AWS Machine Learning Engineer Nanodegree Capstone

Churn classification

Matheus Ribeiro Cerqueira

2023

Definition

Project Overview

Customer churn, also referred to as customer attrition, poses a significant challenge for businesses. It arises when customers discontinue the use of a company's products or services, and high churn rates can have adverse effects on a company's revenue and profitability [1] .

To tackle this issue, machine learning algorithms can be leveraged to identify the factors that contribute to churn. Churn models are designed to identify early warning signs and recognize customers who are more likely to voluntarily leave [2], [3] and [4]. As part of this project, I will be delving into three different algorithms: logistic regression, decision tree, and random forest. Through the application of these three powerful tools, I aim to develop a highly accurate classifier that can predict which customers are likely to churn and which are not.

For this project, the churn dataset was obtained from Kaggle, specifically the Bank Customer Churn Dataset. I trained three different classification models: Logistic Regression, Decision Tree, and Random Forest, and compared their performance using the AUC-ROC score.

Problem Statement

In the fiercely competitive industry, customer churn poses a significant challenge and is critical to the success of many businesses. Churn transpires when a customer decides to shift to a rival that provides a superior solution or product.

It is imperative for any business to identify customers who are likely to churn and implement effective retention strategies for long-term success, given that acquiring new customers is more expensive than retaining current ones. However, manually identifying such customers or using conventional techniques can be arduous and time-consuming.

Hence, developing machine learning techniques that can accurately identify customers at risk of churning based on their past behavior and interactions can enhance the ability to spot and prevent future churning customers. The objective of this project is to create a model capable of accurately predicting churn and enhancing customer retention.

Dataset description

Table 1. Dataset description

Column Name	Description
customer_id	Unique identifier for each customer. This column is not used as an input for the model.
credit_score	A numerical value that represents the creditworthiness of a customer. It is used as an input to the model to predict churn. Higher scores indicate lower risk and vice versa.
country	The country where the customer resides. It is used as an input to the model to predict churn.
gender	The gender of the customer. It is used as an input to the model to predict churn.
age	The age of the customer. It is used as an input to the model to predict churn.
tenure	The number of years that the customer has been with the bank. It is used as an input to the model to predict churn.
balance	The amount of money that the customer has in his/her account. It is used as an input to the model to predict churn.
products_number	The number of banking products that the customer has with the bank. It is used as an input to the model to predict churn.
credit_card	Whether the customer has a credit card with the bank or not. It is used as an input to the model to predict churn.
active_member	Whether the customer is an active member of the bank or not. It is used as an input to the model to predict churn.
estimated_salary	The estimated salary of the customer. It is used as an input to the model to predict churn.
churn	The target variable. A binary variable that indicates whether the customer has left the bank or not during a certain period. It takes a value of 1 if the customer has left and 0 if not.

In this project, I divided the dataset into three subsets: Training (70%), Validation (10%), and Test (20%).

Metrics

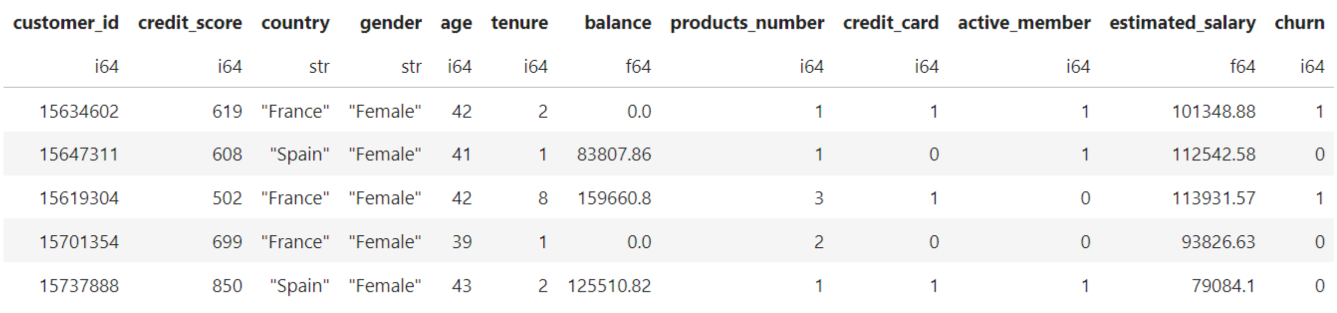
I utilized cross-validation accuracy as my objective metric during training, evaluation, and hyperparameter tuning. To compare models, I used the AUC-ROC score against the test data, and also computed the accuracy score for the test data and generated a confusion matrix.

Analysis

Data Exploration

The dataset contains 10,000 observations with 12 features and is classified into two classes: 0 (not churn or negative class) and 1 (churn or positive class). Figure 1 shows a subset of the dataset, while Figure 2 presents the statistics.

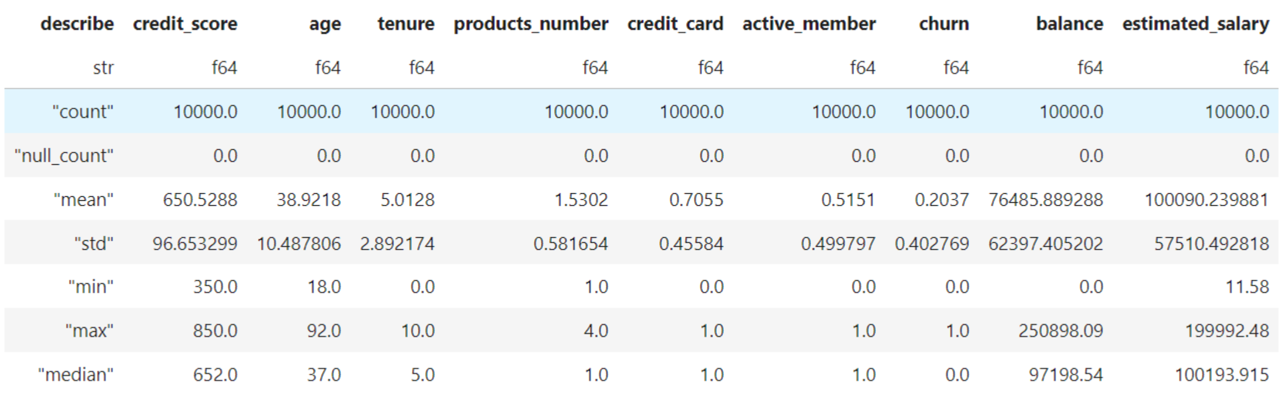
Figure 1: Dataset features



customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
i64	i64	str	str	i64	i64	f64	i64	i64	i64	f64	i64
15634602	619	"France"	"Female"	42	2	0.0	1	1	1	101348.88	1
15647311	608	"Spain"	"Female"	41	1	83807.86	1	0	1	112542.58	0
15619304	502	"France"	"Female"	42	8	159660.8	3	1	0	113931.57	1
15701354	699	"France"	"Female"	39	1	0.0	2	0	0	93826.63	0
15737888	850	"Spain"	"Female"	43	2	125510.82	1	1	1	79084.1	0

- Credit Score (churn): maximum 850 and minimum 350
- Credit Card grouped by class (churn): 7055 (positive class) X 2945 (negative class)
- Total number of churn customer grouped by country: 814 (Germany) X 810 (France) X 413 (Spain)

Figure 2: Dataset statistics

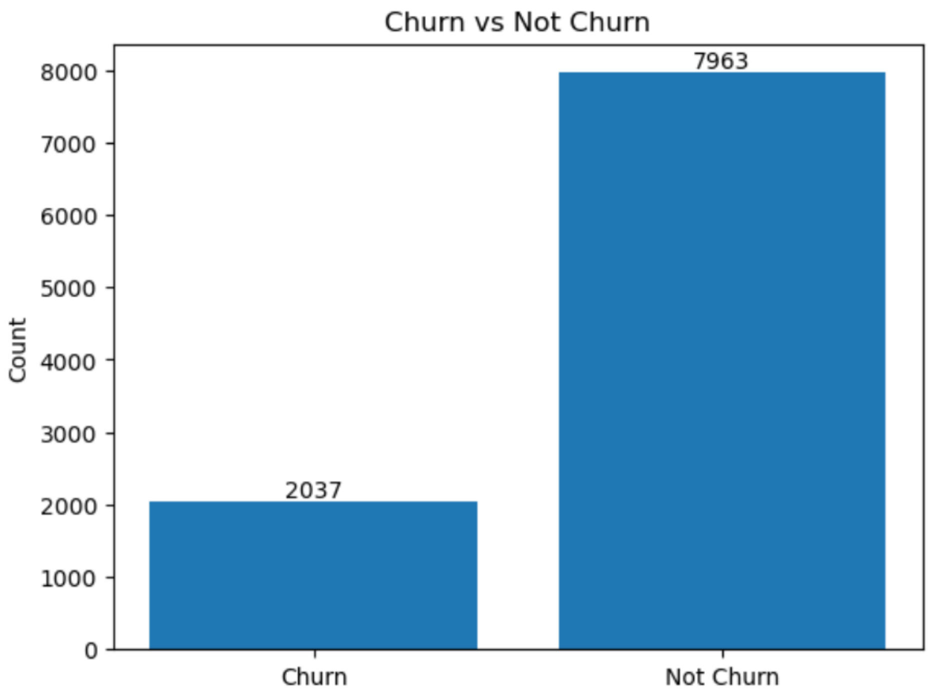


describe	credit_score	age	tenure	products_number	credit_card	active_member	churn	balance	estimated_salary
str	f64	f64	f64	f64	f64	f64	f64	f64	f64
"count"	10000.0	10000.0	10000.0	10000.0	10000.0	10000.0	10000.0	10000.0	10000.0
"null_count"	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
"mean"	650.5288	38.9218	5.0128	1.5302	0.7055	0.5151	0.2037	76485.889288	100090.239881
"std"	96.653299	10.487806	2.892174	0.581654	0.45584	0.499797	0.402769	62397.405202	57510.492818
"min"	350.0	18.0	0.0	1.0	0.0	0.0	0.0	0.0	11.58
"max"	850.0	92.0	10.0	4.0	1.0	1.0	1.0	250898.09	199992.48
"median"	652.0	37.0	5.0	1.0	1.0	1.0	0.0	97198.54	100193.915

Exploratory Visualization

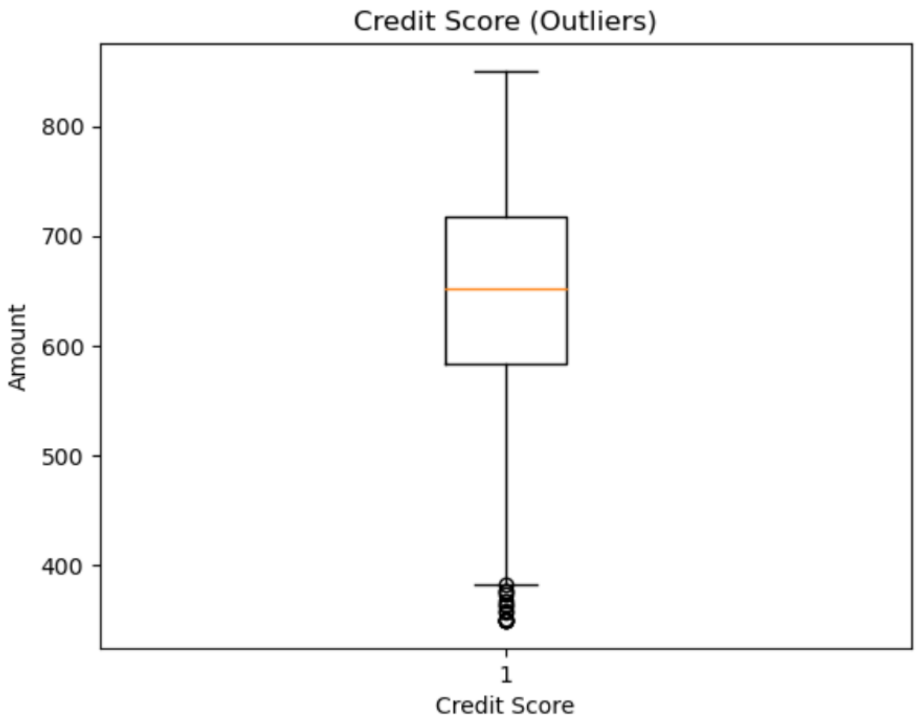
Figure 3 shows that out of the total number of customers in the dataset, 20.37% (2037) are churning customers and 79.63% (7963) are not churning customers.

Figura 3: Total number of churn customers vs not churn



The boxplot depicted in Figure 4 reveals the existence of outliers in the Credit Score feature, which are observations that deviate significantly from the majority of the data points.

Figure 4: Credit Score outliers



Algorithms and Techniques

Supervised classification models are algorithms that utilize labeled data to learn how to classify new, unseen data. These models are commonly applied in image classification, text classification, and speech recognition tasks. Supervised classification models can be divided into two categories: linear models and nonlinear models.

Linear models employ a linear decision boundary to categorize observations into classes, such as churn and non-churn. In contrast, nonlinear models are more intricate yet more flexible than linear models, allowing them to extract complex relationships between input features and output classes using nonlinear decision boundaries.

Logistic Regression is a linear model used for binary classification problems. This algorithm employs a logistic function to generate a probability value between 0 and 1, which represents the likelihood of belonging to a specific class.

A Decision Tree is a type of supervised learning algorithm capable of performing both classification and regression tasks. It is structured like a tree, where each internal node represents a test on a feature, each branch denotes the outcome of the test, and each leaf node represents a class label or numerical value. Unlike Random Forest, Decision Trees are highly intuitive, and their results are easy to interpret since they can be accessed as a graph.

Decision Trees employ Gini impurity or Entropy to select the feature that can best split the data (node) into homogeneous subsets. One characteristic of Decision Trees is that they tend to overfit the training data by fitting it too closely, which is caused by the indeterminate number of parameters prior to training. To mitigate this issue, we can restrict their freedom while training by applying techniques such as modifying the max_depth hyperparameter in scikit-learn.

Random Forest is an ensemble learning algorithm used for classification, regression, and other tasks. Random Forest is trained using a method called bagging, which involves training multiple models on different subsets of the training data and combining their results to make a final prediction.

Random Forest uses a large number of decision trees, and their results are combined to make a final prediction. One key difference between Random Forest and Decision Trees is how they split data (node). Random Forest selects the best feature among a random subset of features instead of finding the best feature. Moreover, Random Forest can reduce overfitting and enhance the accuracy of the model.

In summary, Logistic Regression boasts several advantages, such as its simplicity, interpretability, and ability to handle both numerical and categorical input variables. It is also robust to outliers and can work with both small and large datasets. However, logistic regression may not perform well when the relationship between the input variables and the output variable is highly nonlinear, or when there are interactions between the input variables.

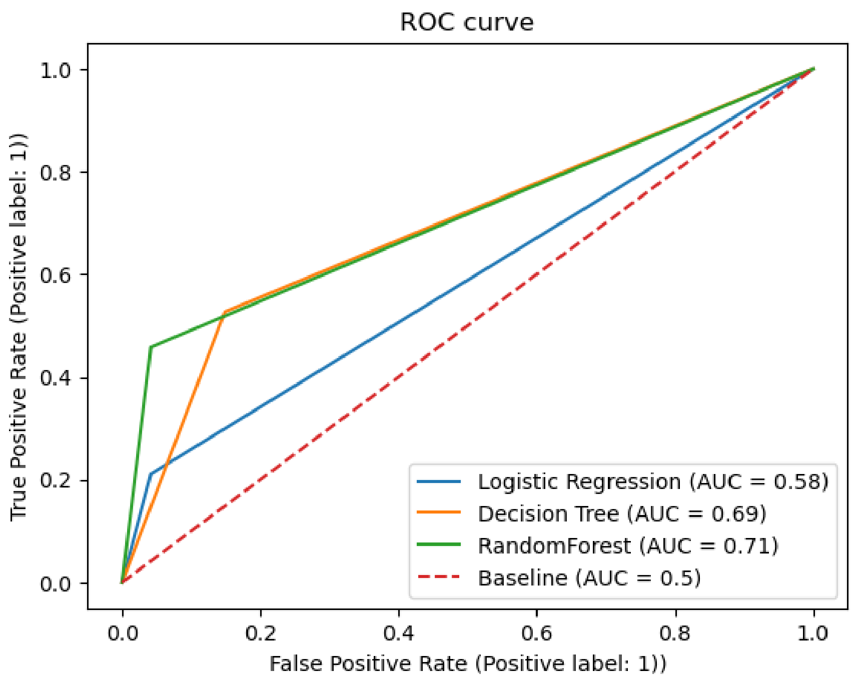
Decision Trees can handle both categorical and continuous input features but are prone to overfitting, which can be alleviated through techniques like pruning and setting a minimum number of samples required to split an internal node. On the other hand, Random Forest can handle high-dimensional data, nonlinear relationships between features, and missing data. It is also less prone to overfitting than a single Decision Tree. Nevertheless, Random Forest models can be computationally expensive to train and require careful tuning of hyperparameters, such as the number of decision trees, the maximum depth of each tree, and the size of the random feature subset.

For this project, I commenced by training and evaluating a set of models on the training data. Subsequently, I employed Synthetic Minority Oversampling Technique SMOTE [7] to the dataset to reduce the impact of imbalance [5] and [6]. I then retrained and re-evaluated the models on this dataset. Finally, I utilized hyperparameter tuning to optimize the models' performance, computed the AUC-ROC score, and compared the models based on the final score.

Benchmark

To establish a performance benchmark, I employed the aforementioned models in their default configurations while working with imbalanced data. The results are presented in Figure 5, which displays the ROC curves and AUC scores for each model, as well as a dashed line baseline that represents a random classification model.

Figure 5: ROC curves and AUC scores for benchmark models.

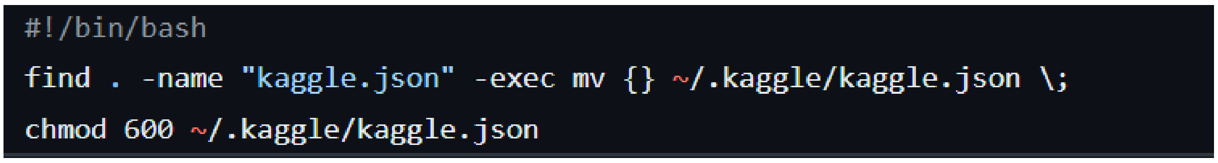


Methodology

Data Download

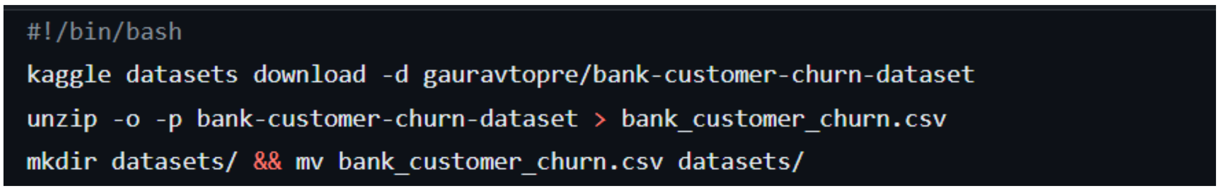
To retrieve the dataset, I utilized the Kaggle API made available by Kaggle. I also wrote a bash script to relocate the Kaggle API file to the appropriate directory and grant it the necessary permissions. Following that, I created an additional bash script that facilitated the download and unzipping of the file.

Figure 6: Bash script to setup Kaggle API



```
#!/bin/bash
find . -name "kaggle.json" -exec mv {} ~/.kaggle/kaggle.json \;
chmod 600 ~/.kaggle/kaggle.json
```

Figure 7: Bash script to download and unzip dataset



```
#!/bin/bash
kaggle datasets download -d gauravtopre/bank-customer-churn-dataset
unzip -o -p bank-customer-churn-dataset > bank_customer_churn.csv
mkdir datasets/ && mv bank_customer_churn.csv datasets/
```

Data Preprocessing

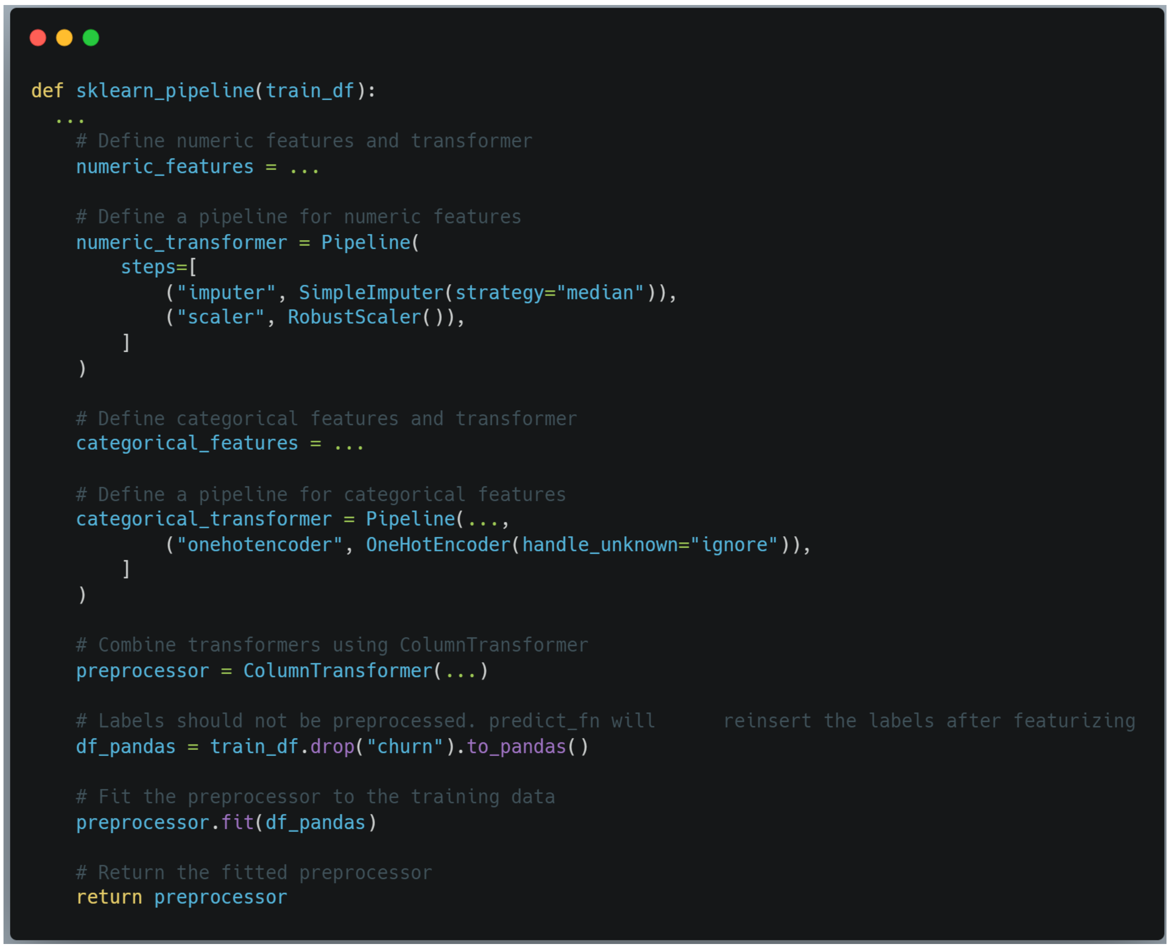
Subsequently, I leveraged Polars, a Rust-based DataFrame library, to import the entire dataset while excluding extraneous features like customer_id. Next, I partitioned the resulting dataset into 70% training data, 10% validation data, and 20% test data using scikit-learn's train_test_split function, which I utilized for training the benchmark models. In addition, I implemented the SMOTE SMOTENC class to resample the dataset before splitting it, and used the resampled dataset to train and evaluate the models, as well as train the models with hyperparameter tuning. For all experiments, I set the random_state to 42 to ensure reproducibility. Finally, I uploaded the resulting training, validation, and test sets to AWS S3.

Implementation

To implement the machine learning training and testing, I opted to use the scikit-learn library due to its user-friendly interface and streamlined workflows, including the use of pipelines for combining tasks such as transformation, dimensionality reduction, and training. Additionally, I utilized SageMaker script mode to enable training and testing of models from scratch. To achieve this, I created Python scripts for each model, as well as a script that incorporated Sckit-learn pipelines.

An overview of the pipeline I developed is presented in Figure 8. For numerical features, I employed a robust scaler to normalize the data and imputed missing values. For categorical features, I used OneHotEncoder to dummy the features and impute missing data. These pipelines were then combined using the ColumnTransformer class to create a unified pipeline, which was fit to the training set.

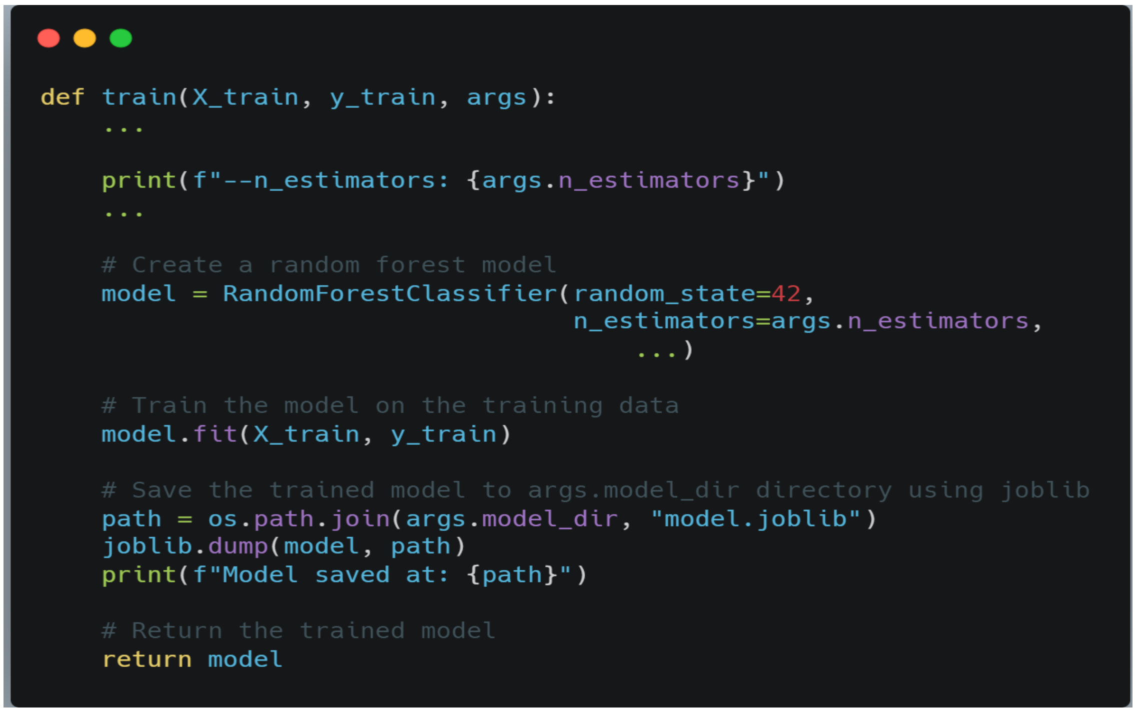
Figura 8: Scikit-Learn pipeline



```python
def sklearn_pipeline(train_df):
  ...
    # Define numeric features and transformer
    numeric_features = ...

    # Define a pipeline for numeric features
    numeric_transformer = Pipeline(
        steps=[
            ("imputer", SimpleImputer(strategy="median")),
            ("scaler", RobustScaler()),
        ]
    )

    # Define categorical features and transformer
    categorical_features = ...

    # Define a pipeline for categorical features
    categorical_transformer = Pipeline(...,
            ("onehotencoder", OneHotEncoder(handle_unknown="ignore")),
        ]
    )

    # Combine transformers using ColumnTransformer
    preprocessor = ColumnTransformer(...)

    # Labels should not be preprocessed. predict_fn will      reinsert the labels after featurizing
    df_pandas = train_df.drop("churn").to_pandas()

    # Fit the preprocessor to the training data
    preprocessor.fit(df_pandas)

    # Return the fitted preprocessor
    return preprocessor
```

To train the model depicted in Figure 9, I established the random_state as 42 and transmitted the hyperparameters via an argparse.ArgumentParser() variable. Subsequently, I fit the training data and validated the results before persisting the model using the joblib library with the default model directory path supplied by SageMaker.

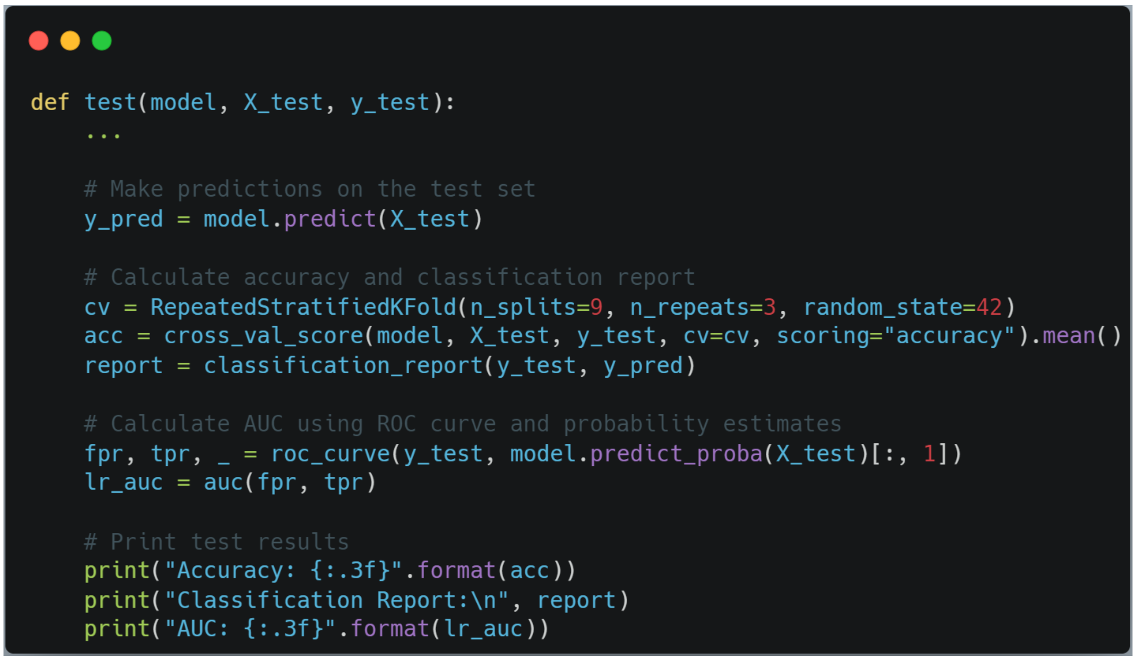
Figura 9: Random Forest train function



```python
def train(X_train, y_train, args):
    ...

    print(f"--n_estimators: {args.n_estimators}")
    ...

    # Create a random forest model
    model = RandomForestClassifier(random_state=42,
                                   n_estimators=args.n_estimators,
                                       ...)

    # Train the model on the training data
    model.fit(X_train, y_train)

    # Save the trained model to args.model_dir directory using joblib
    path = os.path.join(args.model_dir, "model.joblib")
    joblib.dump(model, path)
    print(f"Model saved at: {path}")

    # Return the trained model
    return model
```

As shown in Figure 10, I utilized my model to predict new observations in the validation dataset, while also employing cross-validation via the RepeatedStratifiedKFold method. This involved 9 splits and 3 repeats in order to determine the accuracy of the results, as well as compute the AUC.

Figura 10: Random Forest test function



```python
def test(model, X_test, y_test):
    ...

    # Make predictions on the test set
    y_pred = model.predict(X_test)

    # Calculate accuracy and classification report
    cv = RepeatedStratifiedKFold(n_splits=9, n_repeats=3, random_state=42)
    acc = cross_val_score(model, X_test, y_test, cv=cv, scoring="accuracy").mean()
    report = classification_report(y_test, y_pred)

    # Calculate AUC using ROC curve and probability estimates
    fpr, tpr, _ = roc_curve(y_test, model.predict_proba(X_test)[:, 1])
    lr_auc = auc(fpr, tpr)

    # Print test results
    print("Accuracy: {:.3f}".format(acc))
    print("Classification Report:\n", report)
    print("AUC: {:.3f}".format(lr_auc))
```

Refinement

To refine my models, I created two different scenarios. In the first scenario, I utilized SMOTE to resample my dataset, train and evaluate my models, and then leveraged Hyperparameter optimization to create final models using the same resampled dataset.

For hyperparameter tuning, I ran 10 training jobs and set my objective metric as accuracy. Subsequently, I retrieved the best hyperparameters from my tuning jobs to train my final models.

Table 2 illustrates the accuracy and AUC scores for the benchmark models, while Table 3 demonstrates that the models performed better in terms of accuracy and AUC scores when SMOTE was used. However, the results of the Hyperparameter optimization indicated only slight improvements, with the exception of the Decision Tree model (as depicted in Table 4).

Table 2: Accuracy and AUC scores without SMOTE and Tuning

Model	SMOTE	Hyper. Tuning	Accuracy	ROC AUC
Logistic Regression	FALSE	FALSE	0.81	0.58
Decision Tree	FALSE	FALSE	0.79	0.69
RandomForest	FALSE	FALSE	0.86	0.71

Table 3: Accuracy and AUC scores using SMOTE

Model	SMOTE	Hyper. Tuning	Accuracy	ROC AUC
Logistic Regression	TRUE	FALSE	0.77	0.77
Decision Tree	TRUE	FALSE	0.79	0.79
RandomForest	TRUE	FALSE	0.86	0.86

Table 4: Accuracy and AUC scores using SMOTE and Hyperparameter optimization

Model	SMOTE	Hyper. Tuning	Accuracy	ROC AUC
Logistic Regression	TRUE	TRUE	0.77	0.77
Decision Tree	TRUE	TRUE	0.82	0.82
RandomForest	TRUE	TRUE	0.84	0.85

To measure accuracy using k-fold cross-validation, I utilized the RepeatedStratifiedKFold class from scikit-learn library, with the following parameters: n_splits=9, n_repeats=3, and random_state=42.

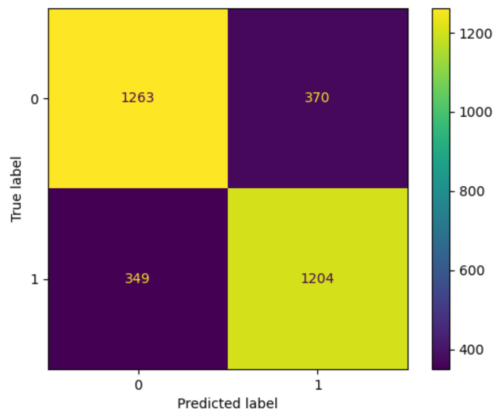
Results

Model Evaluation and Validation

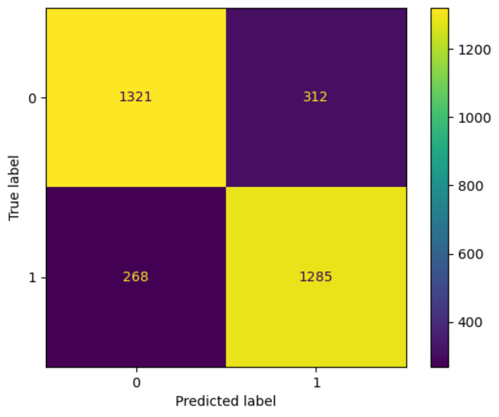
To evaluate the performance of my models, I utilized a validation set during training to perform k-fold cross-validation and compute the AUC score. Afterwards, I utilized a separate test set to validate the models and compute the final accuracy and AUC. To further analyze the results, I created confusion matrices for the final models.

The confusion matrix for the Random Forest (Figure 11C) model showed that the model correctly identified 84.5% of the positive samples and 85.3% of the negative samples. However, the model incorrectly classified 15.5% of the positive samples as negative (false negatives) and 14.7% of the negative samples as positive (false positives). Overall, this confusion matrix suggests that the model is performing well in predicting churned customers, but there is room for further improvement.

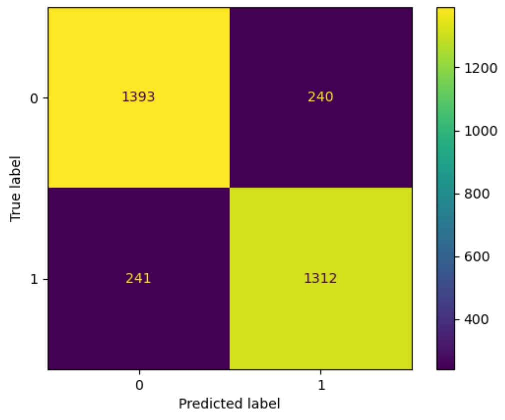
Figure 11: Confusion matrices for Logistic Regression (A), Decision Tree (B) and Random Forest (C)



(A)



(B)



(C)

Afterwards, I compared the models using the AUC-ROC score, using both SMOTE and a combination of SMOTE and Hyperparameter Tuning to determine the best one (as depicted in Figure 12 and Figure 13, respectively).

Figure 12: ROC curves (SMOTE)

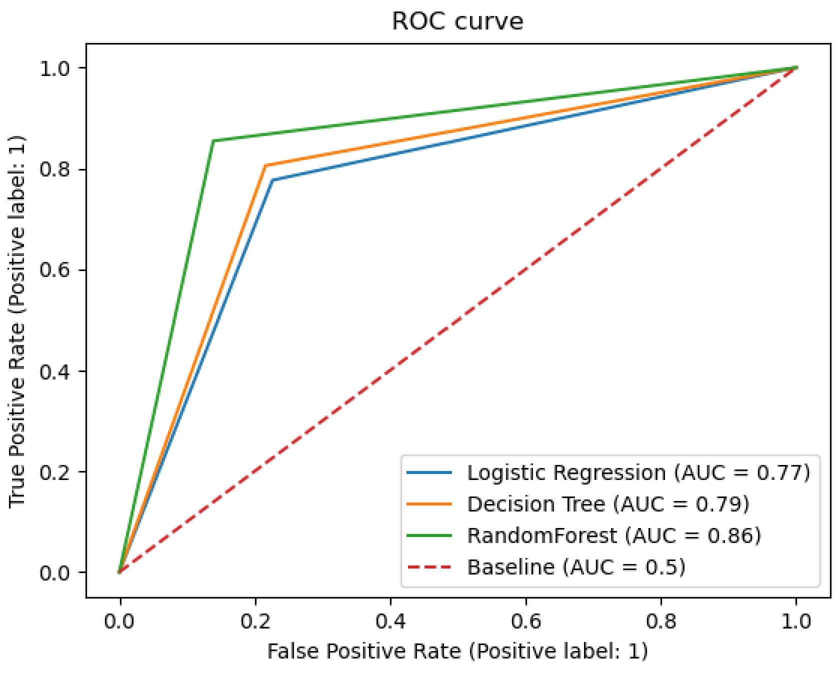
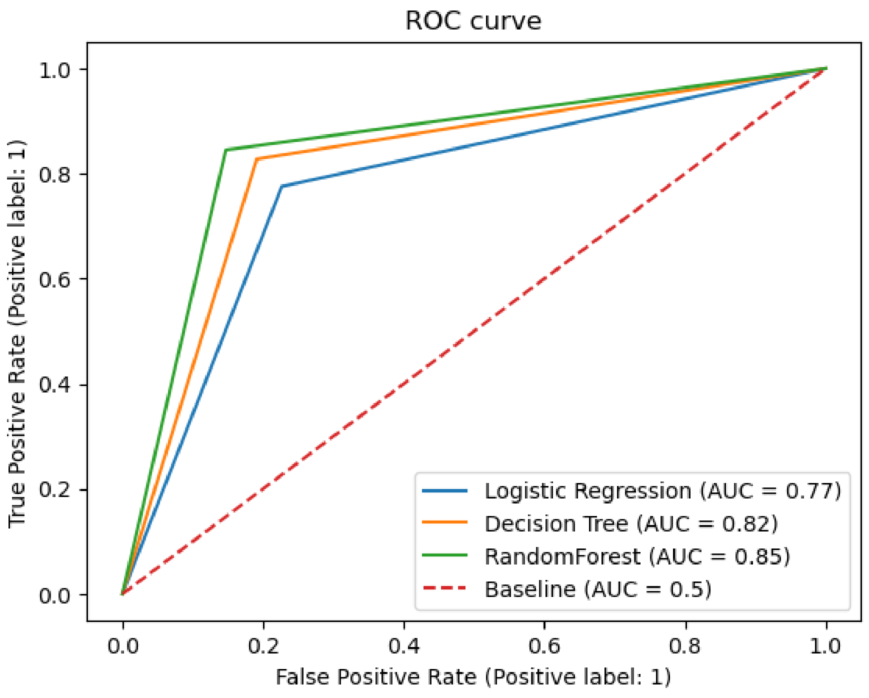


Figura 13: ROC curves (SMOTE + Hyperparameter Tuning)



Justification

The Random Forest model outperformed the other models in all scenarios when using the AUC score as the metric. It showed a slight improvement when trained and evaluated using SMOTE, and had a very close score with hyperparameter optimization. To improve this model, I suggest increasing the number or time of hyperparameter tuning, as setting it to only 10 jobs may not have demonstrated optimal improvement.

Conclusion

In conclusion, the evaluation of the Random Forest model using a confusion matrix and AUC score indicates that the model has shown decent performance in correctly classifying both positive and negative samples. The results suggest that the model can be a valuable tool for predicting churned customers.

References

[1]. N. Forhad, M. S. Hussain, and R. M. Rahman, "Churn analysis: Predicting churners," in Proceedings of the Ninth International Conference on Digital Information Management (ICDIM 2014), Phitsanulok, Thailand, 2014, pp. 237-241, doi: 10.1109/ICDIM.2014.6991433.

[2]. Qureshi, Saad, Ammar Rehman, Ali Qamar, Aatif Kamal, and Ahsan Rehman. "Telecommunication Subscribers' Churn Prediction Model Using Machine Learning." In Proceedings of the 8th International Conference on Digital Information Management (ICDIM 2013), 2013, pp. 133-137, doi: 10.1109/ICDIM.2013.6693977.

[3]. Ullah, Irfan, Basit Raza, Ahmad Malik, Muhammad Imran, Saif Islam, and Sung Won Kim. "A Churn Prediction Model Using Random Forest: Analysis of Machine Learning Techniques for Churn Prediction and Factor Identification in Telecom Sector." IEEE Access, vol. 7, pp. 104634-104647, 2019, doi: 10.1109/ACCESS.2019.2914999.

[4]. Khan, Muhammad, Johua Manoj, Anikate Singh, and Joshua Blumenstock. "Behavioral Modeling for Churn Prediction: Early Indicators and Accurate Predictors of Custom Defection and Loyalty." In Proceedings of the IEEE International Congress on Big Data (BigData Congress), 2015, pp. 7-14, doi: 10.1109/BigDataCongress.2015.107.

[5]. G. Menardi and N. Torelli, "Training and assessing classification rules with imbalanced data," Data Mining and Knowledge Discovery, vol. 28, no. 1, pp. 92-122, 2014, https://doi.org/10.1007/s10618-012-0295-5.

[6]. V. S. Spelmen and R. Porkodi, "A Review on Handling Imbalanced Data," in Proceedings of the 2018 International Conference on Current Trends towards Converging Technologies (ICCTCT), Coimbatore, India, 2018, pp. 1-11, doi: 10.1109/ICCTCT.2018.8551020.

[7]. N. V. Chawla, K. W. Bowyer, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321-357, 2002.

[8]. R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95), San Francisco, CA, USA, 1995, pp. 1137-1143.