BIA 678-A BIG DATA TECHNOLOGIES

TEAM PROJECT REPORT

ON

CUSTOMER CHURN PREDICTION USING PYSPARK

By,

GROUP-6

Authors and their roles:

Gahana Nagaraja (20025607) – Gathering & processing data, designing presentation

Namratha Nagathihalli Anantha (20025756) – Developing & implementing Model,

Niharika Mysore Prakasha (20022638) – Evaluating model & report making

Under the guidance of

Denghui Zhang

Date: 05/06/2024

STEVENS INSTITUTE OF TECHNOLOGY

TABLE OF CONTENTS

1. Introduction 3

2. About PySpark 4

3. Data Description 5

4. Summary Statistics 19

5. Correlation Analysis 19

6. Modeling Process 20

6.1. Model Training 20

6.2. Model Selection 20

6.3. K-fold Cross Validation 20

6.3.1. Naïve Bayes

6.3.2. Random Forest

6.3.3. Gradient Boosted Tree

6.4. Model Evaluation 20

7. Conclusion 20

8. Future Scope 20

9. References 20

10. Appendix 20

1. INTRODUCTION

Customer churn serves as a critical metric for growing businesses, offering valuable insights into customer retention levels, though it is not the happiest measure. Churn prediction, as its name implies, utilizes data-driven techniques to identify customer accounts likely to reduce engagement, terminate subscriptions, or disengage from a company. Customer churn refers to the proportion of customers discontinuing the use of a company's product or service within a specified timeframe. This figure is derived by dividing the number of lost customers during that period, such as a quarter, by the initial number of customers. For instance, if a quarter begins with 400 customers and ends with 380, the churn rate is 5%, reflecting the loss of 5% of the customer base. Ideally, companies should strive for a churn rate close to 0%, necessitating vigilant monitoring and prioritization. Churn prediction entails a series of steps. Initial stages involve identifying risk factors and addressing blind spots. Subsequently, collaboration with other departments such as customer support, marketing and post-sales teams becomes crucial to mitigate detected issues effectively. Additionally, saved accounts also require diligent tracking and monitoring. It harnesses big data and machine learning to identify at-risk customers by analyzing behavior data and extracting meaningful features. Using PySpark's MLlib, predictive models classify customers as churners or non-churners, evaluated for accuracy and AUC. Deployed into production, these models enable real-time churn predictions, fostering proactive retention strategies for sustained business growth and profitability.

2. ABOUT PYSPARK

PySpark, an open-source API for Python and Apache Spark, empowers data scientists to conduct high-performance big data analytics and rapid data processing across datasets of any size. This collaborative framework seamlessly combines Python with Apache Spark's robust capabilities, including MLlib, DataFrames, and SparkSQL, enabling efficient processing and analysis, even for immense datasets. PySpark facilitates seamless transitions between Apache Spark and Pandas, supports stream processing, and interfaces with JVM objects. Its compatibility with external libraries such as GraphFrames and PySparkSQL further enhances its utility for tasks like graph analysis and handling massive data volumes. Specifically in customer churn prediction, PySpark streamlines data preprocessing, feature engineering, model training, and evaluation, facilitating swift analysis and prediction. Leveraging machine learning libraries like MLlib, PySpark empowers the creation of advanced predictive models to identify potential churners effectively. Overall, PySpark significantly boosts the speed, scalability, and efficacy of customer churn prediction efforts.

3. DATA DESCRIPTION

Data for customer churn prediction has been collected from Telecom churn datasets, Kaggle where, each row represents a customer and each column contains customer’s attributes. Here’s a description of each variable in the dataset:

- **State:** This represents the state in which the customer resides. It is a string variable.
- **Account length:** This is the length of time the customer has been with the telecom company, measured in months.
- **Area code:** This is the area code of the customer's phone number.
- **International plan:** This indicates whether the customer has an international calling plan or not.

- **Voice mail plan:** This indicates whether the customer has a voice mail plan or not.
- **Number vmail messages:** If the customer has a voice mail plan, this variable represents the number of voice mail messages the customer has received.
- **Total day minutes:** This represents the total number of minutes the customer has used during the day.
- **Total day calls:** This represents the total number of calls the customer has made during the day.
- **Total day charge:** This represents the total charge (in currency) for the calls made during the day.
- **Total eve minutes:** Similar to total day minutes, this represents the total number of minutes the customer has used during the evening.
- **Total eve calls:** Similar to total day calls, this represents the total number of calls the customer has made during the evening.
- **Total eve charge:** Similar to total day charge, this represents the total charge (in currency) for the calls made during the evening.
- **Total night minutes:** Represents the total number of minutes the customer has used during the night.
- **Total night calls:** Represents the total number of calls the customer has made during the night.
- **Total night charge:** Represents the total charge (in currency) for the calls made during the night.
- **Total intl minutes:** Represents the total number of international minutes used by the customer.
- **Total intl calls:** Represents the total number of international calls made by the customer.

- **Total intl charge:** Represents the total charge (in currency) for the international calls made by the customer.
- **Customer service calls:** Represents the number of customer service calls made by the customer.
- **Churn:** This is the target variable that indicates whether the customer has churned (i.e., stopped using the service) or not.

DATA PREPROCESSING:

The get_data function used in the preprocessing task, streamlines data manipulation tasks, primarily focusing on transforming categorical data into numeric format, a prerequisite for many machine learning algorithms. This custom function likely retrieves the dataset, applies a user-defined function to convert categorical values like "Yes" or "No" into numeric equivalents (often 1 and 0 respectively), and may perform additional preprocessing steps such as handling missing values or scaling features. By encapsulating these tasks, get_data simplifies the data preprocessing pipeline, enhancing efficiency and reproducibility in data analysis and machine learning endeavors.

	0	1	2	3	4
State	KS	OH	NJ	OH	OK
Account length	128	107	137	84	75
Area code	415	415	415	408	415
International plan	0.0	0.0	0.0	1.0	1.0
Voice mail plan	1.0	1.0	0.0	0.0	0.0
Number vmail messages	25	26	0	0	0
Total day minutes	265.1	161.6	243.4	299.4	166.7
Total day calls	110	123	114	71	113
Total day charge	45.07	27.47	41.38	50.9	28.34
Total eve minutes	197.4	195.5	121.2	61.9	148.3
Total eve calls	99	103	110	88	122
Total eve charge	16.78	16.62	10.3	5.26	12.61
Total night minutes	244.7	254.4	162.6	196.9	186.9
Total night calls	91	103	104	89	121
Total night charge	11.01	11.45	7.32	8.86	8.41
Total intl minutes	10.0	13.7	12.2	6.6	10.1
Total intl calls	3	3	5	7	3
Total intl charge	2.7	3.7	3.29	1.78	2.73
Customer service calls	1	1	0	2	3
Churn	0.0	0.0	0.0	0.0	0.0

Fig 1. 'International plan', 'Voice mail plan' and 'Churn' transformed from categorical data into numerical form

4. SUMMARY STATISTICS

Summary statistics are numerical measures that provide an overview or summary of the characteristics of a dataset. These statistics offer insights into the central tendency, dispersion, and shape of the data distribution. In customer churn prediction using PySpark, summary statistics are computed for numeric features to provide a concise overview of the dataset's characteristics. PySpark's built-in functions, such as describe(), facilitate this process by calculating key statistics like mean, standard deviation, and quartiles for relevant numeric variables. These summary statistics offer valuable insights into the distribution and variability of the data, aiding analysts in understanding the patterns and trends that may influence customer churn.

	0	1	2	3	4
summary	count	mean	stddev	min	max
Account length	2666	100.62040510127532	39.56397365334985	1	243
Area code	2666	437.43885971492875	42.521018019427174	408	510
Number vmail messages	2666	8.021755438859715	13.61227701829193	0	50
Total day minutes	2666	179.48162040510135	54.21035022086982	0.0	350.8
Total day calls	2666	100.31020255063765	19.988162186059512	0	160
Total day charge	2666	30.512404351087813	9.215732907163497	0.0	59.64
Total eve minutes	2666	200.38615903976006	50.95151511764598	0.0	363.7
Total eve calls	2666	100.02363090772693	20.16144511531889	0	170
Total eve charge	2666	17.033072018004518	4.330864176799864	0.0	30.91
Total night minutes	2666	201.16894223555968	50.780323368725206	43.7	395.0
Total night calls	2666	100.10615153788447	19.418458551101697	33	166
Total night charge	2666	9.052689422355604	2.2851195129157564	1.97	17.77
Total intl minutes	2666	10.23702175543886	2.7883485770512566	0.0	20.0
Total intl calls	2666	4.467366841710428	2.4561949030129466	0	20
Total intl charge	2666	2.764489872468112	0.7528120531228477	0.0	5.4
Customer service calls	2666	1.5626406601650413	1.3112357589949093	0	9

Fig 2. Summary statistics of the numeric features in the dataset

5. CORRELATION ANALYSIS

In customer churn prediction, analyzing correlations between numeric columns is crucial for understanding relationships among different variables and their potential impact on churn. Utilizing Python packages like seaborn allows for the generation of scatter plots to visualize these correlations. Due to computational constraints, particularly with large datasets, a common strategy is to randomly sample a portion of the data, typically around 10%, for initial exploration. This sampling approach provides a rough overview of the data's correlation structure, enabling analysts to identify potential patterns and relationships without incurring excessive computational costs. By leveraging seaborn and other Python packages for statistical analysis and visualization, analysts can gain valuable insights into the underlying factors driving customer churn, facilitating informed decision-making and targeted intervention strategies.

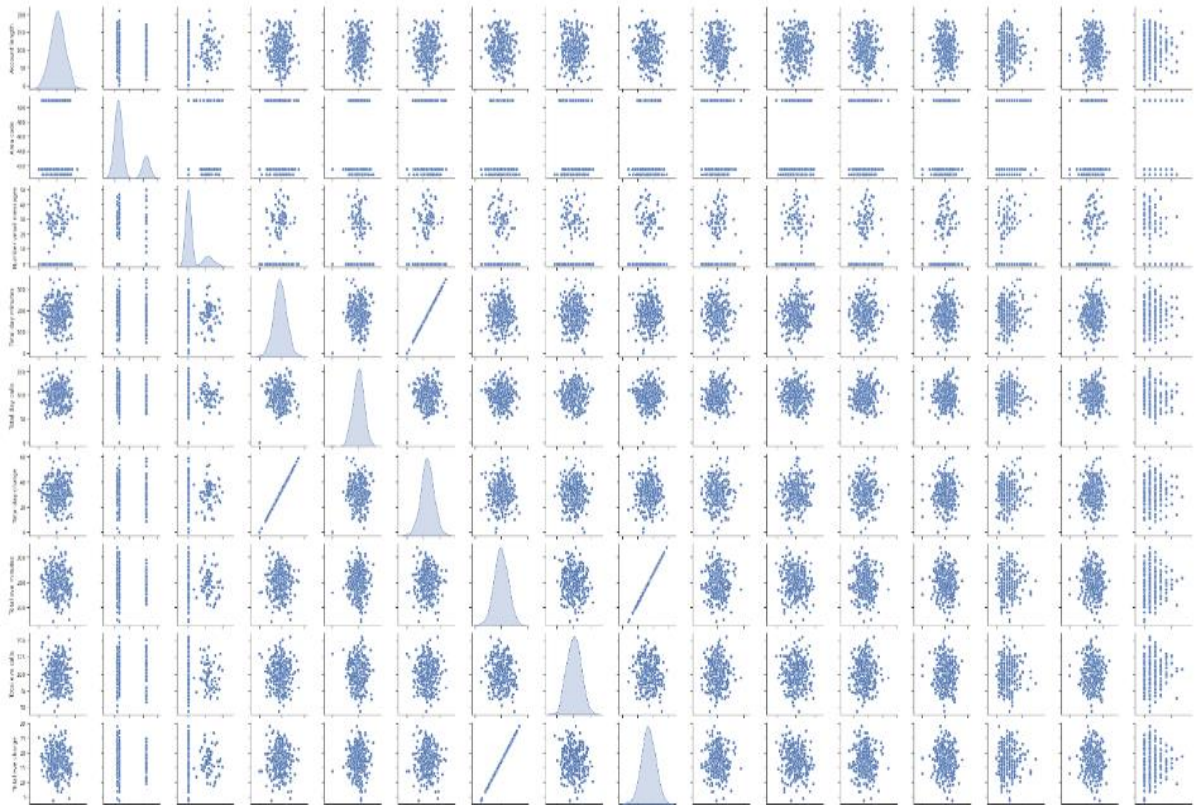

Fig 3. Scatter plots examining the correlation between the numerical columns

6. MODELING PROCESS

The modeling process involves several key steps to ensure the development of an accurate predictive model. Initially, data preprocessing is conducted to clean, transform, and prepare the dataset for analysis. Subsequently, the dataset is split into training and testing sets, with the training set used to train various models. In the realm of model processing, Spark leverages two key libraries: MLlib and ML. MLlib, the older package, offers a wide range of machine learning algorithms suitable for large-scale distributed processing. On the other hand, the ML package, introduced in Spark 2.0, provides higher-level APIs, DataFrame-based pipelines, and supports more modern machine learning techniques. Model selection entails choosing the most suitable algorithm for the task, considering factors like model complexity and performance metrics. To avoid overfitting and assess model robustness, K-fold cross-validation is employed, where the dataset is divided into K subsets, with each subset used as a testing set while the rest are used for training, iteratively. Finally, model evaluation measures the performance of the selected model on unseen data using appropriate evaluation metrics such as accuracy, precision, recall, and F1-score, ensuring the model's effectiveness and generalization capability.

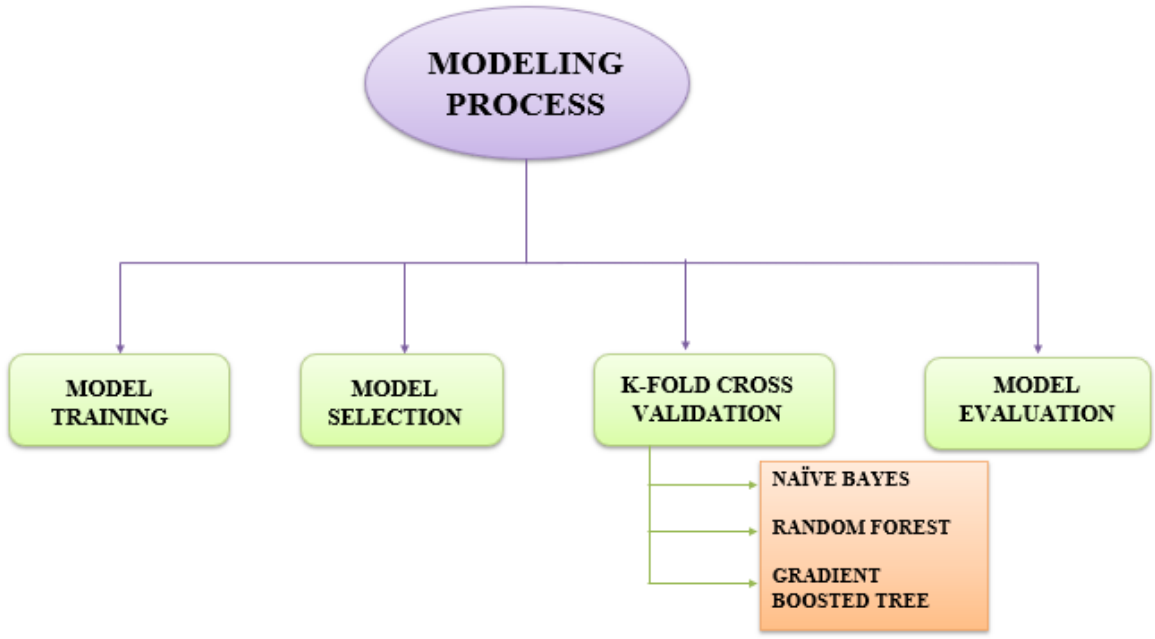

Fig 4. Flowchart of Modeling Process

6.1. MODEL TRAINING

Model training is a fundamental aspect of machine learning where algorithms learn patterns and relationships within data to make predictions or decisions. Within the Spark ecosystem, the MLlib package offers a rich collection of machine learning algorithms for various tasks such as classification, regression, and dimensionality reduction. These algorithms are designed to handle large-scale datasets efficiently in a distributed computing environment. When using MLlib classifiers and regressors, datasets are typically structured as rows, with each row containing both the label or target variable and a list of features. During training, the dataset is split into training and testing sets, often using techniques like cross-validation to ensure robustness and generalization. For instance, in the context of churn prediction, a dataset containing information about customer attributes and behavior can be split into training and testing sets (CV_data).

In a scenario where a random forest classifier is employed, the model is trained using the training data. Parameters such as the maximum number of trees, which in this case is set to 10, determine the complexity of the model and its ability to capture underlying patterns in the data. Additionally, specific features within the dataset play a crucial role in decision-making. For instance, feature 12 (Customer service calls) and feature 4 (Total day minutes) are identified as having high predictive power in determining a customer's likelihood to churn. These features are likely selected based on prior analysis or domain knowledge, indicating their significance in influencing the target variable.

Model training with MLlib involves selecting appropriate algorithms, structuring the data, specifying parameters, and identifying relevant features to build predictive models tailored to specific tasks such as churn prediction.

6.2. MODEL SELECTION

Model selection in machine learning involves identifying the most suitable algorithm and its parameters for a given task. In Spark's ML package, this process is facilitated by a comprehensive API for creating data processing pipelines, which includes data transformers, estimators, and model selectors. These pipelines are particularly useful as they require data to be formatted as DataFrames. Within such pipelines, two key transformers, StringIndexer and VectorIndexer, are commonly used to index label and feature fields, respectively.

Pipelining in the context of data processing involves chaining together multiple data transformation and analysis steps into a cohesive workflow. This technique enables seamless data flow from one processing stage to another, often in a sequential manner. Each step in the pipeline performs specific operations on the data, such as cleaning, feature engineering, or model training, and passes the processed data to the next step. Pipelining not only enhances efficiency by automating and streamlining the data processing process but also ensures reproducibility and scalability of the analysis pipeline. Additionally, it facilitates the integration of various data processing tasks and algorithms, allowing for more complex and sophisticated data analysis workflows.

To perform model selection, the ML package utilizes k-fold cross-validation techniques, a robust method for assessing a model's performance. This involves dividing the dataset into k partitions, using each partition once as the testing dataset while the others serve as the training dataset. Multiple models are trained using these training sets and evaluated against the testing sets, yielding k performance measurements.

The ML package streamlines this process by incorporating k-fold cross-validation into its workflow, coupling it with a parameter grid builder and an evaluator. This setup allows for systematic comparison of models based on their cross-validation performances across various parameters. The parameter grid builder enables the exploration of different model configurations, while the evaluator assesses model performance using predefined metrics.

Ultimately, model selection involves iterating through different combinations of model parameters within the pipeline, training models, and evaluating their performance using k-fold cross-validation. Through this iterative process, the optimal model configuration is determined, ensuring the selection of the most effective algorithm and parameters for the given task.

6.3. K- FOLD CROSS VALIDATION

K-fold cross-validation is a technique used to assess the performance and generalization ability of a machine learning model, particularly advantageous when dealing with limited data. The process involves partitioning the dataset into k equal-sized folds, with each fold serving as both training and validation data iteratively. After training on k-1 folds, the model's performance is evaluated on the remaining fold, resulting in k performance scores. By averaging these scores, a final performance metric is obtained, providing a more reliable estimate compared to a single train-test split. This method effectively reduces variance in the performance estimate and offers a more accurate assessment of the model's generalization capability. It's essential for evaluating various machine learning algorithms like Naive Bayes, Random Forest Classifier, and Gradient-boosted tree classifier, enabling researchers and practitioners to select the most suitable model for their data. K-fold cross-validation aids in understanding how well a model generalizes to new, unseen data.

6.3.1. NAÏVE BAYES

Naive Bayes is a group of simple yet effective probabilistic classifiers that rely on Bayes' Theorem with a key assumption: features are conditionally independent given the class label. This assumption of independence is what makes the algorithm "naive," but it's also the key to its simplicity and computational efficiency. Despite its simplicity, Naive Bayes has been successfully applied in a wide range of domains, including spam filtering, document classification, sentiment analysis, and even medical diagnosis. At its core, Naive Bayes uses the concept of conditional probability to predict the likelihood of a given instance belonging to a particular class. It does this by estimating the conditional probabilities of each feature given the class, then combining these probabilities using Bayes' Theorem to derive the overall probability of the instance belonging to each class. The class with the highest probability is then chosen as the predicted class for the instance.

One of the most significant advantages of Naive Bayes is its computational efficiency. Because it doesn't require complex matrix operations or iterative optimization, it can be applied to large datasets without significant computational overhead. This makes it particularly useful in scenarios where quick predictions are needed, such as real-time spam detection or large-scale text categorization. However, the independence assumption can be a limitation. In real-world data, features often exhibit correlations, and this can lead to suboptimal performance of the algorithm in cases where these correlations play a critical role in defining the classes. Despite this limitation, Naive Bayes often performs surprisingly well even in scenarios where the independence assumption doesn't hold entirely true, thanks to its ability to capture overall trends and patterns in the data.

Overall, Naive Bayes is a valuable tool in the machine learning toolkit, especially for problems that require speed and simplicity. It is often used as a baseline for comparison with more complex algorithms, and its interpretability and ease of implementation make it an attractive option for beginners in machine learning as well as seasoned practitioners.

The ROC curve provided below for Naive Bayes depicts the relationship between False Positive Rate (FPR) and True Positive Rate (TPR), occupying an area of 0.61 under the curve. Naive Bayes classifier achieved an accuracy of 53.67%.

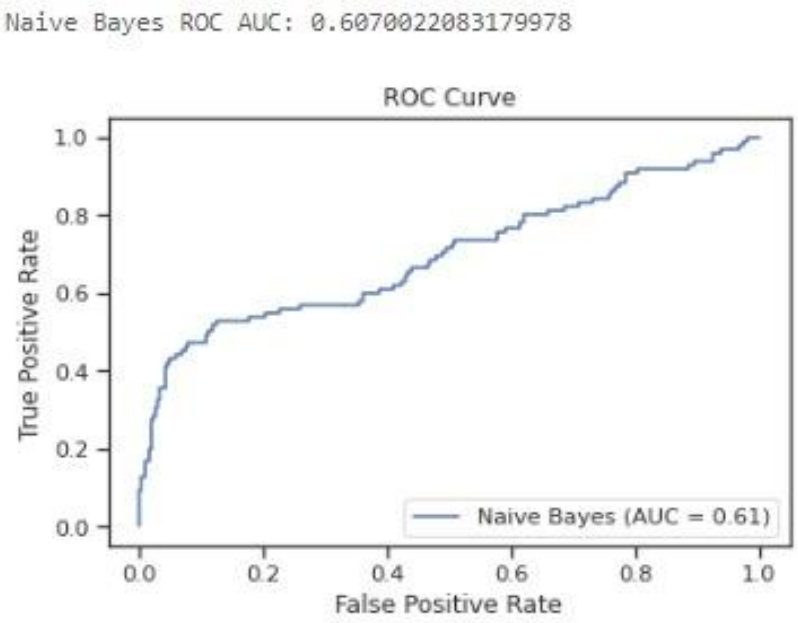

Fig 5. AreaunderROC curve of Naïve Bayes

6.3.2. RANDOM FOREST

Random Forest is a robust ensemble learning algorithm used for both classification and regression tasks. It operates by constructing multiple decision trees during training and then combining their outputs to produce a more accurate and stable result. The "forest" in its name refers to the collection of individual trees, each built from a random subset of the training data and a random selection of features, ensuring diversity and reducing the risk of overfitting.

The process begins by creating multiple bootstrap samples from the original training dataset, a technique known as bagging. Each sample is used to build a separate decision tree, where at each node, the algorithm randomly selects a subset of features to consider when deciding on the best split. This randomness not only creates diverse trees but also guards against the dominance of any one feature or bias in the dataset.

Once the forest of trees is built, predictions are made by aggregating the outputs from all the trees. For classification tasks, this is typically done through a majority vote, where the most common class among the trees is chosen as the final prediction. In regression tasks, the average of the outputs from all the trees is used. This aggregation process leads to a model that is generally more accurate and resilient to noise compared to individual decision trees.

Random Forest's strength lies in its balance between flexibility and robustness. It is capable of handling high-dimensional data, missing values, and categorical variables without extensive preprocessing. Additionally, it inherently provides measures of feature importance, allowing users to gain insights into which features are most influential in making predictions.

However, Random Forest has some limitations. It can require significant computational resources, especially with a large number of trees or complex data. Furthermore, it can be less interpretable than a single decision tree, as the ensemble nature makes it difficult to trace the path leading to a specific prediction. Overall, Random Forest is a versatile and powerful algorithm widely used in various domains, from finance to healthcare, due to its

robustness, accuracy, and ability to handle diverse data types. Its ensemble approach helps mitigate overfitting and ensures consistent performance across different datasets.

The ROC curve for Random Forest depicts the relationship between False Positive Rate (FPR) and True Positive Rate (TPR), occupying an area of 0.88 under the curve. Random Forest classifier achieved an accuracy of 88.91%.

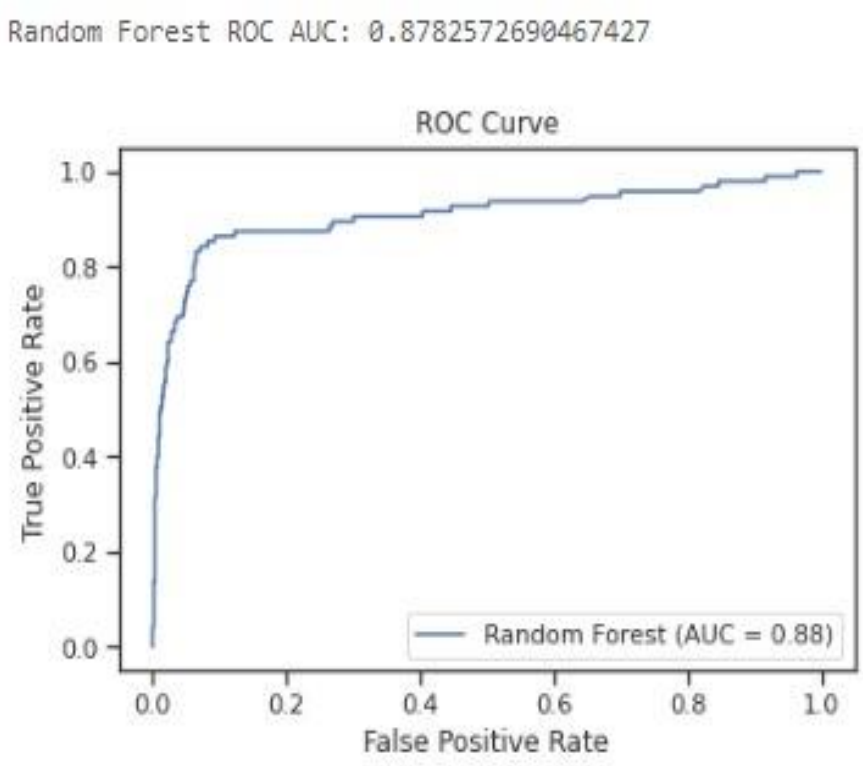

Fig 6. AreaunderROC curve of Random Forest

6.3.3. GRADIENT BOOSTED TREE

Gradient boosted trees, or Gradient Boosting, is a powerful ensemble learning technique used primarily for classification and regression tasks. The concept is rooted in boosting, where multiple weak learners, typically decision trees, are combined to create a strong predictive model. What sets Gradient Boosting apart is its iterative approach to learning, wherein each new tree is built to correct the errors made by the previous trees.

The Gradient Boosting algorithm begins with a simple model, often a decision tree with limited depth, and iteratively adds more trees to improve performance. The key idea is that each tree is trained on the residual errors of the previous iteration. By focusing on these errors, the algorithm incrementally enhances its accuracy, targeting the specific instances or features that the earlier models struggled to predict accurately. This process continues until a predetermined number of trees are created or a certain level of accuracy is achieved.

A crucial component of Gradient Boosting is the learning rate, a hyperparameter that controls how much each new tree contributes to the overall prediction. A lower learning rate requires more trees to reach optimal performance but can lead to more robust and stable results. This balance between the learning rate and the number of trees allows Gradient Boosting to be fine-tuned for a wide range of applications.

One of the major advantages of Gradient Boosting is its flexibility and capacity to create highly accurate models. It can handle a variety of data types and distributions, making it applicable to numerous domains, from finance to healthcare. Additionally, it has built-in mechanisms to avoid overfitting, such as regularization techniques like shrinkage and early stopping.

Gradient Boosting is a highly effective ensemble learning technique that combines the strengths of multiple decision trees to create robust and accurate models. Its iterative approach to learning from errors, combined with flexible hyperparameter tuning, makes it a valuable tool in the machine learning landscape, despite its computational demands and reduced interpretability.

The ROC curve for Gradient Boosted tree depicts the relationship between False Positive Rate (FPR) and True Positive Rate (TPR), occupying an area of 0.86 under the curve. Gradient Boosted Tree classifier achieved an accuracy of 92.05%.

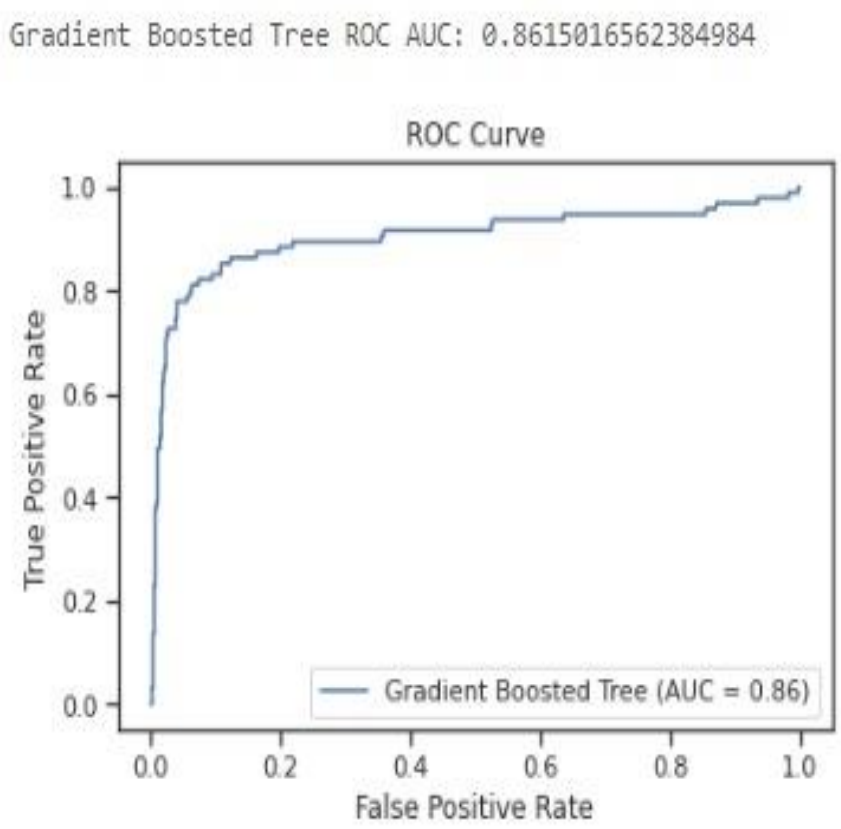

Fig 7. AreaunderROC curve of Gradient Boosted Tree

6.4. MODEL EVALUATION

Model evaluation is a critical step in assessing the performance of machine learning models, ensuring their effectiveness in real-world applications. In the context of Spark MLlib and ML, three essential functions are employed to compute metrics: MulticlassMetrics, BinaryClassificationEvaluator, and MulticlassClassificationEvaluator, each providing insights into model performance across various dimensions.

- **MulticlassMetrics**: from this function, it is possible to compute the confusion matrix, precision per class, weighted precision, recall per class, weighted recall, Fmeasure or score per class, weighted Fmeasure and accuracy.

- **BinaryClassificationEvaluator**: with this function, it is possible to compute two metrics which are the area under ROC and the area under PR.
- **MulticlassClassificationEvaluator**: according to this function, it is able to compute f1, weighted precision, weighted recall and accuracy.

However, a common challenge arises when dealing with imbalanced datasets, as is often the case in predictive analytics. In scenarios where the distribution of target classes is skewed, the model's sensitivity might exhibit bias towards the majority class, potentially overlooking important patterns in the minority class. In the specific case of churn prediction, where the goal is to identify customers at risk of leaving, it's crucial to ensure that the model is sensitive to true churn instances, despite their relatively lower frequency compared to non-churn instances. To address this imbalance, stratified sampling is employed, wherein the minority class is subsampled to match the size of the majority class.

The efficacy of this approach is subject to evaluation. Despite the intention to balance class representation, the performance metrics computed using stratified data may not necessarily outperform those computed using the original imbalanced dataset. In practice, this discrepancy highlights the need for careful consideration of model selection and evaluation techniques. Notably, the Random Forest and Gradient Boosted Classifiers emerge as promising choices, with the Gradient Boosted Trees (GBT) demonstrating particularly balanced performance across various metrics. Ultimately, the selection of the most suitable model hinges on a comprehensive assessment of performance across training, cross-validation, and test datasets, ensuring robustness and generalization to unseen data.

Classifier name	underROC_train	underROC_test	Accuracy_train	Accuracy_test	f1_train	f1_test	wPrecision_train	wPrecision_test	wRecall_train	wRecall_test
NB	0.602	0.607	0.602	0.537	0.602	0.537	0.603	0.810	0.602	0.537
RF	0.889	0.878	0.889	0.889	0.889	0.889	0.892	0.918	0.889	0.889
GBT	0.947	0.862	0.947	0.921	0.947	0.921	0.952	0.925	0.947	0.921

Fig 9. Table of Performance metrics for the algorithms

A Confusion matrix is a fundamental tool for evaluating the performance of a classification model. It provides a comprehensive summary of the model's predictions compared to the actual outcomes. The confusion matrix is a table with four cells: true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). Each cell represents a count of instances where the model correctly or incorrectly predicted the outcome (churn or non-churn) compared to the actual outcome. True positives are cases where the model correctly predicts churn, false positives are cases where the model incorrectly predicts churn when it didn't occur, true negatives are cases where the model correctly predicts non-churn, and false negatives are cases where the model incorrectly predicts non-churn when churn did occur. Analyzing the confusion matrix allows for the calculation of various performance metrics such as accuracy, precision, recall, and F1-score, which is listed above.

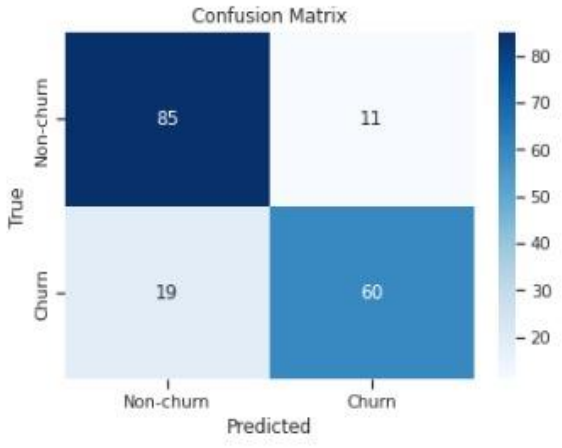

Fig 10. Confusion matrix for 'churn' and 'non-churn'

7. CONCLUSION

Customer churn prediction with PySpark has unveiled a powerful and scalable approach to understanding and forecasting customer attrition. Leveraging PySpark's distributed computing capabilities, analysts can efficiently process and analyze complex datasets, laying the foundation for robust churn prediction models. Through meticulous feature engineering, data transformation, and the utilization of advanced machine learning algorithms like Random Forest and Gradient Boosted Trees, PySpark facilitates comprehensive churn analysis. By employing cross-validation techniques, the reliability and accuracy of these models can be ensured, empowering businesses to proactively identify at-risk customers and devise targeted retention strategies. Notably, among the models evaluated, Gradient Boosted Trees achieved the highest accuracy of 92.05%. Ultimately, PySpark equips organizations with the tools and insights necessary to make data-driven decisions, significantly reducing churn rates and enhancing customer retention.

8. FUTURE SCOPE

The future scope of customer churn prediction involves enhancing accuracy through advanced analytics, real-time monitoring, and expanded model deployment. This includes constructing an ETL pipeline to streamline data processing from SQL servers to PostgreSQL via Apache Spark environments like Databricks. Leveraging techniques such as ensemble learning and real-time monitoring enables businesses to refine prediction models and intervene promptly to prevent churn. Integration with external data sources enriches models with additional insights, while deploying various machine learning models like Decision Trees and SVM broadens the scope of churn prediction, ensuring sustainable business growth by optimizing retention strategies.

9. REFERENCES

[1] Priyanshu Verma, Ishan Sharma, Sonia Deshmukh, and Rohit Vashisht. Customer Churn Analysis using Spark and Hadoop [J]. Int J Performability Eng, 2023, 19(10): 663-675.

[2] Customer churn prediction in telecommunications, Volume 39, Issue 1, by Bingquan Huang,Mohand Tahar Kechadi and Brian Buckley

[3] Machine learning with PySpark P Singh - Berkeley: Apress, 2019 - Springer

[4] Andrew P. Bradley,The use of the area under the ROC curve in the evaluation of machine learning algorithms,Pattern Recognition,Volume 30, Issue 7,1997

[5] Luo, B., Shao, P., Liu, J. 2007. Customer churn prediction based on the decision tree in personal handyphone system service. In International conference on service systems and service management

[6] T. Vafeiadis, K.I. Diamantaras, G. Sarigiannidis, K.Ch. Chatzisavvas, A comparison of machine learning techniques for customer churn prediction, Simulation Modelling Practice and Theory, Volume 55, 2015

[7] Customer churn prediction in telecom using machine learning in big data platform AK Ahmad, A Jafar, K Aljoumaa - Journal of Big Data, 2019 - Springer

[8] Application of optimized machine learning techniques for prediction of churn S Sarkar, S Vinay, R Raj, J Maiti, P Mitra - Computers & Operations 2019 - Elsevier

[9] Comparison of naive bayes, random forest, decision tree, support vector machines, and logistic regression classifiers for text reviews classification T Pranckevičius, V Marcinkevičius - Baltic Journal of Modern Computing, 2017 - bjmc.lu.lv

10.APPENDIX

Graphs:

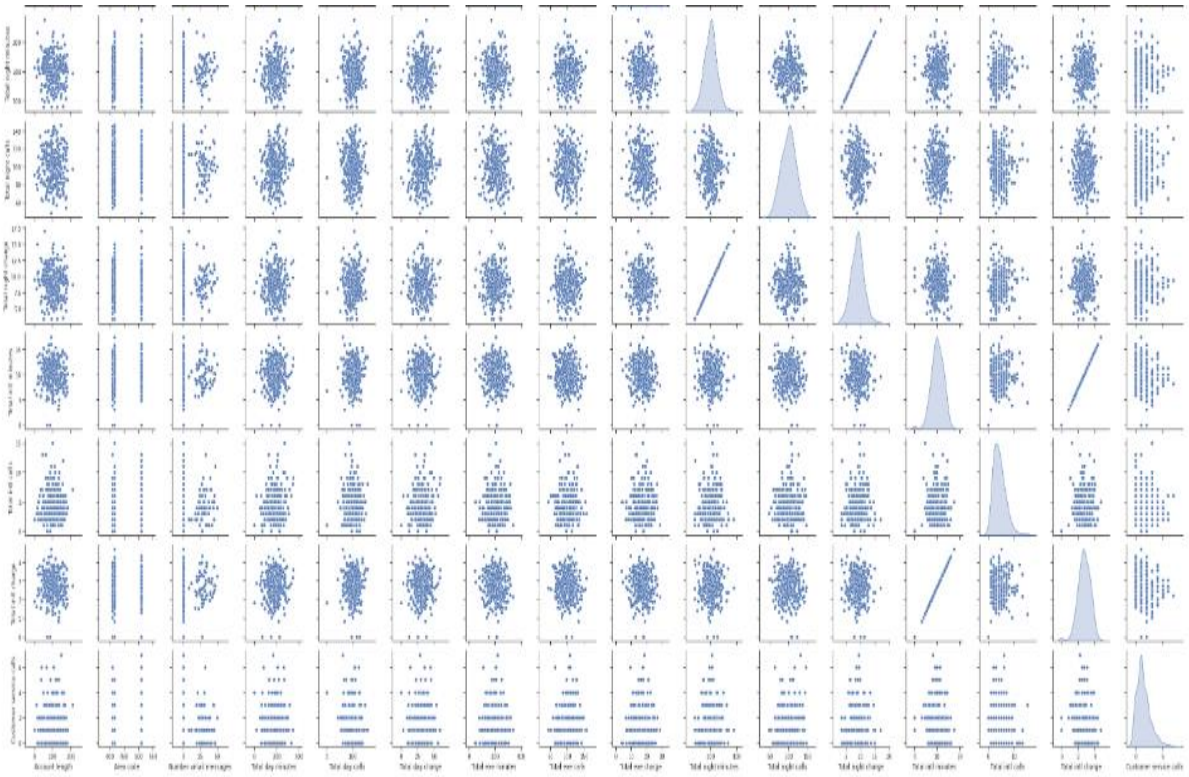

Fig 11. Scatter plots examining the correlation between the numerical columns

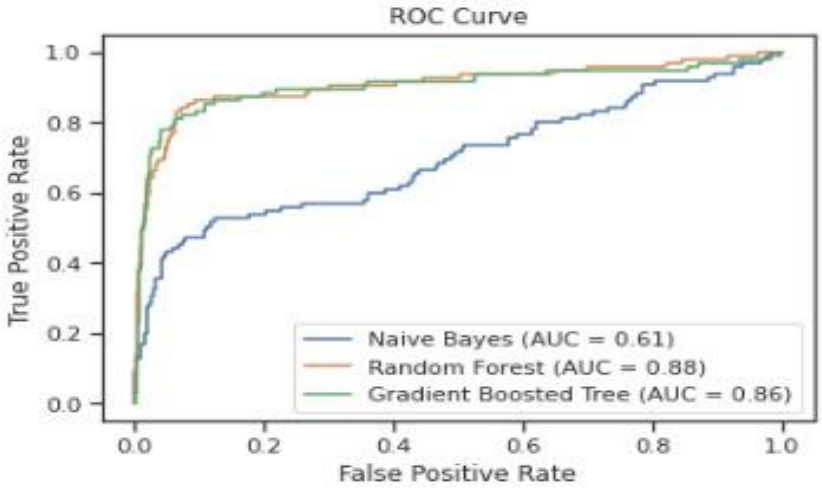

Fig 8. Comparison of all three algorithms using AreaunderROC graph

Datasets:

https://www.kaggle.com/code/bandiatindra/telecom-churn-prediction

https://www.kaggle.com/datasets/muhammedsar/churn-datacsv?select=churn_data.csv