Telecom Churn Prediction

Pavan Atukuri[1]
[1]University of Maryland, College Park, MD
Email: patukuri@umd.edu

***Abstract*—Customer churn poses a significant business issue for telecom operators, reducing revenue and increasing costs. This study examines machine learning models to identify customers most likely to churn. Using a telecom customer dataset with demographic, usage, and subscription features, five classification models were tested: logistic regression, decision trees, K-nearest neighbors (KNN), and naive Bayes and, and their effectiveness is assessed using fundamental metrics, including confusion matrices, classification reports, accuracy, precision, and recall. The decision tree model emerged as the most accurate with an accuracy of *93%* and provided the best balance in precision and recall. The methodology and results demonstrate that machine learning provides an effective solution for better understanding and acting on churn. By anticipating likely churners, telecoms can undertake proactive win-back programs and account management strategies aimed at their most valuable subscribers.**

***Index Terms*—Telecom Data set, Customer Churn, Logistic Regression, Linear Discriminant Analysis (LDA) Classifier, Decision Tree, K-nearest neighbors, Naive Bayes Classifier, Classification report, Confusion matrix, Accuracy and Recall.**

I. Introduction

Customer churn poses a critical problem in the telecom industry, with substantial impacts on revenue. This study aims to effectively predict customer churn using machine learning techniques. The developed churn prediction model will be utilized by telecom companies to identify high-risk customers and target proactive retention programs and incentives towards them The data used for this study contains 3333 records with 20 different features for each record.The features are as follows **State,Account length, Area code,International plan,Voice mail plan, Number vmail messages, Total day minutes,Total day calls, Total day charge, Total eve minutes, Total eve calls, Total eve charge, Total night minutes, Total night calls, Total night charge, Total intl minutes, Total intl calls, Total intl charge, Customer service calls** and **Churn** is the target class that is a dependent variable. The goal is to predict whether the customer with the given features is likely to churn or not.

Section II describes how the Decision Tree, Logistic Regression, KNN, Naive Bayes Classifier, Linear Discriminant Analysis (LDA) were constructed and how their classification performance metrics were calculated. The Results section compare the prediction outcomes of the different models on the telecom churn dataset using quantitative metrics and confusion matrices. The Conclusion summarizes key takeaways, explains why some models outperformed others, and provides guidance on selecting the right algorithm based on accuracy versus interpretability trade-offs and requirements.

II. Methodology

This section elucidates the process of implementing various classification models on the Telecom Churn dataset. It begins with the identification and importation of necessary libraries for the task. Following this, the dataset is loaded and preprocessed. The construction five classification models Linear Discriminant Analysis (LDA) [10], Naive Bayes [9], and compares the results with **Decision Tree Classifier** [6] and **K-NN Classifier** [5] and Logistic Regression [2] on the Telecom Churn dataset. Lastly, the functions employed to compute the metrics are discussed.

A. Libraries and Tools needed

For this analysis, Python version 3.7.6 is used and Google [**?**] Colab, jupyter notebbok [8] is used and the libraries used are pandas with version 2.0, sklearn with version 1.3, numpy with version 1.24, matplotlib with version 3.7.

B. Data Collection

For this analysis, the dataset is loaded into a Pandas DataFrame using the *read_csv()* function, which accepts the file path and returns a DataFrame object using the Pandas [7] *read_csv()* function. This function accepts the *file path* as an input and returns a DataFrame object. This DataFrame-based loading prepares the dataset for efficient access during partitioning, training, and analysis, as described in more detail for the Decision Tree model in a previous study of creating **Decision Tree Classifier** [6] and **K-NN Classifier** [5].

C. Data Preprocessing

Prior to the construction of the models, the dataset is scrutinized for the presence of any null values using the isnull() [18] function. The outcome reveals that the Telecom Churn dataset is devoid of any null values. The dataset also contains several columns which are correlated (Fig 1) and also some columns which are unnecessary for the model building, these **'State', 'Area code', 'Total day charge', 'Total eve charge','Total night charge', 'Total intl charge'** columns were dropped from the datset. Subsequent to this null value check, all binary columns are transformed into numerical ones utilizing the LabelEncoder() [19] function. This label encoding

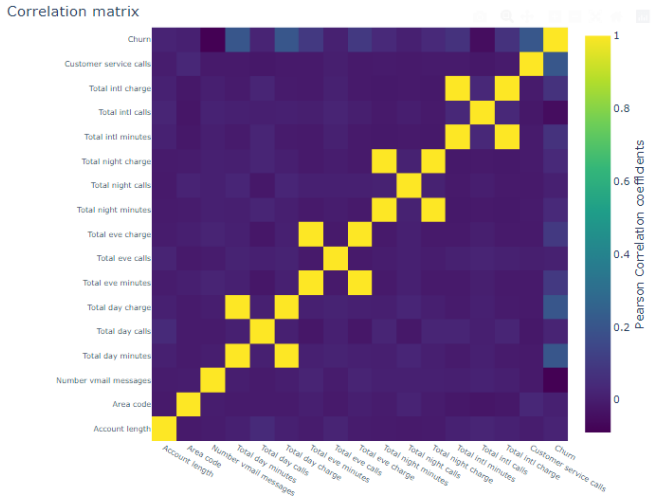

Fig. 1. Correlation Matrix

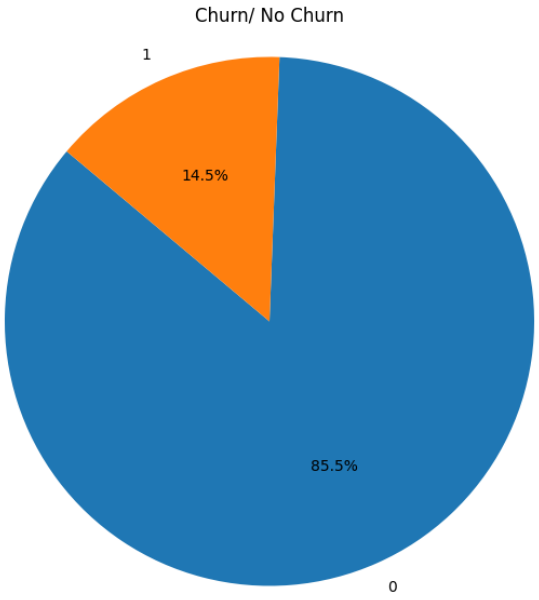

Fig. 2. initial Class Distribution

process is essential for converting categorical data into a numerical format by assigning unique integers to each category, thereby facilitating the interpretation and processing of data by machine learning models. Lastly, standardization is performed for all the numerical columns using the StandardScaler() [19] function, which modifies the data to possess a mean of 0 and a standard deviation of 1. This transformation ensures that the features are comparable and compatible with certain machine learning algorithms.

D. Data Partitioning

The Iris data is split into *70%* training and *30%*test sets using scikit-learn's *train_test_split()* [11] function. This partitioning allows model training on the training data and performance evaluation on the unseen test data. It prevents overfitting and gives an unbiased estimate of real-world performance.

The function takes the data, labels, and test size input. It splits the data into *X_train*, *X_test*, *y_train* and *y_test* arrays for the feature data and labels of the training and test sets.

Listing 1. Train-Test Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)
```

In the above code, 'X' denotes the independent variables, while 'y' denotes the dependent variable. We specify a test size of 0.3, resulting in a *70%* training dataset and a *30%* test dataset. Additionally, the random state parameter is used to guarantee reproducible data splitting. After splitting the data, oversampling is being done to the dataset as the intital classes were imbalanced in the dataset, Oversampling helps to avoid overfitting situtations. This oversampling is done using Randomsampling as shown below.

Listing 2. OverSampling

```
from imblearn.over_sampling import RandomOverSampler
# define oversampling strategy
random_oversample = RandomOverSampler(
    sampling_strategy='minority')
# fit and apply the transform
X_rand_samp, y_rand_samp = random_oversample.
    fit_resample(X_train, y_train)
```

An instance of RandomOverSampler is created with the sampling_strategy parameter set to '**minority**'. This means that the oversampling will be applied to the minority class in the dataset.

The **fit_resample** method of the **RandomOverSampler** instance is called on the training data (X_train and y_train). This method fits the oversampler on the data and returns the oversampled dataset. The oversampled data is stored in X_rand_samp and y_rand_samp.

The Decision Tree, Logistic Regression, and K-NN models are then trained on the preprocessed training data. This process follows similar steps outlined in **Decision Tree Classifier** [6], **Understanding Logistic Regression** [4], **K-Nearest Neighbor Algorithm** [5] respectively.

E. Building models

Scikit-Learn is a versatile library that offers a variety of classes and functions for constructing Machine Learning models. In this study, we construct and evaluate five different models.

The Logistic Regression model is built using the LogisticRegression() function from the Scikit-Learn library. This linear classification algorithm is suitable for binary and multiclass classification tasks. The construction process for this model follows the steps outlined in the paper "Logistic Regression ON IRIS dataset" [4].

The Decision Tree model is constructed using the DecisionTreeClassifier() function from the Scikit-Learn library. This non-linear algorithm recursively splits the data

based on features to make decisions. The construction of this model adheres to the steps described in the paper ”Decision Tree Classifier” [6] .

Listing 3. Decision Tree

```
dt_model = DecisionTreeClassifier(criterion = "entropy",random_state = 100,max_depth=5, min_samples_leaf=50, min_samples_split=50)
```

- criterion = “entropy”: This parameter specifies the function to measure the quality of a split. The options are ‘gini’ for the Gini impurity and ‘entropy’ for the information gain. Here, ‘entropy’ is used.
- random_state = 100: This parameter controls the randomness of the estimator. The features are always randomly permuted at each split, hence, even if max_features=n_features, the results won’t be the same across different calls. However, using an integer value for random_state allows for reproducibility of the results across multiple function calls.
- max_depth=5: This parameter controls the maximum depth of the tree. It helps to prevent overfitting by limiting how deep the tree can go. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- min_samples_leaf=50: This parameter specifies the minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches.
- min_samples_split=50: This parameter specifies the minimum number of samples required to split an internal node. If an integer is provided, then consider min_samples_split as the minimum number. If a float is provided, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.

The K-Nearest Neighbour model is built using the KNeighborsClassifier() function from the Scikit-Learn library. This non-parametric algorithm classifies data points based on the majority class among their k-nearest neighbors. The construction of this model follows the steps detailed in the paper ”K- Nearest Neighbour Algorithm [5]” .

Listing 4. KNN

```
knn = KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
                           weights='uniform')
```

- algorithm=’auto’: This parameter specifies the algorithm used to compute the nearest neighbors. The options are ’ball_tree’, ’kd_tree’, ’brute’, or ’auto’. If ’auto’, the algorithm attempts to determine the best approach based on the values passed to fit method.
- leaf_size=30: This parameter affects the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.
- metric=’minkowski’: This is the distance metric to use for the tree. The default metric is ’minkowski’, and with p=2 is equivalent to the standard Euclidean metric.
- metric_params=None: Additional keyword arguments for the metric function.
- n_jobs=1: The number of parallel jobs to run for neighbors search. If -1, then the number of jobs is set to the number of CPU cores.
- n_neighbors=5: Number of neighbors to use by default for kneighbors queries.
- p=2: Parameter for the Minkowski metric. When p=1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p=2. For arbitrary p, minkowski_distance (l_p) is used.
- weights=’uniform’: Weight function used in prediction. Possible values are ’uniform’, ’distance’, or a user-defined function. With ’uniform’, all points in each neighborhood are weighted equally. With ’distance’, points are weighted by the inverse of their distance.

The Naive Bayes model is constructed using the `GaussianNB()` function from the `sklearn.naive_bayes` module. This variant of the Naive Bayes algorithm assumes that the likelihood of the features follows a Gaussian (normal) distribution. After importing the classifier, an instance of the classifier is created and fitted to the training data. Predictions are then made on the test data. The detailed process involved in creating the Naive bayes Model is presented in the paper [20]

Lastly, the Linear Discriminant Analysis (LDA) is a classifier from the `sklearn.discriminant_analysis` module. LDA is a dimensionality reduction technique commonly used in pattern recognition and machine learning. It is often used as a classification algorithm, particularly in scenarios where the classes are well-separated and follow a normal distribution.The detailed process involved in creating the Naive bayes Model is presented in the paper [20]

F. Calculating the metrics

The *sklearn.metrics* contains a metrics module with various functions for evaluating machine learning model performance. Several of these are leveraged to assess and compare the models in this analysis. The ***confusion_matrix()***, ***accuracy_score()***, ***classification_report()***,***recall_score()***, ***precision_score()***, ***ConfusionMatrixDisplay()***, are functions used and imported to calculate the accuracy, recall, and to generate the confusion matrix and classification report. All these functions and their behaviours were explained in the previous paper Metrics for Classification Models [17]
ROC curves are also generated and the area under the ROC curve is described. The steps involved in building ROC curves are as mentioned in the previous paper [17]

The ***classification_report()***, is generated using *metrics.classification_report()* function from scikit-learn library which will take two mandatory parameters as shown below.

Listing 5. classification_report() functions

```
from sklearn.metrics import classification_report
classification_rep = classification_report(y_test, dt_predictions)
```

In the above code, y_test contains the actual flower types for the test data, which are used to test the classifier's ability to predict. The dt_predictions variable contains the predicted flower types made by the classifier (Decision tree in this case) for the test dataset (X_test).
In the following Results section, different models were compared using the standard metrics like Accuracy, Precision, Recall, Confusion matrix, F1 Score. As all these are standard in every Machine Learning model, these metrics were employed to compare the results.

RESULTS

In this section, the metrics obtained for all the five Machine Learning algorithms LDA, and Naive Bayes, -Decision Tree, Logistic Regression, and K-Nearest Neighbors were compared and compared when applied to the Telecom churn data set.

A. Accuracy

Accuracy is the ratio of correctly predicted observations to the total observations. Higher accuracy means that the model predictions are closer to the actual outcomes. The Decision Tree model has the highest accuracy of 93%, meaning it correctly predicted the class of 93% of the observations in the test set. The Naive Bayes model follows closely with an accuracy of 85%, while the K-Nearest Neighbours (K-NN) model has an accuracy of 84%. Both the Logistic Regression (LR) and Linear Discriminant Analysis (LDA) models have an accuracy of 79%.

B. Precision

Precision is the ratio of correctly predicted positive observations to the total predicted positives. High precision relates to a low false positive rate. The Decision Tree model leads with a precision of 96% for class 0 and 74% for class 1. The Naive Bayes model has a precision of 96% for class 0 and 48% for class 1, while the K-NN model has a precision of 95% for class 0 and 47% for class 1. Both the LR and LDA models have a precision of 95% for class 0 and 38% for class 1.

C. Recall

Recall (Sensitivity) is the ratio of correctly predicted positive observations to the all observations in actual class. The recall scores show that the Naive Bayes model has the highest recall for class 1 at 78%, while the Decision Tree model has the highest recall for class 0 at 96%. The K-NN model has a recall of 86% for class 0 and 73% for class 1. Both the LR and LDA models have a recall of 80% for class 0 and 76% for class 1.

D. F1 Score

The F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. The F1 scores reveal that the Decision Tree model has the highest F1 score of 96% for class 0 and 74% for class 1. The Naive Bayes model has an F1 score of 91% for class 0 and 59% for class 1, while the K-NN model has an F1 score of 90% for class 0 and 57% for class 1. Both the LR and LDA models have an F1 score of 87% for class 0 and around 50% for class 1.

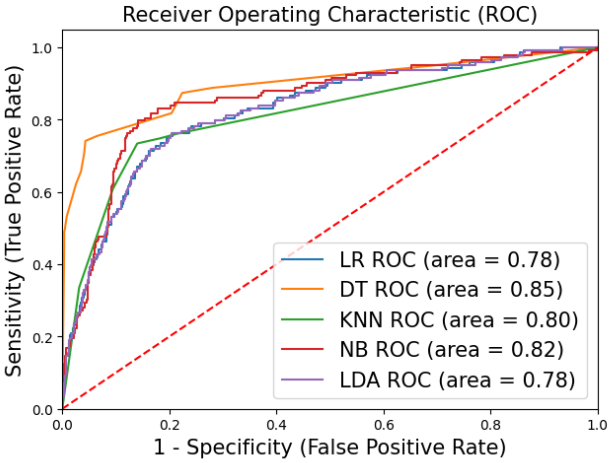

Fig. 3. ROC/AUC curves for different models

Model	Accuracy
Logistic Regression	79%
Decision Tree	93%
K-Nearest Neighbours	84%
Naive Bayes	85%
Linear Discriminant Analysis	79%

E. Receiver Operating Characteristic Curve

The *Area Under the Curve (AUC)* of the *Receiver Operating Characteristic (ROC)* curve is a single scalar value that summarizes the overall performance of the model across all possible classification thresholds. An AUC of 1.0 indicates a perfect classifier, while an AUC of 0.5 represents a model that is no better than random guessing.

The *Logistic Regression* model has an AUC of 0.77. This suggests that there is a 77% chance that the model will be able to distinguish customer churn. This is considered a good level of prediction accuracy, indicating that the model has a reasonable discriminative power.

The *Decision Tree* model has an AUC of 0.85, which is considered very good. This shows that the model has a high measure of separability and is capable of distinguishing between whether a customer churns or not with an 85% probability. This is a strong performance, suggesting that the decision tree model is a robust classifier for this dataset. The *K-Nearest Neighbours (K-NN)* model has an AUC of 0.80, which is also considered good. This indicates that there is an 80% chance that the model will be able to distinguish customer churn. This suggests that the K-NN model, which classifies a data point based on the majority class of its 'k' nearest neighbors, performs well on this dataset.

The *Naive Bayes* model has an AUC of 0.82, indicating a very good level of prediction accuracy. This suggests that

there is an 82% chance that the model, which applies Bayes' theorem with the assumption of conditional independence between every pair of features, will be able to distinguish customer churn.

Lastly, the *Linear Discriminant Analysis (LDA)* model has an AUC of 0.78, indicating a good level of prediction accuracy. LDA is a dimensionality reduction technique often used as a classification algorithm. An AUC of 0.78 suggests that there is a 78% chance that the LDA model will be able to distinguish whether a customer churn happens or not.

DISCUSSION

In conclusion, the results of this study provide valuable insights into the performance of various machine learning models on the given dataset. The Decision Tree model outperformed the other models in terms of accuracy (93%), precision (96% for class 0 and 74% for class 1), recall (96% for class 0 and 74% for class 1), and F1 score (96% for class 0 and 74.13% for class 1), demonstrating its robustness as a classifier for this dataset. However, all models showed a good to very good level of prediction accuracy, with the Naive Bayes model showing promising results, especially in terms of recall for class 1 (78%).

The results were as expected, given the nature of the models and the characteristics of the dataset. The high performance of the Decision Tree model can be attributed to its ability to handle both numerical and categorical data, as well as its capacity for feature interactions. The Naive Bayes model also performed well, particularly in terms of recall for class 1, due to its underlying probabilistic model which is well-suited for binary classification tasks.

This work is important as it provides a comparative analysis of different machine learning models, offering insights into their performance and characteristics. It serves as a guide for researchers and practitioners in selecting appropriate models for similar tasks. Furthermore, it contributes to the broader field of machine learning by demonstrating the application of these models on a real-world dataset.

For future work, hyperparameter tuning could be explored to further improve the performance of the models. Techniques such as Grid Search or Random Search [21] could be used to find the optimal parameters for each model. Additionally, ensemble methods, which combine the predictions of multiple models, could be employed to potentially achieve higher accuracy. The work could also be extended to cover more data or different types of datasets, providing a more comprehensive understanding of the models' performance across various domains. Finally, the interpretability of the models, which is crucial for understanding the decision-making process of the models, could be investigated.

REFERENCES

[1] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
[2] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[3] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
[4] ENPM808L Logistic Regression on IRIS Dataset
[5] ENPM808L KNN Classifier on IRIS Dataset
[6] ENPM808L sklearn Classification
[7] https://pypi.org/project/pandas/
[8] https://jupyter.org/
[9] https://scikit-learn.org/stable/modules/naive_bayes.htmlgaussian-naive-bayes
[10] https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDi
[11] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
[12] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion _matrix.html
[13] https://scikitlearn.org/stable/modules/generated/sklearn.metrics.classification_report.html
[14] https://scikitlearn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
[15] https://scikitlearn.org/stable/modules/generated/sklearn.metrics.recall_score.html
[16] https://scikitlearn.org/stable/modules/generated/sklearn.metrics.precision_score.html
[17] ENPM808L Metrics for Classification Models
[18] https://scikit-learn.org/stable/modules/impute.html
[19] https://datascience.stackexchange.com/questions/44225/how-to-handle-preprocessing-standardscaler-labelencoder-when-using-data-gener
[20] ENPM808L Advanced Classification Models
[21] https://www.kdnuggets.com/2022/10/hyperparameter-tuning-grid-search-random-search-python.html