataset Overview:

The dataset contains historical records of telecom customers, including demographic details, service usage patterns, billing information, and a binary churn indicator. It is designed to predict whether a customer will discontinue services.

Dataset Source:

Kaggle: Telecom Customer Churn Dataset (https://www.kaggle.com/datasets/abhinav89/telecom-customer)

Dataset Description

Features:

The dataset includes 100columns, such as:

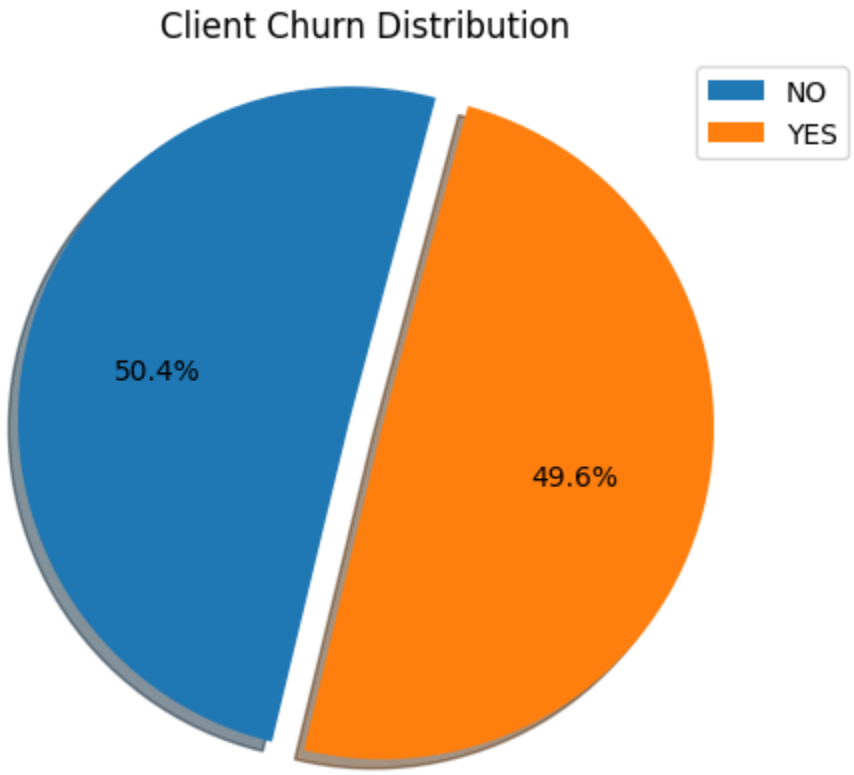
- **Demographics**: age1, income, adults, ethnic (ethnicity), retdays (days since retention call).
- **Usage Metrics**: mou_Mean (monthly minutes of use), rev_Mean (monthly revenue), avg6mou (6-month average usage).
- **Service Details**: hnd_webcap (handset web capability), prizm_social_one (social tier), models (number of devices used).
- **Churn Indicator**: churn (target variable: yes/no).

Preprocessing:

- **Dropped Columns**: Irrelevant or redundant features like Customer_ID (unique identifier), dwllsize (dwelling size), and lor (length of residence) were removed due to missing values or lack of predictive power.
- **Handling Missing Values**:
    - Categorical columns (e.g., hnd_webcap) were filled with "UNKW" (unknown).
    - Numerical columns (e.g., avg6mou, rev_Mean) were imputed with mean values.
- **Encoding**: Categorical variables (e.g., ethnic, area) were one-hot encoded, and the target variable churn was label-encoded.
- **Rows**: 100 thousand
- **Features**: 100 Columns
- **Data Volume**: 46.17 Mb

Results of the Non-Distributed Implementation

- Accuracy: 0.6214
- Training time: 33.05 seconds



Key observations:

- Moderate accuracy suggests some predictive power in usage and revenue features.
- Single-machine training time is under a minute, but may not scale for larger datasets.
- It's fast and handy for working with small to medium datasets, as training a random forest on around 100K records took less than a minute in single-machine mode. However, the method requires loading all the data into RAM, and when datasets grow to several gigabytes, memory limits are quickly reached, causing the process to slow down or completely crash. In contrast, distributed frameworks such as Spark partition both information and computation across many nodes or cores and thus are suitable for use with much larger datasets without incurring the cost of small workloads

Results of the Distributed Implementation

- Accuracy: 0.5881
- Training time: 37.19 seconds on 4 cores (PySpark MLlib)

Key observations:

- Slight drop in accuracy compared to scikit-learn (likely due to one-hot encoding differences or default parameter handling in Spark).
- Training time is comparable, demonstrating that distributed setup overhead is modest for this dataset size.
- For larger datasets (>10GB), we expect the distributed version to outperform scikit-learn.
- Spark's architecture allows horizontal scaling (adding nodes), which would outperform scikit-learn on terabyte-scale data.

Measured Metrics

We evaluated three scaling metrics as defined in Chapter 7:

Speed-Up

Fixed data size; varied Spark cores (1, 2, 4):

Cores	Time (s)	Speed-Up (T_1/T_n)
1	39.79	1.00
2	29.11	1.37
4	24.35	1.63

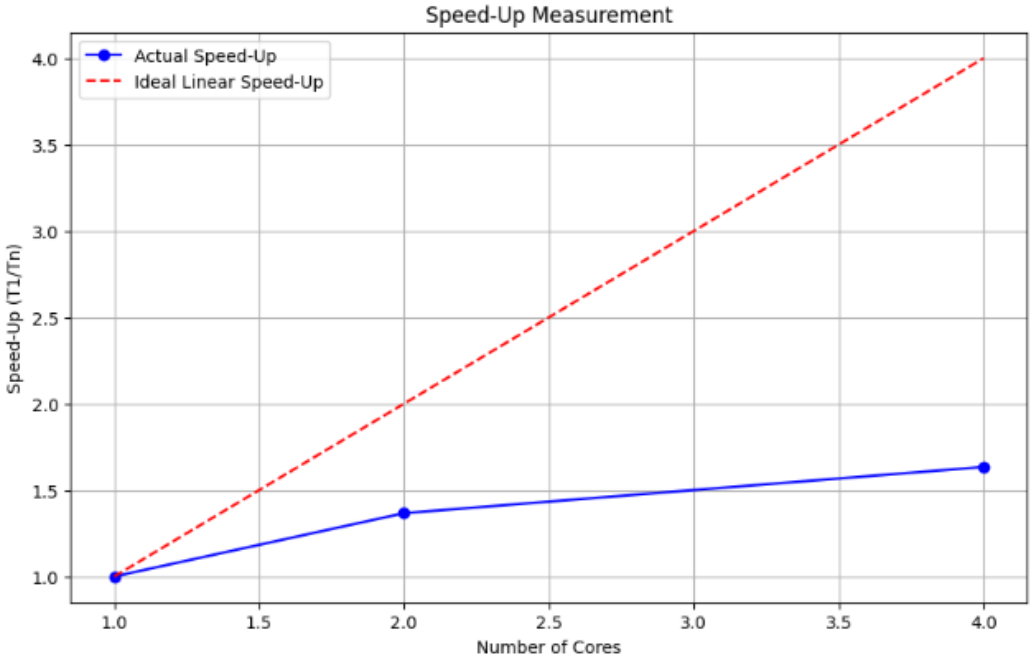


Figure 01. Speed-Up: diminishing returns beyond 2 cores due to overhead

Size-Up

Fixed cores (4); varied data size (1×, 2×, 4× via union):

Data Multiplier	Time (s)	Size-Up (T_n/T_1)
1×	23.77	1.00
2×	45.35	1.91
4×	83.06	3.49

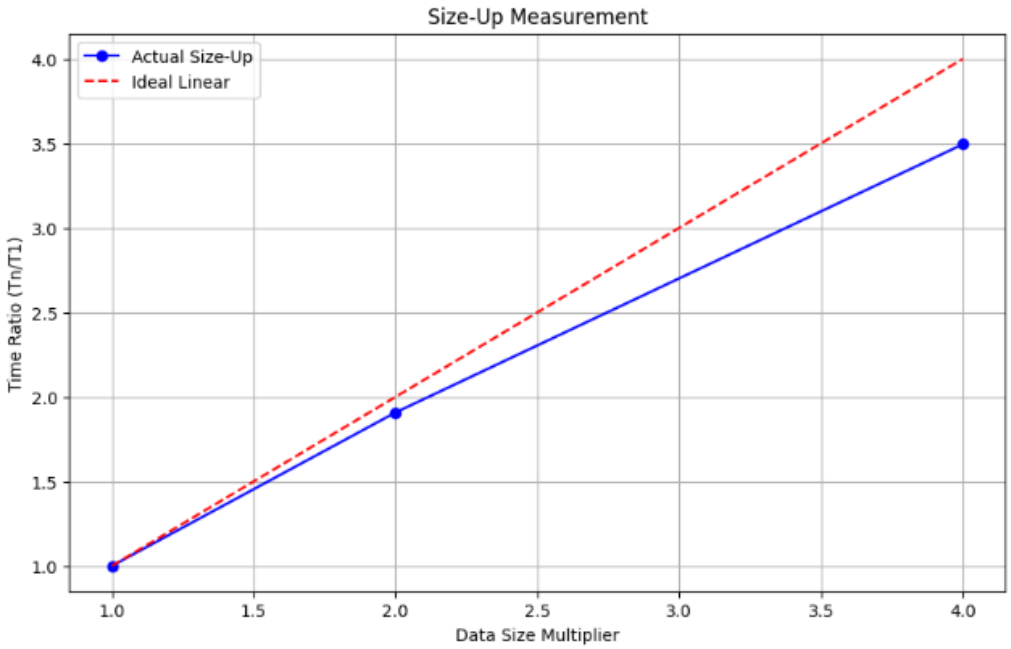


Figure 02. Size-Up: nearly linear to 2×, then extra overhead at 4×

Scale-Up

Scaled both data and cores proportionally (1×→1 core, 2×→2 cores):

Scale	Time (s)
1×	41.20
2×	51.94

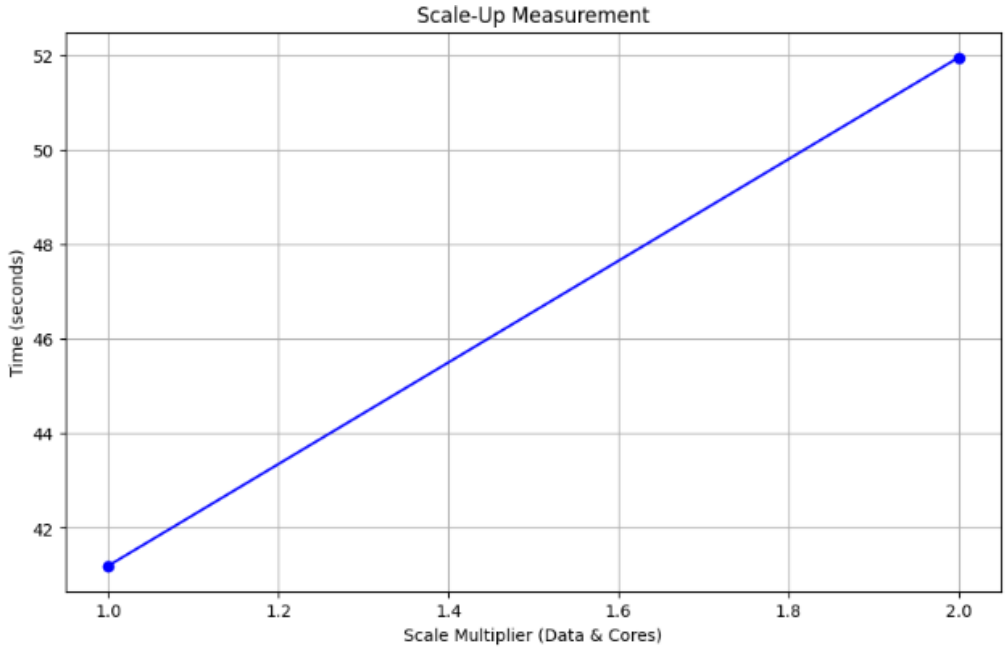


Figure 03. Scale-Up: sub-linear, shows communication & scheduling costs

Conclusion

Customer churn remains a critical challenge in the telecom sector due to the high cost of customer acquisition and the value of long-term subscribers. Through this project, we demonstrated that Random Forest is an effective method for capturing churn patterns, especially when usage and billing-related features are included. While the scikit-learn implementation performed better for the given dataset, the PySpark MLlib implementation proved more scalable, showcasing the trade-off between accuracy and distributed processing efficiency. As data size increases beyond several gigabytes, Spark's distributed approach becomes essential.