INTERNATIONAL UNIVERSITY- VIETNAM NATIONAL UNIVERSITY, HCM CITY

School of Computer Science & Engineering

--------------ඔ 📖 ෫--------------

FINAL REPORT

CUSTOMER CHURN PREDICTION WITH PYSPARK PROJECT

BIG DATA TECHNOLOGY

2024 – 2025

Students:	Trinh Tien Dat	ITDSIU20109
	Phan The Thien	ITDSIU20084
Course By:	Dr. Van Ho Long	

TABLE OF CONTENTS

I.   INTRODUCTION
1. Introduction to the churn prediction problem
- In the highly competitive telecommunications industry, retaining existing customers is critical to the sustainable development of any business. Customer churn—when customers discontinue using a service or switch to another provider—not only decreases revenue but also increases costs because the need to find and attract new customers.
- Predicting customer churn is a crucial solution for telecommunications companies to identify high-risk customers early. Utilizing machine learning techniques on behavioral data and customer information, businesses can determine the factors contributing to churn. This insight allows them to devise effective strategies to enhance customer service and retention

2. Project goals
This project aims to build a highly accurate churn prediction model using real-world data from the telecommunications industry. The primary objective is to identify customers at risk of churn and provide actionable strategic recommendations. These could include optimizing pricing policies, offering personalized incentives, or improving overall service quality to retain customers.

3. Significance of churn prediction in the business sector.
The importance of churn prediction in the telecommunications industry extends far beyond increasing revenue and reducing costs. It also plays a critical role in improving customer experience, fostering long-term loyalty, and strengthening a company's competitive position in the market. Implementing these solutions forms a cornerstone of sustainable development strategies for telecommunications companies.

II.   DATA DESCRIPTION

- The dataset used in the project contains detailed information about a telecommunications company's customers, including attributes related to time spent in service, call behavior, and churn. Below is a detailed description of the columns.
- Number of columns: 20.
- Data type: Includes continuous, discrete, and categorical data columns.
- Target variable: Churn - this is the target variable predicted in the project.
- Important characteristics: Attributes such as International plan, Total day charge, Customer service calls are expected to have a great influence on customers' churn decisions.

Column	Description
State	State or region code where the customer lives.
Account length	Time (number of days) the customer has used the service.
Area code	The customer's area code (geographic area partition).
International plan	Whether or not the customer has registered for an international package (Yes/No).
Voice mail plan	Whether or not the customer has registered for voice mail service (Yes/No).
Number vmail messages	Number of voice messages in the customer's mailbox.
Total day minutes	Total number of daytime calling minutes used by the customer.
Total day calls	Total number of daytime calls the customer made.

Total day charge	Total cost that customers pay for daytime calls.
Total eve minutes	Total number of evening calling minutes used by the customer.
Total eve calls	Total number of evening calls the customer made.
Total eve charge	Total cost that customers pay for evening calls.
Total night minutes	Total number of night calling minutes used by the customer.
Total night calls	Total number of night calls the customer made.
Total night charge	Total cost that customers pay for night calls.
Total intl minutes	Total number of international calling minutes used by the customer.
Total intl calls	Total number of international calls the customer has made.
Total intl charge	Total cost that customers have to pay for international calls.
Customer service calls	Number of times customers contacted customer service.
Churn	Status of whether the customer leaves the service or not (Yes/No).

III.   DATA PREPROCESSING

1. Initializing a Spark session

Spark was chosen for its ability to process large data efficiently, integrate well with machine learning libraries, and expand flexibly. Additionally, its in-memory computing boosts performance for iterative tasks, and it handles streaming data in real-time effectively. The rich API support across multiple languages adds to its versatility

```python
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Churn Prediction with PySpark").getOrCreate()
```

Figure 1: Importing and configuration of Spark Session

2. Fetching and importing churn data

- We will be using the. Telecom Churn Dataset on Kaggle. It consists of cleaned customer activity data (features), along with a churn label specifying whether the customer canceled their subscription or not.
- The two sets are from the same batch but have been split by an 80/20 ratio. We will use the larger set for training and cross-validation purposes, and the smaller set for final testing and model performance evaluation.

```python
df = ["/content/churn-bigml-80.csv"]
df = [spark.read.csv(path, header='true', inferSchema='true') for path in df]
for CV_data in df:
    CV_data.printSchema()
final_test_data = spark.read.csv("/content/churn-bigml-20.csv", header='true', inferSchema='true')
for CV_data in df:
    print("The training dataset contains {} samples.".format(CV_data.count()))
print("The test dataset contains {} samples.".format(final_test_data.count()))
```

Figure 2: Read data from CSV files into a list of DataFrames

- The training dataset contains 2666 samples.
- The test dataset contains 667 samples.

3. Summarize statistics

- Spark DataFrames include some built-in functions for statistical processing. The describe () function performs summary statistics calculations on all numeric columns, and returns them as a Dataframe.

	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

Figure 3: Summary statistics data frame of numeric columns

4. Correlations and Data Preparation

- To analyze the relationship between attributes related to customer churn, we use the seaborn and matplotlib libraries to create a heatmap of the correlation matrix

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate the correlation matrix
correlation_matrix = sampled_data.toPandas().corr()

# Create a heatmap using seaborn
plt.figure(figsize=(12, 10))  # Adjust the figure size as needed
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title('Correlation Matrix of Churn Attributes', fontsize=14, fontweight='bold')
plt.show()
```

Figure 4: Code to visualize data using seaborn

- The heatmap chart below illustrates the correlation between churn attributes. Correlation values from -1 to 1 are represented in different colors, helping to clearly identify strong or weak relationships between attributes. This aids in identifying attributes that may influence customer churn.
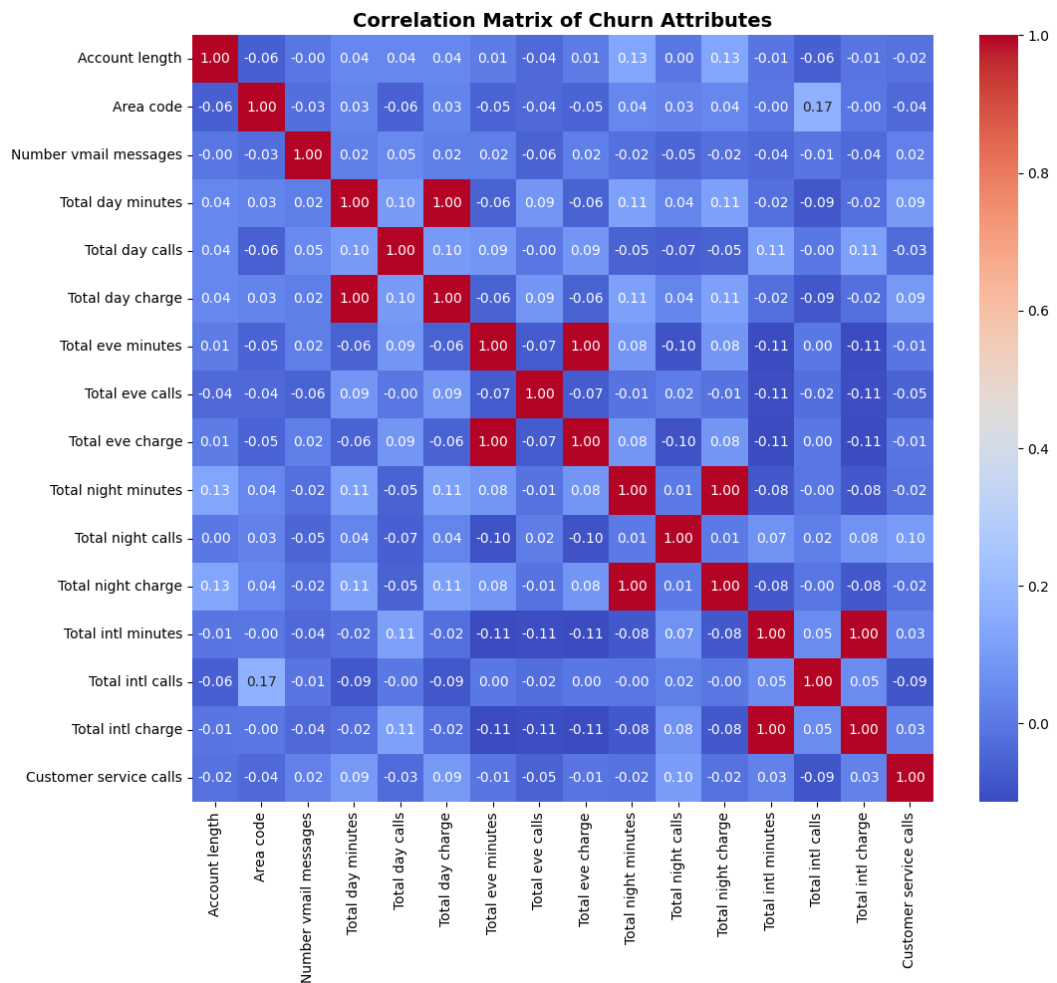
7

Figure 5: Visual chart showing the relationship between pairs of numeric attributes

- It is clear that there are some fields that are highly correlated, i.e. Total Minutes for the Day and Total Charges for the Day. Such correlated data would not be beneficial for model training, so we eliminated them. We did so by dropping one column of each pair of correlated variables along with the State and Region code columns.

```python
from pyspark.sql.functions import split, col, round, when

removeCols = ['State', 'Area code', 'Total day charge', 'Total eve charge', 'Total night charge', 'Total intl charge']

CV_data = get_data(CV_data, removeCols=removeCols)
final_test_data = get_data(final_test_data, removeCols=removeCols)
```

Figure 6: Code to remove unnecessary columns

- Convert categorical data to numbers as required by the machine learning process, using a simple user-defined function to map Yes/ True and No/False to 1 and 0, respectively. All these tasks will be performed using the following get_data function.

```python
def get_data(df, removeCols):
    df = df.drop(*removeCols) \
        .withColumn("Churn", when(df["Churn"] == 'true', 1.0).otherwise(0.0)) \
        .withColumn('International plan', when(df["International plan"] == 'Yes', 1.0).otherwise(0.0)) \
        .withColumn('Voice mail plan', when(df["Voice mail plan"] == 'Yes', 1.0).otherwise(0.0))
    return df
```

Figure 7: Code categorical to Numeric

IV. USING SPARK MLLIB PACKAGE

- Apache Spark MLlib is a machine learning library in Apache Spark, designed for big data processing and analysis.
- Main features:
  - ✓ Big data processing: MLlib is designed to work with data at large scale across computer clusters.
  - ✓ High performance: Use internal memory to increase processing speed.
  - ✓ Good integration: Works well with Hadoop, HDFS, and other data sources.
  - ✓ MLlib supports algorithms such as Logistic Regression, Linear Regression, K-means, and Decision Tree, …

1. Decision Tree Models

- Decision trees have been crucial in data mining and machine learning since the 1960s. They are used for classification and regression, feature selection, and sample prediction. One of their significant advantages is their transparency, making them easy to understand, interpret, and implement in any programming language using nested if-else statements. Additionally, decision trees require minimal data preparation and can handle both categorical and continuous data. To prevent over-fitting and enhance prediction accuracy, they can be limited in depth or complexity, or combined into ensembles like random forests.
- A decision tree is a predictive model that maps observations (features) to conclusions about the item's label or class. It is created using a top-down approach, splitting the dataset into subsets based on a statistical measure, such as the Gini index or information gain via Shannon entropy. This process continues recursively until a subset contains only samples with the same target class or meets a predefined stopping criterion.

2. Model Tranning

- MLlib classifiers and regressors require data sets in a format of rows of type LabeledPoint, which separates row labels and feature lists, and names them accordingly. The custom labelData () function shown below performs the row parsing.
- We pass the prepared data set (CV_data) and split it further into training and testing sets. A decision tree classifier model is then generated using the training data, using a maxDepth of 2, to build a "shallow" tree. The tree depth can be regarded as an indicator of model complexity.

```
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.tree import DecisionTree

def labelData(data):
    return data.rdd.map(lambda row: LabeledPoint(row[-1], row[:-1]))
training_data, testing_data = labelData(CV_data).randomSplit([0.8, 0.2], seed= 42)
print("The two first rows of the training data RDD:")
print(training_data.take(2))
print("=============================")
model = DecisionTree.trainClassifier(training_data,
                                     numClasses=2, # The numberClasses is 2 (Churn = 0 or 1).
                                     categoricalFeaturesInfo={1:2, 2:2},
                                     impurity='gini',
                                     maxDepth=3, # Maximum tree depth is 2 (avoid overfitting)
                                     maxBins=32)
print(model.toDebugString())
```
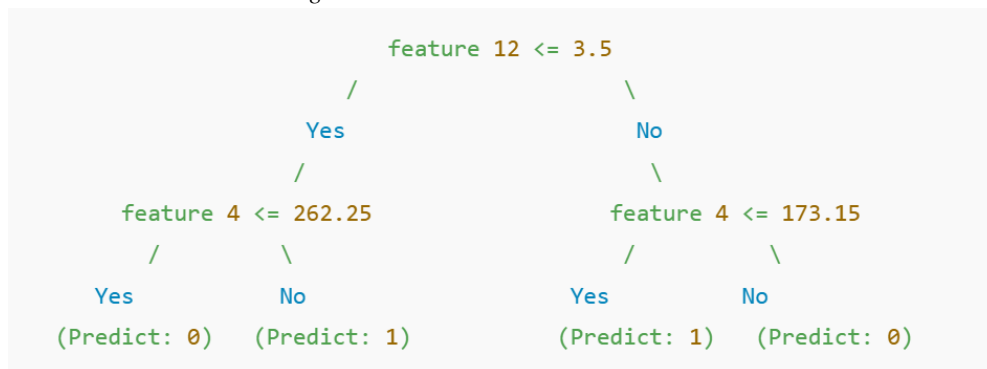
Figure 8: Train the Decision Tree model

```
                        feature 12 <= 3.5
                 /                          \
               Yes                          No
               /                             \
       feature 4 <= 262.25            feature 4 <= 173.15
          /        \                     /          \
       Yes          No                 Yes           No
    (Predict: 0)  (Predict: 1)     (Predict: 1)   (Predict: 0)
```

Figure 9: Visual Representation of Decision Tree Structure

- Based on the model's analysis, we can see that features 12 and 4 are used for decision making and should be considered as having high predictive power to determine a customer's likeliness to churn. These feature numbers map to the fields Customer service calls and Total day minutes. Decision trees are often used for feature selection because they provide an automated mechanism for determining the most important features (root).
- These findings provide actionable insights to help businesses focus on improving customer service quality and better understanding high-usage customers to reduce churn rates.

3. Model Evaluation

- To evaluate the performance of the binary classification model in the problem of predicting churn, we use popular evaluation metrics in machine learning. These metrics not only help us understand the model's accuracy, but also provide insight into how the model classifies different cases (customers churning and not churning).

Metric	Definition	Formula
Precision	The ratio of correctly predicted churn samples to the total predicted churn samples.	$\dfrac{TP}{TP + FP}$
Recall	The ratio of correctly predicted churn samples to the total actual churn samples.	$\dfrac{TP}{TP + FN}$
F1-Score	The harmonic mean of Precision and Recall.	$\dfrac{2.(Precision\ x\ Recall)}{Precision + Recall}$
Accuracy	The ratio of correctly classified samples to the total number of samples.	$\dfrac{TP + TN}{TP + TN + FP + FN}$

Where:
- TP: True Positive (Correctly predicted as churn)
- FP: False Positive (Incorrectly predicted as churn)
- FN: False Negative (Incorrectly predicted as not churn)
- TN: True Negative (Correctly predicted as not churn)

```python
from pyspark.mllib.evaluation import MulticlassMetrics

def getPredictionsLabels(model, test_data):
    predictions = model.predict(test_data.map(lambda r: r.features))
    return predictions.zip(test_data.map(lambda r: r.label))


def printMetrics(predictions_and_labels):
    metrics = MulticlassMetrics(predictions_and_labels)
    print('Confusion Matrix\n', metrics.confusionMatrix().toArray())
    print('Precision of True     ', metrics.precision(1))
    print('Precision of False    ', metrics.precision(0))
    print('Weighted Precision    ', metrics.weightedPrecision)
    print('Recall of True        ', metrics.recall(1))
    print('Recall of False       ', metrics.recall(0))
    print('Weighted Recall       ', metrics.weightedRecall)
    print('FMeasure of True      ', metrics.fMeasure(1.0, 1.0))
    print('FMeasure of False     ', metrics.fMeasure(0.0, 1.0))
    print('Weighted fMeasure     ', metrics.weightedFMeasure())
    print('Accuracy              ', metrics.accuracy)
```

Figure 10: Code to calculate and print out model evaluation indicators

```
Confusion Matrix
 [[442.  18.]
 [ 39.  26.]]
Precision of True     0.5909090909090909
Precision of False    0.918918918918919
Weighted Precision    0.8783081783081783
Recall of True        0.4
Recall of False       0.9608695652173913
Weighted Recall       0.8914285714285715
FMeasure of True      0.47706422018348627
FMeasure of False     0.9394261424017004
Weighted fMeasure     0.8821813329842073
Accuracy              0.8914285714285715
```

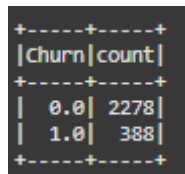*Figure 11: Results model evaluation indicators*

- Model Performance Analysis:

Our model demonstrates a high overall accuracy of 89.1%, indicating a strong overall capability to correctly classify instances. However, there are significant challenges that need to be addressed:

✓ Recall for Churn=True: The recall for the positive class (churn) is notably low at 40%. This indicates that the model is not effectively identifying all potential churners, which is a critical concern for this application.

✓ Data Imbalance: The dataset exhibits an imbalance, with fewer samples for the Churn=True class compared to the Churn=False class. This imbalance affects the model's sensitivity and its ability to accurately identify churn cases.

✓ Strategies such as balancing the dataset, tuning the model parameters, or using ensemble methods may be necessary to improve the model's performance in identifying churners. Ensuring a balanced representation of classes will enhance the model's recall and overall predictive accuracy for critical outcomes.

4. Stratified Sampling

- Our dataset exhibits a significant imbalance between the two classes, "Churn=True" and "Churn=False". Specifically, the "Churn=False" class comprises approximately six times more samples than the "Churn=True" class. This disparity can introduce bias into the model, causing it to skew towards predicting the "Churn=False" class. To address this issue, we employ the stratified sampling method to achieve a balanced distribution of samples between the classes.
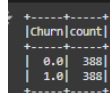


```
+-----+-----+
|Churn|count|
+-----+-----+
|  0.0| 2278|
|  1.0|  388|
+-----+-----+
```

Figure 12: Distribution of Churn Classes in Initial data

- Stratified Sampling method:
  ✓ We have used the sampleBy method to apply stratified sampling on the Churn column. Specifically, the sample rate taken for class Churn=False is calculated by dividing the number of samples of class Churn=True by the number of samples of class Churn=False, while the sample rate of class Churn=True is kept the same is 100%.
  ✓ The sample rate for the Churn=False class is calculated as 388/2278, which reduces the number of samples for this class to balance with the Churn=True class.


```
stratified_CV_data = CV_data.sampleBy('Churn', fractions={0: CV_data.select("Churn").where('Churn == 1').count()/
                                      CV_data.select("Churn").where('Churn == 0').count(), 1: 1.0}, seed = 5323)

stratified_CV_data.groupby('Churn').count().show()

+-----+-----+
|Churn|count|
+-----+-----+
|  0.0|  388|
|  1.0|  388|
+-----+-----+
```

Figure 13: Code Stratified Sampling Process and result

- Let's build a new model using the evenly distributed data set and see how it performs.

```
Confusion Matrix
 [[64.  7.]
  [28. 57.]]
Precision of True        0.890625
Precision of False       0.6956521739130435
Weighted Precision       0.8018873676142697
Recall of True           0.6705882352941176
Recall of False          0.9014084507042254
Weighted Recall          0.7756410256410255
FMeasure of True         0.7651006711409396
FMeasure of False        0.7852760736196318
Weighted fMeasure        0.7742830658588059
Accuracy                 0.7756410256410257
```

Figure 14: The result with data resolved imbalanced

- In conclusion
  - ✓ Auracy decreased (77.6%) but Recall of Churn group improved significantly (67.05%), helping to detect more departing customers.
  - ✓ Churn group F1-Score increased from 46.3% to 76.5%, effectively balancing Precision and Recall.
  - ⇨ **Data balancing has improved churn detection, providing better support for business decisions, although Accuracy has decreased**.

V. USING SPARK MACHINE LEARNING PACKAGE

In this section, we explore the use of the Spark ML package for building and evaluating machine learning models. Spark ML is an enhanced, modern version of MLlib, offering efficient processing and analysis of large datasets. It supports pipelines, enabling seamless chaining of steps from preprocessing to training and evaluation. Additionally, Spark ML is designed to work well with techniques like cross-validation and hyperparameter tuning, which improve model performance and reliability.

1. Learn about Pipeline

The pipeline in Spark ML streamlines the model training process by chaining together data processing and training steps. Key components include:
- StringIndexer: Converts the label column into numeric values.
- VectorAssembler: Combines multiple numeric columns into a single vector column.
- VectorIndexer: Identifies and encodes categorical features.

```python
from pyspark.ml.feature import StringIndexer, VectorIndexer, VectorAssembler
from pyspark.ml import Pipeline

def get_dummy(df, numericCols, labelCol):
    # Combining a given list of columns into a single vector column features
    assembler = VectorAssembler(inputCols=numericCols, outputCol="features")

    # Index labels, adding metadata to the label column
    indexer = StringIndexer(inputCol=labelCol, outputCol='indexedLabel')

    # Automatically identify categorical features and index them
    featureIndexer = VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=2)

    pipeline = Pipeline(stages = [assembler] + [indexer] + [featureIndexer])

    model = pipeline.fit(df)
    data = model.transform(df)

    data = data.withColumn('label', col(labelCol))

    return data.select('features', 'label', 'indexedFeatures', 'indexedLabel')
```

Figure 15: PySpark Data Preprocessing Pipeline

- Overall, using a pipeline helps organize the entire machine learning workflow, making it more structured, efficient, and scalable. This, in turn, allows for better model performance and easier maintenance of the codebase.

2. K-fold cross validation

- K-Fold Cross Validation is a technique for evaluating the performance of a machine learning model by dividing data into k parts (also known as "folds"). This method helps ensure that the model is tested on all parts of the data and not just on a single dataset.
- Operating process:
  - ✓ Data division: The original data is randomly divided into k equal parts.
  - ✓ Training and testing:
    - ❖ Repeat k times
    - ❖ Select a fold to use as a test set.
    - ❖ Use the remaining k-1 to train the model.
  - ✓ Calculate and summarize results:
    - ❖ Evaluate model performance on the test set for each iteration.
    - ❖ Average the results of k tests to get a final assessment of the model.
- Effects of K-Fold cross-validation:
  - ✓ Reduce Overfitting: This method helps minimize the risk of overfitting by ensuring that the model is tested on all parts of the data.
  - ✓ More accurate evaluation: The results of K-Fold Cross Validation provide a more accurate evaluation of model performance because it uses the entire data for both training and testing.
  - ✓ Efficient use of data: This method takes advantage of all available data, especially useful when data is limited.
  - ✓ Balanced test sets: K-Fold Cross Validation ensures that each data sample is used to test the model at least once, making the evaluation fairer and comprehensive.

```
from pyspark.ml.classification import LogisticRegression, DecisionTreeClassifier, RandomForestClassifier
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

paramGrid = (ParamGridBuilder()
            # Logistic Regression parameters
            .addGrid(LogisticRegression.regParam, [0.01, 0.1, 1.0])
            .addGrid(LogisticRegression.elasticNetParam, [0.0, 0.5, 1.0])
            # Decision Tree parameters
            .addGrid(DecisionTreeClassifier.maxDepth, [3, 5, 10])
            .addGrid(DecisionTreeClassifier.impurity, ['gini', 'entropy'])
            # Random Forest parameters
            .addGrid(RandomForestClassifier.numTrees, [10, 50, 100])
            .addGrid(RandomForestClassifier.maxDepth, [3, 5, 10])
            .build())
```

Figure 16: PySpark Model Tuning

- Using CrossValidator and combined with ParamGridBuilder to search for the optimal parameter set, ensuring the model achieves the best results.

3. Model evaluation result

- In addition to key performance metrics such as Precision, Recall, F1-Score, and Accuracy, we also utilize the Area Under the ROC Curve (AUC) to evaluate our model. AUC provides a comprehensive measure of the model's ability to distinguish between positive and negative classes across various threshold settings.
- Besides Decision Tree as above, in this part I use RandomTree and Logistic Regression, then compare their performance to find a good model for the data.
- By comparing these models using these metrics, we can gain a comprehensive understanding of their strengths and weaknesses, allowing us to select the most appropriate model for our specific application. This thorough evaluation ensures that the chosen model not only performs well on accuracy but also on other critical metrics like precision, recall, F1-score, and AUC, providing a balanced and reliable predictive performance.

	Logistic Regression	Decision Tree	Random Forest
underROC_train	0.776	**0.899**	0.881
underROC_test	0.75	0.848	**0.866**
Accuracy_train	0.776	**0.899**	0.881
Accuracy_test	0.751	**0.928**	0.876
F1_train	0.776	**0.899**	0.881
F1_test	0.751	**0.928**	0.876
Precision_train	0.776	**0.912**	0.884
Precision_test	0.86	**0.927**	0.911
Recall_train	0.776	**0.899**	0.881
Recall_test	0.751	**0.928**	0.876

Figure 17: Model Performance Metrics Comparison

- In conclusion,
  - ✓ Logistic Regression (LR): Lowest performance with underROC_test of 0.750.
  - ✓ Decision Tree (DT): the highest accuracy on the test set (Accuracy_test = 0.928).
  - ✓ Random Forest (RF): Solid performance with underROC_test = 0.862 and f1_test = 0.868.
- ⇨ Decision Tree outperforms in terms of accuracy and discrimination, while Logistic Regression performs less well

4. Prediction with Test dataset

Since the Decision Tree model has demonstrated the best performance, we have chosen it to make predictions on the test dataset. The test dataset, which was not utilized during the training phase, provides an unbiased evaluation of the model's ability to generalize to new, unseen data.

```
+-----------+----------+------------------------------------------+
|indexedLabel|prediction|probability                              |
+-----------+----------+------------------------------------------+
|0.0        |0.0       |[0.8580060422960725,0.1419939577039275]|
|1.0        |1.0       |[0.0,1.0]                                |
|1.0        |1.0       |[0.0,1.0]                                |
|0.0        |0.0       |[0.8580060422960725,0.1419939577039275]|
|0.0        |0.0       |[0.8580060422960725,0.1419939577039275]|
|0.0        |0.0       |[0.8580060422960725,0.1419939577039275]|
|0.0        |0.0       |[0.8580060422960725,0.1419939577039275]|
|1.0        |1.0       |[0.0,1.0]                                |
|0.0        |0.0       |[0.8580060422960725,0.1419939577039275]|
|0.0        |0.0       |[0.8580060422960725,0.1419939577039275]|
|0.0        |0.0       |[0.8580060422960725,0.1419939577039275]|
|0.0        |0.0       |[0.782608695652174,0.21739130434782608]|
|0.0        |0.0       |[0.8580060422960725,0.1419939577039275]|
|0.0        |0.0       |[0.8580060422960725,0.1419939577039275]|
|1.0        |0.0       |[0.8580060422960725,0.1419939577039275]|
|0.0        |0.0       |[0.8580060422960725,0.1419939577039275]|
|1.0        |1.0       |[0.0,1.0]                                |
|0.0        |0.0       |[0.8580060422960725,0.1419939577039275]|
|0.0        |0.0       |[0.8580060422960725,0.1419939577039275]|
|0.0        |0.0       |[1.0,0.0]                                |
+-----------+----------+------------------------------------------+
```

Figure 18: Predicting with first 20 values

```
Total samples: 667
Number of correct predictions: 619
Percentage of correct predictions: 92.80%
```

Figure 19: Predicting with all samples in test dataset

16

⇨ The initial prediction accuracy with the first 20 values is perfect at 100%, indicating that the model is highly effective on this small subset. However, when predicting on the entire test dataset, the accuracy drops to 92.8%. This suggests that while the model performs exceptionally well overall, it may face some challenges when dealing with a larger and more diverse set of data. The slight decrease in accuracy highlights the importance of evaluating the model on a comprehensive dataset to get a realistic measure of its performance.

VI. Future Improvements

- Fine-tune Hyperparameters: By fine-tuning the hyperparameters of our Decision Tree, Random Forest, and Logistic Regression models, we can further optimize their performance. This involves systematically searching for the best combination of parameters that maximize the model's predictive accuracy.

- Experiment with Additional Algorithms: Exploring and experimenting with additional machine learning algorithms may provide new insights and potentially yield better results. Techniques such as Gradient Boosting, Support Vector Machines, or Neural Networks could be investigated to see if they offer improvements over our current models.

- Implement Real-time Monitoring: Real-time monitoring for early churn detection is crucial for timely interventions. By integrating the model into a real-time monitoring system, we can promptly identify customers who are likely to churn and take proactive measures to retain them.

VII. References

https://medium.com/swlh/churn-prediction-using-pyspark-a1f4ef0439b3
https://www.kaggle.com/code/mnassrib/customer-churn-prediction-with-pyspark/input